UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA SOFTWARE DE GESTIÓN PARA LA PRODUCCIÓN DE SISTEMAS ELECTRÓNICOS CON MÁQUINAS PICK & PLACE

Titulación: **INGENIERO DE TELECOMUNICACIÓN**

Autor: **D. Sergio Jesús Felipe Delgado**

Tutor: **D. Aurelio Vega Martínez**

Fecha: **Junio 2016**

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA SOFTWARE DE GESTIÓN PARA LA PRODUCCIÓN DE SISTEMAS ELECTRÓNICOS CON MÁQUINAS PICK & PLACE

HOJA DE FIRMAS

Alumno | **Tutor**

Fdo.: D. Sergio Jesús Felipe Delgado | Fdo.: D. Aurelio Vega Martínez

Fecha: Junio 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA SOFTWARE DE GESTIÓN PARA LA PRODUCCIÓN DE SISTEMAS ELECTRÓNICOS CON MÁQUINAS PICK & PLACE

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.: .....

Vocal **Secretario**

Fdo.: ..... Fdo.: .....

Fecha: Junio 2016

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

RESUMEN

El estado actual de la tecnología hace necesaria la fabricación de prototipos electrónicos con componentes de tecnología SMD. El Servicio de Fabricación de Prototipos del IUMA posee varias máquinas pick & place manuales y automáticas para el montaje de estos componentes. Sin embargo, la fabricación de prototipos o pequeñas series con cada una de ellas requiere de métodos de trabajo diferentes.

Con este proyecto se trata de ayudar en el proceso de fabricación creando una herramienta software de ayuda tanto en el montaje manual como en el automatizado.

Para ayudar en el montaje manual, la aplicación crea una imagen de la placa a partir de los archivos Gerber de ésta e indica sobre ella el próximo componente a colocar según se haya planificado el montaje previamente. De esta forma guía al usuario para que realice la tarea de forma rápida y sin cometer errores de posición u omisiones. Alternativamente, permite exportar las imágenes y datos de los sucesivos pasos de montaje en un archivo PDF, evitando la necesidad de disponer de un ordenador en la estación de montaje manual. El software creado se puede usar también para ayudar en el montaje manual de los componentes de inserción.

Para ayudar en el montaje automático la aplicación usa dos vías diferentes para controlar la máquina, dependiendo de qué máquina se va a utilizar en el montaje. Las máquinas más modernas aceptan como entrada un archivo que especifica los parámetros necesarios para el montaje (archivo VIOS Text), por lo que se usa esta vía. Las máquinas más antiguas deben ser controladas directamente por un ordenador externo mediante un puerto serie. Este ordenador debe ir enviando la secuencia de comandos adecuada para activar cada elemento de la máquina en el momento preciso y comprobando simultáneamente las condiciones de error, de acuerdo al programa de montaje que se esté ejecutando.

El software creado se ejecuta bajo Windows y despliega un entorno de ventanas, altamente adaptable a los gustos del usuario y altamente configurable, incluso ante posibles cambios de algunas características de las máquinas (por ejemplo, por reparaciones). Por otra parte, tiene requerimientos de memoria, disco y CPU muy discretos y carece de los límites, en la cantidad de datos de trabajo, del software original de las máquinas más antiguas.

Además evita, en la medida de lo posible, que el usuario introduzca errores inadvertidamente, realizando de forma visual la mayor parte de la preparación previa del montaje. Por ejemplo, accediendo a la cámara que incorporan algunas de las máquinas, muestra la imagen de la placa en tiempo real. El usuario simplemente debe apuntar con el ratón al lugar que desea y hacer clic para introducir las coordenadas en los formularios.

La aplicación es multiventana, lo que permite simultanear la edición de datos de diferente naturaleza. Además, permite trabajar con varias placas simultáneamente, iguales o diferentes, con posiciones y ángulos de giro arbitrarios. Un visor y editor gráfico da la posibilidad de ver la distribución de los componentes sobre la placa y sus características de montaje, permitiendo trasladarlos, girarlos o cambiarlos de estado usando el ratón.

El software creado también permite al usuario importar ficheros de Altium para extraer de ellos los datos de colocación de los componentes sobre la placa. Esto se ha combinado con la creación automática de una base de datos de dimensiones de los componentes, que paulatinamente, conforme se vayan incluyendo nuevos componentes, permitirá automatizar cada vez más la tarea de dimensionarlos.

La aplicación posee un manual del usuario completo y también una guía rápida para el montaje de placas que se puede utilizar para el uso de la aplicación casi sin saber previamente cómo funciona. De esta forma un usuario novel encontrará una lista de pasos a seguir para no perderse y se reduce el tiempo de aprendizaje.

Se completa la aplicación con un instalador que se encarga de crear una entrada en el menú de aplicaciones de Windows para acceder al programa, manual y guía rápida, así como un acceso directo en el escritorio. Además, instala los iconos de la aplicación y de sus archivos de datos, asociando éstos al programa para poder abrirlos haciendo doble clic.

Por último, mencionar que la aplicación creada no sólo permite realizar la colocación de componentes, sino además realizar un dispensado de adhesivo. Esto permite crear placas con componentes en ambas caras.

CAPÍTULO 1. INTRODUCCIÓN

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

1.1. Introducción

En la fabricación de circuitos electrónicos, uno de los principales costes ha sido típicamente el asociado a la mano de obra necesaria en la etapa de montaje de las placas de circuito impreso. Debido a esto, se ha investigado intensivamente y se han desarrollado sistemas para automatizar el montaje todo lo posible [**1**].

De la tecnología THT (Through-Hole Technology –Tecnología de agujeros pasantes–) [**2**] se ha pasado a la tecnología SMT (Surface Mount Technology –Tecnología de montaje superficial–) [**3**] que, no sólo es más fácil de automatizar, sino que además permite una mayor densidad de componentes. Además, para mejorar la competitividad en el mercado global actual, los fabricantes se han ido adaptando a los nuevos estándares de alta calidad, bajo coste y reducido tiempo total de manufactura (TTM -Time To Market-).

Un elemento que ha permitido cambiar el proceso de fabricación en esta dirección hacia una máxima automatización ha sido la creación de diversa maquinaria y sus correspondientes procesos para el montaje automatizado o semiautomatizado de los componentes en la PCB.

Estas máquinas se conocen por diferentes nombres. Así, es común encontrar referencias a ellas como "*máquinas colocadoras de componentes*", "*chipeadoras*", "*máquinas pick & place*" o "*máquinas collect & place*", entre otros. Lo más habitual es designarlas como *"pick & place"* por la función que realizan, puesto que, básicamente, lo que hacen es *"coger"* los componentes a montar de unos alimentadores y *"colocarlos"* en la posición que deben ocupar sobre la superficie de la PCB (*collect & place* se usa en el caso de que la máquina sea capaz de coger varios componentes para luego colocarlos).

En la figura 1 se muestra el proceso que se sigue en el laboratorio del SFP para fabricar un circuito electrónico. Las máquinas objeto de este proyecto entran a formar parte en este proceso en el punto de montaje de componentes, permitiendo una considerable reducción de tiempos en el proceso global.

Normalmente, el proceso comienza con un diseño informático del circuito a fabricar, habitualmente realizado en Altium. A partir del diseño de la PCB del circuito, se pasa a la primera etapa de la producción, que consiste en la fabricación de la PCB.

Para realizar este proceso el laboratorio dispone de dos líneas: una línea mediante fresado y otra mediante atacado químico. Ambas líneas permiten alcanzar una gran precisión y calidad de los acabados.

El siguiente paso es fabricar la máscara que permitirá controlar la cantidad de pasta de soldadura que se aplica a cada pad. La máscara sólo es necesaria para la soldadura de los componentes SMD, ya que los THT se montan y sueldan en otra línea de producción.

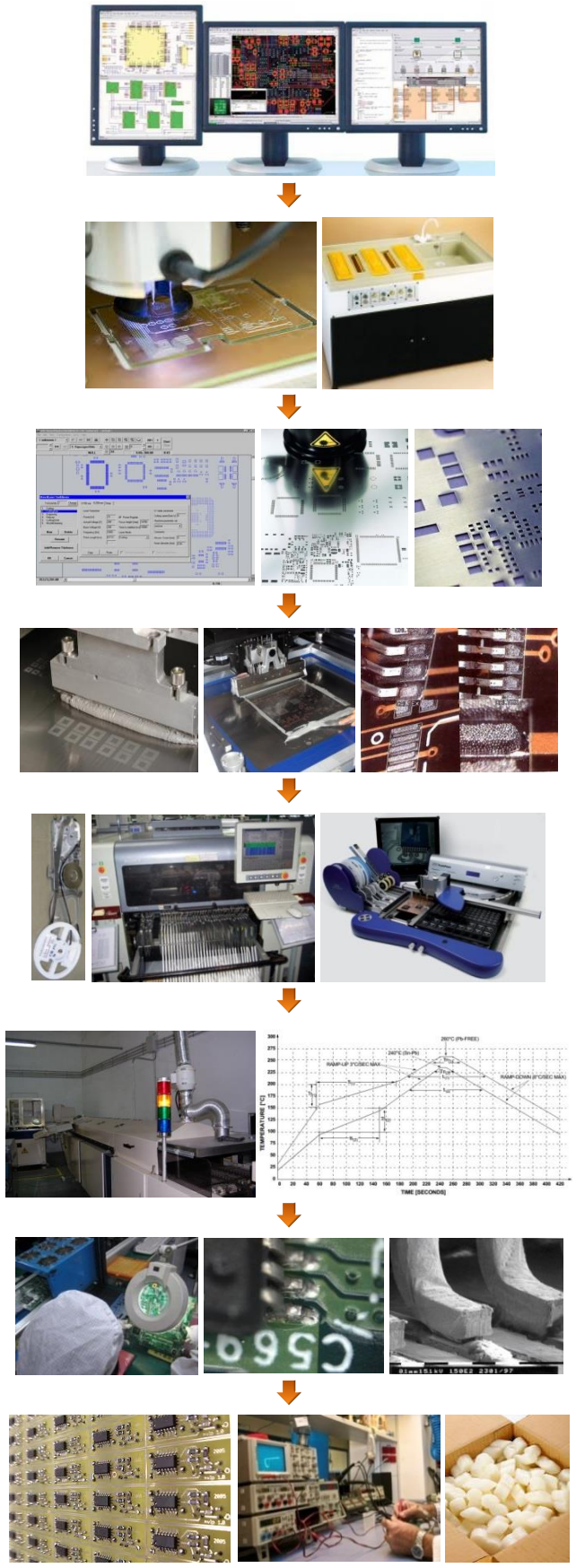


Figura 1. Diagrama del proceso de fabricación de PCB en el laboratorio.

Una vez fabricada, la máscara es fijada sobre la PCB mediante un marco y, mediante una espátula (squeegee), se aplica la cantidad justa de pasta de soldadura a cada pad. Es necesario verificar que este proceso se ha realizado a la perfección, puesto que si no, posteriormente, al salir del horno, podrían aparecer cortocircuitos o terminales sin soldar.

La siguiente etapa consiste en colocar cada componente en su posición sobre la placa. Si algunos componentes necesitan adhesivo para mantenerse en posición hasta que sea definitivamente soldado, en esta etapa se les dispensará. En el laboratorio existen máquinas automáticas y semiautomáticas para realizar la labor de montaje. Como se mencionó anteriormente, el desarrollo de este proyecto tratará de ayudar en la mejora de esta etapa.

A continuación, las placas pasan al horno de refusión, donde la temperatura varía siguiendo un perfil muy preciso, necesario para completar perfectamente la soldadura de todos los componentes SMD.

Tras ser soldadas, las placas pasan por un proceso de limpieza mediante ultrasonidos que elimina las impurezas que hayan podido quedar en la soldadura. Además, se realiza una inspección óptica de la calidad de las soldaduras y se buscan posibles errores del proceso como cortocircuitos o puntos mal soldados.

Por último, se prepara cada circuito individual, cortándolo del panel de montaje si es necesario y se hacen las comprobaciones eléctricas que el solicitante requirió. Posteriormente se realizarán el resto de tratamientos solicitados (por ejemplo montarlo en una estructura o en una caja) y se embalará para su entrega en caso de ser necesario.

1.2. Antecedentes

En el *Laboratorio de Fabricación de Prototipos y Sistemas Electrónicos* del *Servicio de Fabricación de Prototipos* del *Instituto Universitario de Microelectrónica Aplicada*, se han desarrollado anteriormente varios proyectos relacionados con la automatización de la fabricación y montaje de prototipos y pequeñas series. Entre ellos, cabe destacar un proyecto para la automatización de máquinas pick & place manuales y otro proyecto de diseño de una minifresadora de bajo coste, pensada para la fabricación de circuitos impresos de alta calidad para laboratorios, centros de enseñanza o empresas. Además, se ha desarrollado un prototipo de máquina pick & place manual de bajo coste.

El laboratorio dispone de varias máquinas pick & place comerciales tanto para el montaje manual como automático de componentes SMD. Entre ellas, posee dos máquinas pick & place

Cosy SMD Flexpick cedidas fruto del acuerdo de colaboración entre el IUMA y la empresa Inerza S.A. Además de las máquinas, el laboratorio obtuvo multitud de accesorios de las mismas que son necesarios en algunas de las etapas de fabricación vistas anteriormente. Posteriormente se incorporaron otras dos máquinas automáticas más modernas, cuya operación no depende de un ordenador externo, como es el caso de las Cosy, sino que son completamente autónomas.

Para utilizar las máquinas más modernas, actualmente hay que generar un archivo VIOS Text con la información del montaje utilizando el programa Cad2cad, proporcionado por Philips. Para generar el archivo VIOS hay que alimentar al programa con las coordenadas de los componentes sobre la placa. Normalmente esto se hace a partir de archivos de salida de Altium, pero previamente éstos deben ser procesados manualmente por el operador (utilizando una hoja de cálculo) para que el formato de los datos sea el adecuado.

Por otra parte, las máquinas Cosy se han venido controlando hasta ahora con el software que las acompañaba originalmente, proporcionado por el fabricante. Este software es bastante antiguo, pues fue desarrollado para el sistema operativo MS-DOS. Su interfaz de usuario se basa en menús numéricos que permiten al operador cambiar de pantalla para acceder a los distintos parámetros a controlar.

Estos menús numéricos están agrupados según ciertos criterios, haciendo que en ocasiones la tarea se complique por no poder visualizar simultáneamente datos de diferentes pantallas ni tampoco poder cambiar ágilmente entre datos de distinta naturaleza. Por ejemplo, si al trabajar en la pantalla de datos de posición de un componente, se quieren ver datos relativos al alimentador desde donde se cogerá, hay que salir a la pantalla del menú principal, luego entrar en la pantalla de alimentadores y por último avanzar secuencialmente por las pantallas de los diferentes alimentadores hasta llegar a la del alimentador concreto que se quería consultar.

Las máquinas Cosy y su software de control fueron entregados sin manuales de operación, por lo que no hay muchas personas capaces de utilizarlas. Además, el software de control original con menús numéricos y pantallas de opciones puede ser bastante enigmático para el usuario novel. Como resultado, estas costosas máquinas se encuentran infrautilizadas.

Esto nos ha impulsado a analizar en profundidad el funcionamiento de dichas máquinas y así averiguar los métodos de operación necesarios para crear un nuevo software de control que se adapte a estándares modernos y reduzca las dificultades de aprendizaje. De esta forma se podrá dar un uso más continuado a estas máquinas, dándoles una segunda vida.

Por otro lado, cuando se va a realizar el montaje manual de un prototipo, normalmente se deben realizar una serie de preparativos para facilitar el montaje y evitar cometer errores. Estos preparativos incluyen generar diversa información del posicionamiento de los componentes y esquemas gráficos de dónde y cómo van ubicados. Esta tarea previa se suele realizar manualmente, haciendo anotaciones sobre los diseños de la placa en papel y generando listados de los componentes a montar. Posteriormente, toda esta información en papel se traslada a la estación de montaje manual como ayuda al operador.

Todos estos preparativos representan una carga de trabajo manual importante, pero es más necesaria cuanto más compleja es la placa. Por eso sería útil tener una herramienta de ayuda que permita montar la placa sin errores evitando esta sobrecarga de trabajo de preparación.

1.3. Objetivos

Con este proyecto se trata de dar un paso más en la línea de desarrollos anteriores, creando un software que cubra las necesidades expuestas y pueda manejar estas máquinas pick & place para realizar la producción de prototipos y series pequeñas o medianas. Se desea poder crear trabajos de montaje no sólo introduciendo manualmente los parámetros de colocación de cada componente (como se hacía hasta ahora con el software original de la Cosy), sino también capturarlos desde un diseño realizado en Altium, por medio de sus ficheros de salida (como se hace con las máquinas más modernas).

Este software se creará para funcionar en un entorno de ventanas como es Microsoft Windows, posibilitando operar con el ratón y el teclado tal y como es habitual hoy en día. Además, el software permitirá acceder a datos de diferente índole simultáneamente, utilizando para ello diferentes ventanas, solucionando así una de las limitaciones del software original de las Cosy.

También se desea facilitar la operación manual de la máquina Cosy, que se utiliza para el ajuste visual de la posición de los componentes. Para ello se permitirá mover el brazo de la máquina manualmente, a través de la aplicación, visualizando las imágenes tomadas por la cámara directamente en la aplicación y completando automáticamente los parámetros de posición en el formulario.

De esta manera, cuando el operador de la máquina realiza ajustes o comprobaciones mientras trabaja en una placa, no tendrá que cambiar constantemente entre el teclado y monitor del ordenador por una parte y el teclado y monitor de vídeo de la máquina por otra, como es

necesario hacer con el software original. Todo esto acelerará tanto la curva de aprendizaje como el trabajo cotidiano de preparación de las placas para su montaje automatizado.

En una segunda etapa se extenderá el software para ayudar a un operador humano en la tarea de montaje manual. Esto permitirá al operador acelerar este trabajo y realizarlo con una menor probabilidad de cometer un error. Aprovechando el sistema desarrollado, se incluirán opciones adicionales que guiarán al operador indicándole la ubicación de cada componente que debe montar.

Para mejorar las indicaciones, se desarrollará un intérprete simple de archivos Gerber que generará una imagen de la placa. Esta imagen se mostrará en pantalla en un proceso interactivo y sobre ella se indicará la ubicación de cada componente que se vaya a tratar. El proceso irá recorriendo la lista de componentes a montar, esperando a que el usuario le indique que ha completado el montaje del componente actual antes de pasar al siguiente. Además, el operador podrá saltar a cualquier componente de la lista para alterar el orden de montaje en caso de que sea necesario.

Como se comentó antes, actualmente el montaje manual se realiza preparando previamente la información necesaria en papel y trasladándola posteriormente al puesto de trabajo para realizar el montaje. La aplicación a desarrollar ayudará también en esta tarea y la extenderá, generando la información de ayuda al montaje en formato PDF. Esto permitirá gestionar y transmitir esta información por medios electrónicos y por otra parte visualizarla en el puesto de trabajo tanto en papel como por medios electrónicos (tablet, móvil, otro ordenador, etc.).

Además, el software creado en este proyecto ayudará también en el uso de las máquinas pick & place automáticas más modernas de que dispone el laboratorio, generando directamente archivos VIOS Text que puedan ser cargados por éstas. De esta forma se evitará el procesado manual que es necesario realizar actualmente.

1.4. Trabajos relacionados

En el laboratorio se ha desarrollado otro proyecto que trata de analizar en profundidad las máquinas Cosy, generando una serie de documentos técnicos acerca de éstas. Uno de los objetivos de ese proyecto es poder producir módulos, utilizando tecnología actual, que puedan sustituir a los módulos que forman estas máquinas. Al ser totalmente compatibles, podrán interactuar con el resto de módulos que la forman, aumentando de esta forma su vida útil o incluso su funcionalidad.

1.5. Peticionario

Actúa como peticionario académico de este proyecto la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE). Actúa como peticionario técnico el Servicio de Fabricación de Prototipos del Instituto Universitario de Microelectrónica Aplicada de la Universidad de Las Palmas de Gran Canaria.

Con el presente proyecto se pretende cumplir con el requisito académico de presentación del Proyecto de Final de Carrera en la Escuela de Ingeniería de Telecomunicación y Electrónica, para la obtención del título de Ingeniero de Telecomunicación, que proporciona al alumno la capacidad para desempeñar su futuro profesional.

1.6. Estructura de la memoria

La memoria ha sido estructurada en 8 capítulos que se describen brevemente a continuación:

- **Capítulo 1: Introducción.** En este primer capítulo se introduce la tarea a realizar, trabajos relacionados realizados anteriormente y actuales, los objetivos que se pretenden cumplir y la motivación del proyecto.
- **Capítulo 2: Estudio previo.** En este capítulo se desarrollan conceptos básicos sobre la tecnología SMT que nos permitan familiarizarnos con el proceso de fabricación que se lleva a cabo en el laboratorio y, sobre todo con las operaciones que realizan las máquinas colocadoras de componentes (habitualmente conocidas como pick & place). De esta forma se podrá conocer el estado del arte actual en la fabricación SMT para establecer un flujo de trabajo óptimo en la fabricación, que nos ayudará a desarrollar la aplicación para que facilite la tarea en todo lo posible al operador de las máquinas. Además, se comenta la maquinaria y herramientas disponibles en el laboratorio para esta línea de producción.
- **Capítulo 3: Herramientas utilizadas.** En este capítulo se hará un breve acercamiento a las distintas herramientas de ingeniería utilizadas en el desarrollo del proyecto. Algunas herramientas son de tipo hardware, como el analizador lógico o el calibrador utilizado; otras son de tipo software, como el entorno de programación Visual Studio o el software CAD/EDA Altium y otras son intelectuales o procedimentales, como el propio lenguaje de programación utilizado o el flujo de diseño.
- **Capítulo 4: Estudio de las máquinas.** Este capítulo se ha dedicado a conocer con bastante grado de detalle las máquinas pick & place para las que se va a

desarrollar la aplicación. Para ello se han desmontado las máquinas tipo Cosy parcialmente y se ha analizado su funcionamiento, tanto desde el punto de vista electrónico como mecánico y neumático. Además, se describe brevemente el ordenador de control y cómo opera el software original de la máquina, cuya funcionalidad se pretende emular y extender con nuestra aplicación. Para poder controlar las funciones de la Cosy, se llevó a cabo un análisis minucioso del protocolo de comunicaciones de ésta, cuyos resultados se han integrado en la aplicación desarrollada. También se describen otras máquinas pick & place disponibles en el laboratorio que se han estudiado para desarrollar la aplicación.

- **Capítulo 5: Desarrollo de la solución.** Este capítulo de la memoria contiene el núcleo del desarrollo llevado a cabo durante la mayor parte de este proyecto, ya que se explican los puntos importantes del código desarrollado para cumplir los objetivos del proyecto. Es el capítulo de mayor extensión de la memoria y en él se explica con detalle las decisiones tomadas durante la implementación.
- **Capítulo 6: Resultados.** En el sexto capítulo se presentan los resultados obtenidos tras la implementación del proyecto, tanto de las simulaciones realizadas, para lo que se desarrollaron un modo especial de operación del programa y algunas pruebas especiales de rendimiento, como los resultados trabajando con el programa conectado directamente al hardware de la máquina a controlar. Además se presentan otros resultados de índole académica, como ha sido la realización de dos artículos.
- **Capítulo 7: Presupuesto.** En este capítulo se hace balance de los costes derivados de la realización de este proyecto.
- **Capítulo 8: Conclusiones y trabajos futuros.** En este capítulo se desarrollan las conclusiones extraídas del desarrollo de este proyecto, así como posibles líneas de desarrollo futuro que quedan abiertas.

Esta memoria se acompaña de un disco donde se ha incluido una copia de la documentación de este Proyecto Fin de Carrera.

CAPÍTULO 2. ESTUDIO PREVIO

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

2.1. Introducción

En este capítulo se hace un repaso de la tecnología de montaje de componentes y los cambios desde la tecnología THT a la SMD. Se repasan también los procedimientos, necesarios u opcionales, para el montaje SMD, como es el dispensado de adhesivo para pegar los componentes a la placa.

Además se introducen las máquinas pick & place automáticas, que son las que se encargan del montaje de los componentes sobre la placa.

2.2. Tecnología de inserción

Desde hace muchos años la *Tecnología de Montaje Superficial* de componentes (SMT por sus siglas en inglés) ha ido desplazando en gran parte a su antecesora, la *Tecnología de Agujeros Pasantes* o THT, también conocida como *de inserción* [**4**].

La tecnología THT hizo su aparición con las *Placas de Circuito Impreso*, PCI o PCB del inglés *Printed Circuit Boards*, en reemplazo de la tecnología de montaje de componentes sobre chasis metálicos, regletas aislantes con terminales y cableados estructurados como seguramente habremos visto en algún aparato electrónico antiguo.

Los materiales base de las PCB son de buenas propiedades aislantes y adecuada estabilidad térmica, química y mecánica. Sobre esta base se halla laminado el circuito eléctrico en cobre. Los caminos conductores poseen islas con agujeros pasantes a través de los cuales asomarán los terminales de los componentes montados y donde se llevará a cabo la soldadura para la fijación mecánica y unión eléctrica de los componentes al circuito.

Existen PCB de simple cara, doble cara y multicapa. En las PCB de doble capa y multicapa los taladros de inserción de terminales están metalizados e interconectan las diferentes caras y/o capas del circuito.

Para el montaje manual de PCB THT basta una pinza y un alicate para insertar los componentes en su correcta posición e ir cortando los terminales y doblando los extremos salientes para lograr cierto anclaje mecánico que permita soldarlo manualmente sin que caigan de la placa al invertirla.

Para volúmenes de producción pequeños o medianos es conveniente disponer de herramientas de corte y preformado de los terminales de los componentes. Así se les puede dar formas especiales para lograr la separación requerida de la placa y el anclaje mecánico que los mantengan en posición hasta ser soldados sin que se levanten ni caigan por el orificio. A los componentes que, por su baja disipación térmica, no necesiten ser montados con cierta separación de la placa se les puede doblar los terminales en ángulo recto. En la figura 2 se puede ver un ejemplo esquemático de cortes de terminales preformados.

Para grandes volúmenes de producción se realiza el montaje automático con máquinas insertadoras de componentes, dividiéndose según el tipo de encapsulado del componente en insertadoras radiales, axiales o de circuitos integrados DIP.

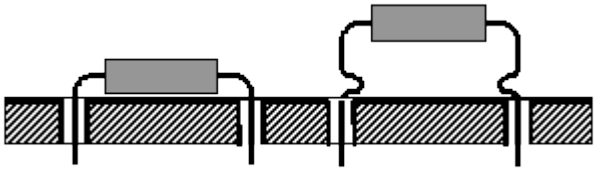


Figura 2. Componentes THT con terminales preformados.

La insertadora DIP toma los componentes de las varillas de circuitos integrados, mientras que la radial y axial lo hacen directamente de los rollos provistos por el fabricante o bien, mediante el paso previo en una máquina secuenciadora, se crean nuevos rollos con la secuencia de componentes que luego serán insertados.

En todos los casos, un cabezal se encarga de tomar el componente, doblar sus terminales y posicionarlo en el ángulo de inserción programado, mientras debajo una mesa X-Y que sostiene a la PCB se ubica en las coordenadas programadas. En ese momento el cabezal baja introduciendo los terminales en los agujeros mientras un segundo cabezal, debajo de la PCB, termina el ciclo mediante un cortado y doblado de los terminales salientes (cut & clinch) para lograr el anclaje mecánico. De esta manera, la PCB puede ser manipulada e invertida sin que se caigan los componentes.

De todos modos algunos componentes especiales como transformadores o conectores pueden requerir ser colocados a mano o mediante insertadoras especiales (Odd-forms).

Una vez la PCB tiene todos los componentes colocados se somete a un proceso de soldadura por ola que los fija mecánicamente de forma definitiva y además asegura la conexión eléctrica. No es normal colocar componentes THT en ambas caras de la PCB debido a que el proceso de soldadura se complicaría en exceso.

2.3. Tecnología de montaje superficial

La *Tecnología de Montaje Superficial* (SMT) emplea componentes SMD que se diferencian de los THT en que no cuentan con terminales de conexión alámbricos, sino que el propio encapsulado posee sus extremos metalizados o con terminales cortos y rígidos, moldeados de forma que le permiten apoyarse sobre la PCB. En la figura 3 se pueden ver algunos ejemplos.

Los componentes SMD pueden ir montados en cualquiera de las dos caras de la PCB y pueden compartir la placa con componentes THT, teniéndose en ese caso una técnica de montaje mixta.

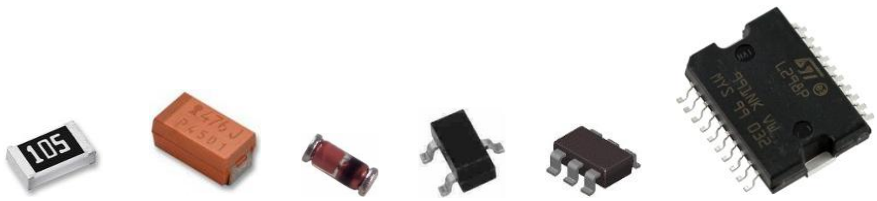


Figura 3. Componentes SMD.

Para la soldadura de componentes SMD, el método preferido consiste en realizar una impresión serigráfica de pasta de soldar sobre los pads de la PCB. Esta pasta de soldar está hecha a base de una aleación de estaño microgranulado y flux. A continuación los componentes son tomados de su embalaje y colocados en las coordenadas programadas mediante máquinas pick & place. Por cuestiones de calidad, precisión, velocidad y practicidad, la colocación manual de SMD sólo se realiza para la fabricación de algunos prototipos.

Una vez puestos los componentes, la placa es introducida en un horno continuo para desarrollar un ciclo térmico que incluye precalentamiento, fusión del estaño, reflujo del mismo y enfriamiento. Este proceso es conocido como soldadura por reflujo (reflow) y los hornos empleados hoy en día pueden ser infrarrojos o de convección forzada (más modernos). Existe también la opción de soldadura en atmósfera inerte, en la que se desplaza el oxígeno atmosférico mediante la inyección de nitrógeno para evitar oxidaciones durante la soldadura.

Figura 4. Proceso de estañado, colocación y soldado SMD.

Cuando se va a realizar el montaje de componentes SMD para luego realizar la soldadura por ola, éstos se fijan mediante un proceso previo de aplicación de adhesivo, el cual se lleva a cabo con una máquina dispensadora de gotas o mediante serigrafía de pegamento.

La colocación de los componentes sobre las gotas se realiza con el mismo tipo de máquinas pick & place mencionadas anteriormente. Tras colocar los componentes sobre las gotas de pegamento, la placa se introduce en un horno donde se desarrollará el proceso de "curado" o endurecimiento del pegamento siguiendo una curva de temperatura adecuada. Una vez curado el pegamento la placa puede ser soldada por baño de ola, como se puede ver en la figura 5.

Como se mencionó anteriormente, en algunas ocasiones se emplea una técnica mixta, que no es más que el resultado de combinar componentes THT con componentes SMT en la misma placa. Esto es muy común ya que, si bien en formato SMD existe casi todo tipo de

componentes, en algunas ocasiones por razones de costo o por otras necesidades se siguen colocando componentes THT.

Figura 5. Proceso de dispensado de pegamento, colocación y soldado SMD.

En este caso, cuando se montan componentes SMD sobre el lado de componentes, como en este lado no se puede usar soldadura por ola se hace primero un tratamiento completo de esta cara usando soldadura por refusión, completando luego el proceso de la cara inferior como se describió anteriormente.

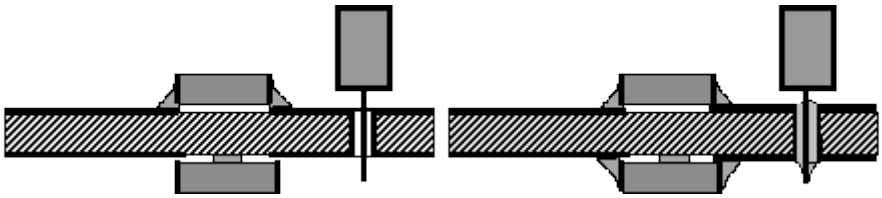


Figura 6. Proceso en placas mixtas (SMD + THT).

2.4. Dispensado de pegamento

El proceso de dispensado de pegamento es parte vital de la técnica de montaje de componentes electrónicos SMD sobre el lado de soldadura por ola de una placa [**4**]. Este dispensado se lleva a cabo antes de la colocación de los componentes SMD y la función de las gotas de pegamento es mantener adheridos los SMD a la placa hasta que hayan sido soldados mediante un baño de ola. Para ello, después de colocar los SMD y antes de la soldadura debe tener lugar un proceso de curado del adhesivo para endurecerlo.

2.4.1. Características de un adhesivo SMD

La mayoría de los adhesivos usados para montaje superficial son del tipo epoxi por lo que deben ser almacenados en refrigerador (aproximadamente a 5 ^{o}C) para prolongar su vida útil.

Algunas de las propiedades deseadas son:

- Buena fluidez.
- Perfil y tamaño de gota consistente.
- Alta solidez (resistencia a fuerzas) tanto fresco como ya curado.
- Curado rápido.
- Flexibilidad y resistencia a shocks térmicos.

- Permitir el dispensado a alta velocidad y tamaños muy pequeños de gota.
- Excelentes propiedades eléctricas sobre la placa una vez curado.
- No debe hacer hilos.
- La gota no debe desplomarse durante el ciclo térmico de curado.
- Colores que resalten sobre el substrato (muy comunes el rojo, naranja y amarillo).

La resistencia mecánica de la unión es crítica para el buen funcionamiento de un adhesivo SMD y está determinada por:

- Grado de adhesión al componente y a la PCB.
- Tamaño y forma de la gota.
- Nivel de curado.

Por otro lado las principales causas de fallo son:

- Curado inadecuado.
- Volumen de gota inadecuado.
- Nivel de adherencia pobre o malo.

2.4.2. Características de una buena gota

El cómo se formará la gota y el tamaño y perfil resultante es determinado por las características reológicas del mismo, que son las que describen sus propiedades de viscosidad y tensión superficial.

Los adhesivos para SMD están diseñados para ser tixotrópicos y esta condición también determinará su forma. La característica tixotrópica es la que le confiere la capacidad de disminuir la viscosidad al hallarse bajo presión permitiendo un buen flujo, pero cuando el adhesivo alcanza la PCB, rápidamente se reestructura y recobra su viscosidad original. Un ejemplo doméstico de un producto tixotrópico es la pasta dental, la cual fluye al presionar el envase pero que, sin embargo, queda firme una vez sobre el cepillo.

La gota debe ser de forma cónica, con pico o redonda hemisférica. No obstante el perfil también es determinado por el volumen dosificado, el diámetro de la boquilla y la distancia de la misma respecto de la placa, por lo que para un mismo volumen dosificado y jugando con estos parámetros es posible lograr gotas flacas y altas hasta bajas y gordas, condicionado esto a que el tamaño final de la gota (una vez colocado el componente) no puede tener un diámetro mayor que la distancia entre pads y que la altura debe ser suficiente como para cubrir el espacio entre la PCB

y el componente, lo cual según el componente oscila entre los 0,05mm y los 0,3mm. La relación ancho/altura de la gota es típicamente de 1,5:1 a 5:1.

2.4.3. Sistemas de dosificación

Básicamente las máquinas dispensadoras para SMD se componen de:

- Un sistema de transporte y posicionamiento de las placas de circuito.
- Un par de ejes de posicionamiento X e Y que posicionan al cabezal sobre la placa.
- Un eje Z (sobre el cual se halla montado el cabezal dosificador).
- Uno o más cabezales dosificadores.

2.4.3.1. Métodos de contacto

Los ejes X e Y se van posicionando en las coordenadas programadas de antemano en el sistema, y una vez en posición descenderá el eje Z llevando consigo al cabezal dosificador. Un sensor solidario a Z hará contacto con la placa y esta señal activará la dosificación de una gota de adhesivo en estas coordenadas.

El eje Z vuelve a subir y así sucesivamente mediante nuevas coordenadas de X e Y se irá completando el número de gotas programadas para dicha placa de circuito. La capacidad de una máquina dispensadora de este tipo se mide en gotas por hora o dph (dots per hour). Dependiendo del número de cabezales por máquina y de la tecnología de los mismos se pueden hallar en el mercado dispensadoras que van desde las 3000 dph hasta 140.000 dph, sujeto esto también a la aplicación específica a desarrollar.

Dentro de estos métodos, llamados de contacto debido al sensor de Z, existen 3 sistemas diferentes de dosificación que se detallan a continuación con sus ventajas y desventajas:

1. Método de "presión-tiempo": Consiste en una jeringa o cartucho hermético conteniendo el adhesivo y conectado a una fuente de aire comprimido a través de una electroválvula. El volumen dosificado dependerá de la presión de aire y del tiempo que éste se aplique a la jeringa. También dependerá del diámetro de la boquilla por la que saldrá el adhesivo y de la distancia respecto de la placa.

   Tiene la ventaja de una fácil operación, preparación y limpieza. La desventaja es una mala repetibilidad. Se ve afectado por cambios de nivel en la jeringa y por cambios de la viscosidad del adhesivo.

2. Método de "bomba a tornillo": Este sistema se basa en el tornillo de Arquímedes y es uno de los denominados de desplazamiento positivo. En este caso la jeringa que

contiene el adhesivo es presurizada en forma permanente y el pegamento inyectado en una cámara que contiene el tornillo de la bomba.

El eje de dicho tornillo se halla solidario a un embrague electromagnético que al actuar lo une a un motor que se halla girando a velocidad constante. Mediante un encoder se determina el número de giros dados por el tornillo lo cual se traduce en volumen de adhesivo desplazado. El tornillo se detiene al desactivar el embrague electromagnético.

La ventaja de este sistema es una buena repetibilidad (±5 %) y buen funcionamiento con diferentes adhesivos. La desventaja es un mayor costo del sistema y mayor complejidad a la hora de la limpieza.

3. Método "bomba lineal": También de desplazamiento positivo esta bomba posee una cámara que es alimentada por una jeringa presurizada como en el caso anterior. Al momento de dosificar una válvula cierra la entrada de pegamento y abre la salida a la boquilla dispensadora. Dentro de esta cámara se halla un pistón que al avanzar un determinado largo desplaza una cantidad exacta de adhesivo que es dispensado sobre la placa.

   La ventaja de este sistema es una mayor repetitividad y la insensibilidad a los cambios de viscosidad. Como desventaja, es más caro pero no más rápido que otros sistemas.

Los métodos de contacto antes descritos se ven limitados en velocidad, ya que para compensar variaciones de altura, una aguja de contacto debe tocar la placa en cada coordenada con el consiguiente movimiento del eje Z y sus tiempos de posicionamiento.

Esto se complica cuando la placa a dispensar adhesivo requiere una variedad de tamaños de gota que exceden el rango determinado por el diámetro de boquilla y el distanciador. En estos casos puede verse que aparecen máquinas con más de un cabezal para poder cubrir todos los rangos de tamaño de gota.

2.4.3.2. Método sin contacto

Este sistema no posee sensor de altura ni eje Z, eliminándose así los correspondientes tiempos de detección y posicionamiento, lo que permite a los ejes X e Y trabajar a mayor velocidad. Esta tecnología de dispensado por chorro conocida como jetting se basa en la inyección del adhesivo dentro de una cámara donde es atemperado para lograr la viscosidad óptima. Luego

un diseño de bola y asiento permite al adhesivo ocupar el espacio dejado por la bola al retirarse del asiento. Cuando la bola regresa la fuerza debida a la aceleración vence el flujo del adhesivo, el cual es proyectado a través de la boquilla hasta chocar contra la placa y formar así una gota de adhesivo.

Esto se lleva a cabo desde una altura de 1 a 3,5 mm de la placa. Para gotas de mayor tamaño el sistema puede dispensar hasta 5 gotas sobre la misma coordenada para lograr el volumen deseado.

Las ventajas de este método son una alta repetitividad (±3 %) y una alta velocidad de dispensado (más rápido que otros sistemas). Como desventaja tiene el ser más caro.

2.4.3.3. Método serigráfico

Con este método se debe preparar un stencil de forma similar a como se hace para dispensar la pasta de soldar. Si bien la velocidad de dispensado es mucho mayor, al lograr dispensar todas las gotas con una sola pasada de la espátula, se pierde la flexibilidad que tienen los sistemas antes descritos de modificar individualmente el tamaño de una gota en particular o de corregir sus coordenadas.

2.4.4. Etapa de curado

Una vez dispensado el adhesivo en la PCB tiene lugar la colocación de componentes e inmediatamente el adhesivo debe ser curado. Mediante el curado, el adhesivo solidifica para poder así sostener a los componentes SMD recién colocados.

El curado se lleva a cabo típicamente en línea mediante hornos de infrarrojos o bien de reflujo por convección forzada. Para bajas producciones el curado puede llevarse a cabo en hornos estáticos.

La mínima temperatura para iniciar el curado es de 100 ^{o}C, oscilando en la práctica entre los 110 ^{o}C y los 160 ^{o}C. Por encima de esto se acelera el proceso pero se obtendrían uniones quebradizas.

2.5. Máquinas de colocación de componentes SMD

2.5.1. Introducción

La primera máquina de colocación SMD se introdujo a principio de los años 80. Estas primeras máquinas, con un único cabezal de colocación, eran lentas (1000 – 2000 cph), poco fiables e imprecisas. Un tiempo después, aparecieron en el mercado máquinas con un sistema de

torreta giratoria y un sistema adicional de reconocimiento por visión. Este tipo de máquina fue diseñado para sistemas de colocación de pequeños chips a alta velocidad (14.000 – 25.000 cph). A mediados de los años 90 las máquinas sufrieron un gran cambio. Se pasó de una máquina con un único cabezal y mecanismo de bandeja a máquinas con múltiples cabezales que podían manejar pequeños y grandes componentes con una mayor facilidad y productividad.

Por otro lado, destacar la gran evolución sufrida por el método de centrado de componentes durante los años 90. Anteriormente, este método se realizaba mediante pinzas mecánicas. El centrado de los componentes basados en sistemas de visión sustituyó los citados mecanismos. Este método se desarrolló inicialmente para la colocación de componentes de paso fino, pero ha mejorado la precisión y flexibilidad de los equipos de colocación usados para componentes de montaje superficial estándar.

Tres elementos definen un buen sistema de colocación [**5**]: el *software*, los mecanismos de ubicación y los alimentadores de ubicación.

2.5.2. Clasificación de las máquinas de colocación

Los sistemas de colocación se puede clasificar de varios modos diferentes: por diseño, por la tasa de colocación (baja velocidad o alta velocidad), por el coste (bajo o alto) o por su funcionalidad (flexible o dedicada). Algunas de estas clasificaciones se solapan entre sí.

Basándonos en las especificaciones y los métodos operacionales, se han clasificados las máquinas de colocación de SMD en cinco categorías [**6**]:

- Doble entrega.
- Multiestación.
- Tipo torreta.
- Multicabezal.
- Secuencial.

2.5.2.1. Máquina de doble entrega (Dual-Delivery)

Estas máquinas constan de una mesa PCB que puede moverse en las direcciones X e Y y que debe estar alineada con respecto al cabezal para llevar a cabo las distintas operaciones; los brazos mecánicos de colocación y el transporte de los componentes sólo pueden moverse en la dirección X. Los cabezales de *pick & place,* situados en los dos extremos del brazo mecánico de longitud fija, sólo pueden moverse entre dos posiciones fijas en la dirección Y. La única e importante característica de este tipo de máquina es que cada operación de *pick & place* se

alterna entre dos lados, es decir, mientras un cabezal realiza la operación de recogida, el otro coloca componentes sobre la placa [**7**].

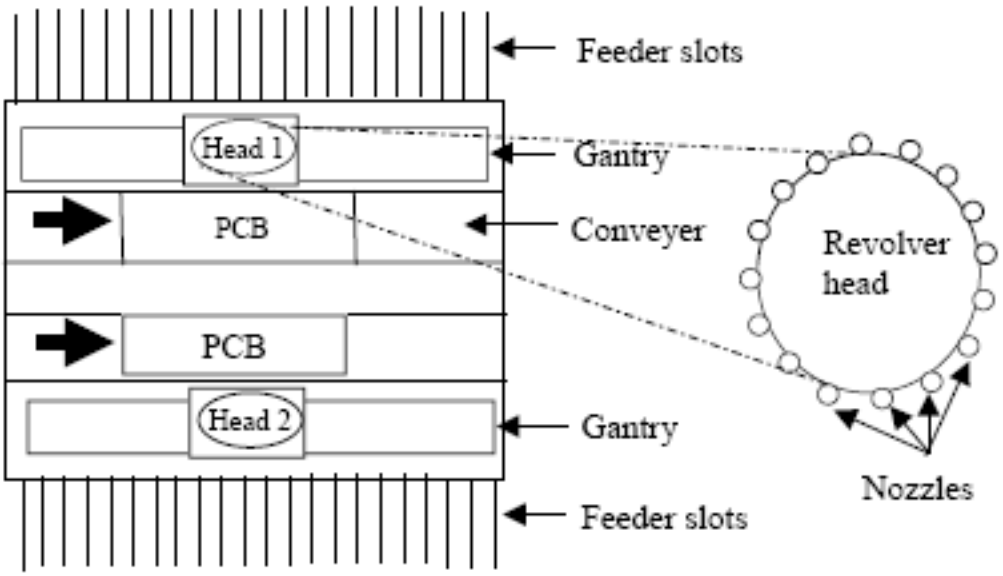


Figura 7. Máquina *pick & place* de doble entrega.

En estas máquinas todos los movimientos de la mesa PCB y de los dosificadores son congelados durante la ejecución de cada operación de p*ick & place*. Por tanto, el tiempo máximo utilizado en el movimiento del brazo mecánico, la mesa PCB y los alimentadores estarán determinados por el tiempo de ciclo.

2.5.2.2. Máquinas multiestación (Multi-Station)

Estas máquinas presentan más de un módulo de colocación, denominado estación, que son mecánicamente idénticos y capaces de ensamblar componentes de forma concurrente.

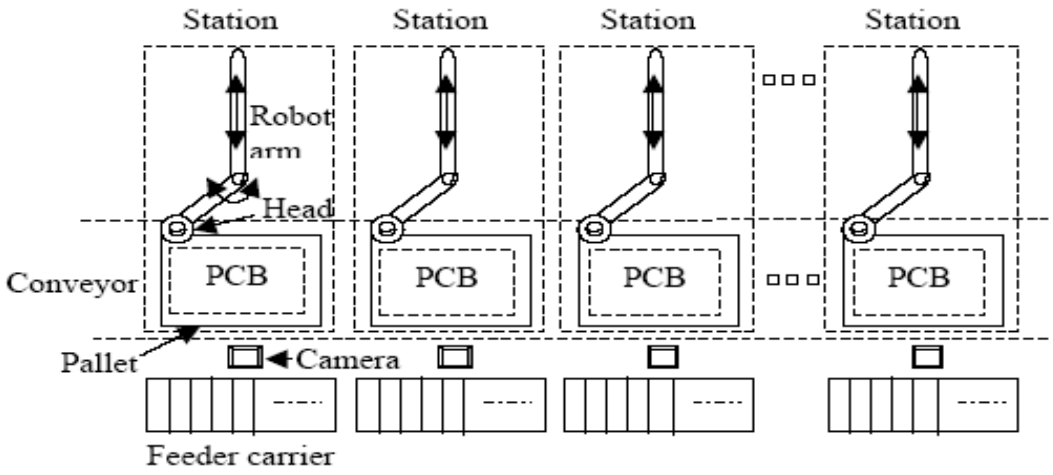


Figura 8. Máquina *pick & place* multiestación.

Las estaciones están conectadas mediante una cinta transportadora que desplaza las PCB fijadas sobre paletas entre las estaciones. Cada estación recibe las coordenadas donde se colocarán los componentes por ciclo de máquina, completándose la secuencia de colocación automáticamente y de forma simultánea a otras estaciones [**8**]. Después de que todas las estaciones hayan finalizados sus trabajos, la cinta se mueve, y el proceso de colocación continúa.

2.5.2.3. Máquinas tipo torreta (Chip Shooter)

Este tipo de máquina es muy popular en sistemas que precisan altas velocidades. Por ello el mercado al que va dirigido son líneas de producción de gran volumen, aunque también pueden utilizarse en aplicaciones de más bajo volumen [**9**].

Este sistema hace uso de un mecanismo de colocado montado sobre una torreta giratoria, con múltiples cabezales, que rota desde una posición fija de recogida hasta las posiciones de colocado, como se puede observar en la figura 9. Generalmente, cada operación de *pick & place* comienza con la recogida de un componente en la posición fija de *pickup*, mientras que un componente es colocado, de forma simultánea, en el lugar predeterminado sobre PCB. Posteriormente, la bandeja donde se posicionan los componentes para su recogida se mueve lateralmente, para así alinear los distintos alimentadores delante de la torreta. Al mismo tiempo, la mesa PCB se mueve a la posición de la siguiente ubicación de colocado.

El número máximo de alimentadores que puede soportar un sistema varía aproximadamente entre 100 y 300 alimentadores de embandado de 8 mm. En caso de utilizarse alimentadores más amplios el número disminuye. Se utilizan exclusivamente alimentadores de embandado porque son el único tipo de alimentadores que tienen la velocidad y fiabilidad suficiente para estos sistemas.

Las tasas de colocación máximas de los sistemas de torreta giratoria varían entre 14.000 a 25.000 cph. Estos sistemas son bastante caros.

Este tipo de máquinas presenta algunas desventajas:

1. El movimiento de la mesa PCB viene impuesto por la aceleración forzada sobre el componente pre-montado.
2. La exactitud de la máquina se ve limitada por el movimiento de la mesa PCB y la vibración de los alimentadores.
3. El uso de los alimentadores de bandeja (tray feeder) no es posible.

4. Se requiere de un alimentador inteligente y monitorizado para realizar las correcciones de los pequeños componentes recogidos.

Por último, debido a la restricción mecánica de la estructura del cabezal tipo torreta, esta máquina no es capaz de realizar operaciones simultáneas de recogida y colocación de componentes

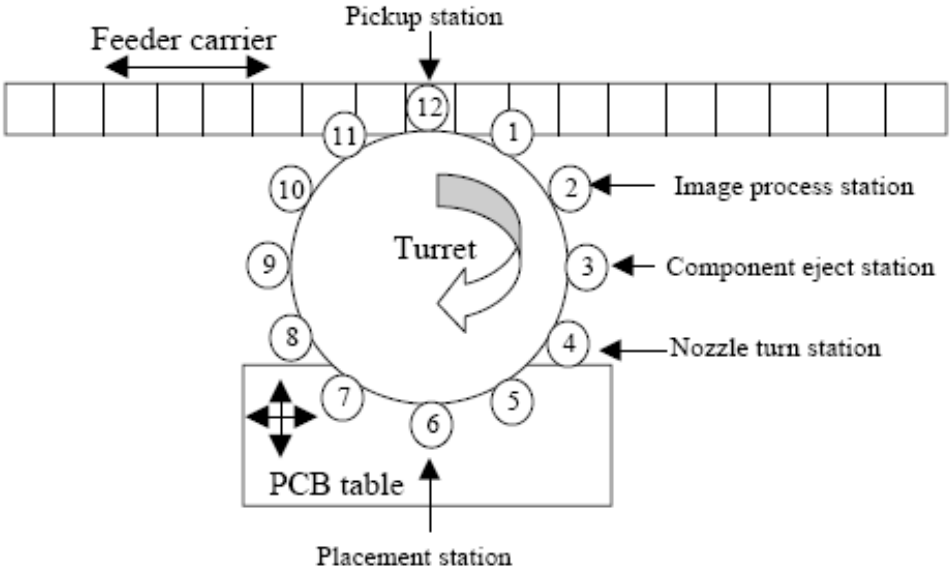


Figura 9. Máquina pick & place tipo torreta.

2.5.2.4. *Máquinas multicabezal (Multi-Head)*

Estas máquinas difieren de las máquinas tipo torreta en el mecanismo de transporte de componentes. En ellas se utiliza un cabezal X-Y para transportar los componentes desde los alimentadores fijos hasta la PCB también fija, mientras que el cabezal de tipo torreta, ésta gira siempre a la misma posición de recogida de componentes suministrados por los alimentadores dinámicos siendo la PCB (móvil) la que determinará la posición de colocado del componente [**10**].

Existen dos tipos de máquinas que presenta este funcionamiento. El primero tipo consta de una mesa PCB y un conjunto de alimentadores estáticos, con un brazo mecánico y un cabezal capaces de moverse de forma simultánea en las direcciones X e Y, para llevar a cabo las operaciones de *pick & place*. El otro tipo de máquinas tiene una mesa X-Y en movimiento y un carro de alimentadores móviles con un brazo mecánico y un cabezal que se trasladan entre puntos fijos de recogida y colocado.

El recorrido de los cabezales comienza recogiendo algunos componentes del alimentador (asumiendo que los cabezales están equipados con las correctas boquillas, de lo contrario se requiere un cambio de éstas) de forma secuencial. Luego, el cabezal y el brazo mecánico se

desplazan (en las direcciones X e Y simultáneamente) hasta la posición donde el componente será colocado, y luego el cabezal bajará (dirección Z) para colocar el componente sobre la placa. Tras ello, se retorna a la posición original para realizar el siguiente movimiento. Los cabezales de la máquina son similares a los cabezales de las máquina tipo torreta. Se diferencian en que están localizadas sobre el brazo y los lugares de recogida y colocado no están fijados.

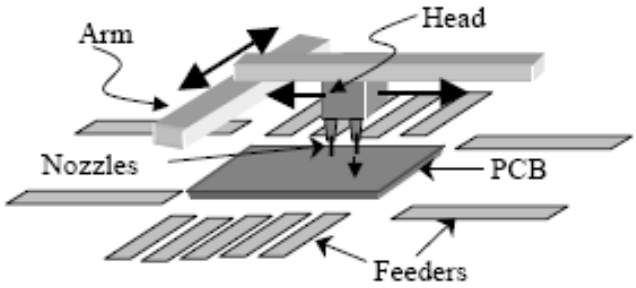


Figura 10. Máquina pick & place multicabezal.

2.5.2.5. Máquinas secuenciales (Sequential)

Este tipo de máquinas tienen un cabezal instalado en la terminación del brazo mecánico. Dicho brazo puede moverse en la dirección X, mientras que el cabezal puede hacerlo al mismo tiempo en la dirección Y. Por otro lado, la boquilla del cabezal se desplaza en la dirección Z para efectuar las operaciones de *pick & place*. La operación comienza con la elección de la boquilla adecuada por parte del brazo mecánico. Posteriormente, se recoge el componente especificado del alimentador y se coloca en el lugar correspondiente de la placa. Si las siguientes operaciones se efectúan con el mismo tipo de boquilla, el brazo se moverá directamente hasta el alimentador para continuar con las siguientes operaciones; en caso contrario, se cambiará la boquilla de forma automática.

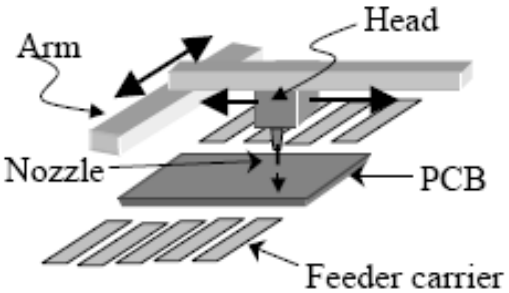


Figura 11. Máquina pick & place secuencial.

2.5.3. Cabezales de colocación

Los cabezales de colocación tienen muchas configuraciones diferentes dependiendo del tipo de equipo utilizado (puente o torreta). Cada cabezal consiste en una o más boquillas de vacío,

alguna clase de palpadores, uno o más husos y posiblemente un mecanismo de centrado, dependiendo de si el centro se logra internamente o externamente.

El cabezal de colocación es responsable de retirar el componente del alimentador, orientarlo correctamente, y colocarlo en la placa. El componente es recogido y transportado por una boquilla de vacío. Estas boquillas cubren un rango limitado de tipos y tamaños de componentes. Para verificar que el componente está realmente bien colocado en la boquilla se comprueba el vacío o se emplea un sistema óptico. Si el componente se pierde antes de la colocación, el sistema hará un nuevo intento, extrayendo otro componente y colocándolo en la placa [**11**].

El palpador es también un elemento esencial en este proceso ya que evita que un componente se aplaste entre la boquilla y la placa. De forma simplificada, el palpador detecta el golpe de la boquilla en el eje vertical (Z), la boquilla entonces libera el componente y vuelve a su posición de origen. El sistema palpador puede estar implementado como un mecanismo a base de muelles, con células de carga, mediante programa o una combinación de los tres. Los sistemas más baratos solamente pueden girar la boquilla en cuatro posiciones (0^o, 90^o, 180^o, 270^o), mientras que los sistemas más caros pueden colocar un componente con incrementos de giro de hasta 1^o.

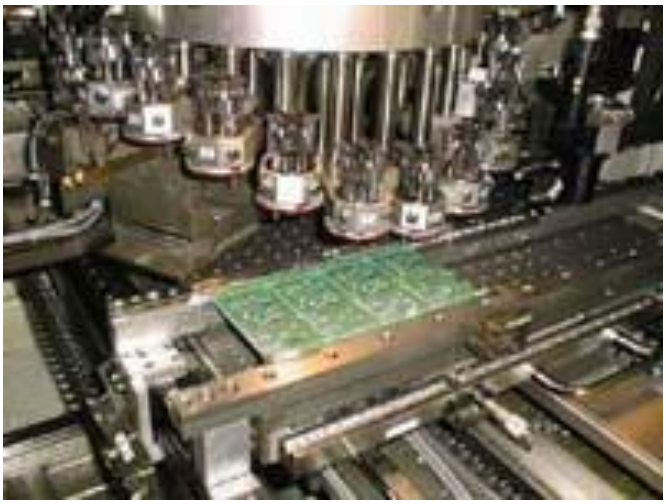


Figura 12. Cabezales de torreta giratoria.

2.5.4. Sistemas de colocación

Los sistemas de colocación se ven afectados directamente por el mecanismo de arrastre, los motores y el método de verificación.

Los mecanismos de transmisión típicos son cables, correas dentadas, tornillos roscados y tornillos de bolas. Los cables y correas dentadas son menos precisos y de poca repetitividad, mientras que los tornillos, normales o de bola, son más precisos y tienen mayor repetitividad [**12**].

Los motores de arrastre son de dos tipos: motores paso a paso o servomotores. Un motor paso a paso avanza el mecanismo de arrastre a incrementos uniformes hasta la posición correcta. Un servomotor avanza continuamente el mecanismo de arrastre hasta la posición correcta. Los motores paso a paso son menos precisos y más lentos que los servomotores. En cambio, los motores paso a paso son más baratos y por ello más utilizados.

Normalmente se emplean codificadores para verificar la posición. Básicamente, se utilizan dos tipos de codificadores: angulares y lineales. Los motores paso a paso usan codificadores angulares o lineales que cuentan pulsos, mientras que los servomotores usan codificadores angulares digitales que entregan un número binario.

2.5.5. Centrado del componente

En un programa de colocación, las coordenadas X-Y están basadas en las distancias desde el cero del programa (generalmente, el centro de uno de los taladros de referencia) al centro de la boquilla de vacío. Para situar correctamente un componente, el centro del componente tiene que estar alineado con el centro de la boquilla, o la distancia entre el centro de la boquilla y el centro del componente tiene que ser conocida.

El centrado se puede hacer internamente o externamente [**13**], como resume la figura 13.

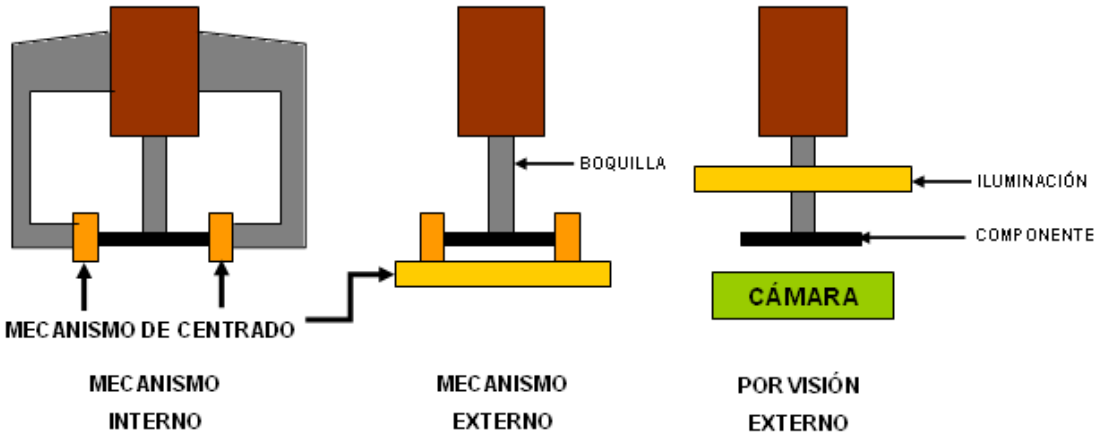


Figura 13. Métodos de centrado del componente.

El centrado interno significa que el mecanismo de centrado está localizado en el cabezal de colocación, mientras que centrado externo significa que el mecanismo de centrado está localizado en el armazón del sistema de colocación. Los primeros sistemas de centrado de

componentes fueron sistemas mecánicos. Los sistemas de colocación más avanzados usan sistemas de centrado basados en visión, especialmente para la colocación de componentes de paso fino.

2.5.5.1. Centrado mecánico

Un sistema mecánico de centrado de componentes intenta recolocar el componente en la boquilla después de retirarlo del alimentador de componentes. Este mecanismo consiste en cuatro garras o dedos con resortes que se cierran alrededor del componente y lo mueven a la posición adecuada. Después de la recolocación, el centroide del componente debe estar alineado con el centro de la boquilla.

Este método es fiable para la mayoría de los componentes. El centrado puede ocurrir internamente o externamente, habiéndose utilizado muchas técnicas diferentes para el centrado mecánico de componentes. El centrado mecánico está sujeto a las tolerancias y variaciones en los propios componentes. Esto no es importante con componentes que tienen una tolerancia de colocación grande.

Sin embargo, los componentes complejos, tales como un PLCC (*Plastic Leaded Chip Carrier*) grande, son más difíciles de centrar mecánicamente. Los componentes de paso fino son muy difíciles, si no imposibles, de centrar mecánicamente. Además de los problemas de centrado, los terminales de paso fino pueden resultar dañados por este mecanismo.

2.5.5.2. Centrado por visión

El centrado de componentes por visión es un método de centrado externo. Es una herramienta muy útil y potente en el proceso de colocación. El centrado por visión es imprescindible para la colocación de componentes de paso fino. En el sistema de colocación generalmente se emplea dos cámaras: una cámara interna enfocada hacia abajo para el reconocimiento de las referencias y un cámara CCD (*Charge-Coupled Device*) de enfocada hacia arriba para el reconocimiento de los componentes. Los principales parámetros de la cámara son su campo visual y su resolución. Los sistemas de visión que pueden ver todo el componente y sus terminales o su mayor parte, operan generalmente más rápido que los sistemas que tienen que completar una vista a base de distintas tomas. Sin embargo, la resolución de los sistemas con campo de visión más pequeño es mejor. La resolución se relaciona directamente con la cantidad de píxeles del CCD.

2.5.6. Alimentadores de componentes

Los alimentadores de componentes son uno de los elementos más críticos en un sistema de colocación. Los buenos alimentadores presentan las siguientes características [**13**]:

1. Utilizan una cantidad mínima de espacio en el sistema de colocación.
2. Proporcionan una alimentación de componentes suave y continua.
3. Permiten cargar componentes fácilmente.
4. Proporcionan posicionamiento preciso de componentes para su recogida por la boquilla.
5. Protegen al componente contra daños mecánicos y eléctricos.
6. Tienen fiabilidad máxima y mantenimiento mínimo.

Y tienen que lograr lo anterior a un precio razonable. Los alimentadores de componentes pueden clasificarse en cinco categorías: por gravedad, vibratorios, de embandado, de bandeja y con corte y preformado. Algunos ejemplos se pueden ver en la figura 14.

Figura 14. Ejemplo de alimentadores de componentes.

2.5.6.1. Alimentadores por gravedad

Estos alimentadores son simples y relativamente baratos ya que emplean la gravedad para llevar los componentes desde el tubo al punto de recogida. Cada alimentador consta de una

base y una o más pistas alimentadoras. Requieren menos espacio que los de embandado. Esto permite a un alimentador de gravedad proporcionar más entradas que un alimentador de embandado en la misma cantidad de espacio. El tubo de plástico que contiene los componentes se monta en la parte de atrás del alimentador en un plano inclinado. Los componentes se deslizan hacia abajo del tubo y en la pista alimentadora.

Los alimentadores de gravedad están diseñados para series pequeñas. Generalmente requieren frecuentes intervenciones del operador, ya sea para añadir componentes o para recolocar un componente mal posicionado. Incluso con una inclinación elevada, algunos componentes se adherirán al tubo. Los alimentadores de gravedad tienen el problema de permitir que los componentes se toquen unos a otros. Los componentes con terminales en dos lados, tales como los SOIC (*Small-Outline Integrated Circuit*), pueden tener problemas con las rebabas del encapsulado en los extremos de los componentes. También se pueden dañar las patas de los componentes con terminales en los cuatro lados, tales como los PLCC, debido a que los terminales de dos lados están en contacto unos con otros bajo presión.

2.5.6.2. Alimentadores vibratorios

Los alimentadores vibratorios son similares a los de gravedad, excepto porque los componentes se mueven a través del alimentador por vibración más que por gravedad. Al igual que los alimentadores de gravedad, estos requieren menos espacio que los alimentadores de embandado. Estos alimentadores consisten en una base vibrante y una o más pistas alimentadoras. Algunos alimentadores insertan un tubo de metal, para contener los componentes, en la pista. Los tubos se montan en la parte de atrás en un plano inclinado para alimentar los componentes en la pista vibradora. Una vez están en la pista, los componentes se deslizan por vibración hasta el punto de recogida.

Estos alimentadores están diseñados también para su aplicación en series pequeñas. Tienen los mismos problemas que los de gravedad, frecuente intervención del operador para añadir componentes o corregir alimentadores defectuosos y que los componentes están en contacto continuo unos con otros. Además, si el componente está vibrando excesivamente en el punto de recogida, la boquilla puede tener dificultades para recoger el componente.

2.5.6.3. Alimentadores de embandado

Los alimentadores de embandado son los más comunes y fiables. Estos alimentadores arrastran una cinta de soporte, que contiene los componentes, a través del alimentador hasta el

punto de recogida. Las cintas de soporte dan la mejor protección a los componentes. La mayoría de los alimentadores de embandado están formados por una unidad monolítica que consta de un soporte de carrete, un mecanismo de arrastre de la cinta de soporte y un mecanismo para despegar la cinta de cobertura. El mecanismo de arrastre puede ser activado mecánica, neumática o eléctricamente. Para evitar que los componentes se salgan de la cinta de soporte durante el arrastre, el alimentador debe arrastrar la cinta primero y después despegar la cinta de cobertura.

Los alimentadores de embandado son apropiados para cualquier volumen de producción, pero son particularmente adecuados para series grandes. El problema ha sido la disponibilidad de componentes, especialmente circuitos integrados, en embandado. Los suministradores de componentes exigen un pedido mínimo determinado para suministrar el componente en embandado y en algunos casos el pedido mínimo excede las necesidades del usuario.

2.5.6.4. Alimentadores de bandeja

Los alimentadores de bandeja han aparecido principalmente para dar soporte a la tecnología de paso fino. La naturaleza delicada de los terminales de paso fino requiere que los componentes estén empaquetados de tal forma que sus terminales no estén en contacto con los terminales ni el cuerpo de otros componentes. Las bandejas también se han usado para alimentar otros tipos de componentes, principalmente en aplicaciones de bajo volumen. Los alimentadores de bandeja pueden ser pequeños y simples o grandes y complejos. Un alimentador de bandeja simple puede consistir en una plataforma de montaje en el bloque alimentador a la que el cabezal de colocación pueda acceder directamente, mientras que los sistemas más complejos consisten en un alimentador de varios niveles con un mecanismo de lanzadera para llevar el componente hasta el cabezal de colocación.

2.5.6.5. Alimentadores con corte y preformado

Los alimentadores que cortan y preforman los terminales durante el proceso de alimentación son caros y complejos. Se usan para los componentes de paso muy fino, como los encapsulados de anillo de soporte moldeado (*TapePak*). Estos alimentadores consisten en un mecanismo que traslada el componente desde un tubo de plástico hasta el alimentador y un mecanismo que separa el componente del anillo de soporte, preforma los terminales de forma de la de gaviota y desecha el anillo. Estos alimentadores solo pueden manejar un tamaño de componente cada vez y requieren un espacio considerable en el sistema de colocación.

CAPÍTULO 3. HERRAMIENTAS UTILIZADAS

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

3.1. Introducción

En este capítulo se describirá brevemente las herramientas de ingeniería hardware y software utilizadas en el desarrollo de este proyecto. También se hará una descripción de las características generales del lenguaje utilizado para codificar la aplicación, C#. Para terminar se presentará el flujo de desarrollo seguido.

3.2. Herramientas hardware

3.2.1. Analizador lógico Link Instruments IO3208A

3.2.1.1. Introducción

El analizador lógico y generador de patrones Link Instruments IO3208A [**14**] es un dispositivo que permite seguir la evolución y registrar un grupo de señales y sus relaciones temporales, entre otras funciones. Aunque se ha clasificado esta herramienta como un dispositivo hardware, se trata de un dispositivo diseñado para ser conectado a un ordenador y trabajar en colaboración con una aplicación software, proporcionada por el fabricante, que permite controlar sus funciones y realizar una cómoda visualización y análisis de los datos capturados.

La conexión se realiza a través de uno de los puertos USB instalados en el ordenador, por lo que puede utilizarse prácticamente cualquier ordenador personal actual de sobremesa o portátil. Sin embargo, el software de control sólo se encuentra en versiones para los sistemas operativos Windows de Microsoft, por lo que su usabilidad se limita a equipos con estos sistemas operativos instalados.

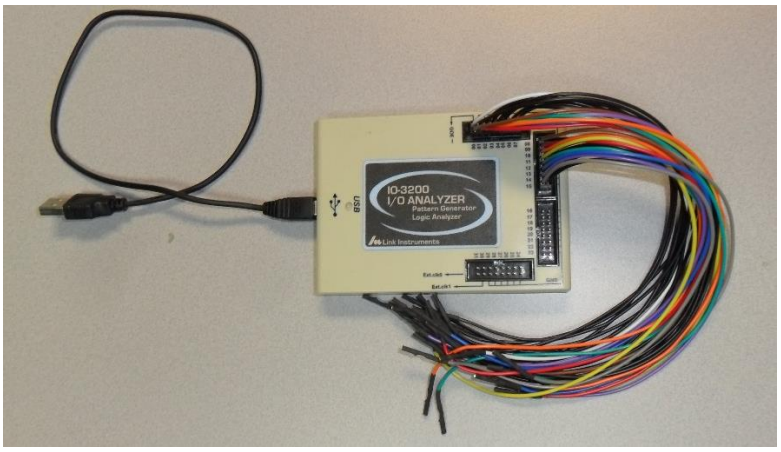


Figura 15. Analizador lógico Link Instruments IO3208A.

3.2.1.2. Análisis realizados

Este analizador lógico se ha utilizado sobre todo en algunas de las pruebas iniciales de comunicación con la máquina, para depurar el programa de pruebas que se estaba desarrollando. El problema surgido fue que, al enviar comandos a la máquina, esta parecía responder correctamente a la mayoría de ellos, pero no se obtenía casi nunca una confirmación de que el comando se ejecutó correctamente o un error en caso de que no fuese así.

Previamente se habían monitorizado las comunicaciones del puerto serie, entre la máquina pick & place y el ordenador de control, mediante la herramienta software *Free Serial*

Port Monitor (descrita más adelante en este capítulo). Se monitorizaron dichas comunicaciones tanto con el software original de la máquina (shaupt.exe) como con nuestro programa de pruebas, comprobando que había una diferencia importante en la cantidad de mensajes de confirmación en sentido máquina → PC.

El analizador lógico fue muy útil para averiguar si realmente la máquina estaba omitiendo los mensajes de confirmación o si los estaba produciendo, pero nuestro programa de pruebas era incapaz de recibirlos. Además, sirvió para obtener una estimación aproximada del tiempo de respuesta de la máquina, utilizando el analizador para ver en qué instante se producían dichas respuestas tras recibir y ejecutar los comandos que le íbamos enviando.

En la figura 16 se puede ver la captura de datos realizada al enviar un comando desde el software original de la máquina, mientras que en la figura 17 se puede ver la captura de datos realizada al enviar un comando desde nuestra aplicación de pruebas. La captura en ambos casos está focalizada en el momento en que se muestra el mensaje devuelto por la máquina al PC.

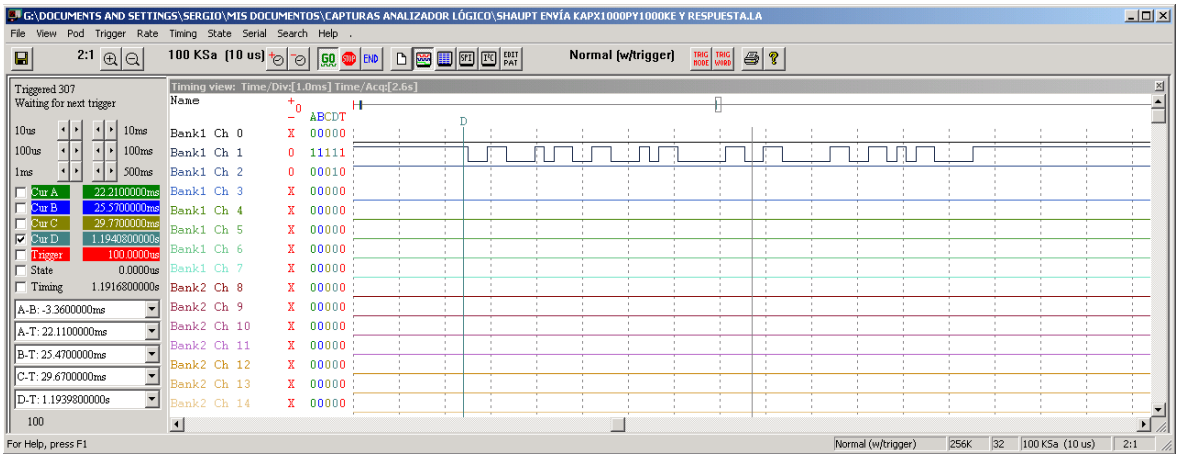


Figura 16. Captura del puerto serie con el software original de la máquina (respuesta).

Superponiéndolos se puede ver que son idénticos y, tras decodificarlos, se verificó que eran la secuencia hexadecimal 46 46 30 30 0D, que corresponde a los caracteres FF00 y el carácter de retorno de carro. Por tanto la señal observada en ambas capturas corresponde con el código con el que la máquina indica que el comando se ejecutó correctamente.

Poder observar las señales eléctricas en el hardware del puerto RS-232 permitió comprobar que la máquina no distinguía entre la aplicación original y nuestra aplicación de pruebas, puesto que enviaba siempre los mismos mensajes de confirmación o de error. Sin embargo, estos mensajes no estaban llegando a la ventana de nuestro programa de pruebas que debía mostrarlos ni al monitor software del puerto serie.

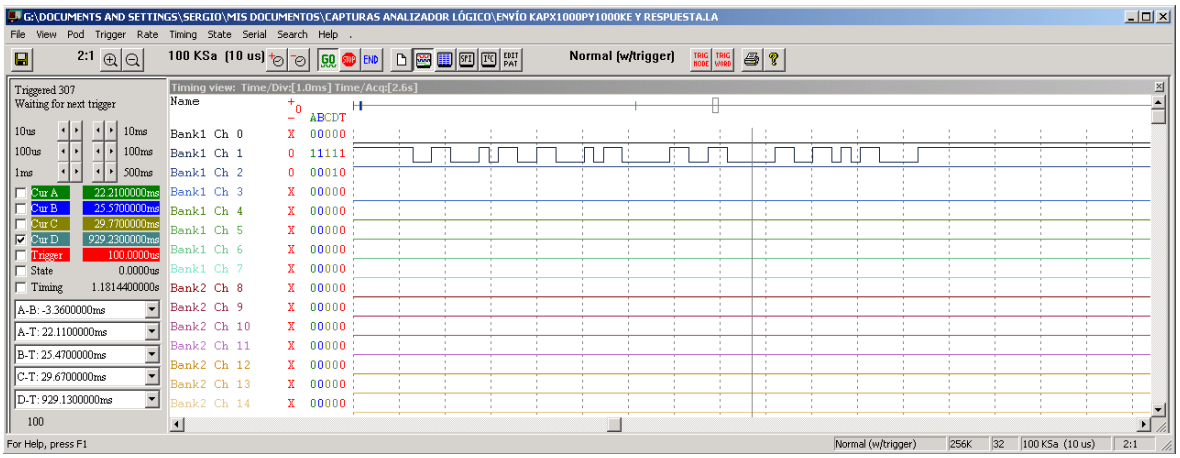


Figura 17. Captura del puerto serie con nuestra aplicación de pruebas (respuesta).

Esto hizo que se orientase la búsqueda del error hacia la forma en que se estaban capturando estos datos desde nuestra aplicación de pruebas, llevándonos a descubrir la causa del error. Esta causa estaba motivada por la forma en que funciona el *framework* de .NET, utilizado desde la aplicación de pruebas que se estaba desarrollando en C#.

Para la recepción de mensajes, se programó un evento que se dispara cada vez que se reciben datos por el puerto serie, para que llamara a la función encargada de recoger los datos recibidos y mostrarlos en una ventana. Cuando se reciben datos a través del puerto serie y se genera el evento que lo notifica a nuestra aplicación, el *framework* ejecuta la función de respuesta al evento que habíamos programado en un nuevo hilo de ejecución, pasando así a haber al menos dos hilos de ejecución: el hilo inicial de nuestro programa y este nuevo hilo que ejecuta en paralelo la función de recepción de datos.

Un problema que surge aquí es que, desde la versión 2 de .NET, no está permitido acceder desde un hilo a los objetos creados en otro hilo, por lo que la función de recepción de datos no puede acceder directamente a la ventana donde debe mostrar los datos.

Este problema se resolvió usando en nuestro segundo hilo un mecanismo de *callback* a una función en el hilo que creó la propia ventana, usando un delegado. Un delegado, básicamente, es un puntero a función capaz de funcionar dentro del código gestionado, seguro en el *framework* .NET, a diferencia de los punteros a función normales, que son muy inseguros.

Sin embargo, tras resolver este problema, se hizo patente que, aunque esta solución funcionaba en la aplicación de pruebas, con la que se estaba analizando el funcionamiento de la máquina, para la aplicación definitiva, el usar un segundo hilo que se ejecute en paralelo para recibir los datos no tiene ninguna ventaja, complicando el diseño innecesariamente.

Esto se debe a que el funcionamiento de la máquina se basa en un comportamiento de comando → respuesta. Este tipo de comportamiento es secuencial, pues tras enviar cada comando a la máquina, hay que esperar la respuesta correspondiente (confirmación o error) antes de enviar otro comando o realizar otra acción. Además, ni nuestra aplicación ni el ordenador de control tienen que realizar tareas computacionalmente pesadas, por lo que no es necesario aprovechar el tiempo entre el comando enviado y la respuesta recibida, haciendo inútil el funcionamiento en paralelo de la tarea de recepción de datos.

Esto nos llevó a la conclusión de que era mucho mejor para nuestros propósitos no utilizar el mecanismo de eventos para recibir datos desde el puerto RS-232 en la aplicación definitiva. En su lugar se utilizaron instrucciones alternativas que permiten realizar dicha recepción de datos secuencialmente, a continuación de la transmisión del comando, programando un tiempo de espera máximo (*timeout*).

3.2.2. Digitalizadora de vídeo

Para adquirir la señal de vídeo generada por la máquina se ha usado una digitalizadora de vídeo de la marca *AVerMedia*, modelo *AVerTV Volar Video Capture USB*, mostrada en la figura 18. Se trata de un dispositivo concebido principalmente para ver en el ordenador la señal de televisión digital terrestre, pero que incorpora además la posibilidad de digitalizar desde fuentes de señal de vídeo compuesto y S-Video.

El dispositivo se conecta al ordenador mediante un puerto USB y dispone de drivers para Windows XP, Windows 7, Windows 8 y Linux, por lo que el cambio de un equipo informático a otro durante el desarrollo y las pruebas no ha supuesto prácticamente ningún problema, salvo instalar previamente el driver en cada equipo que se iba a usar.

Figura 18. Digitalizadora de vídeo.

Este pequeño dispositivo tiene en total tres conectores, como se puede ver en el esquema de conexiones del manual de instrucciones (figura 19):

- Puerto USB: para conectar al ordenador y entregar la señal capturada.
- Conexión de antena estándar de 75Ω: Para conectar una antena para recibir la señal de televisión (TDT).

- Conector combinado S-Video (*Separated-Video* [**15**]) / vídeo compuesto + audio: Para conectar una fuente de vídeo local.

El conector de antena estándar de 75Ω se encuentra en la parte posterior y el conector combinado está en un lateral, permitiendo enchufar el cable que contiene los distintos conectores: el conector mini-DIN de 4 pines para S-Video y las entradas RCA para vídeo compuesto y audio estéreo.

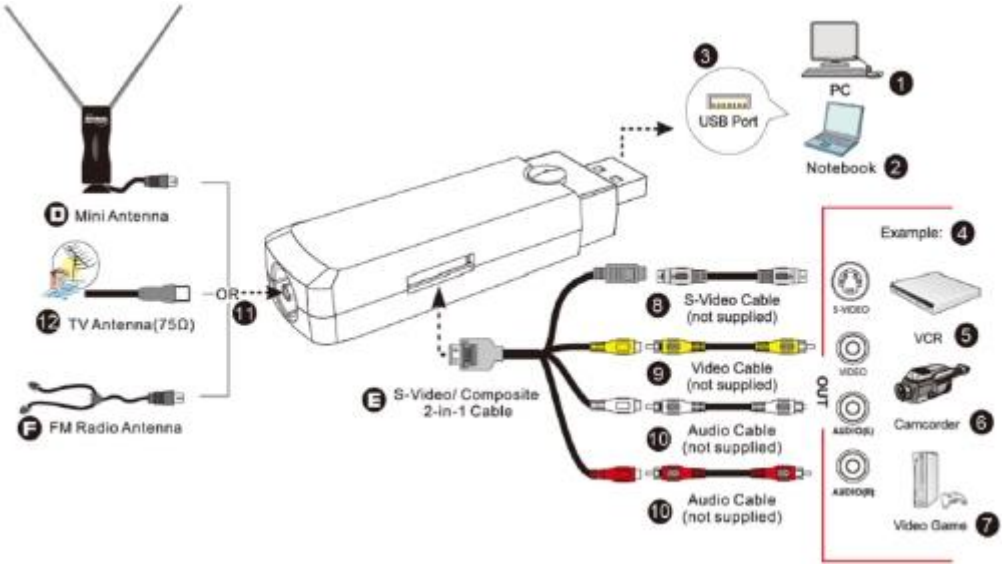


Figura 19. Esquema de conexiones de la digitalizadora de vídeo.

En nuestro caso, sólo hemos usado el conector de entrada de vídeo compuesto, que es el amarillo, ignorando el resto. Como la salida de vídeo compuesto de la máquina tiene un conector BNC y el cable para la conexión de vídeo compuesto de que disponíamos era RCA, se montó un pequeño adaptador, utilizando un conector BNC macho, un conector RCA hembra y un trozo pequeño de cable coaxial de 75Ω.

Para las ocasiones en que no bastaba con la longitud del cable de vídeo RCA, se utilizó como prolongación el cable original que conectaba la máquina con su monitor de vídeo. Como este cable tiene conectores BNC machos en ambos extremos, se utilizó un adaptador BNC hembra-hembra para la unión. En la figura 20 se muestra el conjunto de elementos utilizados con este fin durante el desarrollo de este proyecto.

Puesto que la solución creada no depende de ninguna característica en particular de este dispositivo, es de esperar que su funcionamiento no se vea alterado por usar un modelo diferente, en caso de sustitución.

Figura 20. Digitalizadora de vídeo, cables y adaptadores utilizados.

3.3. Herramientas software

3.3.1. Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) creado por Microsoft para la creación de software utilizando su tecnología .NET Framework.

Se trata de un completo conjunto de herramientas para la creación tanto de aplicaciones de escritorio como de aplicaciones web empresariales para trabajo en equipo. Aparte de generar aplicaciones de escritorio de alto rendimiento, se pueden utilizar las eficaces herramientas de desarrollo basado en componentes y otras tecnologías de Visual Studio para simplificar el diseño, desarrollo e implementación en equipo de soluciones empresariales.

Este entorno permite la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C# y Visual C++ utilizan todos el mismo entorno de desarrollo integrado, que habilita el uso compartido de herramientas y hace más sencilla la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes utilizan las funciones de .NET Framework, las cuales ofrecen acceso a tecnologías clave para simplificar el desarrollo de aplicaciones. En la figura 21 se muestra este entorno de desarrollo mostrando parte del código fuente del proyecto.

Durante la realización de este proyecto se ha utilizado este entorno de desarrollo como sistema principal para la edición, prueba y depuración del código fuente en C# de la aplicación de control de las máquinas pick & place.

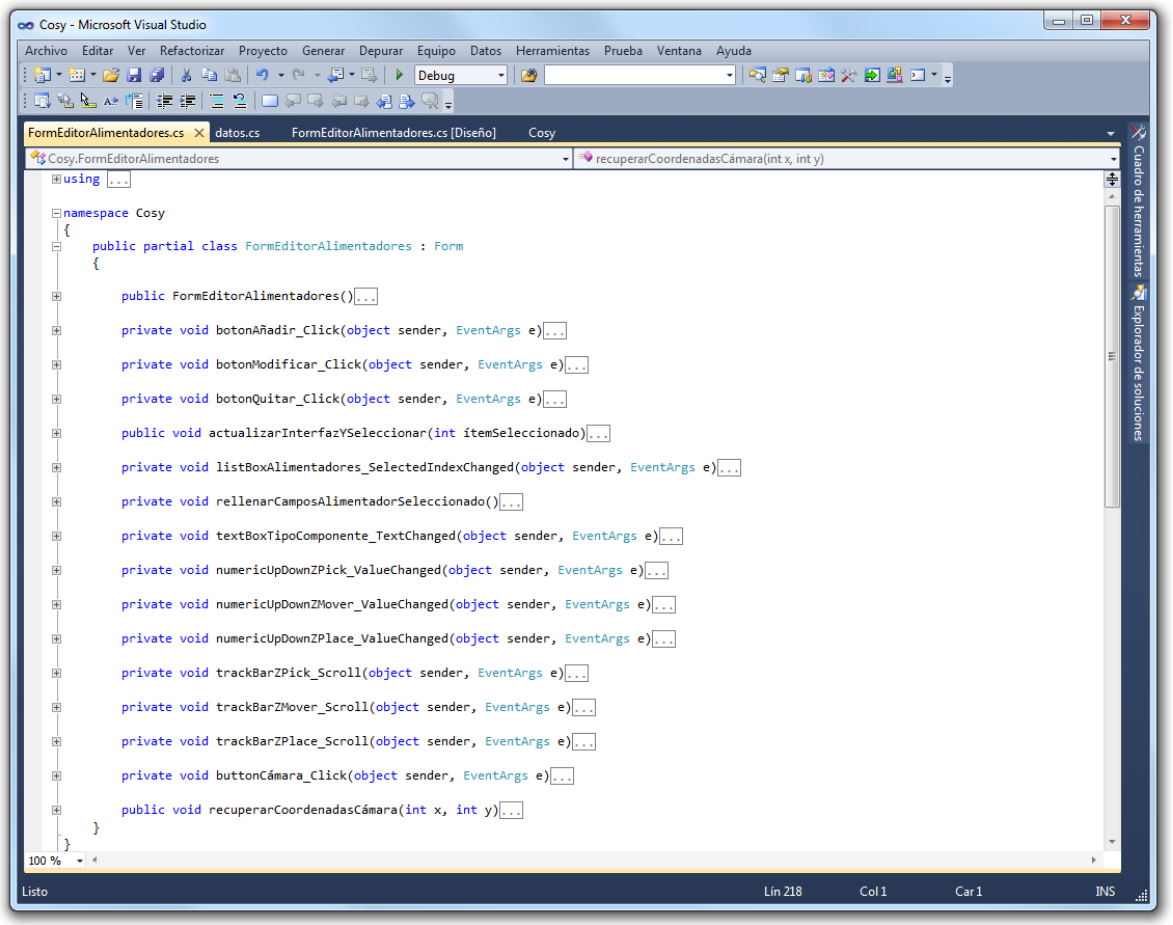


Figura 21. Microsoft Visual Studio mostrando parte del código fuente del proyecto.

También se ha utilizado para realizar algunos cálculos y simulaciones útiles para tomar decisiones durante el desarrollo del programa. Por ejemplo, para estimar de forma práctica la magnitud máxima de los errores producidos al truncar decimales en los ángulos. Se comprobó que, efectivamente, para placas grandes es necesario conservar cierto grado de precisión en el ángulo de giro, puesto que, en caso contrario, algunos componentes alejados del origen de la placa quedarían descolocados apreciablemente, lo que produciría errores en el montaje de la placa.

La primera versión de este entorno de desarrollo data del año 1997, sin embargo, Microsoft se ha ido encargando de liberar una nueva versión de su software cada poco tiempo. Estas nuevas versiones suelen incluir además algún cambio en el número de lenguajes soportados o la actualización de estos lenguajes para incluir nuevas características incrementando las capacidades del entorno constantemente.

La versión de Visual Studio utilizada durante este proyecto ha sido Microsoft Visual Studio 2010 Professional 32 bits.

3.3.2. Free Serial Port Monitor

3.3.2.1. Descripción

De entre los diversos programas existentes que permiten analizar las comunicaciones a través del puerto serie hemos utilizado Free Serial Port Monitor, desarrollado por HDD Software [**16**], porque funciona razonablemente bien para el uso que pretendíamos darle y se puede descargar de la página web de los autores y usar libremente según su licencia [**17**].

El objetivo de usar este programa era registrar todos los datos enviados y recibidos a través del puerto serie para permitirnos analizar de manera sencilla las comunicaciones entre el PC y la Cosy. De esta forma, primero lo utilizamos para tratar de intuir el protocolo original utilizado por el programa Shaupt.exe para gobernar la máquina Pick & Place y posteriormente lo utilizamos para comprobar que los comandos generados por nuestras pruebas y nuestra aplicación eran correctos a bajo nivel.

La interfaz de usuario usa las características generales de los programas desarrollados para Windows, por lo que es fácil de utilizar. Sólo hay que tener en cuenta algunos detalles de operación, como son: empezar siempre a monitorizar el puerto ***antes*** de iniciar la aplicación que va a usar el puerto serie y terminar la monitorización ***después*** de haber cerrado la aplicación. De otro modo, es posible que el programa monitor o el monitorizado se queden "congelados" y sea necesario matar el proceso desde el administrador de tareas de Windows.

De los diferentes modos de operación que tiene el programa y los varios grados de detalle que es capaz de proporcionar, el modo más útil y cómodo para nuestros propósitos es el que muestra como un texto continuo (en hexadecimal y ASCII) las secuencias de datos enviadas y recibidas, pero no muestra detalles de las señales implicadas en la comunicación (IOCTL, estado de las líneas de control, etc.). En la figura 22 se muestra la interfaz de usuario con el programa en acción mientras se realizaba una de las pruebas durante el desarrollo del proyecto.

3.3.2.2. Funcionamiento del programa monitor

El programa es una aplicación combinada de nivel de usuario y de nivel del núcleo destinado a los procesos que operan con la API Win32. El programa puede cambiar la forma estándar de operar de los puertos serie, permitiendo acceder a éstos aun cuando un puerto serie normalmente sólo podría ser usado por una única aplicación en un momento dado.

Está funcionalidad se logra mediante un driver que se instala sobre el driver del puerto serie estándar en el sistema operativo Windows. Este driver recoge la información completa

incluyendo los datos transmitidos y recibidos, así como las señales de control enviadas al equipo conectado a través del puerto por la aplicación de usuario y recibidas del mismo.

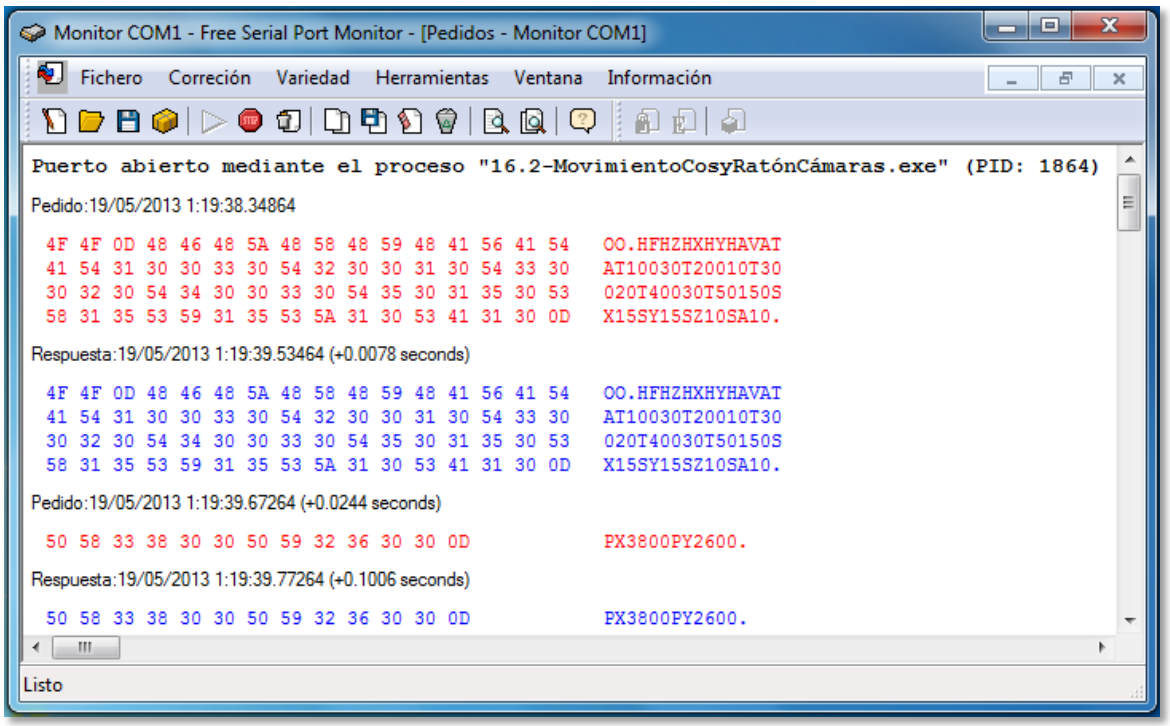


Figura 22. Interfaz de usuario del programa Free Serial Port Monitor mostrando una de las pruebas.

Esta herramienta guarda, además de todos los datos y las señales mencionadas, marcas de tiempo que permiten analizar con exactitud el flujo de señales en el puerto. Además, a diferencia de otros, no altera apreciablemente dichos tiempos, por lo que permite analizar dichas temporizaciones con bastante fiabilidad.

La versión del programa usada durante el proyecto fue desarrollada originalmente para funcionar en Windows NT, Windows 2000 y Windows XP. Sin embargo, según hemos podido comprobar, se puede hacer funcionar también en Windows 7, siempre que el usuario disponga de los permisos adecuados en el sistema operativo.

3.3.3. Altium

Altium Designer es un conjunto de programas para el diseño electrónico en todas sus fases y para todas las disciplinas, ya sean esquemas, simulación, diseño de circuitos impresos, implementación de FPGA, o desarrollo de código para microprocesadores.

No se trata de un conjunto de paquetes sueltos vendidos como una suite y conectados mediante archivos externos, sino de un programa único que crea un entorno y comunica al

usuario con los distintos servidores (por ejemplo el editor de texto, el editor de esquemas o el editor de PCB).

Las características más relevantes de este programa son:

- Conexión a base de datos: Los componentes se extraen, como siempre, de bibliotecas con los distintos tipos de modelos: símbolo, modelo pspice, modelo ibis, modelo 3D, footprint, etc. Estas bibliotecas se puede conectar a una base de datos con conectores ms-jet o bien OCBD, de modo que no hay que preocuparse de mantener los atributos y nos aseguramos que siempre se comprará el código correcto sin tener que pasar el BOM por terceros programas que escojan el código en la base de datos de compras.
- Visores de PCB y Gerbers: Los circuitos impresos dibujados pueden ser visualizados en éste módulo.
- Simulador mixto SPICE: El módulo Device Intelligence incluye un simulador Spice compatible con modelos Pspice, fácilmente obtenibles de los fabricantes.
- Síntesis y simulación FPGA. Conexión JTAG: También se pueden crear proyectos para implementarse en una FPGA, mediante esquemas, código VHDL o Verilog. Se pueden sintetizar y simular en modo lógico/funcional. Si el proyecto pasa la simulación sin errores, se puede volcar a la placa con la FPGA, siempre que la placa disponga de conexión JTAG. También puede depurarse el proyecto con la misma técnica.
- Signal integrity basado en esquema: La simulación de integridad de señales es más adecuada cuando tenemos la placa terminada. Pero también es conveniente comprobar, aún sin tener la placa, cuáles serán las señales que peor se comportan (con una longitud ficticia o media). Esto nos permitirá editar las reglas de longitud para las señales y que se pueda elegir la colocación de componentes más adecuada.
- Un sistema completo de edición de circuitos impresos basado en reglas. Con potentes visores y mensajes. Estos son necesarios para la última tecnología de placas multicapa con vías/microvías enterradas y ciegas.
- El enrutado interactivo es totalmente personalizable con enrutado de arcos, pares diferenciales, ajuste de anchos y vías, modo empujar pistas, eliminación de lazos. El autorouter es de tecnología topológica, un paso más en la tecnología basada en formas que permite trazar situaciones de conectores con pines en zigzag o componentes colocados a 45 grados. Permite enrutar toda la placa o distintas zonas o nets.

- Cantidad de utilidades de postprocesado: gotas, ajuste de longitudes, apantallado de señales, part & pin swapping, entre otras.
- Un completo editor de Gerber/ODB++/taladrado/fresado con utilidades y panelización, así como visualizador 3D que permite una visión real de la tarjeta en tres dimensiones. Soporte MCAD-ECAD para modelos STEP y comprobación de espacio libre en tiempo real.

Para nuestra aplicación, el mayor interés en este programa se puso en sus archivos de salida. A partir del análisis de éstos, la aplicación creada obtiene las características relevantes de los componentes para importarlas a una placa, permitiendo así usar la máquina pick & place para montarlos.

3.3.4. Processing

Processing [**18**] es un lenguaje de programación y un entorno de desarrollo integrado que permite crear y probar programas muy rápidamente. Su utilización es muy sencilla y por ello sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos, principalmente en el mundo del arte y el diseño digital.

Processing es un sistema de código abierto basado en Java que se distribuye bajo la licencia GNU GPL para los sistemas operativos Windows, GNU/Linux y Mac OS X. Actualmente *Processing* dispone de más de cien bibliotecas que extienden enormemente las capacidades originales del lenguaje.

En este proyecto se ha usado *Processing* para probar la corrección de los cálculos para determinar la posición de los componentes sobre el panel de la Cosy dadas su ubicación y giro en la placa y la ubicación y giro de la placa en el panel. En concreto se ha usado como un medio muy rápido para comprobar visualmente la exactitud de las suposiciones y desarrollos iniciales sobre el papel, antes de tener la suficiente soltura con Visual Studio para hacerlo en este IDE.

Se creó una placa a partir de unos parámetros dados y se pobló con componentes creados automáticamente con todos sus parámetros aleatorios, permitiendo mover tanto la placa como cualquier componente sobre ella haciendo clic y arrastrando con el ratón. Además, se permite cambiar el ángulo de giro de la placa o cualquier componente mediante la rueda del ratón.

A partir de los primeros bocetos (*sketch* en la terminología de *Processing*), se afinaron algunos detalles que habían sido pasados por alto al decidir usar un sistema de coordenadas ligeramente diferente al sistema tradicional para representar las relaciones geométricas y

trigonométricas. Este sistema de coordenadas y las ecuaciones que permiten hacer los cálculos para una representación correcta se detallan en el capítulo dedicado al desarrollo de la solución.

En la figura 23 se puede la ventana de ejecución de *Processing* junto a la ventana de editor con el *sketch* creado para comprobar visualmente los cálculos de posicionamiento. Con estas pruebas se consiguió que el cálculo de coordenadas para la representación visual y para la colocación física del componente sobre la placa fuese correcto con cualquier desplazamiento y ángulo tanto del componente como de la placa. Posteriormente se usaron las fórmulas ajustadas en la aplicación desarrollada en C#, tanto para las representaciones gráficas en pantalla como para el montaje físico de los componentes en la sección de producción.

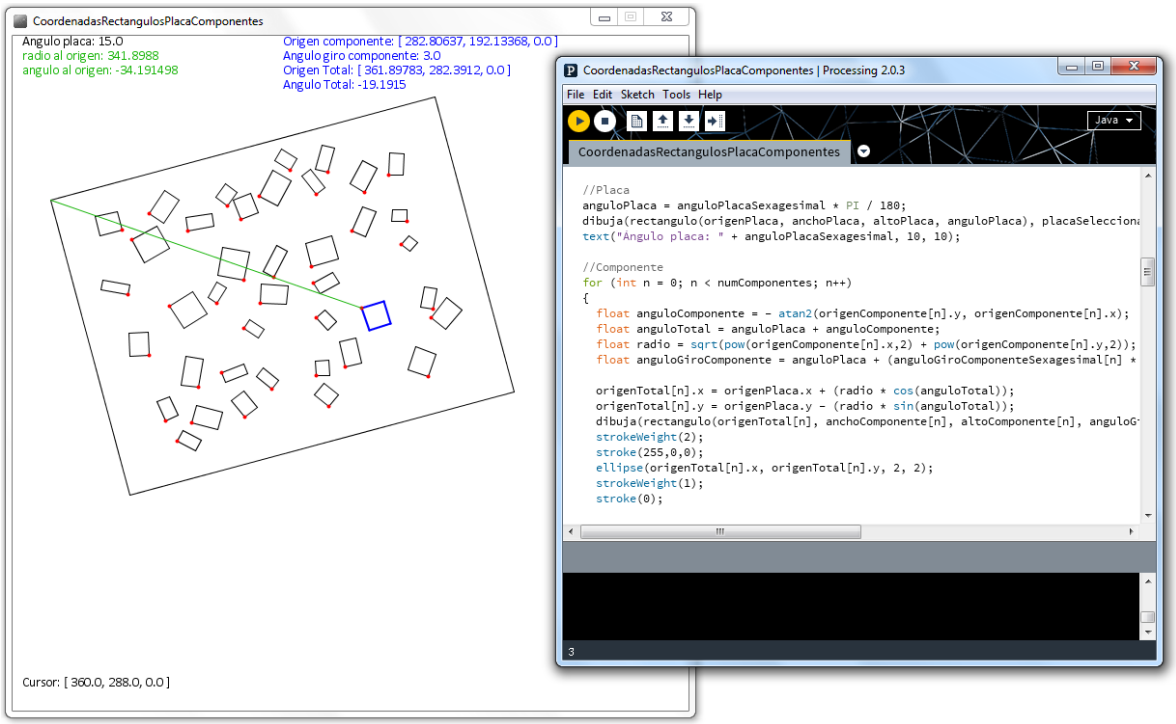


Figura 23. *Sketch* de *Processing* creado, ejecutándose, y editor con el código fuente.

3.3.5. Framework AForge.NET

AForge.NET es un *framework* para C# diseñado para desarrolladores e investigadores en los campos de la visión computerizada y la inteligencia artificial (procesado de imágenes, redes neuronales, algoritmos genéticos, algoritmos de aprendizaje, robótica, etc.)

Este *framework* ofrece numerosas prestaciones de las que probablemente, en la mayoría de los casos, el desarrollador sólo use algunas. Sin embargo, como se encuentra dividido en multitud de espacios de nombres, almacenados en diferentes archivos de biblioteca de enlace dinámico (DLL), sólo es necesario incluir en nuestro proyecto los que necesitamos, evitando saturar la memoria del ordenador con código que no se va a usar.

En nuestro caso, las bibliotecas incluidas se han usado para acceder fácilmente a las imágenes proporcionadas por la cámara de vídeo, incluida en el brazo de la máquina pick & place. En la figura 24 se puede ver la página de ejemplos de uso de fuentes de vídeo en C#, utilizando AForge.NET.

Figura 24. Página de ejemplos de uso de fuentes de vídeo de AForge.NET.

Como se mencionó anteriormente, este *framework* no sólo tiene funciones para acceder a fuentes de vídeo. Para hacernos una idea de las posibilidades de este *framework* mencionaremos brevemente la utilidad de algunos de sus principales espacios de nombres [**19**]:

AForge: El espacio de nombres AForge es el núcleo del *framework* y contiene las principales clases usadas por otros espacios de nombres del *framework* y clases, que además pueden ser usadas independientemente por el desarrollador con diversos propósitos. Entre otros, contiene estructuras para manejar puntos y rangos, un evaluador de expresiones en notación polaca inversa (RPN) y soporte para computación paralela.

AForge.Controls: Este espacio de nombres contiene diferentes controles de interfaz de usuario, que pueden ser utilizados junto a otras clases del *framework*. Incluye diversos controles para seleccionar colores, crear histogramas, mostrar imágenes, reproducción de vídeo y leer datos desde un joystick u otro dispositivo de manipulación, entre otros.

AForge.Fuzzy: Este espacio de nombres contiene un grupo de interfaces y clases para operar con conjuntos difusos.

AForge.Genetic: Este espacio de nombres contiene interfaces y clases para computación genética. Tanto este espacio de nombre como los espacios de nombres que contiene tienen clases que permiten resolver diferentes problemas (optimización, aproximación, predicción, etc.) con la ayuda de algoritmos genéticos, programación genética y programación de expresiones de genes (GEP).

AForge.Imaging: Este espacio de nombres y los que están dentro de él, contienen clases e interfaces para diferentes rutinas de procesado de imagen, como reducción de color, transformaciones, filtros en transformada compleja de Fourier, manejo de diferentes formatos de imagen, generación de texturas, etc.

AForge.MachineLearning: Este espacio de nombres contiene varias clases e interfaces para diferentes algoritmos de aprendizaje.

AForge.Math: Este espacio de nombres y los que contiene a su vez, contienen juegos de herramientas matemáticas que se usan en otros espacios de nombres de este framework, aunque también se pueden usar independientemente. Incluye, entre otras, interfaces y clases para implementar diversas funciones geométricas, transformadas de Fourier, funciones estadísticas, generación de números aleatorios y métricas de diferencia y similitud.

AForge.Neuro: Este espacio de nombres contiene clases e interfaces para computación usando redes neuronales. Incluidos en él y en los espacios de nombres que contiene hay clases e interfaces que permiten la creación de arquitecturas de redes neuronales usuales, así como clases para entrenar estas redes, tanto con aprendizaje supervisado como no supervisado.

AForge.Robotics: Este espacio de nombres contiene a su vez varios espacios de nombres con clases e interfaces que permiten manipular dispositivos robóticos Lego Mindstorm, kits de la compañía Surveyor y otros.

AForge.Vision.Motion: Este espacio de nombres contiene clases e interfaces usadas para detección de movimiento y procesado de señales de vídeo.

AForge.Video: Este espacio de nombres y los espacios de nombres que contiene a su vez contienen clases e interfaces para acceder a varias fuentes y dispositivos de vídeo, detección de movimiento y procesado de señales de vídeo. Este es uno de los espacios de nombres de este

framework que se usa en este proyecto, por lo que describiremos a continuación los principales espacios de nombres que contiene a su vez:

- ***AForge.Video.DirectShow***: Este espacio de nombres contiene clases que permiten acceder a fuentes de vídeo usando el interfaz *DirectShow*.

- ***AForge.Video.FFMPEG***: Este espacio de nombres contiene clases que permiten leer y escribir ficheros de vídeo por medio de la biblioteca FFmpeg.

- ***AForge.Video.Kinect***: Este espacio de nombres contiene clases que permiten acceder al vídeo y los datos de profundidad del dispositivo Microsoft Xbox Kinect.

- ***AForge.Video.VFW***: Este espacio de nombres contiene clases que permiten leer y escribir en ficheros AVI usando la interfaz *Video For Windows*.

- ***AForge.Video.Ximea***: Este espacio de nombres proporciona clases que permiten capturar imágenes de cámaras Ximea.

De los espacios de nombres mencionados, los que se han usado en la aplicación desarrollada en este proyecto son *AForge.Video* y *AForge.Video.DirectShow*. Por tanto, para poder utilizar las bibliotecas necesarias, en el directorio de nuestro proyecto se han incluido los archivos *AForge.Video.dll* y *AForge.Video.DirectShow.dll*, que contienen los espacios de nombres mencionados anteriormente.

3.3.6. iTextSharp

iTextSharp [**20**] es una biblioteca open source que permite crear y manipular archivos PDF desde código. Se trata de la adaptación al framework .NET de otra biblioteca muy conocida, llamada iText, escrita originalmente para el entorno Java. iTextSharp está escrita en C# y aunque se genera a partir de un código fuente separado del código base de iText, se mantiene sincronizado en cada lanzamiento de éste.

Figura 25. iTextSharp permite crear y manipular archivos PDF desde código.

Este software está distribuido bajo la licencia AGPL [**21**] (Affero General Public License), que es una extensión de la licencia GPL para asegurar la cooperación en el caso de que el software

corra en servidores de red. Por tanto, la licencia permite su uso libremente para los propósitos de este proyecto. Además de esta licencia de uso libre, si se desea también se puede comprar una licencia propietaria de iText Group NV [**22**].

iText cumple con la mayoría de estándares PDF modernos, como son el ISO 32000-1 (PDF 1.7), ISO 19005 (PDF/A, para generar archivos PDF destinados a archivo a largo plazo) y el ISO 14289 (PDF/UA, para generar documentos con los requerimientos de accesibilidad para personas con discapacidad).

Aunque esta biblioteca permite incorporar muchas de las características más avanzadas y complejas que admiten los estándares del formato y otras extensiones, para el propósito de este proyecto se ha hecho uso sólo de algunas de las características más básicas.

En concreto se han usado funciones relativas a la incorporación de páginas, párrafos de texto e imágenes. Los textos se usan para identificar el componente que se está tratando y para numerar las páginas. La adición y ajuste de imágenes, para incorporar las imágenes generadas a partir de los ficheros Gerber, modificadas convenientemente para que muestren en cada paso el componente o los componentes a montar.

3.4. Lenguaje C#

3.4.1. Introducción

C# es el lenguaje de propósito general diseñado por Microsoft para su plataforma .NET [**23**]. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por diseñar el lenguaje Turbo Pascal y la herramienta RAD Delphi.

C# es un lenguaje que ha sido que ha sido diseñado específicamente para ser utilizado en la plataforma .NET, pero es posible escribir código para ella en muchos otros lenguajes. Sin embargo, programar para esta plataforma usando C# es más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos innecesarios en .NET heredados de otros lenguajes. Es por esto que habitualmente se dice que C# es el lenguaje nativo de la plataforma .NET.

El lenguaje C# tiene un grado de sencillez y nivel de productividad similares a los de Visual Basic. La sintaxis y estructura del lenguaje es muy parecida a la ce C++ y Java, puesto que, al diseñarlo, Microsoft pretendía facilitar la migración de códigos escritos en estos lenguajes y facilitar la adaptación de los desarrolladores que los usaban.

La idea original de Microsoft era usar Java como lenguaje de la plataforma y, de hecho, tenía pensadas varias modificaciones para mejorarlo y adaptarlo a sus propósitos y hacerlo un lenguaje orientado al desarrollo de componentes. Sin embargo, problemas con Sun [**24**] (la empresa desarrolladora de Java) hicieron a Microsoft plantearse desarrollar un nuevo lenguaje.

Como resultado, C# combina las mejores características de lenguajes preexistentes como son C++, Java o Visual Basic. Sin embargo, C# es hoy en día un lenguaje bastante maduro y en el trayecto de su existencia ha tenido algunos ajustes y modificaciones, pues ya desde el principio Microsoft se propuso potenciarlo, escribiendo la mayor parte de la BCL (Base Class Library) con él. Esto ha hecho que, desde un principio, su compilador fuese el más depurado y optimizado de los incluidos en el .NET Framework SDK (Software Development Kit) [**25**].

3.4.2. Características de C#

El lenguaje C# tiene multitud de características que lo hacen especial y seguidamente se expondrán algunas de las principales:

- Sencillez. C# se ha deshecho de muchos elementos que tienen otros lenguajes y que no son necesarios o incluso pueden ser contraproducentes en .NET, como por ejemplo:
    - El código escrito en C# es **autocontenido**, lo que significa que no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL.
    - No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (**.**) acceder a miembros de espacios de nombres (**::**).
    - El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.
- Modernidad. El lenguaje C# ha incorporado elementos que a lo largo del tiempo han demostrado ser muy útiles para el desarrollo de aplicaciones y que en otros lenguajes de referencia como Java o C++ hay que simular. Un ejemplo de esto es la inclusión de tipos básicos como el *string* para representar cadenas, la distinción de un tipo *bool* específico para representar valores lógicos o la inclusión de un tipo decimal que permite realizar operaciones de alta precisión con reales de 128 bits (que son muy útiles en el mundo financiero). También se ha incluido una instrucción *foreach* que permite recorrer

colecciones con facilidad y que, en el caso del lenguaje C# es ampliable a tipos definidos por el usuario.

- Orientación a objetos. C# es un lenguaje orientado a objetos, como sucede con casi cualquier lenguaje de propósito general actual. Sin embargo, este enfoque orientado a objetos es más puro que otros casos como C++ en tanto que no se admiten ni funciones ni variables globales, sino que todo el código y datos han de definirse dentro de la definición de algún tipo de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

C# contempla todas las características propias del paradigma de programación orientada a objetos: **encapsulación**, **herencia** y **polimorfismo**.

En cuanto a la encapsulación, es importante señalar que además de los típicos modificadores ***public***, ***private*** y ***protected***, C# añade un cuarto modificador, ***internal***, que puede combinarse con ***protected*** e indica que sólo puede accederse al elemento desde su mismo ensamblado.

Respecto a la herencia C# sólo admite herencia simple de clases (al igual que Java) ya que la múltiple (como el caso de C++) provoca más problemas que soluciones. Además, en la mayoría de los casos en que pudiera necesitarse, su utilidad puede ser simulada con facilidad mediante el mecanismo de herencia múltiple de interfaces, que sí está soportada debido a que es una característica propia del CTS.

Por otro lado y al contrario de Java, en C# se ha optado por hacer que todos los métodos sean sellados por defecto y que los métodos redefinibles hayan de marcarse con el modificador ***virtual*** (como en C++), lo que permite evitar errores derivados de redefiniciones accidentales. Además, un efecto secundario de esto es que las llamadas a los métodos serán más eficientes por defecto al no tenerse que buscar en la tabla de funciones virtuales la implementación de los mismos a la que se ha de llamar. Otro efecto secundario es que permite que las llamadas a los métodos virtuales se puedan hacer más eficientemente al contribuir a que el tamaño de dicha tabla se reduzca.

- **Orientación a componentes.** C# incluye en su propia sintaxis elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. La sintaxis de C# permite definir cómodamente **propiedades** (similares a campos de acceso controlado), **eventos** (asociación controlada de funciones de respuesta a notificaciones) o **atributos** (información sobre un tipo o sus miembros).

- **Gestión automática de memoria.** Todos los lenguajes incluidos en .NET tienen a su disposición el recolector de basura del CLR (*Common Language Runtime*). Esto tiene la ventaja de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, como la destrucción de objetos mediante el recolector de basura es indeterminista y sólo se realiza cuando éste se activa –ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el fuente–, el lenguaje también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción ***using***.

- **Seguridad de tipos.** C# incluye mecanismos para asegurar que los accesos a tipos de datos se realicen siempre de forma correcta. Esto permite evitar que se produzcan errores difíciles de detectar debido al acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en el entorno .NET al estar gestionado por un recolector de basura. De entre estos mecanismos se destaca:

    - **Todas las conversiones de ben ser entre tipos compatibles**. Esto es, entre un tipo y otro tipo antecesor suyo, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo, siempre que un objeto del primero almacenase una referencia del segundo (***downcasting***). Lógicamente, esto último sólo puede comprobarlo el CLR en tiempo de ejecución y no el compilador, por lo que en realidad ambos colaboran para asegurar la corrección de las conversiones.

    - No está permitido usar **variables sin inicializar**. El compilador da a los campos un valor por defecto consistente en ponerlos a cero y controla mediante análisis del flujo de control de fuente que no se lea ninguna variable local sin que se le haya asignado previamente algún valor.

    - En los **accesos a los elementos de una tabla** se comprueba siempre que el índice se encuentre dentro del rango de la misma.

    - Por defecto se controlan los desbordamientos en las operaciones aritméticas con constantes desde el compilador. En las operaciones con variables no se controlan los desbordamientos de forma predeterminada para conseguir mayor rendimiento, pero esto se puede cambiar, haciendo que se informe en tiempo de ejecución mediante una excepción.

  - En un programa en C# no se pueden usar punteros, salvo que se marque la zona del código donde se quieren usar como "insegura". Sin embargo, C# incluye **delegados**, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos, pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.

  - Se pueden definir métodos que admitan un número indefinido de parámetros de un cierto tipo, y el lenguaje siempre comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados, al contrario de lenguajes como C/C++.

- **Instrucciones seguras.** En C# se han impuesto algunas restricciones al usar las instrucciones de control más comunes para evitar errores muy frecuentes. Por ejemplo, cualquier expresión condicional ha de ser una expresión lógica y no aritmética. Con esto se evita caer en un error en la condición (muy habitual) por confundir el operador de igualdad (**==**) con el de asignación (**=**). Otro ejemplo destacado es la instrucción ***switch***, en la que cada caso ha de terminar en un ***break*** o ***goto*** que indique cuál es la siguiente acción a realizar. Esto evita la ejecución accidental de casos y facilita su reordenación.

- **Sistema de tipos unificado.** Al contrario de C++, todos los tipos de datos que se definan en C# siempre derivarán, aunque sea de manera implícita, de una clase base común llamada ***System.Object***, por lo que dispondrán de todos los miembros definidos en ésta clase y por tanto serán "objetos".

A diferencia de Java, en C# esto también es aplicable a los tipos de datos básicos del lenguaje. Podría pensarse que el hecho de tener que tratar a los tipos básicos como objetos forzosamente tendría repercusiones negativas en el rendimiento, pero se ha incluido un mecanismo transparente de ***boxing*** y ***unboxing*** con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto puede aplicárseles optimizaciones específicas, por lo que el rendimiento es óptimo.

Además, como todos los tipos del lenguaje derivan de una clase común se facilita en gran medida el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

- **Extensibilidad de tipos básicos.** Mediante **estructuras**, C# permite definir tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos: que se puedan almacenar directamente en pila y se asignen por valor y no por referencia.

Al almacenarse en la pila se optimiza en velocidad su creación, destrucción y acceso, puesto que serán más rápidos que si se almacenasen en el *heap*.

Sin embargo, el hecho de pasarlos como parámetros por valor en la llamada a un método es una característica que podría tener efectos negativos al tener que hacerles una copia. Para evitar estos efectos al pasar estructuras como parámetros de métodos, se da la posibilidad de pasarlos como referencias a pila a través del modificador de parámetro ***ref***.

- **Extensibilidad de operadores.** Con el objeto de facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los tipos de datos básicos predefinidos en el lenguaje, C# permite redefinir, al igual que C++, el significado de la mayoría de los operadores cuando se apliquen a diferentes tipos de objetos. Esto se aplica también a los operadores de conversión, tanto para conversiones implícitas como explícitas.

Las redefiniciones de operadores se hacen de manera inteligente, de forma que, a partir de una única definición de los operadores **++** y **--,** el compilador puede deducir automáticamente como ejecutarlos de manera prefijas y postfija. Además, definiendo operadores simples (como **+**), el compilador deduce cómo aplicar su versión de asignación compuesta (+=) y para asegurar la consistencia, el compilador vigila que los operadores con opuesto siempre se redefinan por parejas (por ejemplo, si se redefine **==**, también hay que redefinir **!=**).

También es posible redefinir el significado del operador **[ ]** para los tipos de datos definidos por el usuario, a través del concepto de **indizador**. De esta forma se consigue que se pueda acceder a los mismos como si fuesen tablas. Esta característica es de mucha utilidad para trabajar con tipos que actúen como colecciones de objetos.

- **Extensibilidad de modificadores.** En el módulo resultante de la compilación, el compilador normalmente incluye metainformación relativa al propio módulo. Además, mediante el concepto de atributos, C# ofrece la posibilidad de añadir a estos metadatos otra información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la biblioteca de reflexión de .NET. Esto, que es una característica propia de la plataforma .NET y no sólo de C#, puede usarse como un mecanismo para definir nuevos modificadores.

- **Versionable.** C# incluye una **política de versionado** que permite la creación de nuevas versiones de tipos sin que la introducción de nuevos miembros provoquen errores

difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

Si una clase introduce un nuevo método cuyas redefiniciones deban seguir la regla de llamar a la versión de su padre en algún punto de su código, difícilmente seguirían esta regla miembros de su misma signatura definidos en clases hijas anteriormente a la definición del mismo en la clase padre; o si introduce un nuevo campo con el mismo nombre que algún método de una clase hija, la clase hija dejará de funcionar. Para evitar que esto ocurra, en C# se toman dos medidas:

- Se obliga a que toda redefinición deba incluir el modificador ***override***, con lo que la versión de la clase hija nunca sería considerada como una redefinición de la versión de miembro en la clase padre ya que no incluiría ***override***. Para evitar que por error un programador incluya este modificador, sólo se permite incluirlo en miembros que tengan la misma signatura que miembros marcados como redefinibles mediante el modificador ***virtual***. Así además se evita el error, tan frecuente en Java, de creer haber redefinido un miembro, pues si el miembro con ***override*** no existe en la clase padre se producirá un error de compilación.
- Si no se considera redefinición, entonces se considera que lo que se desea es ocultar el método de la clase padre, de modo que para la clase hija sea como si aquel nunca hubiese existido. El compilador avisará de esta decisión a través de un mensaje de aviso que puede suprimirse incluyendo el modificador ***new*** en la definición del miembro en la clase hija para así indicarle explícitamente la intención de ocultación.

- **Eficiente.** En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y además no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones y lograr manipular objetos a través de punteros. Para poder hacerlo basta marcar regiones de código como inseguras (mediante el modificador ***unsafe***) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones en las que se necesite una eficiencia y velocidad procesamiento superiores.

- **Compatible.** Para facilitar la migración de programadores a C#, el lenguaje no sólo mantiene una sintaxis muy similar a C, C++ o Java (lo que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes) sino que el CLR

también permite, a través de los llamados ***Platform Invocation Services*** (***PInvoke***), la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLL (*Dynamic-Link Library*) de la API Win32. Además, la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, puesto que éstas muchas veces esperan recibir o devuelven punteros.

También es posible acceder a objetos COM desde código escrito en C#. Para permitirlo, el *.NET Framework SDK* incluye las herramientas ***tlbimp*** y ***regasm*****,** mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM.

Además, también se da la posibilidad de usar controles ActiveX desde código .NET, para lo que se utiliza la utilidad ***aximp***. Para usar objetos .NET desde ActiveX se usa también la ya mencionada ***regasm***.

Por último, decir que el CLS (Common Language Specification o Especificación del Lenguaje Común) es un conjunto de reglas que han de seguir las definiciones de tipos que se hagan usando un determinado lenguaje gestionado si se desea que sean accesibles desde cualquier otro lenguaje gestionado. Obviamente, sólo es necesario seguir estas reglas en las definiciones de tipos y miembros que sean accesibles externamente, y no la en las de los privados. Además, si no importa la interoperabilidad entre lenguajes tampoco es necesario seguirlas.

Para permitir futuros usos o ampliaciones al software desarrollado, imponiendo el menor número de limitaciones posible, se ha decidido usar los tipos permitidos en el CLS. Así por ejemplo, aunque muchos de los datos admitirían perfectamente ser almacenados en enteros sin signo, como éstos no están incluidos en el CLS se guardan en campos tipo *int*.

3.5. Flujo de diseño

Una vez establecida la línea de trabajo a seguir, se realiza un planteamiento de la estrategia de trabajo, en la que se van escalonando las actividades a realizar en una implementación dónde la complejidad del trabajo aumenta en cada paso. Para ello se ha desarrollado un método de trabajo representado en el diagrama de flujo de la figura 26.

El trabajo se ha dividido en tres grandes etapas que se corresponden con las tres grandes áreas de operación que tendrá el programa. En la primera etapa, se han hecho los estudios previos necesarios para la realización del proyecto y se han aplicado especialmente a la tarea de

conseguir un programa funcional que permitiera trabajar con las máquinas pick & place automáticas más antiguas de las que dispone el laboratorio.

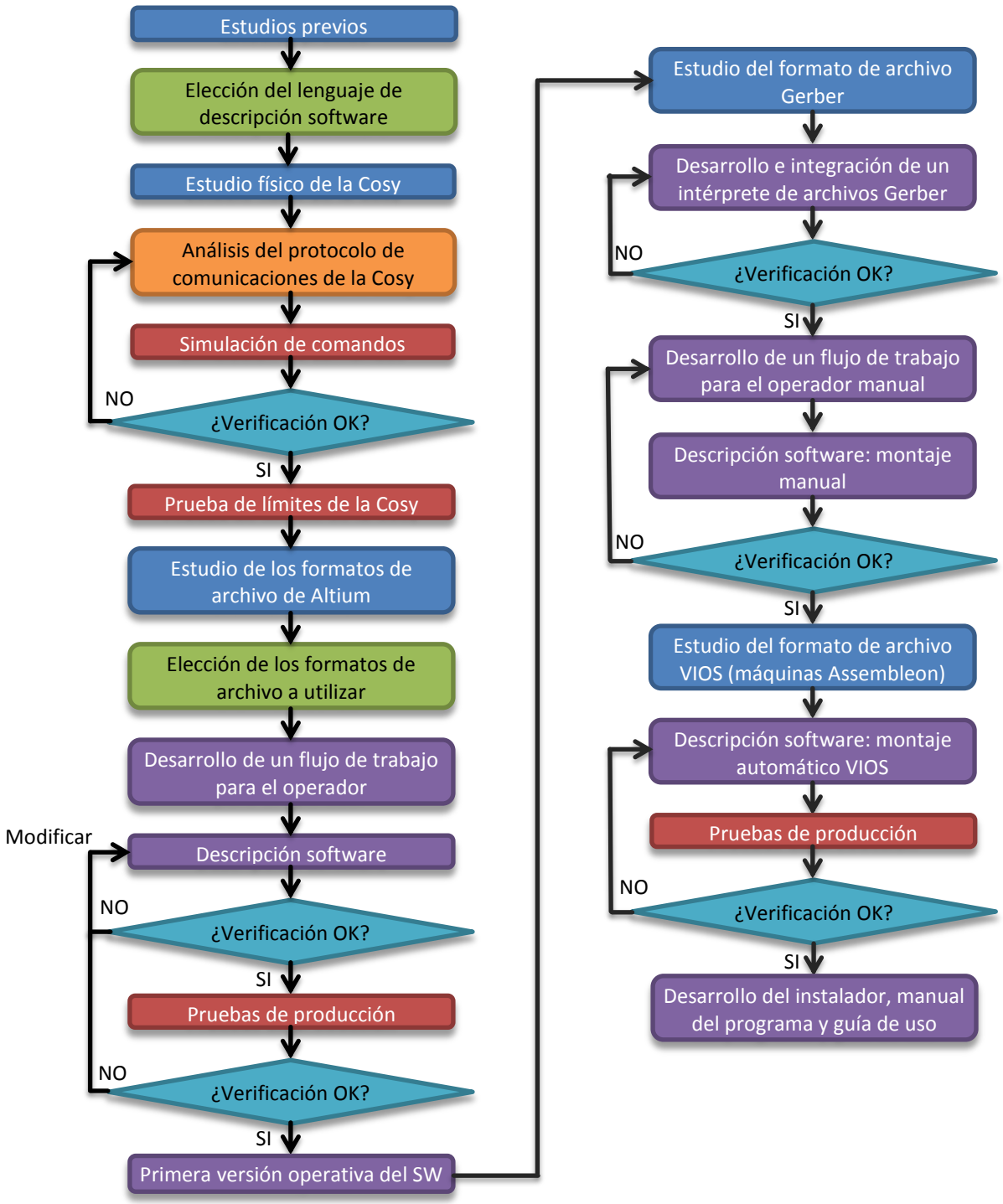


Figura 26. Flujo de diseño seguido.

En la segunda etapa se ha desarrollado aún más el aspecto visual del programa, introduciendo la capacidad de ver el diseño de la placa junto a los datos de colocación adquiridos.

Este aspecto es fundamental para el objetivo principal de esta etapa, que era la adición al programa de la capacidad de asistir en el montaje manual de componentes SMD.

En la última etapa se ha añadido una nueva funcionalidad al programa que permite su utilización para asistir al montaje automático con las máquinas automáticas más modernas de las que dispone actualmente el laboratorio (máquinas Assembleon y Philips).

3.5.1. Elección del lenguaje de descripción software y el entorno de desarrollo

Para el desarrollo de la aplicación, como se comentó anteriormente, se ha elegido el lenguaje C# puesto que se trata de un lenguaje de descripción moderno y funcional adaptado a las tecnologías actualmente en uso en el mundo de la programación. Además, se contó con la experiencia previa del director de este proyecto que había observado algunas deficiencias en otros lenguajes, en módulos que posiblemente serían necesarios en el proyecto actual.

Como entorno de desarrollo se ha elegido Microsoft Visual Studio 2010, puesto que era la versión más moderna del entorno a la que se podía acceder en el momento de tomar la decisión. No obstante, y mientras se tramitaba el acceso con licencia a dicha aplicación, en las primeras etapas se utilizó el paquete de software Microsoft Visual C# 2010 Express, una versión reducida del entorno que se pudo obtener gratuitamente desde la página web de Microsoft [**26**].

Este entorno de programación destaca por la rápida curva de aprendizaje de sus funciones principales, la gran cantidad de herramientas y ayudas a la programación de que dispone y la facilidad para crear aplicaciones de ventanas, que es el principal objetivo de este proyecto.

3.5.2. Estudio físico de la Cosy

Con el objetivo de conseguir una rápida familiarización con la máquina pick & place se inició un estudio pormenorizado del funcionamiento de ésta. Para realizar este estudio se desmontó parcialmente la máquina y se identificó los módulos de que consta y la forma en que operan.

Este estudio permitió conocer de primera mano la tecnología con la que fue construida la máquina, aportando un conocimiento mayor de las posibilidades de esta máquina.

3.5.3. Análisis del protocolo de comunicaciones de la Cosy y simulación de comandos

El análisis del protocolo de comunicaciones entre la Cosy y el PC de control es un paso imprescindible para la realización de este proyecto. Esto es así porque, como se verá más

adelante, casi todo el funcionamiento de la máquina está guiado paso por paso desde el PC de control, por medio de un puerto serie RS-232.

Este análisis finalmente se ha llevado a cabo en varios pasos, puesto que con cada porción del protocolo desvelada, y una vez puesta a prueba, planteaba nuevas incógnitas a revelar, resultando una tarea apasionante.

Las secciones del protocolo descubiertas iban abriendo el camino para probar nuevos comandos que se enviaban a la máquina desde varias aplicaciones de prueba desarrolladas expresamente para esta tarea, simulando parcialmente el montaje de una placa.

Posteriormente, conforme se fue dando forma al programa principal, objetivo de este proyecto, se añadieron muchas de las aplicaciones de prueba a un menú especial de test durante el resto del desarrollo.

3.5.4. Prueba de los límites de la máquina

Para conocer mejor la máquina se hicieron algunas pruebas que tratasen de llevarla un poco más allá de los límites de movimiento de ésta. En todas las pruebas menos una se observó que la respuesta de la máquina era segura, ignorando cualquier intento de avance más allá del límite.

El único elemento en el que no se observó este comportamiento fue en la bandeja de alimentadores, en la que, al enviarle el comando para que pasarse a colocarse en un supuesto cuarto banco de alimentadores (el modelo con el que trabajamos tiene instalada una bandeja para tres bancos de alimentadores solamente) intentó ir más allá del límite.

Esto puede ser debido a la existencia de bandejas alimentadoras para más de tres bancos en este modelo de máquina pick & place, como se comenta más adelante en el estudio de la máquina y a que el firmware de la misma no comprueba el final de carrera.

Esta información resultó muy útil para tener en cuenta el acotar adecuadamente los límites de los parámetros de operación en el software desarrollado.

3.5.5. Estudio de los formatos de archivo de salida de Altium y elección del formato a utilizar

Para poder desarrollar una aplicación que “entienda” el formato de los archivos de salida de Altium fue imprescindible realizar un estudio de la estructura y las características de estos.

Previamente se conocía de estos archivos que podían ser grabados en formato de texto, por lo que su estudio se realizó abriendo algunas muestras con un editor de textos y comprobando que en realidad se trataba de un formato de base de datos en texto plano con un registro por línea y los campos organizados en columnas.

Estas columnas de datos se separan por una cantidad variable de espacios en función del ancho de los datos y además contenían varias columnas de datos irrelevantes para las funciones de una máquina pick & place, por lo que se desarrolló un método para la extracción de los datos significativos de cada componente.

Posteriormente se observó la conveniencia de usar el tipo de archivos CSV, también disponible, por lo que se modificó el código fuente para adecuarlo al formato elegido finalmente.

También se obtuvo información relevante acerca del formato de estos archivos desde la página web de la empresa desarrolladora de la aplicación Altium [**27**].

3.5.6. Desarrollo del flujo de trabajo del operador

Para poner en orden el trabajo a realizar y establecer los pasos lógicos a desarrollar, se ha pensado que sería adecuado crear un flujo de trabajo para el operador de la máquina a la hora de preparar una placa para su producción.

De este flujo de trabajo se extrajeron varios métodos de interacción con el software, en función del trabajo a realizar, ayudando a definir las opciones de las que debía constar el programa.

3.5.7. Desarrollo del software

El desarrollo de la aplicación ha sido la tarea que más tiempo ha consumido, debido inicialmente a la inexperiencia con el lenguaje y el entorno de desarrollo, además de los distintos ciclos de ensayo y error realizados con las máquinas, las simulaciones y las comprobaciones de diversas condiciones de error para tratar de evitarlas en la medida de lo posible.

Para poder realizar pruebas sin estar constantemente conectado a la máquina se desarrolló un modo de pruebas en el programa que permitía hacer producciones “virtuales”, obteniendo un listado de los comandos enviados a la máquina que podían ser chequeadas manualmente para comprobar que eran similares a los del software original. Esto permitió verificar la operación correcta con las partes del protocolo que se iban descubriendo.

Más adelante, estos listados se refinaron, incluyendo marcas temporales y comentarios sobre la operación en curso, añadiéndose su grabación en disco en una carpeta de informes de

producción. De esta forma, terminaron convirtiéndose en los informes de producción de la aplicación final.

Además, el modo de pruebas permitía un desarrollo más ágil, al evitar los ciclos de reinicio de la máquina cuando se produce un bloqueo debido a comandos incorrectos. Por otro lado, este modo de pruebas virtual también supuso un gasto menor en componentes SMD, electricidad y desplazamientos para hacer las pruebas.

Para el desarrollo de la aplicación se ha utilizado el lenguaje C# como se comentó anteriormente, usando el entorno de desarrollo Microsoft Visual Studio 2010 durante casi todo el desarrollo. Al principio se comenzó usando el entorno Microsoft Visual C# 2010 Express, una versión reducida del anterior y también se utilizó Processing para hacer algunas comprobaciones y pruebas antes de tener los conocimientos suficientes para hacerlo con C#.

3.5.8. Pruebas de producción

Para comprobar la correcta operación del programa se ha desarrollado un pequeño montaje de pruebas en Altium. Este montaje, aunque bastante simple, nos ha permitido ir haciendo pruebas para verificar la operación correcta en cada uno de los pasos para la fabricación usando el flujo de trabajo propuesto.

3.5.9. Primera versión operativa del software

En este punto se ha creado un entregable consistente en una primera versión operativa del software. Esta primera versión es capaz de realizar todas las funciones de producción con las máquinas Cosy disponibles en el SFP.

3.5.10. Estudio del formato de archivo Gerber

Para desarrollar las funciones de asistencia en el montaje manual es fundamental poder indicar al operador alguna pista visual acerca de dónde se está trabajando en la placa. Para crear esta ayuda visual se ha decidido usar los archivos Gerber que se generan desde el programa de diseño para mostrar las pistas y los componentes de la PCB.

Aunque los archivos Gerber son archivos de texto, contienen información vectorial que describe los gráficos que se deben generar. Esta descripción se apoya en el uso de primitivas definidas en el estándar y en macros que puede crear el sistema de diseño.

Con el estudio del estándar que regula la creación de estos archivos se ha determinado qué funciones del estándar es necesario implementar para obtener una imagen suficientemente

útil para el propósito de guiar al operador, pero sin aumentar excesivamente la complejidad del código.

3.5.11. Desarrollo e integración de un intérprete de archivos Gerber

Basándonos en la información obtenida en el punto anterior y en las pruebas de desarrollo que se iban realizando, se ha creado una clase que es capaz de leer un archivo Gerber y generar un dibujo de la imagen que representa suficientemente fiel para que sirva de guía al operador.

Esta clase se ha integrado con el resto del código, utilizándola tanto en el código ya creado, para mejorar la representación gráfica previa, como en el código que se ha creado a continuación. Como se ha creado una clase para contener la información relativa a cada archivo Gerber, se ha implementado muy fácilmente la posibilidad de dibujar varias capas en la misma imagen, cada una con su propio color. Esto ha permitido crear dibujos donde es fácil identificar las distintas partes que lo forman.

3.5.12. Desarrollo de un flujo de trabajo para el operador manual

El montaje manual de los componentes de la PCB incluye algunas particularidades que lo diferencian del montaje automatizado desarrollado anteriormente. Estas particularidades se han tenido en cuenta en esta etapa, definiendo qué flujo de trabajo podría seguir un operador para el montaje manual de forma que se minimice la probabilidad de cometer errores.

Para facilitar este flujo de trabajo se ha visto la necesidad de crear algunas opciones adicionales, tanto en nuevos formularios como en algunos de los ya existentes, como por ejemplo la posibilidad de reordenar automáticamente los componentes según diferentes criterios para poder montarlos por grupos o la posibilidad de imprimir la información de montaje por grupos o individualmente, en lugar de mostrarla en pantalla.

3.5.13. Descripción software: montaje manual

Una vez establecidas algunas ideas básicas sobre las necesidades del montaje manual, se pasó a crear los formularios relacionados con éste y su código correspondiente. Tras implementar las ideas desarrolladas anteriormente, se realizaban pruebas que nos ayudaban a perfilar el flujo de trabajo para el montaje manual, realizando cambios en dicho flujo de trabajo e iterando sobre una nueva descripción software hasta que el resultado fuese satisfactorio.

Para el montaje manual se desarrollaron dos métodos de ayuda final al montaje:

- El primero va guiando el montaje interactivamente desde la pantalla del ordenador, mediante un formulario del programa.
- El segundo permite generar un archivo PDF con la información de montaje seleccionada por el operador.

Esta última opción permite trasladar la información de ayuda al montaje hasta el puesto de trabajo sin necesidad de disponer de un ordenador con la aplicación creada en este proyecto instalada. También permite transmitir dicha información fácilmente a otras personas tanto por medios electrónicos como impresa en papel.

3.5.14. Estudio del formato de archivos VIOS

Con el fin de poder utilizar el programa también para preparar el trabajo de montaje que se realiza en el SFP con las nuevas máquinas de montaje automatizado, se ha estudiado el flujo de trabajo que se realiza en el laboratorio a la hora de trabajar con ellas.

Este flujo de trabajo se encuentra descrito en un documento interno que describe una cierta cantidad de pasos que hay que realizar para formatear adecuadamente los archivos BOM, de forma que se puedan procesar por el programa Cad2cad suministrado por el fabricante. Este programa genera un archivo de tipo VIOSText que es el que finalmente se introduce en las máquinas para realizar la producción.

En esta etapa se ha estudiado la posibilidad de generar directamente dicho archivo a partir de los datos que se capturan para el resto de posibilidades de montaje. Para ello se han estudiado algunos ficheros que describen circuitos que se han montado en el pasado en el laboratorio, así como algunos otros de ejemplo.

3.5.15. Descripción software: montaje automático VIOS

Con la información obtenida del análisis anterior, se ha creado una función adicional para el programa que permite crear los archivos VIOS que pueden alimentar a las máquinas más modernas de que dispone el laboratorio. De esta forma se ha logrado cubrir todas las posibilidades de producción del SFP desde el programa realizado en este proyecto.

3.5.16. Desarrollo del instalador, manual del programa y ayuda

Finalmente, se ha creado un programa instalador que se encarga de copiar los archivos de que consta la aplicación en el disco del ordenador de destino. Además, dicho instalador comprueba si el ordenador de destino cumple los requisitos necesarios para la ejecución (básicamente, tener instalado el Microsoft .NET Framework v4.0) y si no lo tiene, lo instala.

El instalador se encarga también de añadir la aplicación al menú inicio de Windows, crear un icono en el escritorio para la aplicación y hacer las asociaciones de archivos en Windows entre la aplicación y sus archivos de datos. Para ello se han creado iconos personalizados, tanto para la aplicación como para sus archivos de datos. Además, para complementar el aspecto visual de la aplicación se han creado, también en esta etapa, iconos y gráficos ornamentales específicos para los formularios de la aplicación.

Adicionalmente, se ha creado una versión "*portable*" del software, por si el usuario no desea instalar la aplicación en el equipo sino copiarlo en una memoria USB u otro soporte.

Para facilitar el aprendizaje de uso del programa desarrollado, se ha creado un extenso manual del usuario que explica todas las opciones del menú del programa, todas las ventanas y muchos de los cuadros de diálogo de la aplicación. Este manual se centra en el uso del programa, desde los archivos de Altium de partida, pasando por el editor de alimentadores hasta la puesta en marcha del proceso de montaje en la máquina pick & place. Además se explica también cómo usar los editores de placas y componentes para el caso de que se necesite crear un diseño desde cero, es decir, sin partir de un archivo de pick & place generado en Altium.

Se incluye una guía rápida que explica brevemente qué pasos seguir para completar el montaje de una placa, partiendo de un archivo BOM de Altium o sin disponer de este archivo, partiendo de cero.

Tanto el manual del usuario como la guía rápida se han incluido en los anexos.

CAPÍTULO 4. ESTUDIO DE LAS MÁQUINAS

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

4.1. Introducción

En este capítulo se describen las máquinas pick & place de que dispone el laboratorio, que fueron estudiadas para poder realizar este proyecto.

La máquina pick & place que fue objeto de un estudio especialmente detallado durante este proyecto es el modelo Cosy SMD Flexpick. Esta máquina fue fabricada en Suiza por la

empresa Kontakt-Systeme AG, Contact Systems of Switzerland, en el año 1998. Actualmente se dispone de dos máquinas de este tipo que se encuentran instaladas en el *Laboratorio de Fabricación de Prototipos y Sistemas Electrónicos* del IUMA, gracias a un acuerdo de colaboración con la empresa Inerza S. A., propietaria de las mismas.

Se trata de máquinas pick & place de un solo cabezal y alineamiento mecánico, de baja capacidad de producción (unos 1.000 componentes/hora). Estas máquinas tienen algunas limitaciones a la hora de trabajar con cierto tipo de tecnologías actuales (como componentes BGA o de paso fino), pero son aptas para trabajar con componentes hasta métrica 0603.

La mejor forma de entender cómo trabaja esta máquina es verla en funcionamiento y el laboratorio ha dispuesto un canal en internet, por medio del que se pone a disposición del público en general un vídeo en el que se muestra a la máquina Cosy en funcionamiento [**28**]. En el vídeo se muestra la producción de algunos módulos pertenecientes al Gran Telescopio de Canarias, Grantecan.

En este capítulo se hará una descripción del interior y exterior de la máquina, así como del ordenador y el software de control original de la misma. Mientras se describe la máquina, se comentarán algunos de los aspectos que se han tenido en cuenta para el desarrollo de la aplicación objetivo de este proyecto.

Para averiguar diversos detalles del funcionamiento de la máquina se desarrollaron pequeñas aplicaciones con objetivos de prueba específicos sobre algunas de sus capacidades. También se desarrolló una pequeña aplicación de pruebas generales que permitía construir y enviar comandos “en bruto” para simular situaciones de producción y determinar mejor el comportamiento de la máquina.

La mayor parte de estas pequeñas aplicaciones fueron añadidas posteriormente a un menú de pruebas en nuestra aplicación que nos permitía avanzar en el conocimiento de la máquina y hacer pruebas adicionales mientras se iba completando su desarrollo. Estas opciones no se han incluido en la versión final del software. Esto se ha hecho así porque permiten un control directo de muchas funciones de la máquina que no tienen utilidad en el montaje normal, según los procedimientos desarrollados. Además, un motivo de peso para omitirlas ha sido que son una fuente de peligros, en caso de que se usen incorrectamente.

Además de este estudio detallado de la Cosy, también se han estudiado el resto de máquinas a las que va destinado el software que se ha diseñado en este proyecto. Estas son las máquinas de montaje manual (semiautomático) LPKF Protoplace y las máquinas de montaje

automático de la marcas Assembleon y Philips disponibles en el laboratorio: la primera modelo MG-5 y la segunda modelo CSM84 III. El estudio se centró en la primera, por ser la más moderna y de mayor uso, pero ambas admiten sistemas similares de preparación del montaje. El estudio de estas otras máquinas se limitó a su principio de operación y su sistema de programación por ficheros, sin realizar un desmontaje y análisis físico de las mismas, puesto que no era necesario.

En el canal del laboratorio también se pueden ver vídeos [**29**] [**30**] de estas máquinas en acción, ejecutando algunas pruebas y montaje de placas.

4.2. Descripción externa de la Cosy

4.2.1. Introducción

En este apartado se describirá el exterior de la máquina, haciendo énfasis en las partes directamente relacionadas con el montaje de las placas y que será necesario controlar desde nuestra aplicación. También se ha incluido en este apartado algunas partes que, desde cierto punto de vista, podrían considerarse de la estructura interior de la máquina, pero que se ha preferido reunir con la parte externa, a la que dan funcionalidad y a la que están íntimamente ligadas, como son los mecanismos del brazo móvil o la estación de recentrado.

4.2.2. Aspecto exterior de la máquina

Como se puede ver en la figura 27, el aspecto difiere bastante de las máquinas que se usan hoy en día en las cadenas de montaje automatizadas, puesto que el panel de trabajo, donde se realizan las operaciones de montaje de componentes sobre la placa está en el exterior de la máquina.

En las máquinas pick & place preparadas para integrarse en una cadena de montaje totalmente automatizada (Como la MG5 o la CSM84 III), un carril transporta la PCB hasta la zona de montaje, que suele estar en el interior de la máquina, y al terminar de montarla la transporta hacia la salida, posiblemente enviándola directamente a la máquina siguiente del proceso.

En el caso de la Cosy, la colocación y fijación de la PCB sobre el panel la realiza manualmente el operador de la máquina, al igual que en las máquinas de montaje manual. Una ventaja de esto es que se pueden ubicar en el panel varias placas individuales simultáneamente, para ser montadas como un único trabajo. Estas placas pueden ser iguales o diferentes y no hay necesidad de panelizarlas, aunque el software que acompaña originalmente a la máquina sólo admite la posibilidad de montar placas iguales, como se verá más adelante en la sección dedicada a éste.

Una máquina con carril de transporte para la PCB, normalmente sólo puede montar una placa cada vez y, si se quieren montar varias simultáneamente, es necesario panelizarlas antes de fabricar la PCB.

Figura 27. Exterior de la máquina Cosy SMD Flexpick.

Esto abre algunas posibilidades de optimización del proceso con esta máquina, ya que si se está preparando un trabajo de montaje con una cierta placa o panel, se podría decidir sobre la marcha incluir otra placa (igual o diferente) en el panel, siempre que quede suficiente espacio para colocarla.

4.2.3. Botón de parada de emergencia

Como se puede ver en la figura 27, en la parte derecha del frontal de la máquina hay un botón de parada de emergencia que se puede ver con más detalle en la figura 28. Si se pulsa, se corta la corriente y la máquina se para, para soltarlo y poderla poner en marcha otra vez, hay que girarlo en la dirección de las flechas.

Existe otro botón similar en la botonera principal de la máquina, aunque el otro incluye la llave de puesta en marcha del sistema, que se verá más adelante.

Figura 28. Botón de parada de emergencia.

4.2.4. Panel de trabajo

El panel de trabajo de la máquina es el área sobre la que se pueden situar las placas que van a ser montadas. Está situado en una posición central de la máquina vista desde arriba, como se puede ver en la figura 29, donde se le ha colocado una PCB de muestra.

A su izquierda se encuentra el área de herramientas auxiliares, donde la máquina puede intercambiar el útil de la boquilla o centrar los componentes de mayor tamaño. A la derecha están el área de desecho de componentes y una zona auxiliar de alimentadores. En la parte superior de la fotografía, que se corresponde con la parte trasera del panel está el brazo móvil que porta los cabezales. Por último, en la parte inferior de la fotografía, que se corresponde con la parte anterior al panel se encuentra la zona principal de alimentadores.

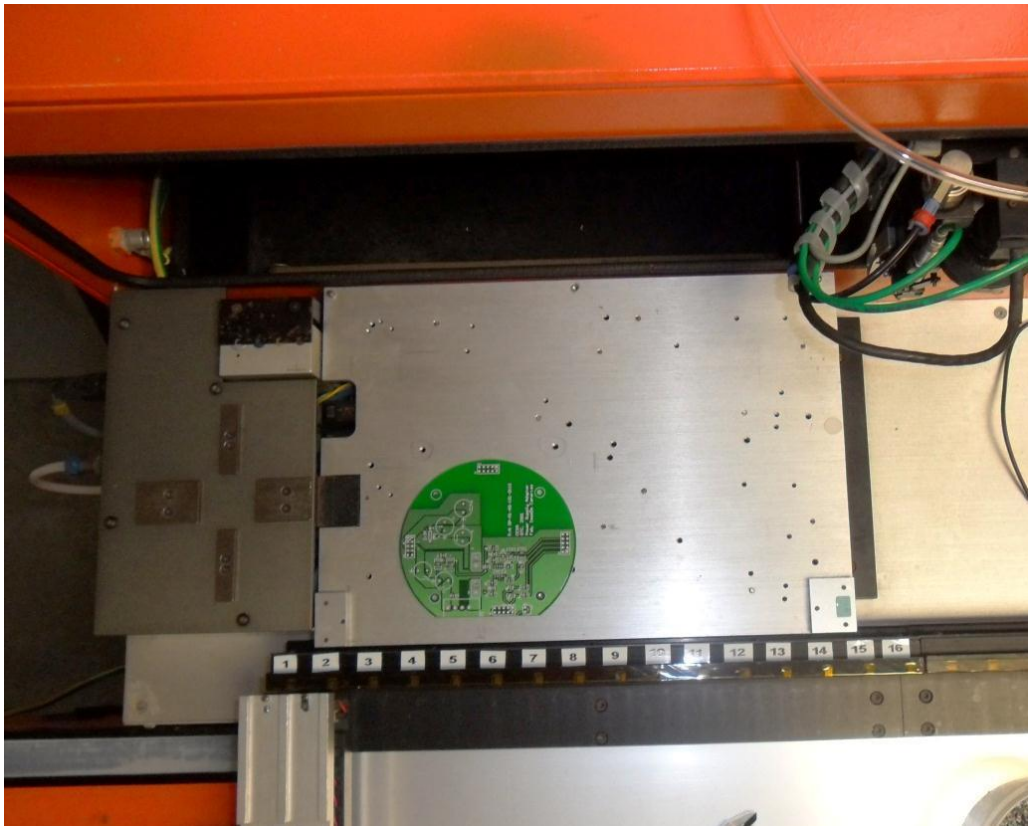


Figura 29. Panel de trabajo de la Cosy.

La placa de aluminio donde se fijarán las placas que van a ser montadas tiene unas dimensiones de 305 mm x 210 mm. Sin embargo, los primeros 10 mm no son accesibles por la cámara, por lo que el área útil, usando la cámara para posicionar los componentes, es de 305 mm x 200 mm. Como se puede ver, esta placa tiene multitud de agujeros de fijación de trabajos realizados anteriormente.

Un detalle muy importante a tener en cuenta es que la máquina considera que ambos ejes crecen en la dirección contraria a la que se suele tomar como dirección creciente del eje (por ejemplo al dibujar la gráfica de una función matemática). De esta forma, para la máquina, el origen de coordenadas está cerca del área de desecho (en el límite derecho de movimiento del brazo, al fondo). En la Figura 30 se han representado los ejes de coordenadas de la máquina en vista cenital y se ha marcado el lugar del origen de coordenadas y las coordenadas de las otras tres esquinas que limitan el área de movimiento del cabezal, en milímetros.

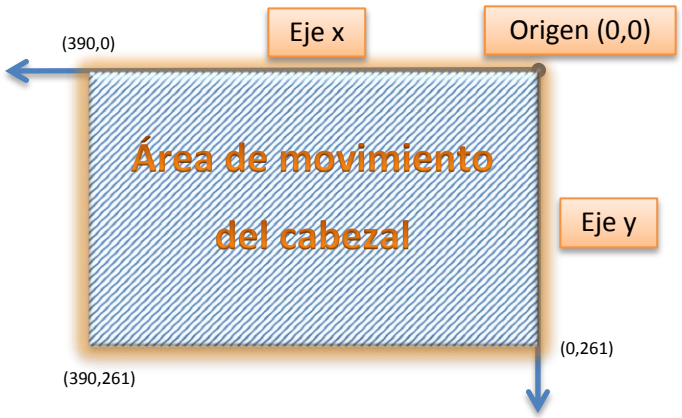


Figura 30. Ejes de movimiento del brazo y coordenadas límite del cabezal de la Cosy.

4.2.5. Herramientas auxiliares

En el área situada a la izquierda del panel de trabajo de la máquina se encuentra una zona de herramientas auxiliares que son necesarias normalmente al trabajar con los componentes de mayor tamaño. En la parte trasera se encuentra la zona de almacenamiento de útiles y, delante de esta, la estación de recentrado.

4.2.5.1. Útiles

La zona de útiles consta de unos huecos donde permanecen almacenados y asegurados los distintos útiles acoplables a la aguja de succión. Estos útiles son cuatro boquillas acoplables a la boquilla principal que permiten coger componentes de mayor tamaño de forma segura. Cuando el cabezal va a cambiar de útil se abre una pequeña compuerta que mantiene cada boquilla fija en su posición.

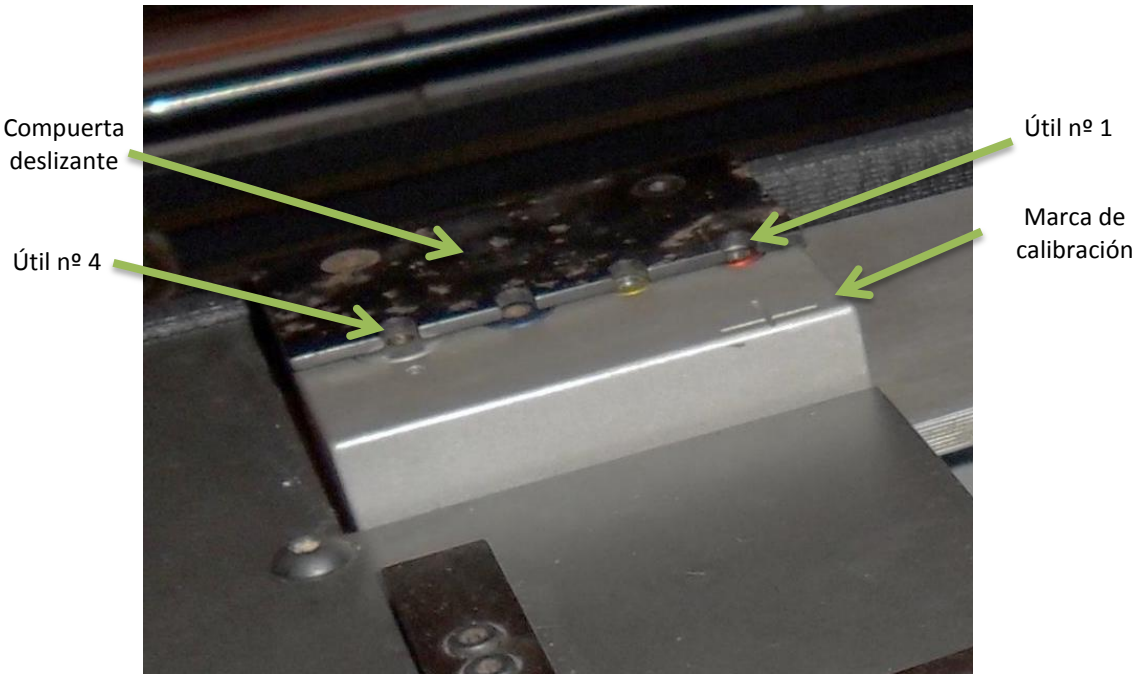


Figura 31. Útiles y marca de calibración.

Además, en el bloque que soporta los útiles se encuentra la marca de calibración óptica de la máquina. Esta marca de calibración es una cruz grabada en la superficie del bloque, cerca del borde. En la figura 31 se pueden apreciar todos estos elementos mencionados.

4.2.5.2. Estación de recentrado

La zona de recentrado, externamente consiste en cuatro piezas planas de metal que serán las guías de recentrado, dos de las cuales se pueden mover en el eje x y las otras dos en el eje y, como se puede ver en la figura 32. Estas piezas tienen un pequeño bisel en la parte inferior para no deformar el patillaje de los componentes en que éste se expande hacia fuera.

Al enviar a la máquina el comando de recentrar un componente, ésta actúa ejecutando una rutina de recentrado, que consiste en una serie de pasos. El primero es depositar el componente en el hueco central que hay entre las guías de recentrado.

Seguidamente activa las guías de recentrado del eje *x*, devolviéndolas a continuación a su posición de reposo para luego activar las del eje *y*, posteriormente devolviéndolas también a su posición de reposo. A continuación vuelve a activar las quías de la misma manera una segunda vez para terminar la operación de recentrado. Por último el cabezal de succión vuelve a coger el componente para llevarlo a la posición donde debe ser colocado.

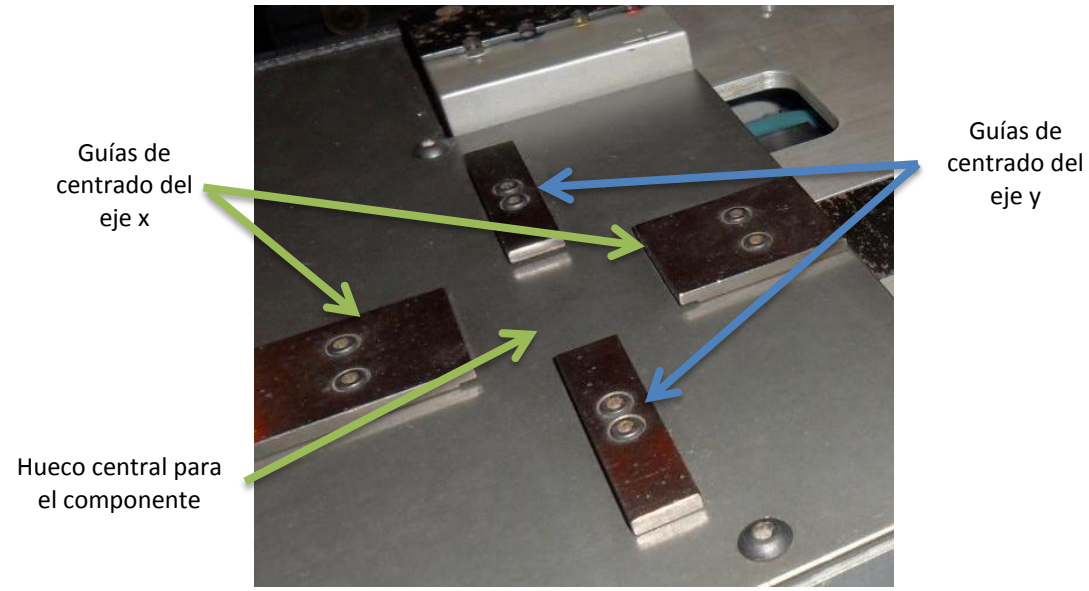


Figura 32. Estación de recentrado.

Bajo la superficie, la máquina alberga dos motores para mover las guías de recentrado. Cada motor mueve ambas guías de un mismo eje simultáneamente usando un mecanismo de correa. En la figura 33 se pueden ver estos componentes, tras retirar la tapa que los cubría.

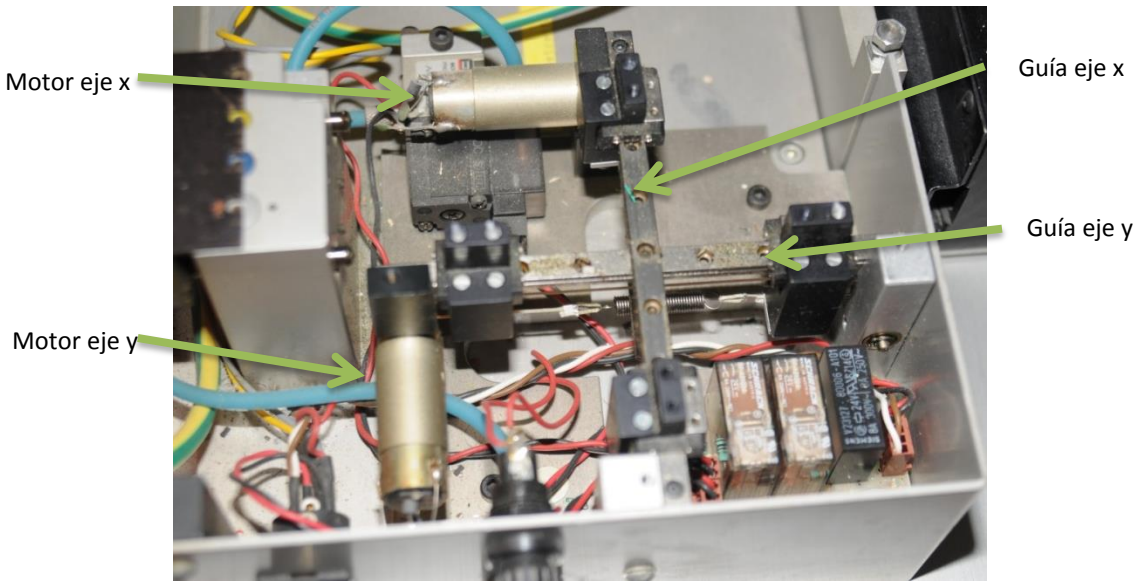


Figura 33. Mecanismo interior de la estación de recentrado.

4.2.6. Brazo móvil

El brazo móvil es la parte de la máquina que realiza una mayor cantidad de trabajo. Se trata de un componente complejo, que consta de varias partes y tiene adosados los cabezales en el extremo exterior. Tiene cuatro grados de libertad de movimiento, puesto que se puede mover linealmente en los ejes x e y para situarse sobre cualquier punto del panel de trabajo, herramientas auxiliares o alimentadores, pero también puede mover el cabezal de succión en el

eje z para subir y bajar al capturar o liberar los componentes y las herramientas auxiliares. Además, la boquilla de captura de los componentes puede girar (eje t) para posicionar los componentes con el ángulo adecuado en la placa que se está montando y también para enganchar o liberar las herramientas auxiliares.

En la figura 34 se muestra la porción del brazo que sobresale al exterior de la máquina con los cabezales. De izquierda a derecha en la imagen se encuentran: el cabezal de succión, que transportará los componentes, la videocámara y el cabezal del dispensador de adhesivo. El cabezal de succión incluye unas pinzas para el centrado de componentes y, un poco más atrás, un pequeño imán. Este imán se puede bajar y subir para activar los sensores de efecto Hall que disparan el avance de los alimentadores. Así, al coger un componente del alimentador, el imán está justo encima del sensor y, activándolo en ese momento, el alimentador sirve un nuevo componente en la posición de captura.

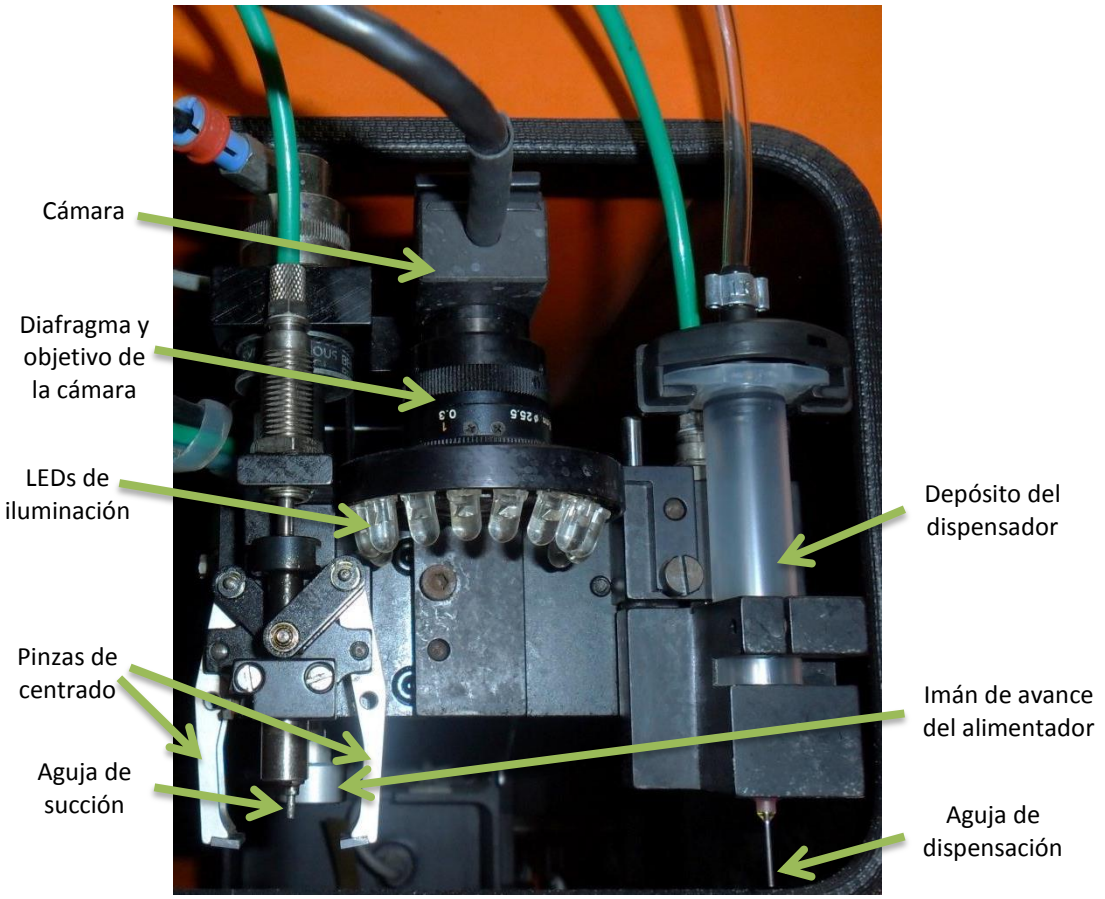


Figura 34. Cabezales en el extremo del brazo móvil.

La videocámara incluye un módulo óptico adosado, que contiene un diafragma y un objetivo, ambos ajustables, además de un conjunto de diodos LED alrededor del frontal del objetivo que se encargan de iluminar el área visible. Con el ajuste del diafragma podemos regular

la cantidad de luz que entra a la cámara para obtener un buen contraste en la imagen, mientras que con el ajuste del objetivo logramos enfocar la imagen para conseguir una buena definición.

Durante las pruebas se pudo comprobar que el sistema de soporte de la cámara tiene una inclinación muy leve que hace que, al girar el ajuste para cambiar la distancia de enfoque, varíe ligeramente las coordenadas del punto enfocado. La variación es de unas pocas décimas de milímetro, por lo que se ajustó y dejó fijo el enfoque en una posición óptima para enfocar los componentes de los alimentadores, las placas del panel y, en menor medida, el fondo de la marca de calibración, que son las zonas de mayor interés.

A la derecha de la cámara está el dispensador de adhesivo. Este tiene un depósito y una aguja de dispensación que se conectan a una unidad de control exterior (manguera transparente). La máquina controla la bajada del eje z del dispensador mediante aire comprimido y tiene un conector en la tapa posterior para gobernar a la unidad exterior del dispensador. En el interior de la máquina, el brazo está soportado por dos pares de raíles que permiten el movimiento en los ejes x e y, como se puede ver en la figura 35. Además se puede apreciar la correa que permite girar el cabezal (eje t) conectada a su motor correspondiente y, casi en el extremo del brazo, se puede ver parte del motor paso a paso que permite subir y bajar la aguja de succión (eje z).

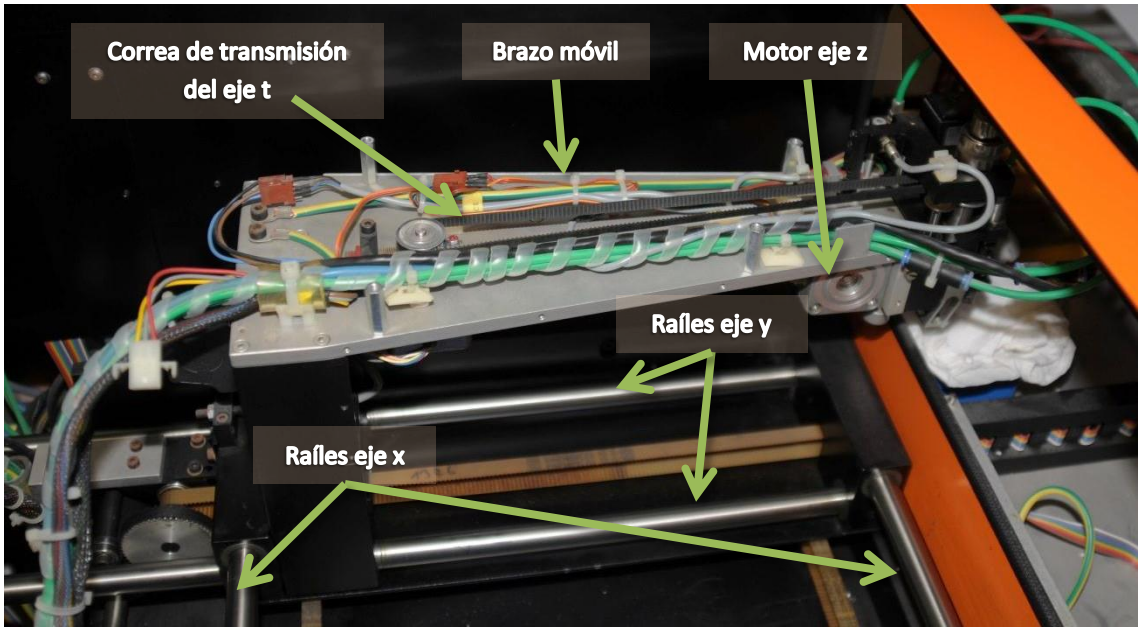


Figura 35. Mecanismos en el interior del brazo de cabezales.

También se pueden ver sobre el brazo las mangueras del circuito neumático que van desde el interior de la máquina hasta el conjunto de elementos del extremo del brazo. Junto con las mangueras, se encuentran agrupados los cables de los sensores y el de vídeo de la cámara.

Una de las mangueras proporciona presión negativa a la boquilla de succión, otra lleva presión a las pinzas de centrado, la tercera lleva presión al imán de avance del alimentador y la última corresponde al dispensador de adhesivo.

Bajo el brazo, junto a los raíles, se aprecian las correas de transmisión de los motores de los ejes x e y.

4.2.7. Área de desecho

El área de desecho es una zona que forma un pequeño hueco a la derecha del panel de trabajo de la máquina y detrás del módulo de alimentadores secundarios. Se puede observar esta zona en la figura 36, visto desde el lado derecho de la máquina.

En esta área se soltarán los componentes que estén en el cabezal cuando se produzcan ciertas situaciones de error, como por ejemplo un fallo en el test de vacío, ya que el componente podría estar mal cogido (por ejemplo suspendido por una arista o un vértice) y no podría colocarse correctamente.

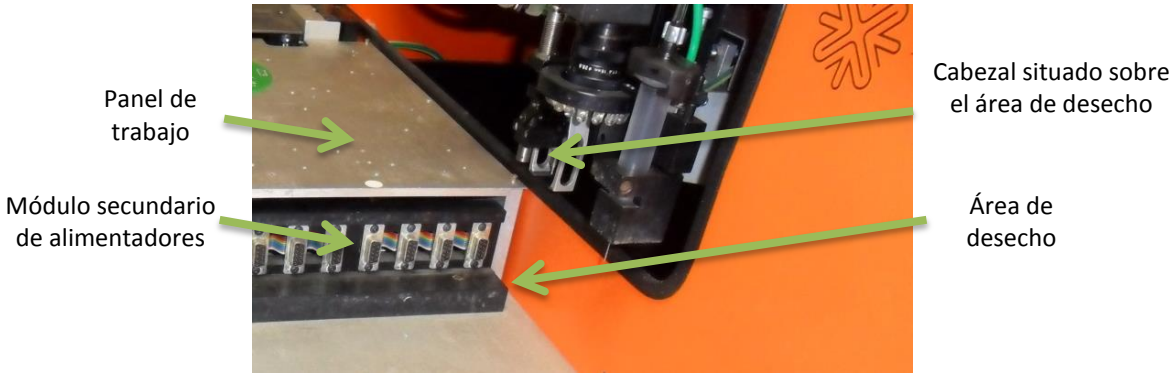


Figura 36. Área de desecho.

4.2.8. Alimentadores

Los alimentadores son las unidades de las que el cabezal obtiene los componentes que debe colocar sobre la placa que se va a montar. Cada alimentador contiene sólo un tipo de componente, salvo algunos casos especiales de alimentadores dobles, triples, etc. Para cada diseño es necesario que el operador de la máquina coloque cada alimentador con el tipo de componente correcto en el lugar que le corresponde. Si no lo hiciese así y lo colocase en otro lugar de los disponibles, deberá especificar en el software los datos de la nueva ubicación de dicho componente.

La máquina dispone de dos módulos donde ubicar los alimentadores, un módulo principal y otro secundario. En la figura 37 se puede ver la máquina con algunos alimentadores colocados en ambos módulos.

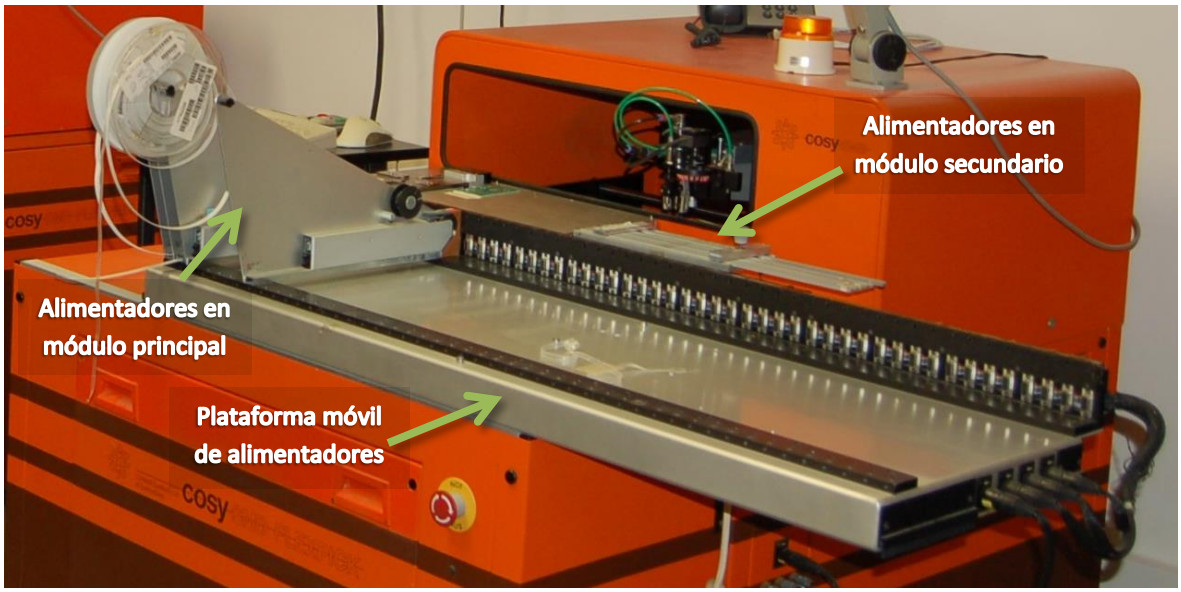


Figura 37. Alimentadores en el módulo principal y el módulo secundario.

4.2.8.1. *Módulo principal de alimentadores*

Delante del panel de trabajo se ubica el módulo principal de alimentadores. Este módulo consta de una base y sobre ella una plataforma móvil que contiene los conectores y las fijaciones de los distintos modelos de alimentadores que se pueden colocar en la máquina, como se puede apreciar en la figura 38.

La máquina permite instalar sobre la base plataformas de varios tamaños para poder añadir más alimentadores. Estas plataformas se pueden mover a izquierda y derecha para permitir el acceso a todos los alimentadores que se monten, puesto que muchos quedarán más allá del alcance del brazo móvil. El movimiento no se puede hacer gradualmente, sino que la máquina lo hace automáticamente por secciones, creando grupos (bancos) de alimentadores, mutuamente excluyentes. En las máquinas existentes en el laboratorio, las plataformas que están montadas actualmente permiten 3 bancos de alimentadores.

El banco con el que se va a trabajar se controla mediante software, enviando comandos a la máquina mediante el puerto serie para especificar cuál se desea usar. En la rutina de inicio, la máquina siempre deja listo para operar el primer banco de alimentadores.

Durante las pruebas realizadas, se comprobó que al enviar la orden de pasar a un cuarto banco la máquina intenta ir más allá del final de la plataforma instalada, por lo que se verificó que

no controla automáticamente los finales de carrera de la plataforma. Por tanto, se ha tenido en cuenta este detalle al diseñar el software para evitar que se produzca una rotura accidental debido a un error del operador.

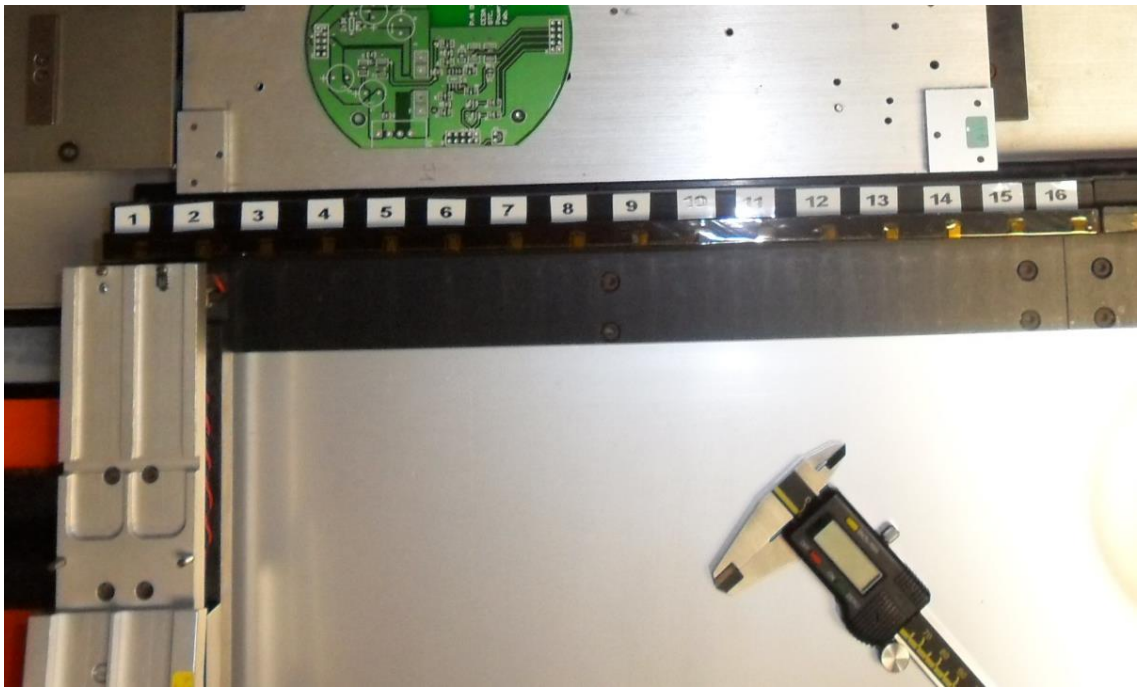


Figura 38. Alimentador doble colocado en el slot nº 1 del primer banco de alimentadores.

Cada banco de alimentadores en la plataforma consta de 16 slots que se pueden cargar con diversos tipos de alimentadores. Normalmente cada alimentador ocupará un slot, pero algunos tipos de alimentadores múltiples ocupan varios slots. En la figura 38 se puede ver el primer banco de alimentadores desde arriba con un alimentador doble colocado en el primer slot. Se puede comprobar que el segundo raíl del alimentador cubre gran parte del segundo slot, imposibilitando colocar otro alimentador en él. Para facilitar el trabajo con los alimentadores, se han rotulado los 16 slots del primer banco, que es el más usado.

En principio, el tipo de alimentador utilizado en cada slot no afecta al diseño del software de control, puesto que en la operativa normal, al preparar el programa de producción de una placa, se debe especificar las coordenadas en las que se coge cada componente, independientemente de en qué slot está instalado su alimentador. Si es importante, sin embargo, conocer el banco en el que se encuentra el alimentador en cuestión, por lo que éste es uno de los datos que se ha guardado en las estructuras de datos del software diseñado.

Existen diversos tipos de alimentadores que se pueden colocar en los slots de la máquina, adecuados a los diferentes tipos de encapsulado y el empaquetado en que son servidos por los

proveedores. Algunos de los alimentadores deben desempaquetar y hacer avanzar un componente cuando el brazo coge al que le precede, dejándolo listo para que el cabezal lo capture durante una operación posterior.

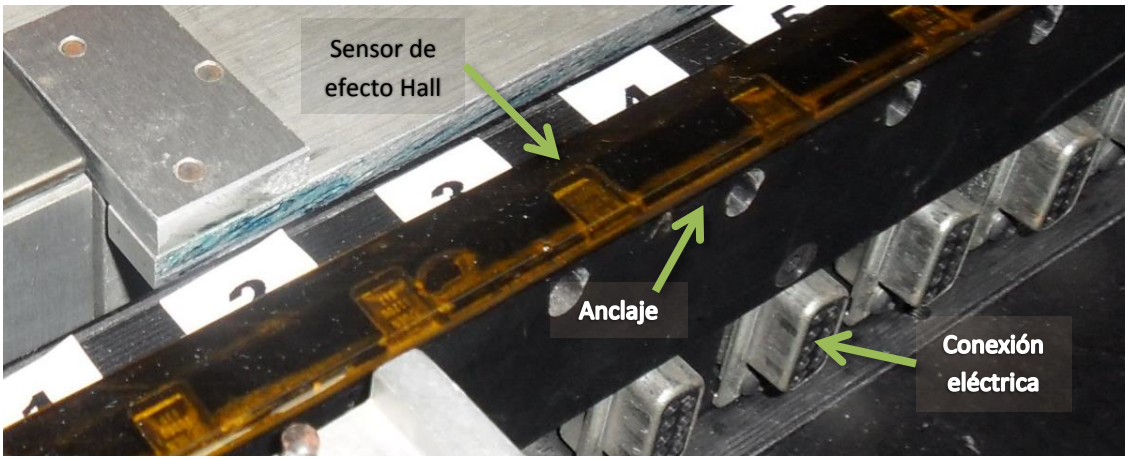


Figura 39. Detalle del sensor, anclaje frontal y conexión eléctrica de los alimentadores.

Para saber cuándo debe activar este mecanismo de avance del componente, la máquina posee un sensor de efecto Hall por cada alimentador. Este sensor está montado en el bastidor de los anclajes frontales de los alimentadores. Algo más abajo del anclaje, cada alimentador tiene la conexión eléctrica con el control de alimentadores, para la que se utiliza un conector DB9. En la figura 39 se puede observar una vista de detalle de estos elementos, los tres elementos rotulados pertenecen al slot número 4 del primer banco.

El sensor será excitado bajando el imán que posee el cabezal, pero sólo para aquellos alimentadores que lo requieran. Por tanto se ha reservado un campo en las estructuras de datos de los alimentadores para indicar si debe activar el avance del componente o no.

En la figura 40 se puede ver el mecanismo interno de uno de los alimentadores de este tipo que hay disponibles en el laboratorio. En la parte inferior de la imagen se halla un motor solidario a un tornillo sin fin que se encarga de mover el sistema de avance y pelado cuando es requerido.

El arrastre mecánico interno se realiza mediante un mecanismo de correa dentada que pasa por el engranaje del sistema motor, el sistema de pelado y el rodillo guía. De esta forma, todo el mecanismo avanza al mismo tiempo. La cinta ya vacía y la cubierta salen hacia atrás, desde donde colgarán para que puedan ser desechadas.

Las plataformas móviles de alimentadores instaladas actualmente en ambas máquinas son iguales y albergan 48 slots cada una.

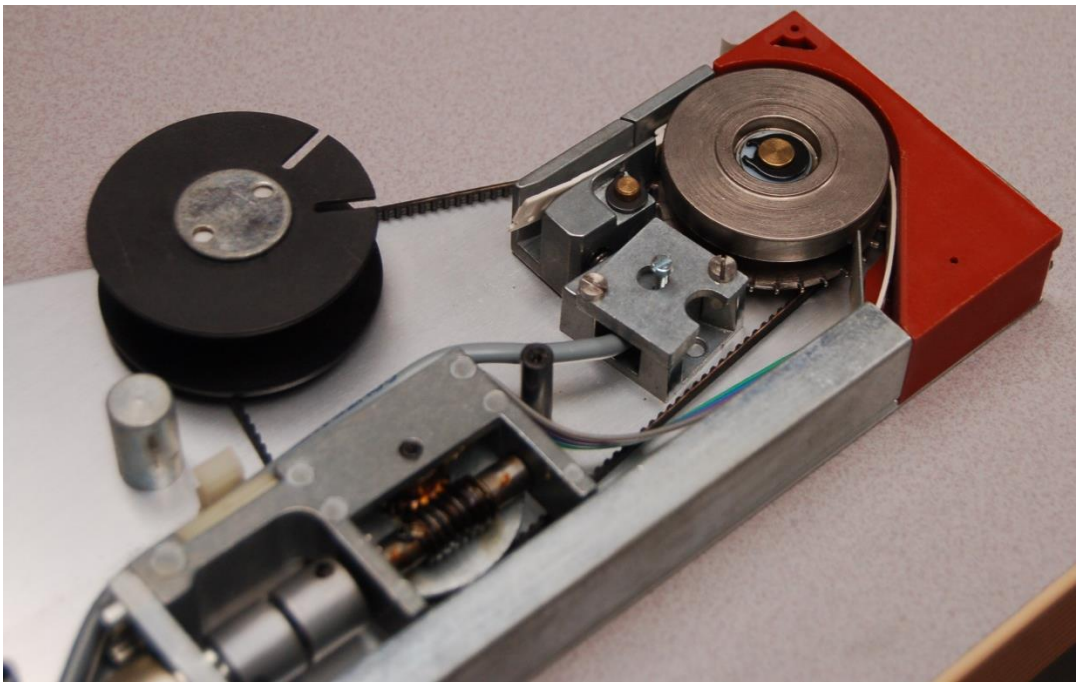


Figura 40. Detalle interior de un alimentador.

4.2.8.2. *Módulo secundario de alimentadores*

Al lado derecho del panel de trabajo, hay un área con 8 slots adicionales para añadir más alimentadores. En la figura 41 se puede ver éste módulo secundario de alimentadores adicionales y su ubicación respecto al módulo principal de alimentadores y el panel de trabajo.

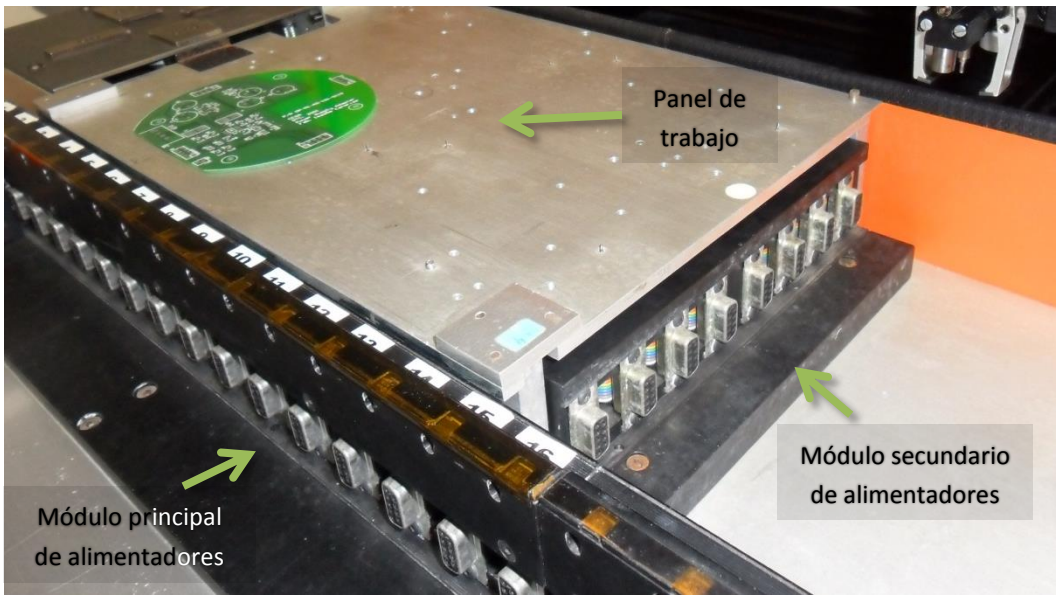


Figura 41. Áreas principal y secundaria de alimentadores.

Estos slots están fijos en la máquina, ya que no están montados sobre la plataforma móvil y por tanto están aparte del sistema de bancos. Además, carecen del bastidor que en el módulo principal contiene el anclaje frontal y el sensor de efecto Hall para el avance de los componentes.

Esto es lógico puesto que al estar montados a 90º respecto al módulo principal de alimentadores, no se puede hacer coincidir el imán con ninguna estructura fija de la máquina cuando está cogiendo un componente. Esto hace que este grupo de slots sólo sea válido para albergar alimentadores de formato más pequeño, como son los de componentes en tubo, que funcionan por gravedad.

Contando ambos módulos, cada máquina tiene actualmente un total de 56 slots para alimentadores.

4.2.9. Teclado

La máquina dispone de un teclado que permite gobernar manualmente el brazo cuando es activado por el software de control que se puede ver en la figura 42. En el software original de la máquina, este teclado se activa cuando se están rellenando los formularios del programa y se llega a las casillas que solicitan las coordenadas donde coger o colocar un componente. El teclado tiene un LED que se enciende para indicar cuándo está activo.

Esto permite llevar el cabezal manualmente hasta la posición deseada, observándola mediante un monitor de vídeo conectado a la cámara instalada en el cabezal. Para ajustarla perfectamente, la imagen que proporciona la cámara tiene una cruz en el centro (mira).

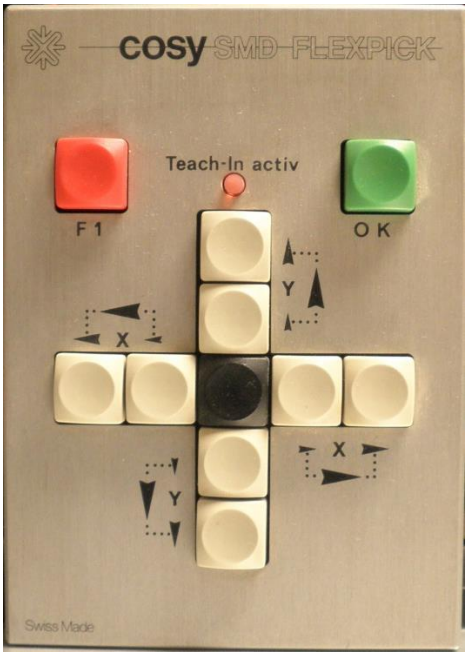


Figura 42. Teclado de control manual del cabezal.

Tras alcanzar la posición exacta un botón del teclado permite enviar las coordenadas en las que se encuentra el cabezal a la máquina. El programa entonces le restará a las coordenadas el

offset de la cámara respecto al cabezal de succión y las almacenará en las casillas correspondientes.

Con este teclado se puede mover el cabezal de la máquina en los ejes *x* e *y* con tres tipos de paso diferentes, según se pulse cada una de las dos teclas correspondientes a la dirección deseada o ambas teclas simultáneamente. De esta manera se puede mover 1 paso cada vez (tecla interior), 10 pasos (tecla exterior) o 100 (ambas teclas simultáneamente) Como cada paso de la máquina es de una décima de milímetro, esto equivale a movimientos de 0,1 mm, 1 mm, o 10 mm con cada pulsación. Si se dejan pulsadas las teclas, cuando el cabezal avance la distancia seleccionada la máquina repetirá la secuencia de nuevo, por lo que seguirá desplazándose, pero "a saltos".

Aunque durante el análisis y las pruebas se desentrañó el funcionamiento del teclado, el protocolo que se debía seguir para activarlo o desactivarlo y el protocolo seguido por la máquina para comunicar al PC las coordenadas especificadas mediante éste, se prefirió no usarlo en la aplicación desarrollada.

La decisión se tomó debido a que se pretende hacer una aplicación de manejo ágil y se consideró que tener que soltar los controles del ordenador para coger los de la máquina era casi equivalente al método empleado por el software original, lo que la hace menos ágil (con la salvedad de que la imagen proporcionada por la cámara de vídeo la tenemos en los formularios de la aplicación en vez de en un monitor de vídeo externo).

Por otra parte, el cable del teclado, sometido al uso que se le ha dado hasta ahora, ya acusa cierto desgaste que podría derivar en una rotura, inutilizando la aplicación hasta que fuese reparado.

Además, el método de colocación del cabezal desarrollado, consistente en hacer clic directamente sobre la imagen de la videocámara en el lugar al que se quiere dirigir el cabezal es mucho más rápido y cómodo. De todas formas se ha incluido un teclado virtual, con funciones similares a las del teclado físico, en el formulario que permite dirigir el brazo de la máquina manualmente.

4.2.10. Panel de conexiones posterior

En la parte posterior de la máquina se encuentra el panel principal de conexiones externas, como son la alimentación, el teclado de control manual del cabezal, el puerto serie, la entrada de aire comprimido, la salida de vídeo y la comunicación con la unidad externa del dispensador de adhesivo, entre otras, como se puede ver en la figura 43.

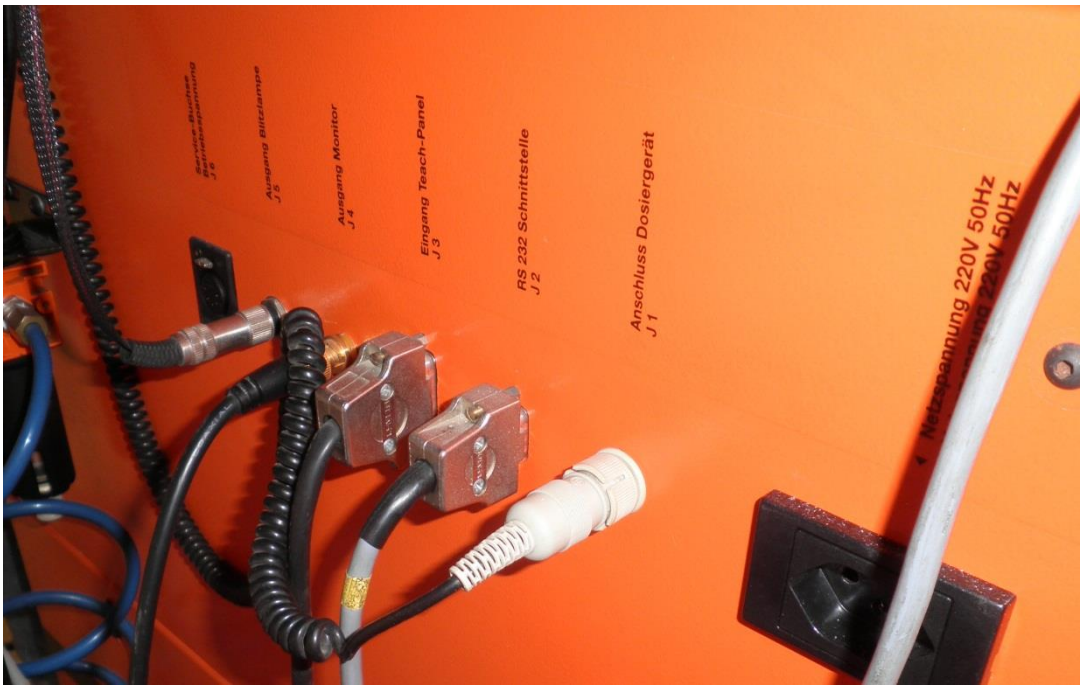


Figura 43. Panel de conexiones posterior.

No es el único panel de conexiones de la máquina, puesto que el cuerpo principal de ésta tiene un panel de conexiones adicional en el lateral derecho para conectar el módulo de alimentadores. Por su parte, el módulo de alimentadores también tiene un pequeño panel en su parte trasera para la interconexión de alimentación y tierra con el cuerpo principal de la máquina, que sirve para que pueda operar el botón de parada de emergencia visto anteriormente.

4.2.11. Panel de controles

En la cubierta izquierda de la máquina se encuentra el panel de controles de la máquina. Como se puede ver en la figura 44, en la parte superior hay un botón de apagado de emergencia que funciona de la misma manera que el frontal. Este botón se distingue del que hay en el frontal porque tiene cerradura, que impide poner la máquina en marcha sin la llave.

Los botones que tiene debajo son luminosos para indicar el estado en que está la máquina. Cuando se activa el botón de la llave, la máquina pasa a estar en "*Stand by*". Para poner en marcha la máquina se debe pulsar el botón "*Power on*". Entonces la máquina ejecuta un pequeño test en el que comprueba la presión de aire, los ejes de movimiento, las pinzas de centrado del cabezal, la tapa del panel de útiles y la estación de centrado. Una vez pasados los test, pasa a estar operativa y disponible para recibir comandos a través del puerto serie.

Los botones "*Alarm on*" y "*Diagnose*" activan la alarma y ejecutan un test de diagnóstico y por tanto no se usan en la operativa habitual. El botón "*Reset*" reinicia la máquina y tampoco

suele ser necesario usarlo. Este botón está protegido con una tapa para evitar reinicios accidentales.

Figura 44. Panel de controles de la máquina.

4.3. Estructura interna de la Cosy

4.3.1. Introducción

En este apartado se describirá la estructura interna de la Cosy con los datos obtenidos durante el estudio que se hizo de la composición y funcionamiento de la misma. Para hacer este estudio se retiraron las tres cubiertas que tapan los accesos al interior de la máquina, señaladas en la figura 45.

La cubierta derecha da acceso al bastidor que soporta los módulos electrónicos, así como a la fuente de alimentación. La cubierta superior permite acceder a la zona de control de aire comprimido, las regletas de conexiones, el cableado de los conectores traseros y todo el sistema electromecánico del movimiento del brazo y cabezal. La cubierta frontal permite acceder al sistema electromecánico del módulo de alimentadores.

El interior de la máquina es bastante espacioso, lo cual es una ventaja para poder observar claramente el cableado entre las partes que la forman y estudiar cómo están organizadas sus funciones. Las cubiertas delantera y lateral son de montaje y desmontaje rápido, la cubierta superior requiere un poco más de cuidado debido a que hay que prestar atención a la lámpara que se encuentra atornillada a ella.

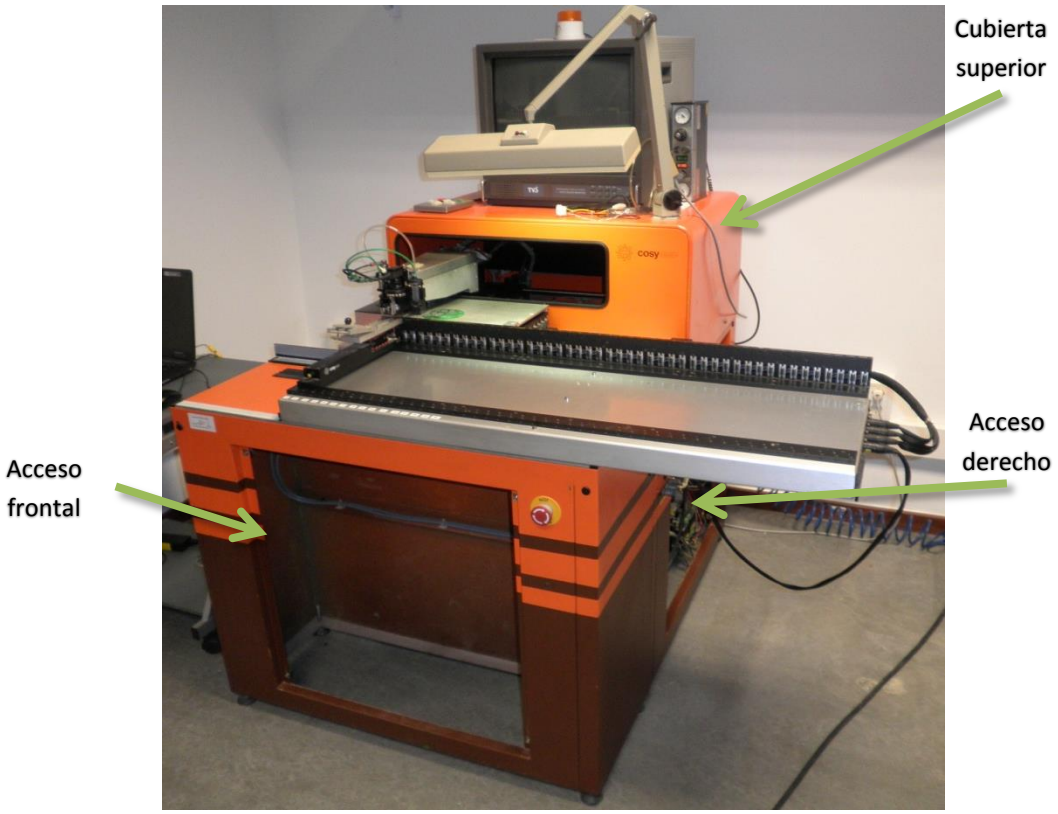


Figura 45. Accesos al interior de la máquina.

4.3.2. Panel de módulos electrónicos

Al abrir la puerta que da acceso al panel electrónico de la máquina, se aprecia que éste está compuesto de módulos conectados en un rack VME estándar de 19'', como se puede observar en la figura 46. Verticalmente, el rack tiene dos *backplanes* donde conectar módulos, ambos para tarjetas Eurocard de altura 3U [**31**] (dimensiones Simple Europa, 100 mm x 160 mm).

En la parte izquierda del panel superior están montados todos los módulos formados por tarjetas de circuito impreso desnudas, mientras que a la derecha se encuentra un módulo en caja que es independiente. En la parte inferior están situados los cinco módulos de control de los motores, montados en cajas y cableados a través de conectores frontales hacia los motores, sus sensores y hacia los canales de comunicación serie con el módulo de control principal.

Tanto las tarjetas de circuito impreso desnudas de la parte superior como los módulos en caja de la parte inferior están conectados a un *backplane*, mientras que el módulo en caja de la fila superior no está conectado al *backplane*, pues, como se ve en la imagen, el backplane superior sólo cubre aproximadamente la mitad del rack.

El *backplane* de la fila superior de tarjetas transporta las señales de los buses de datos, direcciones, control y puertos del módulo de control, así como las líneas de alimentación (+24V,

+5V y GND), mientras que el *backplane* de la fila inferior transporta únicamente las líneas de alimentación.

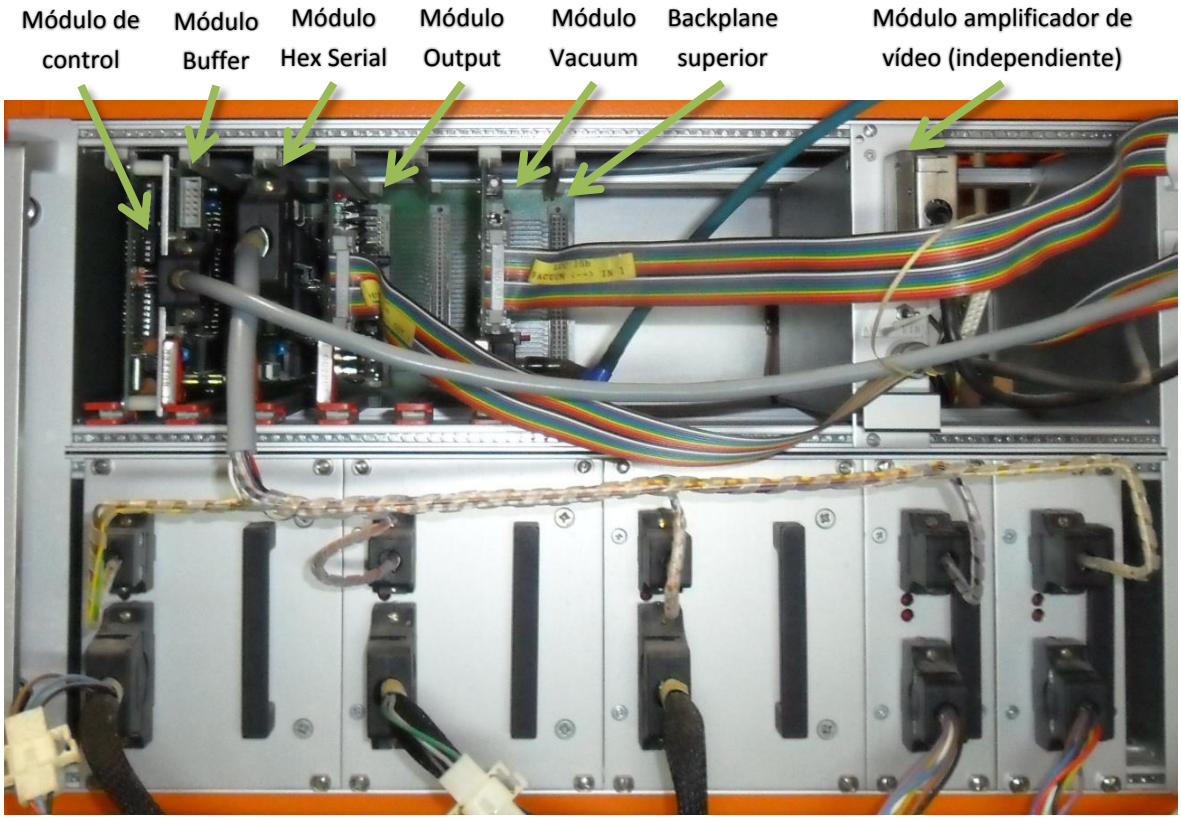


Figura 46. Panel de módulos electrónicos.

Los módulos en caja de la fila inferior son los módulos de control de los cinco motores que posee la máquina, tres de corriente continua y dos paso a paso. Los tres módulos para motores de corriente continua son idénticos e intercambiables, así como los dos paso a paso.

4.3.3. Módulo de control

La máquina está basada en un microcontrolador de 8 bits R6501Q como se puede ver en la placa que alberga el control principal de la máquina, mostrada en la figura 47. Este microcontrolador está montado en un módulo fabricado por Sutter Electronic AG, que incluye la ROM principal, el control de *reset*, el reloj y la lógica de control necesaria para generar las señales de habilitación de otros circuitos integrados y módulos, implementada en una PAL.

Como se puede apreciar en el cristal de cuarzo que se ve en la imagen, el reloj de este microcontrolador es de 4 MHz, suficiente para las tareas que debe llevar a cabo, pues delega en otros microcontroladores el control de algunas partes específicas.

Este microcontrolador de Rockwell está basado en una versión mejorada del microprocesador 6502. Como principales características incluye 192 bytes de RAM, dos

temporizadores / contadores de 16 bits, un puerto serie *full duplex*, 10 fuentes de interrupciones y 32 líneas bidireccionales de entrada / salida organizadas en 4 puertos de 8 bits. De estas 32 líneas, 4 pueden ser activadas por flanco y 8 (el puerto B) disponen de *latch* de entrada.

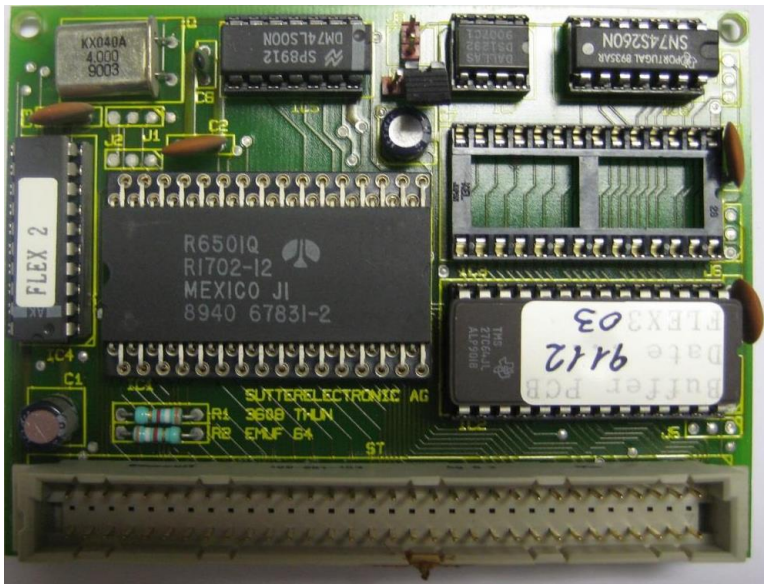


Figura 47. Placa de control principal de la Cosy.

En la parte inferior de la imagen se puede ver un conector de 64 contactos que comunica esta placa con la placa denominada “Buffer” en la máquina. Ambas placas están unidas por este conector y atornilladas por el otro extremo para dar suficiente estabilidad mecánica al conjunto, tal como se puede ver en la figura 48.

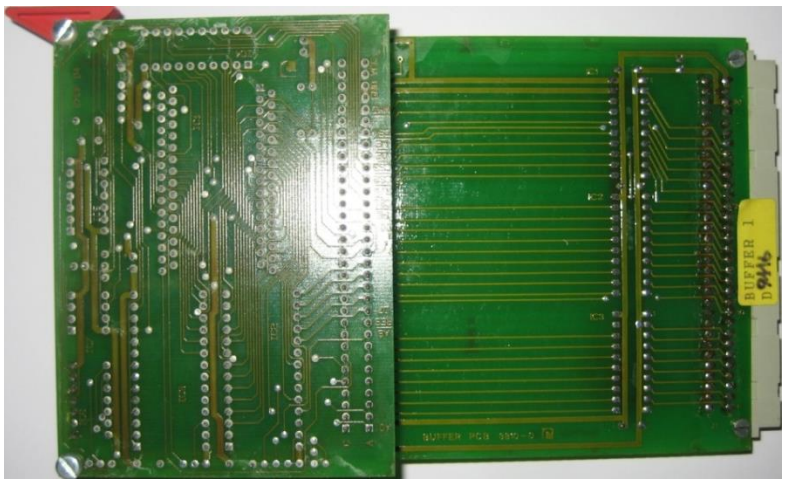


Figura 48. Placa de control unida a la placa Buffer.

La placa Buffer es la que permite conectar el módulo microcontrolador al *backplane* de la máquina, al cual están conectados también el resto de módulos electrónicos. Todos los módulos tienen formato Simple Europa, al igual que la placa Buffer.

A través de la placa Buffer el microcontrolador recibe y emite las señales de sus puertos hacia/desde los diferentes sensores y actuadores presentes en la máquina usando el *backplane*. Además se transmiten las señales de comunicación con el resto de módulos.

En la figura 49 se puede ver la cara de componentes de la placa buffer. Se aprecia la posición del conector de la placa del microcontrolador casi en el centro y los buffers bidireccionales triestado (74LS245) que la conectan al *backplane* en el lado izquierdo.

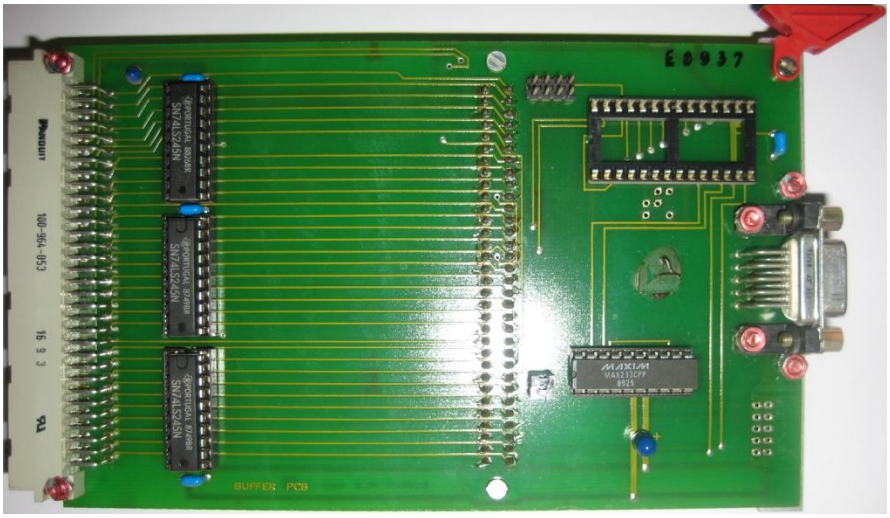


Figura 49. Cara de componentes de la placa Buffer.

4.3.4. Puerto serie externo

Para la comunicación con el ordenador de control, la máquina pick & place usa un puerto serie estándar RS-232. Para gestionar las señales eléctricas relacionadas con este puerto serie se utiliza un circuito integrado MAX233 que está situado en la placa Buffer mencionada anteriormente. En la figura 49 se puede ver (en la parte inferior derecha) este circuito integrado y el conector DB9 que llevará las líneas RS-232 desde el bastidor al conector externo de la máquina pick & place mediante un cable de prolongación.

El puerto serie RS-232 ha estado presente en los ordenadores personales casi desde el comienzo de su éxito comercial y aún hoy en día se encuentra de serie en varios equipos, pero no en todos. Ante el problema de que en los ordenadores personales modernos está empezando a no instalarse de serie (sobre todo en ordenadores portátiles) existen varias soluciones bastante económicas. Se puede instalar un puerto serie como una tarjeta interna en uno de los buses de expansión del sistema o bien como un dispositivo externo conectado a alguno de los puertos disponibles en la carcasa. Para la realización de este proyecto se ha usado un adaptador RS-232 como el de la figura 50, conectado a un puerto USB cuando ha sido necesario.

La norma RS-232 permite el intercambio serie de datos binarios entre un DTE (Data Terminal Equipment) y un DCE (Data Communication Equipment), aunque es común usar ciertas variaciones para otras situaciones.

Figura 50. Adaptador RS-232 para puerto USB.

Físicamente, el puerto RS-232 consiste en un conector tipo DB25 (de 25 pines), pero se ha hecho normal encontrar en ordenadores personales y otros equipos una versión simplificada, con un conector DB9 (de 9 pines). Este es el tipo de conector que poseen los equipos usados durante la realización de este proyecto, tanto en la máquina como en el ordenador.

El cable usado en este interfaz no debe superar los 15 metros con unas velocidades de comunicación bajas (hasta unos 20 Kbps). Sin embargo, es habitual usar cables mucho más cortos y de baja capacidad, con lo que se puede superar ampliamente el valor de la tasa de transmisión de datos. La velocidad de comunicación en nuestro caso está fijada por la máquina pick & place a 4800 baudios.

Los niveles de tensión en el cable, según es estándar, son de -3 a -15 voltios para representar el 1 lógico (marca) y de +3 a +15 voltios para representar el 0 lógico (espacio). Los niveles de tensión elevados permiten mejorar la relación señal/ruido en entornos con fuentes importantes de ruido, como es un entorno de fabricación industrial. En nuestro caso, los niveles de señalización que usan tanto la máquina como el ordenador de control son aproximadamente +/-12 voltios.

El puerto serie puede trabajar en comunicación asíncrona o síncrona, y con canales simplex, half duplex o full duplex. En un canal simplex los datos siempre viajarán en un sentido. En un canal half duplex los datos pueden viajar en uno u otro sentido, pero no simultáneamente, ya que se debe conmutar la línea para cambiar el sentido de la comunicación. En un canal full dúplex, los datos pueden viajar en ambos sentidos simultáneamente. En el caso de la máquina con la que estamos trabajando, la comunicación es asíncrona y full dúplex.

Además de las líneas de datos el interfaz RS-232 posee varias líneas adicionales que se usan en ocasiones para el control del flujo de datos. Así, si el dispositivo que escucha no puede procesar los datos recibidos a la velocidad que los recibe puede "pedir" al que transmite que pare hasta poder seguir. En nuestro caso, la máquina no usa estas señales de *handshaking*, ya que sólo se utilizan los terminales TX, RX y GND del conector.

Para transmitir un conjunto de datos a través del puerto serie hay que dividirlos y encapsularlos en tramas. Tanto el emisor como el receptor deben estar de acuerdo antes de iniciar la comunicación en la estructura que tendrán estas tramas, pues tienen varios parámetros variables y dicha comunicación no será posible si ambos equipos no usan los mismos parámetros.

Esta trama comienza siempre con un bit de *start*, que será un cero lógico, pues en estado de reposo la línea siempre estará a nivel 1 lógico. Le siguen los bits de datos (según el estándar es posible transmitir 5, 6, 7 u 8), empezando por el bit menos significativo y terminando por el más significativo. A continuación se puede incluir, opcionalmente, un bit de paridad, que puede calcularse para paridad par o impar. Por último se emite una señal de *stop* para indicar el fin de la trama, que puede tener una longitud de 1, 1'5 o 2 bits y serán siempre con el nivel de un uno lógico. En el caso de nuestra máquina, el puerto serie está configurado a 8 bits de datos, sin paridad y 1 bit de stop.

En la figura 51 se muestra, a nivel lógico, cómo transmitiría la máquina pick & place una "A" comenzando la trama por el lado izquierdo y terminando por el lado derecho. Posteriormente veremos que el protocolo de comunicación de la máquina pick & place con el ordenador de control se basa en el intercambio de comandos textuales, usando el código ASCII.

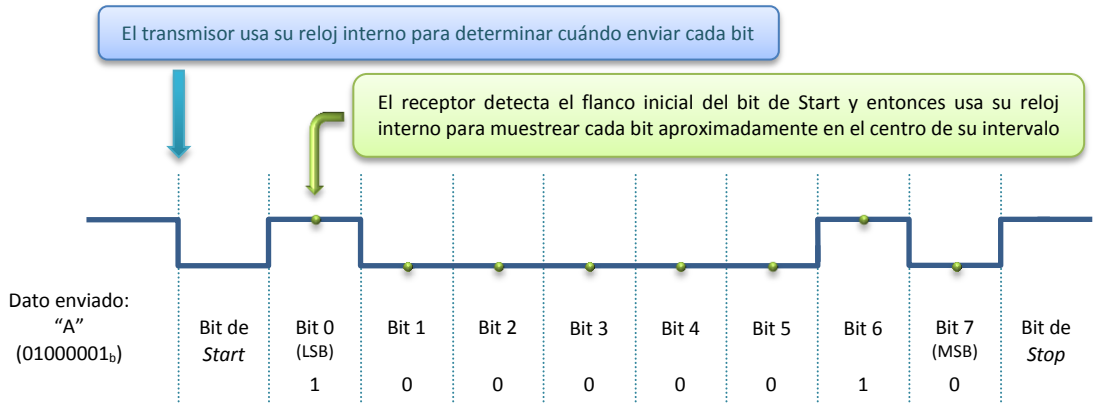


Figura 51. Ejemplo de transmisión serie.

4.3.5. Control de motores

El control directo de cada motor de la máquina pick & place se realiza desde un módulo en caja, conectado en la parte inferior del panel. Cada módulo posee su propio microcontrolador del mismo tipo que el principal (R6501Q), tal como se puede observar en la figura 52.

Figura 52. Módulo de potencia de un motor paso a paso.

Si el microcontrolador principal interpreta (a través de los comandos que recibe desde el ordenador de control desde el puerto serie externo) que debe realizar alguna acción sobre los motores, se pondrá en contacto con el microcontrolador que gobierna la etapa de potencia del motor afectado para indicarle la acción a realizar. Para comunicarse, ambos microcontroladores utilizan un puerto serie, gestionado desde la placa denominada *Hex Serial* que se puede ver en la figura 53. Esta placa contiene 3 DACIA que se encargan de controlar los 5 canales serie hacia los módulos de los motores.

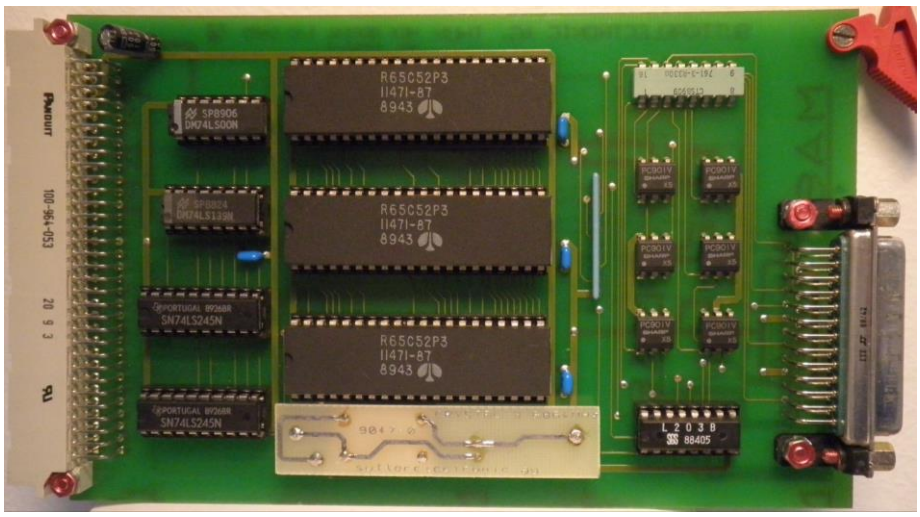


Figura 53. Placa *Hex Serial*.

En la figura 46 se puede ver la conexión de los puertos serie que sale del conector de la placa *Hex Serial* hacia los cinco conectores DB9 que están en la parte superior de los módulos de los motores.

4.3.6. Control de vacío

La placa marcada como "*Input*" contiene, además de varias entradas provenientes de diversos sensores repartidos por toda la máquina, el sensor de vacío. En la figura 54 se puede observar que el conector para las diversas entradas provenientes de los sensores está situado a la derecha. El sensor de vacío en la parte superior derecha de la imagen.

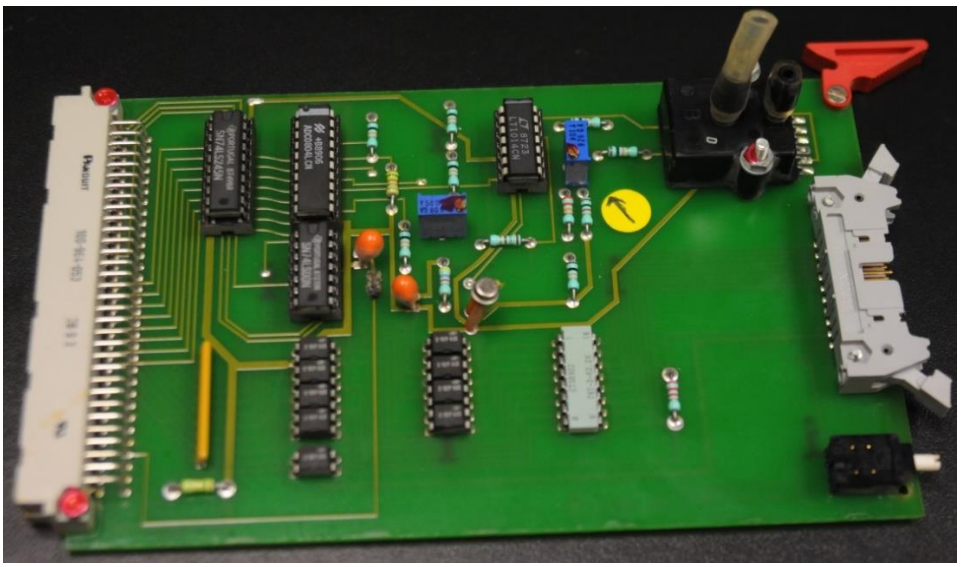


Figura 54. Placa Input.

El sensor se ha desconectado de la manguera que va hasta el regulador de vacío para tomar la fotografía, pero se puede apreciar el tubo de conexión. A su izquierda está su acondicionador de señal y el conversor analógico digital.

En la parte inferior de la placa se puede ver un total de 9 optoacopladores (SFH 610) que se encargan de aislar eléctricamente las entradas de los sensores de las salidas que se dirigen al bus del backplane.

El control neumático de la máquina se encuentra bajo la cubierta superior, como se comentó anteriormente. En la figura 55 se puede ver el detalle de esta zona, donde se pueden ver los dos reguladores, el eyector de vacío y las cuatro electroválvulas que lo forman junto con los tubos de aire comprimido.

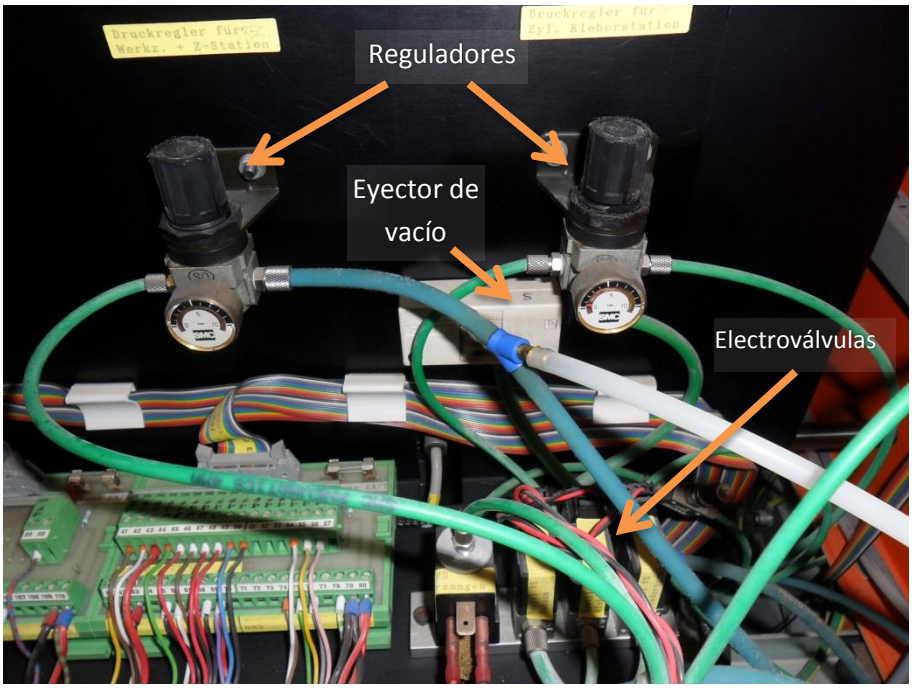


Figura 55. Circuitos neumáticos de la máquina.

Un elemento interesante es el eyector de vacío, que funciona mediante el efecto Venturi. En la figura 56 se muestra el esquema constructivo del eyector, donde el aire a presión entra por la entrada izquierda (SUP) y sale por el silenciador a la derecha. En la salida inferior (VAC) se crea una presión negativa, que se aprovecha en la boquilla para succionar los componentes.

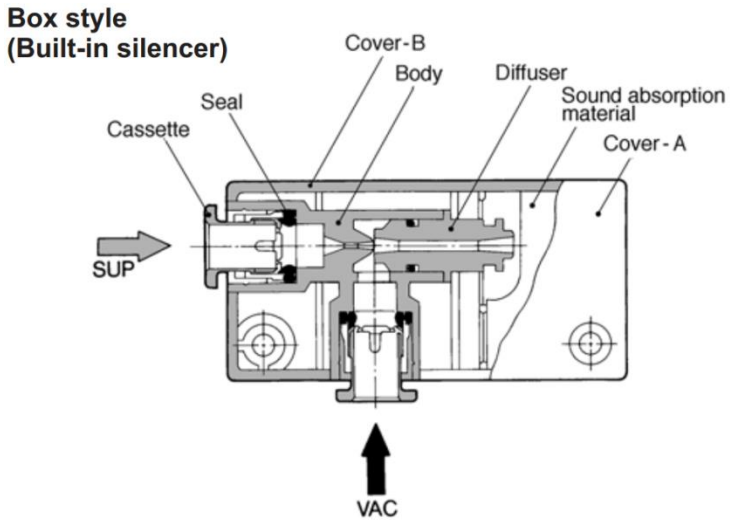


Figura 56. Detalle constructivo del eyector de vacío.

Los reguladores permiten ajustar el nivel de presión para el correcto funcionamiento de la máquina. La presión en la entrada de la máquina, estipulada en las condiciones de trabajo, debe ser de 6 bares. Sin embargo, durante las pruebas se pudo comprobar que con 4 bares seguía

funcionando correctamente. Posiblemente, esto se debe a la curva de funcionamiento del eyector de vacío incorporado en la máquina (ZH10), mostrada en la figura 57.

En la curva se puede ver que la presión de vacío que ofrece el eyector permanece con poca variación (en torno a los -90 kPa, en la cúspide de la curva) ya desde los 0,4 MPa (4 bares) de presión de entrada, que se corresponde con la presión de trabajo mínima observada.

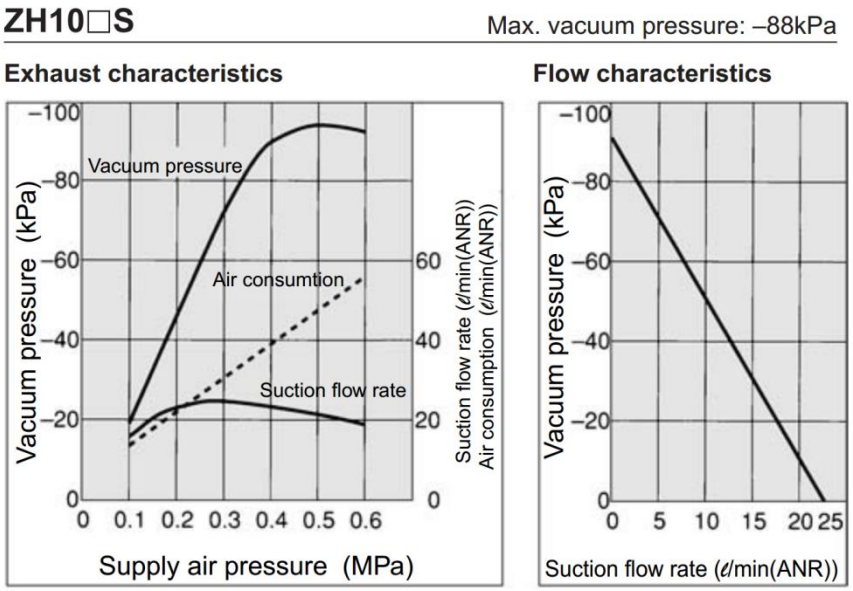


Figura 57. Curvas características del eyector de vacío de la máquina.

4.3.7. Módulo de vídeo

El módulo en caja de la fila superior, mencionado anteriormente, es un módulo de vídeo que se encarga de amplificar la señal captada por la cámara que está montada en el brazo y añadirle una mira para ajustar las coordenadas desde un monitor externo.

Se trata de un módulo independiente, modelo CCD-K2N de Innovationstechnik Gesellschaft für Automation mBH que, aparte de la conexión de alimentación, sólo tiene un cable coaxial de entrada desde la cámara y otro cable coaxial de salida que va directamente al conector de salida de vídeo en la parte posterior de la máquina.

En la figura 58 se pueden ver las tres partes que conforman el sistema de vídeo de la máquina que, salvo porque comparte la fuente de alimentación con el resto de circuitos, es totalmente independiente de éstos.

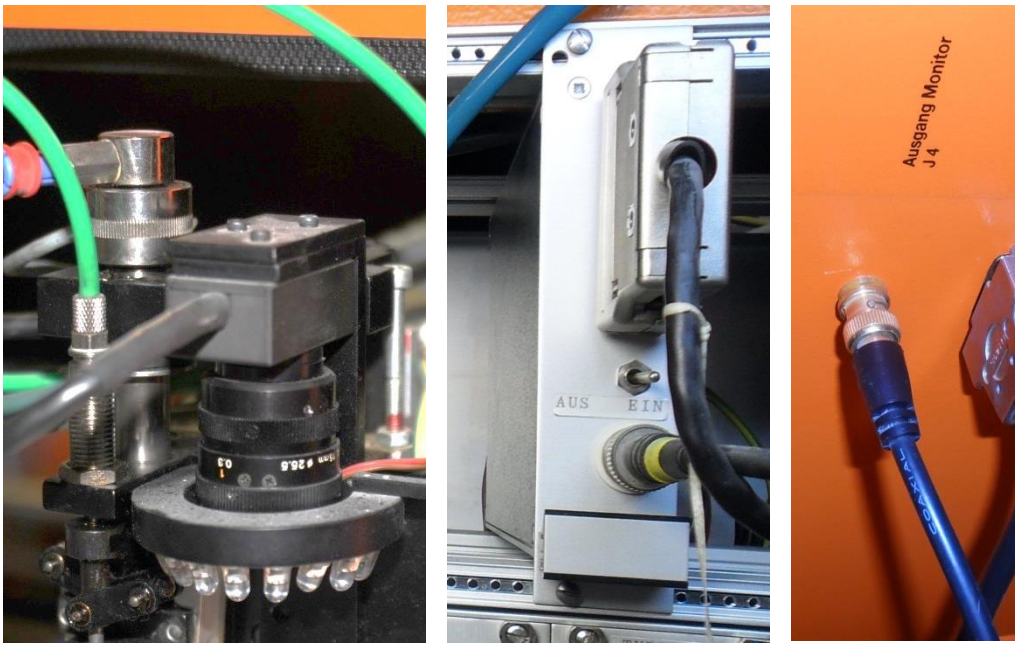


Figura 58. Cámara, módulo de vídeo y conector de salida de vídeo.

La mira consta únicamente de una línea horizontal y otra vertical que se cruzan aproximadamente en el centro del campo visual de la cámara. En la figura 59 se puede ver una de las pruebas realizadas para averiguar las características de esta mira. Como se puede ver, la señal que proporciona la cámara es monocroma y tiene unas bandas negras laterales que reducen un poco la amplitud del campo visual.

En esta prueba se usó un papel milimetrado para tener una medida aproximada del campo visual disponible, siendo este aproximadamente de unos 28 mm x 22 mm. Además, se puede ver que la mira no está en el centro de la imagen, sino que está desplazada ligeramente a la izquierda (algo más de un milímetro), pero este detalle carece de importancia.

En la aplicación desarrollada se ha usado la señal de vídeo proporcionada por la máquina para facilitar al usuario la introducción de coordenadas, que con el programa original de la Cosy se hace con el teclado de la máquina. El usuario simplemente deberá apuntar al punto de interés con la mira. Para ello basta que haga clic con el ratón en el punto de interés que se muestre en la imagen proporcionada por la cámara y nuestra aplicación lo centrará en la mira automáticamente e introducirá las coordenadas elegidas en el campo del formulario que se esté rellenando.

Sin embargo, debido al desplazamiento de la cámara respecto a la boquilla, hay una pequeña parte del panel accesible por la boquilla, pero a la que no es posible apuntar con la mira. Este detalle se explica más adelante, en el capítulo dedicado a la implementación de nuestra aplicación.

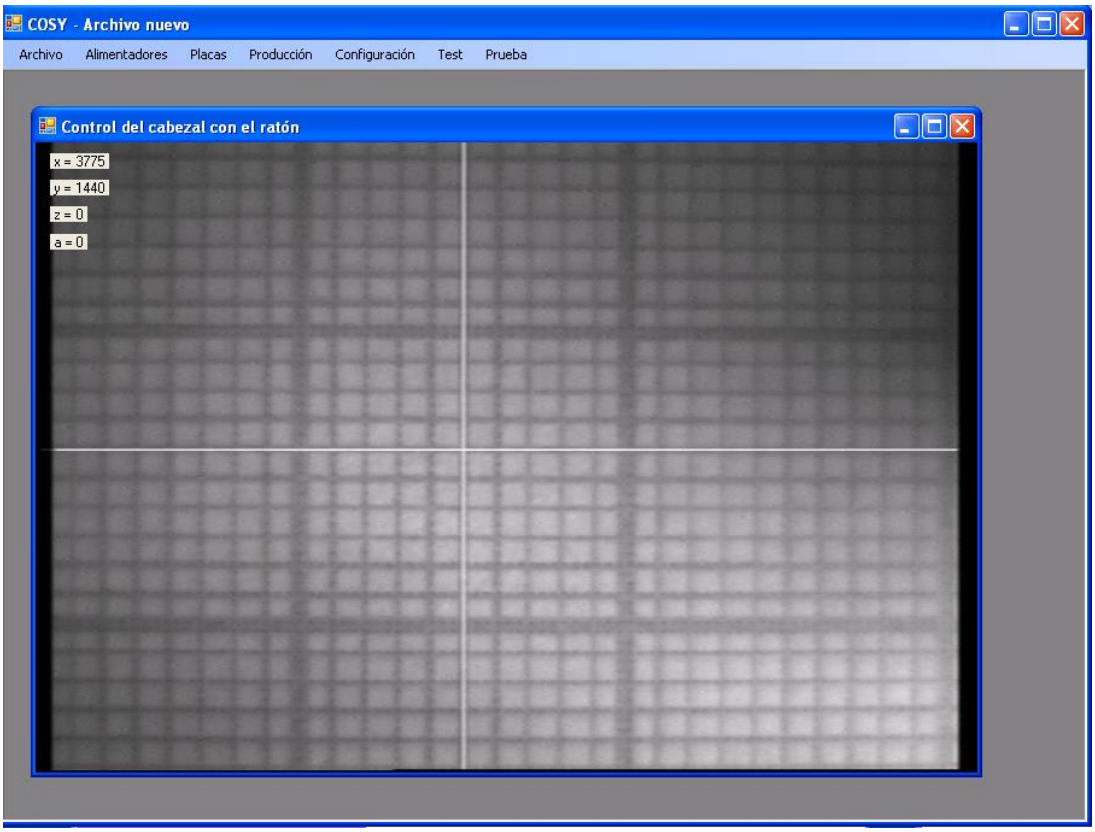


Figura 59. Pruebas de imagen capturando la imagen de la cámara integrada.

4.4. Ordenador de control

Para gestionar las funciones de la máquina es necesario un ordenador de control conectado a ésta mediante un puerto serie. El ordenador debe ir enviando comandos a la máquina que ésta interpretará e irá ejecutando conforme le van siendo enviados. Es posible agrupar una serie de comandos para enviárselos a la máquina de una sola vez, pero la máquina los irá ejecutando secuencialmente según los vaya recibiendo en su buffer, careciendo de la posibilidad de almacenar una secuencia de comandos para realizar su ejecución posteriormente.

La máquina también puede comunicarse con el ordenador enviándole algunos datos de posición usando el teclado de ésta, como se mencionó anteriormente, o también códigos de error, cuando se produce alguno.

El ordenador que está conectado a la máquina es un PC clónico, con un microprocesador Pentium IV y 1 GB de memoria RAM. El sistema operativo instalado es Windows XP Professional y el software de control original de la máquina es el programa *Shaupt.exe*.

Dicho programa es una aplicación en modo texto para el sistema operativo MS-DOS, aunque se ha podido comprobar que funciona correctamente bajo Windows XP. Al ser un programa desarrollado bajo MS-DOS, el programa Shaupt.exe no soporta los nombres largos en el

sistema de archivos, quedando restringidos éstos a 8 caracteres para el nombre y 3 más para la extensión.

Además, el programa Shaupt.exe no hace uso alguno del ratón, utilizando en su lugar un menú principal y unos submenús con un listado de opciones numeradas, a las que se accede escribiendo el número de opción y pulsando la tecla *Enter*.

4.5. Descripción de la LPKF Protoplace

La máquina LKPF Protoplace S [**32**] es un sistema pick & place semiautomático para el montaje profesional de componentes SMD en prototipos y pequeñas tiradas de producción. El diseño ergonómico de esta máquina permite un uso cómodo al operador, permitiéndole montar rápidamente los componentes sobre la PCB. El operador controla manualmente cada paso del proceso mediante una cámara integrada en el cabezal y un monitor que le permite tener una vista ampliada de lo que está haciendo para lograr gran precisión en el posicionamiento.

Figura 60. Máquina pick & place semiautomática LPKF Protoplace.

Para montar un componente con esta máquina, el operador comienza llevando manualmente el cabezal hasta el alimentador desde el que desea coger el componente y baja la boquilla. A continuación activa el vacío mediante un pedal y sube de nuevo la boquilla, capturando el componente deseado. Seguidamente lo lleva hasta la posición de destino sobre la placa y lo orienta adecuadamente. Por último, lo baja a la posición de destino en la placa y

desactiva el vacío para liberarlo de la boquilla, quedando el componente colocado. Los movimientos manuales del cabezal se pueden bloquear tanto en el eje x como en el eje y, de esta forma, se puede hacer un ajuste fino de la posición.

Esta máquina tiene capacidad para albergar hasta 12 alimentadores en el lado izquierdo, como se puede ver en la figura 60. Estos alimentadores pueden ser de tubo (por gravedad) o de carrete (tape & reel). Los alimentadores de tubo pueden servir componentes con encapsulado SO-8 a SO-28 y PLCC28 a PLCC-84. Los alimentadores de carrete pueden servir componentes de 8, 12 y 16 mm.

Como se puede ver también en la figura anterior, el área de trabajo se puede compartir entre la placa que se está montando y alimentadores de bandeja. Además, en la parte trasera de la máquina se encuentra un alimentador motorizado giratorio que almacena componentes en pequeños compartimentos, como se puede ver en el detalle mostrado en la figura 61. En dicha figura también se puede apreciar la pequeña cámara incorporada y la aguja de succión del cabezal.

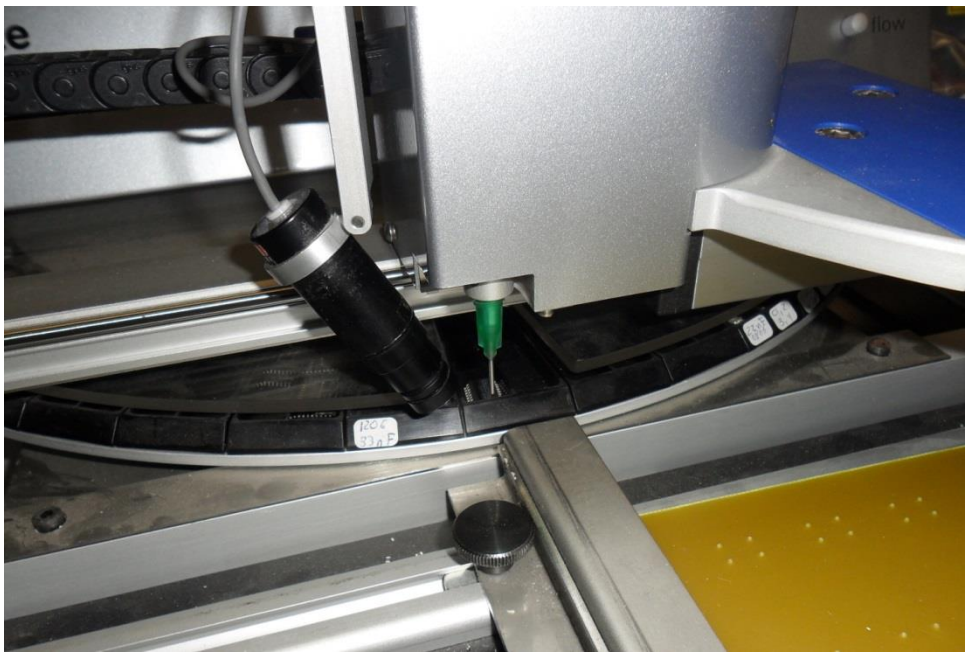


Figura 61. Detalle del cabezal y el alimentador giratorio.

4.6. Descripción de la Assembleon MG-5

La Assembleon MG-5 es la máquina pick & place automática más moderna de que dispone el laboratorio, adquirida en el año 2012. Se trata de una máquina de un cabezal con 5 boquillas que permiten en una sola operación coger hasta 5 componentes para luego

posicionarlos secuencialmente. Esto permite una elevada velocidad de operación de hasta 20.000 componentes por hora.

Como se puede ver en la figura 62, a diferencia de la Cosy estudiada anteriormente, esta máquina si tiene el aspecto típico de las máquinas que se usan hoy en día en las cadenas de montaje automatizadas. Está preparada para integrarse en una cadena de montaje totalmente automatizada, ya que en sus laterales dispone de sendas aberturas estándar para el carril de transporte de las placas entre máquinas. Normalmente la entrada de la placa a montar se efectúa desde el lado izquierdo y la salida de la placa montada se hace por el lado derecho, aunque puede ser configurado de otra forma.

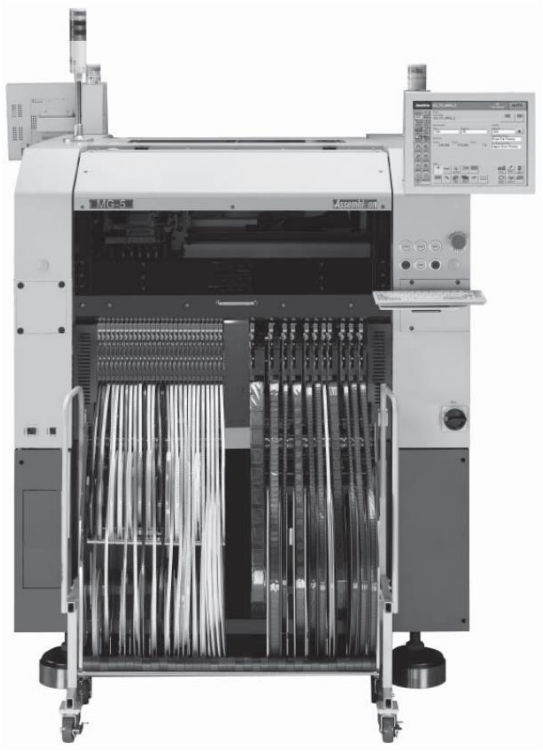


Figura 62. Vista frontal de la MG-5.

Abriendo las tapas que se encuentran en la parte superior, tanto en la parte frontal como en la trasera, se puede ver el interior de la máquina y algunas de las partes de que está constituida. En la figura 63 se puede ver el interior desde la parte delantera. En ella se puede apreciar el gran tamaño del cabezal (situado a la izquierda) que contiene las 5 boquillas mencionadas anteriormente y los mecanismos necesarios para su operación, cuyo detalle se muestra en la figura 64.

La máquina dispone de un ordenador integrado que permite realizar las tareas necesarias para preparar la producción, con su propio monitor, teclado y ratón.

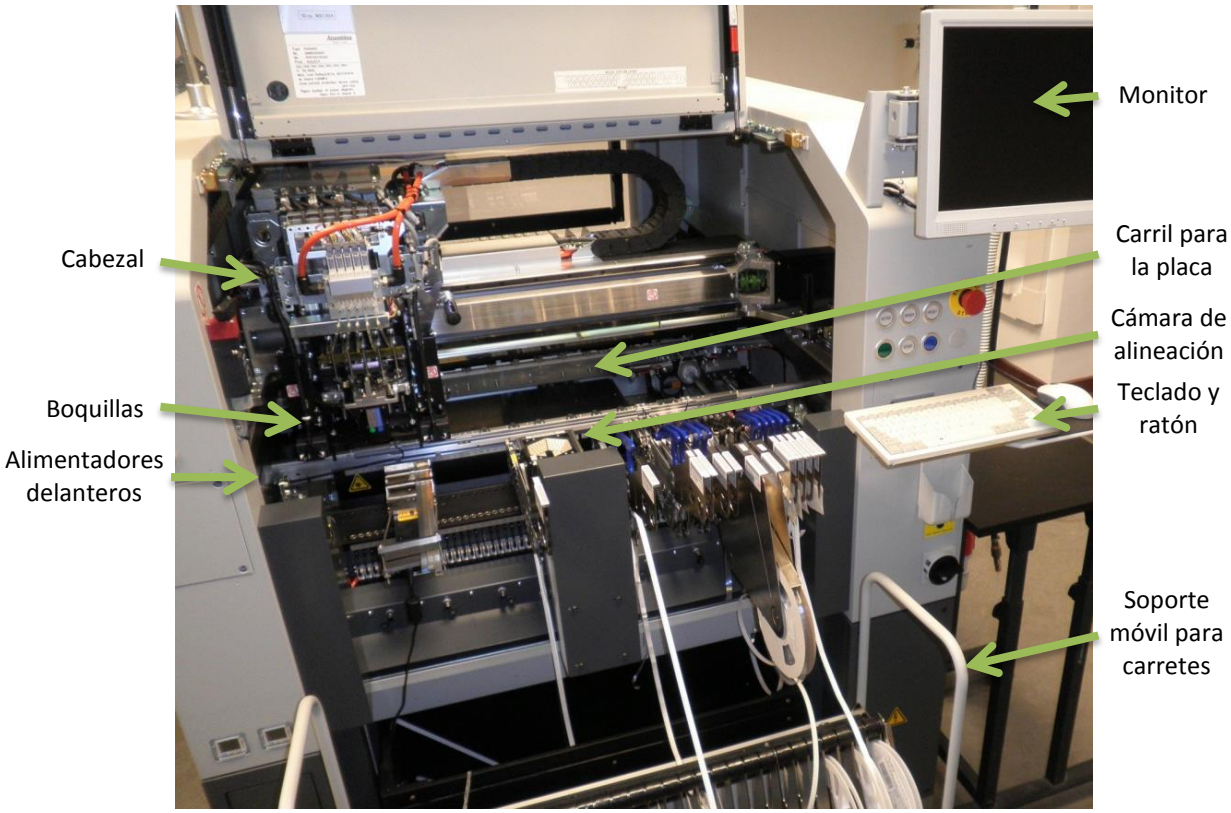


Figura 63. Vista interior delantera de la MG-5.

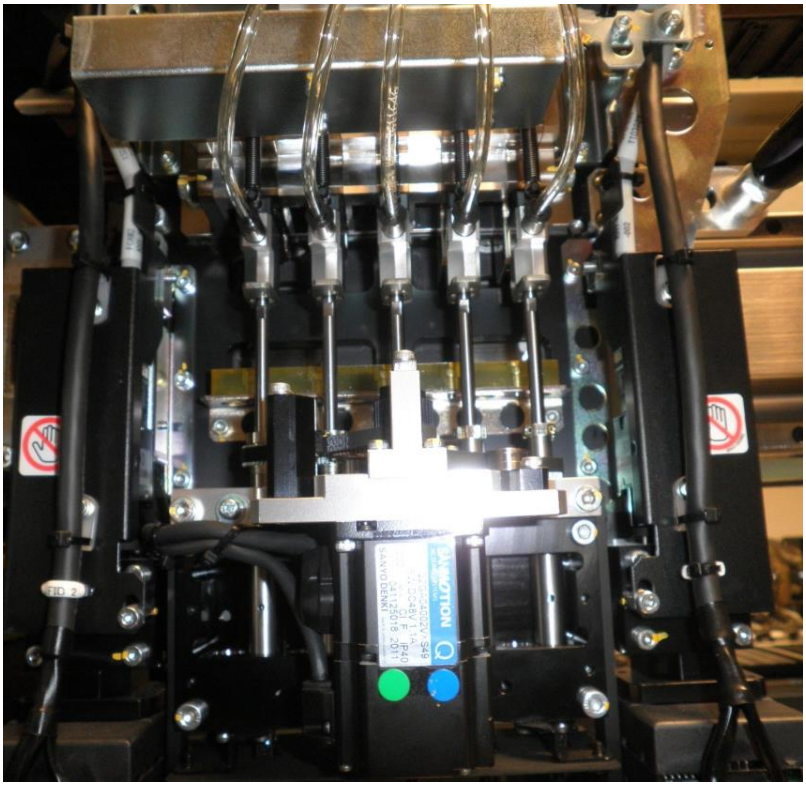


Figura 64. Detalle del cabezal de la MG-5.

Esta máquina puede trabajar con componentes más pequeños que la Cosy estudiada anteriormente, puesto que puede posicionar componentes de hasta métrica 01005. Por otra parte, el tamaño máximo de componente que puede manejar es de 100 × 45 mm. En la figura 65 se pueden ver sus límites de tamaño para los componentes más usuales.

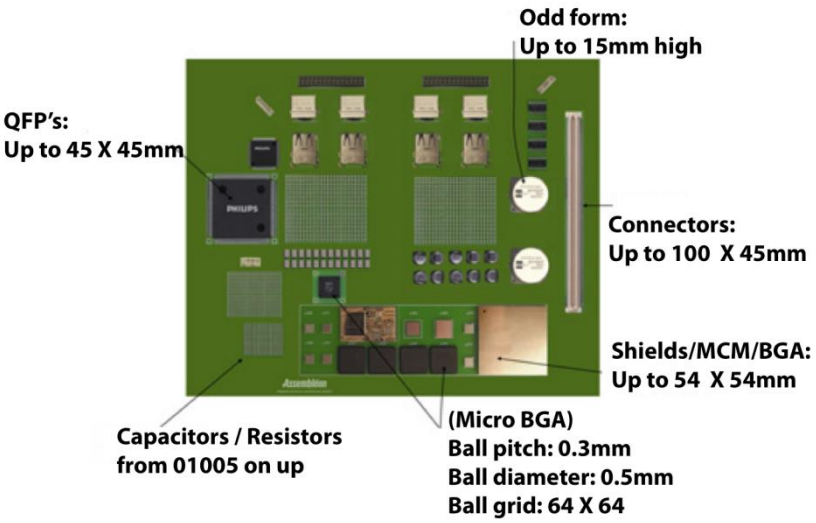


Figura 65. Tamaños de componentes que puede manejar la MG-5

El tamaño máximo de placa que puede manejar es de 57,4 × 46 cm. En total, dispone de espacio para un máximo de 96 alimentadores de carrete, sin contar que dispone de espacio adicional para alimentadores de bandeja.

CAPÍTULO 5. DESARROLLO DE LA SOLUCIÓN

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

5.1. Introducción

Como se ha mencionado anteriormente, la aplicación se ha desarrollado en el lenguaje de programación C#, utilizando para ello el entorno de desarrollo Microsoft Visual Studio Professional 2010. En este capítulo se describirá el desarrollo de la aplicación y se explicará el código fuente de dicha aplicación. Además se plantearán algunos dilemas surgidos durante el desarrollo y se justificarán las decisiones tomadas para su solución.

Salvo en los casos en que es necesario para explicar el funcionamiento del código fuente, en este capítulo no se explica demasiado la aplicación desde el punto de vista del manejo por parte del usuario. Este aspecto se cubre ampliamente en el *manual del usuario* y la *guía rápida para el montaje de placas* que se encuentran en los anexos.

En este capítulo tampoco se ha intentado explicar cada línea de código escrita, sino los puntos principales que definen el modo de operación del programa. El código fuente completo se encuentra también en los anexos.

Como la experiencia del autor con el lenguaje C# y las técnicas de programación empleadas ha ido mejorando a lo largo del desarrollo del proyecto, los primeros módulos realizados han sido refactorizados para que el código sea lo más autoexplicativo posible. Aun así, se han incluido comentarios aclaratorios donde se ha creído necesario.

El entorno de desarrollo utilizado permite la creación rápida de una aplicación de ventanas, poniendo a disposición del desarrollador una serie de herramientas que facilitan gran parte de las tareas comunes.

Una de estas herramientas es un editor visual de formularios para la aplicación (las "ventanas" del programa) que genera todo el código necesario para crear la ventana y todos los controles que el desarrollador incluya en ella, así como para manejar las interacciones que el desarrollador describa.

Este editor generará unos archivos para cada formulario con el mismo nombre que el archivo donde el diseñador ha almacenado la clase que contiene su código, pero terminando en *.designer.cs* o en *.resx*, en lugar de *.cs*, que es lo habitual en los archivos de código C#. Normalmente no se deben editar estos archivos, ya que el editor visual automáticamente introduce cambios en ellos cuando se modifica el formulario.

En los siguientes epígrafes no se tratarán los archivos generados automáticamente, salvo en los casos en que es necesario, pues no han sido creados directamente por el autor sino de manera indirecta, aunque también forman parte del código fuente de la aplicación.

5.2. Estructura

La aplicación desarrollada se ha estructurado a partir de un formulario o ventana principal MDI (Multiple Document Interface – Interfaz de Múltiples Documentos [**33**]) que contiene los menús de acceso a todas las opciones del programa y será el formulario de inicio de la aplicación. Junto a este, otros diecisiete formularios completan la aplicación, además de una gran cantidad

de cuadros de diálogo, sólo mostrados para mostrar un mensaje puntual al usuario, una condición de error o al cargar y guardar archivos.

En la figura 66 se pueden ver los formularios (*form* en Visual Studio) que constituyen la aplicación y sus principales relaciones, coloreados según el menú en el que se encuentran.

Un formulario permite anexar al diseño con el que se está trabajando otros archivos con composiciones de placas realizadas anteriormente desde nuestra aplicación, fusionándolas ambas en el trabajo actual. Desde este formulario se podrá elegir qué se quiere anexar desde el otro archivo (sus alimentadores, sus placas, sus componentes o todos los datos). Con esto se permite al usuario reutilizar trabajos anteriores para acelerar otros trabajos similares. De esta manera, también se puede anexar varias veces la misma placa, para montar simultáneamente una serie de placas iguales en las que sólo cambia su posición sobre el panel de trabajo de la máquina.

La aplicación incluye un formulario que permite ver y procesar los archivos *pick & place* de partida generados desde Altium. En caso de que no estén completos los datos de geometrías de todos los componentes, desde este formulario se puede lanzar simultáneamente el formulario que permite editarlas, para completarlas antes de efectuar la conversión.

Además, la aplicación incluye formularios para ver y editar la configuración de los alimentadores y de las placas que se van a montar, y de todos los componentes que las integran. Adicionalmente se han incluido formularios desde los que se puede ver una representación gráfica de cada placa con la que se está trabajando en el panel actual. En este formulario se puede editar la posición de sus componentes arrastrándolos con el ratón y también cambiar su orientación, girándolos mediante la rueda del ratón. Además, si se detiene el ratón sobre un componente, un pequeño globo emergente muestra sus datos. En caso de que se disponga de archivos Gerber de la placa que se está editando, este formulario mostrará una imagen de la placa de fondo, tras los componentes que se están editando de forma gráfica. Dicha imagen se puede desplazar si es necesario realizar ajustes desde el mismo editor, usando el botón de la rueda del ratón.

Otro formulario permite asignar los archivos Gerber que forman las distintas capas de la imagen a cada placa, así como definir de qué color se desea mostrar cada uno. Un formulario adicional permite ordenar los componentes de la placa atendiendo a sus características principales, permitiendo elegir hasta tres criterios de ordenación simultáneos y jerárquicos. Esto permite definir automáticamente el orden en que los componentes serán montados en la placa en cualquiera de las opciones de producción.

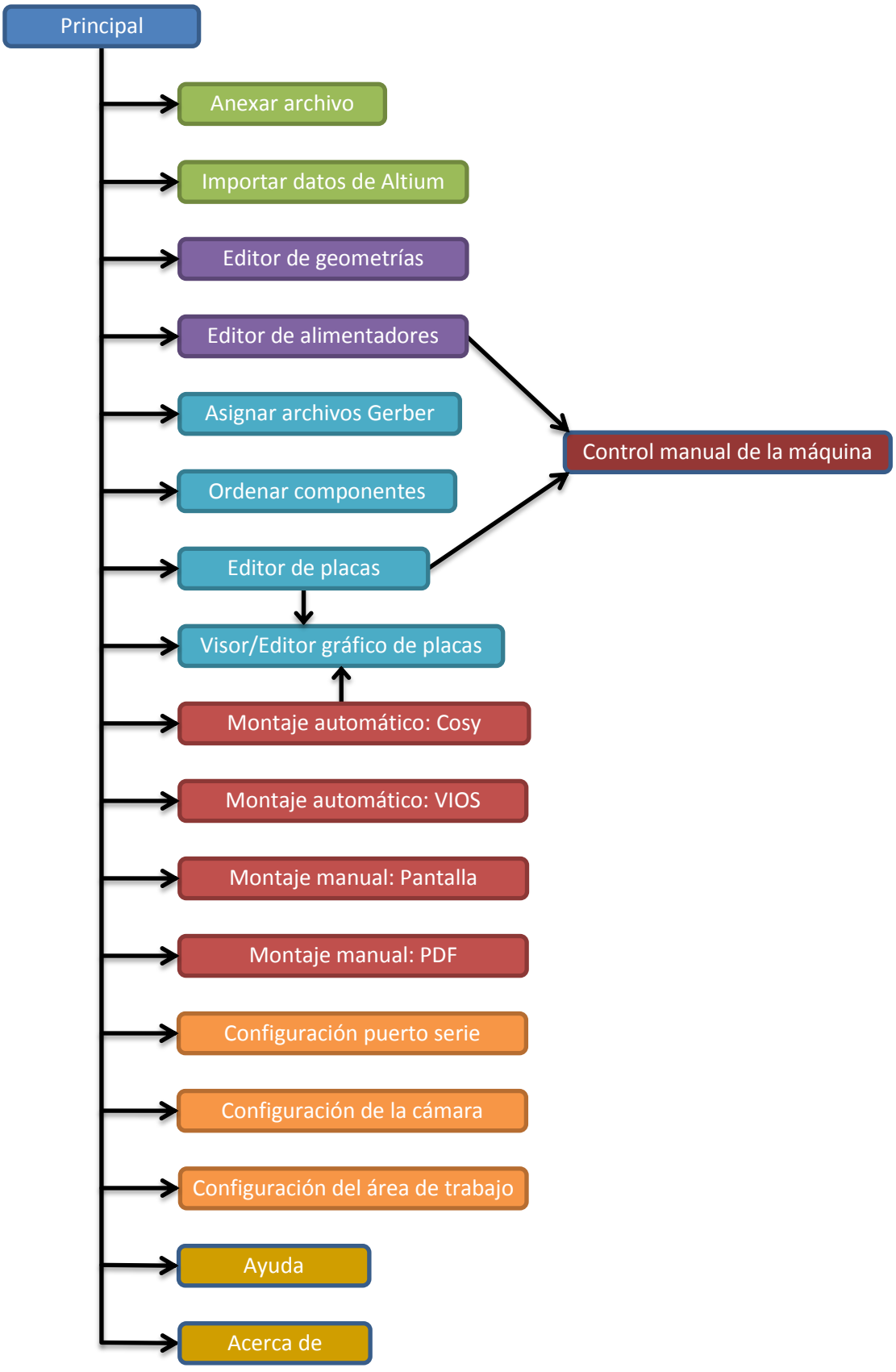


Figura 66. Diagrama de formularios de la aplicación

Existen cuatro formularios de producción que permiten montar la placa siguiendo los diferentes métodos y máquinas disponibles en el laboratorio. El formulario de montaje automático mediante la Cosy permite que el usuario seleccione qué placas desea montar de todas las que estén cargadas en memoria en ese momento y otras opciones de montaje para posteriormente iniciar el montaje de los componentes. Si el usuario lo desea, éste formulario se apoya en el editor gráfico de placas para ir representando visualmente el estado en que se encuentra cada componente al mismo tiempo que se ejecuta su montaje. Para ello va cambiando el color de los componentes en el dibujo de la placa conforme van pasando por las diferentes etapas.

Tanto el formulario de montaje automático como el de montaje manual apoyado en documento PDF permiten elegir la placa que se va a montar para luego ser procesada y obtener el archivo de salida. En el caso de desear generar un archivo PDF, varias opciones permiten definir los detalles del documento generado.

El formulario de montaje manual interactivo muestra en pantalla una vista parecida a la del editor gráfico, añadiendo una serie de controles en el margen izquierdo que permiten al operador moverse por el listado de componentes, controlar la visualización o actualización de éstos y mostrar una vista global de la placa, indicando la zona detallada en la vista principal.

En el menú de configuración se ha incluido un formulario que permita seleccionar el puerto serie que se va a usar para comunicar con la máquina y otro que permite seleccionar y ajustar la cámara de vídeo de la máquina pick & place para usarla durante la edición de datos. Un tercer formulario de configuración permite definir los límites del área de trabajo. Se pueden configurar unas dimensiones arbitrarias, pero el formulario permite configurar automáticamente estas dimensiones para las máquinas disponibles en el laboratorio sin necesidad de conocerlas previamente.

Aunque en el diagrama de la figura 66 se incluye un apartado de ayuda, no se ha contado como un formulario de la aplicación, ya que, tanto para el *manual del usuario* como para la *guía rápida* que se han incluido, se lanza otro proceso que inicia el visor de documentos PDF instalado en el equipo. El manual del usuario es un documento extenso, pero se ha dotado de enlaces en la tabla de contenidos y el índice de figuras, que permiten una rápida navegación hasta el punto que interese leer en cada momento.

Como se mencionó en el capítulo dedicado a describir la máquina, durante el desarrollo de la aplicación, ésta contenía multitud de formularios adicionales agrupados principalmente bajo una opción adicional (llamada Test) en el menú principal.

Algunos servían para ir probando la operativa de la máquina Cosy, con formularios que permitían manejarla por completo manualmente, bien utilizando los controles incluidos en el mismo, bien utilizando un panel virtual en pantalla, que hacía al cabezal seguir los movimientos del ratón, utilizando los botones de éste para activar y desactivar el vacío o cambiar el modo de trabajo de la rueda del ratón. Según el modo, ésta podía servir para subir y bajar el componente o para girarlo a izquierda o derecha.

Otros formularios nos permitían la construcción de comandos complejos, tanto en texto plano como en hexadecimal, para enviarlos en secuencia a la máquina, permitiendo la simulación de pasos intermedios de una producción o condiciones de error.

Formularios adicionales permitían comprobar otras situaciones, hacer pruebas de dibujo de archivos Gerber, realizar cálculos de error adicionales para determinar si se cumplían las necesidades de precisión y probar ideas adicionales para irlas incorporando al resto de posibilidades de la aplicación.

Todo este contenido adicional se ha eliminado de la aplicación final puesto que es innecesario para la operativa normal de la misma. Los datos obtenidos con los análisis realizados mediante estos formularios han sido, en gran parte, incluidos en el desarrollo de las opciones disponibles.

Además, se decidió eliminar también las opciones que permiten un control manual de la boquilla de la máquina Cosy, pues representan un riesgo para la integridad de la máquina o la seguridad de las personas si son mal usadas por un usuario inexperto.

Durante las etapas finales del desarrollo y pruebas realizadas se ha refactorizado una gran parte del código creado inicialmente, aplicando las nuevas técnicas y conocimientos aprendidos durante éste, dando lugar a un código fuente más claro. Esto ha hecho que parte del código se hiciese autoexplicativo, haciendo innecesarios los comentarios en muchas ocasiones, por lo que los comentarios redundantes también se han eliminado del código fuente.

5.3. Inicio de la aplicación

Para el usuario, el formulario principal de la aplicación es aquel por donde se inicia ésta. Sin embargo, en realidad el entorno de desarrollo crea automáticamente el punto de inicio de la

ejecución en la clase *Program*, que está en el archivo *Program.cs*. Se ha modificado el código de esta clase, en principio generado automáticamente, para incluir algunas características extra de usabilidad a la aplicación.

El código generado automáticamente de esta clase incluía solamente el método *Main* de la figura 67. Como se puede observar, después de hacer algunas inicializaciones, simplemente ponía en marcha el formulario principal de la aplicación (*formMDI*).

```
static class Program
{
    /// <summary>
    /// Punto de entrada principal para la aplicación.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new FormMDI());
    }
}
```

Figura 67. Código fuente original de la clase *Program*.

Se ha modificado esta clase para permitirle recibir parámetros desde el sistema operativo, quedando de la forma mostrada en la figura 68. Así, asociando en Windows los archivos de datos de nuestra aplicación (con extensión .pnp) al ejecutable de la aplicación, ésta recibirá el nombre del archivo como parámetro en caso de que el usuario haga doble clic sobre un archivo de datos o lo arrastre sobre el icono de la aplicación.

De esta forma podremos hacer que al iniciarse la aplicación, ésta cargue en memoria el archivo de datos que el usuario activó en el explorador de Windows. Esto se hace asignando el valor del primer argumento recibido (si existe) al campo *rutaFichero* de la clase *datos* en nuestra aplicación. Cuando dicha clase sea llamada, comprobará si se pasó la ruta de un archivo para cargarlo en memoria automáticamente.

Una segunda característica añadida a la clase *Program* original es la detección de otras instancias de la aplicación ejecutándose en la máquina. De esta forma, podemos cancelar la ejecución de esta segunda instancia si lo deseamos.

Si al ejecutarse el método *Main*, éste detecta que había otra instancia de la aplicación ejecutándose, avisa al usuario de que no es posible acceder al mismo puerto ni la misma cámara desde ambas y le deja elegir entre seguir adelante o cancelar la ejecución de esta segunda instancia, como se muestra en la figura 69.

```
static class Program
{
    /// <summary>
    /// Punto de entrada principal para la aplicación.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        if (aplicaciónYaEjecutándose())
        {
            if (MessageBox.Show("Ya hay una instancia de la aplicación ejecutándose. Cada puerto
                serie o cámara puede ser controlado por una única aplicación. ¿Desea
                continuar?", Ya se está ejecutando la aplicación", MessageBoxButtons.OKCancel,
                MessageBoxIcon.Exclamation) == System.Windows.Forms.DialogResult.Cancel)
            {
                return;
            }
        }
        if (args.Length > 0)
        {
            datos.rutaFichero = args[0];
        }
        Application.Run(new FormMDI());
    }

    static bool aplicaciónYaEjecutándose()
    {
        Process procesoActual = Process.GetCurrentProcess();
        Process[] procesosEnEjecución = Process.GetProcesses();

        // Recorre los procesos en ejecución, buscando otro con el mismo nombre que el actual.
        foreach (Process proceso in procesosEnEjecución)
        {
            if ((proceso.Id != procesoActual.Id) &&
                (proceso.ProcessName == procesoActual.ProcessName))
            {
                return true;
            }
        }
        return false;
    }
}
```

Figura 68. Código fuente modificado de la clase *Program*.

Para realizar la detección se ha agregado a la clase *Program* el método *aplicaciónYaEjecutándose*, que devuelve un valor booleano. Este valor será true si ya había una instancia en ejecución o false si esta instancia es la única en ejecución.

De esta forma, es posible usar una instancia de la aplicación, por ejemplo, para realizar un trabajo de montaje en la máquina pick & place Cosy, mientras que con otra instancia se hacen los preparativos previos de otro trabajo.

También sería posible usar el mismo ordenador para controlar las dos máquinas pick & place de este modelo que hay en el laboratorio, ejecutando dos instancias de la aplicación. Para ello, por supuesto, el ordenador deberá disponer de dos puertos serie y dos digitalizadoras de vídeo independientes y cada instancia se debe configurar para usar una pareja diferente.

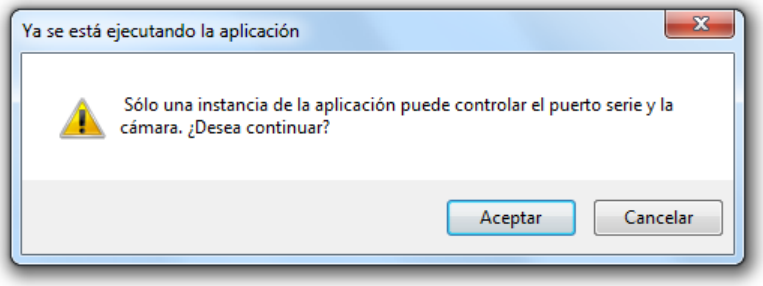


Figura 69. Cuadro de diálogo *Ya se está ejecutando la aplicación*.

En ese caso, mientras las dos máquinas están realizando sus trabajos de montaje, el operador podría iniciar una tercera instancia para ir haciendo preparativos de otro trabajo más.

5.4. Formulario principal de la aplicación

5.4.1. Inicialización

El código de este formulario se encuentra en la clase *formMDI*, contenida en el archivo *formMDI.cs*. Lo primero que hace el constructor predeterminado de la clase al iniciarse es inicializar todos los componentes de la clase. Este comportamiento es el general para cualquier formulario creado con Visual Studio.

Se ha modificado el constructor para que, además, inicialice todos los parámetros de funcionamiento de la aplicación, cargándolos desde las propiedades de configuración de la aplicación, explicadas más adelante. Los datos que la aplicación maneja desde distintos formularios se guardan en una clase especial, denominada *datos* y que está contenida en el archivo *datos.cs*. El método *datos.inicializar* se encarga de hacer la mayor parte de las inicializaciones necesarias.

Durante las inicializaciones se intenta abrir el puerto serie usado la última vez, que está guardado en las propiedades de configuración de la aplicación. Si no se puede abrir (por ejemplo, porque estuviese en uso por otra instancia de la aplicación), inicialmente le mostrará al usuario el aviso de la figura 70 (adaptado al puerto que estuviese configurado en ese momento).

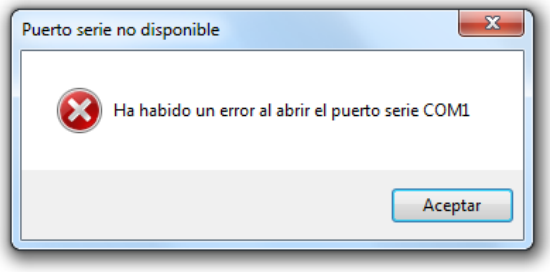


Figura 70. Cuadro de diálogo *Puerto serie no disponible*.

En caso de que no exista el puerto serie que se ha predeterminado en la aplicación (COM1 en el ejemplo anterior), en lugar de mostrar el cuadro de diálogo anterior, se abrirá la ventana de configuración del puerto serie, mostrada en la figura 71, que permite seleccionar uno de los puertos serie instalados en el ordenador. Si no se desea que se muestren estos avisos y formularios cuando no se puede abrir el puerto serie predeterminado, basta desmarcar el *checkBox Comprobar al iniciar*.

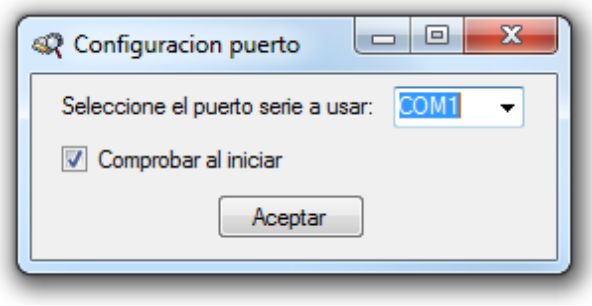


Figura 71. Formulario de configuración del puerto serie.

Si no se selecciona un puerto serie de los disponibles y se cierra la ventana que lo solicita, la aplicación mostrará al usuario un mensaje avisándole de que la comunicación con la máquina no será posible mientras no lo haga desde el menú *Configuración*, pero le permitirá continuar utilizando el programa. Si el usuario sigue adelante sin configurar un puerto serie válido, en las opciones en que se necesite comunicar con la máquina Cosy volverá a salir una ventana de aviso como la mostrada en la figura 72.

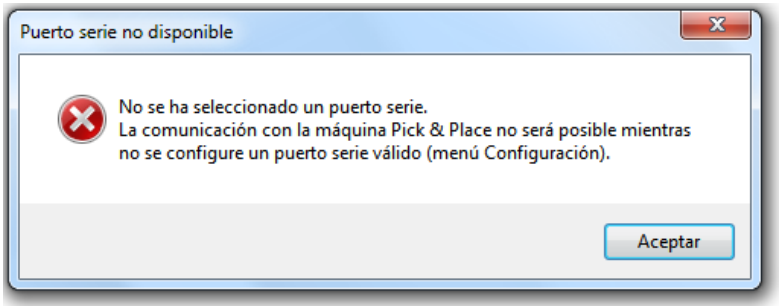


Figura 72. Ventana de aviso de puerto serie no seleccionado.

En caso de que en el ordenador no exista ningún puerto serie, en lugar del formulario de configuración del puerto serie de la figura 71, la aplicación se lo indicaría con el cuadro de diálogo de la figura 73.

En este caso, la aplicación no podría trabajar con las máquinas Cosy, pero si con el resto de máquinas. Como vimos anteriormente, la Cosy depende completamente de la comunicación con el ordenador conectado a su puerto serie para realizar cualquier función de montaje.

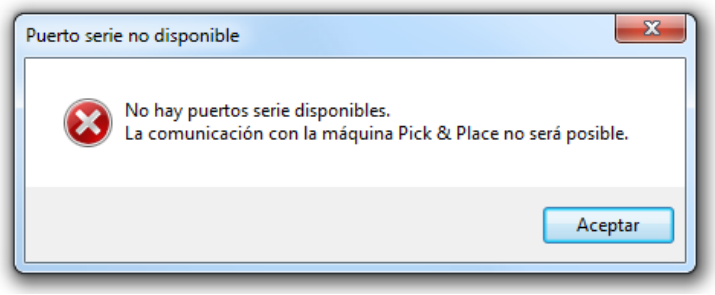


Figura 73. Ventana de aviso de que no hay puertos serie disponibles en el ordenador.

5.4.2. Ventana principal

Una vez hechas las inicializaciones necesarias, se muestra la ventana del formulario principal de la aplicación que se puede ver en la figura 74. Desde el formulario principal se tiene acceso al menú de la aplicación que permite lanzar las distintas ventanas y acciones posibles. Este menú tiene seis opciones que permiten lanzar las acciones agrupadas bajo los nombres: *Archivo, Alimentadores, Placas, Producción, Configuración* y *Ayuda*.

Figura 74. Formulario principal de la aplicación.

Este formulario interactúa con el resto de formularios de la aplicación para crearlos y mostrarlos cuando el usuario lo solicita mediante el menú y guarda una referencia de ellos para evitar crear más de una instancia de cada uno. También guarda en la clase *datos* las rutas en disco (*paths*) de los archivos con que se está trabajando la aplicación para crear los archivos auxiliares a partir de la misma ruta, creando subcarpetas en ésta como veremos más adelante.

Estos archivos auxiliares son básicamente archivos de registro de todo lo acontecido durante el montaje de los componentes sobre las placas (*logs*) en el caso de la Cosy y copias de seguridad automáticas de los archivos de trabajo y de geometrías, cuando van a ser sobrescritos al guardar.

Todos los menús y todas sus respectivas opciones tienen teclas rápidas para acceder a ellos. Dichas teclas se muestran cuando el usuario pulsa la tecla ALT. En la figura 75 se puede ver como ejemplo el menú archivo completamente desplegado, mostrando las teclas rápidas como letras subrayadas en los menús y las opciones.

Además, como es habitual en los sistemas operativos actuales, cada opción que abra un nuevo cuadro de diálogo o formulario lo indica con unos puntos suspensivos al final (…).

En el menú *Archivo* se agrupan las funciones para crear nuevos trabajos, guardar en disco el trabajo realizado, cargar en memoria trabajos realizados anteriormente, fusionar trabajos creando composiciones de placas más complejas e importar trabajos realizados desde Altium.

También se pueden cargar y grabar otros archivos accesorios, como son los archivos de definición de las dimensiones de los componentes (geometrías) propias de la aplicación e importar los archivos de geometrías utilizados por el software Cad2cad (packages.dim). Por último, en este menú se encuentra también la opción para salir del programa.

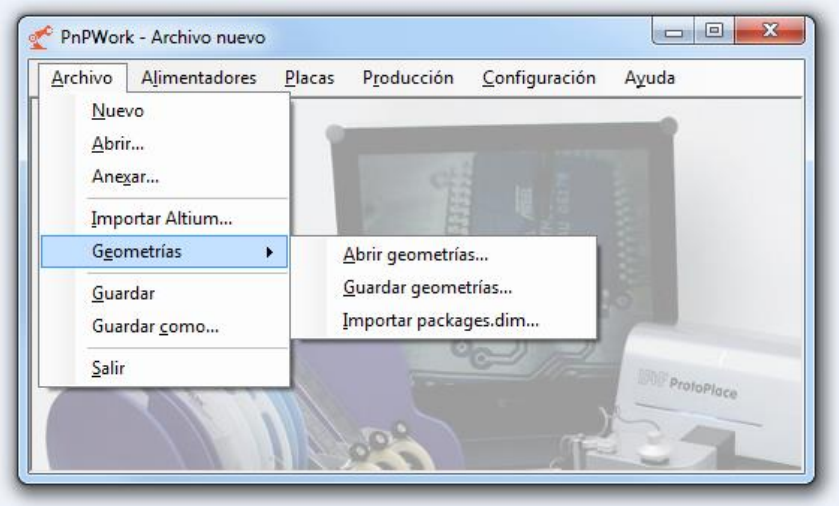


Figura 75. Menú *Archivo* desplegado, mostrando las teclas rápidas.

La primera opción del menú *Archivo* (*Archivo* → *Nuevo*) permite vaciar las estructuras de datos en memoria para comenzar un trabajo nuevo desde cero. Cuando se inicia el programa se crean vacías las estructuras de datos en memoria, y esta opción tiene el mismo efecto. Por ello, al iniciarse el programa, en la barra de título ya aparece el texto “Archivo nuevo” junto al nombre del programa, indicando esta condición.

Si al seleccionar esta opción, se había modificado cualquier contenido de las estructuras de datos guardadas en memoria mediante los editores incluidos en nuestra aplicación, el

programa avisará al usuario que había datos sin guardar y pedirá confirmación antes de borrarlo todo mediante un cuadro de diálogo.

Para saber si se han hecho modificaciones en las estructuras de datos almacenadas en memoria se ha creado un campo público booleano en la clase *datos* llamado *modificados*. De esta manera, desde el resto de la aplicación se accederá a él como *datos.modificados*, que es autoexplicativo (*if (datos.modificados)…*). Cualquier método que cambie el valor de las estructuras de datos en memoria que albergan los alimentadores o las placas con sus componentes, pondrá en este campo un valor *true*. En la medida de lo posible se ha intentado seguir este sistema de nombres autoexplicativos en todos los elementos del código fuente.

5.4.3. Interfaz MDI

En el menú configuración, que se puede ver en la figura 76, existe una opción especial, llamada *Interfaz MDI*, que permite variar el comportamiento de este formulario y el de todos sus hijos.

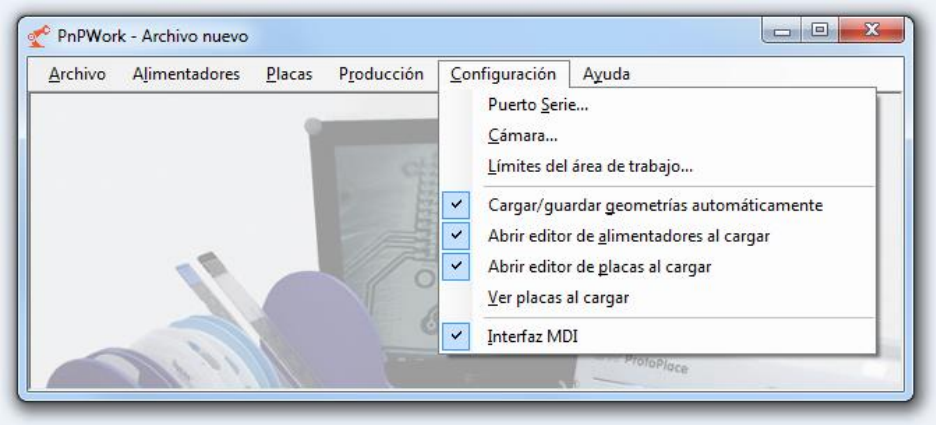


Figura 76. Menú *Configuración* desplegado.

De manera predeterminada, esta opción está marcada, haciendo que este formulario se comporte como un formulario MDI estándar, es decir, contiene en su interior al resto de formularios que abra (salvo los formularios modales), como se puede ver en la figura 77.

Si los formularios hijos no caben en el área del formulario padre o se desplazan fuera de sus límites, aparecerán barras de desplazamiento en éste, como se puede apreciar en la imagen.

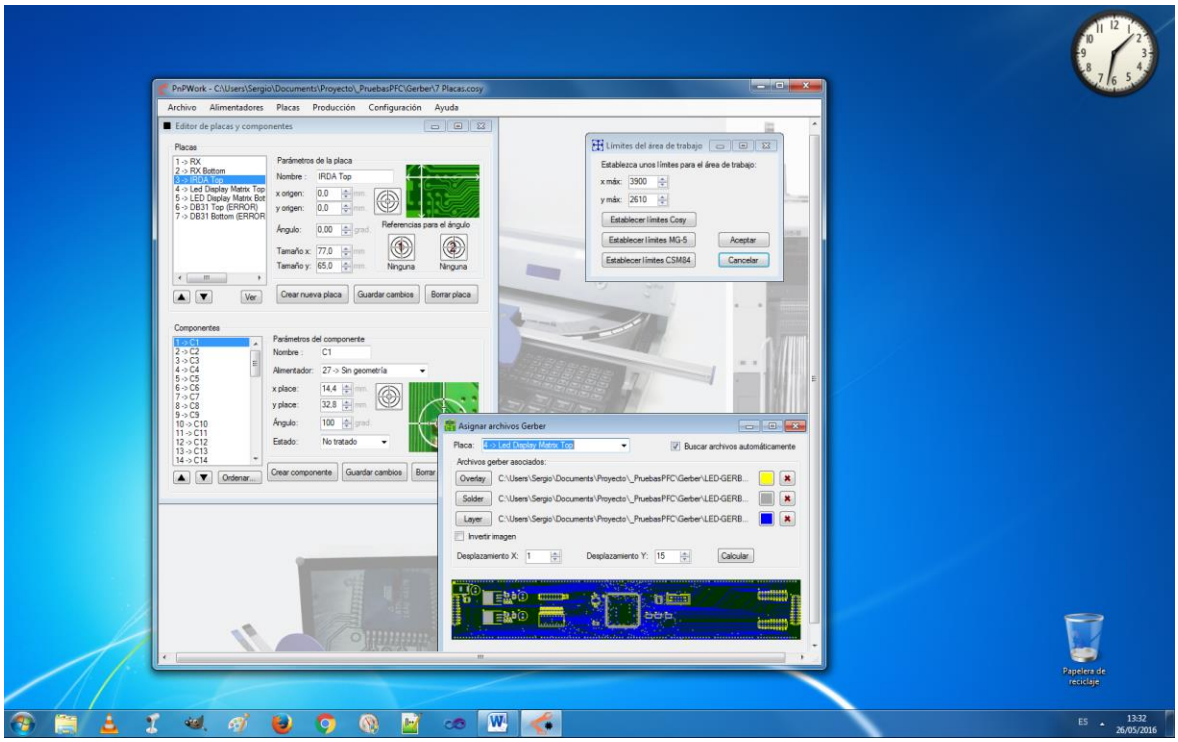


Figura 77. Aplicación en modo Interfaz MDI.

Si se desmarca la opción, este formulario deja de ser MDI y por tanto el contenedor del resto. En ese caso se puede manejar cada ventana como un elemento separado en Windows (formulario SDI), como se puede ver en la figura 78.

En la parte inferior se ha hecho clic en el icono agrupado de la aplicación en la barra de tareas para que se despliegue. En el despliegue se puede ver que se muestra cada ventana de la aplicación como una ventana independiente, mostrando los iconos particulares de cada ventana en lugar del icono general de la aplicación.

Esto se ha hecho así para permitir un cierto grado de personalización de la aplicación a los gustos del usuario. Además, no está claro cuál es el tipo de ventanas más útil y productivo, la propia Microsoft ha pasado, en sus diferentes versiones del paquete ofimático Office, de SDI a MDI y luego otra vez a SDI.

Sin embargo, en ocasiones puede ser necesario estirar un formulario más de lo habitual. Por ejemplo, si al importar un archivo de Altium se ve que se han incluido muchos campos, difícilmente se podrá leer el contenido de todos los campos a la vez si no se estira el formulario. En esas ocasiones puede ser necesario poder sacar el formulario de los límites de su padre.

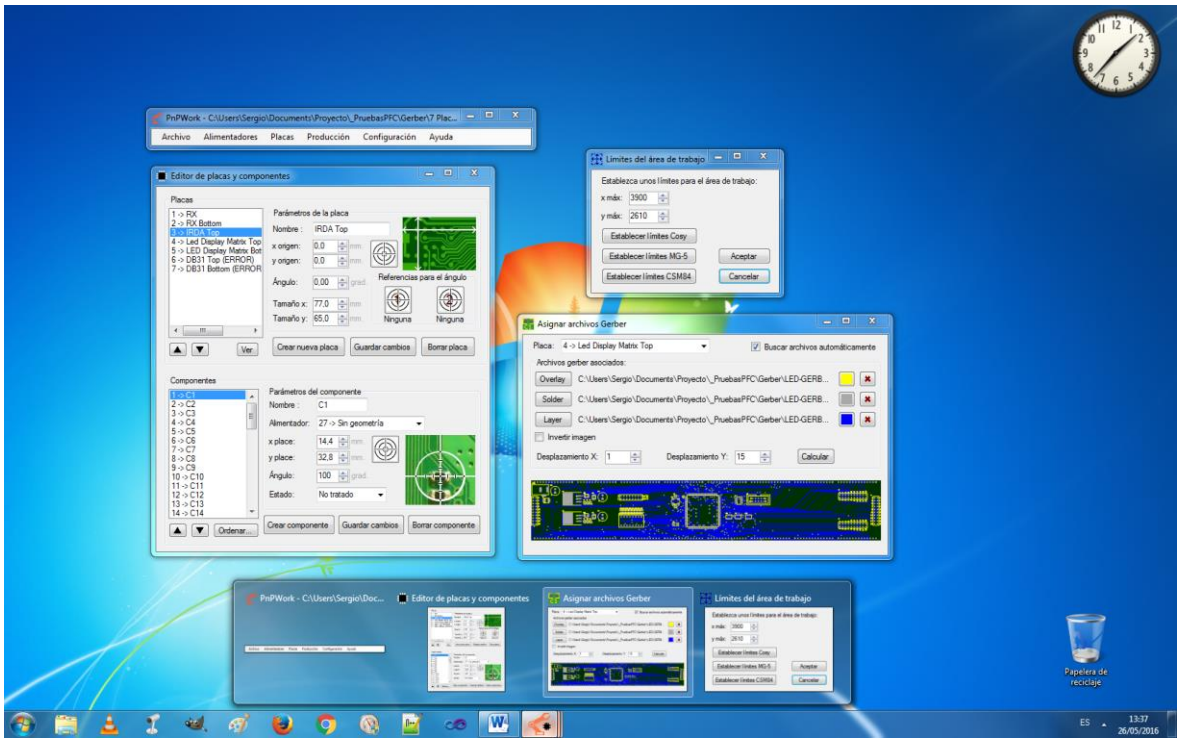


Figura 78. Interfaz MDI desactivada.

El código fuente para realizar este cambio se muestra en la figura 79.

```
private void interfazMDIToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (interfazMDIToolStripMenuItem.Checked)
    {
        // Configura la interfaz con ventanas independientes.
        interfazMDIToolStripMenuItem.Checked = false;
        Settings.Default.EstadoInterfaz = this.WindowState;
        this.WindowState = FormWindowState.Normal;
        Settings.Default.AltoInterfaz = this.Height;
        Settings.Default.AnchoInterfaz = this.Width;
        this.Height = 62;
        this.Width = 560;
    }
    else
    {
        // Configura la interfaz MDI.
        interfazMDIToolStripMenuItem.Checked = true;
        this.Height = Settings.Default.AltoInterfaz;
        this.Width = Settings.Default.AnchoInterfaz;
        this.WindowState = Settings.Default.EstadoInterfaz;
    }

    // Actualiza las ventanas hija.
    Form padre = interfazMDIToolStripMenuItem.Checked ? this : null;
    for (int n = 1; n < Application.OpenForms.Count; n++)
        while (Application.OpenForms[n].MdiParent != padre)
            Application.OpenForms[n].MdiParent = padre;
}
```

Figura 79. Código fuente del método *interfazMDIToolStripMenuItem_Click*.

Una curiosidad encontrada durante el desarrollo de esta parte del código, que tal vez sea un bug del sistema operativo o de .NET, se refleja en la última instrucción del método. Tras

comprobar que, aleatoriamente, algunas ventanas hija no salían o no volvían a entrar en la ventana padre. Sin encontrar ninguna explicación para ello, se probó a asignar el valor del campo *MdiParent* repetidamente hasta que cambiase el valor efectivamente y esto solucionó el problema.

5.5. Propiedades de configuración de la aplicación

En las primeras versiones de la aplicación, los parámetros de configuración de la misma y las constantes relacionadas con la operación de la máquina se definieron mediante constantes integradas en la aplicación. En la figura 80 se puede ver un extracto parcial de la definición de estas constantes a modo de ejemplo.

```
//Constantes que definen el hardware de la máquina:
public const int xMax = 3900;                   // Coordenadas máximas del cabezal de
public const int yMax = 2610;                   // la COSY, expresado en décimas de mm.
public const int zMax = 200;                    // y ángulo de giro máximo del cabezal,
public const int aMax = 3200;                   // expresado en grados centesimales.

public const int offsetCamaraX = 340;           // Offset desde la cámara a la aguja de
public const int offsetCamaraY = 100;           // succión, expresado en décimas de mm.

public const int offsetDispensadorX = 840;      // Offset desde el dispensador de adherente
public const int offsetDispensadorY = -100;     // a la aguja, expresado en décimas de mm.
```

Figura 80. Ejemplo de las constantes integradas en la aplicación.

En las versiones posteriores, se almacenó esta información al área de *configuración de la aplicación* [**34**]. La *configuración de la aplicación* es una característica del entorno que nos permite almacenar y recuperar valores de propiedades y otra información de nuestra aplicación. También nos permite conservar las preferencias personalizadas de las aplicaciones y los usuarios en el equipo en que se ejecuta la aplicación.

Para acceder a la funcionalidad proporcionada por esta característica del entorno, se ha usado la clase *Properties.Settings*, creada por el entorno en el espacio de nombres de nuestra aplicación. Para referenciarla convenientemente, se ha incluido, al inicio de los ficheros en que se usa, la línea que se muestra en la figura 81. El entorno de desarrollo genera todo el código de esta clase automáticamente y nosotros podemos usarla en cualquier parte de nuestra aplicación.

```
using cosy.Properties;
```

Figura 81. Referencia a las propiedades de la aplicación.

Estas propiedades de configuración de la aplicación son almacenadas en un archivo de configuración con formato XML, permitiendo la edición del mismo con cualquier editor de texto

plano o usando una aplicación específica. El archivo se guarda automáticamente en el mismo directorio que el archivo ejecutable de la aplicación y con el mismo nombre que dicho ejecutable, añadiendo la extensión *.config*.

El hecho de incluir en este archivo las constantes usadas para configurar apropiadamente la aplicación, permite al usuario tener acceso a éstas, pudiéndolas cambiar por otros valores en caso de que sea necesario. En la figura 82 se muestra como ejemplo la parte del contenido de este archivo XML que hace referencia a las constantes de la figura 80, comprobándose que es muy fácil deducir el significado de cada sección.

Cada etiqueta *<setting>* incluye un atributo *name*, cuyo valor (puesto entre comillas) especifica el nombre de la propiedad almacenada, mientras que el contenido de la etiqueta *<value>* (hija de la etiqueta *<setting>*) es el valor que se desea asignar a dicha propiedad en nuestra aplicación.

Cada etiqueta *<setting>* termina en su correspondiente *</setting>* y cada etiqueta *<value>* termina en su correspondiente *</value>*. Además cada etiqueta hija está completamente anidada dentro de su etiqueta padre, tal como prescriben las reglas XML.

```
<setting name="xMax" serializeAs="String">
    <value>3900</value>
</setting>
<setting name="yMax" serializeAs="String">
    <value>2610</value>
</setting>
<setting name="zMax" serializeAs="String">
    <value>200</value>
</setting>
<setting name="aMax" serializeAs="String">
    <value>3200</value>
</setting>
<setting name="offsetCámaraX" serializeAs="String">
    <value>340</value>
</setting>
<setting name="offsetCámaraY" serializeAs="String">
    <value>100</value>
</setting>
<setting name="offsetDispensadorX" serializeAs="String">
    <value>840</value>
</setting>
<setting name="offsetDispensadorY" serializeAs="String">
    <value>-100</value>
</setting>
```

Figura 82. Ejemplo de las constantes integradas en un archivo XML de configuración.

En condiciones normales, y para las máquinas con las que opera nuestra aplicación nunca será necesario editar este archivo. Sin embargo, el hecho de disponer de esta posibilidad proporciona un grado de flexibilidad adicional para adaptar la aplicación a situaciones no previstas durante el desarrollo de este proyecto, como por ejemplo, operar con otra máquina ligeramente diferente o alterar alguno de los límites por una avería irreparable de las máquinas actuales que limiten alguno de sus parámetros.

Esta adaptación la podría hacer cualquier usuario con los conocimientos mínimos sobre la estructura de un lenguaje de marcas de tipo XML mencionados en los párrafos precedentes. Además, los cambios estarían listos para funcionar al instante, reiniciando la aplicación. Por tanto no hay necesidad de que sea un programador cualificado el que haga los cambios ni de editar el código fuente de la aplicación y recompilarla.

Además de guardar las constantes que definen el funcionamiento de la aplicación, se ha guardado en las *propiedades de la configuración de la aplicación* las opciones que ha elegido el usuario en el menú *Configuración*. Así se ha logrado que, cuando se pone en marcha la aplicación, cargue los valores elegidos para estas opciones desde este archivo para configurarla como el usuario la dejó la última vez que la usó.

También se ha guardado de la misma manera el nombre del archivo de geometrías usado por última vez, ya que lo más probable es que siempre se utilice el mismo. De esta forma, y manteniendo seleccionada la opción "*Cargar/guardar geometrías automáticamente*" del menú *Configuración* de nuestra aplicación, éste archivo se cargará en memoria automáticamente cuando se inicia la aplicación, estando así las geometrías siempre disponibles sin que el operador tenga que intervenir.

5.6. Copias de seguridad automatizadas

Cada vez que se llama al método que se encarga de guardar un fichero (salvo los ficheros de registro, que se explican en el apartado de producción, lo primero que hace éste es llamar a su vez al método encargado de realizar una copia de seguridad con la misma ruta. De esta forma, si el usuario se equivoca y sobrescribe un archivo sin querer, siempre puede abrir la carpeta de copias de seguridad y recuperar el archivo original. El código de este método es el que se muestra en la figura 83.

Para evitar saturar el disco con copias de seguridad, la aplicación tiene un parámetro de configuración que permite especificar la cantidad máxima de copias de seguridad que puede hacer de cada archivo que se guarde. El valor máximo configurado inicialmente es de 10 copias de

seguridad adicionales para cada archivo que se grabe, pero el usuario lo puede cambiar por otro valor mediante las propiedades de configuración de la aplicación.

```
public static bool hacerBackup(string rutaFichero)
{
    if (File.Exists(rutaFichero))
    {
        try
        {
            string directorioFichero = Path.GetDirectoryName(rutaFichero);
            string nombreFichero = Path.GetFileName(rutaFichero);
            string directorioBackup = Path.Combine(directorioFichero, dirBackup);

            if (Directory.Exists(directorioBackup))
            {
                for (int n = numArchivosBackup; n > 0; n--)
                {
                    string nombreFicheroActual = nombreFichero + ".backup" + (n - 1).ToString();
                    string ficheroActual = Path.Combine(directorioBackup, nombreFicheroActual);
                    if (File.Exists(ficheroActual))
                    {
                        string nombreFicheroSiguiente = nombreFichero + ".backup" + n;
                        string ficheroSiguiente = Path.Combine(directorioBackup,
                                                  nombreFicheroSiguiente);
                        if (File.Exists(ficheroSiguiente)) File.Delete(ficheroSiguiente);
                        File.Move(ficheroActual, ficheroSiguiente);
                    }
                }
            }
            else
            {
                Directory.CreateDirectory(directorioBackup);
            }
            nombreFichero += ".backup1";
            string ficheroBackup = Path.Combine(directorioBackup, nombreFichero);
            File.Copy(rutaFichero, ficheroBackup);
        }
        catch
        {
            return false;
        }
    }
    return true;
}
```

Figura 83. Código fuente del método *hacerBackup*.

Así, al grabar un archivo sobre la versión anterior de sí mismo, la aplicación, por medio del método *hacerBackup* lo que hace en realidad es mover el archivo viejo a la carpeta *PNPWork_Backup* (el nombre de la carpeta también es un parámetro modificable por el usuario) para grabar el archivo deseado sin perder el anterior. Al mover el archivo viejo le añade la extensión *.backup*, pero si en la carpeta de copias de seguridad ya existía ese nombre, renombra antes al anterior con la extensión .backup1. Si al renombrar un archivo anterior también existía, antes renombra el que va a ser sustituido por .backup2 y así sucesivamente, hasta llegar a un nombre no usado o al límite de copias de un mismo archivo mencionado anteriormente, en cuyo caso lo borra.

De esta forma, cada vez que el usuario selecciona grabar el archivo de trabajo, en la carpeta de copias de seguridad se va creando una sucesión de archivos con el nombre del trabajo actual y las extensiones .backup, .backup1, .backup2, etc. En este esquema de nombres, en el archivo .backup1 tendremos siempre la última copia, en el .backup2 la penúltima, y así, mientras mayor es el número, más vieja será la copia. Esto facilita la labor de buscar un archivo sobrescrito por error, en caso de que sea necesario, incluso si se ha hecho varias veces.

Si la carpeta de copias de seguridad no existía, el método *hacerBackup* se encarga de crearla. Las copias de seguridad de los archivos de geometrías siguen exactamente el mismo patrón que los archivos de datos de montaje.

5.7. Estructuras de datos

5.7.1. Introducción

La aplicación que se ha desarrollado debe albergar una gran cantidad de datos en memoria, debido a todas las variables posibles que afectan a cada componente presente en cada placa y a cada placa de que conste la operación de montaje que se vaya a preparar. En este apartado se describirán todos estos datos, los criterios tenidos en cuenta y la solución adoptada.

Todos los datos se han almacenado en matrices unidimensionales (también llamadas *arrays* unidimensionales o vectores), en los que cada elemento de la matriz es una estructura de datos creada para albergar las diferentes características que representan al objeto que se trata.

5.7.2. Ejes de referencia

Se ha decidido tomar como ejes de referencia para todas las coordenadas guardadas en las estructuras de datos de la aplicación el que se muestra en la figura 84. Este sistema no es el mismo que utiliza la máquina Cosy, lo que nos obliga a realizar la conversión a las coordenadas de la máquina durante la producción. Sin embargo, se ha elegido así porque será mucho más intuitivo para el operador que el eje x crezca hacia la derecha en lugar de hacia la izquierda (tal como se vio anteriormente que hace la Cosy), puesto que prácticamente todos los sistemas de coordenadas que estamos acostumbrados a ver usan este criterio y ayudará al operador a cometer menos errores.

En cuanto al eje y, se ha dejado en la misma dirección que la usada por la máquina porque es muy común usar esta dirección al dibujar en la pantalla de ordenador. De esta manera, al coincidir los ejes elegidos con los ejes de la pantalla, se facilitan los cálculos necesarios para la representación gráfica.

Este sistema de referencia se ha usado tal como se muestra para definir la posición de las placas en el panel de la máquina y también la posición de cada componente en la placa. De esta forma, la esquina superior izquierda (en vista cenital y antes de cualquier rotación) será el punto de referencia que se tome como origen de coordenadas.

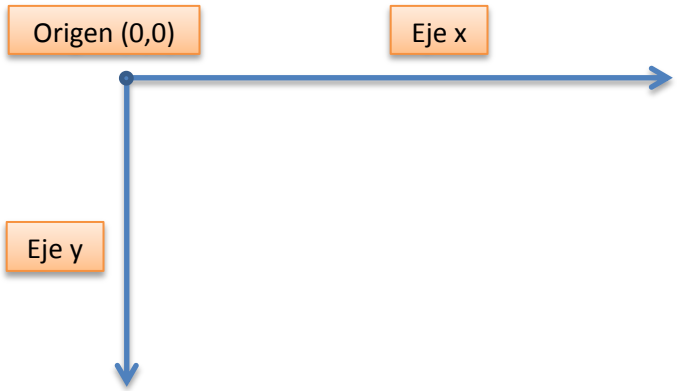


Figura 84. Ejes de coordenadas x, y usados para almacenar los datos.

Aunque inicialmente se usó también la esquina superior izquierda como punto de referencia para los componentes, posteriormente se tomó la decisión de hacer una pequeña variación y tomar como punto de referencia para los componentes el centro geométrico de los mismos.

Esta decisión se tomó motivada por dos factores. El factor más importante fue que normalmente la aguja debe capturar el componente desde su alimentador aproximadamente por el centro y las operaciones de centrado en la máquina siempre se hacen para que la aguja quede en el centro geométrico del componente. Además, al analizar los ficheros de salida de Altium se comprobó que proporcionaba las coordenadas del centro del componente, las del primer terminal y las del punto de referencia de Altium (que puede ser una ubicación diferente en distintos componentes). Esto hace que las coordenadas centrales sean las más apropiadas.

El segundo factor es que al rotar un componente desde su centro, no se produce desplazamiento del punto de colocación del componente, mientras que si se rota teniendo la referencia en la esquina, hay que calcular el desplazamiento desde la referencia. Por tanto, tomando el centro como referencia se simplifican los cálculos para la colocación del componente con cualquier rotación.

5.7.3. Estructura de placas

Como la aplicación puede manejar varias placas simultáneamente, será necesario almacenar metódicamente algunos de sus parámetros característicos, que son relevantes para el

montaje. Para guardar estos parámetros se ha definido una estructura que se puede ver en la figura 86.

En las primeras versiones se identificaban las placas por su número de orden en el vector en que se almacenaban, pero posteriormente se añadió espacio para guardar un nombre en la estructura de datos de la placa. De esta forma se facilita la identificación de cada placa al usuario, que puede ser importante si se van a manejar varias placas distintas simultáneamente.

La posición de los componentes en la placa será descrita de forma relativa a las coordenadas de origen de la placa. Esto se ha hecho así para que los datos del montaje de los componentes de cada placa sean independientes de la posición donde se ubique ésta en el panel en la máquina. De esta forma, se podrá cambiar una placa de sitio, por ejemplo al importarla desde otro trabajo anterior para incorporarla a una nueva composición, sin cambiar los datos de colocación de los componentes que incorpora.

Por tanto, necesitaremos almacenar la posición absoluta del panel donde se encuentran estas coordenadas de referencia de la placa. Estas coordenadas de referencia se tomarán siempre como las de la esquina superior izquierda de la placa.

También será útil guardar las dimensiones de la placa, pues así la representación gráfica será más ajustada a la placa real. Además se podrá detectar fácilmente y de manera automática algunos errores simples en la posición de algún componente que lo hagan caer fuera del área de la placa.

El programa creado no limita el número total de placas con las que se puede trabajar simultáneamente. El número de placas con las que puede trabajar nuestra aplicación está limitado únicamente por el área disponible en la zona de trabajo de la máquina y el tamaño que ocupan las placas. Por tanto, es posible que en una composición de placas sobre el panel queden huecos, que se pueden aprovechar ocupándolos con otras placas. Esto puede dar lugar a placas giradas en el panel, con lo que la posición de sus componentes deberá recalcularse teniendo en cuenta el ángulo de giro de la placa. Para tener esto en cuenta es necesario almacenar también el ángulo de giro de la placa.

En la representación gráfica de la placa con sus componentes siempre se dibujará la placa sin girar, puesto que los datos de posición de un componente en su placa son relativos a ésta y no les afecta este ángulo cuando se van a dibujar en pantalla.

En realidad, el número máximo de placas con las que puede trabajar la aplicación estará limitado también por la cantidad de memoria disponible en el ordenador de control para albergar los datos relativos a éstas. Sin embargo, como se ha comprobado, analizando el espacio de memoria ocupado por la aplicación y sus datos, es más probable que se agote antes el espacio físico donde ubicar las placas que la memoria del ordenador.

Durante el desarrollo inicial de la aplicación se estudió el tipo de datos que sería más apropiado para almacenar los valores necesarios. Al ir extrayendo información del análisis hecho a la máquina Cosy, se hizo evidente que ésta sólo trabajaba con números enteros, siendo todas las medidas de posición expresadas en décimas de milímetro y los ángulos expresados como grados centesimales (gradianes) enteros. Por tanto, se tomó la decisión de almacenar sus datos también como enteros en nuestra aplicación (multiplicando por una potencia de 10 igual al número de decimales de precisión obtendremos el entero equivalente).

Al guardar el ángulo de giro de la placa como un entero, sufrimos un error de redondeo, que como máximo será de medio grado centesimal. Tras algunos análisis, se vio que en algunos casos este error podía ser significativo. Hay que tener en cuenta que el error de posición que produce el redondeo, que con medio grado pudiera parecer pequeño, empieza a ser significativo al alejarnos del vértice desde donde se mide el ángulo, como se puede ver en la figura 85.

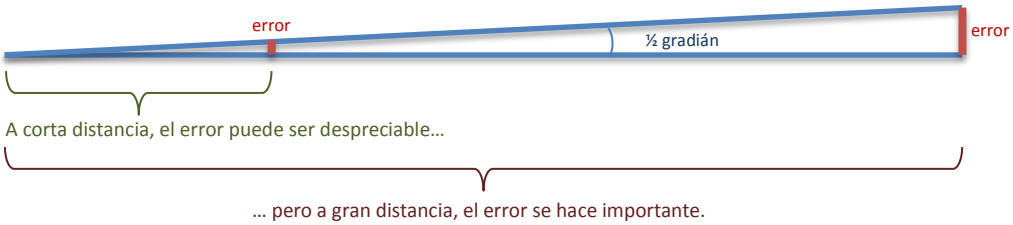


Figura 85. Efecto del redondeo del ángulo en el error de colocación (no a escala).

Para distancias del orden de unos pocos centímetros (el tamaño máximo de los componentes que se montarán normalmente con la máquina), el error máximo es comparable a la resolución de la máquina (una décima de milímetro), que es un error aceptable.

Sin embargo, para una placa de un tamaño cercano al máximo admisible del panel (algo más de treinta centímetros) el componente más alejado podría sufrir desviaciones en su posición de más de dos milímetros. Para el caso de los componentes más pequeños con los que la máquina puede trabajar, este error representa que el componente puede quedar completamente fuera de su ubicación en la placa, lo que no es aceptable.

Por tanto se tomó la decisión de no guardar el ángulo de la placa en un campo tipo *int* (redondeado al entero más cercano), sino que se ha almacenado en un campo de tipo *float*. En el caso de la Cosy necesitamos al menos dos decimales de precisión en el ángulo para que los errores sean despreciables, o lo que es los mismo, inferiores a la resolución de la máquina.

Además de los datos físicos de la placa, se ha decidido almacenar en la estructura de datos de cada placa una referencia al formulario que contiene su representación gráfica. Esto se ha hecho así porque la función para la representación gráfica puede ser llamada desde diversos formularios de la aplicación, pero no se desea crear un formulario nuevo cada vez para dibujar la placa, sino reutilizar el que ya está abierto con la representación gráfica de esa placa, en caso de que exista. Así evitamos que se muestren dos formularios iguales en la aplicación.

Una solución alternativa, si no se almacenase ninguna referencia a los formularios que se van creando, sería ir comprobando uno por uno los forms abiertos hasta ahora por la aplicación usando la propiedad *Application.OpenForms*, que es una colección que contiene referencias a todos ellos. Recorriendo esta colección, para ver a qué tipo pertenecen sus elementos, se podrían aislar los que pertenecen a la clase *formPlaca* y, usando una propiedad definida o un campo de acceso público, podríamos identificar a qué placa pertenecen.

Sin embargo, esta no es una solución mejor que almacenar una referencia en la estructura de la placa, puesto que es un método más lento (hay que recorrer un bucle comprobando los forms hasta encontrar el que se busca) y tampoco ahorra memoria, puesto que el código y los datos necesarios para hacer todas esas operaciones llegará a ocupar mucha más memoria que el dato extra guardado en la estructura de la placa.

A continuación tenemos el dato más importante que se guarda en la estructura de una placa, que es el conjunto de componentes que contiene. Todos los componentes que contiene la placa se guardan en una matriz unidimensional incluida en la estructura de la placa. Esta matriz constará de un elemento por cada componente, que será a su vez otra estructura, que albergará todos los datos relativos a la colocación de ese componente. Esta estructura para cada componente se describe con mayor detalle más adelante.

Los siguientes campos de la placa sirven para asignar a la placa sus archivos Gerber asociados. Con esta información se podrá generar una imagen de la placa para mostrar en pantalla detrás de los componentes, haciendo mucho más fácil su identificación visual. Los campos *overlay*, *solder* y *layer* están formados por una estructura que almacena la ruta del archivo Gerber correspondiente y el color con que se deberá dibujar. El campo

desplazamientoGerber es una estructura de tipo *Point* que guarda el offset que hay entre las coordenadas de los componentes en el archivo BOM y en la imagen Gerber correspondiente.

Ha sido necesario guardar este desplazamiento debido a que los archivos desde donde se toman los datos no guardan explícitamente unas coordenadas de origen común. El software desarrollado debe hacer una estimación y guardar la diferencia que pudiese existir para hacer una representación correcta.

```
public struct placa
{
    public string nombre;        // Identificador de la placa para el usuario.
    public int x;                // Coordenadas del vértice de referencia (esquina
    public int y;                // superior izquierda) expresadas en décimas de mm.
    public float t;              // Ángulo de rotación de la placa, levógiro y centesimal.
    public int anchoX;           // Tamaño horizontal de la placa.
    public int anchoY;           // Tamaño vertical de la placa.
    public FormPlaca form;       // Form para dibujar esta placa en pantalla.
    public componente[] componentes;     // Matriz de componentes contenidos en la placa.
    public datosGerber overlay;
    public datosGerber solder;
    public datosGerber layer;
    public Point desplazamientoGerber;
    public bool invertir;

    public void añadirComponente(componente comp)
    {
        componente[] c2 = new componente[this.componentes.Length + 1];
        this.componentes.CopyTo(c2, 0);
        c2[this.componentes.Length] = comp;
        this.componentes = (componente[])c2.Clone();
    }

    public void quitarComponente(int indice)
    {
        componente[] c2 = new componente[this.componentes.Length - 1];
        Array.Copy(this.componentes, c2, indice);
        Array.Copy(this.componentes, indice + 1, c2, indice,
                   this.componentes.Length - indice - 1);
        this.componentes = (componente[])c2.Clone();
    }

    public Point[] vértices()
    {
        // Devuelve las coordenadas de los vértices de la placa
        double a = t * Math.PI / 200;    // Convierte el ángulo a radianes para usar Math.
        Point punto1 = new Point(x, y);
        Point punto2 = new Point((int)(x + anchoX * Math.Cos(a)),
                         (int)(y - anchoX * Math.Sin(a)));
        Point punto3 = new Point((int)(x + anchoX * Math.Cos(a) + anchoY * Math.Sin(a)),
                         (int)(y - anchoX * Math.Sin(a) + anchoY * Math.Cos(a)));
        Point punto4 = new Point((int)(x + anchoY * Math.Sin(a)),
                         (int)(y + anchoY * Math.Cos(a)));
        Point[] vertices = { punto1, punto2, punto3, punto4 };
        return vertices;
    }
}
```

Figura 86. Definición de la estructura *placa*.

Por último, el campo invertir es útil en caso de que se esté trabajando con la cara inferior de la placa. En este caso, como sabemos, es necesario invertir las imágenes de la placa para verla correctamente.

En las primeras líneas del listado de la figura 86 se puede ver el código que define la estructura de datos de la placa que se ha descrito anteriormente. Sin embargo, el código completo de la estructura es mayor, puesto que se han incluido además tres métodos que operan con los datos de la placa.

Los dos primeros métodos se usan para añadir un componente al final o quitar un componente cualquiera de la matriz de componentes de esta placa, identificado mediante su índice. Es necesario implementar métodos que realicen estas tareas debido a que, las matrices en C# son de dimensiones fijas y, una vez creadas, no se pueden redimensionar conservando sus elementos.

El problema se ha resuelto, como se aprecia en el código expuesto, creando una matriz temporal con el nuevo tamaño (llamada c2 en el código), copiando los datos apropiados a la nueva matriz y devolviendo el resultado a la matriz original mediante una operación de clonación de la matriz temporal, con lo que la matriz original pasa a tener la nueva dimensión requerida.

El tercer método mostrado devuelve un vector con cuatro puntos (estructura *Point* de C#) que representan las coordenadas de los cuatro vértices de la placa en el panel de trabajo de la máquina, teniendo en cuenta su ángulo de rotación para el cálculo.

5.7.4. Estructura de componentes

Como se mencionó en el apartado anterior, los datos necesarios de cada componente se almacenan en una estructura (*struct*). Cada placa tiene asignado un vector de estructuras de componentes donde estarán incluidos todos los componentes de esa placa.

Siempre que hablamos de los componentes de una placa nos referimos a los que se van a tratar mediante nuestra aplicación. La placa puede necesitar otros componentes adicionales que no se montarán con la máquina con la que estamos trabajando, por lo que no aparecerán en los datos que maneja nuestro programa por ser irrelevantes para su propósito.

En la figura 87 se muestra el código de la estructura componente que define todos sus datos. La estructura incluye además unos métodos que operan con los datos del componente, tal como vimos anteriormente con la estructura de la placa.

Como es obvio, en la estructura de datos de un componente es necesario guardar la posición donde debe ser colocado y el ángulo de giro de éste. También se deberá especificar el alimentador desde donde debe ser cogido. Además, cada componente llevará asignado un nombre que servirá para que el usuario identifique fácilmente cada componente.

```
public struct componente
{
    public int x;               // Coordenadas del centro del componente,
    public int y;               // relativas al origen de la placa.
    public int t;               // Ángulo de giro del componente, centesimal y levógiro.
    public int alimentador;     // Índice del alimentador donde coger este componente.
    public string nombre;
    public estadoComponente estado;      // Estado de montaje del componente.

    public bool esVisible(int x0, int y0, int x1, int y1)
    {
        // Comprueba si el componente tiene algún vértice dentro del rectángulo horizontal
        // definido por sus esquinas superior izquierda (x0,y0) e inferior derecha (x1,y1).
        try
        {
            double a = t * Math.PI / 200;   // Convierte el ángulo a radianes.
            int w = datos.alimentadores[this.alimentador].anchoX;
            int h = datos.alimentadores[this.alimentador].anchoY;

            // Calcula las coordenadas de los vértices del componente.
            int xa = this.x;
            int ya = this.y;
            int xb = (int)(x + w * Math.Cos(a));
            int yb = (int)(y - w * Math.Sin(a));
            int xc = (int)(x + w * Math.Cos(a) + h * Math.Sin(a));
            int yc = (int)(y - w * Math.Sin(a) + h * Math.Cos(a));
            int xd = (int)(x + h * Math.Sin(a));
            int yd = (int)(y + h * Math.Cos(a));

            // Comprueba si alguno de los vértices está dentro del rectángulo.
            if (((xa > x0) && (xa < x1) && (ya > y0) && (ya < y1)) ||
                ((xb > x0) && (xb < x1) && (yb > y0) && (yb < y1)) ||
                ((xc > x0) && (xc < x1) && (yc > y0) && (yc < y1)) ||
                ((xd > x0) && (xd < x1) && (yd > y0) && (yd < y1)))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        catch
        {
            // En caso de que no exista el alimentador u otro error, devuelve false.
            return false;
        }
    }

    public Point[] vértices()
    {
        // Devuelve las coordenadas de los vértices del componente relativas a su centro.
        return vértices(0, 0);
    }

    public Point[] vértices(int x0, int y0)
    {
        // Devuelve las coordenadas de los vértices del componente desplazado a x0, y0.
        try
        {
            double a = t * Math.PI / 200;    // Convierte el ángulo a radianes.
            int w2cost = (int)(datos.alimentadores[alimentador].anchoX * 0.5 * Math.Cos(a));
            int w2sent = (int)(datos.alimentadores[alimentador].anchoX * 0.5 * Math.Sin(a));
            int h2cost = (int)(datos.alimentadores[alimentador].anchoY * 0.5 * Math.Cos(a));
            int h2sent = (int)(datos.alimentadores[alimentador].anchoY * 0.5 * Math.Sin(a));
            Point punto1 = new Point(x + x0 - w2cost - h2sent, y + y0 + w2sent - h2cost);
            Point punto2 = new Point(x + x0 + w2cost - h2sent, y + y0 - w2sent - h2cost);
            Point punto3 = new Point(x + x0 + w2cost + h2sent, y + y0 - w2sent + h2cost);
            Point punto4 = new Point(x + x0 - w2cost + h2sent, y + y0 + w2sent + h2cost);
            Point[] vertices = { punto1, punto2, punto3, punto4 };
            return vertices;
```

```
        }
        catch
        {
            // En caso de que no exista el alimentador o cualquier otro error, devuelve
            // un vector de coordenadas vacío.
            return new Point[4];
        }
    }
}
```

Figura 87. Definición de la estructura *componente*.

En este caso, todos los datos numéricos se almacenan en campos enteros por los motivos comentados anteriormente. El ángulo de giro también, puesto que, como se dijo antes, la máquina sólo acepta ángulos de giro enteros. Además, vimos que para los componentes más grandes, el error del redondeo produce un error en la colocación inferior al paso mínimo de movimiento de la máquina (y por tanto también inferior a sus tolerancias).

Por último, en esta estructura se almacenará el estado de montaje en que está actualmente este componente. El estado servirá principalmente para saber si este componente ya fue colocado con éxito o aún no lo ha sido. Si ya fue colocado, por ejemplo en una producción anterior interrumpida, al continuar la producción será ignorado. Si no ha sido colocado con éxito, se incluirá entre los componentes que hay que tratar.

Además, se utilizará también el estado durante la producción con la Cosy para representar gráficamente los componentes con diferentes colores, según el estado del montaje en el que estén. De esta manera el operador tendrá una referencia visual de qué componentes han sido colocados correctamente y cuáles han fallado.

El primer método, llamado *esVisible*, devuelve un valor booleano que indica si algún vértice del componente está dentro del rectángulo definido por sus parámetros de entrada. El segundo método (*vértices*) devuelve las coordenadas de los vértices del componente.

Este método se puede llamar de dos formas diferentes, por lo que tiene dos implementaciones. En la primera (sin parámetros) devuelve las coordenadas relativas a su propio centro. En la segunda implementación, devuelve las mismas coordenadas, pero relativas al origen proporcionado en los parámetros de entrada.

El estado en que está el componente tiene un tipo especial, *estadoComponente*, cuyos valores proceden de una enumeración, cuya definición se muestra en la figura 88. En las primeras versiones de nuestra aplicación se codificaba el estado como un entero, asignándole los valores del 0 al 5 según el siguiente criterio:

0 → Componente no tratado aún.

1 → El componente se está cogiendo o se están haciendo los preparativos previos para ello (posicionamiento, cambio de útil en la boquilla y/o cambio de banco de alimentadores).

2 → El componente se está colocando o se están haciendo los preparativos previos para ello (posicionamiento y/o centrado).

3 → Componente colocado con éxito.

4 → Componente no colocado por error al cogerlo del alimentador.

5 → Componente no colocado por error al colocarlo sobre la placa.

Usando la enumeración los valores numéricos asignados serán los mismos, pero hace innecesario recordar los códigos numéricos de cada estado mientras se desarrolla el programa ni al editar manualmente los datos de un componente. Esto redunda en un código más claro y menos propenso a errores [**35**].

```
public enum estadoComponente
{
    No_Tratado,
    Cogiendo,
    Colocando,
    Colocado,
    Error_Cogiendo,
    Error_Colocando
}
```

Figura 88. Definición de la enumeración de posibles estados de un componente.

En la estructura de datos del alimentador será donde se especifiquen el resto de las características relacionadas con el tratamiento de este componente, puesto que serán comunes a todos los componentes que se tomen del mismo alimentador.

5.7.5. Estructura de alimentadores

La mayoría de los datos almacenados en las estructura de alimentadores se usan en el montaje de placas mediante la Cosy, por lo que están orientados a las necesidades de esta máquina. El montaje manual sólo requiere de algunos datos básicos sobre los alimentadores (el tipo de componente y sus dimensiones) puesto que será el operador el que se encargue de cogerlo y colocarlo en su posición. En el caso de montaje con las máquinas Philips / Assembleon, la configuración de alimentadores se suele hacer sobre la propia máquina, por lo que tampoco se usará aquí la información extra almacenada.

La estructura donde se guardan las características de un alimentador contendrá las coordenadas donde se debe coger el componente, así como el banco en que se encuentra el

alimentador, para poder desplazar la bandeja de alimentadores a la posición correcta antes de llevar el cabezal hasta la posición de captura.

Otros datos necesarios para un manejo físico correcto del componente es la altura a la que se debe posicionar la aguja de succión para cogerlo, transportarlo y colocarlo, así como el útil que hay que adaptar a la aguja a modo de boquilla para garantizar que el componente se adhiere con suficiente fuerza para no descolocarse mientras se maneja.

Además, es necesario especificar la velocidad a la que se debe manejar este componente en cada uno de los cuatro ejes de movimiento de la máquina (*x*, *y*, *z*, ángulo de giro) para que su manejo sea suficientemente ágil, pero evitando que se descoloque en la boquilla o que se caiga debido a un movimiento excesivamente rápido. Estos parámetros dependerán en cada caso de las dimensiones, forma y peso del componente.

Debido a las características del componente, puede ser necesario centrarlo en cualquier combinación de las tres posibilidades que ofrece la máquina. Estas son: centrado en el eje *x* con las pinzas que tiene la boquilla, centrado en el eje *y* con las mismas pinzas (antes lo gira 90°) y centrado en ambos ejes llevándolo a la estación de centrado.

Adicionalmente, en los casos en que sea necesario, hay que especificar cuántas gotas de adhesivo se deben dispensar para este componente y si es necesario activar el avance de componente del alimentador bajando el imán para excitar el sensor de efecto Hall visto anteriormente.

También se incluyen en esta estructura las dimensiones del componente, usadas principalmente para dibujar el componente en pantalla, pero también usadas en la detección de posibles errores (un componente que sobresale fuera de la placa). En la figura 89 se puede ver el código de esta estructura que describe todos los datos mencionados.

Cuando se importa una placa desde un fichero de Altium, muchas de las características de sus componentes, que se deben añadir a los datos de su alimentador correspondiente, están ausentes. Por tanto, al crear el alimentador para éste componente de forma automática no se conocen todos los datos y hay que rellenar estos datos temporalmente con los valores típicos más comunes. Más tarde, el operador editará los alimentadores y corregirá los valores necesarios.

En caso de que no se vaya a montar con la Cosy, no es necesario cambiar los valores por omisión que la aplicación introduce.

```
public struct alimentador
{
        public string tipoComponente;   // Tipo de componente que hay en el alimentador.
        public int x;                   // Coordenadas en las que hay que posicionar.
        public int y;                   // la aguja para coger el componente.
        public int banco;               // Banco en el que se encuentra el alimentador.
        public int anchoX;              // Ancho y largo del componente .
        public int anchoY;              //
        public int útil;                // Útil con el que hay que coger el componente.
        public int zPick;               // Altura a la que hay que coger el componente.
        public int zMover;              // Altura a la que hay que transportar el comp.
        public int zPlace;              // Altura a la que hay que soltar el componente.
        public int velocidadX;          // Velocidades a las que hay que manipular el comp.
        public int velocidadY;          // en cada eje.
        public int velocidadZ;          //
        public int velocidadA;          //
        public int gotasDispensador;    // Nº de gotas de adherente.
        public bool avanceAlimentador;  // El alimentador requiere avance.
        public bool centradoEstación;   // El componente requiere centrado en estación.
        public bool centradoPinzasX;    // El componente requiere centrado en las pinzas.
        public bool centradoPinzasY;    // El comp. requiere centrado en las pinzas a 90°.

        public alimentador(string tipoComponente, int x, int y, int anchox, int anchoy,
                        int zPickPlace)
        {
            this.tipoComponente = tipoComponente;
            this.x = x;
            this.y = y;
            this.banco = 1;
            this.anchoX = anchox;
            this.anchoY = anchoy;
            this.útil = 0;
            this.zPick = zPickPlace;
            this.zMover = 15;
            this.zPlace = zPickPlace;
            this.velocidadX = datos.velocidadMaxX;
            this.velocidadY = datos.velocidadMaxY;
            this.velocidadZ = datos.velocidadMaxZ;
            this.velocidadA = datos.velocidadMaxA;
            this.gotasDispensador = 0;
            this.avanceAlimentador = false;
            this.centradoEstación = false;
            this.centradoPinzasX = false;
            this.centradoPinzasY = false;
        }
}
```

Figura 89. Definición de la estructura *alimentador*.

Para realizar esta tarea se ha añadido al código de la estructura de alimentadores un constructor que inicializa la mayor parte de los campos a los valores más comunes. Como se puede ver en la figura 89, este constructor recibe como parámetros el nombre del alimentador, las coordenadas donde se encuentra y la altura a la que debe ser cogido y colocado el componente (típicamente se usa la misma, puesto que la boquilla del cabezal es retráctil y tiene un muelle para adaptarse a pequeñas diferencias entre la altura del alimentador y de la placa).

Las velocidades de los diferentes movimientos para este componente se toman de los valores máximos de trabajo predeterminados en la aplicación, mientras que el resto de valores se inicializan a los valores más comunes.

5.7.6. Estructura de geometrías

La estructura de geometrías sólo se usa en caso de que se quiera importar un fichero BOM de Altium. Durante la importación, se obtienen a partir del fichero de Altium los datos de colocación necesarios para el componente. Sin embargo, este tipo de ficheros adolece de la falta de especificaciones físicas básicas sobre los componentes.

Para solventar este problema, típicamente se suele crear un archivo .CSV (Comma-Separated Values [**36**]) que consiste en una serie de valores que representan una base de datos. Estos valores están separados por comas para representar los distintos campos de un registro y por saltos de línea, para representar en cada línea un registro diferente. Es común crearlos desde una hoja de cálculo, grabando la tabla finalmente con este formato.

Nuestra aplicación puede leer un archivo .CSV para crear o completar la estructura de geometrías en memoria, pero también contiene un editor interno que nos permite crear o completar esta estructura en cualquier momento y sin necesidad de usar otros programas externos. Además, la aplicación puede grabar los archivos .CSV de geometrías para ser editados con un editor externo o cargados de nuevo en otra ocasión.

Adicionalmente, la aplicación puede importar el archivo de geometrías usado habitualmente en el laboratorio para trabajar con el programa Cad2cad (packages.dim) e introducir los datos en su estructura de geometrías. Cad2cad es una herramienta proporcionada por Philips a sus clientes para preparar la producción con sus máquinas pick & place automáticas.

La configuración predeterminada al instalar el programa o ejecutarlo por primera vez (se puede cambiar en el menú *Configuración*) es que automáticamente cargue y grabe estos archivos sin la intervención del operador, lo que le facilitará la tarea en el trabajo diario con la aplicación al estar disponibles siempre los datos de geometrías creados en sesiones anteriores y sólo tener que introducir los datos de los componentes que no han sido usados anteriormente.

Con la configuración predeterminada siempre se cargará el último archivo de geometrías que se usó al iniciar el programa y se grabará con el mismo nombre al cerrar el programa. Por tanto el operador deberá tenerlo en cuenta, si no quisiera sobrescribir el archivo preexistente con los nuevos datos que haya podido añadir durante la sesión de trabajo. En ese caso puede grabarlo en un archivo diferente o desmarcar en el menú de configuración la opción *Cargar / guardar geometrías automáticamente*.

En memoria, los datos de geometrías de componentes están representados mediante un vector de elementos, con un elemento para cada geometría almacenada. Este elemento consiste en una estructura de datos al igual que otros datos expuestos en apartados anteriores.

La estructura de geometrías es muy simple, puesto que únicamente contiene un nombre para cada geometría y las dimensiones de ancho, largo y alto del componente. En la figura 90 se puede ver el código de la estructura que define cada elemento de la geometría.

```
public struct geometría
{
        public string nombre;   // Nombre de la geometría del componente
        public int dimX;        // Tamaño del componente en la dimensión x
        public int dimY;        // Tamaño del componente en la dimensión y
        public int dimZ;        // Alto del componente

        public geometría(string nombre, int dimX, int dimY, int dimZ)
        {
                this.nombre = nombre;
                this.dimX = dimX;
                this.dimY = dimY;
                this.dimZ = dimZ;
        }
}
```

Figura 90. Definición de campos de la estructura geometría.

Como se puede apreciar, en el código de la estructura se ha incluido un constructor básico que permite inicializar todos los datos de la estructura en el momento de crear una instancia en memoria.

5.8. Formato de los archivos

5.8.1. Archivos de datos

Para almacenar los datos con los que trabaja la aplicación, se ha creado un formato de archivos propio. Los datos se guardan en el archivo como cadenas de texto, separadas por un carácter delimitador que nos permite distinguir entre un campo y el siguiente, de forma similar al formato de archivos CSV.

Las primeras versiones del programa grababan archivos separados para alimentadores y placas (con sus componentes), pero pronto se vio la necesidad de que unos datos fuesen acompañados por los otros. Conforme se iba desarrollando la aplicación se iba produciendo la necesidad o conveniencia de introducir cambios en el formato de archivo, por lo que se fue añadiendo un número de versión al archivo para ir siguiendo estos cambios y evitar equivocaciones con los archivos de pruebas. El número de versión del formato definitivo de los archivos de la aplicación es 1.9.

Aunque todos los datos se graban en un único archivo, se decidió dejar la posibilidad de anexar a las estructuras de datos presentes en memoria sólo una parte de los datos guardados en un archivo. De esta forma se puede, por ejemplo, cargar en memoria los alimentadores creados para un trabajo anterior, pero no sus placas. Así, se puede ahorrar mucho trabajo de preparación previa del montaje, si los alimentadores reutilizados van a ser los mismos.

Los archivos que graba la aplicación están pensados para que se puedan abrir con una aplicación externa para efectuar cambios, comprobaciones o incluir comentarios por parte del usuario. Se pueden abrir con un editor de texto, pero lo ideal para ver claramente separados los datos por columnas es un programa de hoja de cálculo, como se puede ver en el ejemplo de la figura 91.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Fichero de datos de alimentadores y placas v1.9															
2	Creado el 28/05/2016 16:33:51															
3																
4	Tipo	Nº	Componente	x	y	Banco	Tamaño x	Tamaño y	Útil	z Pick	z Mover	z Place	Velocidad x	Velocidad y	Velocidad z	Velocidad giro
5	Alimentador	1	1K (2010)	500	2600	1	20	10	0	170	15	170	9	9	10	10
6	Alimentador	2	47nF (2515)	700	2600	1	25	15	1	170	15	170	9	9	10	10
7	Alimentador	3	100nF (3015)	900	2600	1	30	15	2	170	15	170	9	9	10	10
8	Alimentador	4	22nF (2515)	1200	2600	1	20	35	3	170	15	170	9	9	10	10
9	Alimentador	5	EEPROM (SOP80)	1400	2600	1	80	80	2	170	15	170	8	8	9	9
10	Alimentador	6	ROM (JLEAD150)	1600	2600	2	150	150	4	170	15	170	6	6	9	9
11	Alimentador	7	RAM (QFP170)	1800	2600	1	300	300	4	170	15	170	5	5	8	8
12																
13	Tipo	Nº	Nombre	x	y	Ángulo	Tamaño x	Tamaño y								
14	Placa	1	Prueba compleja	400	100	0	800	600								
15	Tipo	Nº	Nombre	Alimentador	x	y	Ángulo	Estado								
16	Componente	1	R1	1	20	25	0	3								
17	Componente	2	C2	2	107	67	100	3								
18	Componente	3	C3	3	53	201	49	3								
19	Componente	4	R3	1	20	45	0	3								
20	Componente	5	R4	1	20	65	0	3								
21	Componente	6	R5	1	15	90	100	3								
22	Componente	7	R2	1	25	290	100	2								
23	Componente	8	C4	2	127	67	100	1								
24	Componente	9	C5	2	147	67	100	4								
25	Componente	10	R6	1	33	90	100	5								
26	Componente	11	R7	1	400	25	0	0								
27	Componente	12	R8	1	186	143	50	0								
28	Componente	13	R9	1	198	155	50	0								
29	Componente	14	R10	1	209	167	50	0								
30	Componente	15	R11	1	220	179	50	0								
31	Componente	16	C6	2	111	141	150	0								
32	Componente	17	C7	2	95	157	150	0								
33	Componente	18	C8	2	79	173	150	0								
34	Componente	19	IC1	5	267	72	0	0								
35	Componente	20	IC2	6	149	241	50	0								
36	Componente	21	IC3	7	390	354	50	0								

Figura 91. Fichero de datos de alimentadores y placas abierto con una hoja de cálculo.

El separador predeterminado es el punto y coma, porque la aplicación de hoja de cálculo utilizada para las pruebas iniciales en los ordenadores de desarrollo utilizaba este carácter como separador al grabar sus archivos CSV. De esta forma se lograba un reconocimiento inmediato de los archivos simplemente asociando la extensión de nuestros archivos para que se abrieran con dicha aplicación de hoja de cálculo al hacer doble clic sobre ellos y así se facilitaba la depuración.

El separador puede ser cambiado por el usuario, para adaptarlo al más conveniente en su caso, puesto que se encuentra grabado en las propiedades de configuración de la aplicación. Así,

podría usar tabuladores, con los que se vería mejor si se abriera con un editor de texto, o coma, que es el separador comúnmente utilizado para un archivo CSV.

Como se puede ver, el archivo comienza siempre con una identificación del tipo de archivo en la primera línea y la fecha y hora de la creación en la segunda línea y una tercera línea en blanco. A continuación se graba un comentario que sirve de encabezamiento de los alimentadores para las columnas en caso de importar el archivo como archivo CSV en una aplicación de hoja de cálculo, seguida de todos los alimentadores presentes en memoria.

Tras otra línea en blanco, se incluyen todas las placas, separadas a su vez por líneas en blanco. Cada placa consta de tres secciones: una línea con los datos de la placa, un grupo de líneas con los datos de cada componente de esa placa y otro grupo de líneas con los datos de la estructura de placas que asocian los archivos Gerber (rutas, colores, desplazamiento e inversión). Antes de los datos de la placa y de los componentes se incluye una línea de comentario con los nombres de las columnas de datos, al igual que en los alimentadores.

El formato de archivo creado soporta que el usuario escriba comentarios en él, puesto que interpreta sólo aquellas líneas que comiencen por una de las palabras "clave". De esta forma, está permitido hacer anotaciones de cualquier clase, siempre que ninguna línea comience con una de estas palabras clave. En caso de que se deseen hacer anotaciones, es recomendable, aunque no obligatorio, empezar cada línea con algún símbolo (por ejemplo #), que servirá tanto para que el usuario reconozca rápidamente qué líneas ha añadido como para evitar que la aplicación trate de interpretarla como datos.

Las palabras clave elegidas para empezar las líneas de datos son:

- *Alimentador* → La línea contendrá los datos de un alimentador.
- *Placa* → La línea contendrá los datos de una placa (pero no los de sus componentes).
- *Componente* → La línea contendrá los datos de un componente que pertenece a la última línea tipo "*Placa*" leída hasta el momento (placa actual).
- *Overlay* → La línea contendrá la ruta del archivo Gerber y el color de la capa de serigrafía de la placa actual.
- *Solder* → La línea contendrá ruta Gerber y color de la capa de estañado.
- *Layer* → La línea contendrá ruta Gerber y color de la capa de pistas.
- *Desplazamiento* → La línea contendrá el offset a aplicar a la imagen Gerber.
- *Invertir* → La línea indica si esta placa se muestra invertida.

En los casos *Alimentador*, *Placa* o *Componente*, el siguiente campo de la línea será siempre un número de orden, que la aplicación ignorará al leerlo, pero se guarda en el archivo para facilitar la lectura al usuario. El número de orden coincide con el número de orden mostrado en los editores de alimentadores, placas y componentes.

El tercer campo es el primero que contiene datos de la aplicación y será siempre el nombre del alimentador, placa o componente. El resto de datos se guardan también aproximadamente en el mismo orden en que se muestran en los formularios de los editores. De esta forma el usuario que abra un archivo para ver su contenido no tendrá problemas para localizar la información que desea.

De esta forma los campos de una línea de tipo “Alimentador” serán los siguientes:

1. Palabra clave “Alimentador”
2. Número de orden
3. Tipo de componente (nombre)
4. Coordenada x
5. Coordenada y
6. Banco
7. Tamaño en la dirección x
8. Tamaño en la dirección y
9. Útil
10. Altura para coger
11. Altura para mover
12. Altura para colocar
13. Velocidad en el eje x
14. Velocidad en el eje y
15. Velocidad en el eje z
16. Velocidad de giro
17. Número de gotas del dispensador de adhesivo
18. Avance del alimentador
19. Centrado en la estación de centrado
20. Centrado con las pinzas en el eje x
21. Centrado con las pinzas en el eje y

Los campos de una línea de tipo “Placa” son los que se muestran a continuación:

1. Palabra clave “Placa”
2. Número de orden
3. Nombre de la placa
4. Coordenada x del origen
5. Coordenada y del origen
6. Ángulo de giro
7. Tamaño en la dirección x
8. Tamaño en la dirección y

Por último, los campos de una línea de tipo “Componente” son los siguientes:

1. Palabra clave “Componente”
2. Número de orden
3. Nombre del componente
4. Número de alimentador
5. Coordenada x de colocación sobre la placa
6. Coordenada y de colocación sobre la placa
7. Ángulo de giro
8. Estado de montaje

Hay que tener en cuenta un detalle, ya que los números de orden mostrados en pantalla no coinciden con los números de índice de las matrices donde se guardan los datos. Las matrices siempre empiezan con el índice 0, pero para un aspecto más normal, en pantalla los números de orden siempre empiezan por uno. Para ello siempre se suma 1 al índice de la matriz cuando se va a mostrar en pantalla.

Al guardar un archivo también se suma 1 al número de alimentador para que así coincida con el número de orden del alimentador grabado. Esto se tiene en cuenta al cargar el archivo de nuevo, restándole 1 al número de alimentador del componente, para que vuelva a coincidir con el índice correcto de la matriz de alimentadores.

En la figura 92 se muestra el código fuente del método *grabarArchivo*, que muestra todas las operaciones para crear el formato de archivo que se ha explicado anteriormente. Como se puede ver, lo primero que se hace antes de grabar el archivo de datos es realizar una copia de seguridad automática si ya existía en disco un archivo con el mismo nombre. De esta forma será posible volver a una versión anterior del archivo, buscando la copia en la carpeta de *backups.*

```
public static void guardarFichero(string rutaFichero)
{
    // Graba el fichero de placas y alimentadores v1.9
    hacerBackup(rutaFichero);
    FileInfo fi = new FileInfo(rutaFichero);
    StreamWriter sw = fi.CreateText();
    if (sw != null)
    {
        // Primero se graba la cabecera del fichero:
        sw.WriteLine("Fichero de datos de alimentadores y placas v1.9");
        sw.WriteLine("Creado el " + Convert.ToString(DateTime.Now));
        sw.WriteLine();

        // A continuación se graba un comentario como cabecera de los alimentadores:
        sw.WriteLine("Tipo" + separador + "Nº" + separador
                + "Componente" + separador
                + "x" + separador
                + "y" + separador
                + "Banco" + separador
                + "Tamaño x" + separador
                + "Tamaño y" + separador
                + "Útil" + separador
                + "z Pick" + separador
                + "z Mover" + separador
                + "z Place" + separador
                + "Velocidad x" + separador
                + "Velocidad y" + separador
                + "Velocidad z" + separador
                + "Velocidad giro" + separador
                + "Gotas Dispensador" + separador
                + "Avance Alimentador" + separador
                + "Centrado Estación" + separador
                + "Centrado Pinzas x" + separador
                + "Centrado Pinzas y");

        // A continuación se graban los alimentadores:
        for (int n = 0; n < alimentadores.Length; n++)
        {
            sw.WriteLine(marcaAlimentador + separador + (n + 1).ToString() + separador
                    + alimentadores[n].tipoComponente + separador
                    + alimentadores[n].x + separador
                    + alimentadores[n].y + separador
                    + alimentadores[n].banco + separador
                    + alimentadores[n].anchoX + separador
                    + alimentadores[n].anchoY + separador
                    + alimentadores[n].útil + separador
                    + alimentadores[n].zPick + separador
                    + alimentadores[n].zMover + separador
                    + alimentadores[n].zPlace + separador
                    + alimentadores[n].velocidadX + separador
                    + alimentadores[n].velocidadY + separador
                    + alimentadores[n].velocidadZ + separador
                    + alimentadores[n].velocidadA + separador
                    + alimentadores[n].gotasDispensador + separador
                    + alimentadores[n].avanceAlimentador + separador
                    + alimentadores[n].centradoEstación + separador
                    + alimentadores[n].centradoPinzasX + separador
                    + alimentadores[n].centradoPinzasY);
        }
        sw.WriteLine();

        // A continuación se graban las placas con sus componentes:
        for (int n = 0; n < datos.placas.Length; n++)
        {
            // Antes de cada placa se graba un comentario como cabecera:
            sw.WriteLine("Tipo" + separador + "Nº" + separador
                    + "Nombre" + separador
                    + "x" + separador
                    + "y" + separador
                    + "Ángulo" + separador
                    + "Tamaño x" + separador
                    + "Tamaño y");
            sw.WriteLine(marcaPlaca + separador + (n + 1).ToString() + separador
```

```
                    + datos.placas[n].nombre + separador
                    + datos.placas[n].x + separador
                    + datos.placas[n].y + separador
                    + datos.placas[n].t + separador
                    + datos.placas[n].anchoX + separador
                    + datos.placas[n].anchoY);

            // Antes de los componentes se graba un comentario como cabecera:
            sw.WriteLine("Tipo" + separador + "Nº" + separador
                    + "Nombre" + separador
                    + "Alimentador" + separador
                    + "x" + separador
                    + "y" + separador
                    + "Ángulo" + separador
                    + "Estado");
            for (int i = 0; i < datos.placas[n].componentes.Length; i++)
            {
                sw.WriteLine(marcaComponente + separador + (i + 1).ToString() + separador
                    + datos.placas[n].componentes[i].nombre + separador
                    + (datos.placas[n].componentes[i].alimentador + 1).ToString()+ separador
                    + datos.placas[n].componentes[i].x + separador
                    + datos.placas[n].componentes[i].y + separador
                    + datos.placas[n].componentes[i].t + separador
                    + (int)datos.placas[n].componentes[i].estado);
            }

            //A continuación se graban los datos Gerber:
            if (File.Exists(datos.placas[n].overlay.fichero))
            {
                sw.WriteLine(marcaOverlay + separador
                        + datos.placas[n].overlay.fichero + separador
                        + datos.placas[n].overlay.color.ToArgb() + separador);
            }
            if (File.Exists(datos.placas[n].solder.fichero))
            {
                sw.WriteLine(marcaSolder + separador
                        + datos.placas[n].solder.fichero + separador
                        + datos.placas[n].solder.color.ToArgb());
            }
            if (File.Exists(datos.placas[n].layer.fichero))
            {
                sw.WriteLine(marcaLayer + separador
                        + datos.placas[n].layer.fichero + separador
                        + datos.placas[n].layer.color.ToArgb());
            }
            sw.WriteLine(marcaDesplazamiento + separador
                        + datos.placas[n].desplazamientoGerber.X + separador
                        + datos.placas[n].desplazamientoGerber.Y);
            sw.WriteLine(marcaInvertir + separador
                        + datos.placas[n].invertir);
            sw.WriteLine();
        }
        sw.Close();
        sw.Dispose();
        modificados = false;
    }
}
```

Figura 92. Código fuente del método *grabarArchivo*.

De forma predeterminada (aunque es un valor configurable), el número máximo de copias de seguridad de un archivo será de diez, por lo que tras grabar el archivo sucesivamente con el mismo nombre, se podría recuperar la versión que había hasta diez grabaciones antes.

En la figura 94 se muestra el código del método *cargarArchivo*. Este método está sobrecargado, incluyendo así la posibilidad de cargar y actualizar todo, o elegir qué partes del

archivo cargar y si se actualizan los números de alimentador en los componentes. Si al llamarlo se le indica únicamente el nombre del archivo a cargar, se cargarán y actualizarán todos los datos del archivo.

Si se especifica el parámetro *procesarAlimentadores* con valor *true*, se cargarán todos los alimentadores que se encuentren en el archivo.

Si se especifican el parámetro *procesarPlacas* con el valor *true*, se cargarán todas las placas que se encuentren en el archivo, pero sus componentes sólo se cargarán también si se especifica el parámetro *procesarComponentes* con el valor *true*. En caso contrario, las placas se cargarán, pero vacías de componentes.

Si se especifica que no se actualicen los números de alimentador en los componentes que se carguen, poniendo el parámetro *actualizarAlimentadores* a *false*, se dejarán los mismos números de índice que se lean. Esto puede ser útil para cargar una placa, pero no con los alimentadores con que fue grabada, sino con otra versión de éstos existente en memoria o en otro archivo diferente.

La función *limitar*, mostrada en la figura 93 se asegura de que los valores que se cargan no superan en ningún caso los valores máximos establecidos por la máquina que se va a utilizar. En el caso de los alimentadores, algunos parámetros son específicos de la Cosy, como vimos en la descripción de su estructura de datos, por lo que se limitan a los valores máximos de ésta directamente.

```
public static int limitar(int valor, int mínimo, int máximo)
{
    return Math.Max(mínimo, Math.Min(máximo, valor));
}

public static float limitar(float valor, float mínimo, float máximo)
{
    return Math.Max(mínimo, Math.Min(máximo, valor));
}

public static decimal limitar(decimal valor, decimal mínimo, decimal máximo)
{
    return Math.Max(mínimo, Math.Min(máximo, valor));
}
```

Figura 93. Código fuente de la función *limitar*.

Como se puede ver, esta función está sobrecargada para aceptar los diferentes tipos de datos con los que se va a usar a lo largo de todo el programa.

```
public static int cargarFichero(string rutaFichero)
{
  return cargarFichero(rutaFichero, true, true, true, true);
}

public static int cargarFichero(string rutaFichero, bool procesarAlimentadores,
                bool procesarPlacas, bool procesarComponentes, bool actualizarAlimentadores)
{
  // Carga el fichero de placas y alimentadores v1.7
  FileInfo fi = new FileInfo(rutaFichero);
  int error = 0;        // Variable para guardar el código de error. 0 = Ok
  if (fi.Exists)
  {
    StreamReader sr = fi.OpenText();  // Abre el fichero como texto.
    string linea;
    int numLinea = 0;
    int numAlimentadores = alimentadores.Length;
    while ((!sr.EndOfStream) & (error == 0))
    {
      try
      {
        // Lee una línea y separa sus elementos en un vector para interpretarlos.
        numLinea++;
        linea = sr.ReadLine();
        char[] division = { separador };
        string[] elementos = new string[50];
        elementos = linea.Split(division, 50, StringSplitOptions.RemoveEmptyEntries);
        if (elementos.Length > 0)
        {
          switch (elementos[0])
          {
            case marcaAlimentador:
              if (procesarAlimentadores)
              {
                alimentador a;
                a.tipoComponente = elementos[2].ToString();
                a.x = limitar(int.Parse(elementos[3]), 0, xMax);
                a.y = limitar(int.Parse(elementos[4]), 0, yMax);
                a.banco = limitar(int.Parse(elementos[5]), 1, 3);
                a.anchoX = limitar(int.Parse(elementos[6]), 0, xMax);
                a.anchoY = limitar(int.Parse(elementos[7]), 0, yMax);
                a.útil = limitar(int.Parse(elementos[8]), 0, 4);
                a.zPick = limitar(int.Parse(elementos[9]), 0, 200);
                a.zMover = limitar(int.Parse(elementos[10]), 0, 66);
                a.zPlace = limitar(int.Parse(elementos[11]), 0, 200);
                a.velocidadX = limitar(int.Parse(elementos[12]), 0, velocidadLimiteMaxX);
                a.velocidadY = limitar(int.Parse(elementos[13]), 0, velocidadLimiteMaxY);
                a.velocidadZ = limitar(int.Parse(elementos[14]), 0, velocidadLimiteMaxZ);
                a.velocidadA = limitar(int.Parse(elementos[15]), 0, velocidadLimiteMaxA);
                a.gotasDispensador = limitar(int.Parse(elementos[16]), 0, 99);
                a.avanceAlimentador = Boolean.Parse(elementos[17]);
                a.centradoEstación = Boolean.Parse(elementos[18]);
                a.centradoPinzasX = Boolean.Parse(elementos[19]);
                a.centradoPinzasY = Boolean.Parse(elementos[20]);
                añadirAlimentador(a);
              }
              break;
            case marcaPlaca:
              if (procesarPlacas)
              {
                placa p;
                p.nombre = elementos[2];
                p.x = limitar(int.Parse(elementos[3]), 0, xMax);
                p.y = limitar(int.Parse(elementos[4]), 0, yMax);
                p.t = limitar(float.Parse(elementos[5]), 0, 400);
                p.anchoX = limitar(int.Parse(elementos[6]), 0, xMax);
                p.anchoY = limitar(int.Parse(elementos[7]), 0, yMax);
                p.componentes = new componente[0];  // Crea un vector de componentes vacío.
                p.form = null;
                añadirPlaca(p);
              }
              break;
            case marcaComponente:
```

```
            if (procesarComponentes)
            {
              componente c;
              c.nombre = elementos[2];
                if (procesarAlimentadores && actualizarAlimentadores)
              {
                c.alimentador = numAlimentadores - 1 + int.Parse(elementos[3]);
              }
              else
              {
                c.alimentador = int.Parse(elementos[3]);
              }
              c.x = limitar(int.Parse(elementos[4]), 0, xMax);
              c.y = limitar(int.Parse(elementos[5]), 0, yMax);
              c.t = limitar(int.Parse(elementos[6]), 0, 400);
              c.estado = (estadoComponente)limitar(int.Parse(elementos[7]), 0, 5);
              placas[placas.Length - 1].añadirComponente(c);
            }
            break;
          case marcaOverlay:
            if (procesarPlacas)
            {
              placas[placas.Length - 1].overlay.fichero = elementos[1];
              placas[placas.Length - 1].overlay.color =
                                  Color.FromArgb(int.Parse(elementos[2]));
            }
            break;
          case marcaSolder:
            if (procesarPlacas)
            {
              placas[placas.Length - 1].solder.fichero = elementos[1];
              placas[placas.Length-1].solder.color=Color.FromArgb(int.Parse(elementos[2]);
            }
            break;
          case marcaLayer:
            if (procesarPlacas)
            {
              placas[placas.Length - 1].layer.fichero = elementos[1];
              placas[placas.Length-1].layer.color=Color.FromArgb(int.Parse(elementos[2]));
            }
            break;
          case marcaDesplazamiento:
            if (procesarPlacas)
            {
              placas[placas.Length - 1].desplazamientoGerber.X = int.Parse(elementos[1]);
              placas[placas.Length - 1].desplazamientoGerber.Y = int.Parse(elementos[2]);
            }
            break;
          case marcaInvertir:
            if (procesarPlacas)
            {
              placas[placas.Length - 1].invertir = bool.Parse(elementos[1]);
            }
            break;
        }
      }
    }
    catch
    {
      error = numLinea;   // Si se produjo algún error en una línea, la indica.
    }
  }
  sr.Close();             // Cierra el archivo y libera los recursos utilizados.
  sr.Dispose();
}
else error = -1;  // Si el fichero no existe se devuelve el código de error -1.

return error; // Devuelve el estado final: 0=Ok, -1=fichero inexistente, Nº= línea error.
}
```

Figura 94. Código fuente del método *cargarArchivo*.

5.8.1.1. *Anexión parcial de archivos al trabajo actual*

La característica de poder procesar sólo ciertos tipos de datos de los que contiene el archivo que se acaba de comentar, se usa en la opción del menú *Archivo* → *Anexar*. Ésta muestra una ventana con casillas de selección para cada tipo de datos, permitiendo al usuario especificar qué partes del archivo quiere cargar en memoria, como se muestra en la figura 95.

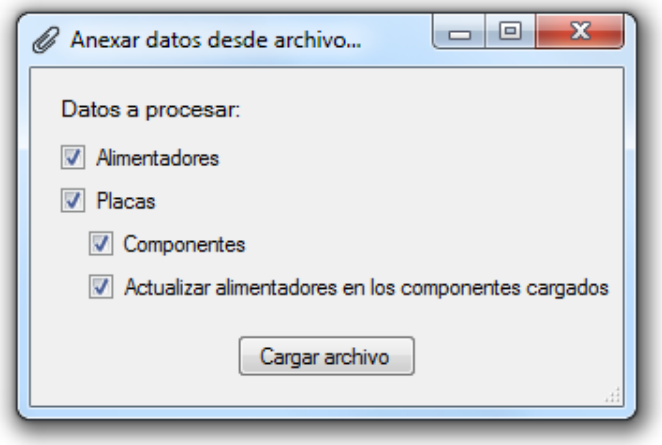


Figura 95. Formulario *Anexar datos desde archivo*.

En la figura 96 se puede ver el código ejecutado cuando se pulsa el botón *Cargar archivo* en el formulario.

```
private void buttonCargar_Click(object sender, EventArgs e)
{
    if (checkBoxAlimentadores.Checked || checkBoxPlacas.Checked)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter =
            "Archivos de composición de placas (*.pnp)|*.pnp|Todos los archivos (*.*)|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            int error = datos.cargarFichero(openFileDialog.FileName,
                        checkBoxAlimentadores.Checked, checkBoxPlacas.Checked,
                        checkBoxComponentes.Checked, checkBoxActualizar.Checked);
            if (error != 0)
            {
                MessageBox.Show("Error en el archivo en la línea " + error,
                    "Error en el archivo", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
            else
            {
                datos.modificados = true;
                this.Close();
            }
        }
    }
    else
    {
        MessageBox.Show("Seleccione antes algún dato que cargar", "Información",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

Figura 96. Código fuente del método *buttonCargar_Click*

Aparte de las comprobaciones de situaciones erróneas, la tarea prácticamente se reduce a lanzar el cuadro de diálogo Abrir para seleccionar el archivo y llamar al método *cargarFichero* comentado en la sección anterior, enviándole como parámetros el nombre del archivo a cargar y las casillas que el usuario marcó.

Gracias a la versatilidad del método de carga de archivos de datos, el código de este formulario es uno de los más breves de los que componen la aplicación, puesto que facilita enormemente la tarea de separar los datos.

5.8.2. Archivos de geometrías

Los archivos de geometrías son mucho más simples que los archivos de datos, puesto que sólo tienen que guardar 4 datos por geometría y siguen el mismo orden que se declaró en la estructura de geometrías en la figura 90.

En la figura 97 se puede ver el código del método que se encarga de guardar las geometrías.

```
public static bool guardarGeometrías(string rutaFichero)
{
    // Graba el fichero de geometrías v1.4
    hacerBackup(rutaFichero);
    try
    {
        FileInfo fi = new FileInfo(rutaFichero);
        StreamWriter sw;
        if ((sw = fi.CreateText()) != null)
        {
            // Primero se graba la cabecera del fichero.
            sw.WriteLine("# Fichero de geometrías Cosy v1.4");
            sw.WriteLine("# Creado el " + Convert.ToString(DateTime.Now));
            sw.WriteLine();

            // A continuación se graba un comentario cabecera de las geometrías.
            sw.WriteLine("# Nombre" + separador + "DimX" + separador
                + "DimY" + separador + "DimZ");

            // A continuación se graban las geometrías.
            for (int n = 0; n < geometrías.Length; n++)
            {
                sw.WriteLine(geometrías[n].nombre + separador + geometrías[n].dimX + separador
                    + geometrías[n].dimY + separador + geometrías[n].dimZ);
            }
            sw.WriteLine();
            sw.Close();
            sw.Dispose();
            geometríasModificadas = false;
        }
        return true;
    }
    catch
    {
        return false;    // Si hubo algún error en la operación, lo notifica.

    }
}
```

Figura 97. Código fuente del método *guardarGeometrías*.

Como se puede ver, lo primero que hace es una copia de seguridad del archivo sobre el que se va a guardar, porque lo más probable es que la grabación de geometrías siempre se haga de forma automática y sobre el mismo archivo del que se leyeron. Si el usuario deja activada la opción de configuración Cargar/guardar geometrías automáticamente, estas operaciones se harán con el último archivo de geometrías que el usuario abrió.

El código del método que carga las geometrías de disco es casi tan simple como el anterior, y se muestra en la figura 98.

```
public static int cargarGeometrías(string rutaFichero)
{
    //Carga el fichero de geometrías v1.4
    geometrías = new geometría[0];      // Crea un array vacío de geometrías
    int error = 0;  //En esta variable se guarda el estado de error de la operación. 0=Ok
    try
    {
        FileInfo fi = new FileInfo(rutaFichero);
        if (fi.Exists)
        {
            StreamReader sr = fi.OpenText();          // Abre el fichero como texto
            string linea;
            int numLinea = 0;
            while ((!sr.EndOfStream) & (error == 0))  // Detecta el final del archivo o error.
            {
                try
                {
                    numLinea++;
                    linea = sr.ReadLine();
                    char[] division = { separador };
                    string[] elementos = new string[50];
                    elementos = linea.Split(division, 50,
                                    StringSplitOptions.RemoveEmptyEntries);
                    if ((elementos.Length > 0) && (elementos[0][0] != '#'))
                    {
                        geometría g;
                        g.nombre = elementos[0];
                        g.dimX = int.Parse(elementos[1]);
                        g.dimY = int.Parse(elementos[2]);
                        g.dimZ = int.Parse(elementos[3]);
                        añadirGeometría(g);      // Se añade la nueva geometría al vector.
                    }
                }
                catch
                {
                    error = numLinea;   // Si se produjo algún error en la línea, la anota.
                }
            }
            geometríasModificadas = false;
            sr.Close();     // Se cierra el archivo y liberan los recursos utilizados.
            sr.Dispose();
        }
        else error = -1;    // Si el fichero no existe se devuelve el código de error -1
    }
    catch
    {
        error = -1;
    }
    return error;     // Devuelve resultado: 0=Ok, -1=fichero inexistente, Nº=Línea con error.
}
```

Figura 98. Código fuente del método *cargarGeometrías*.

Al principio del fichero de geometrías se graban unos comentarios que identifican el número de versión del archivo de geometrías y la fecha y hora de creación. A continuación se deja una línea en blanco y en la siguiente línea se graba otro comentario que será un encabezamiento para los datos que se van a guardar. Para simplificar los archivos de geometrías, todas las líneas vacías o que empiecen por el símbolo # no se interpretan, por lo que todas las líneas de comentarios empiezan por este carácter. A partir de la siguiente línea se graban los datos de cada geometría en una línea aparte, separándolos con el mismo separador que se usa en los archivos de datos.

El código que devuelve el método *cargarGeometrías* es similar al del método *cargarArchivo* explicado anteriormente. En caso de que se detecte una línea errónea, el método devuelve el número de la línea. Si el fichero no existe, devuelve -1 y si todo fue bien devuelve 0.

5.9. Puerto Serie

Para acceder al puerto serie desde el entorno .NET se ha usado la clase *SerialPort*, utilizando sus propiedades, métodos y eventos para evitar la programación a más bajo nivel de la API de Windows. La clase *SerialPort* está disponible en .NET desde la versión 2.0 [**37**].

Para poder usar la clase, es necesario incluir en la aplicación los espacios de nombres que contienen los recursos que ésta usa. En concreto, estos espacios de nombres son *System* y *System.IO.Ports*.

Es necesario buscar los puertos serie disponibles en el equipo antes de poder empezar el proceso de selección y configuración de uno de ellos. El método *GetPortNames* de la clase *SerialPort* devuelve un *array* con los nombres de todos los puertos COM[1] del sistema. Hay que tener en cuenta que el índice de cada elemento del array no tiene nada que ver con el número del puerto COM, ya que puede haber ausencias y además *GetPortNames* no garantiza que el *array* devuelto esté ordenado. Por tanto, como dichos nombres se mostrarán en un *comboBox* de nuestra aplicación (figura 71), para que estén ordenados se pondrá el valor true en su propiedad *Sorted.*

Antes de acceder a un puerto o de configurar sus parámetros, la aplicación debe crear un objeto *SerialPort*. Este objeto tiene una serie de propiedades que definirán los parámetros y el estado del puerto. Los parámetros de transmisión por defecto al crear el objeto son 9600 bps, sin paridad, 1 bit de stop, y sin control de flujo. Además, el nombre del puerto que se defina en la

[1] En los ordenadores tipo PC, los puertos serie se han denominado tradicionalmente COM1, COM2, etc.

propiedad *PortName* debe coincidir con uno de los nombres del *array* devuelto por el método *GetPortNames*.

La aplicación guarda el nombre del puerto usado la última vez que se utilizó con el propósito de que el usuario no tenga que volver a configurarlo si no ha cambiado. Normalmente el nombre de un puerto serie interno no cambiará a menos que el usuario lo modifique voluntariamente en su cuadro de propiedades del *Administrador de Dispositivos* de Windows. Sin embargo, el nombre de un puerto serie virtual de un adaptador USB (como el de la figura 50) podría cambiar automáticamente al ser conectado a un puerto USB distinto al de una ocasión anterior.

Por tanto, es necesario comprobar que el nombre del puerto que se pretende abrir está en la lista de nombres de los puertos serie instalados en el sistema. Esto se realiza en la fase de inicialización del programa, en el procedimiento *Inicializar()* de la clase *datos*, antes de llamar a la función que abre y configura el puerto, tal como se muestra en la figura 99.

Tras abrir un puerto, la aplicación que lo abre obtiene acceso exclusivo al puerto. Ninguna otra aplicación puede abrirlo o acceder al mismo hasta que el puerto sea cerrado por la aplicación que lo abrió. Si otro programa intentase abrir un puerto que está en uso se produciría la excepción *UnauthorizedAccessException*. Para evitar esto, se puede usar la propiedad *IsOpen* del puerto antes de intentar abrirlo.

```
// Intenta configurar el puerto serie que se usó la última vez que corrió el programa.
nombrePuertoSerie = Settings.Default.NombrePuertoSerie;

// Obtiene la lista de puertos serie disponibles y, si existe, intenta abrirlo.
string[] puertosDisponibles = SerialPort.GetPortNames();
if (puertosDisponibles.Contains(nombrePuertoSerie))
{
    if (!abrirPuertoSerie())
        MessageBox.Show("Ha habido un error al abrir el puerto serie " +
                datos.nombrePuertoSerie, "Puerto serie no disponible",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
}

// Si el puerto serie preseleccionado no está disponible, avisa al usuario y le muestra
// el form de configuración para que elija.
else
{
    MessageBox.Show("El puerto " + nombrePuertoSerie +
        " no está disponible.\nPor favor, antes de continuar seleccione un puerto.",
        "Puerto serie no disponible", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    FormConfiguracionPuerto fcp = new FormConfiguracionPuerto();
    fcp.ShowDialog(formPadre);
}
```

Figura 99. Selección y apertura del puerto serie.

Sin embargo, aunque poco probable, es posible que otro proceso abra el puerto después que la aplicación ha determinado que está cerrado con *IsOpen* y antes de llamar al método *Open* para abrirlo. Para evitar que la aplicación se detenga inesperadamente debido a la excepción que se produciría, es mejor capturar cualquier excepción que se pudiera producir durante la apertura del puerto y avisar al usuario. Para simplificar, no se muestra esta parte en el código anterior ni otras comprobaciones dependientes de la configuración de ususario que también se realizan, en el Anexo III se puede ver el código completo de esta parte de la función de inicialización.

En la figura 100 se muestra el código fuente del procedimiento que se encarga de abrir y configurar adecuadamente el puerto serie.

```
public static bool abrirPuertoSerie()
{
    try
    {
        // En caso de que no se haya creado aún el objeto, lo crea.
        if (puertoSerie == null) puertoSerie = new SerialPort();

        // En caso de que se vaya a cambiar de puerto, cierra el puerto abierto
        // previamente y configura el nombre del nuevo puerto a abrir.
        if (puertoSerie.PortName != nombrePuertoSerie)
        {
            if (puertoSerie.IsOpen) puertoSerie.Close();
            puertoSerie.PortName = nombrePuertoSerie;
        }

        // Configura el resto de parámetros del puerto.
        puertoSerie.BaudRate = Settings.Default.VelocidadPuertoSerie;
        puertoSerie.DataBits = Settings.Default.BitsDatosPuertoSerie;
        puertoSerie.StopBits = Settings.Default.BitsStopPuertoSerie;
        puertoSerie.Parity = Settings.Default.ParidadPuertoSerie;

        // Se usarán timeoutos para determinar si la máquina responde a los comandos.
        datos.puertoSerie.ReadTimeout = 5000;      // Timeout de recepción
        datos.puertoSerie.WriteTimeout = 5000;     // Timeout de transmisión

        // Establece el carácter de fin de las cadenas transmitidas.
        puertoSerie.NewLine = "\r";

        // Abre el puerto serie si no se encontraba ya abierto.
        if (!puertoSerie.IsOpen) puertoSerie.Open();

        // Si el puerto serie está preparado, lo señaliza.
        if (((puertoSerie != null) && puertoSerie.IsOpen))
        {
            datos.puertoSerie.DiscardInBuffer();     // Se limpian los buffers de entrada
            datos.puertoSerie.DiscardOutBuffer();    // y salida
            puertoSerieDisponible = true;
        }
    }

    // Si el puerto serie no está preparado, lo señaliza.
    catch
    {
        puertoSerieDisponible = false;
        return false;
    }
    return true;
}
```

Figura 100. Apertura y configuración del puerto serie.

Este procedimiento es llamado desde dos puntos del programa: al iniciarse el programa, desde el procedimiento *Inicializar()* de la clase *datos* y al seleccionar un nuevo puerto en la ventana de configuración del puerto serie, desde el procedimiento *buttonAceptar_Click()* de la clase *formConfiguraciónPuerto.*

En el código de la figura 100 puede verse que los parámetros de comunicación del puerto serie no se establecen a los valores fijos determinados anteriormente (4800, n, 8, 1), sino que son obtenidos a partir de las propiedades de configuración de nuestra aplicación, mencionadas anteriormente. Se ha decidido hacerlo así para dejar abierta la posibilidad de variar estos parámetros por defecto sin necesidad de editar el código fuente ni recompilar la aplicación, en un hipotético caso en que se quisiera reutilizar la aplicación con otros parámetros de configuración del puerto serie.

Los valores de las propiedades se almacenan en el archivo XML con extensión *.config*, que se describió anteriormente y que puede ser fácilmente editado en caso de que se quiera cambiar cualquiera de los valores por omisión. Con los datos almacenados en dichas propiedades durante el desarrollo de la aplicación, la configuración del puerto es equivalente a la mostrada en la figura 101.

```
// Configura el resto de parámetros del puerto.
puertoSerie.BaudRate = 4800;
puertoSerie.DataBits = 8;
puertoSerie.StopBits = System.IO.Ports.StopBits.One;
puertoSerie.Parity = System.IO.Ports.Parity.None;
```

Figura 101. Configuración del puerto serie.

5.10. Editor de alimentadores

En el caso de trabajar con la Cosy, tanto si se importa un diseño desde Altium como si se va a crear desde cero, es necesario especificar desde dónde y cómo se debe coger y además cómo se debe tratar cada tipo de componente que se va a montar. Con el resto de máquinas también hay que especificar algunos de estos parámetros El lugar para hacer esto es el editor de alimentadores, que se puede ver en la figura 102.

Como se puede ver, este editor permite especificar todos los valores que hay que almacenar en la estructura de alimentadores comentada anteriormente. En la parte izquierda de la ventana hay un listado que contiene todos los alimentadores definidos, identificados por su número de orden y el tipo componente que contienen. En la parte derecha se ven todos los parámetros del alimentador que se está editando. En la parte inferior de la ventana hay tres

botones que permiten crear un nuevo alimentador utilizando los parámetros que se ven actualmente en la ventana, guardar los cambios realizados a dichos parámetros en el alimentador que está seleccionado o borrar el alimentador que está seleccionado.

Si no hay alimentadores al abrir este formulario, todas las casillas aparecerán con valores predeterminados, listos para crear el primer alimentador a partir de estos valores cuando se pulse el botón. Si se pulsa el botón "*Crear nuevo alimentador*" mientras se están viendo los parámetros de otro, y sin haber cambiado ninguno, se creará un nuevo alimentador que será una copia idéntica al que se estaba viendo, salvo por su número de orden en el vector de alimentadores.

Los cambios realizados en los parámetros de un alimentador no se guardan en el vector de alimentadores hasta que se pulsa el botón "Guardar cambios" para evitar modificaciones accidentales. Por tanto, si tras realizar cambios se quieren descartar, solo hay que seleccionar otro alimentador o cerrar la ventana sin pulsar el botón de guardar.

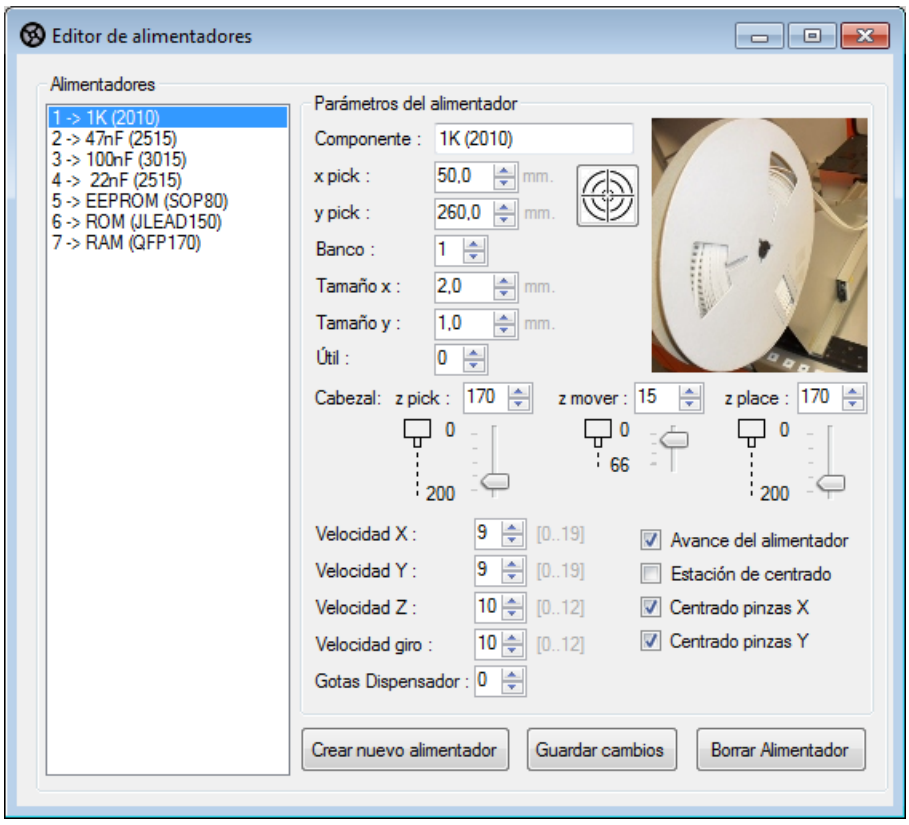


Figura 102. Formulario Editor de alimentadores.

Lo habitual es que gran cantidad de componentes compartan muchos parámetros de sus alimentadores, distinguiéndose únicamente por el tipo de componente de que se trata y su posición en la zona de alimentadores. Seleccionando en la lista el alimentador de un componente similar al que se quiere crear aparecerán rellenos todos los campos del formulario con sus valores,

debiéndose cambiar sólo aquellos que difieran y pulsar el botón “Crear nuevo alimentador”. Esta forma de trabajar facilita y acelera la tarea de introducción de datos de alimentadores, que es algo que hay que hacer siempre que se va a trabajar con la Cosy, a menos que tomen los datos desde los alimentadores creados para trabajos anteriores, anexándolos al trabajo actual mediante el menú *Archivo* → *Anexar*.

El botón de la mira permite abrir el formulario de control directo del brazo de la Cosy, utilizando la mira incluida en la imagen de la cámara para “apuntar” al lugar por donde se quiera coger el componente en su alimentador (normalmente será el centro del componente). Mientras movemos la cámara, los valores x e y se actualizan constantemente, por lo que al salir del formulario de la cámara ya estarán indicadas las coordenadas de captura del componente.

Esto hace que para componentes similares a uno existente en la lista, el trabajo de crear su alimentador se reduzca a seleccionar el similar, escribirle un nuevo nombre en el campo “*Componente*”, localizarlo con la cámara y finalmente pulsar “*Crear nuevo alimentador*”.

Para conseguir la actualización automática de las coordenadas, se ha creado un método encargado de recibir la comunicación de datos desde el formulario de la cámara. En la figura 103 se puede ver el código relacionado con el uso de la cámara.

En primer lugar, el método *buttonCámara_Click* responderá a la pulsación del botón de la mira creando un nuevo formulario de control manual con la cámara. A continuación registra en éste el método que recogerá las actualizaciones de coordenadas, llamado *recuperarCoordenadasCámara*, apoyándose en el delegado *devolverCoordenadasCallback*. Un delegado es el equivalente aproximado a un puntero a función del lenguaje C, pero válido para el código seguro de C#. Finalmente se le indican como coordenadas de partida las que se muestran en los campos de la ventana y se muestra en pantalla.

```
private void buttonCámara_Click(object sender, EventArgs e)
{
    FormControlManualConCámara fc = new FormControlManualConCámara();
    fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasCámara));
    fc.IrACoordenadas(numericUpDownX.Value, numericUpDownY.Value);
    fc.Show();
}

public void recuperarCoordenadasCámara(int x, int y)
{
    numericUpDownX.Value = (decimal)x / 10;
    numericUpDownY.Value = (decimal)y / 10;
}
```

Figura 103. Código fuente de los métodos *buttonCámara_Click* y *recuperarCoordenadasCámara*.

Por su parte, el método *recuperarCoordenadasCámara* será llamado desde el formulario de control manual con la cámara cada vez que se produzca una actualización de las coordenadas, comunicándole el nuevo valor de éstas. Este método se limita a poner los valores recibidos en los campos del formulario para que el usuario pueda ver que se están actualizando. Como los valores recibidos son valores de coordenadas, que están expresadas en décimas de milímetro, se dividen entre diez para pasarlas a milímetros antes de mostrarlas.

El delegado *devolverCoordenadasCallback* se ha definido para que haga referencia a métodos que devuelvan *void*, teniendo como entrada dos parámetros enteros (que será la firma de todos los métodos para recuperar coordenadas), tal como se puede ver en la figura 104.

```
public delegate void devolverCoordenadasCallback(int x, int y);
```

Figura 104. Declaración del delegado *devolverCoordenadasCallback*.

Para evitar inconsistencias en los datos almacenados, ningún campo se puede dejar vacío o contener datos erróneos, por lo que se comprueba esto explícitamente en caso de ser necesario. En la figura 105 se puede ver el formulario indicando que hay un error en el *textBox* para el tipo de componente, pues se ha dejado vacío. En caso de que esté vacío, este campo se vuelve de color rojo y se inhabilitan los botones que podrían almacenar este valor incorrecto, que son los de crear un nuevo alimentador y guardar cambios.

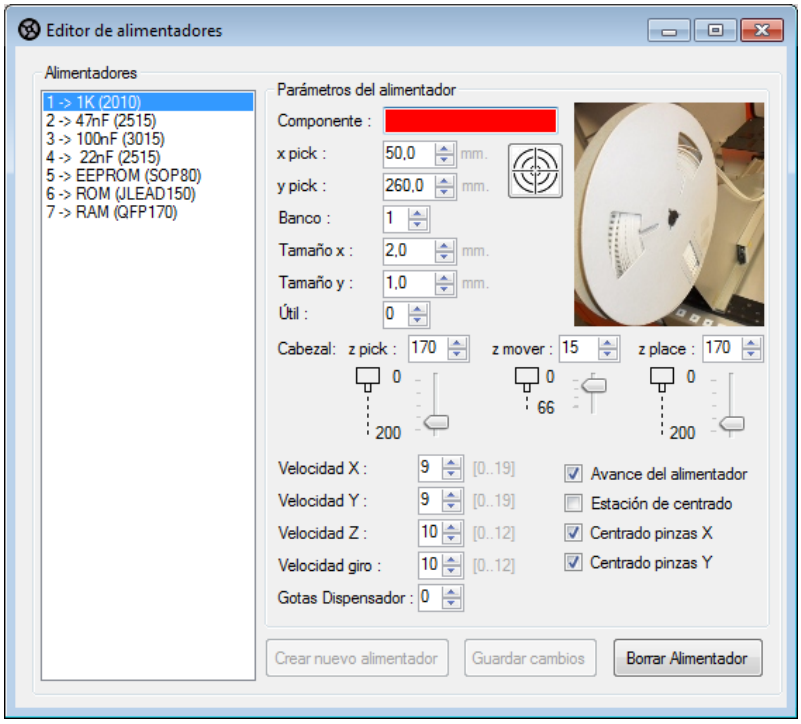


Figura 105. Formulario Editor de alimentadores mostrando un error.

Se obtendrá el mismo comportamiento si el campo contiene alguno de los caracteres no permitidos (coma, punto y coma y comillas). Estos caracteres podrían dar problemas al manejar los ficheros, pues podrían ser mal interpretados por la aplicación al coincidir con caracteres del marcado. En la figura 106 se muestra el código fuente de la comprobación de datos de este control.

```
private void textBoxTipoComponente_TextChanged(object sender, EventArgs e)
{
    if (datos.cadenaCorrecta(textBoxTipoComponente.Text))
    {
        textBoxTipoComponente.BackColor = SystemColors.Window;
        botonAñadir.Enabled = true;
        if (datos.alimentadores.Length > 0) botonModificar.Enabled = true;
    }
    else
    {
        textBoxTipoComponente.BackColor = Color.Red;
        botonAñadir.Enabled = false;
        botonModificar.Enabled = false;
    }
}
```

Figura 106. Código fuente del método *textBoxTipoComponente_TextChanged*.

Cada vez que el texto del control cambie, se llama a este método, que comprueba si la cadena escrita en el control es correcta. En función de esto establece el color de fondo del control y habilita o deshabilita los botones mencionados anteriormente.

Como varios controles de distintos formularios comprobarán las mismas condiciones, se ha escrito un método de apoyo, llamado *cadenaCorrecta*. Este método se encuentra en la clase datos, junto con el resto de datos y métodos comunes de la aplicación. El código de este método se puede ver en la figura 107.

```
public static bool cadenaCorrecta(string cadena)
{
    if (cadena.Length == 0) return false;
    foreach (char c in caracteresNoPermitidos)
    {
        if (cadena.Contains(c)) return false;
    }
    return true;
}
```

Figura 107. Código fuente del método *cadenaCorrecta*.

Como se puede ver en la figura, el código devuelve false si la cadena tiene longitud cero o si contiene cualquiera de los caracteres no permitidos. Estos caracteres no permitidos están definidos como una matriz ce caracteres, declarada en el área de constantes y campos de la clase datos como muestra la figura 108.

De esta forma se permite hacer cambios en los caracteres usados con algún significado especial en el formato de ficheros creado sin tener que editar manualmente todos los métodos de comprobación de errores. Así, durante el desarrollo de la aplicación fue posible hacer pruebas, cambiando o añadiendo a esta matriz los caracteres deseados.

Un detalle de la implementación es que si en lugar de declarar el campo *static*, se declara *const* (que sería más apropiado, al ser valores constantes), la aplicación no compila correctamente, puesto que hay una restricción que es que un campo *const* de un tipo de referencia distinto de *String* (las matrices son tipos de referencia), sólo se puede inicializar con *null*.

```
public static char[] caracteresNoPermitidos = { ';', ',', '"' };
```

Figura 108. Declaración de los caracteres no permitidos.

En la versiones iniciales, los campos numéricos fueron creados también con controles textBox, por lo había que comprobar explícitamente los posibles errores de conversión numérica en estos controles, haciendo el código más largo y complejo.

En las últimas versiones, para los campos numéricos se ha preferido usar el control *numericUpDown* en lugar del clásico *textBox* por varias razones. Una de ellas es que dispone de unos pequeños botones a la derecha que permiten hacer incrementos o decrementos del contenido, lo que permite ajustar algunos valores, como el banco o el útil, con un simple clic del ratón la mayoría de las veces.

El otro motivo es el chequeo de valores erróneos en el control. Mientras que un *textbox* acepta cualquier carácter alfanumérico, el control *numericUpDown* sólo permite caracteres numéricos. Además, con las propiedades *Minimum*, *Maximum DecimalPlaces* del control se puede especificar el rango de valores válidos y el número de decimales que se desean.

Aún en el caso de que el usuario intencionadamente vacíe el campo, el valor almacenado por éste siempre será el último valor válido que contenía anteriormente. De esta forma un alimentador siempre se crea con todos sus parámetros dentro de los valores permitidos sin necesidad de oscurecer el código fuente con comprobaciones explícitas de errores numéricos.

Para mantener la consistencia de los datos, un alimentador sólo se puede borrar si no está en uso en ningún componente. En caso de que algún componente esté usando el alimentador se

mostrará una ventana de aviso al usuario. En la figura 109 se puede ver el código que se ejecuta al pulsar el botón para quitar un alimentador.

```
private void botonQuitar_Click(object sender, EventArgs e)
{
    if (datos.quitarAlimentadorSiNoUsadoYReajustarComponentes(listBoxAlimentadores.SelectedIndex))
    {
        actualizarInterfazYSeleccionar(listBoxAlimentadores.SelectedIndex);
        datos.modificados = true;
    }
    else
    {
        MessageBox.Show("El alimentador está en uso, no se puede borrar.\r\n",
                "No se puede borrar el alimentador", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Figura 109. Código fuente del método *botonQuitar_Click*.

Como se puede ver en el código anterior, al pulsar el botón para quitar el componente se llama al método *quitarAlimentadorSiNoUsadoYReajustarComponentes* de la clase *datos*, que se encarga de hacer todas las comprobaciones y el trabajo necesario y devuelve *true* en caso de que se pudiese quitar y *false* si no se hizo.

Como es lógico, cuando se elimina un alimentador, los elementos del vector que estaban en posiciones posteriores al alimentador eliminado se desplazarán a una posición anterior para que no queden "huecos" en el vector. Esto obliga a comprobar todos los componentes para cambiar su número de alimentador en caso de que se haya visto modificado. De no hacerlo se producirían graves inconsistencias en los datos, que echarían a perder todo el trabajo.

El código fuente de los métodos *quitarAlimentadorSiNoUsadoYReajustarComponentes* y sus métodos de apoyo *alimentadorNoUsado* y *quitarAlimentador*, se puede ver en la figura 110.

En el código se observa que, en caso de que el alimentador no se esté usando, un bucle recorre todas las placas y, en su interior, otro bucle recorre todos los componentes, buscando códigos de alimentador superiores al que se va a eliminar para corregirlo. Tras esta operación de ajuste, se puede quitar el alimentador definitivamente de forma segura.

Para comprobar si el alimentador no se está usando, es necesario recorrer también todos los componentes de todas las placas, verificándolo uno por uno, como se puede ver en el código del método *alimentadorNoUsado*.

Para eliminar un alimentador en cualquier posición, puesto que en C# las matrices son de dimensión fija, se crea otra matriz con un elemento menos y se copian los elementos anteriores y

posteriores al que se desea eliminar de la matriz original. Finalmente se clona la nueva matriz sobre la original, que queda así redimensionada.

```
public static bool quitarAlimentadorSiNoUsadoYReajustarComponentes(int alimentadorPedido)
{
    if (alimentadorNoUsado(alimentadorPedido))
    {
        // Primero reajusta el índice de los alimentadores posteriores en los componentes
        for (int p = 0; p < datos.placas.Length; p++)
        {
            for (int c = 0; c < datos.placas[p].componentes.Length; c++)
            {
                if (datos.placas[p].componentes[c].alimentador > alimentadorPedido)
                {
                    datos.placas[p].componentes[c].alimentador--;
                }
            }
        }

        quitarAlimentador(alimentadorPedido);
        return true;
    }
    else return false;
}

public static bool alimentadorNoUsado(int alimentadorPedido)
{
    // Comprueba si ningún componente usa el alimentador dado
    for (int p = 0; p < placas.Length; p++)
    {
        for (int c = 0; c < placas[p].componentes.Length; c++)
        {
            if (placas[p].componentes[c].alimentador == alimentadorPedido) return false;
        }
    }
    return true;
}

public static void quitarAlimentador(int alimentadorPedido)
{
    alimentador[] a2 = new alimentador[datos.alimentadores.Length - 1];
    Array.Copy(datos.alimentadores, a2, alimentadorPedido);
    Array.Copy(datos.alimentadores, alimentadorPedido + 1, a2, alimentadorPedido,
               datos.alimentadores.Length - 1 - alimentadorPedido);
    datos.alimentadores = (alimentador[])a2.Clone();
}
```

Figura 110. Código fuente de los métodos implicados al borrar un alimentador de forma segura.

5.11. Editor de placas y componentes

5.11.1. Edición de parámetros de las placas

Puesto que la máquina pick and place Cosy no tiene un sistema de posicionamiento automatizado para las placas en el panel de trabajo, será el operador quien las coloque, decidiendo en ese momento el lugar que ocuparán sobre el panel y la orientación que tendrán. Por ello, normalmente será necesario especificar manualmente en nuestro programa los datos que definen la posición de la placa en dicho panel.

Para esta tarea se ha creado el formulario “Editor de placas y componentes”, mostrado en la figura 111. Como se puede ver en la parte superior de la imagen, el formulario permite especificar para cada placa todos los parámetros de su estructura en el grupo de controles “Parámetros de la placa”.

A la izquierda se muestra un listado con todas las placas que están en memoria actualmente. Haciendo clic sobre cualquiera de ellas se actualizará la ventana con sus datos y los de sus componentes. Bajo el listado hay dos botones que permiten cambiar la posición de la placa seleccionada, subiéndola o bajándola en la lista y un tercer botón permite ver la representación gráfica de la placa y sus componentes, abriendo para ello su editor gráfico.

Al igual que sucedía con el editor de alimentadores, todos los campos numéricos se introducen usando controles tipo *numericUpDown*, que facilitan tanto la entrada de datos por parte del usuario, como la comprobación de que los valores están dentro de los límites admisibles, evitando saturar el código fuente de comprobaciones de error en la introducción de datos.

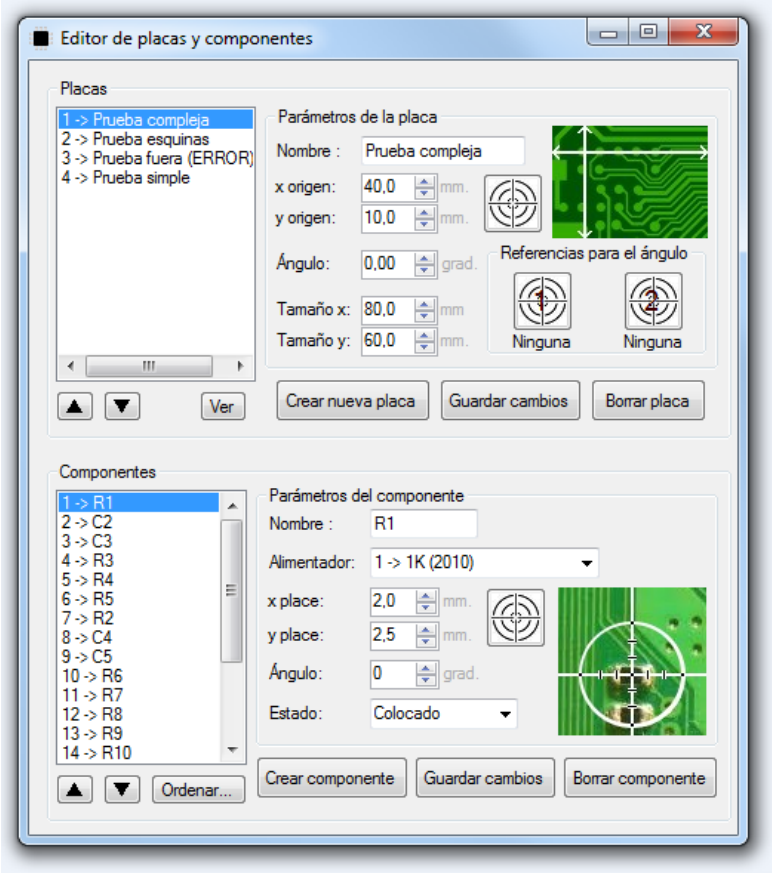


Figura 111. Formulario Editor de placas y componentes.

En el listado, se puede ver que el nombre de cada placa aparece precedido por su número de orden. Al reordenar la posición de las placas se cambia su ubicación en la matriz de placas. Por tanto, cuando se vayan a montar las placas desde el menú producción, se respetará el orden que el usuario haya especificado.

Además, si los datos de la placa o de alguno de sus componentes contiene algún error que impidiera montar la placa, aparecerá "(ERROR)" escrito detrás del nombre de la placa. En la tercera placa de ejemplo de la figura 111 se ha introducido deliberadamente un error en la posición de un componente, situándolo fuera de los límites de la placa, para ilustrarlo.

Como sucedía en el caso del editor de alimentadores, para evitar inconsistencias, todos los parámetros de una placa deben tener datos correctos, por lo que la aplicación se encarga de verificarlo. Como se comentó anteriormente, los controles *numericUpDown* no presentan problemas en este aspecto, pero no ocurre lo mismo con los campos de tipo *textBox*.

Se debe especificar un nombre para la placa aunque, como se comentó anteriormente, su única utilidad es que el operador la identifique fácilmente. Si se intenta dejar el campo vacío, se pondrá el fondo rojo y se deshabilitarán los botones para crear una nueva placa y guardar cambios, al igual que sucede en el editor de alimentadores. Lo mismo sucederá si se introducen caracteres no permitidos, ya que la comprobación de errores es idéntica a la mostrada en la figura 106, pero aplicada al campo "*Nombre*".

Sin embargo, con los campos numéricos se permitirá introducir valores que hagan que la placa quede fuera de los límites (sin superar individualmente los límites del área de trabajo de la máquina). Todos los campos numéricos permiten introducir valores desde cero hasta el límite máximo de la máquina en el eje correspondiente. En el caso de la Cosy, para la posición *x* del origen y el tamaño en la dirección del eje *x*, el límite es de 390 mm. Para la posición *y* del origen y el tamaño en la dirección del eje *y* el límite es de 261 mm.

Estos son los valores predeterminados al instalar la aplicación o ejecutarla por primera vez, aunque desde el menú configuración se pueden especificar fácilmente los límites de cada máquina o incluso introducir unos límites arbitrarios especificados por el usuario. Para el ángulo se ha limitado el máximo a 399,99 grados centesimales (gradianes), puesto que 400 grados centesimales es lo mismo que 0 al haber dado una vuelta completa.

Esto hace que si, por ejemplo, se pone sobre el panel de la Cosy una placa en la posición (300,0), con 0° de giro y su tamaño es de 100 mm x 100 mm, ésta se salga de los límites, al estar su borde derecho fuera de las dimensiones del panel de trabajo. Sería posible (y relativamente

fácil) hacer que la aplicación impusiese límites para evitar que se introduzcan datos numéricos incorrectos, pero se ha decidido permitir este tipo de incorrecciones para dar mayor flexibilidad al usuario durante el trabajo. Permitiéndole introducir valores ligeramente incorrectos mientras está haciendo los ajustes podría facilitársele la tarea, hasta que todo quede bien colocado sobre el panel de trabajo de la máquina.

Cuando se de esta situación, tras guardar los valores incorrectos se marcarán con el fondo en rojo todos los valores numéricos, indicando que hay errores en los parámetros físicos de la placa. Para evitar riesgos, el módulo de producción de la Cosy ignorará todas las placas con errores, por lo que no será posible iniciar el montaje de estas placas mientras no se corrijan dichos errores, aunque sí permitirá montar la que estén correctas.

En la imagen de la izquierda de la figura 112 se muestra el aspecto del formulario con un error en el campo nombre (no deja guardar el error creando una nueva placa o guardando los cambios), mientras que en la parte derecha se muestra la misma placa con un error introducido deliberadamente que hace que la placa sobresalga de los límites del panel de trabajo, tras guardar los cambios.

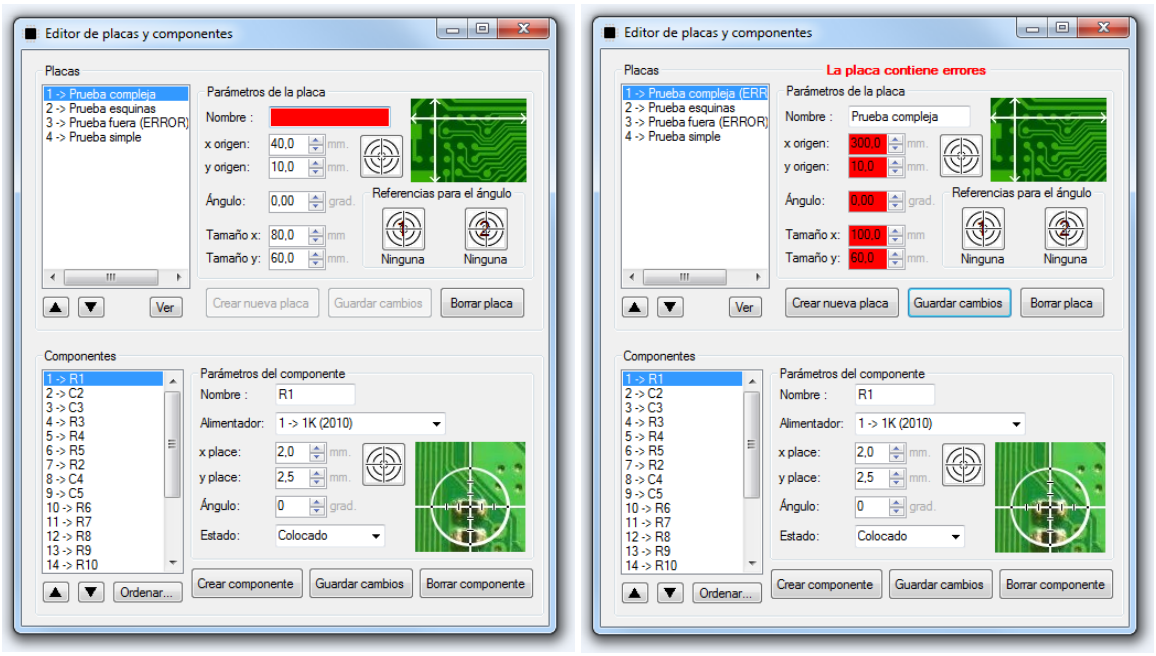


Figura 112. Formulario Editor de placas y componentes con dos tipos de error en los datos.

Como se puede ver, cuando se está mostrando una placa que contiene errores grabados en sus datos, se muestra un mensaje que destaca en rojo sobre sus parámetros, indicándolo. En el caso del ejemplo es obvio, pues ya lo estaba indicando con el fondo rojo de sus parámetros y en el nombre de la placa en el listado. Sin embargo, si el nombre de la placa es muy largo, el aviso

puede quedar oculto a la derecha del nombre y, en caso de que el error estuviese en uno de sus componentes, podría ser en uno que no esté a la vista, como veremos con más detalle más adelante, en la sección dedicada a la edición de componentes. De esta forma se avisa al usuario más claramente.

La posición de la placa y el ángulo que forma con el eje x de nuestro sistema de coordenadas son muy importantes en la colocación de los componentes, puesto que son la referencia a partir de la que se calcula la posición de todos los componentes que hay en la placa. En ambos casos, se puede especificar el dato directamente en la caja de la ventana o también de forma semiautomática, usando la mira de la cámara.

Para usar la mira de la cámara, es conveniente que el diseñador de la PCB incluya ciertas marcas de referencia, en ocasiones llamadas fiduciarios o fiduciales [**38**], aunque es posible usar otros puntos significativos como marcas de referencia.

Para especificar la posición del origen con la cámara, basta pulsar el botón de la mira para que aparezca la ventana de control manual del brazo de la máquina. Es conveniente que el diseñador de la PCB incluya una marca de referencia en la esquina superior izquierda de la placa, pues éste será el origen de referencia tomado por el sistema gráfico desarrollado para calcular la posición de los componentes.

Una vez centrada la referencia del origen de la placa en la mira de la cámara, el parámetro estará introducido correctamente en el editor de placas y componentes de forma automática. El código fuente de esta característica es similar al mostrado anteriormente para obtener las coordenadas del alimentador, mostrado en la figura 103, por lo que no es necesario repetirlo aquí.

Para especificar de forma semiautomática el ángulo de giro de la placa es necesario señalar con la mira de la cámara dos marcas de referencia que describan una línea paralela al eje *x* en el diseño original de la PCB. La aplicación se encargará de calcular el ángulo entre el segmento marcado y el eje *x*, conociendo por tanto el ángulo de rotación de la placa y ajustando el parámetro en el editor de placas automáticamente mientras se apunta a las marcas de referencia.

De manera predeterminada, al pulsar el botón de la primera referencia se inicia la búsqueda en el origen de la placa, por lo que se acelera el trabajo si se escoge éste como primera marca de referencia, aunque el diseñador tiene total libertad sobre los puntos que quiere usar como referencia. Por tanto, resulta conveniente, aunque no es obligatorio, que en el diseño

original de la PCB se incluya otra marca de referencia en la misma coordenada *y*, pero en el extremo derecho de la placa para buscarla con el botón de la segunda referencia, teniendo así dos referencias: una en la esquina superior izquierda de la placa y otra en la esquina superior derecha.

Tras pulsar cada botón de referencia, el texto debajo del botón, que inicialmente pone "*Ninguna*", cambia para indicar las coordenadas del panel que se han tomado como referencias para calcular el ángulo. Es muy importante que la referencia número 1 esté a la izquierda de la número 2 en el diseño de la PCB, porque si no, el ángulo calculado será incorrecto.

Para calcular el ángulo se utiliza la fórmula, donde Δy y Δx son las diferencias entre las coordenadas correspondientes de las referencias.

$$\alpha = \tan^{-1}\frac{\Delta y}{\Delta x}$$

En la figura 113 se puede ver el código relacionado con el cálculo del ángulo. Como se observa, los métodos *buttonCámaraReferencia1_Click* y *buttonCámaraReferencia2_Click* no se diferencian casi del método *buttonCámara_Click* del editor de alimentadores, mostrado anteriormente en la figura 103. La única diferencia entre estos métodos es que cada uno crea un delegado que apunta hacia su propio método para recuperar las coordenadas y además le envían las coordenadas del origen de la placa como coordenadas iniciales.

Por otra parte, los métodos encargados de recuperar las coordenadas de las referencias (*recuperarCoordenadasReferencia1Cámara* y *recuperarCoordenadasReferencia2Cámara*) guardan las coordenadas recibidas en los campos de la clase referencia1 y referencia2. Éstos campos son instancias de una estructura muy simple para almacenar los puntos dados, que únicamente tiene dos campo enteros llamados *x* e *y*.

Tras almacenar el valor, cada una actualiza el texto que hay bajo su botón correspondiente. Por último, si ya se ha especificado algún valor para la otra referencia, se llama al método *calcularÁnguloPlaca*. Para detectar esto, el método encargado de mostrar los valores iniciales en los controles de la ventana también inicializa las referencias a cero.

Éste método calcula la diferencia entre las coordenadas x y las coordenadas y de ambas referencias y su arcotangente, según la fórmula anterior. Como Math.Atan2 devuelve el ángulo en radianes, será necesario multiplicarlo por $200/\pi$ para pasarlo a gradianes (grados centesimales), que es la unidad con la que opera la Cosy. Por último, se almacena el valor calculado en el control para que se muestre en pantalla.

Como la ventana de la cámara notifica las nuevas coordenadas cada vez que hay un cambio, se verá cambiar el ángulo cada vez que se mueva el brazo de la máquina mientras se ajustan las coordenadas de la segunda referencia que se introduzca. Por tanto, al salir de la ventana de la cámara ya estará escrito el valor del ángulo buscado.

```
private void buttonCámaraReferencia1_Click(object sender, EventArgs e)
{
  FormControlManualConCámara fc = new FormControlManualConCámara();
  fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasReferencia1Cámara));
  fc.IrACoordenadas(numericUpDownXPlaca.Value, numericUpDownYPlaca.Value);
  fc.ShowDialog();
}

public void recuperarCoordenadasReferencia1Cámara(int x, int y)
{
  referencia1.x = x;
  referencia1.y = y;
  labelReferencia1.Text = x + "," + y;
  if ((referencia2.x != 0) || (referencia2.y != 0)) calcularÁnguloPlaca();
}

private void buttonCámaraReferencia2_Click(object sender, EventArgs e)
{
  FormControlManualConCámara fc = new FormControlManualConCámara();
  fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasReferencia2Cámara));
  fc.IrACoordenadas(numericUpDownXPlaca.Value, numericUpDownYPlaca.Value);
  fc.ShowDialog();
}

public void recuperarCoordenadasReferencia2Cámara(int x, int y)
{
  referencia2.x = x;
  referencia2.y = y;
  labelReferencia2.Text = x + "," + y;
  if ((referencia1.x != 0) || (referencia1.y != 0)) calcularÁnguloPlaca();
}

private void calcularÁnguloPlaca()
{
  double deltaX = referencia2.x - referencia1.x;
  double deltaY = referencia2.y - referencia1.y;
  double ánguloEnGradianes = -200 * Math.Atan2(deltaY, deltaX) / Math.PI;
  if (ánguloEnGradianes < 0) ánguloEnGradianes += 400;
  numericUpDownÁnguloPlaca.Value = (decimal)ánguloEnGradianes;
}
```

Figura 113. Código fuente de los métodos para calcular el ángulo de la placa a partir de dos referencias.

Si la placa va a ser colocada en el ángulo original del diseño (normalmente horizontal, es decir, paralela al eje x) no es necesario tener en cuenta el ángulo y se puede dejar a 0 este valor, aunque no está de más medirlo, para que el programa tenga en cuenta las pequeñas desviaciones que se podrían producir.

Las dimensiones también son importantes, puesto que se usan en el programa para determinar si hay componentes situados fuera de los límites de la placa, aunque aquí no son críticas como lo son la posición y el ángulo. Éste es uno de los errores en los datos que más

fácilmente se puede detectar para evitar placas mal montadas y la aplicación lo comprueba antes de permitir que se inicie el montaje de una placa con la Cosy desde el menú *Producción*.

Sólo cuando haya al menos una placa creada, se activará la parte inferior de la pantalla, que se utilizará en caso de que haya que editar los datos de los componentes.

5.11.2. Edición de parámetros de los componentes

Habitualmente se importarán los diseños desde archivos de pick & place de Altium, por lo que lo normal será que no sea necesario editar manualmente los parámetros de los componentes. Sin embargo, en algunas ocasiones puede ser necesario cambiar alguno de los datos, debido a alguna variación en los componentes que se utilizarán en el montaje final, o añadir/eliminar componentes por una modificación de última hora.

También será necesario editar manualmente un componente insertado desde un archivo de Altium si no se crea una entrada para éste en el archivo de geometrías. En el momento de la interpretación del archivo de pick & place de Altium, la aplicación asigna a todos los componentes desconocidos un alimentador genérico, creado automáticamente para evitar inconsistencias en los datos. En este caso el usuario deberá indicar, con el editor de componentes, el alimentador real que corresponde a cada uno de estos componentes.

En los casos en que se desee introducir los datos del diseño manualmente, el editor de parámetros de los componentes también es imprescindible, puesto que los datos que se editan aquí definen de qué alimentador hay que coger cada componente y los detalles de su colocación sobre la placa.

Desde el editor de componentes, que se puede ver en la figura 111, se pueden modificar los datos de los componentes de la placa o crear componentes nuevos. El funcionamiento de este editor es similar al del editor de placas o el de alimentadores. A la izquierda se encuentra el listado de componentes de la placa que esté seleccionada en la parte superior de la ventana, mientras que a la derecha se pueden ver y editar los parámetros del componente seleccionado en dicho listado.

Al igual que en el resto de editores, al crear un nuevo componente, éste se añadirá al final del vector de componentes en memoria y por tanto inicialmente aparecerá al final de la lista de componentes.

En el módulo de producción, los componentes se montarán en el orden en que están en el vector de componentes, o sea, en el orden mostrado en el listado. Si se desea cambiar el orden de

montaje de los componentes se puede utilizar los botones que hay debajo del listado para subir o bajar el componente seleccionado u ordenarlos automáticamente con el botón *Ordenar*.

En caso de que se pulse el botón *Ordenar* se mostrará un formulario modal que permitirá el ordenamiento automático de los componentes de la placa actual, en orden ascendente o descendente y por hasta tres criterios diferentes en orden.

Además, el editor de componentes permite modificar cualquiera de los campos de la estructura de datos del componente, incluso el estado de montaje en el que se encuentra. De esta forma se ofrece al operador la máxima flexibilidad para adaptar el programa de montaje a cualquier necesidad mientras se prepara una placa.

El comportamiento frente a errores en el nombre y en los datos numéricos es similar al caso del editor de alimentadores, puesto que las necesidades son las mismas. En la figura 114 se muestra a la izquierda el editor de componentes con el campo "*Nombre*" del componente vacío y a la derecha un error en la coordenada *x* del componente que hace que este caiga parcialmente fuera de la placa. Al estar su centro en la coordenada 0, parte de su cuerpo sobresaldrá por la izquierda de la placa.

Como en el caso del editor de alimentadores, un error en el nombre no permitirá guardar los cambios, pero si se permite guardar con errores numéricos. Así se facilita la tarea del operador mientras se realizan los ajustes aunque, lógicamente, no se podrá montar la placa con la Cosy hasta que sean subsanados.

Además, en caso de que algún componente tenga datos erróneos, su nombre en el listado estará marcado con el texto "(ERROR)" a la derecha, como sucedía en la edición de placas con el nombre de las placas en su listado. Adicionalmente, sobre sus parámetros aparece en rojo el texto "El componente tiene errores".

Si un componente tiene errores, su placa indicará error también, aunque el error no esté en los parámetros físicos de la placa. De esta manera se ayuda al operador a localizar las placas con errores y los motivos del error cuando los listados contengan múltiples placas con multitud de componentes en cada una de ellas.

Así, si existe un error en un componente de una placa que no es la placa que se está visualizando, se sabrá inmediatamente del problema al ver el error en el listado de placas. Al pulsar sobre la placa, si ésta tiene tantos componentes que no caben todos en la parte visible y la indicación del componente erróneo está en la parte no visible del listado, se verá el mensaje en

rojo “La placa contiene errores”. Sin embargo, ningún campo de la placa estará en rojo, deduciéndose inmediatamente que es un componente el causante del error. Al desplazar el listado de componentes hacia abajo, se encontrará rápidamente el componente erróneo gracias al texto al lado de su nombre. Al pulsar sobre el componente aparecerá en rojo el texto “El componente tiene errores” y los campos marcados en rojo también, habiéndose localizado así rápidamente la causa del problema.

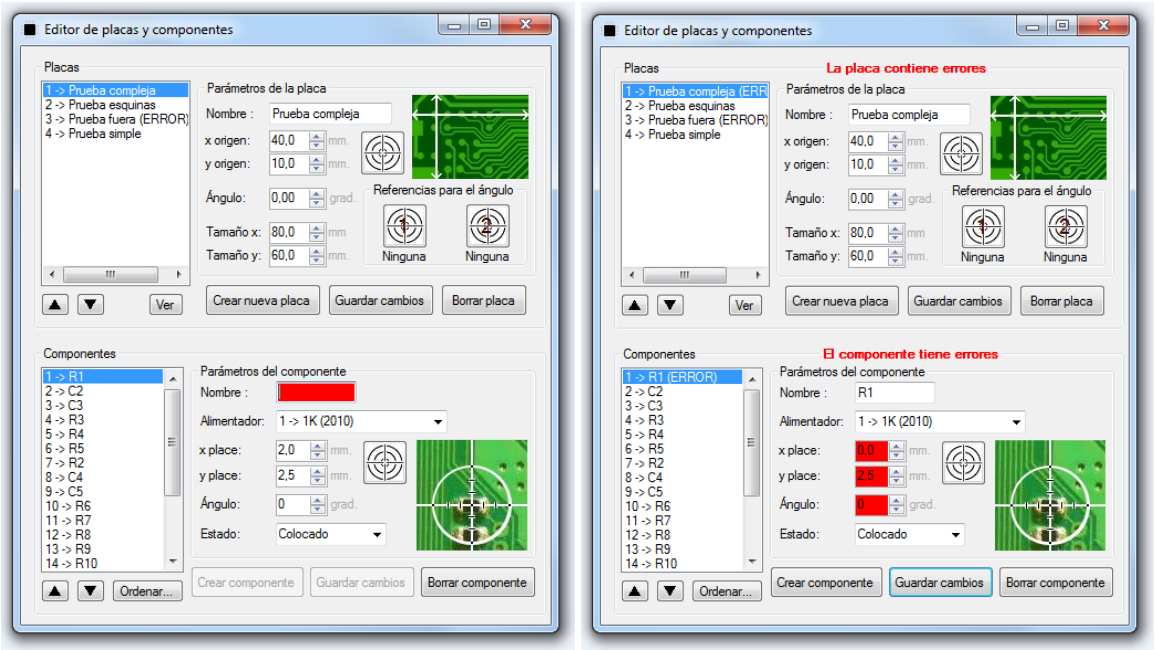


Figura 114. Formulario Editor de placas y componentes con dos tipos de error.

Para seleccionar la posición de colocación de los componentes se puede usar la mira de la cámara al igual que en ocasiones anteriores. El código que permite hacer esto es similar al explicado anteriormente, por lo que no se repetirá aquí.

Cuando se utiliza la mira de la cámara para seleccionar las coordenadas de colocación de un componente, tiene en cuenta automáticamente que las coordenadas se han de tomar con referencia a la posición de la placa. Por tanto la aplicación no deja seleccionar coordenadas menores que cero o mayores que las dimensiones de la placa, pues caerían fuera de ésta. Esto carece de importancia para el usuario, puesto que éste se limitará a apuntar con la mira al lugar de la placa donde desea colocar el componente.

Si el usuario intenta colocar un componente más allá de los límites de la placa, verá que las coordenadas de este podrán variar hasta que tenga su centro en el borde, no permitiendo aumentar más el valor de esa coordenada (o disminuirlo hacia valores negativos).

Para facilitar los cálculos y las comprobaciones, cuando se desea establecer la posición de un componente en una placa utilizando la cámara, se deberá hacer con la placa puesta a 0°. El botón para seleccionar las coordenadas del componente usando la mira de la cámara se desactivará si la placa tiene un ángulo distinto de cero.

En caso de que se desee crear una placa manualmente, que debe montarse girada en el panel, y se quiera utilizar la cámara para especificar las coordenadas de los componentes, se debe hacer todo el proceso con la placa a 0°. Una vez creados sus componentes y especificados sus parámetros, se podrá girar la placa o cambiarla a otra posición libremente.

5.12. Ordenar componentes

Los componentes se montarán en el orden especificado en el listado, por lo que se ha incluido un formulario que permite ordenarlos de forma automática atendiendo a los criterios que se desee. Este formulario se puede ver en la figura 115 y, como se puede apreciar, permite especificar hasta tres criterios de ordenación simultáneamente. En caso de que se quisiese ordenar atendiendo a más de tres criterios a la vez, basta con ejecutarlo dos veces en secuencia, indicando la primera vez los criterios menos relevantes y la segunda vez los criterios más importantes.

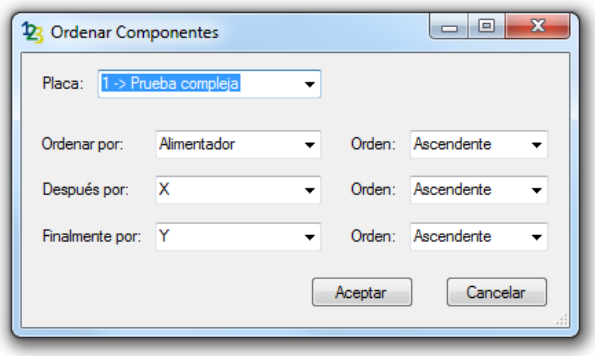


Figura 115. Formulario Ordenar Componentes.

El usuario podrá usar esta opción por ejemplo para montar los componentes agrupados por alimentador. Combinando esta opción con unas posiciones de componentes en los alimentadores bien escogidas, el usuario puede evitar constantes cambios de banco de alimentadores o de útil en la Cosy. De esta manera se reduciría la cantidad de operaciones a realizar durante un programa de montaje. Teniendo en cuenta que la operación de cambio de banco es la más lenta de todas las que es capaz de realizar la Cosy, con mucha diferencia, se lograrán grandes ahorros de tiempo en caso de usar más de un banco. En cuanto a las operaciones de montaje y desmontaje de los útiles de la boquilla, son operaciones rápidas, pero

minimizarlas tiene la ventaja (además del ahorro de tiempo) de disminuir el desgaste, aumentando su vida útil.

Los componentes de cada placa se almacenan en un array, y para ordenar sus elementos, la clase Array posee un método *Sort*. Sin embargo, el método *Sort* proporcionado por el framework sólo puede ordenar los tipos simples nativos, por lo que no puede ordenar un tipo complejo como la estructura *componente* que hemos diseñado. Para solucionar esto se ha creado una función de ordenación propia. Esta función de ordenación es inyectada al método Sort de la clase Array por medio de un delegado, como se puede apreciar en el código del método de ordenación mostrado en la figura 116.

```
private void ordenar()
{
    Array.Sort(datos.placas[comboBoxPlaca.SelectedIndex].componentes,
        delegate(componente c1, componente c2)
        {
            int resultado = comparar(c1, c2, comboBoxCampo1.SelectedIndex,
                                comboBoxOrden1.SelectedIndex);
            if (resultado == 0 && comboBoxCampo2.SelectedIndex > 0)
            {
                resultado = comparar(c1, c2, comboBoxCampo2.SelectedIndex - 1,
                                comboBoxOrden2.SelectedIndex);
            }
            if (resultado == 0 && comboBoxCampo3.SelectedIndex > 0)
            {
                resultado = comparar(c1, c2, comboBoxCampo3.SelectedIndex - 1,
                                comboBoxOrden3.SelectedIndex);
            }
            return resultado;
        });
}

private int comparar(componente c1, componente c2, int factor, int orden)
{
    int[] comparación = new int[6];
    comparación[0] = c1.alimentador.CompareTo(c2.alimentador);
    comparación[1] = c1.nombre.CompareTo(c2.nombre);
    comparación[2] = c1.x.CompareTo(c2.x);
    comparación[3] = c1.y.CompareTo(c2.y);
    comparación[4] = c1.t.CompareTo(c2.t);
    comparación[5] = c1.estado.CompareTo(c2.estado);
    return comparación[factor] * (orden == 0 ? 1 : -1);
}
```

Figura 116. Código fuente de los métodos de ordenación.

Nuestra función de ordenación se apoya en el método *comparar*, mostrado en la misma figura, que le indica el orden de los dos elementos comparados, según el factor de ordenación escogido y si se desea un orden ascendente o descendente.

5.13. Visor/editor gráfico de componentes

5.13.1. Descripción general

El editor gráfico de componentes permite ver la distribución de componentes en una placa y cambiar algunos de los parámetros relativos a la colocación de éstos de forma más intuitiva que el editor normal. Para ello dibuja una representación gráfica de la placa con sus componentes y permite ver los parámetros de colocación del componente al pasar el ratón sobre el nombre del mismo. Además, el componente también se puede arrastrar a otra ubicación utilizando el ratón o girarlo con la rueda central.

Si se dispone de archivos Gerber de la placa que se va a montar, se pueden asociar a la ésta para que el editor gráfico muestre la imagen detrás de los componentes que se están editando. Esto hace mucho más cómoda la tarea y permite trabajar más rápido a la hora de verificar la placa o corregir posibles errores.

Al editor gráfico de componentes se puede acceder desde el menú placas. En el submenú "*Ver placa*" aparecerán una serie de opciones, una por cada placa, que identifica la placa que se quiere ver con su número y nombre. Estas opciones se van modificando cada vez que cambian las placas almacenadas en memoria. También se puede acceder desde el *Editor de placas y componentes* con el botón "*Ver*" que hay debajo del listado de placas. En este caso se abriría la placa seleccionada en el listado. En la figura 117 se muestra la ubicación de ambas opciones.

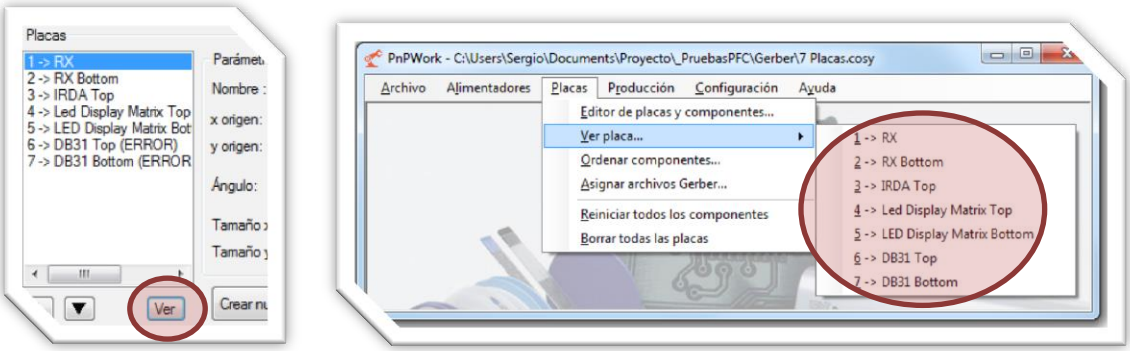


Figura 117. Detalles de las dos formas de acceder al editor gráfico.

En la figura 118 se puede ver el formulario de edición de componentes mostrando una placa de ejemplo. Esta placa de ejemplo se ha creado desde cero con el programa, introduciendo manualmente cada componente, por lo que no dispone de archivos Gerber asociados y se muestran los componentes contra un fondo negro.

Como se puede observar, en la representación gráfica se utilizan diferentes colores para representar los componentes según el estado de tratamiento en el que estén. El relleno de color de los componentes es semitransparente, por lo que se puede ver a través de él lo que hay debajo, como puede ser otros componentes o la imagen de la placa creada a partir de los archivos Gerber. En el ejemplo de la figura 118 se puede apreciar que al ser semitransparente se mezcla con el color de fondo (negro), oscureciéndose en este caso. La línea del lado norte (superior) del componente se dibuja más gruesa para identificar rápidamente si está girado.

Muchos de los estados por los que puede pasar un componente sólo tienen sentido cuando se monta la placa usando la máquina pick & place Cosy. En esta placa de ejemplo se ha editado manualmente el campo estado de varios componentes para que se muestren todas las posibilidades disponibles, que son las siguientes:

- Amarillo claro: Se utiliza para representar componentes que aún no se han tratado.
- Azul: El componente se está cogiendo o se están haciendo los preparativos previos para hacerlo.
- Violeta: El componente ha sido cogido del alimentador con éxito y se está colocando o se están haciendo los preparativos previos para hacerlo.
- Verde: El componente ha sido colocado correctamente.
- Rojo: El componente ha fallado y, tras un número de reintentos fallidos, ha dado error al cogerlo y se desistió de tratarlo.
- Naranja: El componente ha fallado y, tras un número de reintentos fallidos, ha dado error al colocarlo y se desistió de tratarlo.

Lo normal es que en una placa sin procesar todos los componentes se muestren de color amarillo claro. En una placa procesada sin errores todos los componentes serán verdes. Si la placa procesada tuvo errores, además del verde se pueden ver componentes de color rojo o naranja. En una placa con el programa de montaje en mitad del proceso pueden aparecer componentes en cualquier color. Sin embargo, los colores azul y violeta no se verán simultáneamente en condiciones normales, puesto que la máquina procesa los componentes de uno en uno y no puede estar cogiendo un componente al mismo tiempo que coloca otro.

En la figura 118 se puede apreciar que en la barra de título se muestra el número y nombre de la placa, además de sus medidas. En la parte inferior de la ventana hay una barra de

estado que muestra las coordenadas de la placa en las que se encuentra el ratón, expresadas en milímetros.

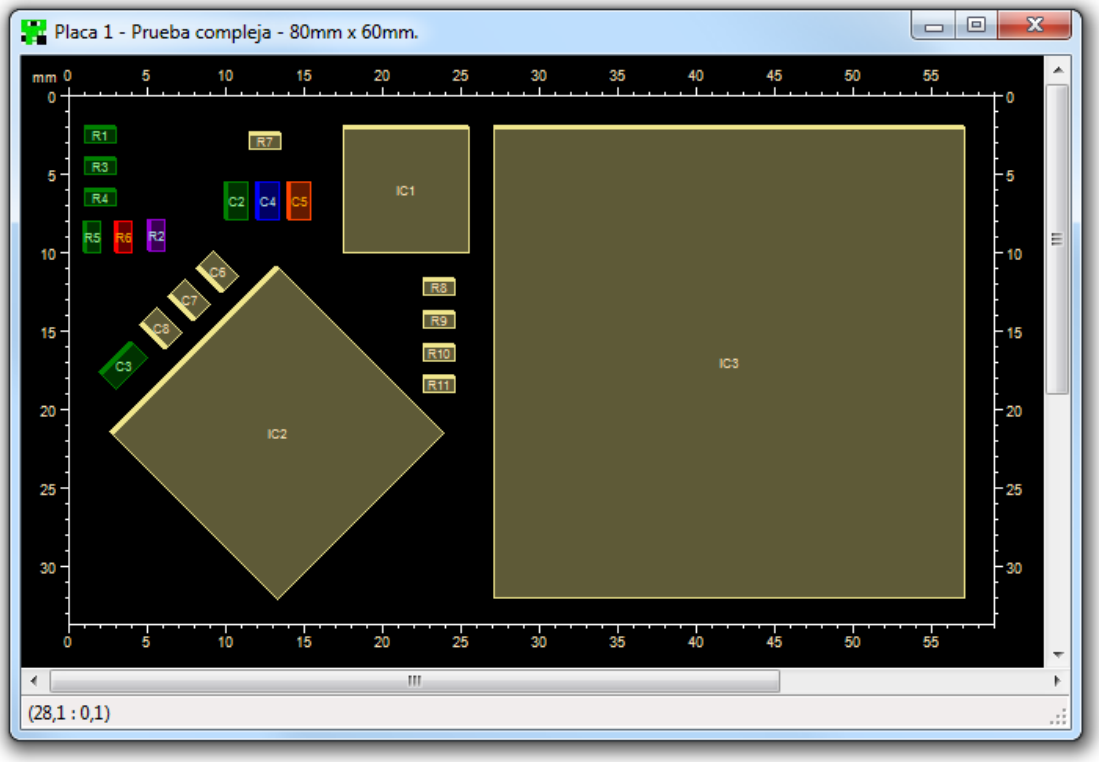


Figura 118. Formulario del editor gráfico de componentes.

También se puede ver que la representación incluye una escala graduada en milímetros para ayudar a interpretar la ubicación de los componentes. Además, en caso de que la placa sea más ancha o alta que la ventana, aparecerán automáticamente barras de desplazamiento horizontal o vertical respectivamente. Cuando se desplaza la placa dentro de la ventana la escala la sigue, de forma que siempre muestra la ubicación correcta de los componentes.

El desplazamiento de la placa en la ventana se podrá realizar utilizando las barras de desplazamiento, de la forma habitual en el entorno de Windows, o también haciendo clic y arrastrando con el ratón la placa que se muestra en la ventana. La acción de arrastrar con el ratón sirve tanto para desplazar un componente como para desplazar la vista sobre la placa, dependiendo de si al arrastrar se hace clic sobre el nombre de un componente o sobre cualquier otra parte del contenido de la ventana respectivamente.

Para ayudar a evitar errores, cuando el cursor del ratón pasa por encima del nombre de un componente, su imagen cambia a una mano (normalmente es la flecha de Windows) y se muestra un pequeño bocadillo (tooltip) con los parámetros de ese componente, como se puede

ver en la figura 119. De esta manera se indica que si se actúa sobre los botones del ratón se van a alterar las propiedades de ese componente.

Mientras se está desplazando la placa por la ventana, el cursor del ratón se cambia por otro cursor con cuatro flechas hacia arriba, abajo, izquierda y derecha, que indica que se está moviendo la vista de la placa en la ventana.

Además, se puede desplazar la vista utilizando la rueda del ratón, que en esta ventana pasa a tener cuatro funciones. Si se gira la rueda del ratón estando el cursor sobre el nombre de un componente se girará el componente. Si se gira sobre cualquier otra parte de la placa, se desplazará la placa verticalmente, pero si no hay barra de desplazamiento vertical porque el alto de la placa se ve por completo en la ventana, entonces la rueda pasará automáticamente a utilizarse para el desplazamiento horizontal.

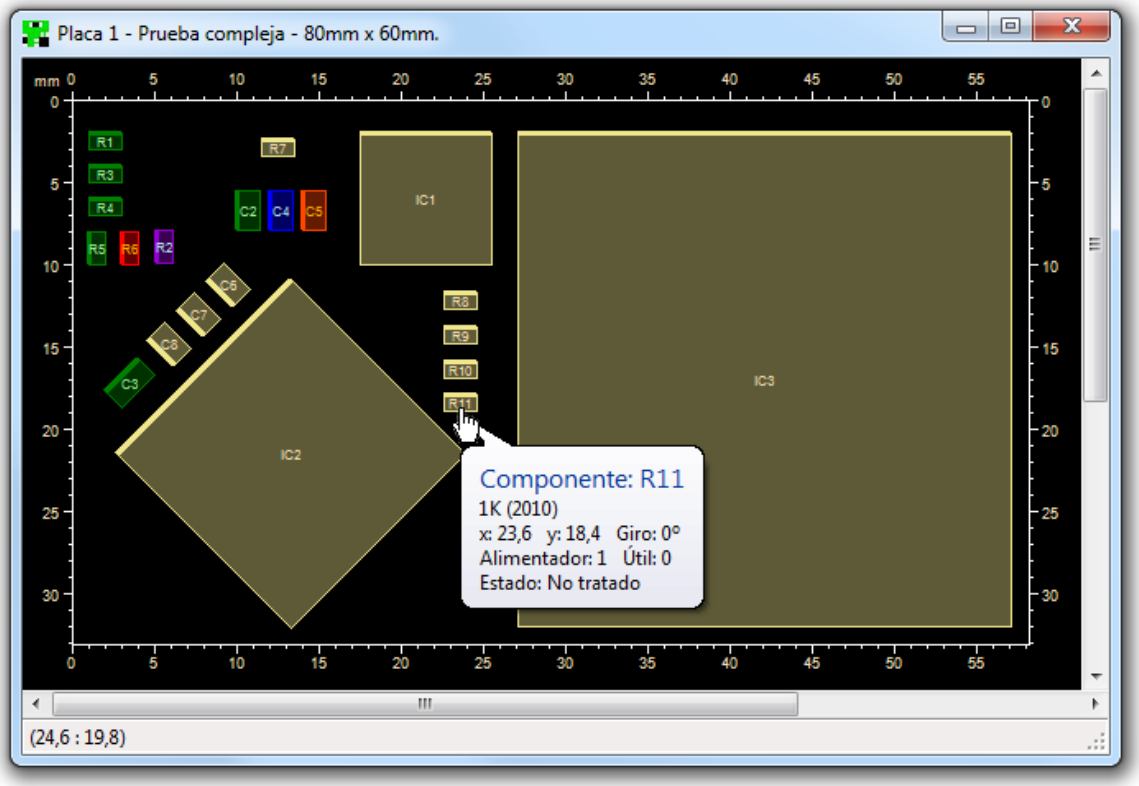


Figura 119. Editor gráfico mostrando los detalles de un componente.

En caso de que la placa tenga sus archivos Gerber asociados, se mostrará la imagen de la placa detrás de los componentes como se puede ver en la figura 120. En esta imagen de ejemplo se puede ver que se sólo se han dejado los componentes SMD para esta etapa del montaje y se han eliminado los componentes THT.

En caso de mostrarse una imagen Gerber, si se hace clic y se arrastra con el botón de la rueda del ratón (botón central) se desplazará la imagen Gerber que se muestra detrás de los componentes. De esta forma se puede ajustar visualmente el desplazamiento de la imagen si fuese necesario.

Si se hace clic con el botón derecho sobre el nombre de un componente se le cambiará de estado. Si estaba en estado "*No tratado*" pasará al estado "*Colocado*", si estaba en cualquier otro estado, pasará al estado "*No tratado*".

Cuando se usa el editor gráfico de componentes para cambiar de estado, mover o girar un componente, éste lo comunica inmediatamente al formulario editor de placas y componentes, en caso de que esté abierto, de forma que los cambios se reflejan inmediatamente en él. En ese caso se selecciona automáticamente la placa y el componente que se editó en el formulario del editor gráfico y se actualizan los datos mostrados. En caso de que los cambios de posición o giro provoquen un error se reflejará en el formulario editor de placas y componentes tal como se ha explicado anteriormente.

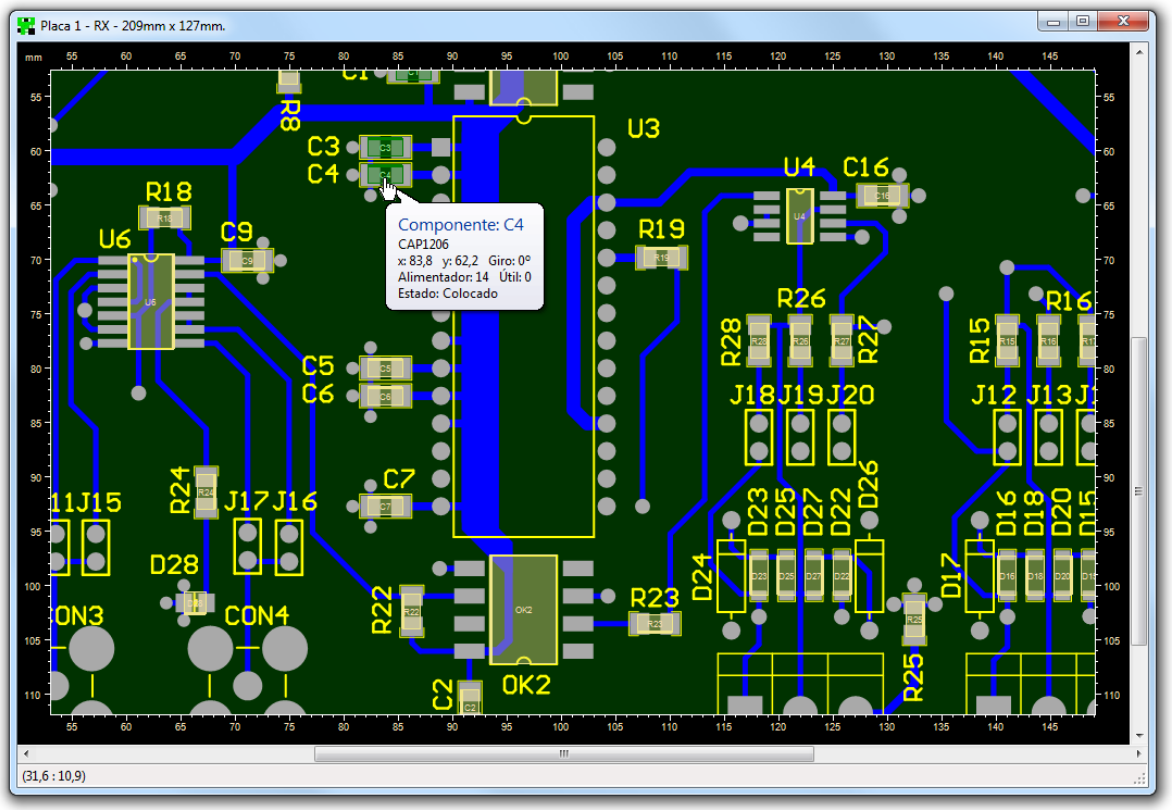


Figura 120. Editor gráfico mostrando detalles de un componente en una placa con archivos Gerber asociados.

Además, en caso de que se mueva un componente fuera de los límites de la placa se mostrará en la barra de título un mensaje, a la derecha del texto existente, que avisa de que la

placa tiene errores. Este mensaje también se mostrará si la placa tiene errores de posición o dimensiones que la hagan salirse de los límites del panel de trabajo.

Todas estas posibilidades aumentan la manejabilidad de la aplicación mediante una mayor agilidad de movimientos durante la edición o visualización de una placa, aunque ha sido a costa de una mayor complicación del código creado para el formulario.

Para utilizar los eventos producidos por la rueda del ratón es necesario registrar el método que hemos creado para manejar dichos eventos. Sin embargo, como en el panel de selección de eventos de Visual Studio no aparece explícitamente el evento *MouseWheel* (que es el que se necesita para responder al giro de la rueda del ratón) el entorno de desarrollo no realizará el registro automáticamente por nosotros mediante su código autogenerado. Por tanto, se ha incluido la línea mostrada en la figura 121 en el código del constructor de la clase para registrarlo manualmente.

El editor gráfico de componentes se puede usar como representación gráfica del proceso de montaje de la placa. Si se activa la opción "*Ver producción*" en la ventana de producción (figura 148), se mostrará en primer plano la representación gráfica de la placa que se está montando y se van cambiando los colores de los componentes, en tiempo real mientras la máquina los va tratando, para reflejar el estado en que están.

```
this.MouseWheel += new MouseEventHandler(FormPlaca_MouseWheel);
```

Figura 121. Registro del evento MouseWheel.

Durante el montaje, la edición queda deshabilitada, por lo que no se pueden hacer cambios y este formulario queda como un simple visor de lo que le está pasando a los componentes sobre la placa. El *checkBox* "*Ver producción*" se puede activar o desactivar en cualquier momento, antes o durante la producción.

5.13.2. Dibujo de la placa

El principal método de esta clase es el que realiza el dibujo de la placa sobre el fondo del formulario vacío, cuyo código se puede ver en la figura 122. El formulario ya almacena un campo con el número de la placa que debe dibujar, por lo que el método no recibe parámetros.

Lo primero que hace el método es comprobar que no va a tratar de dibujar sobre un formulario cerrado, ya que si lo hiciese se provocaría una excepción no gestionada, que abortaría el programa.

```
private void dibujarPlaca()
{
    if (this.IsDisposed) return;
    Color colorComponente;
    System.Drawing.Brush pincelTexto;

    // Pone el texto de la barra de título.
    this.Text = "Placa " + (numplaca + 1).ToString() + " - " +
                datos.placas[numplaca].nombre + " - " + Convert.ToString(p.anchoX / 10) +
                "mm x " + Convert.ToString(p.anchoY / 10) + "mm.";
    if (!datos.placaOK(numplaca)) this.Text += "  ATENCIÓN: Placa con errores";

    // Ajusta las barras de desplazamiento, evitando que las reglas se salgan de las
    // dimensiones de la placa cuando se redimensiona con el scroll al final.
    if ((xp0 + anchoVisibleX) > p.anchoX) xp0 = p.anchoX - anchoVisibleX;
    if ((yp0 + anchoVisibleY) > p.anchoY) yp0 = p.anchoY - anchoVisibleY;
    ajusteScrollBars();

    // Crea un lienzo buffer para dibujar con la imagen Gerber de fondo.
    Bitmap imagen = new Bitmap(this.Width, this.Height);
    Graphics lienzo = Graphics.FromImage(imagen);
    lienzo.Clear(colorFondo);
    if (imagenGerber != null) lienzo.DrawImage(imagenGerber, x0 - xp0 –
                datos.placas[numplaca].desplazamientoGerber.X, y0 - yp0 –
                datos.placas[numplaca].desplazamientoGerber.Y);

    // Dibuja los componentes con su nombre.
    for (int i = 0; i < p.componentes.Length; i++)
    {
        switch (p.componentes[i].estado)
        {
            case estadoComponente.Cogiendo:
                colorComponente = Color.Blue;
                pincelTexto = Brushes.LightBlue;
                break;
            case estadoComponente.Colocando:
                colorComponente = Color.DarkViolet;
                pincelTexto = Brushes.LightBlue;
                break;
            case estadoComponente.Colocado:
                colorComponente = Color.Green;
                pincelTexto = Brushes.LightGreen;
                break;
            case estadoComponente.Error_Cogiendo:
                colorComponente = Color.OrangeRed;
                pincelTexto = Brushes.Orange;
                break;
            case estadoComponente.Error_Colocando:
                colorComponente = Color.Red;
                pincelTexto = Brushes.Orange;
                break;
            default:    // Componente no tratado
                colorComponente = Color.Khaki;
                pincelTexto = Brushes.Wheat;
                break;
        }
        lienzo.DrawPolygon(new Pen(colorComponente),
                p.componentes[i].vértices(x0 - xp0, y0 - yp0));
        Color colorRelleno = Color.FromArgb(transparenciaRelleno, colorComponente);
        lienzo.FillPolygon(new SolidBrush(colorRelleno), p.componentes[i].vértices(x0 - xp0,
                y0 - yp0));
        lienzo.DrawLine(new Pen(colorComponente, 3), p.componentes[i].vértices(x0 - xp0,
                y0 - yp0)[0], p.componentes[i].vértices(x0 - xp0, y0 - yp0)[1]);
        tamañoTexto = lienzo.MeasureString(p.componentes[i].nombre, fuenteComponentes);
        lienzo.DrawString(p.componentes[i].nombre, fuenteComponentes, pincelTexto,
                new Point((int)(p.componentes[i].x + x0 - xp0 - tamañoTexto.Width / 3),
                (int)(p.componentes[i].y + y0 - yp0 - tamañoTexto.Height / 3)));
    }

    // Borra los bordes para que no se vean los componentes que sobresalen.
    lienzo.FillRectangle(Brushes.Black, 0, 0, this.Width, y0);
    lienzo.FillRectangle(Brushes.Black, 0, y0, x0, anchoVisibleY);
    lienzo.FillRectangle(Brushes.Black, 0, y0+anchoVisibleY, this.Width,
```

```
                this.Height-y0-anchoVisibleY);
    lienzo.FillRectangle(Brushes.Black, x0+anchoVisibleX, y0,
                this.Width-x0-anchoVisibleX, anchoVisibleY);

    // Dibuja las 4 escalas y el marco.
    dibujarEscala(lienzo, x0, y0, anchoVisibleX, xp0, 'u');
    dibujarEscala(lienzo, x0, y0, anchoVisibleY, yp0, 'l');
    dibujarEscala(lienzo, x0, y0 + anchoVisibleY, anchoVisibleX, xp0, 'd');
    dibujarEscala(lienzo, x0 + anchoVisibleX, y0, anchoVisibleY, yp0, 'r');
    lienzo.DrawRectangle(Pens.White, x0, y0, anchoVisibleX, anchoVisibleY);
    tamañoTexto = lienzo.MeasureString("mm", fuenteEscalas);
    lienzo.DrawString("mm", fuenteEscalas, pincelRótulosEscalas, x0 - tamañoTexto.Width - 5,
                y0 - tamañoTexto.Height - 5);

    // Copia el dibujo hecho en el buffer en la ventana para que se vea y libera recursos.
    Graphics g = this.CreateGraphics();
    g.DrawImage(imagen, 0, 0);
    lienzo.Dispose();
    imagen.Dispose();
    g.Dispose();
}
```

Figura 122. Código fuente del método *dibujarPlaca*.

A continuación se pone el texto en la barra de título con los datos de la placa. Posteriormente se ajustan las barras de desplazamiento, pues puede que la operación de dibujo de la placa se haya disparado debido a que se cambió el tamaño del formulario. De esta forma, el tamaño de las barras y sus botones reflejará la proporción entre la parte de la placa vista y la parte oculta. Se tiene en cuenta en esta operación evitar que las reglas se salgan de las dimensiones de la placa, ajustando el final de la ventana con el final de la placa.

A continuación se crea un nuevo lienzo para dibujar basado en una imagen de las mismas dimensiones que la ventana. Esta técnica se llama de “doble *buffer*” y es bastante conocida en el mundo de la programación gráfica para evitar el parpadeo del contenido de la ventana. La técnica se basa en evitar dibujar directamente sobre la memoria de pantalla, haciéndolo sobre una imagen tampón (*buffer*). Cuando se termina el dibujo, se copia la imagen tampón sobre el buffer real de la memoria de pantalla, reduciéndose así el número de operaciones de dibujo sobre la pantalla a sólo una.

De esta forma, cuando el hardware de la tarjeta gráfica accede a la memoria gráfica para refrescar la imagen del monitor siempre encuentra dibujos terminados, nunca dibujos a medio hacer.

Con esto que se evita que algunas veces lo dibuje entero y otras sólo hecho parcialmente, con el resto en negro. La repetición aleatoria de estos patrones de dibujo, a veces completos, a veces sólo parcialmente completados es la que produce el molesto efecto de parpadeo cuando no se usa esta ingeniosa técnica.

Tras crear el buffer se comprueba si existe imagen Gerber de esta placa y se dibuja sobre el fondo negro del buffer. El método de entrada de este formulario ya se habrá encargado de generar esta imagen si existen archivos Gerber asociados a la placa. A continuación se dibuja cada componente de la placa en el buffer, eligiendo los colores de la línea y del texto del nombre en función de su estado. La línea norte de cada componente se redibuja con un trazo ligeramente más grueso para destacarla. Cuando termina con todos los componentes, borra los bordes para que no se vea la parte de los componentes que sobresalen del área visible. Si hubiese alguno, esto hará que sólo se vea parcialmente.

La última operación de dibujo es la de las escalas y el marco que rodea al área visible. El dibujo de las escalas no es trivial, por lo que se ha separado esta parte del dibujo en el método *dibujarEscala*, que es capaz de dibujar una escala según varios parámetros y se verá más adelante.

Como se ha decidido dibujar escalas en los cuatro bordes del área visible, durante el dibujo de la placa se llama cuatro veces a este método, una para cada borde, cambiando los parámetros de entrada adecuadamente para cada caso.

Para terminar la representación gráfica de los componentes de la placa, se completa la segunda parte del método del doble buffer, copiando el dibujo realizado en la imagen temporal sobre la ventana.

Tras terminar, se indica al recolector de basura que puede liberar la memoria de los buffers. Aunque el funcionamiento del recolector de basura es automático y no es necesario indicarle cuando operar, pues el sistema lo ejecuta en su propio hilo, de forma paralela a nuestra aplicación, se ha decidido indicarle que puede liberar la memoria ocupada por estos recursos en particular porque se trata de recursos que podrían llegar a consumir bastante memoria.

La operación de dibujo de una placa se puede llegar a hacer muchas veces por segundo si se está desplazando la vista o moviendo un componente y, en caso de trabajar a pantalla completa con una placa grande, cada buffer podría ocupar varios megabytes de memoria.

Si el recolector de basura tardase en percatarse de que estos objetos ya no son necesarios, podrían acumularse y verse forzado a recuperarlos de golpe si empieza a escasear la memoria en un ordenador de bajos recursos. En este caso, el rendimiento podría verse afectado, por lo que se le ha indicado aquí que puede liberar la memoria ocupada por éstos lo antes posible.

5.13.3. Dibujo de las escalas

Para dibujar las escalas se ha implementado un método dentro de la clase, que realiza el dibujo genérico de las líneas de graduación en el lienzo que se le proporcione como parámetro. Como se puede ver en el código del método, mostrado en la figura 123, este método debe recibir también las coordenadas en las que empieza la escala y la longitud de ésta, el valor inicial de la escala y un último parámetro que indica la posición de la escala.

El parámetro de posición define si la escala es la de encima del gráfico, debajo, a la izquierda o a la derecha, según los siguientes valores:

- u: Escala superior
- d: Escala inferior
- l: escala izquierda
- r: escala derecha

Con este dato y algunas constantes que han sido predefinidas, el método determina si tiene que dibujar la escala en horizontal o en vertical y la posición de las líneas de graduación y rótulos numéricos.

En lugar de definir una forma diferente de dibujar para cada una de las cuatro escalas se ha realizado de una única manera, calculando previamente el valor de unos factores de incremento y desplazamiento vertical y horizontal, utilizando para ello el parámetro de posición. Con estos factores se calcula las coordenadas donde dibujar las líneas de la escala y donde escribir el texto de los rótulos con unas simples sumas y multiplicaciones, como se puede ver en el código comentado.

Hay que tener en cuenta además que, si la escala no comienza en una línea de división, habrá que calcular el desplazamiento desde el origen de dibujo de la escala hasta la primera línea de división que haya que dibujar. Este desplazamiento se sumará a las coordenadas calculadas para las líneas de escala, lo que se ha tenido en cuenta iniciando el bucle en el valor de este desplazamiento.

La fuente y el pincel de los rótulos, así como la pluma para dibujar las escalas están definidos globalmente para permitir su uso también por otros métodos.

```
private void dibujarEscala(Graphics lienzo, int xr0, int yr0, int longitud, int valorInicial,
                string posición)
{
    // Dibuja las líneas de graduación de las escalas y las rotula.

    // La posición de las líneas y el texto será según el valor del parámetro “posición”:
    //  u = encima
    //  d = debajo
    //  l = izquierda
    //  r = derecha

    const int divisionMenor = 10;      // Separación entre divisiones menores de la escala.
    const int divisionMayor = 50;      // Separación entre divisiones numeradas de la escala.

    int x, y;
    float xRótulo, yRótulo;
    string rótulo;

    // Desplazamiento de la primera división. Valor en el rango [0..divisionMenor-1]
    int desplazamientoInicial = divisionMenor - valorInicial % divisionMenor;
    // Factor de incremento horizontal. Vale 1 para escalas horizontales.
    int incX = (posición == "u" || posición == "d") ? 1 : 0;
    // Factor de incremento vertical. Vale 1 para escalas verticales.
    int incY = (posición == "l" || posición == "r") ? 1 : 0;
    // Posición de los rótulos encima (-1) debajo (1) o a nivel (0) de la escala.
    int arrAba = posición == "u" ? -1 : posición == "d" ? 1 : 0;
    // Posición de los rótulos a izquierda (-1), derecha (1) o a nivel (0) de la escala.
    int izqDer = posición == "l" ? -1 : posición == "r" ? 1 : 0;

    for (int n = desplazamientoInicial ; n <= longitud ; n += divisionMenor)
    {
        // Calcula las coordenadas de la siguiente línea de la escala en función de los
        // factores de incremento horizontal y vertical.
        x = xr0 + n * incX;
        y = yr0 + n * incY;

        if (((valorInicial+n) % divisionMayor) != 0)
        {
            // Dibuja la línea de las divisiones menores.
            lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 2, y + arrAba * 2);
        }
        else
        {
            // Dibuja la línea de las divisiones mayores.
            lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 5, y + arrAba * 5);
            // Crea el rótulo de las divisiones mayores.
            rótulo = Convert.ToString((xp0 * incX + yp0 * incY + n) / 10);
            // Mide el tamaño que tendrá el rótulo.
            tamañoTexto = lienzo.MeasureString(rótulo, fuenteEscalas);
            // Calcula las coordenadas de escritura del rótulo.
            xRótulo = x - incX * tamañoTexto.Width / 2 + incY * izqDer * 6 –
                          (posición == "l" ? tamañoTexto.Width : 0);
            yRótulo = y - incY * tamañoTexto.Height / 3 + incX * arrAba * 6 –
                          (posición == "u" ? tamañoTexto.Height : 0);
            // Escribe el rótulo.
            lienzo.DrawString(rótulo, fuenteEscalas, pincelRótulosEscalas, xRótulo, yRótulo);
        }
    }
}
```

Figura 123. Código fuente del método *dibujarEscala*.

5.14. Selección de coordenadas usando la cámara

En el caso del montaje de los componentes usando la máquina pick & place Cosy, la captura de los componentes desde su alimentador y el montaje correcto en las placas depende de que se hayan especificado sus coordenadas con bastante precisión. Tanto en el formulario del

editor de alimentadores, como en el editor de placas y componentes hay varios campos en los que especificar las coordenadas x e y de un punto.

Para facilitar la tarea de localizar las coordenadas exactas que se deben introducir en estos campos se ha añadido un formulario auxiliar que permite buscar las coordenadas de un punto usando la cámara incorporada en el brazo como guía.

Para usar la cámara para obtener las coordenadas de un punto sobre el área de trabajo hay que tener en cuenta que ésta se halla desplazada con respecto a la boquilla de succión. Este desplazamiento es de 34 milímetros en el eje x y 10 milímetros en el eje y. Como el módulo de vídeo genera una mira para apuntar, dibujando una cruz aproximadamente en el centro de la imagen, sólo nos interesan las zonas que se pueden alcanzar con dicha mira.

El desplazamiento mencionado hace que una parte del área alcanzable por la boquilla no se pueda centrar en la cámara. Al mismo tiempo, este desplazamiento hace que podamos apuntar con la cámara a zonas que el cabezal no puede alcanzar. En la figura 124 se ha ilustrado este efecto, representando en amarillo el área que sólo puede cubrir la boquilla, en azul celeste el área que sólo puede cubrir la cámara y en verde el área que pueden cubrir ambos.

Como se puede comprobar en la imagen, la zona en la que es posible utilizar la cámara para determinar las posiciones de los elementos a montar está limitada al área rectangular delimitada a la izquierda por el offset de la cámara en el eje x, y a la derecha por la coordenada máxima del eje x que se le puede enviar a la máquina. En la parte superior la limita el offset de la cámara en el eje y, mientras que en la parte inferior el límite viene impuesto por la máxima coordenada y que se le puede enviar a la máquina.

Estas características del funcionamiento de la máquina se han tenido en cuenta, para el correcto funcionamiento del programa, en cuatro constantes guardadas como propiedades de la aplicación, que tienen los valores:

- Offset x cámara: 340
- Offset y cámara: 100
- Coordenada x máxima: 3900
- Coordenada y máxima: 2610

En la descripción de la máquina realizada anteriormente, se mencionó que se ajustó el enfoque de la cámara con una solución de compromiso para ver más nítidamente las zonas de interés. Con este enfoque, la posición de la marca de calibración quedó centrada en la mira de la

cámara en las coordenadas (3812, 399), según el esquema de referencia de la Cosy, que en la convención de referencia usada en nuestros ejes se corresponde con la posición del cabezal (88,399). Teniendo en cuenta el desplazamiento de la cámara, para ver la marca de calibración centrada en la cámara nuestra aplicación debe llevar el cabezal a la posición (428,499).

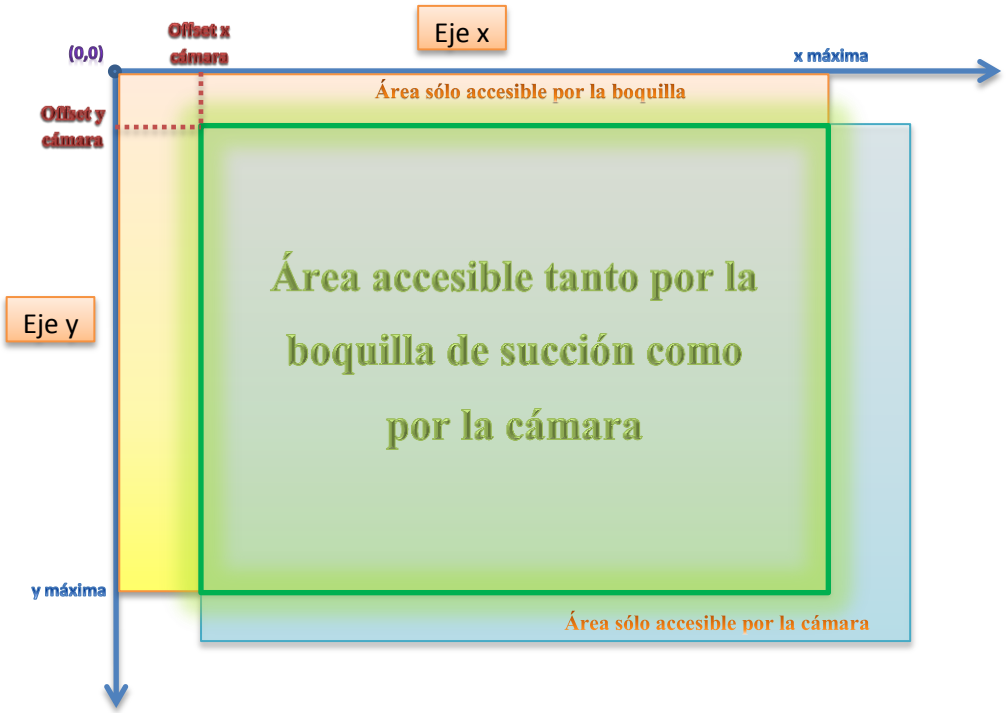


Figura 124. Área accesible por la boquilla y visible por la cámara.

En el formulario que permite seleccionar posiciones utilizando la cámara, que se puede ver en la figura 125, se ha incluido un botón que permite ir directamente a las coordenadas de la marca de calibración. De esta forma se puede comprobar durante su uso que los ejes están calibrados correctamente.

Además, para permitir un posicionamiento más preciso si se trata de especificar la posición de un alimentador, desde este formulario se puede cambiar el banco de alimentadores simplemente seleccionándolo en su control.

Para permitir la máxima flexibilidad, se pueden especificar unas coordenadas exactas a las que mover el cabezal escribiéndolas o bien hacer un ajuste manual mediante el movimiento paso a paso con los botones de dirección. El tamaño del paso se puede seleccionar de 0,1 milímetros (el paso mínimo de la máquina), 1 milímetro, 10 milímetros o bien de 100 milímetros.

Inicialmente se probó con un tamaño de paso variable, definido por el propio usuario mediante un control *numericUpDown*, que le permitía tanto escribirlo como aumentarlo o

disminuirlo con las flechas del control. Sin embargo, se observó que esta forma de trabajar era más confusa y lenta que la selección entre pasos fijos bien definidos, por lo que se optó por ésta última.

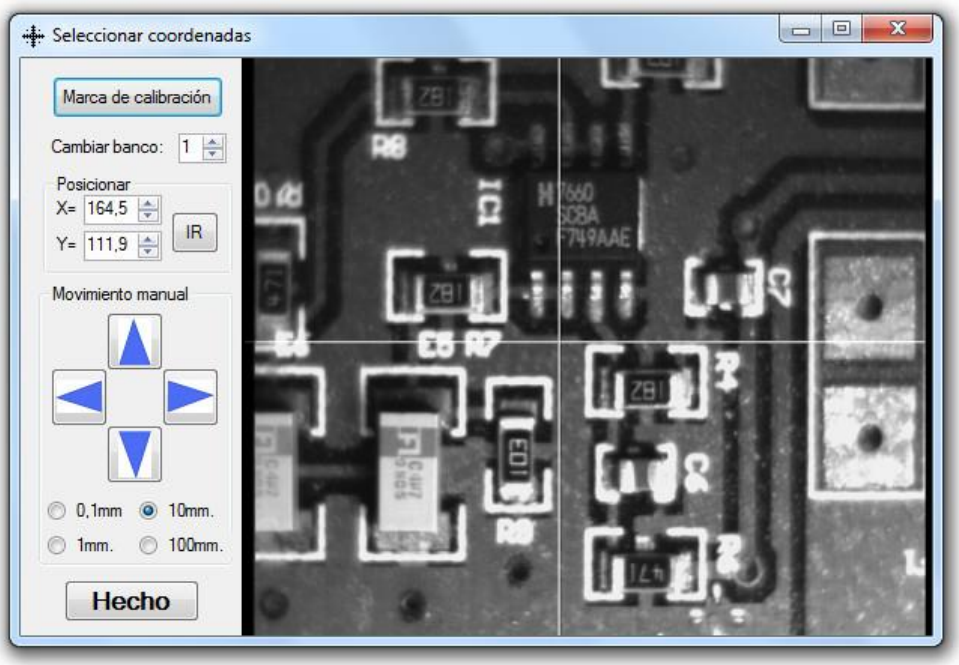


Figura 125. Formulario de selección de coordenadas usando la cámara.

Además de estos métodos, el usuario puede hacer clic en la imagen de la cámara para seleccionar el lugar al que se debe mover el cabezal directamente, que es mucho más cómodo. Sin embargo, según se puede apreciar en la figura 59, la zona visible es bastante limitada (28 mm x 22 mm), por lo que antes de poder seleccionar un punto directamente en la imagen, ésta se deberá aproximar al objetivo. Se puede ir aproximando al objetivo haciendo clics sucesivos sobre la imagen cerca del borde de ésta, en la dirección hacia la que se quiere ir, o usando los métodos alternativos.

La ventana se puede redimensionar libremente por el usuario, provocando que la imagen de la cámara sea escalada automáticamente al tamaño actual de la ventana. De esta manera el usuario puede ampliar la imagen si lo necesita para ver mejor el área donde está trabajando simplemente pulsando el botón de maximizar o tirando de los bordes de la ventana.

Si se pulsa con el botón derecho sobre la imagen de la cámara, se mostrará la ventana de configuración del interfaz de vídeo asociado a la cámara, que normalmente permitirá hacer ajustes de brillo, contraste, etc. En la figura 126 se puede ver las propiedades de configuración disponibles con la digitalizadora de vídeo utilizada durante este proyecto, descrita anteriormente, que en este caso comprende dos paneles. En el primero, se puede elegir el formato de la señal de

vídeo de la cámara que se está capturando y en el segundo es donde podemos hacer los ajustes mencionados.

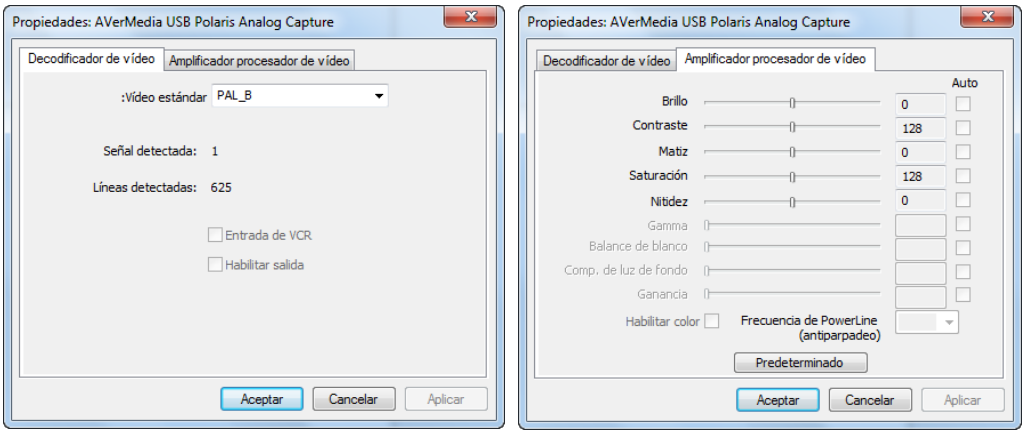


Figura 126. Ajustes de configuración de la cámara.

5.15. Clase Gerber

5.15.1. Introducción

Con el objetivo de ayudar lo más posible al operador a orientarse en las tareas de edición y montaje manual, se decidió mejorar la representación mostrando una imagen de fondo con el dibujo de la placa sobre la que se está trabajando.

Para realizar esta tarea se ha creado una clase capaz de leer los archivos Gerber del diseño de la placa e interpretarlos, generando la imagen correspondiente a la placa. En la figura 127 se puede ver un ejemplo de imagen (en el formulario de montaje manual) obtenida usando la clase *Gerber* desarrollada. Esta clase Gerber se utiliza también para dibujar la placa en el editor gráfico de componentes explicado anteriormente.

Para mantener la complejidad del intérprete de archivos Gerber acotada dentro de unos límites razonables, se hicieron diversas pruebas durante el desarrollo que llevaron a tomar algunas decisiones de optimización. Esto ayudó también a mantener acotado el retardo introducido durante la generación de la imagen.

A la hora de implementar el intérprete para el formato de archivos Gerber, se comprobó que la mayor parte de los ficheros de que se disponía para las pruebas usaban ampliamente algunas características del estándar, mientras que otras no se usaban o eran muy poco usadas. Así, aunque el estándar Gerber especifica multitud de funciones posibles, un análisis visual del

contenido de estos archivos[2] permitió ver que los comandos y procedimientos más usados correspondían a un reducido grupo del total de posibilidades. Esto ayudó a determinar qué partes del estándar se podían obviar para reducir el esfuerzo de la implementación sin que afectara demasiado a la calidad de la imagen obtenida.

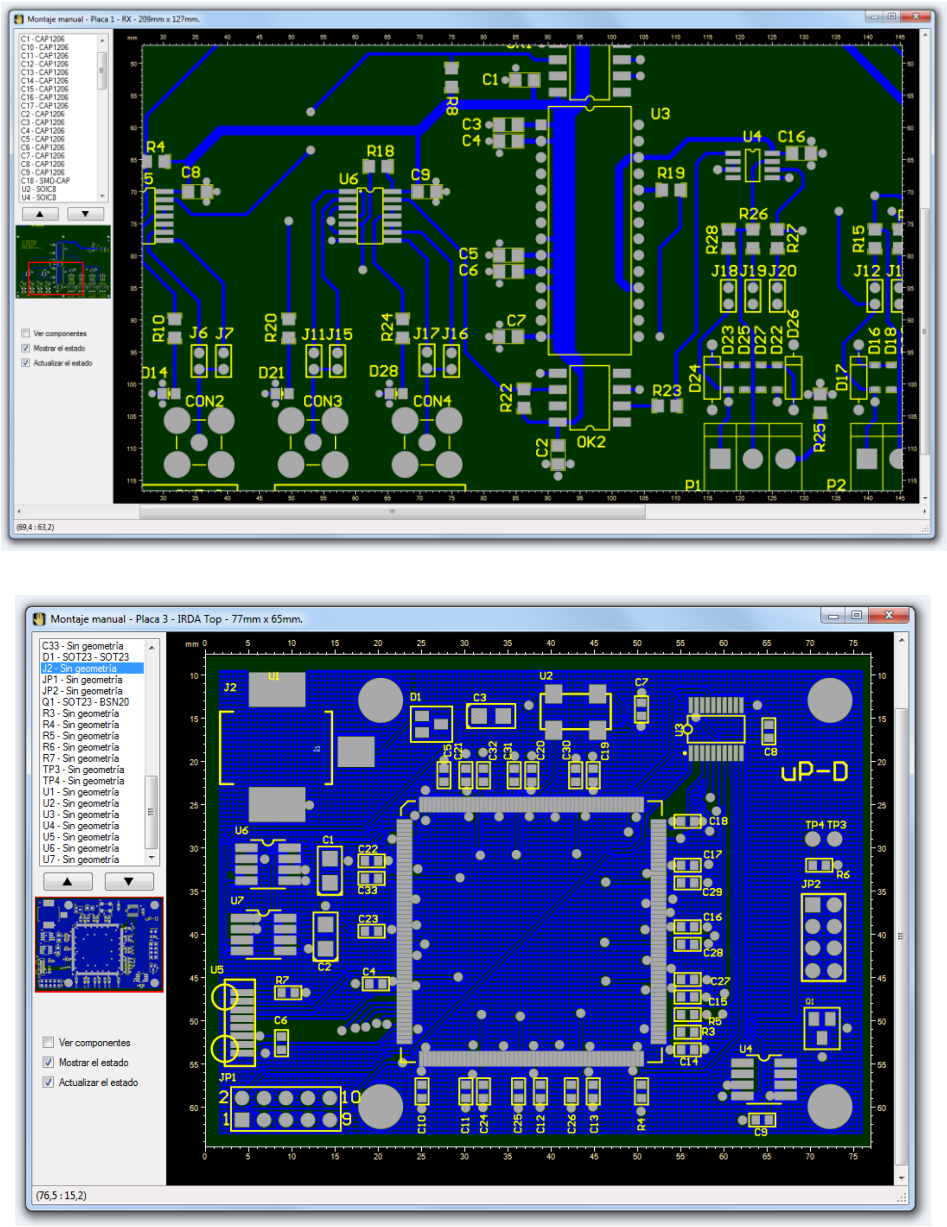


Figura 127. Ejemplos de imágenes creadas a partir de los archivos Gerber.

Hay que tener en cuenta que el motivo principal de la visualización de los archivos Gerber es la ayuda al montaje manual. Es decir, el operador que esté haciendo el montaje utilizará estas imágenes para localizar rápidamente la ubicación del componente con el que está trabajando en

[2] Se trata de archivos de texto, relativamente fáciles de leer cuando se conocen los comandos básicos del estándar.

ese momento. Como no hay otra finalidad más que la ayuda visual, la exactitud en todos y cada uno de los detalles de la imagen no es un factor crítico.

Por tanto colegimos que probablemente no sea necesario implementar todas las funcionalidades del formato Gerber, sólo aquellas usadas más frecuentemente para definir la forma de las pistas y áreas. De esta forma, se simplificó bastante esta tarea y se redujo el tiempo de implementación de esta funcionalidad de la aplicación.

Como es posible que distintos programas generen archivos Gerber de forma diferente, se decidió examinar algunos de los archivos de los que se disponía en el laboratorio. Los ficheros analizados corresponden a placas montadas anteriormente en el SFP y por tanto, constituían un buen ejemplo del tipo de archivos Gerber que tendría que manejar la aplicación habitualmente.

De éste análisis se concluyó que en todos los casos estudiados se puede obtener una representación bastante buena utilizando las funciones DRAW, FLASH y REGION del estándar Gerber con un reducido número de aperturas e implementando únicamente la interpolación lineal.

Por supuesto, hay que implementar algunas características adicionales del formato, que son necesarias para una correcta interpretación del resto de datos, como las aperturas, las unidades, etc. Sin embargo, para nuestro caso, no es necesario añadir algunas de las funciones más complejas, como las aperturas macro o las interpolaciones circulares, puesto que la cantidad de esfuerzo necesario no se compensará con una mejora importante de la imagen obtenida en la gran mayoría de los casos.

Una vez realizada la implementación básica se vio que, con un esfuerzo de implementación moderado, era factible modificar el código para incorporar algunos modos de interpolación circular, por lo que también se implementó esta característica. Esto mejoró notablemente las imágenes, sobre todo para acelerar el trabajo del operador con la orientación correcta de los circuitos integrados, puesto que es habitual que su dibujo en la serigrafía incluya marcas o muescas de orientación trazadas mediante arcos.

5.15.2. Implementación de la clase

La clase *Gerber* se ha creado con dos métodos principales que deben ser utilizados para obtener una imagen a partir de un archivo. En primer lugar se debe llamar al método *interpretarFichero* y en segundo lugar se debe llamar al método *dibujarImagen.* Si bien normalmente sólo se llamará a *interpretarFichero* una única vez, al método *dibujarImagen* se le llamará habitualmente más de una vez.

El método *interpretarFichero* se utiliza para crear una estructura de datos interna que contiene una representación de todos los elementos que formarán parte de la imagen. Para hacerlo se le debe proporcionar como parámetro de entrada la ruta del fichero que se desea leer.

Es un método bastante grande, a pesar de que se ha intentado mantener acotada su complejidad, como se ha mencionado anteriormente, interpretando sólo las partes más útiles del estándar. Para facilitar su comprensión se muestra un diagrama de flujo simplificado de las funciones que realiza cada parte del código por orden en la figura 128.

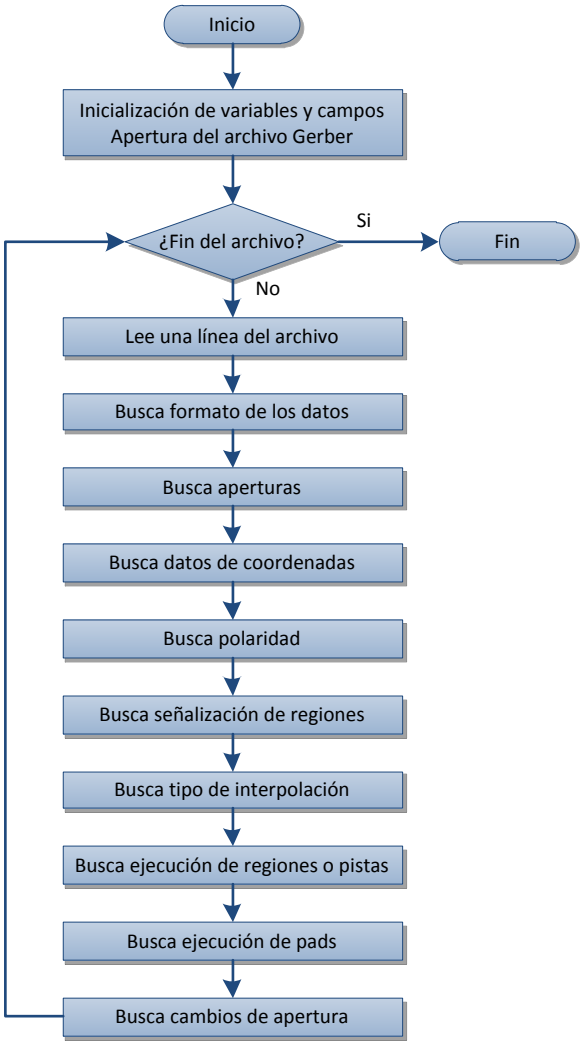


Figura 128. Diagrama de flujo simplificado del método *interpretarFichero*.

Resumidamente, lo que hace este método, cuyo código fuente se puede examinar en la figura 129, es ir leyendo el archivo línea a línea y utilizar tanto búsquedas de cadena como expresiones regulares para encontrar los patrones representativos de los comandos y datos relevantes, actualizando el estado y las estructuras de datos en consecuencia.

```
public bool interpretarFichero(string rutaFichero)
{
  try
  {
    double xi = 0;
    double yi = 0;
    double xf = 0;
    double yf = 0;
    double signoGiro = 1;
    int aperturaActual = 0;
    bool modoRegión = false;
    bool modoDark = true;
    bool interpolaciónLineal = true;

    FileInfo fi = new FileInfo(rutaFichero);
    if (fi.Exists)
    {
      StreamReader sr = fi.OpenText();
      string linea;
      while (!sr.EndOfStream)
      {
        // Lee una línea del archivo
        linea = sr.ReadLine();

        // Formato de los datos
        if (linea.Contains("%MOIN")) enPulgadas = true;
        if (linea.Contains("%MOMM")) enPulgadas = false;
        if (linea.Contains("%FS"))
        {
          int índice = linea.IndexOf("%FS") + 3;
          leadingZerosOmitidos = linea[índice] == 'L';
          Match digitosXEncontrado = Regex.Match(linea.Substring(índice + 2), @"X*\d{2}");
          if (digitosXEncontrado.Success) enteros = int.Parse((digitosXEncontrado.Value)[1].ToString());
        }

        // Aperturas
        Match adEncontrado = Regex.Match(linea, @"%ADD*\d{1,4}");
        if (adEncontrado.Success)
        {
          datosApertura ap = new datosApertura();
          switch (linea.Substring(adEncontrado.Index + adEncontrado.Length, 2))
          {
            case "C,":
              ap.tipo = tipoApertura.circular;
              int indiceDatoC = linea.IndexOf(",") + 1;
              int longitudDatoC = linea.IndexOf("*") - indiceDatoC;
              ap.x = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoC,
                  longitudDatoC).Replace('.', ',')));
              ap.y = 0;
              aperturas.Add(ap);
              break;
            case "O,":
            case "R,":
              ap.tipo = tipoApertura.rectangular;
              int indiceDatoX = linea.IndexOf(",") + 1;
              int longitudDatoX = linea.IndexOf("X") - indiceDatoX;
              ap.x = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoX,
                  longitudDatoX).Replace('.', ',')));
              int indiceDatoY = linea.IndexOf("X") + 1;
              int longitudDatoY = linea.IndexOf("*") - indiceDatoY;
              ap.y = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoY,
                  longitudDatoY).Replace('.', ',')));
              aperturas.Add(ap);
              break;
            default:
              ap.tipo = tipoApertura.desconocida;
              ap.x = 0;
              ap.y = 0;
              aperturas.Add(ap);
              break;
          }
        }

        // Coordenadas
        Match xEncontrado = Regex.Match(linea, @"X-*\d{1," + (enteros + decimales).ToString() + "}");
        if (xEncontrado.Success)
        {
          xi = xf;
          xf = convertirAmm(convertirANúmero(xEncontrado.Value.Substring(1)));
          xmin = xf < xmin ? xf : xmin;
          xmax = xf > xmax ? xf : xmax;
        }
        Match yEncontrado = Regex.Match(linea, @"Y-*\d{1," + (enteros + decimales).ToString() + "}");
        if (yEncontrado.Success)
```

```
{
  yi = yf;
  yf = convertirAmm(convertirANúmero(yEncontrado.Value.Substring(1)));
  ymin = yf < ymin ? yf : ymin;
  ymax = yf > ymax ? yf : ymax;
}

// Polaridad
if (linea.Contains("%LPC")) modoDark = false;
if (linea.Contains("%LPD")) modoDark = true;

// Regiones
if (linea.Contains("G36*"))
{
  modoRegión = true;
  datosRegión región = new datosRegión();
  región.x = new ArrayList();
  región.y = new ArrayList();
  datosGráficos gráfico;
  gráfico.tipo = tipoGráfico.región;
  gráfico.datos = región;
  gráfico.colorInvertido = modoDark;
  elementosGráficos.Add(gráfico);
}
if (linea.Contains("G37*")) modoRegión = false;

// Tipo de interpolación
if (linea.Contains("G01*")) interpolaciónLineal = true;
if (linea.Contains("G03*"))
{
  interpolaciónLineal = false;
  signoGiro = -1;
}
if (linea.Contains("G02*"))
{
  interpolaciónLineal = false;
  signoGiro = 1;
}

// Ejecución de pistas / regiones
if (modoRegión)
{
  if (linea.Contains("D01"))
  {
    datosGráficos gráfico = (datosGráficos)elementosGráficos[elementosGráficos.Count - 1];
    ((datosRegión)gráfico.datos).x.Add(xf);
    ((datosRegión)gráfico.datos).y.Add(yf);
  }
  if (linea.Contains("D02"))
  {
    datosRegión región = new datosRegión();
    región.x = new ArrayList();
    región.y = new ArrayList();
    datosGráficos gráfico;
    gráfico.tipo = tipoGráfico.región;
    gráfico.datos = región;
    gráfico.colorInvertido = modoDark;
    elementosGráficos.Add(gráfico);
  }
}
else
{
  if ((linea.Contains("D01")) && interpolaciónLineal)
  {
    if (!xEncontrado.Success) xi = xf;
    if (!yEncontrado.Success) yi = yf;
    datosPista pista;
    pista.xi = xi;
    pista.yi = yi;
    pista.xf = xf;
    pista.yf = yf;
    datosApertura aperturaAUsar = (datosApertura)aperturas[aperturaActual];
    pista.ancho = aperturaAUsar.x;
    datosGráficos gráfico;
    gráfico.tipo = tipoGráfico.draw;
    gráfico.datos = pista;
    gráfico.colorInvertido = modoDark;
    elementosGráficos.Add(gráfico);
  }
```

```
          if ((linea.Contains("D01")) && !interpolaciónLineal)
          {
            double i = 0;
            double j = 0;
            if (!xEncontrado.Success) xi = xf;
            if (!yEncontrado.Success) yi = yf;
            Match iEncontrado = Regex.Match(linea, @"I-*\d{1," + (enteros + decimales).ToString() + "}");
            if (iEncontrado.Success) i = convertirAmm(convertirANúmero(iEncontrado.Value.Substring(1)));
            Match jEncontrado = Regex.Match(linea, @"J-*\d{1," + (enteros + decimales).ToString() + "}");
            if (jEncontrado.Success) j = convertirAmm(convertirANúmero(jEncontrado.Value.Substring(1)));
                   datosArco arco;
            arco.diámetro = Math.Sqrt(Math.Pow(i, 2) + Math.Pow(j, 2)) * 2;
            arco.xOrigen = xi + i - arco.diámetro / 2;
            arco.yOrigen = yi + j + arco.diámetro / 2;
            arco.ánguloInicial = - Math.Atan2(-j, -i) * (180 / Math.PI);
            arco.ánguloABarrer = Math.Abs(Math.Atan2(yf - (yi + j), xf - (xi + i)) * (180 / Math.PI) -
                  arco.ánguloInicial);
            arco.ánguloABarrer = signoGiro * Math.Min((arco.ánguloABarrer < 0.5) ? 360 : arco.ánguloABarrer,
                  360); // El ángulo no puede ser muy pequeño por bug en DrawArc
            datosApertura aperturaAUsar = (datosApertura)aperturas[aperturaActual];
            arco.ancho = aperturaAUsar.x;

            datosGráficos gráfico;
            gráfico.tipo = tipoGráfico.arco;
            gráfico.datos = arco;
            gráfico.colorInvertido = modoDark;
            elementosGráficos.Add(gráfico);
          }
        }

        // Ejecución de pads
        if (linea.Contains("D03"))
        {
          datosFlash flash;
          flash.x = xf;
          flash.y = yf;
          flash.apertura = (datosApertura)aperturas[aperturaActual];

          // Actualiza dimensiones de la placa
          datosApertura aperturaUsada = (datosApertura)aperturas[aperturaActual];
          double ancho = aperturaUsada.x;
          double alto = aperturaUsada.y;
          xmin = xf - ancho < xmin ? xf - ancho : xmin;
          xmax = xf + ancho > xmax ? xf + ancho : xmax;
          ymin = yf - alto < ymin ? yf - alto : ymin;
          ymax = yf + alto > ymax ? yf + alto : ymax;
          datosGráficos gráfico;
          gráfico.tipo = tipoGráfico.flash;
          gráfico.datos = flash;
          gráfico.colorInvertido = modoDark;
          elementosGráficos.Add(gráfico);
        }

        // Cambio de apertura
        for (int n = 0; n < aperturas.Count; n++)
        {
          string aperturaBuscada = "D" + (n + 10).ToString();
          if (linea.Contains(aperturaBuscada)) aperturaActual = n;
        }
      }
      return true;
    }
    else
    {
      return false;
    }
  }
  catch
  {
    return false;
  }
}
```

Figura 129. Código fuente del método *interpretarFichero*.

Las primeras líneas son de inicialización, apertura del archivo y la definición de un bucle *while* para recorrer todas las líneas del archivo. Las secciones siguientes del código se han

marcado encabezándolas con comentarios para que se puedan identificar fácilmente según el diagrama de flujo anterior.

Tras leer cada línea del archivo, lo primero que se encuentra es una sección que se encarga de detectar si en la línea se encuentran comandos que indiquen el formato en que hay que interpretar los datos. Si se encuentra alguno de estos comandos se actualizan las variables de estado correspondientes. A continuación se buscan los posibles elementos de dibujo existentes en la línea y se van creando en memoria unas estructuras de datos que los represente.

Tras buscar el formato de los datos, el código pasa a buscar la definición de aperturas. Para mantener la correspondencia entre los índices de apertura del archivo Gerber con los índices de nuestra estructura de aperturas en memoria, se almacenan todas las aperturas que aparecen tanto si se van a usar como si no. Para simplificar, podríamos definir una apertura en términos de diseño gráfico como “el pincel con el que se va a pintar”.

El término apertura viene de los orígenes de los archivos Gerber, en que éstos eran usados para gobernar fotoploters con los que se creaban los fotolitos para fabricar las placas. En la figura 130 se puede ver una representación esquemática del funcionamiento de dichos fotoploters. Como se puede ver, su funcionamiento se basa en enfocar un haz de luz sobre una mesa que se puede mover en los ejes x-y. Sobre la mesa se encuentra fijado el fotolito que va a ser tratado. Un obturador corta el haz de luz cuando no hay que “dibujar” sobre el fotolito.

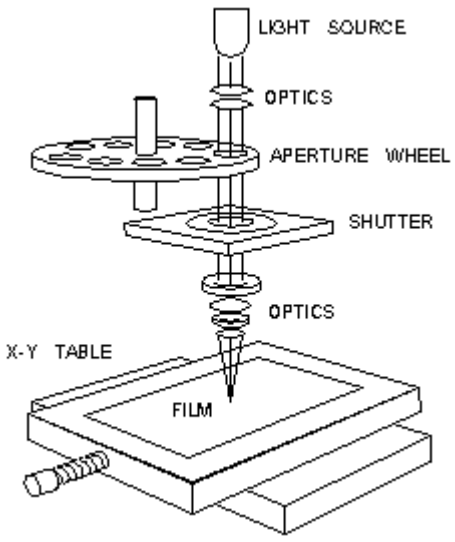


Figura 130. Representación esquemática del funcionamiento de un fotoplóter.

Aunque el formato Gerber era algo diferente entonces, las aperturas hacían referencia a agujeros reales en una pieza opaca (*aperture wheel*) que se podían seleccionar para dar forma y tamaño al haz de luz.

A continuación de las aperturas, en el código del método *interpretarFichero* se detecta la aparición de nuevas coordenadas (x o y) que quedarán registradas en las variables de soporte hasta detectar el comando al que hacen referencia, pues podrían significar cosas diferentes según sea el comando.

Seguidamente se detecta si hay cambios de polaridad y se registra en la variable *modoDark* hasta que sea necesario utilizarla, al igual que las coordenadas. Para simplificar, podríamos decir que la polaridad hace referencia a si los próximos elementos dibujan o borran, o sea, si se van a dibujar con el color de tinta o con el color de fondo.

A continuación se detecta si se entra o se sale del modo "*Región*" y se registra en la variable *modoRegión*. Una región es simplemente un área que hay que rellenar (por ejemplo, podría ser un plano de masa), definida por una serie de límites que vendrán especificados en las líneas siguientes hasta que se detecte el comando para salir del modo región. Si se entra en modo región, se crea un objeto región y se prepara para que reciba los datos relativos a las fronteras de la región. Básicamente, esto consiste en crear unas listas en las que almacenar todas las coordenadas de los límites de la región.

A partir de este punto se detectan los comandos que ejecutan dibujos comunes. Básicamente se trata de dos comandos, que son *Draw* y *Flash* (D01 y D03 en el archivo Gerber). Se pueden entender estos términos a partir del origen de los archivos Gerber para gobernar fotoploters, tal como se puede ver en la figura 130. Si se mantiene el obturador (*shutter*) abierto mientras se mueve el plano x-y, se creará una línea, mientras que si se abre y cierra el obturador sin mover el plano x-y se creará una imagen de la apertura actual, como se representa en la figura 131.

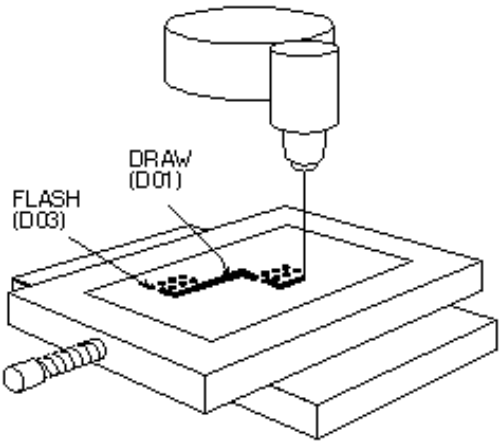


Figura 131. Dibujo de pistas y pads en un fotoplóter.

En la siguiente sección del código se detectan los tramos de las pistas que se dibujan y se almacenan sus coordenadas y el ancho que debe tener. Se tiene en cuenta que si se está definiendo una región, los datos no hacen referencia a tramos de pistas, sino que se están definiendo la frontera de una región y hay que almacenarlos en su lista.

A continuación se detecta el dibujo de pads y se guarda en la estructura los datos relativos a las coordenadas y el tipo de apertura usada. El comando *Flash* se usa para el dibujo de los pads. Como normalmente todas las placas normalmente tendrán al menos dos fiduciales en esquinas opuestas, que se dibujarán mediante el comando *Flash*, se aprovecha en este punto para actualizar las dimensiones de la placa registradas.

Por último se detecta si se ha realizado un cambio de apertura, que se registra para que afecte a los próximos comandos que se interpreten.

La estructura de datos extraída se almacena en dos variables de tipo *arrayList*, que son llamadas *aperturas* y *elementosGráficos*. En la primera se van guardando, como su nombre indica, las definiciones de aperturas, mientras que en la segunda se guardan todos los elementos gráficos que se van encontrando al ir leyendo las sucesivas líneas del fichero Gerber. De esta forma, al analizar el fichero sólo es necesario hacer una pasada para almacenar todos los elementos identificables.

En la figura 132 se pueden ver las definiciones de los tipos de datos que se han realizado para guardar toda esta información.

La estructura *datosPista* guarda las coordenadas de inicio y fin de cada tramo de pista que se debe crear con interpolación lineal, además de su ancho. Como optimización, no se guarda la apertura usada puesto que, aunque calcular la sombra que esta produciría sería más preciso, requeriría un tiempo mucho mayor y un código fuente más complejo y lento a la hora de dibujar. Sin embargo, la ventaja sería únicamente que las terminaciones de las pistas tendrían la forma de la apertura. En las pruebas realizadas se ha comprobado que, como prácticamente todos los tramos de pistas terminan en un pad o en otro tramo de pista, éstos solapan la terminación de la pista haciendo inútil todo ese esfuerzo extra.

La estructura *datosFlash* se usa principalmente para el dibujo de los pads, por lo que aquí si se guarda la apertura que se debe usar junto a las coordenadas donde se ubica.

La estructura *datosApertura* guarda el tipo de apertura y los datos básicos de sus dimensiones. Como optimización se ha simplificado esta estructura, reduciéndolo todo a

aperturas rectangulares o circulares, puesto que no tratamos de hacer un dibujo de la placa con una precisión extrema, sino lo suficientemente preciso que sirva para reconocer el lugar de la placa donde se está trabajando. Esto ha servido también para simplificar las rutinas de dibujo.

```
public struct datosPista
{
    public double xi;
    public double yi;
    public double xf;
    public double yf;
    public double ancho;
}

public struct datosFlash
{
    public double x;
    public double y;
    public datosApertura apertura;
}

public struct datosApertura
{
    public tipoApertura tipo;
    public double x;
    public double y;
}

public enum tipoApertura
{
    circular,
    rectangular,
    desconocida
}

public struct datosRegión
{
    public ArrayList x;
    public ArrayList y;
}

public struct datosArco
{
    public double xOrigen;
    public double yOrigen;
    public double diámetro;
    public double ánguloInicial;
    public double ánguloABarrer;
    public double ancho;
}

public struct datosGráficos
{
    public tipoGráfico tipo;
    public object datos;
    public bool colorInvertido;
}

public enum tipoGráfico
{
    draw,
    flash,
    región
}
```

Figura 132. Tipos de datos para las estructuras de datos Gerber.

Se ha creado un tipo de apertura, llamada *desconocida*, que sirve para identificar la existencia de aperturas no interpretadas por los métodos implementados. Esto ha sido necesario para mantener la consistencia en los índices de las aperturas almacenadas en memoria respecto de los índices de aperturas en el fichero Gerber, simplificando el código necesario para su sincronización.

La estructura *datosRegión*, como ya se comentó anteriormente, guarda las coordenadas de los puntos que forman los límites de la región en sendas listas. Con estos puntos se creará y rellenará un área poligonal.

La estructura *datosArco* guarda los datos necesarios para dibujar un tramo de pista con interpolación circular. Para hacer más rápida la rutina de dibujo, que es la que más conviene optimizar en este aspecto, se guardan los datos que se van a necesitar para la función *DrawArc*, que será la encargada de dibujarlos. Se almacenan las coordenadas de la esquina superior izquierda del cuadrado circunscrito por la circunferencia de la que forma parte el arco a dibujar, el diámetro de dicha circunferencia, el ángulo del punto inicial y el ángulo que barre el arco a dibujar, además del ancho de la línea con que debe ser dibujado.

Por último la estructura *datosGráficos* guarda elementos compuestos por instancias de los tipos de datos anteriores, por lo que define un primer campo que indica que tipo de gráfico es. Además de guardar el objeto gráfico, también registra si se debe dibujar con el color de tinta o de fondo en el campo booleano *colorInvertido*. Como hemos visto anteriormente, los tipos de elementos gráficos que guardará serán *draw*, *flash* y *region*, que serán las pistas, los pads y las áreas poligonales.

El método *interpretarFichero* debe convertir correctamente los datos numéricos que aparecen con frecuencia en el fichero Gerber, por lo que estas funciones se han separado en dos métodos externos que permiten hacer cómodamente las conversiones.

El método *convertirANúmero*, mostrado en la figura 133, se encarga de convertir a un valor numérico las distintas cifras de coordenadas almacenadas en el archivo Gerber. Esta tarea no es tan simple como en principio pudiera parecer, puesto que la forma de almacenar los números en los archivos Gerber es algo especial.

Esto se debe también al origen de los archivos Gerber para gobernar fotoploters que se comentó anteriormente. En esos tiempos, la información de los elementos de la placa se suministraba a la máquina en cinta de papel perforado [**39**], un soporte de almacenamiento de capacidad bastante limitada.

Esto hacía que cada byte con información redundante ocupara un espacio precioso, por lo que se eliminó toda posible redundancia codificando los números de forma que se eliminara la coma decimal y las ristras de ceros a la izquierda o la derecha. Esta decisión introdujo la necesidad de especificar en los comandos de configuración del archivo Gerber la cantidad de cifras enteras y decimales que iban a tener y realizar la descodificación posterior de los valores numéricos.

Lo primero que hace es comprobar el signo, guardarlo y eliminarlo de la cadena. A continuación rellena la cadena con ceros a la izquierda o a la derecha hasta completar el número de dígitos que debe tener cada cifra. El número de dígitos que tiene que tener cada cifra se habrá tomado de la configuración, al principio del fichero, y se habrá guardado en los campos *enteros* y *decimales*. El campo *enteros* contendrá el número de cifras enteras que debe tener el número y el campo *decimales* contendrá el número de cifras decimales que debe contener.

Para saber si debe rellenar con ceros por la izquierda o por la derecha, al leer la configuración se habrá guardado el valor *true* en el campo *leadingZerosOmitidos* en caso de tener que rellenar por la izquierda y *false* en caso contrario.

Tras insertar la coma decimal en el lugar que le corresponde, se convierte la cadena a un valor de tipo *double*, teniendo en cuenta el signo guardado anteriormente, para devolverlo como resultado de la llamada.

```
private double convertirANúmero(string cadena)
{
    double signo = cadena[0] == '-'? -1 : 1;
    cadena = (cadena[0] == '-' || cadena[0] == '+') ? cadena.Substring(1) : cadena;
    if (leadingZerosOmitidos) cadena = cadena.PadLeft(enteros + decimales, '0');
    else cadena = cadena.PadRight(enteros + decimales, '0');
    cadena = cadena.Insert(enteros, ",");
    return Convert.ToDouble(cadena) * signo;
}
```

Figura 133. Código fuente del método *convertirANúmero*.

El método *convertirAmm*, mostrado en la figura 134, convierte la cantidad que se le pase en su parámetro de entrada a milímetros. Para ello comprueba si en la configuración se detectó que el archivo Gerber tenía las cantidades en pulgadas o no (almacenada en el campo *enPulgadas*). En caso de que así sea, devuelve la cantidad convertida, pero si ya estaba en milímetros, devuelve la misma cantidad.

Aunque el método resultante tiene un código muy corto, se simplifica la lectura del código en los lugares donde se usa, ya que éste se hace autoexplicativo.

```
private double convertirAmm(double cantidad)
{
    if (enPulgadas) return cantidad * 25.4;
    return cantidad;
}
```

Figura 134. Código fuente del método *convertirAmm*.

El método *dibujarImagen* de la clase Gerber que se muestra en la figura 135 se utiliza para generar una imagen a partir de la estructura de datos creada por el método *interpretarFichero*. La imagen se genera teniendo en cuenta el factor de escala que se haya definido para la clase. Tener la función de dibujo de esta forma, en un método aparte, permite crear imágenes de distintos tamaños del mismo archivo Gerber sin tener que volver a leerlo desde el disco.

Este método recibe como parámetro la referencia al bitmap donde se desea obtener la imagen de salida. Lo primero que hace es crear y configurar un lienzo donde dibujar a partir del bitmap anterior. Seguidamente crea la pluma con la que dibujará las pistas y el pincel con el que rellenará los pads y las regiones. La pluma se configura con los bordes redondeados en las terminaciones por los motivos explicados anteriormente.

A continuación entra en un bucle *for* que dibuja todos los elementos gráficos almacenados. Mediante una sentencia switch se distingue el tipo de gráfico de que se trata, puesto que, como comentamos anteriormente, cada elemento tiene parámetros diferentes y se debe dibujar de forma diferente.

Los pads circulares o rectangulares se dibujan como elipses rellenas o rectángulos rellenos respectivamente, usando los métodos *FillEllipse* y *FillRectangle* de .NET. Las pistas con interpolación lineal se dibujan como líneas usando el método *DrawLine*, mientras que las que usan interpolación circular se dibujan mediante el método *DrawArc*. Por último, las regiones se dibujan como polígonos rellenos usando el método *FillPolygon.*

Para cada elemento que se dibuja, se multiplican tanto sus coordenadas como sus dimensiones por el factor de escala. De esta forma todos los elementos que forman parte del dibujo se crean ya escalados. Con esto se aplica una de las ventajas inherentes a los gráficos de naturaleza vectorial, que es una imagen de mejor calidad gráfica que la que se conseguiría escalando la imagen resultante, mediante las funciones de escalado de gráficos que posee el framework.

```
public void dibujarImagen(ref Bitmap imgBuffer)
{
    Graphics buffer = Graphics.FromImage(imgBuffer);
    buffer.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
    Pen pluma = new System.Drawing.Pen(colorDeDibujo);
    pluma.EndCap = System.Drawing.Drawing2D.LineCap.Round;
    pluma.StartCap = System.Drawing.Drawing2D.LineCap.Round;
    SolidBrush rellenoDibujo = new SolidBrush(colorDeDibujo);
    SolidBrush rellenoFondo = new SolidBrush(colorDeFondo);
    SolidBrush relleno;

    // Dibujar objetos gráficos
    for (int n = 0; n < elementosGráficos.Count; n++)
    {
        datosGráficos gráfico = (datosGráficos)elementosGráficos[n];
        switch (gráfico.tipo)
        {
            case tipoGráfico.flash:
                datosFlash flash = (datosFlash)gráfico.datos;
                relleno = gráfico.colorInvertido ? rellenoDibujo : rellenoFondo;
                switch (flash.apertura.tipo)
                {
                    case tipoApertura.circular:
                        double radio = flash.apertura.x;
                        buffer.FillEllipse(relleno, (float)((flash.x - radio / 2) * escala),
                                  (float)(imgBuffer.Height - (flash.y + radio / 2) * escala),
                                  (float)(radio * escala), (float)(radio * escala));
                        break;
                    case tipoApertura.rectangular:
                        double ancho = flash.apertura.x;
                        double alto = flash.apertura.y;
                        buffer.FillRectangle(relleno, (float)((flash.x - ancho / 2) * escala),
                                  (float)(imgBuffer.Height - (flash.y + alto / 2) * escala),
                                  (float)(ancho * escala), (float)(alto * escala));
                        break;
                }
                break;
            case tipoGráfico.draw:
                datosPista pista = (datosPista)gráfico.datos;
                pluma.Width = (float)(pista.ancho * escala);
                pluma.Color = gráfico.colorInvertido ? colorDeDibujo : colorDeFondo;
                buffer.DrawLine(pluma, (float)pista.xi * escala, imgBuffer.Height –
                          (float)pista.yi * escala, (float)pista.xf * escala, imgBuffer.Height –
                          (float)pista.yf * escala);
                break;
            case tipoGráfico.región:
                datosRegión región = (datosRegión)gráfico.datos;
                PointF[] puntos = new PointF[región.x.Count];
                for (int i = 0; i < región.x.Count; i++)
                {
                    puntos[i].X = (float)((double)región.x[i] * escala);
                    puntos[i].Y = (float)(imgBuffer.Height - (double)región.y[i] * escala);
                }
                if (puntos.Length > 1)
                {
                    relleno = gráfico.colorInvertido ? rellenoDibujo : rellenoFondo;
                    buffer.FillPolygon(relleno, puntos);
                }
                break;
            case tipoGráfico.arco:
                datosArco arco = (datosArco)gráfico.datos;
                pluma.Width = (float)(arco.ancho * escala);
                pluma.Color = gráfico.colorInvertido ? colorDeDibujo : colorDeFondo;
                buffer.DrawArc(pluma, (float)arco.xOrigen * escala, imgBuffer.Height –
                          (float)arco.yOrigen * escala, (float)arco.diámetro * escala,
                          (float)arco.diámetro * escala, (float)arco.ánguloInicial,
                          (float)arco.ánguloABarrer);
                break;
        }
    }
}
```

Figura 135. Código fuente del método *dibujarImagen*.

5.15.3. Limitaciones de la clase

Como se ha explicado anteriormente, con la implementación del formato de archivos Gerber que se ha realizado en este proyecto no se ha tratado de conseguir la máxima fidelidad de las imágenes obtenidas, puesto que no es nuestro objetivo fabricar la PCB. Nuestro objetivo es montar los componentes sobre la PCB. Por tanto, se pretende mostrar las imágenes obtenidas a partir de los archivos Gerber para proporcionar las instrucciones de montaje al operador, de forma que realice la tarea ágilmente y sin errores. El operador usará las imágenes obtenidas a partir de los archivos Gerber únicamente para orientarse durante el montaje. En este apartado profundizaremos un poco más en las limitaciones que tiene nuestra implementación Gerber.

El formato de archivos Gerber incluye varios tipos de elementos que pueden ser utilizados para definir las formas a dibujar. Como se ha mencionado, se realizó un minucioso estudio del código perteneciente a varios archivos de ejemplo, obtenidos de placas reales fabricadas en el laboratorio del SFP, pudiendo observar algunas detalles interesantes que nos llevaron a pensar que era posible implementar esta característica al programa evitando que el esfuerzo necesario fuese excesivo.

En concreto, nos dimos cuenta que la inmensa mayoría de los elementos de dibujo utilizados en los archivos analizados, pertenecían a un subgrupo del total de elementos posibles del estándar. Eso nos hizo pensar que posiblemente implementando únicamente este subgrupo de elementos, se podría representar el dibujo de la placa con precisión suficiente para orientar al operador, aunque algunos detalles no se mostraran fielmente.

Tras realizar las pruebas, se vio que los resultados eran más que satisfactorios, teniendo en cuenta el uso que se le iba a dar a las imágenes. En los apartados anteriores se ha descrito las partes del estándar implementadas.

Una de las principales características no implementadas fue la posibilidad que tiene el estándar de definir aperturas macro. Debido a la complejidad que estas aperturas pueden llegar a tener, puede ser muy complicado extraer parámetros simplificados de estas. Por tanto se complicaría y ralentizaría el método necesario para dibujar la placa (pensemos en el caso de tener que dibujar una pista o un arco con estas aperturas). Por tanto, se optó por no representarlas durante las primeras pruebas, comprobándose que aun así los gráficos obtenidos eran más que aceptables.

También se simplificaron las aperturas ordinarias aceptadas, puesto que se comprobó que las aperturas con formas más irregulares no se usaban, sustituyéndose en ocasiones por dibujos

mediante las primitivas más simples. Además, la apertura ovalada (*obround* en el estándar) se sustituyó por una apertura rectangular del mismo ancho y alto, puesto que son muy similares, pero la rectangular es más fácil de dibujar (y por tanto, también más rápida).

En el caso de los diseños estudiados, se pudo comprobar que dibujar todos los elementos utilizando únicamente la interpolación lineal (e ignorando la interpolación circular que define el estándar) no afectaba de forma importante a la identificación necesaria para el montaje de los componentes SMD. De hecho, los errores de interpolación introducidos causaron la aparición de líneas rectas en los huecos de separación de los planos de tierra con los pads de los componentes THT incorporados en la placa, como se muestra en los detalles que refleja la figura 136.

Como se mencionó anteriormente, estos errores no representaban un problema importante para la identificación de la ubicación del componente que se iba a montar, tanto por la pequeña incidencia en la imagen como porque se trata de pequeñas imperfecciones en vías y pads de componentes THT que no representan el foco principal de nuestro objetivo al añadir esta característica.

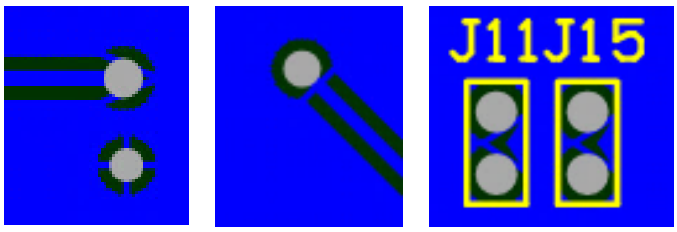


Figura 136. Detalles de errores sin interpolación circular.

Una incidencia algo superior tiene el hecho de que, al carecer de interpolación circular, no se representa bien el dibujo de la serigrafía de los componentes tubulares con terminales radiales, como los condensadores electrolíticos típicos.

Tras codificar la segunda versión de la clase Gerber, incluyendo la interpolación circular, se puede ver que muchos de los pequeños errores de imagen anteriores han desaparecido, como se puede ver en la figura 137.

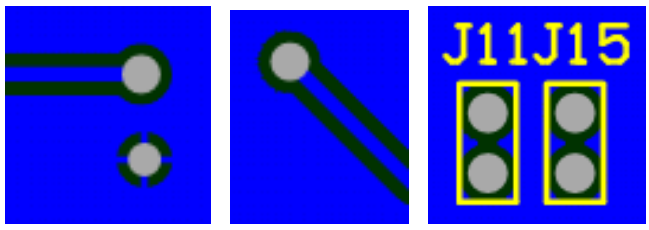


Figura 137. Detalles de errores corregidos al implementar la interpolación circular.

Además de lo anterior, ahora se muestran los detalles curvos de los componentes circulares o con arcos de circunferencia, como los condensadores electrolíticos y las muescas curvas de los circuitos integrados, tal como se puede ver en la figura 138.

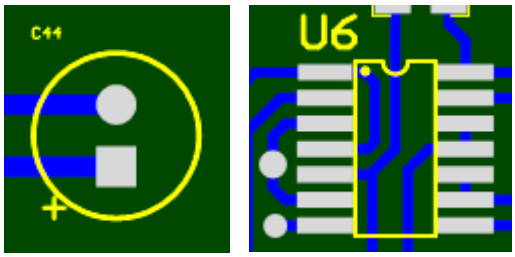


Figura 138. Detalles de componentes que usan interpolación circular.

Hay que precisar que estos resultados se han logrado implementando únicamente la interpolación circular multicuadrante, puesto que en los ficheros analizados era ésta la única utilizada. Si en otros diseños diferentes, el software usado para crear la PCB usase la interpolación circular en un solo cuadrante, los arcos resultantes serían reducidos a líneas rectas. En nuestro análisis dedujimos que el uso de esta interpolación podría ser bastante improbable, puesto que únicamente sirve para crear arcos de menos de 90°, que también se podrían hacer con la multicuadrante.

Otro elemento del estándar que ha sido obviado, puesto que no se ha observado su uso en ninguno de los archivos estudiados (además que posiblemente no tiene demasiado sentido hacerlo) es la posibilidad de asignar una cantidad de dígitos enteros y decimales diferentes a las coordenadas x de los asignados a las coordenadas y.

Tampoco se ha implementado la posibilidad de especificar las coordenadas como valores relativos a la última posición usada. Además de ser una fuente de posibles errores de interpretación, las últimas versiones del estándar han calificado esta característica como obsoleta y no se recomienda su uso.

Otras características del estándar, tanto de las declaradas como obsoletas (y por tanto no recomendadas), como de las vigentes tampoco han sido implementadas, pero, como se ha comentado anteriormente, en los circuitos de prueba se ha comprobado que el impacto en la utilidad de las imágenes para nuestros propósitos es bajo.

5.16. Asignar archivos Gerber

Para asociar a cada placa que se tenga en memoria sus archivos Gerber correspondientes se ha creado un formulario específico. Este formulario se activa desde la opción *Asignar archivos Gerber* que se encuentra en el menú *Placas* y se puede ver en la figura 139.

El formulario permite seleccionar, mediante un comboBox la placa sobre la que se va a trabajar, para luego asignarle hasta 3 capas de archivos Gerber. Se ha decidido limitar el número de capas a 3 como compromiso entre una imagen suficientemente útil, un uso conservador de la memoria y un retardo despreciable en la generación de imágenes (algunos archivos Gerber pueden incluir un número muy grande de comandos).

Se pudo comprobar que, normalmente, las capas más útiles para ayudar al montaje manual eran:

- *Overlay*: Contiene la serigrafía de componentes y textos sobre la placa.
- *Solder*: Contiene los pads, vías y otros elementos a estañar.
- *Layer*: Contiene las pistas y planos de cobre de la placa.

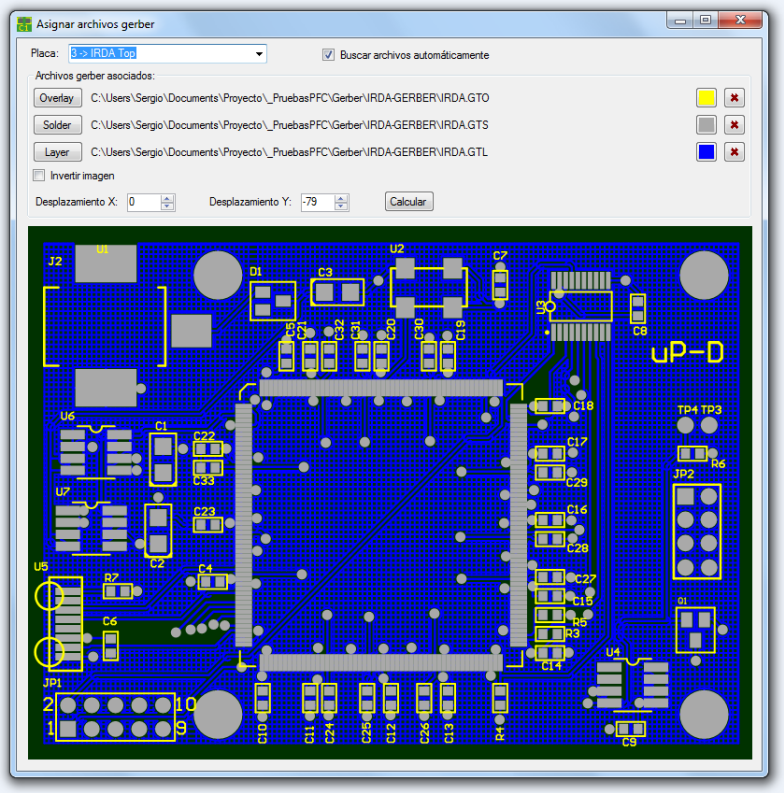


Figura 139. Formulario Asignar archivos Gerber.

Para una representación correcta, las capas se deben disponer en el orden en que se han indicado, es decir, debajo irá la capa *layer*, sobre esta la capa *solder* y encima de todas la capa

overlay. Por tanto sus datos se han dispuesto en pantalla en este mismo orden para facilitar su comprensión.

Pulsando el botón con el nombre de la capa se puede seleccionar el archivo desde un cuadro de diálogo *Abrir* estándar. Cuando se pulse aceptar en este cuadro de diálogo, se mostrará la ruta a la derecha del botón y la imagen en la parte inferior del formulario. De esta forma se puede deducir de un vistazo si se han elegido los archivos Gerber correctos.

Si está marcada la opción buscar archivos automáticamente, también se buscarán en la misma ruta otros archivos con extensiones conocidas para el resto de las capas y se rellenarán automáticamente, por lo que normalmente bastará con elegir el primero de ellos. De esta forma se agiliza bastante esta tarea.

A la derecha de la ruta de cada archivo de capa Gerber, se muestran dos botones. El primero permite elegir con qué color se representará esa capa y el segundo permite eliminar la asociación de archivo de la capa.

La selección del color se realiza mediante un cuadro de diálogo *Color* estándar de *Windows*, como se puede ver en la figura 140. Una vez elegido el color, el botón cambiará de color para reflejar el color elegido para esa capa. Los colores predeterminados para las capas se han elegido similares a los que se usan para la representación en otros programas: amarillo para la capa overlay, gris para la capa solder y azul para la capa layer. El fondo de la placa se representa de color verde oscuro.

Como sabemos, en caso de que se vaya a trabajar con la cara bottom, normalmente se deberá invertir la imagen, por lo que se ha dispuesto un *checkBox* para hacerlo. En ese caso se recalculan las posiciones de los componentes para que se ubiquen correctamente en la imagen invertida.

Por último, hay que hacer notar que, como los archivos con que trabajamos no contienen información específica de las coordenadas donde se encuentran los límites de la placa, puede ser necesario realizar ajustes en la posición de la imagen cuando se vaya a dibujar junto con los componentes. Se han dispuesto dos controles *textBox* por si se desea ajustar manualmente el desplazamiento de la imagen de forma numérica, aunque normalmente nuestra aplicación calculará dicho desplazamiento sin cometer grandes errores. El botón *Calcular* permite volver a realizar el cálculo automático del desplazamiento en caso de que el usuario haya cambiado manualmente su valor.

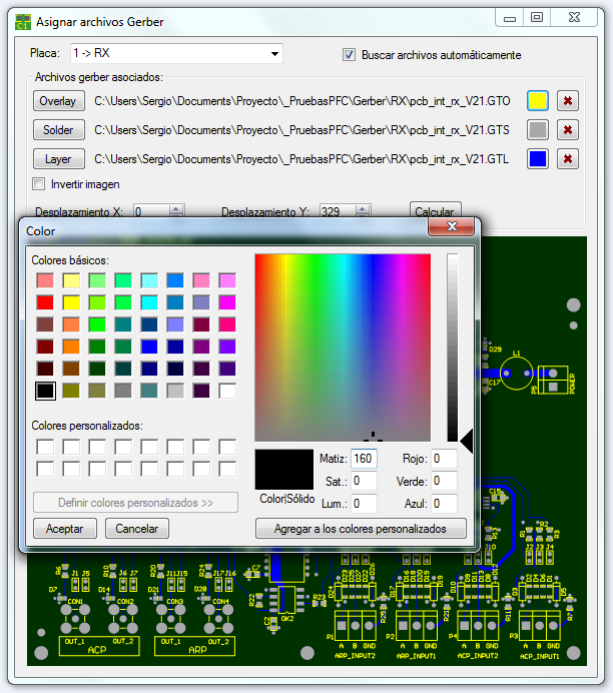


Figura 140. Cuadro de diálogo de selección de color para las capas Gerber.

Estos controles se han puesto en el formulario para dar mayor utilidad a los usuarios avanzados. Normalmente la aplicación realizará el cálculo correcto automáticamente al seleccionar los archivos Gerber por lo que no es necesario cambiar los valores que se muestran. En caso de que quedase algún desplazamiento no corregido automáticamente entre la imagen generada y los componentes, el usuario de la aplicación puede ajustarlo mejor de manera visual usando el ratón (botón central) para desplazar la imagen en el editor gráfico o en el mismo formulario de producción manual.

Todos los ajustes que se realicen de las imágenes Gerber asociadas a cada placa se graban en el archivo del montaje que estamos realizando. Por tanto, al cargar un archivo de disco para volver a trabajar con él en otra sesión se conservarán todos los ajustes que se hayan hecho anteriormente, desde este formulario o desde cualquier otro.

Cuando se redimensiona la ventana del formulario, automáticamente se vuelve a dibujar la placa, ampliando o reduciendo la imagen lo necesario para que quede ajustada al tamaño disponible en la ventana.

5.17. Producción

5.17.1. Descripción general

En esta sección de la aplicación el operador podrá poner en marcha el proceso de montaje de las placas que se han preparado previamente. El menú *"Producción"* contiene las

distintas posibilidades que incorpora el programa para asistir en el montaje de las placas. La aplicación dispone de cuatro formularios diferentes orientados al montaje de las placas, cada uno asignado a una de las opciones de este menú.

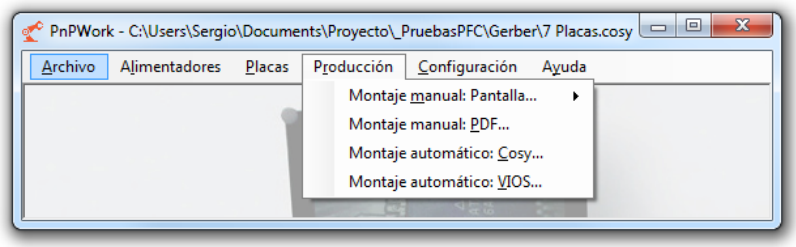


Figura 141. Menú *Producción*.

Las dos primeras opciones permiten la asistencia al montaje manual de componentes, mientras que las dos últimas se utilizarán en el montaje automático.

Para el montaje manual, se tiene la posibilidad de que el programa vaya asistiendo en el montaje al operador de forma interactiva, desde la pantalla del ordenador, o bien generar un archivo PDF con la documentación de montaje necesaria, que el operador podrá llevar hasta el puesto de montaje impresa o en un medio electrónico (ordenador portátil, tablet, ebook, teléfono móvil, etc.).

Para el montaje automático se ha orientado el programa hacia el uso de las máquinas pick & place de que dispone el laboratorio, que son fundamentalmente de dos tipos: las máquinas del tipo Cosy Flexpick, que requieren que un ordenador externo gobierne todo el proceso de montaje y las de Assembleon / Philips, que disponen de su propio sistema de control integrado. Para utilizar estas últimas actualmente se realiza un preprocesado de los archivos generados desde el programa de diseño utilizado (normalmente Altium). Posteriormente se alimenta la máquina con el archivo resultado de este preproceso que contiene los datos de ubicación de los componentes a montar (archivo VIOS).

Mediante estas dos opciones del menú, la aplicación puede controlar directamente las máquinas del tipo Cosy mediante un puerto serie o generar los archivos de control que alimentarán a las máquinas Assembleon / Philips.

De esta forma, la aplicación creada intenta cubrir todo el espectro de posibilidades de montaje de componentes SMD disponible actualmente en el laboratorio, e incluso se puede utilizar para asistir en el montaje manual de los componentes THT.

5.17.2. Montaje manual: Pantalla

Esta opción del menú se utilizará para guiar a la persona que va a montar la placa utilizando métodos manuales durante las operaciones de montaje de los componentes. En el montaje manual que se lleva a cabo en el laboratorio se utilizan máquinas pick & place como la de la figura 142, que facilitan la tarea.

La forma de guiar al montador será mostrando una imagen de la placa y sobre ella irá indicándole qué componente debe mostrar y dónde. Cuando haya montado dicho componente, el usuario pulsará cualquier tecla y la aplicación lo marcará como montado, continuando al siguiente por orden.

De esta forma se logrará que incluso una persona con poca experiencia en este tipo de montajes realice la tarea sin cometer errores. El objetivo es no sólo que cada componente se coloque en el lugar correcto, sino que no se omita ningún componente durante el montaje por descuido.

Se ha planteado esta función de la aplicación para permitir también un control menos estricto del montaje a usuarios más avanzados, pudiéndose saltar en cualquier momento al componente que se desee, cambiar la ubicación de la vista o cambiar el estado de los componentes manualmente, además de distintas opciones de visualización de los componentes montados y pendientes de montar.

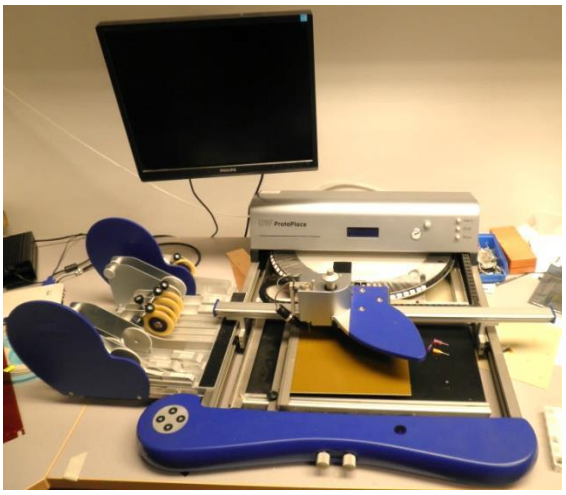


Figura 142. Máquina pick & place manual utilizada en el laboratorio.

Esta opción del menú se ha creado como un submenú dinámico, al igual que la opción "*Ver placa…*" del menú "*Placa*" explicado anteriormente. Por tanto, con cada nueva placa que se cree o cargue desde archivo, se creará una nueva opción del submenú con el nombre de la placa. Para activar esta opción hay que hacer clic sobre el nombre de la placa que se desea montar o

pulsar la combinación de teclas de acceso al menú correspondiente, como se puede ver en la figura 143.

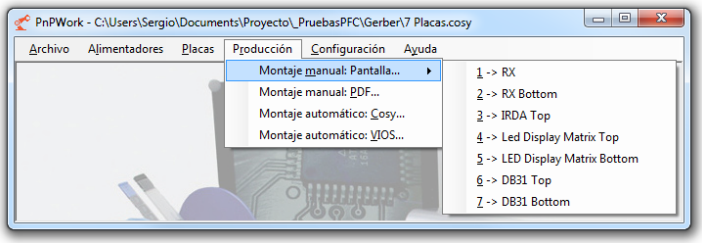


Figura 143. Submenú de montaje manual en pantalla.

Al activar el montaje manual en pantalla, el software comprueba si existen los archivos Gerber asociados a la placa que se va a montar, puesto que si no existen o no hay archivos Gerber asociados, la ayuda visual sería poco útil. En caso de que la placa no tenga asociados archivos Gerber, la aplicación lo notificará al usuario y lo dirigirá hacia el formulario que le permite asignar los archivos Gerber a la placa si así lo desea.

El formulario de montaje manual en pantalla muestra al lado izquierdo la lista de componentes que van a ser montados en el orden en que están almacenados en memoria. Previamente el montador habrá ordenado los componentes de acuerdo al criterio que haya considerado conveniente, probablemente agrupándolos por alimentador. Uno de estos componentes será aquel sobre el que esté puesto el foco en la etapa del montaje en la que se encuentre en cada momento. Este componente estará resaltado y centrado en la pantalla.

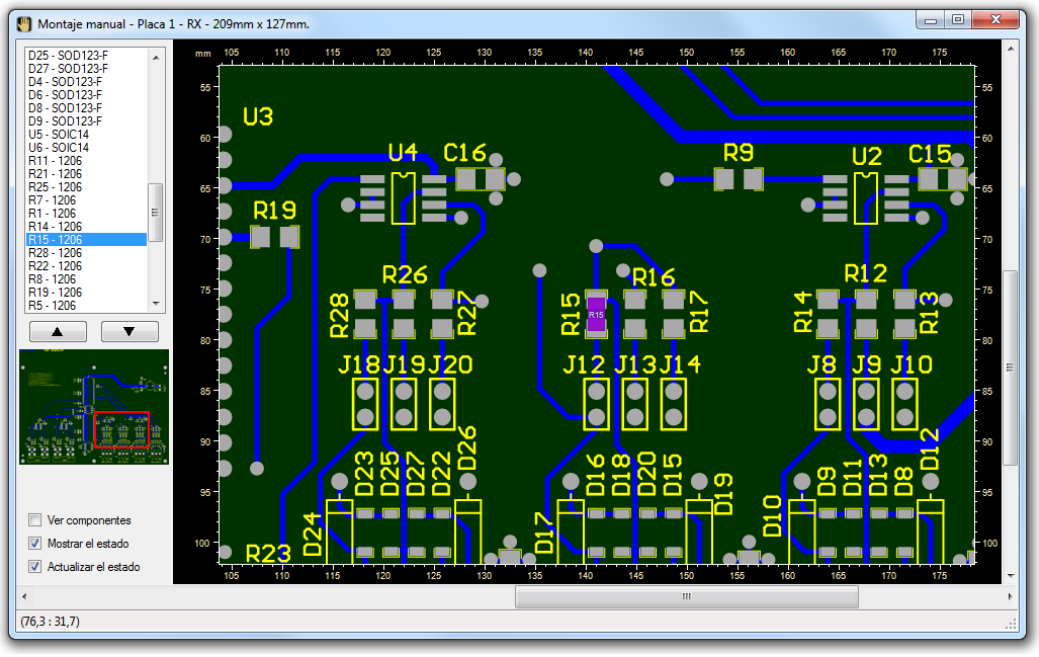


Figura 144. Formulario *Montaje manual*.

Bajo el listado se han dispuesto dos botones con flechas que permiten desplazarse por el listado componente a componente, aunque si se pulsa casi cualquier tecla del teclado se avanzará al siguiente componente. Si se pulsa la tecla de retroceso o la flecha arriba, se retrocederá al componente anterior. Además, si con el ratón se selecciona cualquier componente del listado, el foco cambiará inmediatamente al componente seleccionado.

En el lado derecho de la pantalla se muestra una vista de la placa centrada en el componente activo en cada momento. Si en esta vista se pulsa con el ratón sobre otro componente, el foco cambiará inmediatamente al nuevo componente seleccionado, reflejándose en el listado y centrándose la imagen de la placa en ese componente.

En caso de que la opción "*Mostrar el estado*" esté seleccionada, los componentes se mostrarán con el color que refleja su estado actual, según los colores indicados anteriormente. En la figura 144 se aprecia que el componente que se está tratando en este momento (cuyo estado se cambia automáticamente a "*Cogiendo*" cuando es seleccionado) se muestra en color violeta. Si se desactiva esta opción, todos los componentes se mostrarán del mismo color que los componentes no tratados, independientemente del estado que tengan registrado en memoria.

Si la opción "*Ver componentes*" está seleccionada, se mostrarán todos los componentes que se están montando, como se puede ver en la figura 145. Para distinguir en cualquier caso el componente seleccionado del resto, todos los componentes se muestran semitransparentes, salvo el componente seleccionado, que se muestra casi opaco.

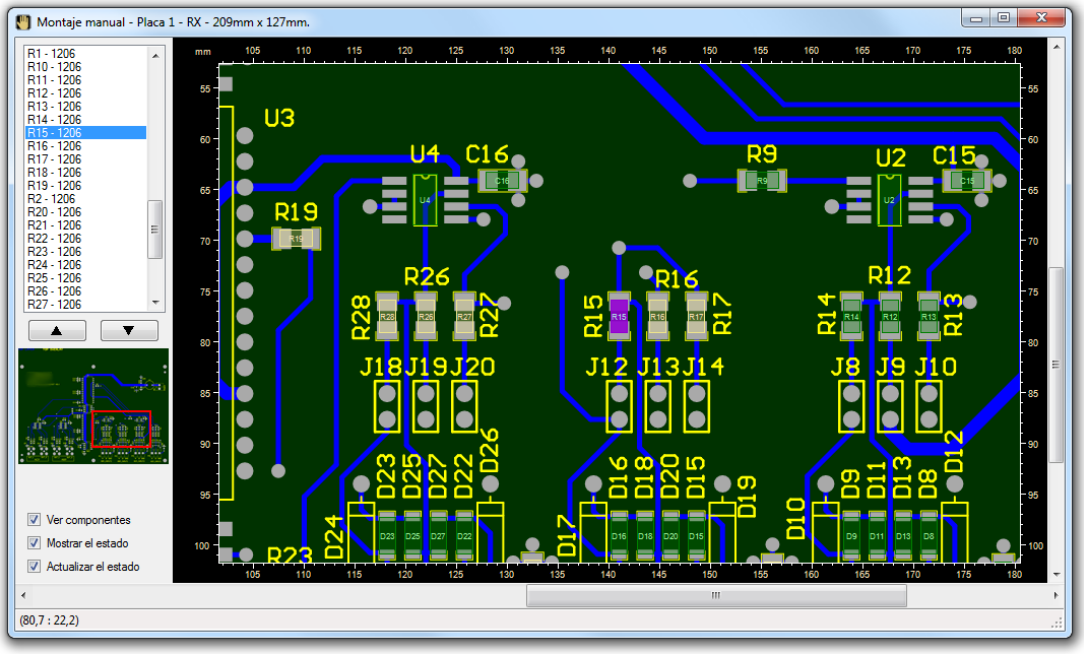


Figura 145. Formulario *Montaje manual* mostrando los componentes y su estado.

Si la opción "*Actualizar el estado*" se encuentra activada, se va actualizando progresivamente el estado de los componentes tratados, cambiando los colores con que se representan en pantalla de forma acorde a su estado. En el caso del montaje manual, al seleccionar un componente, este pasará al estado "*Colocando*" y cuando se pasa a otro componente se cambia al componente anterior al estado "*Colocado*". En la figura 145 se pueden ver varios componentes en los estados "*No tratado*" y "*Colocado*". Además, el componente actual está en el estado "*Colocando*".

Si esta opción no está activada, el estado de los componentes no se actualiza, por lo que cada componente se mostrará con el color que le corresponda según el estado que tenga almacenado en memoria.

Entre el listado de componentes a tratar y las opciones comentadas, hay una vista en miniatura de la placa que se está montando completa. Sobre esta vista, se muestra un rectángulo rojo que se corresponde con el área visible en la vista detallada de la parte derecha de la ventana. Si se hace clic sobre esta miniatura, ésta pasará a la parte derecha de la pantalla y se ampliará lo máximo posible para ocupar el espacio disponible. De esta forma, el montador tendrá una vista global de la placa de mejor calidad gráfica que resalta el área sobre el que está trabajando en ese momento. Para eliminar esta vista ampliada y volver a la vista normal, basta hacer clic de nuevo sobre la miniatura o bien sobre la imagen ampliada en la parte derecha.

Mientras se trabaja con la vista normal en la parte derecha de la ventana, están disponibles algunas características y controles adicionales con el ratón. Al igual que en la vista del editor gráfico de placas, se ha designado la zona central de cada componente como un área activa sensible al ratón. En esta zona central se ubica el nombre del componente. Al pasar el ratón sobre el área activa de un componente, se muestra un bocadillo similar al del editor gráfico, que muestra el nombre del componente, su tipo, coordenadas, ángulo de giro y su estado.

Si se hace clic con el botón izquierdo del ratón sobre el área activa de un componente, éste pasará inmediatamente a ser el nuevo componente seleccionado, actualizándose el listado, la vista en miniatura y la vista detallada del lado izquierdo de forma acorde con la nueva selección. De esta forma, el usuario puede controlar de forma gráfica a qué componente desea pasar a continuación. A diferencia del comportamiento del formulario visor / editor gráfico, en este formulario se ha eliminado la posibilidad de cambiar la posición de los componentes ni su ángulo de giro con los botones izquierdo y central del ratón para evitar errores del usuario.

Haciendo clic con el botón derecho sobre el área activa de cualquier componente se cambia su estado inmediatamente al estado "*Colocado*" o "*No tratado*" según el estado en que estuviese anteriormente. Si el componente se encontraba en el estado "*No tratado*", se cambia al estado "Colocado". En otro caso se cambia al estado "*No Tratado*". Se puede forzar el cambio de estado de cualquier componente sin importar si está seleccionado o no. En caso de forzar el cambio de estado del componente seleccionado, este dejará de estar en el estado "*Colocando*".

Si se hace clic con el botón central (normalmente en la rueda del ratón) sobre cualquier parte de la vista detallada y se arrastra el ratón, se cambiará la posición de la imagen de fondo. Esta imagen de fondo es la imagen de la placa creada a partir de los archivos Gerber que ésta tiene asociados. Con este control se puede cambiar manualmente la posición de la imagen de fondo en caso de que sea necesario. A diferencia del comportamiento del formulario visor / editor gráfico, los ajustes de posición que se hagan desde el formulario de montaje manual no quedan almacenados en la estructura de la placa, por lo que se no se guardarán con el resto de datos de la placa si se guarda ésta en disco. En cambio, si se guardarán los cambios de estado de los componentes.

5.17.3. Montaje manual: PDF

Con esta opción el usuario puede generar un archivo PDF con la información que le guiará para montar los componentes de la placa. Este archivo PDF se podrá imprimir en papel o copiar a un dispositivo móvil para disponer de él en cualquier lugar sin necesidad de un ordenador que le asista en el montaje. En la figura 146 se muestra el aspecto de este formulario y las opciones de exportación que se pueden seleccionar. En la parte inferior siempre aparecerá el número total de componentes seleccionados y cuántas imágenes de ayuda se van a generar. Ambas cantidades dependen de las opciones elegidas en el formulario.

En primer lugar, el usuario debe elegir mediante un *comboBox* qué placa se va a exportar. De forma predeterminada, al crearse el formulario siempre estará seleccionada la primera placa.

El primer grupo de opciones está formado por dos *radioButton* y permite determinar qué componentes se van a mostrar: todos los componentes de la placa o bien sólo aquellos que no estén en el estado "*Colocado*". El siguiente grupo de *radioButton* permite definir si se desea que cada imagen que se genere muestre la colocación de un solo componente o bien de todos los componentes que están asociados al mismo alimentador.

En la parte derecha se encuentran algunas opciones adicionales sobre la apariencia que tendrán los gráficos generados. El *checkBox* "Incluir comentarios" determina si se incluye la

numeración de cada imagen y el tipo de componente que se está mostrando en dicha imagen. El *checkBox* "*Fondo claro*" cambia el color de fondo de la imagen de la placa por un color muy claro, casi blanco. Esto permite ahorrar mucha tinta en caso de que se vayan a imprimir en papel. El grupo de *radioButton* que se encuentra a continuación permite rotar 90° la imagen que se genera. Esto es útil sobre todo si la placa es más ancha que alta, puesto que se aprovecha mejor el espacio de la hoja, generando imágenes más grandes.

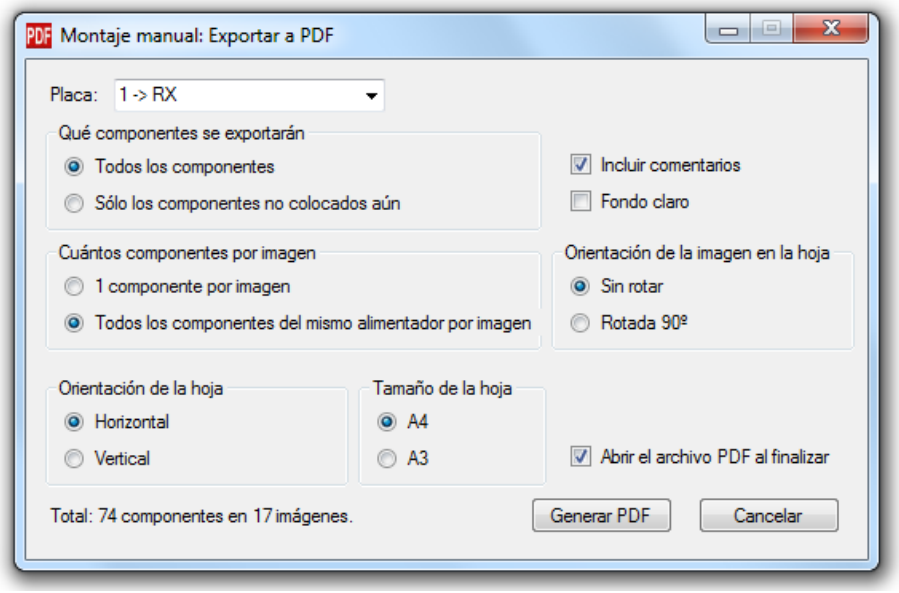


Figura 146. Formulario Montaje manual: PDF.

Hay que tener en cuenta que la opción anterior sólo rota las imágenes, no la numeración ni el comentario del tipo de componente. Por eso en la parte inferior del formulario se incluye una opción adicional que permite definir la orientación de la hoja en su conjunto. Poniendo la hoja con orientación horizontal los textos también se rotarán, quedando apaisados. Un último grupo de *radioButton* permite seleccionar el tamaño de la hoja. Esta opción es útil si se va a imprimir en papel. El último *checkBox* del formulario especifica si al terminar el proceso la aplicación debe abrir el archivo PDF resultante. Para que se abra correctamente, el ordenador debe tener instalado algún software capaz de interpretar este tipo de archivos.

Al pulsar el botón "*Generar PDF"* se inicia el proceso de creación del archivo llamando al proceso *buttonGenerarPDF_Click* cuyo código fuente se puede ver en la figura 147. En primer lugar se muestra al usuario un cuadro de diálogo de tipo *saveFileDialog* para que elija la ruta y el nombre del archivo que se va a guardar. Si se realiza la operación correctamente, tras las inicializaciones de los parámetros básicos que se utilizarán, se determina el tamaño de página y orientación elegidas, se crea el documento PDF de acuerdo a ese tamaño y se abre, inicializando algunos metadatos del archivo.

```
private void buttonGenerarPDF_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "Archivos PDF (*.pdf)|*.pdf|Todos los archivos (*.*)|*.*";
    if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
    {
        placa p = datos.placas[comboBoxPlacas.SelectedIndex];
        int transparenciaRelleno = 230;
        Color colorComponente = Color.Red;
        Color colorRelleno = Color.FromArgb(transparenciaRelleno, colorComponente);
        Brush pincelTexto = Brushes.Wheat;
        System.Drawing.Font fuenteComponentes = new System.Drawing.Font("Arial", 6);
        SizeF tamañoTexto = new SizeF();
        int númeroImagen = 1;
        iTextSharp.text.Rectangle tamañoPágina = radioButtonA4.Checked ? PageSize.A4 : PageSize.A3;
        Document documento = new Document(tamañoPágina, 30, 30, 40, 25);
        PdfWriter.GetInstance(documento, new FileStream(saveFileDialog1.FileName,
                 FileMode.OpenOrCreate));
        if (radioButtonHojaHorizontal.Checked) documento.SetPageSize(tamañoPágina.Rotate());
        documento.Open();
        documento.AddTitle("Información de montaje de componentes");
        documento.AddCreator("Sergio J. Felipe Delgado");

        // Crea la imagen base
        Bitmap imagenGerber = generarImagenGerber();

        if (radioButton1AlimentadorPorImagen.Checked)
        {
            // Imprime los componentes agrupados por alimentador
            for (int a = 0; a < datos.alimentadores.Length; a++)
            {
                Bitmap imagen = new Bitmap(imagenGerber.Width, imagenGerber.Height);
                Graphics lienzo = Graphics.FromImage(imagen);
                lienzo.DrawImage(imagenGerber, 0, 0);

                // Añade todos los componentes que usan este alimentador
                bool hayComponentes = false;
                for (int n = 0; n < p.componentes.Length; n++)
                {
                    if (a == p.componentes[n].alimentador)
                    {
                        if (radioButtonExportarNoColocados.Checked && (p.componentes[n].estado ==
                                  estadoComponente.Colocado)) continue;
                        else
                        {
                            hayComponentes = true;
                            lienzo.DrawPolygon(new Pen(colorComponente), p.componentes[n].vértices
                                  (p.desplazamientoGerber.X, p.desplazamientoGerber.Y));

                            lienzo.FillPolygon(new SolidBrush(colorRelleno), p.componentes[n].vértices
                                  (p.desplazamientoGerber.X, p.desplazamientoGerber.Y));
                            tamañoTexto = lienzo.MeasureString(p.componentes[n].nombre,
                                  fuenteComponentes);
                            lienzo.DrawString(p.componentes[n].nombre, fuenteComponentes, pincelTexto,
                                  new Point((int)(p.componentes[n].x + p.desplazamientoGerber.X -
                                  tamañoTexto.Width / 2.5), (int)(p.componentes[n].y +
                                  p.desplazamientoGerber.Y - tamañoTexto.Height / 3)));
                        }
                    }
                }

                // Sólo se rellena esta hoja si hay componentes usados en el alimentador actual
                if (hayComponentes)
                {
                    if (checkBoxIncluirComentarios.Checked)
                    {
                        documento.Add(new Paragraph("Página " + númeroImagen++ + " de " +
                                  totalImágenes));
                        documento.Add(new Paragraph("Componente seleccionado: " +
                                  datos.alimentadores[a].tipoComponente));
                    }

                    // Rota la imagen si se ha seleccionado
                    if (radioButtonRotar.Checked) imagen.RotateFlip(RotateFlipType.Rotate270FlipNone);

                    // traslada la imagen creada al documento, la redimensiona y la coloca
                    iTextSharp.text.Image imagenDocumento = iTextSharp.text.Image.GetInstance(imagen,
                          (iTextSharp.text.BaseColor)null);
```

```
                    imagenDocumento.ScaleToFit(documento.PageSize.Width - documento.RightMargin –
                          documento.LeftMargin, documento.PageSize.Height - documento.TopMargin –
                          documento.BottomMargin - 60);
                    imagenDocumento.SpacingBefore = 10f;
                    imagenDocumento.SpacingAfter = 1f;
                    imagenDocumento.Alignment = iTextSharp.text.Image.MIDDLE_ALIGN;
                    documento.Add(imagenDocumento);
                    documento.NewPage();
                }
            }
        }
        else
        {
            //Imprime los componentes individualmente
            for (int n = 0; n < p.componentes.Length; n++)
            {
                if (radioButtonExportarNoColocados.Checked && p.componentes[n].estado ==
                          estadoComponente.Colocado) continue;
                if (checkBoxIncluirComentarios.Checked)
                {
                    documento.Add(new Paragraph("Página " + númeroImagen++ + " de " +
                          totalImágenes));
                    string textoComponente = p.componentes[n].nombre + " - " +
                          datos.alimentadores[p.componentes[n].alimentador].tipoComponente;
                    documento.Add(new Paragraph("Componente seleccionado: " + textoComponente));
                }
                Bitmap imagen = new Bitmap(imagenGerber.Width, imagenGerber.Height);
                Graphics lienzo = Graphics.FromImage(imagen);
                lienzo.DrawImage(imagenGerber, 0, 0);

                // Añade el componente
                lienzo.DrawPolygon(new Pen(colorComponente), p.componentes[n].vértices
                          (p.desplazamientoGerber.X, p.desplazamientoGerber.Y));
                lienzo.FillPolygon(new SolidBrush(colorRelleno), p.componentes[n].vértices
                          (p.desplazamientoGerber.X, p.desplazamientoGerber.Y));
                tamañoTexto = lienzo.MeasureString(p.componentes[n].nombre, fuenteComponentes);
                lienzo.DrawString(p.componentes[n].nombre, fuenteComponentes, pincelTexto,
                          new Point((int)(p.componentes[n].x + p.desplazamientoGerber.X –
                          tamañoTexto.Width / 2.5), (int)(p.componentes[n].y +
                          p.desplazamientoGerber.Y - tamañoTexto.Height / 3)));

                // Rota la imagen si se ha seleccionado
                if (radioButtonRotar.Checked) imagen.RotateFlip(RotateFlipType.Rotate270FlipNone);

                // traslada la imagen creada al documento, la redimensiona y la coloca
                iTextSharp.text.Image imagenDocumento = iTextSharp.text.Image.GetInstance(imagen,
                          (iTextSharp.text.BaseColor)null);
                imagenDocumento.ScaleToFit(documento.PageSize.Width - documento.RightMargin –
                          documento.LeftMargin, documento.PageSize.Height - documento.TopMargin –
                          documento.BottomMargin - 60);
                imagenDocumento.SpacingBefore = 10f;
                imagenDocumento.SpacingAfter = 1f;
                imagenDocumento.Alignment = iTextSharp.text.Image.MIDDLE_ALIGN;
                documento.Add(imagenDocumento);
                documento.NewPage();
            }
        }
        documento.Close();

        // Muestra el archivo PDF
        if (checkBoxAbrirPDF.Checked) Process.Start(saveFileDialog1.FileName);
        this.Close();
    }
}
```

Figura 147. Código fuente de *buttonGenerarPDF_Click().*

A continuación se genera la imagen base de la placa a partir de los archivos Gerber. Esta imagen base se usará en cada una de las imágenes de ayuda que se generen, puesto que será el fondo común a todas.

El siguiente paso es determinar si se debe imprimir una imagen por componente o una imagen por alimentador. En el primer caso se recorren todos los componentes para dibujar cada uno sobre el fondo en una imagen en que sólo aparezca ese. En el segundo caso hay que recorrer todos los alimentadores, buscando todos los componentes que usan cada alimentador para dibujar todos estos componentes sobre la misma imagen de fondo.

Seguidamente, en cada paso se rota la imagen generada (si se eligió esa opción) para posteriormente trasladar la imagen al documento. La imagen se redimensiona y se recoloca para que encaje en la hoja y se coloca centrada.

Al terminar el proceso para todos los componentes o alimentadores, se cierra el documento y, en caso de que se haya elegido la opción, se abre el documento PDF generado.

5.17.4. Montaje automático: Cosy

En esta sección de la aplicación el operador podrá poner en marcha el proceso de montaje de las placas que se han preparado previamente, utilizando la máquina pick & place Cosy descrita anteriormente para realizar dicho montaje.

Para utilizar esta opción, es necesario que el ordenador esté conectado a la máquina mediante un puerto serie estándar RS-232 y con la máquina encendida antes de pulsar el botón de iniciar la producción, que veremos posteriormente.

Al seleccionar esta opción, se abre una nueva ventana que contiene el formulario de opciones de producción, mostrado en la figura 148, desde donde se puede controlar los detalles de la producción e iniciar ésta.

En esta ventana se muestra una lista con todas las placas que se pueden procesar de entre las que consta la composición sobre la que se está trabajando. En esta lista no se muestra ninguna de las placas que contengan errores (como por ejemplo componentes situados fuera de los límites de la placa). Sin embargo, aunque no se muestren, en caso de que existan placas con errores se indica con un mensaje en la parte superior para avisar al operador.

El ejemplo de la figura 148 se ha creado con un error en la placa número 3 para que aparezca dicho mensaje. Como se puede ver en la imagen, la placa número 3 no aparece en el listado porque contiene un error.

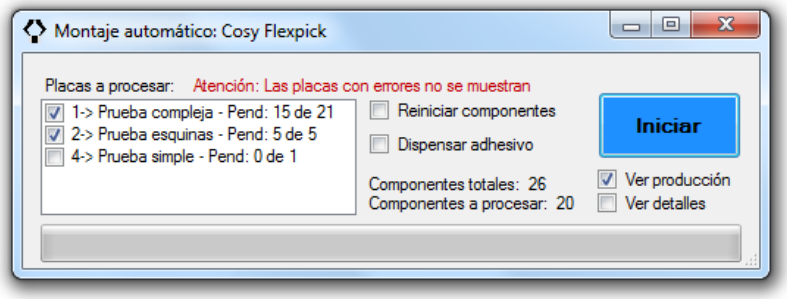


Figura 148. Formulario inicial de montaje con la Cosy.

La lista indica el número y nombre de la placa, cuántos componentes quedan por colocar y el total de componentes que tiene. Para saber el número de componentes pendientes de colocar, cada placa dispone de un método que recorre su lista de componentes, comprobando el estado de cada uno de ellos y contabilizándolos.

En la lista, cada placa tiene un *checkbox* para seleccionar si se quiere incluir en la tarea de producción que se va a iniciar o no. Al cargar este formulario se inicializa preseleccionando para la producción todas las placas que tengan componentes pendientes de ser montados, pero el usuario puede marcar o desmarcar las que necesite.

Junto a la lista hay otro *checkbox* que permite reiniciar el estado de los componentes de cada placa seleccionada durante la producción. Cuando se inicia la producción de cada una de las placas seleccionadas, si esta opción está seleccionada se reinicia el estado de todos sus componentes al estado "*No colocado*". De esta forma se forzará al programa a colocar de nuevo todos los componentes independientemente del estado en que estuvieran registrados en memoria previamente. El resto de placas que no estén marcadas para su producción no serán alteradas.

De forma predeterminada esta opción no aparece marcada y no se reinicia el estado de los componentes pues si algunos de estos tienen un estado "*Colocado*" normalmente será porque ha habido una producción parcial anterior que ha dejado la placa en un estado intermedio. En este estado, lo normal es que se quiera continuar la producción de esta placa desde el punto en que se interrumpió, evitando poner un nuevo componente sobre otro ya colocado anteriormente.

El botón "*Iniciar*" permite dar comienzo a la producción y una vez puesta en marcha cambia para permitir detenerla, como se puede ver en la figura 149. Se ha elegido dar colores llamativos y unas dimensiones mayores a este botón para facilitar su localización en caso de necesitar cancelar una producción rápidamente. Al detener una producción mediante el botón del

formulario no se para inmediatamente, sino que antes se termina de colocar el componente que se está procesando y luego se aparcan los útiles, el cabezal y el banco de alimentadores.

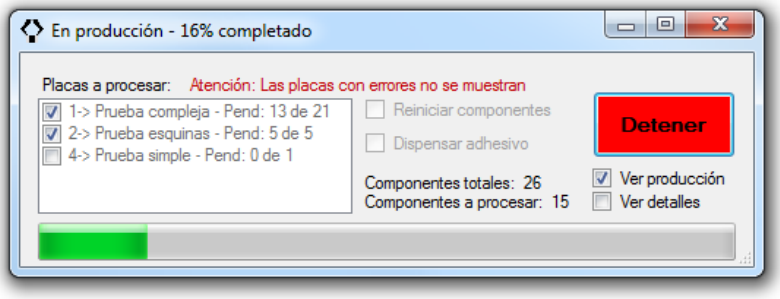


Figura 149. Formulario de producción con un montaje en curso.

En caso de que se deba detener la producción con urgencia la máquina dispone de botones de seguridad en su frontal y en el lateral izquierdo, que son más apropiados que el botón del formulario porque la detienen inmediatamente.

El *checkbox* "*Ver producción*" permite ver gráficamente qué componente se está tratando y en qué fase se encuentra, puesto que tras cada cambio en el estado de un componente se muestra una vista de la placa actualizada en su formulario de edición gráfica.

Debajo de este *checkbox* se muestran el total de componentes que contienen las placas seleccionadas y el total de componentes pendientes de procesar. Si se ha seleccionado la opción de reiniciar los componentes, ambas etiquetas mostrarán el mismo número de componentes, lógicamente.

En la parte inferior del formulario se encuentran una barra de progreso que se va completando a medida que se procesan los componentes y un área de texto que muestra un informe detallado en tiempo real del componente que se está produciendo. Además se va mostrando el paso concreto en que está su procesamiento a medida que se van produciendo cambios. También se muestran los comandos enviados a la máquina a través del puerto serie y la respuesta obtenida desde la misma, ambos con una marca horaria que ayude a la depuración de posibles errores durante la producción.

Como lo normal es que no sea necesario mostrar el informe detallado, el formulario se inicia contraído para evitar que dicho informe sea visible, pero se puede mostrar en cualquier momento, como se muestra en la figura 150. El formulario también se puede estirar para mostrar más líneas del informe en caso de necesidad. Para estirarlo basta coger el formulario por el borde con el ratón o usar el botón de maximizar, como cualquier ventana normal de Windows. Para contraer o estirar el formulario también se puede utilizar el *checkbox* "*Ver detalles*".

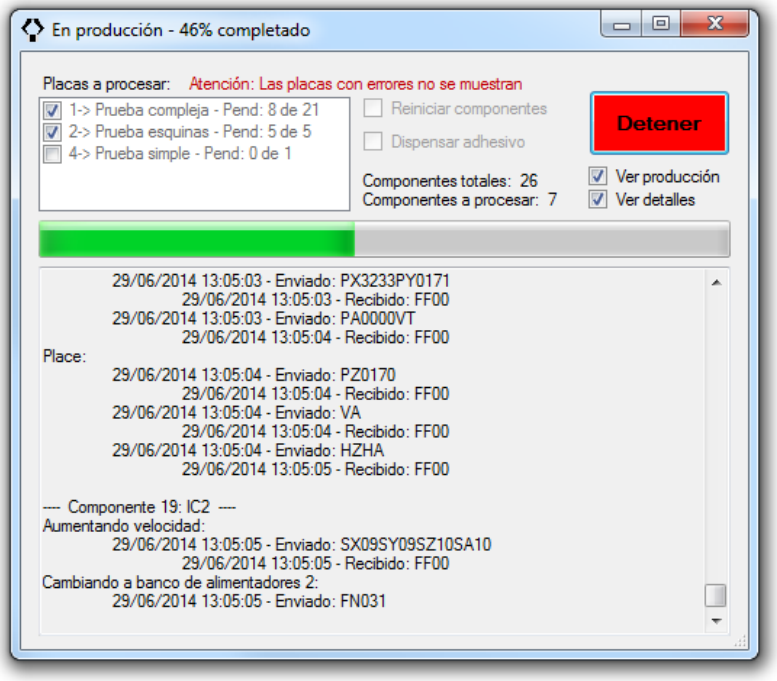


Figura 150. Formulario de producción con detalles de un montaje en curso.

5.17.5. Trabajo de producción en segundo plano

Resulta evidente que los programas secuenciales que constan de una única hebra de control resultan más fáciles de implementar y depurar que los programas con múltiples hebras que comparten, entre otros recursos, un mismo espacio de memoria.

En el caso de existir paralelismo, ya sea a través de procesos independientes o mediante hebras dentro de un proceso, la ejecución de un programa no es determinista porque depende del tiempo de CPU que le asigna el sistema operativo a cada hebra. La indeterminación introducida por el entrelazado de las operaciones de las distintas hebras provoca la aparición de errores difíciles de detectar y más aún de corregir.

El diseño correcto de una aplicación concurrente puede permitir que un programa complete una mayor cantidad de trabajo en el mismo periodo de tiempo, pero también sirve para la creación de interfaces de usuario que respondan mejor a las órdenes de éste. Cuando una aplicación tiene que realizar alguna tarea larga, su interfaz debería seguir respondiendo a las órdenes que el usuario efectúe. La ventana de la aplicación debería refrescarse y no quedarse en blanco. Los botones existentes para cancelar una operación deberían cancelar la operación de un modo inmediato y no al cabo de un rato, cuando la operación ya ha terminado de todos modos.

En un primer desarrollo del formulario de producción sucedía exactamente esto: la interfaz parecía haberse quedado congelada durante la producción, aunque la máquina hacía su

tarea, y no volvía a responder hasta que había terminado la tarea de producción encargada. Como consecuencia, el botón de cancelar era incapaz de detener el proceso de producción hasta que se había colocado la última pieza. Adicionalmente, si mientras se realizaba la producción se cambiaba a otro programa y se volvía de nuevo, la ventana no se refrescaba, quedando completamente en blanco hasta que terminaba la producción.

Lo más habitual al crear un programa utilizando una herramienta de desarrollo como Visual Studio es hacer una programación orientada a eventos en la que las tareas a realizar se desarrollan en los métodos manejadores de los eventos, como por ejemplo en el método manejador del evento *Click* de un botón. Estos eventos los provocan, entre otros, los controles del formulario cuando el usuario interactúa con la interfaz.

Un único hilo de ejecución se encarga de ir atendiendo la cola de eventos generados e ir saltando a los diferentes métodos manejadores de eventos en secuencia. Cuando se termina de ejecutar el método que gestiona un evento es cuando se pasa al siguiente.

Cuando el usuario hace clic sobre un botón se añade una entrada a la cola de eventos pendientes de ser atendidos (en realidad se añaden varias, puesto que la pulsación de un botón provoca varios eventos, pero admitamos esta simplificación para entender el problema). Si el hilo de ejecución estaba ocupado en un método que tarda en terminar su ejecución, se retrasará la acción que debía realizar el botón hasta que todos los eventos anteriores de la cola hayan terminado. Esto causaba que no funcionase el botón de cancelar la producción durante su ejecución.

Por otra parte, cuando un control necesita redibujarse en la pantalla, añade a la cola su evento *Paint*, pero mientras el hilo de ejecución no atienda al evento no se generará la imagen del control en la pantalla y por tanto no habrá refresco. Esto causa los desagradables efectos estéticos descritos anteriormente.

Jakob Nielsen [**40**] sugiere que cualquier operación que se pueda efectuar en una décima de segundo puede considerarse inmediata, por lo que no requerirá mayor atención por nuestra parte al implementar la aplicación. Cuando una operación se puede realizar por completo en un segundo, se puede decir que no interrumpe demasiado el trabajo del usuario, si bien resulta conveniente utilizar, al menos, una barra de progreso. Sin embargo, en operaciones que tarden varios segundos, el usuario difícilmente se mostrará paciente y será aconsejable que nuestra aplicación sea capaz de efectuar la operación concurrentemente con otras tareas.

El principal objetivo del uso de hilos o procesos es mejorar el rendimiento del sistema, y éste no siempre se mide en la cantidad de trabajo que se realiza por unidad de tiempo. De hecho, el uso de hilos se emplea a menudo para reducir el tiempo de respuesta al usuario de una aplicación.

Para mejorar la respuesta de la interfaz de usuario de nuestra aplicación se ha implementado un sistema de dos hilos ejecutándose en paralelo durante la producción. Uno de los hilos será el hilo principal de la aplicación, que se encargará de la interfaz de usuario. El otro hilo trabajará en segundo plano y se creará específicamente para llevar a cabo la tarea de producción.

Puesto que el montaje de los componentes de una composición de placas puede ser una tarea realmente larga (en función de la cantidad de componentes a procesar y las operaciones a realizar con cada componente), se optó por mostrar una barra de progreso y otros indicadores en el formulario de producción que permitan al usuario conocer los avances realizados. Con la implementación en dos hilos estos indicadores pasaron a ser realmente efectivos, permitiendo su actualización instantánea, desde el punto de vista del usuario. Además, se añadió la posibilidad de mostrar gráficamente las placas indicando en tiempo real, mediante colores, el estado en que se encuentra cada componente.

El formulario de producción se muestra de forma "*modal*", esto es, que durante el tiempo que se muestra no se puede trabajar con ningún otro formulario de la aplicación. Este tipo de formulario se utiliza a menudo en casi todos los programas para Windows, por ejemplo para los cuadros de diálogo de abrir un archivo, guardar, notificaciones al usuario, etc. En la figura 151 se muestra la sección de código de nuestra aplicación que crea nuestro formulario de producción y lo muestra de forma modal.

```
// Crea el formulario de producción y lo muestra de forma "modal"
FormProduccion fpr = new FormProduccion();
fpr.ShowDialog();
```

Figura 151. Creación del formulario modal de producción.

Se ha tomado esta decisión para evitar la posibilidad de que el operador genere inconsistencias en los datos almacenados en memoria y *condiciones de carrera* [**41**] entre los dos hilos que se ejecutan en paralelo durante la producción (imaginemos por ejemplo el efecto que podría tener que el usuario borre un alimentador utilizado durante la producción en curso).

Como sabemos, una *condición de carrera* se puede dar cuando dos hilos que se ejecutan en paralelo en un sistema tienen la capacidad de modificar simultáneamente los mismos datos. Si uno de los hilos lee un dato y lo modifica según su programa, pero antes de almacenarlo en su lugar en memoria el otro hilo lee y modifica el mismo dato según sus propios parámetros, esta segunda lectura no habrá tenido en cuenta la modificación anterior, así que ambos hilos guardarán su propia versión del dato modificado independientemente.

Cuando el último hilo en guardar almacene el dato en su posición de memoria sobrescribirá el dato actualizado previamente por el otro hilo sin haber tenido en cuenta dicha actualización. Esto lleva a una grave inconsistencia de datos en memoria que podría hacer inservibles los datos o tener consecuencias perniciosas [**42**] en la ejecución de los hilos, puesto que el resultado final no sólo depende del algoritmo implementado, sino de la temporización que asigne el planificador de tareas del sistema operativo a cada uno de los hilos que queremos ejecutar.

Se ha puesto especial cuidado en que los dos hilos de nuestra aplicación que se ejecutan simultáneamente durante la producción no modifiquen los mismos datos simultáneamente, especialmente que el hilo principal del programa no modifique ningún dato necesario para la producción. Aunque ambos hilos pueden necesitar acceder a todos los datos para leerlos, en cada caso sólo uno de los dos puede modificar los que le corresponden.

Sin embargo, esto se complicaría si se permitiese que el hilo principal del programa quede controlado por otro formulario, pues obligaría a establecer un mecanismo de integridad de datos y de control de acceso a éstos más complejo (utilizando bloqueos y secciones críticas, por ejemplo) y esto a su vez podría llevar a la aplicación a nuevos problemas propios de los sistemas concurrentes (interbloqueo, inanición, etc.).

Además, difícilmente existirá la necesidad de acceder a otros formularios como puede ser el editor de geometrías, alimentadores, componentes o placas durante una fase de producción.

En cualquier caso, la producción es una tarea suficientemente delicada y costosa para justificar la dedicación en exclusiva de la aplicación durante los minutos en que se llevará a cabo. Por todo esto se ha considerado que la manera más simple de evitar el problema es hacer este formulario *modal*.

Como alternativa a mostrar el formulario de forma modal tenemos la posibilidad de deshabilitar explícitamente mediante programación el resto de formularios de la aplicación, pero habría que tratar de una forma especial al formulario principal, puesto que si se deshabilita un

formulario MDI se deshabilitarán sus hijos, que incluiría a nuestro formulario de producción. Por tanto, en vez de deshabilitar el formulario principal habría que deshabilitar la barra de menús incluida en el. Sin embargo, esto acarrearía un nuevo problema a resolver que sería que, si el usuario maximiza el formulario de producción, los botones para minimizar, restaurar y cerrar el formulario quedarían deshabilitados también al pasar a alojarse en la barra de menús del formulario principal. Como vemos, la opción más sencilla es mostrar el formulario como modal, que resuelve todos los problemas cambiando una simple llamada a su método *Show* por *ShowDialog*.

Adicionalmente, mientras está activo el hilo secundario, se ha tomado la precaución de evitar que el usuario pueda cerrar el formulario puesto que, aunque lo hiciese, el segundo hilo se seguiría ejecutando normalmente hasta el final de la producción. Para hacer esto simplemente se ha implementado un método de respuesta a su evento *FormClosing*, cancelando el cierre la ventana en caso de que el hilo secundario estuviese trabajando, como se puede observar en la figura 152.

```
private void FormProduccion_FormClosing(object sender, FormClosingEventArgs e)
{
    if (backgroundWorkerProducir.IsBusy) e.Cancel = true;
}
```

Figura 152. Respuesta al evento FormClosing del formulario de producción.

En diversas pruebas que se realizaron se pudo comprobar que, al estar un hilo de la clase ejecutándose aún, el hecho de cerrar el formulario sólo hacía que dejase de mostrarse, pero el recolector de basura no destruía el objeto para recuperar su memoria, pues seguía estando activo, por lo que la producción iniciada continuaba normalmente. Además, si volvíamos a abrir este formulario mientras la producción seguía en curso (a partir de su referencia, guardada previamente), se mostraba exactamente igual que si no se hubiese cerrado.

La ventana estaba actualizada al estado de producción actual puesto que el segundo hilo seguía provocando los eventos de actualización, que el primer hilo atendía aún con la ventana no visible. Además, los formularios que mostraban gráficamente las placas en producción (también manejados por el primer hilo) seguían actualizándose correctamente, cambiando los colores de los componentes según se iban tratando.

Aunque en principio puede parecer que el cerrar el formulario no afecta negativamente al proceso de producción iniciado, recordemos que esta ventana la estábamos mostrando de forma modal para bloquear el uso del resto de la aplicación, evitando inconsistencias en los datos de

producción provocadas por un usuario descuidado. Si permitiésemos que ese formulario se cierre, automáticamente se vuelven a habilitar el resto de formularios, con lo que los datos estarían en peligro nuevamente.

Se consideró la posibilidad de utilizar el evento de cierre del formulario para enviar una solicitud de cancelación al, igual que si se pulsase el botón *Detener*, pero finalmente se decidió que era mejor idea obligar al usuario a cancelar la operación sólo a través de su botón. Así sería una acción menos impulsiva y más meditada.

5.17.6. El control *BackgroundWorker*

El hilo secundario de nuestra aplicación, que se encarga de realizar la producción, se ha implementado utilizando el control *BackgroundWorker*. Introducido en el .NET Framework 2.0, este control permite realizar operaciones computacionalmente costosas o duraderas en un hilo diferente al de la interfaz, de manera que la aplicación siga respondiendo al usuario mientras el trabajo se procesa en segundo plano.

De forma general es posible iniciar cualquier cantidad de hilos secundarios utilizando la clase *Thread* incluida en la plataforma .NET. Sin embargo, usando el control *BackgroundWorker*, la gestión de hilos queda encapsulada en el control. De esta manera el desarrollador no tiene que lidiar, entre otros, con la creación y control de los hilos, delegados (parecidos a los punteros a función del lenguaje C) o la invocación de los métodos de la manera correcta y desde el hilo correcto. Esto facilita bastante la tarea y hace el uso del control *BackgroundWorker* una opción muy atractiva.

Este componente incluye, entre sus múltiples características: soporte de cancelación, gestión de errores, información de progreso (que nos permitirá el paso de información al hilo principal) y detección de la finalización de la tarea en segundo plano. Además, al ser un control que podemos incluir en el formulario en tiempo de diseño, podemos modificar sus propiedades de diseño desde el entorno Visual Studio y será éste el que genere las líneas de código necesarias, disminuyendo la probabilidad de cometer errores de programación.

El control se encarga de generar eventos en el hilo adecuado según las llamadas que el desarrollador realiza a sus métodos de disparo. De esta forma, su utilización es tan simple como rellenar los métodos de respuesta a dichos eventos con el código que se quiere ejecutar en cada caso.

En el código del formulario de producción se han incluido tres métodos necesarios para trabajar con este control. El método *DoWork*[3] es el que se encarga de realizar la tarea de producción en segundo plano. El método *ProgressChanged* se utiliza para informar al hilo principal cada vez que hay un cambio en el progreso de la producción. Por último el método *RunWorkerCompleted* es llamado una vez terminada la ejecución del método *DoWork*.

Los métodos *ProgressChanged* y *RunWorkerCompleted* se ejecutan en el hilo principal de la aplicación, mientras que el método *DoWork* es el único que se ejecuta en el hilo secundario. La ejecución de estos métodos se dispara mediante eventos provocados por el control que podemos inducir mediante llamadas a los métodos del control que gestionan el cambio de contexto.

Para que el control *BackgroundWorker* lance el hilo secundario ejecutando el método *DoWork* hay que llamar a su método *RunWorkerAsync*. Debe evitarse a toda costa la modificación de cualquier control de la interfaz de usuario desde el método *DoWork*, puesto que generaría una excepción no controlada que detendría el programa. Cada vez que se desea actualizar la interfaz de usuario desde el método *DoWork* se debe llamar al método *ReportProgress* que pasará los datos que deseemos al método *ProgressChanged.* Como éste último se ejecuta en el hilo principal puede hacer cambios en los controles de la interfaz de usuario sin provocar excepciones.

Para evitar confusiones, se decidió reordenar el código de la clase, agrupando los métodos que se ejecutan en el hilo principal al principio del listado y los que se ejecutan en el hilo secundario (*DoWork* y los llamados dentro de éste) al final, como se muestra en la figura 153, donde se han contraído todos los elementos para que quepan en una imagen y se han identificado los grupos en color verde y rojo respectivamente.

Típicamente, se suele pasar a *ProgressChanged* únicamente un entero que indica el porcentaje de trabajo realizado, aunque también se puede añadir un objeto que podría contener cualquier cantidad de datos. En nuestro caso siempre pasamos como primer parámetro el porcentaje entero de componentes tratados en relación al total de componentes pendientes.

Sin embargo, cuando un componente cambia de estado, si está marcado el *checkbox* rotulado “Ver producción”, añadimos un segundo parámetro para enviar en qué placa está. Así *ProgressChanged* puede detectarlo y llamar al método *mostrarPlaca* de la placa afectada para actualizar la vista gráfica de la placa y hacer que el componente cambie de color en la pantalla.

[3] El nombre predeterminado de los métodos relacionados con este control es mucho más grande, pero en interés de una mayor claridad se ha recortado en este texto, dejando únicamente la terminación, que es la parte que mejor describe su función. Así, el nombre original de este método en el código de nuestra aplicación es *backgroundWorkerProducir_DoWork*, pero en este texto se hace referencia a el como *DoWork* por claridad. Lo mismo se ha aplicado al resto de elementos de este control.

Figura 153. Métodos del hilo principal y del hilo secundario.

Cuando *DoWork* termina su ejecución pasa a ejecutarse *RunWorkerCompleted* en el hilo principal, que se encarga de devolver el aspecto original al formulario, actualizarlo y desbloquear los controles que se bloquearon al iniciar la producción para evitar que el usuario cambiase datos durante ésta. Además detecta si la salida del hilo secundario se realizó porque se completó la

producción, se canceló ésta o se produjo una excepción que forzó la terminación y muestra una ventana con el mensaje correspondiente.

En caso de que se completase la producción con algunos componentes sin colocar debido a errores durante la producción, se avisa al operador para que lo tenga en cuenta y examine los resultados con mayor detalle.

Cuando desde el hilo principal se desea cancelar la ejecución del hilo secundario se le debe notificar al *BackGroundWorker* llamando al método *CancelAsync*. En nuestra aplicación esto se producirá cuando se pulse el botón de detener la producción (que es el mismo que para iniciarla, pero con otro aspecto, como se vio anteriormente).

Esto hará que se active la propiedad *CancellationPending* del *BackgroundWorker*, que puede ser comprobada periódicamente desde el método *DoWork* para ser detectada y realizar una salida "limpia". Si en el método *DoWork* detectamos la activación de *CancellationPending*, antes de salir activaremos su parámetro e.*Cancel*. De esta forma, cuando se produzca la llamada a *RunWorkerCompleted* estará activado su parámetro *e.Cancelled* y sabrá que ese fue el motivo de la terminación del hilo.

En caso de que se produjese una excepción no gestionada dentro del método *DoWork* se dará fin a la ejecución de éste y la excepción será capturada por el *BackgroundWorker*. Al ejecutarse la llamada a *RunWorkedCompleted* éste tendrá activado el parámetro *e.Error*, indicando la excepción que provocó la finalización del hilo. Esto lo aprovechamos en nuestra aplicación para notificar a *RunWorkedCompleted* que se ha interrumpido la producción debido a un error no recuperable e informar al usuario.

5.17.7. Inicio y cancelación del proceso de producción

Al pulsar el botón de inicio se ejecuta el método de respuesta al evento *Click*, cuyo código se muestra en la figura 154. Lo primero que se hace en el código mostrado es comprobar si el *BackgroundWorker* está ocupado. Si lo estuviera, querría decir que ya habíamos pulsado el botón Iniciar anteriormente para empezar la producción y éste habría cambiado, siendo ahora el botón de cancelación.

En ese caso, se deshabilita el botón para impedir nuevas pulsaciones y se cambia su apariencia para notificar al usuario que se ha iniciado el proceso de cancelación de la producción y que debe esperar a que finalice el procesamiento del componente en curso. Finalmente se llama al método *CancelAsync* del *BackgroundWorker* para solicitar la cancelación, como se explicó anteriormente, y se retorna.

Si al comprobar el estado del *BackgroundWorker*, éste no estaba ocupado quiere decir que debemos iniciar la producción, por lo que se comprueba si hay algún impedimento para comenzarla. Los impedimentos comprobados son que no haya ningún componente pendiente de ser colocado entre las placas seleccionadas o que no se haya configurado un puerto serie válido. Si alguna de estas comprobaciones falla se retorna avisando antes al usuario mediante un cuadro de diálogo para que sepa por qué no se inicia la producción.

En caso de que se pueda iniciar la producción, se comienza por deshabilitar los controles que permiten cambiar las condiciones de la producción, que son la lista de placas y el *checkbox* para reiniciarlas. Si el usuario desea cambiar los parámetros de producción primero deberá detener la producción actual.

```
private void buttonIniciar_Click(object sender, EventArgs e)
{
    // Comprueba si hay una producción en curso para saber si se va a cancelar o iniciar
    if (backgroundWorkerProducir.IsBusy)
    {
        buttonIniciar.Enabled = false;
        buttonIniciar.Text = "Espere";
        buttonIniciar.BackColor = Color.Gray;

        // Solicita cancelar la producción
        backgroundWorkerProducir.CancelAsync();
    }
    else
    {
        if (totalPendientes < 1)
        {
            MessageBox.Show("No hay componentes pendientes de montar.",
                        "No hay componentes pendientes",
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
        else
        {
           // Comprueba si el puerto serie está disponible para empezar
            if (!datos.puertoSerieDisponible)
            {
                MessageBox.Show("No se ha seleccionado un puerto serie.\nLa comunicación con
                        la máquina Pick & Place no será posible mientras no se configure un
                        puerto serie válido (menú Configuración).", "Puerto serie no
                        disponible", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                // Bloquea los controles para evitar cambios durante la producción
                checkBoxReiniciar.Enabled = false;
                checkedListBoxPlacas.Enabled = false;

                buttonIniciar.Text = "Detener";
                buttonIniciar.BackColor = Color.Red;
                textBoxRegistro.Text = "";

                // Inicia el hilo de producción
                backgroundWorkerProducir.RunWorkerAsync();
            }
        }
    }
}
```

Figura 154. Código fuente del método *buttonIniciar_Click*.

A continuación se cambia el aspecto del botón "*Iniciar*" para que pase a ser el botón "*Detener*". Por último, se limpia el *textbox* que mostrará el registro de sucesos durante la producción y se inicia el hilo de producción realizando la llamada a *RunWorkerAsync*.

5.17.8. Método *DoWork*

Tras la llamada a *RunWorkerAsync*, se inicia la ejecución de método *DoWork* en un segundo hilo, como vimos anteriormente. El código de este método se puede ver en la figura 155 donde, analizándolo de forma general, se observa que al principio hace algunas inicializaciones necesarias y posteriormente, dentro de un bloque *try* entra en un bucle, montando las placas seleccionadas por orden. Para montar los componentes de cada placa, *DoWork* llama al método *producirPlaca*, indicándole la placa a producir. Tras cada llamada al método *producirPlaca* se comprueba si se solicitó cancelar el trabajo de producción para salir anticipadamente del bucle en ese caso.

Tras terminar la ejecución del bucle, se "*aparca*" la máquina, desmontando el útil del cabezal, si lo hubiese, devolviendo el cabezal al origen y seleccionando el segundo banco de alimentadores. Se selecciona éste y no el primero porque en el modelo de la mesa de alimentadores disponible en el laboratorio es el que la deja en una posición centrada. Por último, se da por finalizada la ejecución del segundo hilo tras actualizar el progreso alcanzado.

En caso de un error grave que aborte la producción mediante una excepción, en el bloque *catch* se registra el motivo del error y se relanza la excepción para que sea capturada también por *RunWorkerCompleted* e informe al operador, como se explica más adelante, en el epígrafe dedicado al manejo de errores.

Analizando el código con algo más de detalle, vemos que lo primero que hace el método *DoWork* es guardar la referencia al *BackgroundWorker* que sostiene el hilo en el que está funcionando en un campo de la clase para que esté disponible para cualquier método llamado durante la ejecución de este hilo.

A continuación se inicializan algunos de los campos que guardan el estado, como son el número de componentes tratados y de componentes que no han podido ser montados y la velocidad actual de todos los ejes. Ésta se inicializa a cero para obligar al método *establecerVelocidadMáxima* a enviar todos los comandos de velocidad siempre al iniciar una producción, en previsión de que alguno tuviese otro valor inadvertidamente.

```
private void backgroundWorkerProducir_DoWork(object sender, DoWorkEventArgs e)
{
    worker = sender as BackgroundWorker;
    componentesTratados = 0;
    componentesFallados = 0;
    velocidadXActual = 0;
    velocidadYActual = 0;
    velocidadZActual = 0;
    velocidadAActual = 0;
    registro = DateTime.Now.ToString() + "  -###- Iniciada la producción -###-\r\n\r\n";
    datos.puertoSerie.DiscardOutBuffer();
    datos.puertoSerie.DiscardInBuffer();
    try
    {
        // Secuencia de inicialización de la Cosy, inicialización de los ejes,
        // establecimiento de los tiempos de retardo y las velocidades de los ejes.
        transmitir("OO");
        transmitir("HFHZHXHYHAVATAT10030T20010T30020T40030T50150");
        esperarConfirmación(10);
        worker.ReportProgress(0);

        // Produce cada placa en secuencia.
        for (int n = 0; n <= checkedListBoxPlacas.CheckedIndices.Count - 1; n++)
        {
            int índicePlaca = (int)índicesDePlacasMostradas[
                              checkedListBoxPlacas.CheckedIndices[n]];
            producirPlaca(índicePlaca);

            // Comprueba si se ha pulsado el botón de detener.
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                registro += "\r\n" + DateTime.Now.ToString() +
                              " -###- Producción cancelada -###-\r\n\r\n";
                break;
            }
        }

        // Al terminar desmonta el útil si había alguno y pone en su
        // posición inicial al cabezal y a la bandeja de alimentadores.
        registro += "\r\nAparcando...\r\n";
        establecerVelocidadMáxima();
        desmontarÚtilActual();
        transmitirYConfirmar("HFHZHXHYHAVA");
        seleccionarBanco(2);
    }
    // Captura las excepciones para registrar los errores y las relanza para abortar y
    // mostrar un mensaje de error en RunWorkerCompleted.
    catch (PeligroException)
    {
        registro += "\t\t" + DateTime.Now.ToString() +
                    "#### PELIGRO DE INTEGRIDAD ####\r\n";
        throw;
    }
    catch (TimeoutException)
    {
        registro += "\t\t" + DateTime.Now.ToString() +
                    "#### ERROR TIMEOUT PUERTO SERIE ####\r\n";
        throw;
    }

    if (!e.Cancel) registro += "\r\n" + DateTime.Now.ToString() +
                    " -###- Producción finalizada -###-\r\n\r\n";
    worker.ReportProgress(porcentajeTerminado());
}
```

Figura 155. Código fuente del método *DoWork*.

Seguidamente se inicializa el registro de producción con un mensaje que indica la fecha y hora de inicio de la producción. Todas las acciones llevadas a cabo durante la producción se van registrando en un campo de tipo *string* llamado "*registro*" y su contenido es volcado en el *textbox* que muestra el informe de producción cada vez que se produce una actualización del estado del *BackgroundWorker* (en la llamada a *ProgressChanged*).

En este punto del método *DoWork* se inicializa el campo *registro*, pero todos los métodos de la clase implicados en la producción van escribiendo en él las tareas que se van llevando a cabo, así como los errores detectados. De esta manera, en caso de que se produzca un error, el operador pueda analizar los pasos realizados para detectar cuál ha sido la fuente del mismo.

Además, al terminar la producción se graba automáticamente el contenido del registro en un archivo *.log* en disco desde el método *RunWorkerCompleted*, por lo que el operador puede postergar el análisis de lo sucedido a otro momento más conveniente. El registro de producción se guarda en una carpeta llamada *COSYLog* dentro de la carpeta de trabajo actual y no sobrescribe registros anteriores, sino que siempre se genera un fichero nuevo a partir del nombre del fichero de la composición en que se está trabajando y un número autoincrementado.

Además de registrar todos los pasos dados sobre cada uno de los componentes, como se mencionó anteriormente, se registra todo el tráfico con el puerto serie, añadiéndole marcas horarias que permitan un análisis lo más completo posible.

Todas las operaciones de producción siguientes se realizan dentro de un bloque *try*. Esto se ha hecho así para simplificar la gestión de errores, como se explica más adelante.

Lo primero que se hace en dicho bloque es enviar los comandos de inicialización a la Cosy. La secuencia enviada incluye varios comandos encadenados, puesto que la máquina es capaz de separarlos y ejecutarlos correctamente por orden. La secuencia incluye la inicialización de todos los ejes, incluido el vacío en la boquilla y el establecimiento de los tiempos de retardo para varias operaciones básicas (tiempos de espera para el test de vacío, retardo de colocación y tiempos de apertura y cierre de las pinzas en las operaciones de centrado).

Como se puede ver en el código, la transmisión de comandos a la máquina se realiza mediante un método llamado *transmitir*. El código de este método es muy simple y se puede ver en la figura 156.

Lo único que hace el método *transmitir* es registrar lo que se quiere enviar al puerto serie con una marca horaria y posteriormente enviarlo, completando el envío con un código de "*Enter*"

(carácter ASCII número 13) mediante la llamada al método *WriteLine*. Esto se ha hecho así para ahorrarnos añadir manualmente dicho carácter, pues es imprescindible al ser lo que indica a la Cosy que ha terminado el mensaje transmitido.

```
private void transmitir(string bufferTX)
{
    registro += "\t" + DateTime.Now.ToString() + " - Enviado: " + bufferTX + "\r\n";
    datos.puertoSerie.WriteLine(bufferTX);
}
```

Figura 156. Código fuente del método *transmitir*.

Tras la transmisión, es necesario esperar confirmación desde la Cosy que nos indique si todo ha ido bien. Esto se hace en la llamada al método *esperarConfirmación*, que puede incluir como parámetro un tiempo máximo de espera en segundos, como en el código que estamos analizando. El método *esperarConfirmación*, cuyo código se puede ver en la figura 173, es importante para la detección de errores y se analiza con detalle más adelante, en el epígrafe sobre el tratamiento de errores de producción, pero baste decir aquí que si regresa significa que todo ha ido bien y la Cosy ha confirmado que el comando se ha ejecutado con éxito.

Para terminar la tarea de inicialización, el hilo de producción hace su primera notificación de progreso indicando un 0% de avance. De esta manera se inicializa la barra de progreso, que estaba en un estado indeterminado, puesto que podíamos haber iniciado una producción anterior que la hubiese dejado con otro valor.

En el bucle *for* que sigue a continuación, se averigua el número de placa de cada placa seleccionada en el *checkedListBox* y se llama al método *producirPlaca* con el número de placa.

De este método se puede regresar porque se ha terminado de montar los componentes de la placa o bien porque hay una petición de cancelación pendiente de ser atendida. Por tanto lo siguiente que se hace es comprobar el estado de la propiedad *CancellationPending* del *BackgroundWorker* y si había una solicitud de cancelación, se atiende.

La atención de la solicitud consiste en activar el parámetro *e.Cancel* para que a *RunWorkerCompleted*, en el hilo principal, le llegue la notificación de que se atendió la cancelación de la tarea, se registra y se finaliza la ejecución del bucle mediante un *break* para no seguir montando placas.

Una vez fuera del bucle, tanto si se ha terminado la tarea correctamente como si se canceló, se aparca la máquina. Para ello se establece la velocidad máxima de movimientos para

desmontar el útil que estuviese montado en la aguja, reiniciar el cabezal (que lo llevará al origen de coordenadas) y seleccionar el 2º banco, como se mencionó anteriormente.

Para enviar los comandos de inicialización del cabezal (y muchos otros en el resto de esta clase) se usa el método *transmitirYConfirmar* en lugar del método *transmitir* visto anteriormente. El código de este método se puede ver en la figura 157, donde se puede ver que simplemente llama en secuencia a los métodos *transmitir* y *esperarConfirmación*.

```
private void transmitirYConfirmar(string bufferTX)
{
    transmitir(bufferTX);
    esperarConfirmación();
}
```

Figura 157. Código fuente del método *transmitirYConfirmar*.

Como las operaciones con la máquina mencionadas anteriormente están dentro de un bloque *try*, en caso de producirse alguna excepción no gestionada internamente, se lanzará la ejecución del bloque catch que coincida con el tipo de excepción producida.

En este punto se capturan sólo las excepciones PeligroException y TimeoutException únicamente para registrar el motivo de la interrupción de la tarea de interrupción, así que ambas se vuelven a relanzar tras ser registradas. En el epígrafe sobre el tratamiento de errores se tratará su utilidad con más profundidad.

Lo último que hace el método DoWork antes de retornar es informar del progreso alcanzado mediante una llamada al método ReportProgress del BackgroundWorker. De esta forma se actualizará la barra de progreso y el número de componentes pendientes antes de dar por finalizado el hilo.

5.17.9. Montaje de los componentes

5.17.9.1. Pasos necesarios para realizar el montaje

Para realizar el montaje correcto de cada componente, es necesario seguir una serie de pasos, algunos de ellos variables según las diferentes opciones posibles para cada componente. Para el desarrollo de este proyecto se ha estudiado con detenimiento los métodos utilizados por el software antiguo para realizar muchas de las tareas más comunes, extrayendo procedimientos que podían sernos útiles en el nuevo desarrollo.

A continuación se muestran por orden todos los pasos que pueden ser necesarios para el montaje de un componente:

1. Establecer las velocidades máximas en todos los ejes.
2. Inicializar ángulo y altura de la aguja.
3. Dispensar el adhesivo (si es necesario).
4. Cambiar el útil de la aguja al adecuado para este componente.
5. Seleccionar el banco de alimentadores donde se encuentra el componente.
6. Mover el cabezal hasta la posición de este componente en el alimentador.
7. Establecer la velocidad de la aguja para coger el componente
8. Bajar la aguja hasta la altura de pick de este componente.
9. Activar el vacío para coger el componente.
10. Subir la aguja hasta la altura de transporte de este componente.
11. Activar el avance de componente del alimentador.
12. Hacer test de pérdida del componente, para desechar el componente en caso de fallo y volver a coger otro igual (vuelta al punto 8).
13. Establecer la velocidad de transporte y giro para este componente.
14. Centrar el componente en la estación de centrado.
15. Mover el cabezal hasta la posición de colocación de este componente.
16. Centrar el componente con las pinzas del cabezal.
17. Giro del componente a su ángulo de colocación.
18. Hacer test de pérdida del componente, para desechar el componente y repetir su captura en caso de fallo.
19. Bajar la aguja hasta la altura de colocación.
20. Desactivar el vacío en la aguja.
21. Subir la aguja.

El primer punto es una optimización de tiempos no imprescindible para el montaje correcto de los componentes, pero que hemos añadido porque puede hacer más rápida la producción si se encuentran intercalados componentes de montaje lento con componentes que se pueden montar a alta velocidad. La velocidad a la que se puede montar cada componente depende básicamente de lo estable que sea tras capturarlo con la aguja del cabezal (debido a su masa e inercias), por lo que los componentes pequeños y ligeros se podrán manejar mucho más rápidamente que los grandes y pesados.

Por otra parte, como normalmente tanto tras las operaciones de inicialización de la máquina como tras soltar un componente se deja la aguja inicializada correctamente, es innecesario enviar comandos para realizar este paso explícitamente, pudiéndonos saltar el punto 2 en nuestra implementación.

Además, los pasos 4, 5, 7, 11, 13, 14, 16 y 17 dependen de lo que necesite cada componente (y algunos de ellos además del estado en ese momento de la máquina) y por tanto puede que no sea necesario ejecutarlos.

Por otra parte, en caso de que sea necesario desechar un componente debido a un fallo en el test de pérdida, se llevará hasta la zona de desecho para soltarlo allí. Si no se hiciese así, el componente podría caer sobre un área sensible y provocar daños mayores. Por ejemplo, si es un componente algo voluminoso y cae sobre una de las placas que se está montando puede desplazar componentes ya colocados o tapar áreas pendientes, impidiendo así la correcta producción de la placa. Esto obligaría a rehacer manualmente parte del trabajo, lo que llevaría a retrasos y mayores costes de producción que el que supone desechar un componente sospechoso.

Para desechar un componente es necesario seguir también una serie de pasos que se muestran a continuación:

1. Subir la aguja para minimizar el riesgo de tropiezos (si no estaba ya subida).
2. Mover el cabezal hasta la posición de desecho.
3. Desactivar el vacío.
4. Ejecutar los movimientos de liberación del componente, para que si está levemente atascado caiga.

En nuestro caso, el test de pérdida siempre se hace con el cabezal subido a la altura de transporte del componente, por lo que no se envía a la máquina el comando para ejecutar el primer paso. Además, se ha tenido en cuenta un número máximo de reintentos cuando falla el test de pérdida para evitar que el proceso deje de avanzar si encuentra un componente conflictivo.

Por su parte, el centrado de un componente en la estación de centrado requiere también realizar una serie de pasos ordenados para hacerlo correctamente, que son mostrados a continuación:

1. Llevar el componente al centro de la estación de centrado.
2. Bajar el componente.
3. Desactivar el vacío.
4. Subir la aguja.
5. Activar el centrado (la Cosy ejecuta su rutina de centrado automáticamente).
6. Bajar la aguja.

7. Activar el vacío para coger nuevamente el componente ya centrado.
8. Subir la aguja.
9. Mover el cabezal fuera de la estación de centrado.

También se ha estudiado el procedimiento que realiza la máquina para montar y desmontar los útiles de la aguja, obteniendo la siguiente lista de pasos para montar un útil:

1. Inicializar altura y ángulo de la aguja.
2. Abrir la puerta del panel de útiles.
3. Llevar el brazo a la posición de entrada al panel de útiles.
4. Llevar el brazo hasta el útil solicitado.
5. Bajar la aguja hasta la posición de captura del útil.
6. Inicializar altura y ángulo de la aguja.
7. Cerrar la puerta del panel.

En el caso del desmontaje del útil, se observó que la única diferencia con el montaje era que se cerraba la puerta del panel antes de subir la aguja, o sea, se invertían los puntos 6 y 7 de la lista anterior.

5.17.9.2. Implementación de los pasos

Como las operaciones de desecho y de centrado en la estación son siempre iguales, e independientes de otras, en la implementación se han incluido en métodos separados para facilitar la creación y prueba del código de la aplicación. También por conveniencia se han creado métodos separados para otras operaciones más o menos independientes, como son el montaje y desmontaje de los útiles, el establecimiento de las velocidades máximas y la selección del banco de alimentadores, tratados más adelante en este epígrafe.

El montaje de los componentes se controla e inicia desde el método producirPlaca. El método *producirPlaca*, cuyo código se puede ver en la figura 158, es llamado desde *DoWork* y recibe como parámetro el número de placa que debe montar.

Por conveniencia, al principio se inicializa el campo *placaActual* con la estructura de datos de la placa que se debe montar. Así estará disponible para este método y los llamados a partir de él y de esta forma se acortan las referencias a los datos, simplificando el código y reduciendo la posibilidad de cometer errores.

A continuación, en caso de que se hubiese marcado la opción "*Reiniciar componentes*", se inicializa el estado de todos los componentes de la placa, para después entrar en un bucle *while* en el que se monta cada componente de la placa por orden.

```
private void producirPlaca(int p)
{
    registro += "\r\n## Placa " + p + " en producción ##\r\n";
    placaActual = datos.placas[p];

    // Reinicia el estado de los componentes de la placa si se marcó el checkBox.
    if (checkBoxReiniciar.Checked) datos.reiniciarPlaca(p);

    // Procesa todos los componentes de la placa en secuencia.
    int reintentosComponenteActual = 0;
    int c = 0;   // Índice del componente
    while (c < placaActual.componentes.Length)
    {
        try
        {
            // Comprueba si este componente ya ha sido colocado.
            if (placaActual.componentes[c].estado == estadoComponente.Colocado)
            {
                registro += "\r\n---  Componente " + c + ": " +
                        placaActual.componentes[c].nombre +
                        "---\r\n--- Ya estaba colocado. ---\r\n";
            }
            else
            {
                colocarComponente(c, p);
                componentesTratados++;
                reintentosComponenteActual = 0;
            }
            worker.ReportProgress(pocentajeTerminado());
            c++;
        }
        catch (PérdidaException)
        {
            // Captura la excepción de pérdida del componente para recuperarse y continuar
            registro += "#### ERROR DE PÉRDIDA ####\r\n";
            desecharComponente(); // Desecha el componente que dio error de pérdida.
            if (reintentosComponenteActual > datos.reintentosMax)
            {
                // Si excede el nº de reintentos, lo añade a la cuenta, lo marca y prosigue.
                if (placaActual.componentes[c].estado == estadoComponente.Cogiendo)
                        placaActual.componentes[c].estado = estadoComponente.Error_Cogiendo;
                if (placaActual.componentes[c].estado == estadoComponente.Colocando)
                        placaActual.componentes[c].estado = estadoComponente.Error_Colocando;
                componentesFallados++;
                componentesTratados++;
                reintentosComponenteActual = 0;
                c++;
            }
            else
            {
                reintentosComponenteActual++;
            }
        }
        if (worker.CancellationPending) return;
    }
}
```

Figura 158. Código fuente del método *producirPlaca*.

Para montar los componentes, primero comprueba si el componente que se va a tratar ya estaba colocado y, si no lo estaba, llama al método *colocarComponente* y actualiza el número de

componentes tratados. Por último provoca la actualización del interfaz e incrementa el índice para tratar al siguiente componente.

Antes de pasar a tratar al siguiente componente, se comprueba si la solicitud de cancelación del *BackgroundWorker* está activada. Si lo está, simplemente retorna a *DoWork*, que será donde será tratada dicha solicitud, como vimos anteriormente.

En caso de que durante las operaciones necesarias para colocar el componente se detecte un fallo en el test de vacío del cabezal (test de pérdida), en este método se capturará la excepción que se producirá y se reintentará la colocación el número de veces que indique el campo *reintentosMax*, almacenado con el resto de parámetros de operación de nuestra aplicación en la clase *datos*. Por defecto este campo está inicializado con 2 reintentos, por lo que se intentará colocar el componente tres veces antes de ignorarlo y pasar al siguiente componente.

En caso de que un componente tenga que ser ignorado por sobrepasar el número de reintentos máximo, se incrementará el campo *componentesFallados* para avisar al operador al final del proceso. Además, el estado del componente será actualizado a *Error_Cogiendo*, si falló el test de vacío mientras estaba en estado *Cogiendo*, o bien a *Error_Colocando*, si el fallo se produjo en el estado *Colocando*.

En cualquier caso, se realiza una operación de desecho del componente por si está aún en el cabezal en una postura que impide colocarlo correctamente, como por ejemplo cogido por un vértice o una arista. La operación de desecho consiste en llevar el componente al área de desecho, desactivar el vacío en la boquilla y provocar las sacudidas del cabezal apropiadas, como se vio anteriormente, y el código de este método se puede ver en la figura 159.

```
private void desecharComponente()
{
    registro += "Desechando componente:\r\n";
    transmitirYConfirmar("PX0020PY0020VAHZHZ");
}
```

Figura 159. Código fuente del método *desecharComponente*.

Como se dijo anteriormente, el proceso de colocar un componente individual sobre una placa se realiza llamando al método *colocarComponente*. El método *colocarComponente* empieza el proceso de colocación inicializando algunas variables locales, como se puede ver en el código de la figura 160.

```
private void colocarComponente(int c, int p, int reintento)
{
    string bufferTX = "";
    componente componenteActual = placaActual.componentes[c];
    alimentador alimentadorActual = datos.alimentadores[componenteActual.alimentador];
    registro += "\r\n----  Componente " + c + ": " + componenteActual.nombre + "  ----\r\n";
    if (reintento > 0)
    {
        registro += "----  Reintento " + reintento.ToString() + "  ----\r\n";
    }

    //Maximiza velocidades y dispensa las gotas de pegamento que requiera este componente
    establecerVelocidadMáxima();
    if (checkBoxDispensarAdhesivo.Checked && (alimentadorActual.gotasDispensador > 0) &&
        (reintento == 0)) dispensarAdhesivo(componenteActual);

    // Pone el componente en estado "Cogiendo" y lo muestra en pantalla:
    placaActual.componentes[c].estado = estadoComponente.Cogiendo;
    if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);

    // Selecciona banco de alimentadores y coloca  el útil adecuado en el cabezal:
    seleccionarBanco(alimentadorActual.banco);
    montarÚtil(alimentadorActual.útil);

    // Coloca el cabezal sobre el alimentador:
    registro += "Posicionando Pick:\r\n";
    bufferTX = "PX" + cuatroDígitos(datos.xMax - alimentadorActual.x)
               + "PY" + cuatroDígitos(alimentadorActual.y);
    transmitirYConfirmar(bufferTX);

    // Establece la velocidad de elevación y coge el componente:
    registro += "Pick:\r\n";
    bufferTX = "";
    if (velocidadZActual > alimentadorActual.velocidadZ)
    {
        velocidadZActual = alimentadorActual.velocidadZ;
        bufferTX = "SZ" + dosDígitos(velocidadZActual);
    }
    bufferTX += "PZ" + cuatroDígitos(alimentadorActual.zPick)
        + "VEPZ" + cuatroDígitos(alimentadorActual.zMover);

    // Avance del alimentador:
    if (alimentadorActual.avanceAlimentador) bufferTX += "FA001";

    // Test de vacío:
    bufferTX += "VT";
    transmitirYConfirmar(bufferTX);

    // Pone el componente en estado "Colocando" y lo muestra en pantalla si procede:
    placaActual.componentes[c].estado = estadoComponente.Colocando;
    if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);

    // Establece la velocidad de transporte y giro del componente:
    bufferTX = "";
    if (velocidadXActual > alimentadorActual.velocidadX)
    {
        velocidadXActual = alimentadorActual.velocidadX;
        bufferTX = "SX" + dosDígitos(velocidadXActual);
    }
    if (velocidadYActual > alimentadorActual.velocidadY)
    {
        velocidadYActual = alimentadorActual.velocidadY;
        bufferTX += "SY" + dosDígitos(velocidadYActual);
    }
    if (velocidadAActual > alimentadorActual.velocidadA)
    {
        velocidadAActual = alimentadorActual.velocidadA;
        bufferTX += "SA" + dosDígitos(velocidadAActual);
    }
    if (bufferTX != "")
    {
```

```
            registro += "Reduciendo velocidad:\r\n";
            transmitirYConfirmar(bufferTX);
        }

        // Centrado en la estación de centrado:
        if (alimentadorActual.centradoEstación) centrarEnEstaciónDeCentrado();

        // Lleva el cabezal hasta la posición de colocación del componente:
        Point coordenadasPlace = calculaCoordenadasPlace(componenteActual);
        bufferTX = "PX" + cuatroDígitos(datos.xMax - (int)origenX) + "PY" +
                    cuatroDígitos((int)origenY);
        transmitirYConfirmar(bufferTX);

        // Centrado con las pinzas del cabezal:
        bufferTX = "";
        if (alimentadorActual.centradoPinzasX) bufferTX += "ZX";
        if (alimentadorActual.centradoPinzasY) bufferTX += "ZY";
        if (bufferTX != "")
        {
            registro += "Centrando con pinzas:\r\n";
            transmitirYConfirmar(bufferTX);
        }

        // Giro del componente y test de vacío:
        bufferTX = "";
        int anguloDeGiro = (int)placaActual.t + componenteActual.t;
        if (anguloDeGiro != 0) bufferTX = "PA" + cuatroDígitos(anguloDeGiro);
        bufferTX += "VT";
        transmitirYConfirmar(bufferTX);

        // Pone el componente sobre la placa y retira la aguja:
        registro += "Place:\r\n";
        transmitirYConfirmar("PZ" + cuatroDígitos(alimentadorActual.zPlace));
        transmitirYConfirmar("VA");
        transmitirYConfirmar("HZHA");

        // Pone el componente en estado Colocado y lo muestra en pantalla si procede.
        placaActual.componentes[c].estado = estadoComponente.Colocado;
        if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);
    }
```

Figura 160. Código fuente del método *colocarComponente*.

La variable local *bufferTX* se usa como buffer temporal para componer comandos complejos, en los que existen partes opcionales, antes de ser enviados por el puerto serie. En *componenteActual* se guarda una copia de los datos del componente que se está montando y en *alimentadorActual* estarán los datos del alimentador en el que se encuentra el componente actual. Tanto *componenteActual* como *alimentadorActual* no son estrictamente necesarias, pero se han incluido para simplificar el código, haciendo significativamente más cortas algunas líneas de código al simplificar las referencias a los datos necesarios y reduciendo por tanto la probabilidad de cometer errores.

A continuación se realiza la optimización de velocidad, mencionada cuando se habló de la lista de pasos necesarios, llamando al método *establecerVelocidadMáxima*, cuyo código se puede ver en la figura 161. Este método restablece la velocidad máxima en todos los ejes en que ésta se hubiese cambiado en alguna operación anterior. Para ello simplemente verifica que los campos de seguimiento de la velocidad actual no sean distintos a las velocidades máximas correspondientes.

En caso de que no sean iguales, los actualiza y va incluyendo en una variable local *bufferTX* el comando que enviará a la Cosy para restablecerlas.

Al terminar con las velocidades de los cuatro ejes, comprueba si hay comandos pendientes de enviar en *bufferTX* y en ese caso hace una anotación en el registro y envía a la máquina los comandos generados.

```
private void establecerVelocidadMáxima()
{
    // Establece la velocidad máxima de movimientos del cabezal si habían sido cambiadas.
    string bufferTX = "";
    if (velocidadXActual != datos.velocidadMaxX)
    {
        velocidadXActual = datos.velocidadMaxX;
        bufferTX += "SX" + dosDígitos(velocidadXActual);
    }
    if (velocidadYActual != datos.velocidadMaxY)
    {
        velocidadYActual = datos.velocidadMaxY;
        bufferTX += "SY" + dosDígitos(velocidadYActual);
    }
    if (velocidadZActual != datos.velocidadMaxZ)
    {
        velocidadZActual = datos.velocidadMaxZ;
        bufferTX += "SZ" + dosDígitos(velocidadZActual);
    }
    if (velocidadAActual != datos.velocidadMaxA)
    {
        velocidadAActual = datos.velocidadMaxA;
        bufferTX += "SA" + dosDígitos(velocidadAActual);
    }
    if (bufferTX != "")
    {
        registro += "Aumentando velocidad:\r\n";
        transmitirYConfirmar(bufferTX);
    }
}
```

Figura 161. Código fuente del método *establecerVelocidadMáxima*.

Los métodos *dosDígitos* y *cuatroDígitos* se usan para formatear adecuadamente las cantidades antes de ser enviadas a la Cosy. Ésta espera que ciertos datos, como las velocidades, le sean proporcionadas siempre con dos dígitos, independientemente del valor que sea. Lo mismo ocurre con datos como las posiciones, que siempre espera que sean cantidades de cuatro dígitos sin importar qué valor concreto sea. Para ello, en ambos casos, hay que rellenar a la izquierda con ceros.

En la figura 162 se puede ver el código del método dosDígitos y en la figura 163 el del método cuatroDígitos, pudiéndose ver que en ambos casos se convierte el número que se le pasa como parámetro a texto para luego averiguar su longitud.

```
private string dosDígitos(int n)
{
    string texto = Convert.ToString(n);
    switch (texto.Length)
    {
        case 2:
            break;
        case 1:
            texto = "0" + texto;
            break;
        default:
            texto = "00";
            break;
    }
    return texto;
}
```

Figura 162. Código fuente del método *dosDígitos*.

Con el dato de la longitud se añade, mediante una instrucción *switch*, la cantidad de ceros necesaria por la izquierda. Para cualquier caso erróneo que pudiera suceder, se devuelve siempre el valor 0, con la cantidad de dígitos apropiada.

```
private string cuatroDígitos(int n)
{
    string texto = Convert.ToString(n);
    switch (texto.Length)
    {
        case 4:
            break;
        case 3:
            texto = "0" + texto;
            break;
        case 2:
            texto = "00" + texto;
            break;
        case 1:
            texto = "000" + texto;
            break;
        default:
            texto = "0000";
            break;
    }
    return texto;
}
```

Figura 163. Código fuente del método *cuatroDígitos*.

Tras regresar del método *establecerVelocidadMáxima*, se comprueba si es necesario dispensar gotas de adhesivo para este componente. Se dispensará adhesivo sólo si se ha activado el *checkbox* "Dispensar Adhesivo", este componente tiene asignado un número de gotas mayor que cero y no se esté reintentando su colocación, pues en caso de ser un reintento, ya se habrá dispensado el adhesivo.

En la figura 164 se puede ver el código de este método. En primer lugar se registra la acción para pasar luego a calcular las coordenadas donde hay que situar la aguja del dispensador

y mover el brazo. Por último se dosifica la cantidad de gotas especificada en los datos del alimentador de este componente.

Continuando con el método *colocarComponente*, a continuación se selecciona el banco de alimentadores en que se encuentra el componente, para lo que llama al método *seleccionarBanco*, pasándole como parámetro el banco donde está el alimentador del componente actual. El código de este método se puede ver en la figura 165 y lo que hace inicialmente es determinar si hay que cambiar de banco o no, comparando el parámetro recibido con el campo de seguimiento *bancoActual*.

```
private void dispensarAdhesivo(componente componenteActual)
{
    // Posicionar aguja del dispensador
    registro += "Dispensando adhesivo:\r\n";
    Point coordenadasDispensar = calculaCoordenadasPlace(componenteActual);
    coordenadasDispensar.X += datos.offsetDispensadorX;
    coordenadasDispensar.Y += datos.offsetDispensadorY;
    string bufferTX = "PX" + cuatroDígitos(datos.xMax - coordenadasDispensar.X) + "PY" +
                cuatroDígitos(coordenadasDispensar.Y);
    transmitirYConfirmar(bufferTX);

    // Dispensar gotas
    transmitirYConfirmar("LI" +
dosDígitos(datos.alimentadores[componenteActual.alimentador].gotasDispensador));
}
```

Figura 164. Código fuente del método dispensarAdhesivo.

Si debe cambiar, transmite el comando apropiado, espera la finalización del cambio y actualiza el campo de seguimiento *bancoActual*. Si no es necesario cambiar de banco, simplemente retorna.

```
private void seleccionarBanco(int nuevoBanco)
{
    // Cambia el banco de alimentadores si no está ya puesto.
    if (nuevoBanco != bancoActual)
    {
        registro += "Cambiando a banco de alimentadores " + nuevoBanco + "\r\n";
        switch (nuevoBanco)
        {
            case 1:
                transmitir("FN001");
                break;
            case 2:
                transmitir("FN031");
                break;
            case 3:
                transmitir("FN061");
                break;
        }
        esperarOK(10);
        bancoActual = nuevoBanco;
    }
}
```

Figura 165. Código fuente del método *seleccionarBanco*.

Tras retornar del método *seleccionarBanco*, el siguiente paso en el método *colocarComponente* es el cambio de útil en el cabezal en caso de que sea necesario, llamando para ello al método *montarÚtil*, cuyo código se puede ver en la figura 166.

Lo primero que hace éste método es comprobar si es necesario el cambio de útil, para lo que compara el útil pedido con el campo de seguimiento *útilActual*. Si son iguales, no es necesario hacer ninguna operación, así que simplemente retorna, pero si no son iguales, inicia el proceso de cambio del útil.

Como se puede ver, en caso de que el cabezal tuviese montado otro útil, este método llama primero a *desmontarÚtilActual*, que se encarga de desmontarlo y dejar la aguja desnuda (código de útil 0). A continuación se comprueba si el número del útil pedido está entre los valores posibles, que son del 1 al 4, para empezar a enviar comandos a la máquina.

La secuencia de comandos enviados se corresponde con la observada en la etapa de análisis de la máquina, aunque, según el útil pedido, se calcula la coordenada x a la que hay que mover el brazo (la coordenada y es la misma para todos). En el primer envío se ordena a la máquina inicializar la altura y ángulo de la aguja y abrir la puerta del panel de útiles. El segundo posiciona el brazo en la posición de entrada al panel de útiles. Por último, se envía un tercer grupo de comandos compuesto por el posicionamiento de la aguja en las coordenadas del útil concreto que se ha pedido, bajar el cabezal hasta la altura de captura del útil, inicialización de la altura y ángulo de la aguja (subir la aguja) y cierre de la puerta del panel.

Para finalizar, se actualiza el campo de seguimiento *útilActual* para que refleje el nuevo útil colocado en el cabezal y se retorna.

```
private void montarÚtil(int útilPedido)
{
    // Coloca el útil pedido en el cabezal, desmontando el que estuviese previamente
    if (útilActual != útilPedido)
    {
        if (útilActual != 0) desmontarÚtilActual();
        if (útilPedido >= 1 && útilPedido <= 4)
        {
            registro += "Montando útil " + útilPedido + ":\r\n";
            transmitirYConfirmar("HZTEHA");
            transmitirYConfirmar("PX3300PY0200");
            transmitirYConfirmar("PX3" + Convert.ToString(33 + útilPedido * 12) +
                        "0PY0316PZ0170HZHATA");
            útilActual = útilPedido;
        }
    }
}
```

Figura 166. Código fuente del método *montarÚtil*.

El método *desmontarÚtilActual*, cuyo código se puede ver en la figura 167, comprueba si está montado alguno de los útiles de la máquina (códigos del 1 al 4) para luego enviar los comandos de desmontaje apropiados en caso de que así sea. Como se puede ver, los comandos enviados son los mismos que para el montaje con un pequeño cambio en el orden en que son enviados.

```
private void desmontarÚtilActual()
{
    if (útilActual >= 1 && útilActual <= 4)
    {
        registro += "Desmontando útil " + útilActual + "\r\n";
        transmitirYConfirmar("TEHZHA");
        transmitirYConfirmar("PX3300PY0200");
        transmitirYConfirmar("PX3" + Convert.ToString(33 + útilActual * 12) +
                "0PY0316PZ0170TAHZHA");
        útilActual = 0;
    }
}
```

Figura 167. Código fuente del método *desmontarÚtilActual*.

Tras retornar del método *montarÚtil*, en el método *colocarComponente* se prepara y transmite el comando para colocar el cabezal sobre el alimentador del componente.

A continuación se prepara un comando compuesto en *bufferTX* que establece la velocidad vertical del componente y se coge éste, realizando un test de vacío para comprobar si ha fallado la captura.

Para coger el componente se programa la velocidad del eje z para este componente, almacenada en la estructura de datos del alimentador en *velocidadZ* y se baja hasta la altura de captura, almacenada en *zPick*, se activa el vacío y se eleva de nuevo hasta la altura de transporte de este componente, almacenada en *zMover*. En caso de que el alimentador de este componente requiera un impulso para avanzar al siguiente, se añadirá el comando que baja y vuelve a subir el imán que lo proporciona. Por último, se añade a *bufferTX* el comando que realiza el test de vacío y se envía.

Hay que hacer notar que en este punto se puede interrumpir la ejecución de este método debido al test de vacío, pero no se abortará la producción. En caso de que falle la captura del componente se disparará la excepción *PérdidaException* y el programa saltará al manejador de este error.

El manejo del error de pérdida del componente, como vimos anteriormente, se realiza en el método *producirPlaca* y consiste en comprobar cuántas veces ha fallado este componente para reintentar su captura nuevamente o bien pasar al siguiente.

Si la operación de captura resultó exitosa, se pone el componente en estado *Colocando* y se provoca la actualización del interfaz. A continuación se programan las velocidades de traslación y rotación específicas de este componente. Estas son importantes pues, dependiendo de la masa y la forma del componente, se podrían producir inercias durante su manejo que lo desplacen de su posición en el cabezal o que lo suelten. Para hacerlo se compone el comando en función de qué velocidades deben actualizarse, según los campos de seguimiento de las velocidades y los datos almacenados en el alimentador actual, de forma parecida a como actúa el método *establecerVelocidadMáxima*.

A continuación el componente es centrado en la estación de centrado, en caso de ser necesario. Para ello se llama al método centrar*EnEstaciónDeCentrado* si el alimentador actual así lo especifica en el campo *centradoEstación*.

El código de este método se puede ver en la figura 168, donde se ha condensado el algoritmo de centrado descrito anteriormente. El primer grupo de comandos transmitidos llevan el componente hasta el punto de descarga de la estación de centrado, lo baja y desactiva el vacío para soltarlo. El segundo grupo de comandos sube el cabezal y ejecuta la rutina de centrado preprogramada en la máquina dos veces. En la tercera transmisión se baja de nuevo el cabezal y se activa nuevamente el vacío para volver a capturar el componente.

El siguiente comando únicamente sube el componente hasta una altura de transporte segura para sacarlo de la estación de centrado para, en el último comando transmitido situar el cabezal fuera de la estación de centrado, de nuevo sobre el panel de trabajo.

```
private void centrarEnEstaciónDeCentrado()
{
    registro += "Estación de centrado:\r\n";
    transmitirYConfirmar("PX3870PY1250PZ0140VA");
    transmitirYConfirmar("PZ0045ZSZS");
    transmitirYConfirmar("PZ0150VE");
    transmitirYConfirmar("PZ0045");
    transmitirYConfirmar("PX3300PY1250");
}
```

Figura 168. Código fuente del método *centrarEnEstaciónDeCentrado*.

De vuelta en el método *colocarComponente* se realiza el cálculo de la posición de colocación del mismo. Para el cálculo de las coordenadas hasta las que hay que trasladar el componente se tiene en cuenta la posición y ángulo de giro de la placa sobre el panel de trabajo. La deducción de las fórmulas que permiten hallarlas se explica más adelante, en el epígrafe de este capítulo dedicado al cálculo de las coordenadas de un componente.

Una vez situado el componente en su posición de montaje sobre la placa se efectúa el centrado con las pinzas del cabezal en caso de ser necesario. Se puede realizar el centrado sólo en el eje x, sólo en el eje y o en ambos ejes, por lo que se va componiendo el grupo de comandos a enviar en la variable *bufferTX* como en ocasiones anteriores, enviándolo al finalizar si contiene algún comando.

Como el cabezal sólo dispone de pinzas en el eje x, la máquina realiza el centrado en el eje y girándolo automáticamente 90° y devolviéndolo posteriormente a su posición original.

A continuación se gira el componente para que quede colocado en el ángulo correcto sobre la placa. El ángulo que se debe girar el componente será la suma del ángulo de giro de la placa y el ángulo de giro de este componente, como se verá en el apartado dedicado a la deducción de las fórmulas más adelante. Sólo se envía el comando de giro en caso de que el ángulo de giro sea distinto de 0°, obviamente. Además, antes de colocar el componente sobre la placa se realiza un test de vacío para comprobar si sigue en su sitio tras las operaciones a que ha sido sometido.

En caso de que todo sea correcto, el siguiente grupo de comandos baja el componente hasta su altura en la placa, obtenida desde *zPlace* en la estructura de datos del alimentador actual y desactiva el vacío del cabezal. Tras esto se inicializa la altura y ángulo de giro del cabezal, quedando éste listo para el siguiente componente.

Para finalizar, se actualiza el estado del componente a "*Colocado*" y se provoca de nuevo la actualización del interfaz de usuario, retornando del método a continuación.

5.17.9.3. Cálculo de la posición de colocación del componente.

Para calcular la posición de colocación de cada componente es necesario tener en cuenta varios factores, puesto que esta información está repartida entre diferentes datos grabados en las estructuras que maneja el programa.

Hay que tener en cuenta que la posición de colocación almacenada en la estructura del componente es relativa al punto tomado como origen en la placa en la que va colocado, pero esta placa puede estar a su vez colocada en cualquier lugar del panel de la Cosy. Algo que nunca sucederá es que la posición del origen de la placa esté en el origen del panel (al menos nunca sucederá si éste origen es un punto real de la placa). Esto se debe a que el origen de coordenadas del panel está en la zona de útiles, a la izquierda, fuera de la zona que permite la fijación de placas para su montaje.

Además, como se ha dotado a la estructura de datos de la placa de un parámetro de giro, la placa no tiene que estar situada paralela al eje x, pudiendo estar rotada cualquier ángulo. Esto se ha hecho así para dar una mayor flexibilidad y, por ejemplo, permitir aprovechar huecos entre placas no rectangulares para situar otras placas que encajen y poblarlas todas en un solo programa de montaje.

Este mayor grado de flexibilidad trae aparejada una mayor complejidad en el desarrollo de nuestra aplicación. Si no se permitiese un ángulo de giro, bastaría con sumar las coordenadas del origen de cada placa a las coordenadas de cada uno de sus componentes para obtener las coordenadas del panel donde se debe situar la aguja para colocar el componente.

Sin embargo, la adición del ángulo de giro de la placa lleva aparejada no solo una rotación adicional del componente, sino un desplazamiento adicional en los ejes x e y que hay que calcular. Todo esto se puede apreciar mejor en la figura 169, en la que nos basaremos para deducir las fórmulas de cálculo.

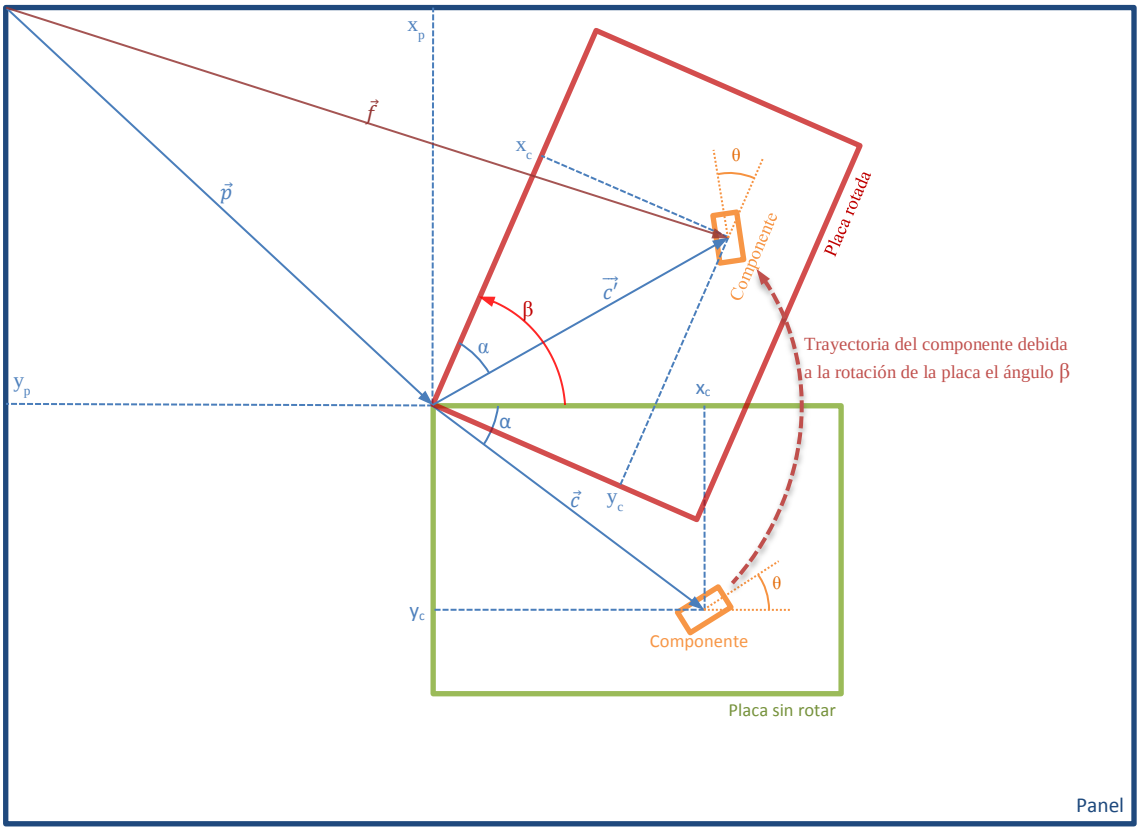


Figura 169. Coordenadas y ángulo final de un componente al rotar la placa.

En la figura se ha representado como un rectángulo azul el área del panel, el rectángulo verde representa la placa en sus coordenadas, pero sin ser rotada, el rectángulo rojo representa la posición de la placa una vez rotada y en naranja se ha representado un componente,

mostrándose tanto en la placa antes de ser rotada como en la placa rotada para apreciar el desplazamiento que sufre debido a la rotación.

El origen de coordenadas del panel se encuentra en la esquina superior izquierda del rectángulo azul, según nuestra propia convención, comentada anteriormente y que se muestra en la figura 84. Entonces, las coordenadas del punto de origen de la placa (su esquina superior izquierda, sin tener en cuenta el ángulo de rotación) vienen dadas por el vector $\vec{p}$, cuyas componentes cartesianas son (x_p,y_p). Dentro de la placa, la posición del componente viene dada por el vector $\vec{c}$, cuyas componentes son (x_c,y_c), que al rotar la placa se convierte en el vector $\overrightarrow{c'}$.

La posición final del componente vendrá dada por la suma de los vectores $\vec{p}$ y $\overrightarrow{c'}$, que en la ilustración se ha llamado $\vec{f}$. Conocemos las componentes del vector $\vec{p}$, pero en principio el vector $\overrightarrow{c'}$ es desconocido y tendremos que calcularlo.

Para calcular las componentes de este vector podemos usar las reglas trigonométricas que conocemos, para lo que necesitamos saber su módulo y ángulo que forma con el eje x. A partir de la imagen se puede deducir que el módulo del vector $\overrightarrow{c'}$ es el mismo que el de $\vec{c}$ y se puede obtener usando el teorema de Pitágoras, utilizando las coordenadas del componente:

$$\left|\overrightarrow{c'}\right| = |\vec{c}| = \sqrt{{x_c}^2 + {y_c}^2}$$

También a partir de la imagen, se puede deducir que el ángulo que forma el vector $\overrightarrow{c'}$ con el eje x será la resta del ángulo β, menos el ángulo α.

El ángulo β es el ángulo que ha rotado la placa, por lo que es un dato conocido (se encuentra almacenado en la estructura de datos de la placa, como vimos anteriormente). En la figura 169 se ha dibujado este ángulo representando su arco con una punta de flecha para identificar claramente cuál ha sido la rotación sufrida por la placa. El ángulo α, aunque en principio es desconocido, se puede calcular muy fácilmente pues es la arcotangente del cociente de las coordenadas cartesianas del componente, que son datos conocidos:

$$\alpha = \tan^{-1}\frac{y_c}{x_c}$$

Como resultado de los cálculos anteriores, tenemos que las componentes (x_f,y_f) del vector de posición final del componente sobre el panel de trabajo de la Cosy, $\vec{f}$, serán:

$$x_f = x_p + \left|\overrightarrow{c'}\right| \cos(\beta - \alpha)$$

$$y_f = y_p - \left|\vec{c'}\right| \sin(\beta - \alpha)$$

Donde se ha tenido en cuenta el hecho de que el eje y de coordenadas en nuestro caso está invertido respecto a su sentido normal (crece hacia abajo, cuando habitualmente se toma como creciente hacia arriba).

Sólo nos falta conocer el ángulo que hay que girar el componente antes de depositarlo. A partir de la figura 169 se deduce que el ángulo total que hay que rotar el componente respecto al eje x es:

$$\theta_f = \beta + \theta$$

El ángulo θ es el ángulo de rotación del componente en su placa, por lo que también es un dato conocido (se encuentra almacenado en la estructura de datos del componente). Por tanto ya tenemos todos los cálculos necesarios para posicionar un componente, sabidas las coordenadas del origen y ángulo de giro de su placa y sabidas también las coordenadas y ángulo de giro del componente respecto al origen de su placa.

Con las fórmulas obtenidas, se creó el código del método *calcularCoordenadasPlace*, cuyo código se puede ver en la figura 170. Los comentarios indican el término que se está calculando en cada caso.

```
private Point calculaCoordenadasPlace(componente componenteActual)
{
    // Para calcular la posición de colocación se tiene en cuenta el giro de la placa.
    double ánguloDelVectorComponenteEnPlacaSinRotarEnRadianes = Math.Atan2(componenteActual.y,
                componenteActual.x);                                    // alfa
    double ánguloDeLaPlacaEnRadianes = placaActual.t * Math.PI / 200;   // beta
    double ánguloDelVectorComponenteEnPlacaRotadaEnRadianes = ánguloDeLaPlacaEnRadianes –
                ánguloDelVectorComponenteEnPlacaSinRotarEnRadianes;     // beta - alfa
    double móduloVectorComponenteEnPlaca = Math.Sqrt(Math.Pow(componenteActual.x, 2) +
                Math.Pow(componenteActual.y, 2));                       // Módulo del vector c
    int coordenadaXFinal = placaActual.x + (int)(móduloVectorComponenteEnPlaca *
                Math.Cos(ánguloDelVectorComponenteEnPlacaRotadaEnRadianes));    // xf
    int coordenadaYFinal = placaActual.y - (int)(móduloVectorComponenteEnPlaca *
                Math.Sin(ánguloDelVectorComponenteEnPlacaRotadaEnRadianes));    // yf
    return new Point(coordenadaXFinal,coordenadaYFinal);
}
```

Figura 170. Código fuente del método *calcularCoordenadasPlace*.

5.17.10. Actualización de la interfaz de usuario durante la producción

Cuando se llama al método *ReportProgress* del *BackgroundWorker* se genera un evento *ProgressChanged* en el hilo principal de la aplicación, cuyo método asociado se puede ver en la figura 171.

Como se observa, este método lo único que hace es actualizar la interfaz de usuario para mostrar el estado en que se encuentra la producción. Lo primero que hace es actualizar la barra de progreso a partir del parámetro *e.ProgressPercentage*, que es un dato que siempre se envía desde el segundo hilo. A continuación se actualiza la barra de título del formulario de producción con el porcentaje procesado. Seguidamente se actualiza el *label* que contiene el número de componentes pendientes de ser procesados.

Para terminar la actualización del formulario se vuelca el contenido del campo donde llevamos el registro de operaciones en el *textbox* que lo muestra y se lleva la vista actual del *textbox* hasta el final del texto. De esta forma el texto aparenta ir haciendo scroll hacia arriba cada vez que se añaden líneas nuevas.

Tras terminar con la actualización del formulario se comprueba si se ha pasado un segundo dato, mediante el parámetro *e.UserState*. Si éste no es *null* quiere decir que hemos enviado el número de placa en que estamos trabajando y queremos que se represente visualmente, por lo que se llama al método *mostrarPlaca*, pasándole como parámetro el número de placa a representar. Éste se encargará de representar la placa visualmente, actualizando así los colores de los componentes en la vista de la placa correspondiente.

```
private void backgroundWorkerProducir_ProgressChanged(object sender,
                ProgressChangedEventArgs e)
{
    progressBarProducción.Value = Math.Min(e.ProgressPercentage,
                progressBarProducción.Maximum);
    this.Text = "En producción - " + e.ProgressPercentage.ToString() + "% completado";
    labelPendientes.Text = (totalPendientes - componentesTratados).ToString();
    textBoxRegistro.Text = registro;
    textBoxRegistro.SelectionStart = textBoxRegistro.TextLength;
    textBoxRegistro.ScrollToCaret();
    if (e.UserState != null) datos.mostrarPlaca((int)e.UserState);
}
```

Figura 171. Código fuente del método *ProgressChanged*.

5.17.11. Tratamiento de los errores de producción

En una primera versión de los métodos implicados en la producción, éstos devolvían siempre un valor lógico (*bool*) indicando si había tenido éxito o no. Esto hacía que en todas las llamadas a cualquiera de ellos hubiese que comprobar el valor devuelto para, en caso de error, tratarlo o retornar a su vez indicando error en la operación. De esta forma el error que no fuese tratado *in situ* se iba propagando hacia atrás en la pila de llamadas hasta alcanzar el método que podía tratar el error.

Como hemos visto, la colocación del componente es una operación bastante compleja y cargada de pasos opcionales, que hay que incluir en el proceso o no en función del estado de diversas condiciones. Esto hacía que el código fuente estuviese plagado de comprobaciones de error que entorpecía su lectura y comprensión para un seguimiento ágil de las operaciones realizadas.

Para simplificar la detección y el tratamiento de errores en cualquiera de las etapas del proceso de producción se ha decidido utilizar el mecanismo de excepciones que posee el lenguaje C#. Se han creado excepciones para tratar las condiciones de error que devuelve la máquina, añadidas a algunas ya preexistentes en el sistema como son las excepciones por *timeout* del puerto serie.

Con esto se ha logrado centralizar en gran medida el tratamiento de errores, que es una solución más elegante [**35**]. Además se ha aligerado el código fuente al dejar de comprobar errores tras cada llamada, haciéndolo mucho más fácil de leer y entender.

Las nuevas excepciones creadas para indicar las condiciones de error devueltas por la máquina se muestran en el listado de la figura 172. Se han creado dos excepciones para tratar los principales errores que nos interesan: la pérdida de un componente por el cabezal y un posible bloqueo de la máquina por la detección de una condición que ponga a ésta en una situación de peligro de integridad.

Tal vez el caso más representativo de este peligro de integridad sería intentar mover el brazo de la máquina con el cabezal demasiado bajo. En ese caso se correría el riesgo de que la aguja tropiece con los componentes colocados en la placa, descolocándolos, o peor aún, con alguna de las estructuras fijas de la máquina, dañándose la aguja y posiblemente la zona de impacto en la máquina.

```
public class PérdidaException : Exception
{
    public PérdidaException() : base("El test de pérdida ha fallado.") { }
    public PérdidaException(string mensaje) : base(mensaje) { }
    public PérdidaException(string mensaje, Exception anidada) : base(mensaje, anidada) { }
}

public class PeligroException : Exception
{
    public PeligroException() : base("Posible peligro de integridad de la Cosy.") { }
    public PeligroException(string mensaje) : base(mensaje) { }
    public PeligroException(string mensaje, Exception anidada) : base(mensaje, anidada) { }
}
```

Figura 172. Excepciones definidas para reflejar los errores de la máquina.

Como se puede ver en la figura 172, las nuevas excepciones creadas son clases que derivan de la clase *Exception* e implementan los tres constructores mínimos recomendados por Microsoft [**43**], heredando de la clase base. En la definición del constructor sin parámetros se ha incluido un breve texto explicativo de la causa de la excepción, que es el que irá incluido en la instancia de la excepción cuando ésta sea lanzada, si no se especifica otro al crear la instancia.

La principal fuente de excepciones será el método *esperarConfirmación* que, como se puede ver en la figura 173 es quien provoca explícitamente las nuevas excepciones creadas según el código devuelto por la máquina tras cada comando enviado. Éste método es llamado tras la transmisión de cada comando a la máquina del que haya que esperar una respuesta por parte de ésta.

La misión del método consiste en esperar a que la máquina devuelva el código indicador de que todo ha ido bien. Una vez recibido este código simplemente retorna, pero si se recibe un código de error, genera la excepción adecuada para tratarlo.

Si el mensaje esperado no llega, se activará el timeout del puerto serie, lanzando la excepción *TimeoutException* incluida en el *framework*. Para la correcta operación del *timeout*, antes de entrar en el bucle de espera se programa el tiempo máximo de espera para la recepción.

El método *esperarConfirmación* está sobrecargado, pues se han creado dos métodos con el mismo nombre, pero que reciben diferentes parámetros. Uno de ellos recibe como parámetro cuántos segundos de tiempo de espera debe permitir. El otro no recibe parámetros, sino que calcula el tiempo de espera automáticamente a partir de la velocidad actual de movimiento del cabezal, haciendo una estimación simplificada del tiempo máximo de recorrido del brazo por el panel. Durante las pruebas se comprobó que el tiempo que tardaba el cabezal en recorrer la máxima distancia posible a la menor velocidad no superaba los 22 segundos, por lo que este es el tiempo máximo que se esperará en cualquier caso.

La mayor parte de las veces basta con dejar que el método elija automáticamente el tiempo de espera, para evitar tener en cuenta constantemente cuanto tiempo se estima que tardará la máquina en realizar la operación. Sin embargo, hay casos como el cambio de banco de alimentadores, en que se trata de una operación lenta y su velocidad no depende de la del brazo, por lo que se optó por enviar al método el tiempo que se quiere esperar explícitamente. Para el cambio de alimentadores, los tiempos invertidos fueron de 7 segundos entre bancos contiguos y 13 segundos de un extremo a otro.

```
public void esperarConfirmación()
{
    // Calcula el tiempo de espera de OK en función de la velocidad del cabezal.
    int tiempoEspera = 22 - Math.Min(velocidadXActual, velocidadYActual);
    esperarConfirmación(tiempoEspera);
}

public void esperarConfirmación(int tiempoEspera)
{
    datos.puertoSerie.ReadTimeout = 1000 * tiempoEspera;
    while (true)
    {
        string bufferRX = "";
        bufferRX = datos.puertoSerie.ReadLine();
        registro += "\t\t" + DateTime.Now.ToString() + " - Recibido: " + bufferRX + "\r\n";
        if (bufferRX.Contains("FF00")) return;
        else if (bufferRX.Contains("FF02")) throw new PérdidaException();
        else if (bufferRX.Contains("FF23")) throw new PeligroException();
    }
}
```

Figura 173. Código fuente de *esperarConfirmación*.

La excepción *TimeoutException* también puede ser lanzada automáticamente desde el método *transmitir*, en caso de que la transmisión de un comando falle al ejecutar el método *WriteLine*, como se puede ver en la figura 156. En este caso el tiempo de espera está programado desde que arrancó el programa, durante la fase de inicialización. En ese momento se abre el puerto serie, si está disponible, y se configuran todos sus parámetros. Se ha estimado que un tiempo de espera de 5 segundos es más que suficiente para cualquier operación del puerto, por lo que este es el valor que ha sido predeterminado.

En realidad el tiempo necesario para la transmisión nunca será tan grande, puesto que en diversas pruebas empíricas de la capacidad del buffer de recepción de la Cosy se determinó que era capaz de albergar aproximadamente unos 200 bytes de comandos antes de empezar a sufrir pérdidas. Con esta cantidad de datos y una velocidad de 4800 bps tenemos un tiempo total de transmisión ligeramente menor de 0,5 segundos, por lo que se ha estimado adecuado un *timeout* 10 veces superior.

Las excepciones generadas provocan la interrupción inmediata del método actual. Si se generan durante la ejecución de una instrucción que se encuentra dentro de un bloque *try*, el control pasará inmediatamente al bloque *catch* asociado que la capture. Si no hay un bloque catch capaz de capturar la excepción provocada, ésta subirá al siguiente método en la pila de llamadas, que será el método que llamó a aquel en que se produjo. Si la llamada al método inferior se produjo dentro de un bloque try se vuelve a repetir el proceso, pasando la ejecución del programa al bloque catch que capture la excepción producida.

Si éste tampoco captura la excepción, se repite el proceso, recorriendo así la pila de llamadas a métodos en orden inverso hasta encontrar uno que la capture. Si se llega al método inicial de la aplicación sin ser capturada, se interrumpe la ejecución de la misma.

Aprovechando la propagación de las excepciones hacia atrás en la pila de llamadas, se ha situado el código de control de errores de producción en los métodos de nivel superior, donde se capturará cualquier error manejable producido en uno de los métodos de nivel inferior.

La captura de las excepciones se realiza en distintos lugares según si se considera si se trata de un error recuperable o irrecuperable. Consideraremos un error recuperable la pérdida de un componente, puesto que el test de pérdida nos informará en los puntos críticos del algoritmo de colocación mediante la excepción correspondiente.

Como vimos anteriormente, la pérdida de un componente se trata desechando el componente y reintentando el mismo componente o pasando al siguiente, según el número de reintentos que se lleven realizados. Por tanto, el mejor sitio para colocar la captura de la excepción de pérdida es el método *producirPlaca*. Éste método es quien llama a *colocarComponente*, que es quien realiza los tests de pérdida tras coger el componente y antes de colocarlo, como se puede ver en la figura 160.

De esta forma, si el bucle que recorre todos los componentes de la placa llamando a *colocarComponente* para cada uno de ellos lo situamos dentro de un bloque *try*, como se ve en código de la figura 158, tendremos oportunidad de capturar las excepciones producidas en niveles inferiores en su bloque *catch* aparejado.

Una vez capturada la excepción y tras desechar el componente fallido, para tratar este error basta no incrementar el contador de componentes del bucle (c), así éste repetirá el mismo componente. Además vamos contando los reintentos que hacemos en la variable local *reintentosComponenteActual*, para pasar al componente siguiente si se supera el número máximo de reintentos por componente. En caso de que se supere, ignoraremos este componente para pasar al siguiente, para lo que incrementamos el contador de componentes del bucle y restauramos el contador de reintentos, para que en el siguiente componente empiece de nuevo desde cero.

Además, como se puede ver en el código, se incrementa el contador de componentes que han fallado (el campo *componentesFallados* de la clase) y también se actualiza el estado del componentes en la estructura de datos de la placa. Si el componente estaba en estado *Cogiendo* se pasa al estado *Error_Cogiendo* y si estaba en estado *Colocando*, se pasa al estado

Error_Colocando. Esto afectará al color con que se representa el componente en la vista de la placa. Se han escogido los colores rojo y naranja para que resalten claramente los componentes que han dado error, facilitándole al operador la localización para su reparación.

En la versión actual del programa se consideran errores irrecuperables el resto de errores que se pudieran producir durante la producción. Los errores tenidos en cuenta para hacer un seguimiento individualizado son los errores de *timeout* generados en el puerto serie y los errores de peligro generados por la máquina, comentados anteriormente.

Como se mencionó anteriormente cuando se comentó el control *BackgroundWorker*, cualquier excepción no capturada por el código que se ejecuta en el segundo hilo provocará su finalización, siendo capturada por este control. Al pasar el primer hilo a ejecutar el método *RunWorkerCompleted*, se le entregará una notificación de que se finalizó el hilo debido a una excepción a través del parámetro *e.Error*.

Si no se terminó la ejecución del segundo hilo debido a una excepción, este parámetro valdrá *null*. En caso de que la terminación se produjera debido a una excepción, *e.Error* contendrá la excepción que se produjo.

5.17.12. Terminación del proceso de producción

Cuando termina el proceso de producción, el control *BackgroundWorker* genera el evento *RunWorkerCompleted* en el hilo principal, por lo que éste pasa a ejecutar el método que atiende este evento, cuyo código podemos ver en la figura 174.

Como se puede ver en el código, en éste método se hacen fundamentalmente 2 tareas. La primera de ellas es restaurar la interfaz de usuario al estado inicial, desbloqueando los controles bloqueados al iniciar la producción y recuperando el aspecto tanto del botón *Iniciar* como de la barra de título del formulario.

La segunda tarea es analizar la causa de la finalización para informar al usuario. La finalización del proceso de producción se puede deber a 3 causas:

1. Se produjo un error irrecuperable durante la producción.
2. El usuario canceló la operación de producción.
3. Se completó el montaje de todos los componentes pendientes.

Como se ve en el código, en primer lugar se comprueba si el parámetro *e.Error* contiene un valor distinto de *null*. Si es así, su contenido será la excepción que ha interrumpido la ejecución del hilo, por lo que se envía al registro y se llama al método que finalmente lo graba en disco y da

por finalizado el mismo, actualizando finalmente su *textBox*. Además se muestra cuadro de diálogo que avisa al usuario de que ha habido un error que requiere su atención sobre la máquina.

```
private void backgroundWorkerProducir_RunWorkerCompleted(object sender,
                RunWorkerCompletedEventArgs e)
{
    // Restaura y desbloquea los controles
    buttonIniciar.Text = "Iniciar";
    buttonIniciar.BackColor = Color.DodgerBlue;
    buttonIniciar.Enabled = true;
    this.Text = "Producción";

    checkBoxReiniciar.Enabled = true;
    checkedListBoxPlacas.Enabled = true;
    actualizaListaPlacas();

    if (e.Error != null)
    {
        // Se produjo una excepción no gestionada o relanzada en el DoWork
        registro += "-------- ERROR IRRECUPERABLE. PRODUCCIÓN INTERRUMPIDA --------\r\n" +
                e.Error.Message + "\r\n";
        finalizarYGrabarRegistro();
        MessageBox.Show("Error: " + e.Error.Message +
                "\r\n\r\nEs necesario revisar la máquina, puede haber sufrido una parada
                de seguridad.\r\nEn ese caso, reiníciela.",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else if (e.Cancelled)
    {
        // Se canceló la producción
        registro += "------- Producción cancelada antes de ser completada -------\r\n";
        finalizarYGrabarRegistro();
        MessageBox.Show("Se cancelo la producción antes de ser completada", "Producción
                cancelada", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    else
    {
        // La producción terminó normalmente
        if (componentesFallados == 0)
        {
            registro += "------- Producción completada sin errores -------\r\n";
            finalizarYGrabarRegistro();
            MessageBox.Show("Se ha completado la producción. Se han montado todos los
                    componentes.", "Producción completada exitosamente",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            registro += "------- Producción completada. " + componentesFallados + "
                    componentes no se han podido montar -------\r\n";
            finalizarYGrabarRegistro();
            MessageBox.Show("Producción terminada. "+ componentesFallados + " componentes
                    no se han podido montar.", "Producción terminada",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
}
```

Figura 174. Código fuente del método *RunWorkerCompleted*.

Como vimos anteriormente desde nuestra aplicación se generan dos excepciones que pueden dar lugar a este final. Una de ellas es un error de timeout en el puerto serie, que normalmente se produciría en caso de que se interrumpiera la comunicación a través del puerto.

Si este fuese el caso es necesaria la atención del operador para que restablezca la conexión y determine, según el momento del montaje en que se haya producido, la posibilidad de iniciar la reanudación del proceso.

La otra excepción se genera en caso de que se reciba de la máquina un código de error grave, relacionado en gran medida con un posible peligro de integridad. La máquina se bloquea automáticamente, encendiendo la luz de alarma, cuando ha detectado una condición de posible peligro de integridad. En esta situación ya no responderá a comandos enviados a través del puerto serie, siendo necesario reiniciarla o apagarla desde los botones de su panel en el lateral izquierdo. Por tanto, nada más se puede hacer desde el punto de vista de nuestra aplicación, sino avisar al usuario para que lo haga, analice la causa del peligro y determine las acciones a llevar a cabo para poder continuar la producción.

Si la finalización se debió a que el usuario canceló la operación, se detectará porque el parámetro *e.Cancelled* estará activado. En ese caso, se realizan las mismas acciones que en caso de error, cambiando los mensajes y el icono del cuadro de diálogo mostrado.

En caso de que la finalización se debiera a que se completó el montaje de los componentes, se comprueba si hubo componentes que no se colocaron mediante el campo *componentesFallados*, registrando y avisando al usuario del número de componentes que no se pudieron montar.

5.17.13. Grabación de archivos de registro de la producción

En cualquier caso, en el método *runWorkerCompleted* siempre se llama al método *GrabaRegistroYFinalizarlo* antes de mostrar el cuadro de diálogo al usuario que le informa del estado en que terminó la producción. El código de este método se puede ver en la figura 175. Este método se encarga de guardar el registro en un archivo de disco y añadirle un mensaje final en la pantalla, indicando si pudo guardarlo o no.

Por defecto, los archivos de registro se graban en la carpeta *COSY_Log* dentro de la misma carpeta donde está el fichero con el que se está trabajando, pero el usuario puede cambiarlo modificándolo en las propiedades de configuración de la aplicación. Si la carpeta no existe, el método la crea.

El nombre del archivo de registro siempre comienza con el nombre del fichero de trabajo, al que se le añade la terminación "*Producción_*" más un número que se va aumentando automáticamente y la extensión *.log*.

```
private void GrabarRegistroYFinalizarlo()
{
    try
    {
        string directorioDeTrabajo = Path.GetDirectoryName(datos.rutaFichero);
        string nombreFichero = Path.GetFileNameWithoutExtension(datos.rutaFichero);
        string directorioLog = Path.Combine(directorioDeTrabajo, datos.directorioDeLogs);
        string rutaficheroLog;
        string nombreFicheroLog;

        // Crea el directorio si no existía.
        if (!Directory.Exists(directorioLog)) Directory.CreateDirectory(directorioLog);

        //Busca un nombre libre
        int n = 1;
        do
        {
            nombreFicheroLog = nombreFichero + " Producción_" + n + ".log";
            rutaficheroLog = Path.Combine(directorioLog, nombreFicheroLog);
            n++;
        }
        while (File.Exists(rutaficheroLog));

        // Graba el archivo de registro
        StreamWriter fichero = new StreamWriter(rutaficheroLog, false);
        fichero.Write(registro);
        fichero.Close();
        registro += "\r\nRegistro guardado.\r\n";
    }
    catch
    {
        registro += "\r\nNo se pudo guardar el registro.\r\n";
    }
    finally
    {
        textBoxRegistro.Text = registro;
        textBoxRegistro.SelectionStart = textBoxRegistro.TextLength;
        textBoxRegistro.ScrollToCaret();
    }
}
```

Figura 175. Código fuente del método *GuardarRegistroYFinalizarlo*.

La cantidad de archivos de registro de producción no se ha limitado artificialmente como se hizo con la cantidad de copias de seguridad. Esto se ha hecho así para dejar constancia en el disco de todas las veces que se ha realizado (o intentado) el montaje y qué ha sucedido en cada caso.

El límite natural de archivos de registro para cada trabajo que se haga con la máquina es el rango máximo de un entero en el lenguaje utilizado, o sea 2.147.483.647 archivos de registro. Si el usuario llegara a superar dicha cantidad de montajes de un mismo trabajo (cosa extremadamente improbable), no se seguirían grabando los archivos de registro de ese trabajo a menos que borre alguno de los registros anteriores.

5.17.14. Montaje automático: VIOS

Para realizar trabajos con las máquinas automáticas más modernas de las que dispone el laboratorio, normalmente se genera la información de montaje en un archivo de tipo VIOS Text a partir de los ficheros generados por Altium. Este archivo será posteriormente cargado en la

máquina con la que se va a trabajar y se completa el preprocesado directamente en la máquina, especificando la información de alimentadores y otros datos necesarios.

Tras seleccionar esta opción, se le presenta al usuario el formulario de la figura 176. Como se puede ver, es un formulario bastante simple en el que el usuario sólo debe elegir la placa de la que generar el archivo VIOS y pulsar el botón "*Generar*". Al pulsarlo, se le muestra un cuadro de diálogo *guardar archivo* estándar de Windows para que elija la ubicación y nombre del archivo VIOS de salida.

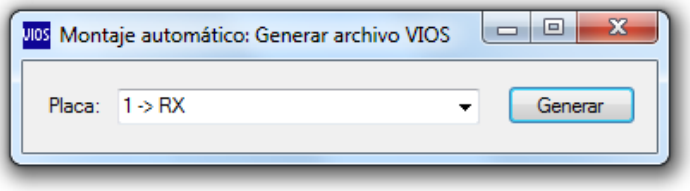


Figura 176. Formulario de montaje mediante archivo VIOS.

Una vez generado el archivo, el usuario lo trasladará a la máquina para seguir en ella el proceso de preparación del montaje como es habitual.

Para conocer el formato de loa archivos VIOS se han analizado algunos archivos de placas de ejemplo, encontrando que son archivos de texto que están estructurados en distintas secciones.

En una primera sección se encuentran algunos datos generales sobre la placa, como por ejemplo su nombre, para pasar luego a especificar cada componente que se debe montar con sus coordenadas de montaje y finalmente las coordenadas significativas de la placa o placas que componen el panel. Esta sección termina con la marca "*End_of_BD*".

Una segunda sección describe los alimentadores y las características físicas de los componentes que termina con la marca "*End_of_FD*". La última sección cierra el archivo y termina con la marca "*End_of_MD*".

Los archivos contienen multitud de datos, por lo que su análisis es una tarea larga y, aunque la aplicación genera archivos que la máquina carga, su análisis completo es una tarea aún en progreso.

5.18. Programa de instalación de la aplicación

La aplicación se puede ejecutar en cualquier ubicación de disco, siempre que se mantengan los ficheros que la componen dentro de la misma carpeta. Por tanto, en este aspecto, es una aplicación "portable" en la terminología usada habitualmente.

Sin embargo, se ha creado un proyecto de instalación en la solución de Visual Studio para que, si el usuario lo desea, se instale en la carpeta *Archivos de Programa*. Además, el instalador crea una entrada en el *Menú de Inicio* de Windows y un icono personalizado en el *Escritorio*. El instalador también asocia la extensión de archivos .pnp al ejecutable de la aplicación.

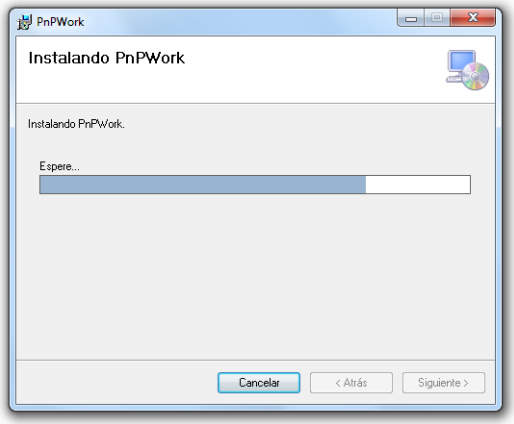


Figura 177. Ventana de progreso de la instalación.

Se han creado iconos personalizados para la aplicación y para los archivos .pnp. También se han creado iconos para todos los formularios del programa. De esta manera se les da un aspecto más profesional. Además, se han creado imágenes ilustrativas para incluir en algunos formularios y una imagen de fondo para el formulario MDI principal. Para editar todos los gráficos se ha usado Gimp, pues, además de ser muy fácil de usar, es software libre.

El instalador además, prepara el ordenador de destino para la ejecución correcta de la aplicación, instalando el *framework* .NET v4.0 si no estaba ya instalado en el ordenador. El proyecto de instalación permite elegir entre descargar los archivos necesarios para la instalación del *framework* de internet o incluirlos en el propio instalador. Se ha decidido incluir todos los archivos necesarios en el propio instalador para evitar descargas durante la instalación, que la ralentizarían. Sin embargo, esta opción tiene la desventaja de aumentar el tamaño del programa instalador.

CAPÍTULO 6. RESULTADOS

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

6.1. Introducción

En este capítulo se presentarán los resultados obtenidos tras la implementación de este proyecto. Durante el desarrollo se han hecho pruebas que verifican el correcto funcionamiento de la aplicación descrita en los capítulos anteriores. Como prueba final, se ha evaluado el resultado del programa al montar una placa completa con la máquina pick & place Cosy, comprobándose que la máquina efectivamente monta los componentes con la aplicación desarrollada. Además, se ha creado una prueba adicional para evaluar el rendimiento en el uso de la memoria, el disco y la

CPU. En otro ámbito, a partir de este proyecto se han presentado dos artículos en el congreso TAEE 2016.

6.2. Montaje de una placa completa

En esta prueba se ha utilizado una placa disponible en el laboratorio, creando en el programa la placa, sus componentes y los alimentadores necesarios partiendo de cero. En la figura 178 se puede ver el ordenador de control con el montaje creado, conectado a la Cosy con la placa de pruebas fijada a su panel de trabajo y los alimentadores de componentes montados.

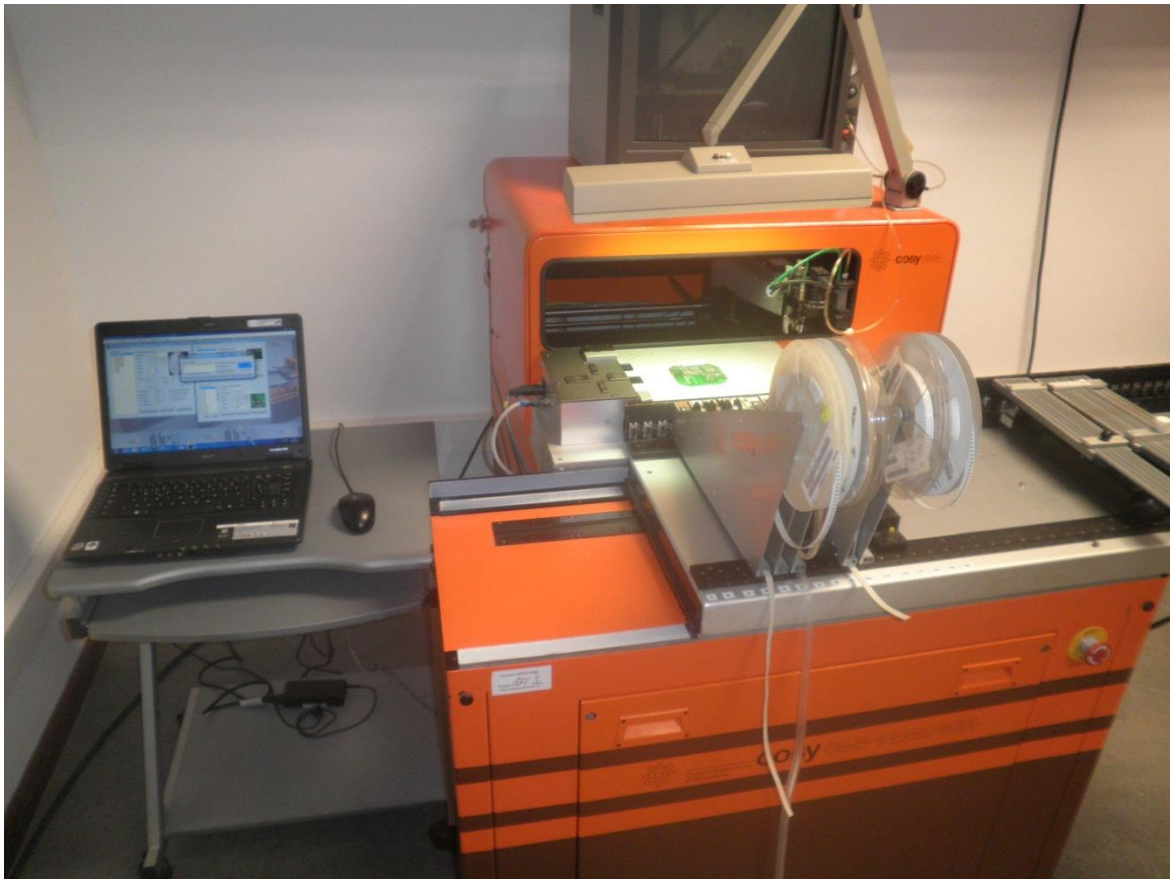


Figura 178. Ordenador de control conectado a la máquina, preparada con la placa de prueba.

Una vez terminada la preparación previa del trabajo, se inició el montaje, quedando los componentes como muestra la figura 179. Todos los componentes fueron colocados en su posición, por lo que podemos considerar la prueba un éxito y que el programa realiza su función correctamente.

Como se puede ver en la imagen, no se depositó pasta de soldar sobre los pads puesto que el objetivo era comprobar que la aplicación colocaba correctamente los componentes. Para evitar que los componentes se descolocasen demasiado durante las operaciones de la máquina, se roció la placa con adhesivo en spray. Esto les dio a los pads suficiente viscosidad para que el

desplazamiento de los componentes una vez colocados fuese mínimo, aunque no tanta como la que normalmente tendrían con la pasta de soldar.

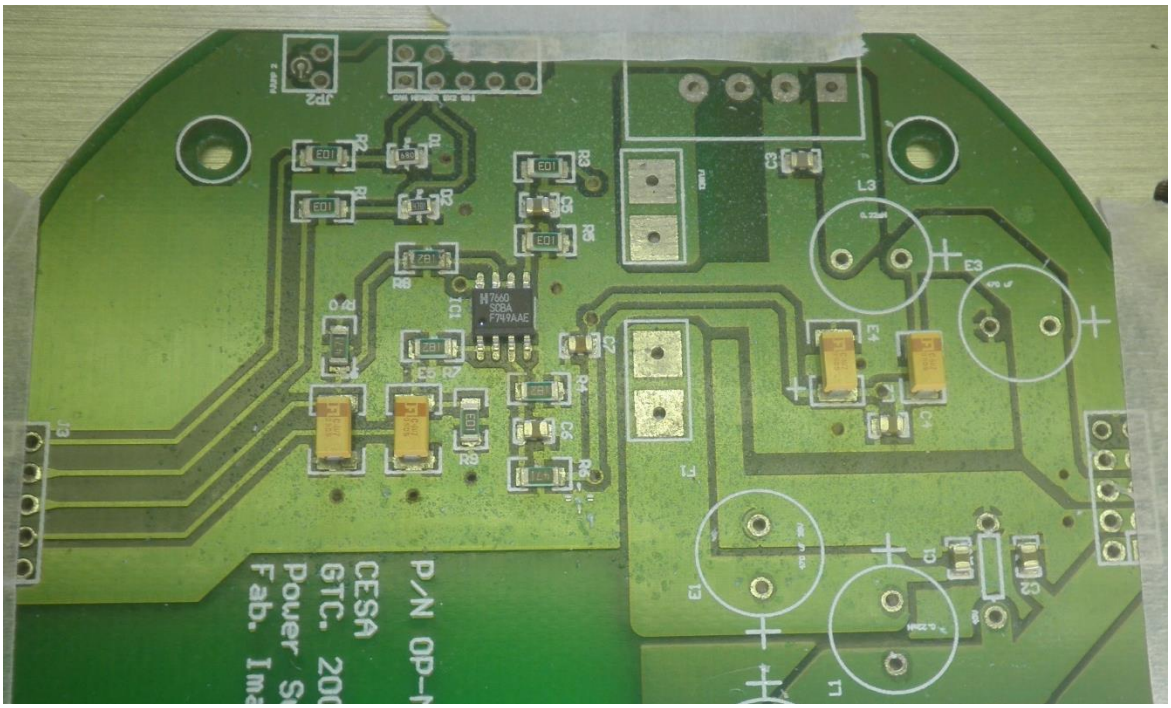


Figura 179. Componentes colocados sobre la placa.

6.3. Prueba de requerimientos de disco, memoria y CPU

6.3.1. Características de la prueba

Para probar los requerimientos de memoria de la aplicación, se creó una composición de placas y alimentadores, buscando posibles problemas de rendimiento bajo condiciones de trabajo con gran cantidad de datos. La prueba creada se puede ver cargada en el programa en la figura 180 y contenía los siguientes elementos:

- 200 alimentadores
- 100 placas
- 100 componentes a montar en cada placa

El número de alimentadores es sensiblemente superior al número de alimentadores que se usarán en cualquier situación. Como se vio en el capítulo 3, dedicado al análisis de las máquinas pick & place, el total de slots disponibles para alimentadores de la Cosy es de 56, algo más de la cuarta parte de esta cantidad. Aun usando alimentadores múltiples, difícilmente se superará esta cifra en el uso cotidiano de la aplicación. Las otras máquinas automáticas soportan un total de algo menos de 100 alimentadores de carrete, por lo que tampoco es probable que se alcance la cifra de alimentadores de la prueba.

Con 100 placas y 100 componentes por placa hacen un total de 10.000 componentes a montar, que es una cantidad sensiblemente superior a la usada en condiciones reales de producción. Para hacernos una idea de la magnitud que representa, esta es la cantidad aproximada de componentes necesaria para cubrir toda el área útil del panel de trabajo de la Cosy, si se colocasen ordenadamente uno junto a otro formando una matriz y todos los componentes tuviesen unas dimensiones de 2 mm x 1 mm, dejando 1 mm de separación entre ellos.

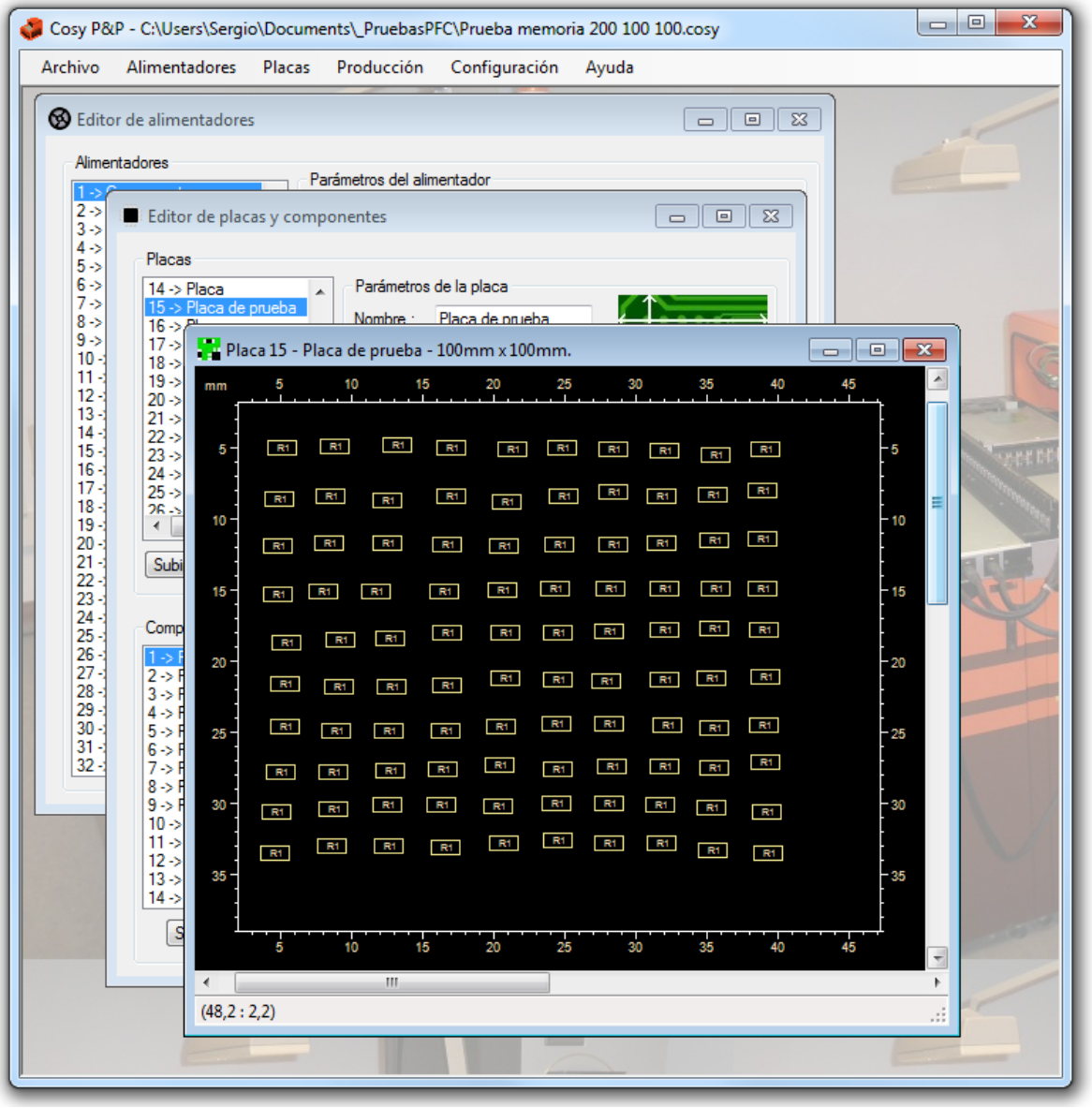


Figura 180. Prueba de requerimientos de disco, memoria y CPU.

La prueba consistió en iniciar la aplicación y cargar el archivo de datos creado anteriormente, pasando a continuación a dejarlo a punto para iniciar el montaje, abriendo la ventana de producción. En cada paso se tomó la medida de memoria ocupada y se realizó la

prueba diez veces de forma similar para tomar la media, puesto que se observó pequeñas diferencias entre una prueba y otra.

6.3.2. Espacio en disco ocupado

En las condiciones mencionadas anteriormente, el archivo de datos creado ocupó tan sólo 402 kilobytes en disco. El espacio ocupado no es una cantidad grande, si lo comparamos con archivos de datos de otras aplicaciones de uso habitual.

La grabación de archivos se ha optimizado de forma que sólo se graban datos relativos a los archivos Gerber si se asocian archivos válidos. Asociando todos los archivos Gerber posibles a todas las placas de la composición descrita, el espacio ocupado por el archivo de datos pasó a aumentar en 30 KB, que sigue siendo una cantidad casi despreciable para los estándares de almacenamiento actuales.

En cuanto los archivos necesarios para ejecutar la aplicación, el archivo ejecutable de ésta ocupa 759 KB, junto con las DLL incluidas y el archivo de configuración XML ocupan en total aproximadamente 4,63 MB. El resto de archivos de la aplicación son el manual del usuario y la guía rápida, ambos en formato PDF, que suman aproximadamente 4 MB. Esto hace un total de menos de 9 MB, que es un tamaño bastante discreto comparado con otras aplicaciones.

Como comparación, los archivos del programa original de MS-DOS (shaupt.exe) ocupan en conjunto 405 KB. Por otra parte el archivo de datos para una placa de 24 componentes, como la de la figura 179, ocupaba 52,5 KB, mientras que la misma placa en la aplicación creada para este proyecto ocupó 2,11 KB.

Tras un pequeño análisis de los archivos de datos de shaupt.exe, se intuyó que esto se debe a que aparentemente, además de los datos del montaje, guarda las secuencias de comandos para operaciones comunes (cambiar de útil, centrado, etc.) así como datos relativos a las características físicas de la máquina con la que está trabajando (offset de la cámara y del dispensador, velocidades, etc.). En nuestro caso, todos los datos de la máquina se guardan en un lugar común: las propiedades de configuración de la aplicación, en un archivo XML. Por otro lado, las secuencias de comandos para las tareas comunes de la máquina están programadas en el propio código de la aplicación.

De esta manera, el archivo de datos del montaje sólo contiene los datos relativos a alimentadores, placas y componentes (aparte de la información relativa a los archivos Gerber). Por tanto, podemos concluir que la eficiencia en el uso de disco por parte de la aplicación de este proyecto es superior a la del programa original.

6.3.3. Memoria RAM ocupada

Para medir la memoria RAM ocupada por la aplicación se utilizó el Administrador de tareas de Windows, que tiene la capacidad de mostrar la cantidad de memoria ocupada por los distintos procesos que hay iniciados en la máquina.

Esta prueba se realizó en dos situaciones. Inicialmente se hizo utilizando los ejecutables del primer entregable (primera versión), que sólo incluía el código para trabajar con la máquina Cosy. Posteriormente se repitió la última parte con el ejecutable final (segunda versión), que dispone de todas las opciones de trabajo.

Para hacer la primera prueba se utilizó inicialmente un ordenador PC clónico con un procesador de un solo núcleo y 1 GB de memoria RAM, bajo el sistema operativo Windows XP, con el programa compilado en modo *Debug*. El análisis realizado de la memoria ocupada, al poner en marcha la aplicación y cargar el fichero de datos creado arrojó los resultados de la tabla 1.

Acción realizada	Memoria ocupada
Se inicia la aplicación con su acceso directo	17.348 KB
Se pulsa en el menú Archivo → Abrir y se abre el cuadro de diálogo para abrir fichero (sin seleccionar ninguno aún)	22.864 KB
Se carga en memoria el fichero de prueba	24.712 KB
Se pulsa en el menú Producción → Iniciar Producción y se abre la ventana de producción, con todas las placas preseleccionadas e indicando que quedan 10.000 componentes por montar	25.576 KB

Tabla 1. Resultados de la prueba de la 1ª versión en Windows XP, con modo Debug (valores medios).

Posteriormente la prueba se repitió en un ordenador más moderno, un Acer Aspire M1610, con 4 GB de RAM y bajo el sistema operativo Windows 7. La prueba se realizó de la misma forma que la anterior sobre Windows XP.

En esta ocasión el programa se compiló en modo *Release*, para eliminar la sobrecarga que añade el modo *Debug*. Esta sobrecarga se debe a que Visual Studio añade código y datos necesarios para la depuración del programa y mostrar al usuario los puntos conflictivos del código fuente.

El resultado fue que la aplicación pasó a ocupar casi un 40 % menos de memoria, como se puede ver en la tabla 2. También se puede ver que, si se configura la aplicación para que abra los editores automáticamente al cargar el archivo de datos, la memoria ocupada aumenta aproximadamente en un 14 % respecto a si se dejan los editores cerrados.

Acción realizada	Memoria ocupada (editores cerrados)	Memoria ocupada (editores abiertos)
Se inicia la aplicación con su acceso directo	6.816 KB	6.792 KB
Se pulsa en el menú Archivo → Abrir y se abre el cuadro de diálogo para abrir fichero (sin seleccionar ninguno aún)	14.760 KB	14.588 KB
Se carga en memoria el fichero de prueba	15.456 KB	17.800 KB
Se pulsa en el menú Producción → Iniciar Producción y se abre la ventana de producción, con todas las placas preseleccionadas e indicando que quedan 10.000 componentes por montar	15.628 KB	17.796 KB

Tabla 2. Resultado de la prueba de la 1ª versión en Windows 7, con modo Release (valores medios).

Como se puede ver en las cifras de la última fila, la cantidad de memoria ocupada no es excesiva para los estándares de hoy en día.

A primera vista, en los datos medios de la tabla anterior, las dos primeras filas deberían contener cantidades iguales. Aunque la diferencia es muy pequeña, esto no es así porque el valor de memoria ocupada no es una cantidad fija, sino que varía frecuentemente, dependiendo del momento en que se active el recolector de basura para recuperar la memoria ocupada por los objetos que ya no están en uso. Por tanto, el valor de memoria ocupada va variando con el tiempo aun cuando el usuario no esté realizando ninguna acción sobre la aplicación.

Repitiendo la prueba con la versión final del ejecutable se obtuvieron los resultados de la tabla 3. Se puede ver, que tras cargar el mismo archivo de datos, el consumo de memoria aumentó a casi el doble. Este aumento en el consumo de memoria es lógico, puesto que la segunda versión declara varios campos necesarios para la gestión de las imágenes Gerber que no existían en la primera y el sistema reserva su memoria aunque no se usen.

Acción realizada	Memoria ocupada
Se inicia la aplicación con su acceso directo	20.856 KB
Se pulsa en el menú Archivo → Abrir y se abre el cuadro de diálogo para abrir fichero (sin seleccionar ninguno aún)	33.388 KB
Se carga en memoria el fichero de prueba	30.720 KB
Se pulsa en el menú Producción → Iniciar Producción y se abre la ventana de producción, con todas las placas preseleccionadas e indicando que quedan 10.000 componentes por montar	30.384 KB

Tabla 3. Resultado de la prueba de la 2ª versión en Windows 7, con modo Release (valores medios).

En la figura 181 se muestran los datos anteriores en una gráfica que resume cómo varía la memoria ocupada con las acciones llevadas a cabo. La curva central se corresponde con la ocupación de memoria en modo *Debug*. Esta curva se ha incluido como curiosidad, puesto que este modo sólo se usó durante el desarrollo de la aplicación. Las versiones finales se han compilado en modo *Release*, que corresponden con las curvas inferiores para el primer entregable (V1) y la superior para la versión final (V2).

Se puede comprobar que la diferencia de memoria ocupada al tener abiertos los formularios editores o tenerlos cerrados es bastante pequeña. Sin embargo, el código y las librerías extra de la aplicación en la versión 2 hacen que la ocupación de memoria de la aplicación se incremente en unos 15 MB en la última versión.

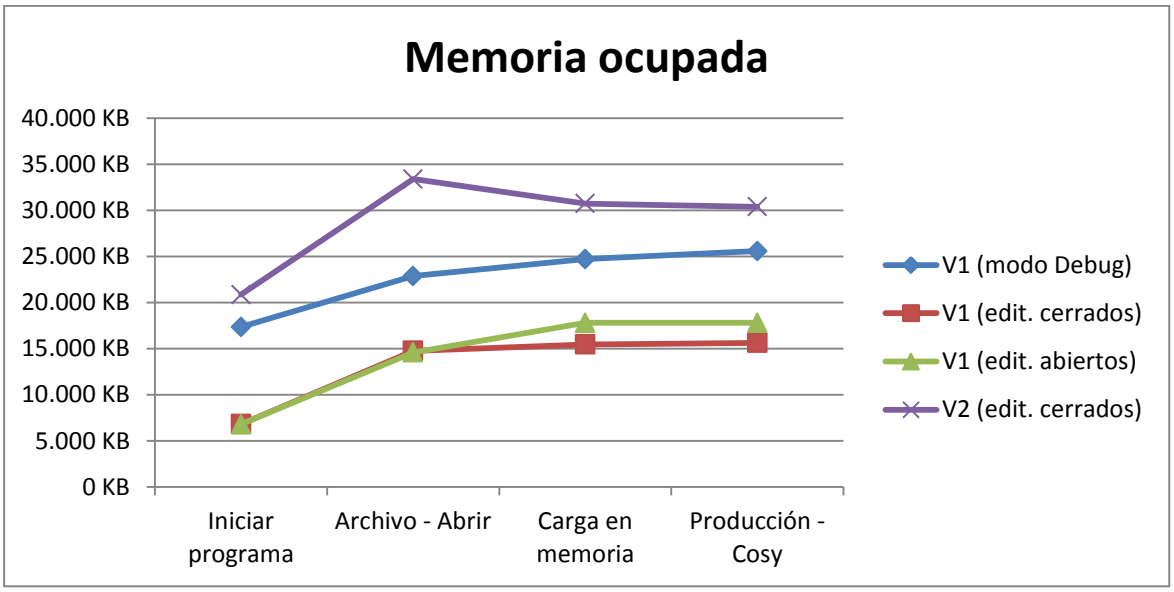


Figura 181. Gráfica de evolución de la memoria ocupada.

Se realizó una última prueba de rendimiento de memoria, abriendo el visor/editor gráfico de cada placa, que muestra la disposición de los componentes sobre su superficie. Con las 100 ventanas abiertas, la aplicación ocupó 49.412 KB de memoria con el ejecutable del primer entregable, mientras que con el ejecutable de la aplicación final ocupó 81.956 KB. Como se puede ver, la memoria ocupada es un 63% mayor, lo cual es lógico tal como se dijo anteriormente.

6.3.4. Carga de la CPU

Para comprobar la carga de la CPU se aprovechó la característica del Administrador de tareas de Windows que la muestra junto a la memoria ocupada. También se observó la velocidad de funcionamiento de la aplicación subjetivamente, desde el punto de vista del usuario.

Al cargar el archivo con la configuración de abrir los editores desactivadas, no se notó ralentización alguna del programa. Tras activar las opciones de configuración "Abrir editor de alimentadores al cargar" y "Abrir editor de placas al cargar", se repitió la prueba. Esto hizo que tras cargar el archivo en las estructuras de memoria tuviera que abrir las dos ventanas mencionadas y cargar en sus listados todos los datos de memoria para mostrarlos en pantalla. Esta operación tardó unos 2 segundos.

A continuación se activó la opción de configuración "Ver placas al cargar" y se repitió la prueba. Esto hizo que tuviese que abrir una nueva ventana para el editor gráfico de cada placa y dibujar sus 100 componentes en pantalla antes de pasar a abrir la siguiente. El tiempo total de esta operación fue de 11 segundos. Como en total eran 100 placas, esto significa que el programa dibujó 9 placas por segundo. Estas placas no tenían asociados archivos Gerber y por tanto sólo se dibujaron los componentes.

Por su parte, el uso de la CPU reflejado por el Administrador de Tareas durante el breve instante que dura el arranque de la aplicación fue del 15 %, mientras que el uso de CPU más elevado se produjo mientras mostraba las 100 ventanas mencionadas anteriormente, en que no pasó del 50 %.

Esto era de esperar debido a que, aunque es una cantidad total de componentes para montar respetable, desde el punto de vista del software, la cantidad de datos que representa no es excesivamente grande comparada con la velocidad de proceso de los microprocesadores relativamente modernos (el modelo Aspire M1610 tiene aproximadamente 9 años de antigüedad). En un equipo actual, las cifras de consumo de CPU seguramente se verían disminuidas.

6.4. Artículos derivados de este proyecto

Como resultado adicional de este proyecto se han presentado dos artículos en la asociación TAEE que han sido aprobados para su comunicación en el XII Congreso TAEE [**44**] (Sevilla, 22 al 24 de junio, 2016).

TAEE [**45**] es una asociación de profesores de enseñanza superior cuyo objetivo es mejorar la docencia en el ámbito de la electrónica mediante la reflexión conjunta de los problemas a los que se enfrenta, la generación de recursos didácticos, principalmente con base tecnológica, y el fomento de la reutilización y la generación cooperativa del conocimiento.

La finalidad del TAEE es la promoción y el desarrollo de metodologías activas de aprendizaje y de una enseñanza con una fuerte vinculación a la práctica profesional y en estrecha vinculación con las buenas prácticas en investigación y desarrollo.

El primer artículo se titula "*Software de Control para Máquinas Pick & Place*". En él se presenta fundamentalmente el análisis realizado a la máquina pick & place Cosy y la aplicación que se desarrolló como primer entregable en la primera etapa de este proyecto. También se describe el método de trabajo creado para operar con esta máquina en combinación con el software creado.

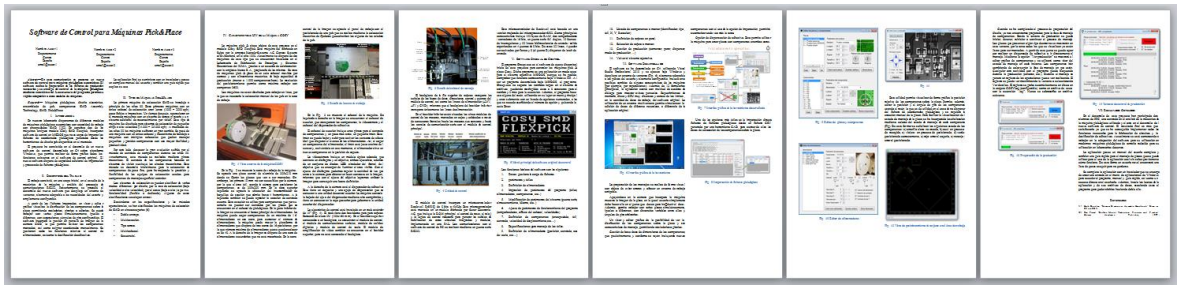


Figura 182. Artículo *"Software de Control para Máquinas Pick & Place"*.

El segundo de los artículos se titula "*Herramienta de ayuda a la producción de prototipos electrónicos con tecnología SMD*". En éste se describe la aplicación realizada en este proyecto dando énfasis a la parte desarrollada a partir de la segunda etapa y al uso en el laboratorio para la producción de prototipos tanto mediante métodos manuales como automáticos, así como líneas futuras de desarrollo.

Adicionalmente, durante la celebración del congreso se hará una demostración práctica de funcionamiento del software. En ésta, los asistentes recibirán explicaciones del método de trabajo con las máquinas pick & place del laboratorio, cómo utilizar esta aplicación como ayuda en los procesos de fabricación y cómo es posible usarla para mejorar los procesos con cualquier otra máquina de montaje SMD manual o semiautomática en centros de enseñanza o pequeños talleres.

Las comunicaciones presentadas se publicarán en abierto, además de en el libro de actas general del congreso, en los repositorios digitales de la Asociación TAEE y en el de aquellas entidades públicas con las que se tiene convenio, con el fin de proporcionar la máxima visibilidad a nivel internacional. El congreso también cuenta con la co-esponsorización técnica de IEEE, con lo que, opcionalmente, también pueden ser publicadas en IEEE Xplore.

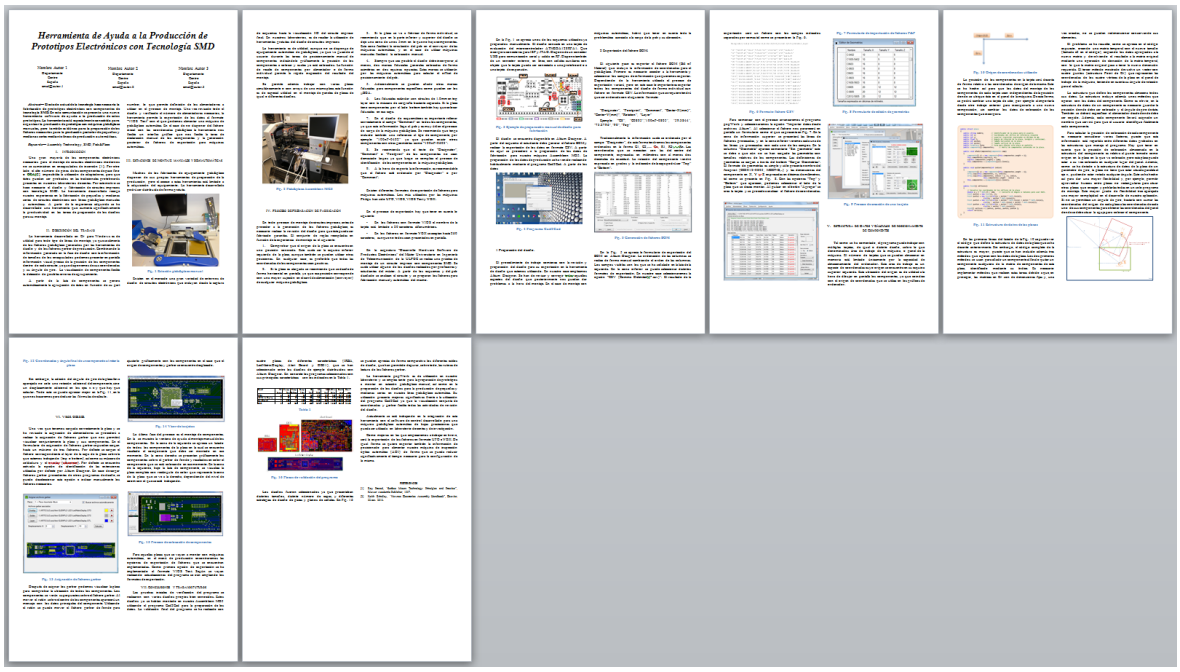


Figura 183. Artículo "*Herramienta de ayuda a la producción de prototipos electrónicos con tecnología SMD*".

La idea que se persigue con esta presentación es que la aplicación sea útil más allá del laboratorio del SFP, dándole mayor visibilidad y permitiendo que se pueda usar en otros centros de enseñanza o de producción.

CAPÍTULO 7. PRESUPUESTO

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

7.1. Introducción

A lo largo de este presupuesto se hace balance de los costes tanto parciales como totales presentes en este proyecto, distinguiendo entre ellos cuatro grupos principales: recursos materiales, costes de ingeniería, material fungible y costes de edición del proyecto.

7.2. Recursos materiales

En este apartado se realiza el análisis de los costes derivados del uso del material *hardware,* paquetes *software* y el mantenimiento de éstos. Estos equipos *hardware* y paquetes *software* presentan un coste de amortización, en función del período de tiempo usados.

7.2.1. Recursos hardware

En él se incluye el cálculo del coste total para los recursos *hardware* usados para la realización de este proyecto. Aunque ocasionalmente se pueden haber usado otros equipos para una prueba concreta, se ha trabajado principalmente en dos equipos informáticos:

- Ordenador de sobremesa Acer Aspire M1610
- Ordenador portátil Acer Travelmate 5720

Ambos recursos *hardware* se han utilizado para el desarrollo del proyecto bajo la plataforma Windows XP Professional y Windows 7 Professional.

Para el estudio de la máquina *pick & place* y las pruebas de la aplicación creada ha sido necesario acceder a una de las máquinas Cosy SMD Flexpick instaladas en el SFP del IUMA, pero pertenecientes a Inerza S.A., gracias al convenio de colaboración IUMA-Inerza. También ha sido necesario utilizar brevemente el analizador lógico Link Instruments IO3208.

El software creado se usó en el ordenador portátil, trabajando en combinación con un adaptador USB ←→ RS-232 y una digitalizadora de vídeo modelo *AVerMedia AVerTV Volar Video Capture USB,* en las ocasiones en que se probó el programa conectado a la máquina.

Para todos los recursos informáticos se ha supuesto una amortización a tres años por un promedio de un usuario. La tabla 4 resume todos estos recursos.

Tras consultar, se confirmó que la máquina pick & place Cosy había sido completamente amortizada en sus años de uso en la empresa, anteriores a su etapa actual en el laboratorio, por lo que no se puede amortizar de nuevo.

El resto de máquinas pick & place mencionadas en este proyecto no han sufrido un tiempo de uso significativo para incluirlas en los cálculos, puesto que la metodología seguida en el desarrollo y pruebas de este proyecto no lo ha requerido.

Descripción	Coste	Factor amortización		Coste por mes	Meses	Coste Final
		A 3 años	Usuarios			
Acer Aspire M1610 Intel® Core™ 2 Duo E4500 2,2 GHz 4 GB RAM, Microsoft Windows XP Professional	800,00 €	0,028 (1/36)	1	22,40 €	24	537,60 €
Ordenador Portátil Acer Travelmate 5720 Intel® Core™ 2 Duo T5670 1,8 GHz 4 GB RAM, Microsoft Windows 7 Professional	800,00 €	0,028	1	22,40 €	24	537,60 €
Analizador lógico Link Instruments IO3208	550,00 € (750 $)	0,028	1	15,40 €	0,1 (3 días)	1,54 €
Digitalizadora AVerMedia AVerTV Volar Video Capture USB	40,00 €	0,028	1	1,12 €	2	2,24 €
Cable USB ←→ RS-232	10,00 €	0,028	1	0,28 €	2	0,56 €
Total del costes de recursos *hardware*						**1.079,54 €**

Tabla 4. Costes asociados a los recursos hardware.

7.2.2. Recursos software

El *software* utilizado, no asociado al *hardware*, ha sido Visual Studio Professional 2010, Office 2010, Free Serial Port y Gimp. Tanto Visual Studio como Office 2010 se han obtenido con licencia universitaria. En el caso de Visual Studio por medio del programa Dreamspark de Microsoft.

Durante un breve periodo inicial se utilizó Visual C# 2010 Express, en lugar de Visual Studio, pero este software lo ofrece Microsoft también gratuitamente. La licencia de Free serial Port permitía el uso libre para uso no comercial, como es el caso de este proyecto, por lo que no tuvo coste asociado. Gimp es software libre y gratuito, por lo que tampoco tuvo coste asociado.

En cuanto a las bibliotecas DLL incorporadas a la aplicación desarrollada, todas eran de uso libre para los propósitos de este proyecto, por lo que tampoco han tenido coste asociado.

7.2.3. Coste asociado a la explotación de las infraestructuras

Este coste es producido por el personal técnico empleado para el mantenimiento de las infraestructuras y está formado por un técnico de tiempo completo para un promedio de cien usuarios. El coste de estos recursos se detalla en la tabla 5.

Descripción	Coste (€/año)	Cantidad	Factor de Amortización (100 usuarios)	Coste por mes (€)	Meses	Coste Final (€)
Un técnico a tiempo completo	12.040,00	1	0,01	120,40	24	2.889,60
Total de costes asociados a la explotación de las infraestructuras						**2.889,60**

Tabla 5. Costes asociados a la explotación.

7.3. Costes de ingeniería

Se ha invertido un total de 24 meses a media jornada en el desarrollo de este proyecto. Durante este tiempo se han llevado a cabo tareas de formación y estudio del lenguaje, estudio y análisis de la máquina, desarrollo de los métodos de operación, desarrollo de la aplicación, validación y pruebas, documentación y redacción de la memoria del proyecto. Como muchas de estas tareas se solapan en el tiempo se han estimado los períodos de tiempo en función de la tarea predominante. En este presupuesto se ha incluido los costes de formación, al ser específicos para el desarrollo de este proyecto. Además, se ha considerado que la tarifa por costes de ingeniería aplicada a un ingeniero *junior* en España a media jornada asciende a un total de 1.500 €/mes, englobando en la misma el salario, la Seguridad Social, y el IRPF (Impuesto sobre la Renta de las Personas Físicas). La tabla 6 muestra los costes de ingeniería desglosados.

Descripción	Meses	Coste (€/mes)	Coste Final (€)
Formación y estudio del lenguaje	5	1.500,00 €	**7.500,00 €**
Estudio y análisis de la máquina	1	1.500,00 €	**1.500,00 €**
Desarrollo de los métodos de operación y desarrollo de la aplicación	15	1.500,00 €	**22.500,00 €**
Validación y pruebas	1	1.500,00 €	**1.500,00 €**
Documentación y memoria	2	1.500,00 €	**3.000,00 €**
Total de costes de ingeniería			**36.000,00 €**

Tabla 6. Costes de ingeniería.

7.4. Material fungible

En este apartado se pretende englobar los gastos derivados de los materiales utilizados en la realización del proyecto, como son: papel, discos para copias de seguridad, componentes y otro material fungible del laboratorio. Se calcula un coste de 200 €.

7.5. Coste de edición

El coste de edición de la memoria (discos, folios, gastos de impresión y encuadernación) ha sido de 400 €.

7.6. Coste total

En la siguiente tabla se muestra el coste total en función de los costes parciales indicados en las secciones anteriores.

Descripción	Coste (€)
Recursos Materiales	3.969,14 €
Costes de Ingeniería	36.000,00 €
Material Fungible	200,00 €
Edición de la memoria	400,00 €
Coste total antes de impuestos	**40.569,14 €**
7 % IGIC	**2.839,84 €**
Coste total después de impuestos	**43.408,98 €**

Tabla 7. Costes totales.

El coste total asciende a 43.408,98 €.

CAPÍTULO 8. CONCLUSIONES Y TRABAJOS FUTUROS

Herramienta software de gestión para la producción de sistemas electrónicos con máquinas pick & place

8.1. Introducción

En este capítulo se presentan las conclusiones generales de este Proyecto Fin de Carrera, resaltando la consecución de los objetivos planteados inicialmente en el capítulo 1. Para finalizar, se plantearán posibles líneas de trabajo que quedan abiertas tras su realización.

8.2. Conclusiones

En este proyecto se ha estudiado en profundidad el funcionamiento de la máquina Cosy SMD Flexpick, extrayendo gran parte de los datos técnicos de operación de la misma, su capacidad para realizar diferentes subtareas del proceso de montaje de componentes SMD y sus límites físicos para estas subtareas.

Una vez analizado el hardware y software original de esta máquina se analizó su protocolo de comunicaciones y se han desarrollado los métodos de trabajo para usar con el nuevo software de control. Estos métodos se han orientado hacia un uso más ágil de la aplicación del que representaba el software original, tratando de potenciar la capacidad de simultanear la edición de los diferentes datos necesarios para montar los componentes.

Posteriormente se han utilizado los datos extraídos de los análisis anteriores para desarrollar una aplicación informática que reemplace al software original de la misma. La nueva aplicación ofrece, además de las funciones relevantes del software original, nuevas funciones, inexistentes anteriormente que extienden las capacidades de uso. Estas funciones permiten al operador tratar ágil y visualmente los parámetros del montaje.

A continuación, se ha estudiado el método de trabajo con otras máquinas en el laboratorio del SFP, en concreto la máquina semiautomática LKPF Protoplace S y las máquinas automáticas Assembleon MG-5 y Philips CSM 84 III. Con la información obtenida, se han añadido nuevos módulos a la aplicación desarrollada que permiten usarla con estas máquinas u otras, tanto manuales como automáticas.

De especial interés es el módulo creado para la producción manual, ya que este módulo permitirá extender su utilidad no sólo al SFP, donde fue desarrollado, sino posiblemente a multitud de usuarios en otros centros que no disponen de recursos para adquirir máquinas de montaje automatizado. Con este objetivo, se han presentado dos artículos en el congreso TAEE 2016 que amplían enormemente la visibilidad de este trabajo y del SFP en la comunidad.

La aplicación incluye muchas características que facilitan el trabajo al usuario, que se resumen en el siguiente apartado. Se puede concluir que **la aplicación desarrollada cumple con**

creces los objetivos iniciales planteados, añadiendo características adicionales y nuevos objetivos que no estaban en los requerimientos iniciales, pero que se fueron incorporando a medida que se desarrollaba el proyecto.

8.2.1. Resumen de características de la aplicación desarrollada

La aplicación desarrollada permite montar componentes SMD utilizando cualquiera de las máquinas automáticas o manuales (semiautomáticas) existentes en el laboratorio del SFP. Además de montar los componentes, en el caso de la máquina pick & play Cosy, la aplicación puede dosificarles previamente el adhesivo necesario para montar componentes por ambas caras. La aplicación se puede usar además como ayuda en la etapa de montaje de componentes de inserción, mediante las opciones de montaje manual.

Para el montaje automático usando la Cosy, la aplicación desarrollada es capaz de dialogar directamente con la máquina pick & place a través del puerto serie, implementando el algoritmo completo de montaje, enviándole los comandos necesarios para ejecutarlo y testeando las posibles condiciones de error.

Para el montaje automático con las otras máquinas, la aplicación genera un fichero en formato VIOS Text que indica a la máquina los parámetros necesarios de los componentes y la placa. Esto permitiría utilizar el programa también con otras máquinas diferentes, pero que operen a partir de este tipo de archivos.

Para el montaje manual, la aplicación asiste interactivamente al montador indicándole en pantalla uno por uno los componentes a mostrar y la ubicación sobre la placa. Los componentes se pueden ordenar utilizando como criterio cualesquiera de sus parámetros (hasta 3 parámetros simultáneamente) de forma que queden agrupados por alimentador, nombre, coordenadas, etc. Además, la aplicación permite generar unas instrucciones gráficas de montaje en formato PDF, en las que se identifican los componentes a montar en cada paso y se muestran sobre una imagen de la placa. Adicionalmente a la ordenación mencionada anteriormente, cada paso se puede hacer de forma individual o agrupando los componentes en una imagen por alimentador.

La aplicación es capaz de leer e interpretar los archivos Gerber de la placa, generando una imagen de ésta que muestra sus pistas, pads y serigrafía (u otras capas que elija el usuario). Esta imagen se utiliza en el editor gráfico de placas y componentes, en las vistas en pantalla durante el montaje manual interactivo y en la generación de documentos PDF.

La aplicación es capaz de interpretar ficheros BOM de Altium para integrar en la placa automáticamente los componentes a montar. Para ello se apoya en un archivo de dimensiones de

los componentes, que por defecto se carga y graba automáticamente, creando una base de datos de componentes en la que se van acumulando componentes usados en el pasado, reduciéndose así el trabajo nuevo a realizar. Aunque este archivo tiene un formato de texto muy fácil de editar con una aplicación externa (archivo CSV de Excel), la aplicación incluye un editor interno que hace más cómoda la tarea al operador. También es posible importar archivos de dimensiones usados habitualmente con otros programas, como Cad2cad.

En caso de que no se disponga del archivo de salida de Altium, los editores integrados en la aplicación permiten crear fácil y rápidamente todos los datos necesarios. Se han diseñado los formularios para minimizar la posibilidad de introducir datos erróneos. Para ello se han puesto límites a los datos que es posible introducir en cada campo (configurables por el usuario según las características de la máquina) y avisos evidentes para el usuario cuando hace algo incorrecto.

Durante la edición de datos de montaje con la Cosy, se puede utilizar la cámara integrada en la máquina para definir la posición de cualquier elemento y el ángulo de giro de las placas sobre el panel. Se ha mejorado el sistema de posicionamiento del software anterior, incluyendo la posibilidad de hacer clic con el ratón directamente sobre la imagen obtenida de la cámara para especificar el punto deseado. En este caso la aplicación se encarga de calcular las coordenadas correspondientes y rellenar el formulario automáticamente.

La aplicación permite anexar al trabajo actual los datos que se deseen de cualquier trabajo anterior grabado en disco, siendo una nueva fuente de ahorro de tiempo. Además, esta característica permite a la aplicación trabajar fácilmente en el montaje de placas panelizadas (agrupadas). Las placas panelizadas pueden ser iguales o diferentes y pueden estar giradas o separadas por distancias diferentes. Se pueden anexar individualmente o en conjunto y se puede usar la cámara para definir muy rápidamente su ubicación y ángulo de giro.

El software creado guarda automáticamente las preferencias del usuario seleccionadas en el menú de configuración. También guarda las dimensiones y estado de la ventana principal al cerrarse. De esta forma, al iniciarse nuevamente, queda configurada igual y con la misma apariencia que el operador definió la última vez que la usó. Además, el usuario puede cambiar en cualquier momento la configuración de la aplicación para trabajar con una sola ventana principal que contiene al resto (MDI) o como ventanas independientes, según sus preferencias.

La aplicación es altamente configurable, permitiendo modificar todos los parámetros físicos relevantes de las máquinas y otros datos fijos de la aplicación, para que el usuario cambie los valores en caso de necesidad. Se pueden establecer las dimensiones del área de trabajo según

la máquina que se va a utilizar o bien establecer dimensiones arbitrarias. Otros datos de configuración avanzada se guardan en un archivo XML junto al ejecutable de la aplicación y no están disponibles en el menú de configuración para evitar su modificación por usuarios inexpertos. Aun así, su edición es muy sencilla y se explica en el manual del usuario.

El límite teórico para el número de alimentadores, placas o componentes de un montaje que la aplicación puede manejar es de algo más de 4 mil millones [**46**], por lo que en la práctica podemos decir que el programa no tiene límite para la cantidad de elementos que puede tratar en un caso real. Esto lo hemos comprobado en una de las pruebas del capítulo 6.

De los resultados de las pruebas realizadas en el capítulo 6 se extrae la conclusión de que los requerimientos de memoria, procesador y disco de la aplicación son muy pequeños, por lo que se puede utilizar en un ordenador que haya quedado obsoleto para otras tareas.

Se ha creado un programa instalador que facilita al usuario la preparación del ordenador para ejecutar la aplicación. Este instalador detectará automáticamente si el ordenador cumple los requisitos software mínimos para iniciar la aplicación e instalará lo que falte. No es necesario descargar de internet este software extra, puesto que se ha incluido en el instalador.

El instalador crea un grupo en el menú inicio del usuario, con accesos directos a la aplicación, el archivo léame, el manual del usuario y la guía rápida para el montaje. Estos dos últimos también son accesibles desde el menú ayuda de la aplicación. El instalador crea también la asociación de archivos necesaria para que al hacer doble clic sobre un archivo de datos en el explorador de Windows, éste se abra automáticamente con la aplicación.

Se ha dado un aspecto profesional a la aplicación, creando iconos para ésta, sus archivos de datos y cada ventana del programa, junto a otros detalles como las funciones del instalador.

La aplicación permite la ejecución de varias instancias simultáneas, lo detecta y avisa al usuario de las restricciones de una segunda instancia sobre los recursos que están en uso en la primera instancia. La segunda instancia se puede usar, por ejemplo, para hacer operaciones de un segundo trabajo mientras el primero se está montando con la máquina Cosy. Teóricamente, con dos puertos serie instalados en el ordenador y dos digitalizadoras de vídeo, es posible utilizar completamente ambas máquinas Cosy existentes en el laboratorio, controladas por el mismo ordenador. Así, desde el mismo ordenador se pueden hacer los preparativos previos en una máquina mientras la otra monta componentes o poner ambas máquinas existentes a montar componentes simultáneamente.

8.3. Trabajos futuros

Como continuación del trabajo realizado en este proyecto, se pueden plantear nuevos trabajos sobre el código desarrollado para mejorar el uso de las máquinas pick & place.

- En el montaje automático con la Cosy se puede realizar una optimización adicional a la de ordenar por alimentadores para realizar el montaje. Si se van a montar varias placas que están en el panel, en lugar de recorrer en orden el vector de placas para montarlas de una en una, sería posible crear una lista de montaje que incluya todos los componentes de todas las placas. Reordenando los componentes a nivel global y montando los componentes de todas las placas simultánea e indistintamente se multiplica el ahorro de tiempo y desgaste mencionado anteriormente. Esto ayudará a darle más tiempo vida a una máquina con bastantes años de producción, pero muy capaz de seguir en línea.
- La aplicación implementa un zoom muy básico en la vista de montaje manual, que permite reducir el tamaño de lo que se está viendo para ver toda la placa y saber dónde está ubicado el componente que se está montando. Sin embargo, la función que genera la imagen Gerber es capaz de generar una imagen con cualquier factor de escala y sin defectos de pixelado pues se trata de gráficos vectoriales. Se podría aprovechar esto para, fácilmente incorporar un factor de zoom continuo, por ejemplo, utilizando la rueda del ratón para ampliar y disminuir el tamaño.
- Se pueden crear nuevos métodos en el apartado de producción para generar otros tipos de archivos de control para máquinas iguales o diferentes a las que posee el laboratorio. Existen muchos tipos de archivo para el montaje automatizado que, por lo general, dependen del fabricante o modelo de máquina con el que se trabaje. De esta forma sería posible usar la aplicación para trabajar con más modelos de máquinas pick & place. De especial interés se ha encontrado el formato YGX, pues se trata de un archivo XML, con campos bien definidos y del que puede ser posible disponer de más información de partida que con el formato VIOS.

BIBLIOGRAFÍA

[1] Seventeenth IEEE/CPMT International Electronics Manufacturing Technology Symposium, "Manufacturing Technologies – Present and Future," , Octubre 1995.

[2] Wikipedia. (2013, abril) Tecnología de agujeros pasantes. [Online]. http://es.wikipedia.org/wiki/Tecnolog%C3%ADa_de_agujeros_pasantes

[3] Wikipedia. (2013, abril) Tecnología de montaje superficial. [Online]. http://es.wikipedia.org/wiki/Tecnolog%C3%ADa_de_montaje_superficial

[4] Roberto Heyer, *Tecnologías de armado de módulos electrónicos*, 1st ed. Buenos Aires: Dunken, 2004.

[5] O. E. Flippo, J. J. V. D. Klundert, F. C. R. Spieksma Y. Crama, *The assembly of printed circuit boards: a case with multiple machines and multiple board types*.: European Journal of Operational Research, 98, 457-472, 1997.

[6] Fernando J. Vittes, *Optimizing th performance of a chip shooter machine*., 1999.

[7] S. Grotzinger, D. Johnson J. Ahmadi, *Component allocation and partitioning for a dual delivery placing machine*.: Operations Research, 36, 176-191, 1997.

[8] P.C. Nelson, T. M. Tirpak W. Wang, *Optimization of high speed multistation SMT placement machines using evolutionary algorithms*.: IEEE Transactions on electronics Packaging Manufacturing, 22(2), 137-146, 1999.

[9] J. E. Kobza, F. J. Vittes K. P. Ellis, *Development of placement time estimator function for a turret style surface mount placement machine*.: Robotic and Computing Integrated Manufacturing, 18, 241-254, 2002.

[10] G Kendall M. Ayob, *Real-time scheduling for multi headed placement machine*. Francia: Proceedings of the 5th IEEE International Symposium on Assembly and Task Planning, ISATP'03, 128-133, Julio 2003.

[11] R. W. Clayton, T. A. Feo J. F. Bard, *Machine setup and component placement in printed board circuit board assembly*.: International Journal of Flexible Manufacturing Systems, 6, 5-31, 1994.

[12] F. J. Vittes, J. E. Kobza K. P. Ellis, *Optimizing the performance of a surface mount placement machine*.: IEEE Transactions on Electronic Packaging Manufacturing, 24(3), 160-170, 2001.

[13] P. Belangia J. R. Rownland, *Tecnología de Montaje Superficial Aplicada*.: Paraninfo, 1994.

[14] Link Instruments Inc. (2013, mayo) USB Logic Analyzer and Pattern Generator for Windows. [Online]. http://www.linkinstruments.com/logana32.htm

[15] Wikipedia. (2014, marzo) S-Video. [Online]. http://www.avermedia.eu/avertv/sp/Product/ProductDetail.aspx?Id=440

[16] HDD Software. (2013, mayo) Free Serial Port Monitor RS232 Serial Communication Data Sniffer Analyzer. [Online]. http://www.serial-port-monitor.com/

[17] HDD Software. (2013, mayo) Download HDD Software Free Serial Port Monitor. [Online]. http://www.serial-port-monitor.com/free-serial-port-monitor-downloads.html

[18] Processing.org. (2013, June) Processing.org. [Online]. http://processing.org/

[19] AForge.NET. (2013, mayo) AForge.NET Framework Documentation. [Online]. http://www.aforgenet.com/framework/docs/

[20] iText Group NV. (2016, abril) iText. [Online]. http://itextpdf.com

[21] Bruno Lowagie. (2016, abril) iTextpdf License. [Online]. https://github.com/itext/itextpdf/blob/master/LICENSE.md

[22] iText Group NV. (2016, abril) iText. [Online]. http://itextpdf.com/

[23] Microsoft Corporation. (2013, mayo) Lenguaje Visual C#. [Online]. http://msdn.microsoft.com/es-es/library/aa287558%28v=vs.71%29.aspx

[24] Fernando Ujaldón. (2013, mayo) Sun y Microsoft resuelven el juicio sobre Java (24/1/2001). [Online]. http://www.dealerworld.es/archive/sun-y-microsoft-resuelven-el-juicio-sobre-java

[25] Wikipedia. (2013, mayo) Kit de desarrollo de software. [Online]. http://es.wikipedia.org/wiki/Kit_de_desarrollo_de_software

[26] Microsoft Corporation. (2013, mayo) Productos de Visual Studio 2010 Express. [Online]. http://www.microsoft.com/visualstudio/esn/products/visual-studio-2010-express

[27] Altium Limited David Parker. (2008, Oct.) Pick and Place Output Options. [Online]. http://wiki.altium.com/display/ADOH/Pick+and+Place+Output+Options

[28] Aurelio Vega. (2013, mayo) IUMA SFP Pick_Place Cosy. [Online]. http://www.youtube.com/watch?v=Jrzw7n4W484

[29] Aurelio Vega. (2016, abril) IUMA SFP Pick&Place Philips CSM 84 III. [Online]. https://www.youtube.com/watch?v=bfxlF1ooxHY

[30] Aurelio Vega. (2016, abril) IUMA SFP Pick&Place MG5. [Online]. https://www.youtube.com/watch?v=fc8C_R2nk2Q

[31] IEE-SA Standards Board. (1998, septiembre) IEEE Standard for Mechanical Core Specifications for Microcomputers Using IEC 60603-2 Connectors. [Online]. https://ph-dep-ese.web.cern.ch/ph-dep-ese/crates/standards/1101_1.pdf

[32] LPKF Laser & Electronics AG. (2016, abril) ProtoPlace S. Semi-automatic SMT assembly system. [Online]. http://www.lpkfusa.com/products/pcb_prototyping/smt_assembling/protoplace_s/

[33] Wikipedia. (2013, Marzo) Interfaz de múltiples documentos. [Online]. http://es.wikipedia.org/wiki/Interfaz_de_m%C3%BAltiples_documentos

[34] Microsoft Corporation. (2013, Mayo) Configuración de la aplicación en formularios Windows Forms. [Online]. http://msdn.microsoft.com/es-es/library/0zszyc6e%28v=vs.100%29.aspx

[35] Robert C. Martin, *Código Limpio. Manual de estilo para el desarrollo ágil de software*. Madrid, España: Ed. Anaya Multimedia, 2012.

[36] Fundación Wikimedia, Inc. (2013, mayo) CSV. [Online]. http://es.wikipedia.org/wiki/CSV

[37] Janet L. Axelson, *Serial Port Complete*, 2nd ed. Madison, WI: Lakeview Research LLC, 2007.

[38] Wikipedia. (2014, marzo) Marcador de referencia. [Online]. http://es.wikipedia.org/wiki/Marcador_de_referencia

[39] Steve DiBartolomeo. (2016, abril) D-codes, Apertures & Gerber Plot Files. [Online]. http://www.artwork.com/gerber/appl2.htm

[40] Jacob Nielsen, *Usability Engineering*. EEUU: Academic Press, 1993.

[41] Wikipedia. (2014, febrero) Race Condition. [Online]. http://en.wikipedia.org/wiki/Race_condition

[42] Wikipedia. (2014, febrero) Therac-25. [Online]. http://es.wikipedia.org/wiki/Therac-25

[43] Microsoft Corporation. (2014, febrero) Cómo: Crear excepciones definidas por el usuario. [Online]. http://msdn.microsoft.com/es-es/library/87cdya3t%28v=vs.110%29.aspx

[44] Asociación TAEE. (2016, mayo) Congreso TAEE 2016. [Online]. http://www.taee2016.org/index.php/es/

[45] TAEE. (2016, mayo) Tecnología, Aprendizaje y Enseñanza de la Electrónica. [Online]. http://taee.euitt.upm.es/

[46] Microsoft Corporation. (2013, septiembre) Declarar variables de matriz. [Online]. http://msdn.microsoft.com/es-es/library/cc436629%28v=vs.71%29.aspx

[47] Microsoft Corporation. (2013, mayo) Microsoft.NET Framework 4 (instalador web). [Online]. http://www.microsoft.com/es-es/download/details.aspx?id=17851

[48] (2013, mayo) Circuito impreso. [Online]. http://es.wikipedia.org/wiki/Circuito_impreso

ACRÓNIMOS

ASCII	American Standard Code for Information Interchange
BGA	Ball Grid Array
BNC	Bayonet Neill-Concelman
BOM	Bill of Materials
CAD	Computer Aided Design
CCD	Charge-Coupled Device
CLR	Common Language Runtime
CLS	Common Language Specification
CPU	Central Processing Unit
CSV	Comma-Separated Values
CTS	Common Type System
DACIA	Dual Asynchronous Communications Interface Adapter
DCE	Data Circuit-Terminating Equipment
DIN	Deutches Institut für Normung
DIP	Dual In-line Package
DLL	Dynamic-Link Library
DTE	Data Terminal Equipment
EDA	Electronic Design Automation
EITE	Escuela de Ingeniería de Telecomunicación y Electrónica
IDE	Integrated Development Environment
IUMA	Instituto Universitario de Microelectrónica Aplicada

MDI	Multiple Document Interface
PCB	Printed Circuit Board
PCI	Placa de Circuito Impreso
PLCC	Plastic Leaded Chip Carrier
RAD	Rapid Application Development
SDI	Single Document Interface
SFP	Servicio de Fabricación de Prototipos
SMD	Surface Mount Device
SMT	Surface Mount Technology
SOIC	Small-Outline Integrated Circuit
THT	Through-Hole Technology
TTM	Time to Market
UART	Universal Asynchronous Receiver/Transmitter
ULPGC	Universidad de Las Palmas de Gran Canaria

PLIEGO DE CONDICIONES

Durante el desarrollo de este proyecto se ha utilizado principalmente un ordenador de sobremesa Acer Aspire M1610 y un ordenador portátil Acer Travelmate 5720. En ambos casos se trata de ordenadores basados en un microprocesador Intel Core 2 Duo con 4 GB de memoria RAM. En ambos ordenadores se instalaron los sistemas operativos de Microsoft Windows XP SP3 y Windows 7 SP1. En todos los sistemas bajo los que se desarrolló el proyecto se instaló el entorno de desarrollo Microsoft Visual Studio 2010 Professional. Bajo estas condiciones, el software desarrollado ha sido capaz de ejecutar las pruebas de funcionamiento correctamente.

Sin embargo, los requisitos mínimos para ejecutar la aplicación PnPWork son menores. En principio, podría bastar cualquier ordenador con el sistema operativo Windows en el que se pueda instalar el software Microsoft .NET Framework v4.0, que contiene las bibliotecas .NET, imprescindibles para que funcione correctamente cualquier programa realizado con el entorno de desarrollo mencionado anteriormente. Instalar este software es un requerimiento imprescindible, pero se puede obtener gratuitamente desde la web de Microsoft [**47**]. Para facilitar la tarea al usuario, el programa de instalación desarrollado contiene este software y lo instalará en el ordenador de destino en caso de que sea necesario.

Para ejecutar el instalador desarrollado para la aplicación, es necesario que el sistema operativo del ordenador de destino tenga al menos la versión 3.1 del sistema Windows Installer. Para facilitar la tarea al usuario, el instalador desarrollado para la aplicación contiene también este software y lo instalará en el equipo de destino en caso de que sea necesario.

Se ha estimado como requisito mínimo para ejecutar la aplicación un ordenador con un microprocesador Pentium III (o compatible), 128MB de memoria RAM y el sistema operativo Windows XP SP3. Sin embargo, para que el sistema funcione con fluidez, se recomienda al menos 512 MB de memoria RAM.

También es imprescindible contar, al menos, con un puerto serie RS-232 libre para la comunicación con la máquina pick & place Cosy. El puerto serie debe poderse configurar para la transmisión a 4800 bps con 8 bits de datos, sin paridad y un bit de stop.

Trabajando con esta máquina, el software se ha diseñado para que, durante la preparación de datos, se use en combinación casi constante con la videocámara que incorpora.

Por esto se recomienda que el ordenador incorpore una tarjeta digitalizadora de vídeo para obtener la imagen de dicha videocámara. Este no es un requisito imprescindible, pero su carencia disminuirá enormemente la facilidad de uso. Por el mismo motivo, es muy recomendable que el ordenador de control disponga de un ratón o dispositivo apuntador equivalente.

Si se desea leer en pantalla el *manual del usuario* o la *guía rápida para el montaje de placas*, el ordenador debe tener instalado un programa que permita leer documentos PDF.

La aplicación desarrollada tiene la capacidad de generar instrucciones de montaje manual en formato PDF. Si este documento se va a imprimir será necesaria una impresora. Si en vez de imprimirlo se va a utilizar mediante un dispositivo electrónico como ordenador, tablet u otro, éste tendrá que tener instalado también un programa capaz de leer dichos documentos PDF.

ANEXO I. MANUAL DEL USUARIO

En este anexo se ha incluido una copia del manual del usuario que acompaña a la aplicación desarrollada. El manual del usuario es un documento extenso donde se explica cómo iniciar la aplicación y todos los menús y ventanas de la misma.

También se explican los procedimientos para instalar la aplicación, tanto con el programa instalador creado como copiando manualmente los archivos necesarios para su ejecución, así como los requerimientos para poder ejecutarla.

En el manual del usuario se incluye una explicación de cómo realizar la edición de las propiedades de configuración de la aplicación y se explica el funcionamiento de las copias de seguridad automatizadas de la aplicación. De esta manera el usuario con experiencia podrá cambiar el comportamiento de la aplicación y recuperar los archivos sobrescritos por error.

El archivo proporcionado al usuario se puede encontrar en la carpeta de la aplicación con el nombre "*Manual del usuario.pdf*".

PnPWork

Manual del usuario

Tabla de contenidos

INTRODUCCIÓN

PnPWork es una aplicación informática que le ayudará al montaje de componentes SMD y THT. Con PnPWork podrá controlar todos los aspectos del montaje automatizado de componentes de montaje superficial utilizando la máquina Cosy SMD Flexpick, además permite generar archivos VIOS text para introducir los datos de producción en máquinas pick & place compatibles con este formato y también ayuda a la producción manual de sistemas electrónicos SMD mediante máquinas pick & place manuales y semiautomáticas o producción manual con componentes de inserción.

Con PnPWork podrá utilizar ficheros pick & place de Altium para automatizar una gran parte del trabajo de preparación previa al montaje, reduciendo así los tiempos de producción. Además, con PnPWork podrá utilizar un fichero de definición de geometrías para automatizar la conversión de ficheros pick & place de Altium o importar ficheros .dim de Cad2cad.

Si desea especificar manualmente los parámetros de colocación de componentes, podrá realizarlo desde los editores integrados, que le permitirán controlar cualquier aspecto de la misma.

PUESTA EN MARCHA DEL PROGRAMA

Antes de poner en marcha el programa deberá tenerlo instalado en su ordenador. Para conocer las opciones de instalación, consulte la sección "*Instalación de la aplicación*" más adelante.

Para iniciar PnPWork puede hacer clic en el icono instalado en el menú inicio de Windows, bajo el grupo PnPWork. También puede ejecutarlo haciendo doble clic sobre el icono instalado en el escritorio. En caso de que haya decidido copiar usted mismo los archivos del programa a una carpeta de su disco duro y no usar el instalador proporcionado, podrá poner en marcha el programa ejecutando el archivo *PnPWork.exe*.

PnPWork se integra con el entorno Windows, permitiéndole iniciar el programa haciendo doble clic sobre un archivo *.pnp* desde el explorador de Windows o arrastrándolo sobre el icono de la aplicación en el escritorio. En este caso, al iniciarse el programa, se cargarán automáticamente los datos contenidos en el archivo *.pnp*. En caso de que haya copiado usted mismo los archivos del programa, deberá realizar primero la asociación de archivos *.pnp* con el ejecutable *PnPWork.exe*, de la forma habitual en Windows.

El programa utiliza un puerto serie RS-232 para comunicarse con la máquina Cosy. Cuando se configura un puerto serie en el programa, este lo guarda en sus propiedades de configuración, de forma no tenga que volver a configurarlo en adelante cada vez que inicie la aplicación. Si no está disponible el puerto usado la última vez (o el puerto por defecto en caso de ejecutar el programa por primera vez) porque lo está usando otra aplicación, la aplicación puede mostrar una ventana de aviso como la de la figura 184 si está configurada para ello.

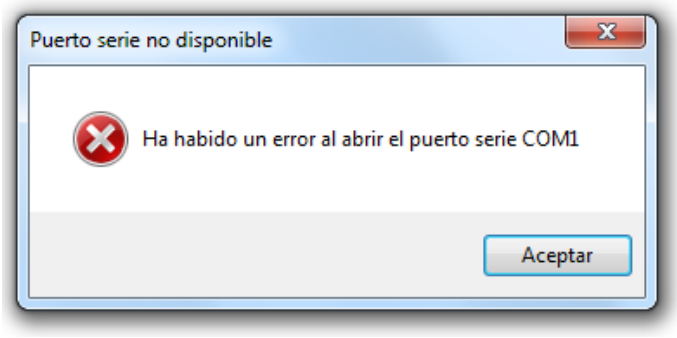


Figura 184. Ventana de aviso al intentar abrir un puerto serie en uso.

Si el puerto serie configurado en una ocasión anterior no estuviese presente (por ejemplo si es un puerto externo, conectado a través de un interfaz USB que se ha desconectado o ha

cambiado de nombre) no se mostrará la ventana anterior. En su lugar, el programa puede pedirle que indique en que puerto está conectada la máquina, antes de iniciar la ventana principal, mediante el cuadro de diálogo de la figura 185. Si en la ventana de la figura 185 se desmarca la casilla "*Comprobar al iniciar*", no se mostrará ninguna de las ventanas ni cuadros de diálogo de configuración del puerto serie mencionados en esta sección al iniciar el programa.

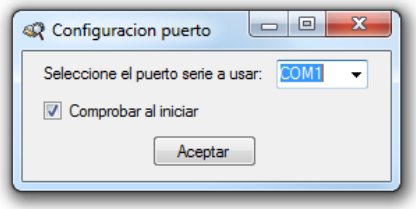


Figura 185. Ventana de *configuración del puerto serie*.

Si cierra la ventana sin seleccionar un puerto serie y sin desmarcar "*Comprobar al iniciar*", la aplicación le mostrará el mensaje de la figura 186, avisándole de que la comunicación con la máquina no será posible mientras no lo haga desde el menú *Configuración*. Sin embargo, podrá continuar utilizando el programa, aunque en las opciones en que se necesite comunicar mediante el puerto serie volverá a salir una ventana de aviso de que el puerto no está disponible.

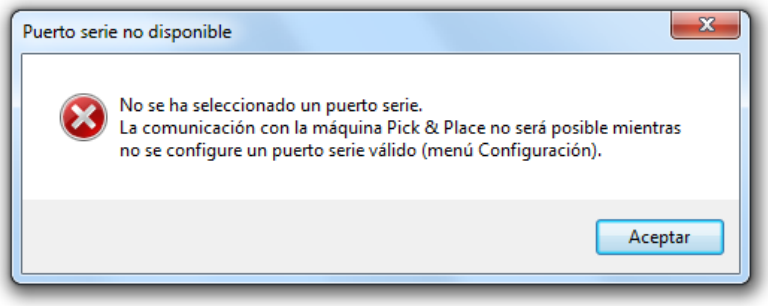


Figura 186. Ventana de aviso de falta de comunicación con la máquina.

Puede iniciar la aplicación más de una vez en el mismo equipo. En caso de que la aplicación detecte que ya se estaba ejecutando otra instancia, le podrá mostrar el cuadro de diálogo de la figura 187, en el que le avisa de que no podrá controlar la máquina pick & place Cosy desde esta segunda instancia, puesto que los recursos necesarios están ocupados.

En caso de que decida continuar, puede utilizar esta segunda instancia de la aplicación para preparar otra producción mientras la primera realiza un montaje en la máquina pick & place Cosy o realiza cualquier otra tarea. En esta ocasión no se le pedirá que configure un puerto serie, pero la aplicación mostrará los cuadros de diálogo que le avisan de su necesidad si intenta comunicar con la Cosy.

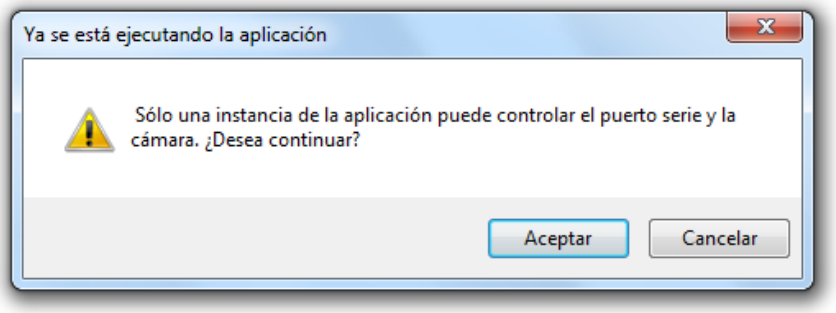


Figura 187. Cuadro de diálogo *de aviso que ya se estaba ejecutando la aplicación*.

MENÚS DE LA APLICACIÓN

Desde la ventana principal se tiene acceso al menú de la aplicación que permite lanzar las distintas ventanas y acciones posibles. Este menú tiene seis opciones que permiten lanzar las acciones agrupadas bajo los nombres: Archivo, Alimentadores, Placas, Producción, Configuración y Ayuda, como se puede ver en la figura 188.

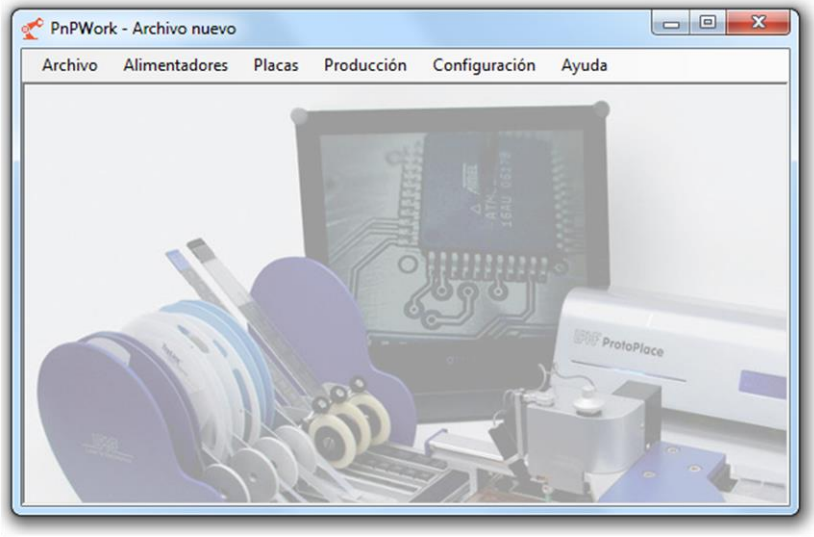


Figura 188. Ventana principal de la aplicación.

Si se pulsa la tecla ALT, se activará la barra de menús y se puede ver que cada opción aparece con una letra subrayada, como se muestra en la figura 189. Cualquier menú se puede activar desde teclado pulsando la combinación de teclas ALT + la tecla que aparece subrayada.

Una vez desplegado el menú que desea, verá que sus opciones también tienen una letra subrayada. Para activarla pulse la letra subrayada en la opción que desee. También puede usar las teclas de las flechas para mover el cursor entre los distintos menús y sus opciones. La flecha hacia abajo despliega un menú seleccionado que no esté desplegado, como el menú *Archivo* en la imagen de ejemplo. Cuando tenga activada la opción que desea, pulse la tecla *Enter* (↵ *Intro* o *Return* en algunos teclados).

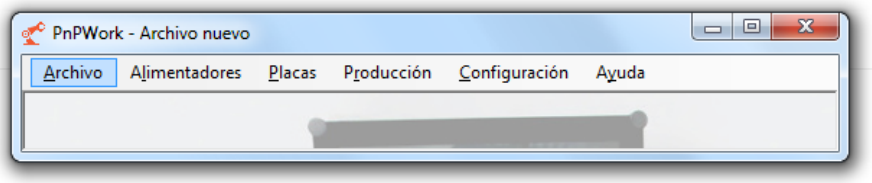


Figura 189. Menú principal mostrando las teclas rápidas.

Menú Archivo

En este menú se encuentran las opciones más comunes para cargar o guardar los datos con los que opera la aplicación. En la figura 190 puede ver este menú desplegado completamente, mostrando todas las opciones de que consta y que se describen a continuación. Como es habitual, las opciones que le dirigen a una nueva ventana se indican con puntos suspensivos y las que abren un submenú, se indican con una flecha hacia la derecha.

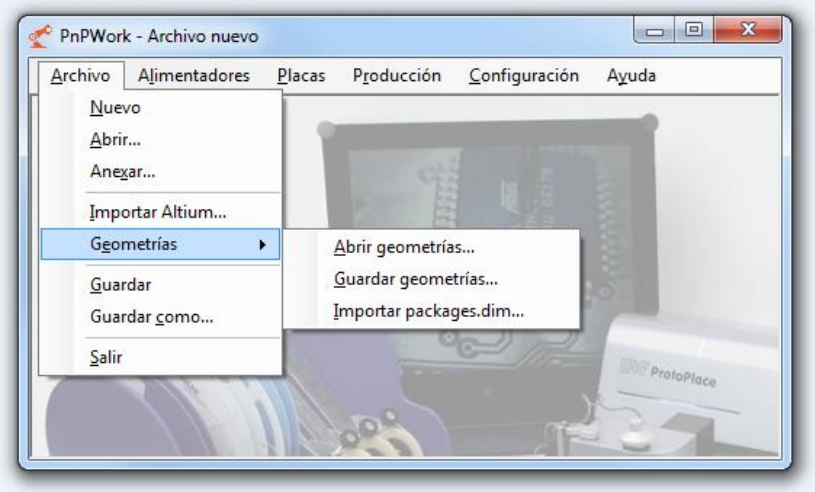


Figura 190. Menú *Archivo* desplegado.

Nuevo

Esta opción elimina todos los datos de montaje que estuviesen almacenados en memoria, tanto de los alimentadores como de las placas y sus componentes, dejando a la aplicación preparada para iniciar un nuevo trabajo.

Al poner en marcha la aplicación desde su icono se inicia ya de esta forma, por lo que en ese caso no es necesario seleccionar esta opción.

¡Atención! Esta opción borrará permanentemente todos los datos de montaje almacenados en memoria, por lo que, si desea conservarlos, deberá asegurarse de guardarlos antes en disco. En caso de que haya realizado alguna modificación no guardada de los datos de montaje, la aplicación le mostrará un cuadro de diálogo como el que se muestra en la figura 191, que le pedirá que confirme la operación de borrado.

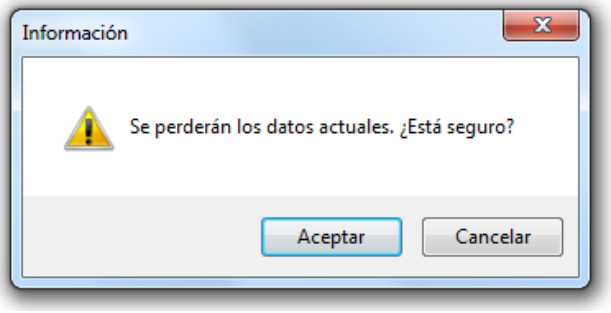


Figura 191. Cuadro de diálogo de confirmación de borrado de datos.

Abrir

Esta opción elimina todos los datos de montaje almacenados actualmente en memoria (si los hubiese) y carga un nuevo conjunto de datos de montaje desde el archivo que le especifique. Para ello muestra el cuadro de diálogo *Abrir* que se muestra en la figura 192.

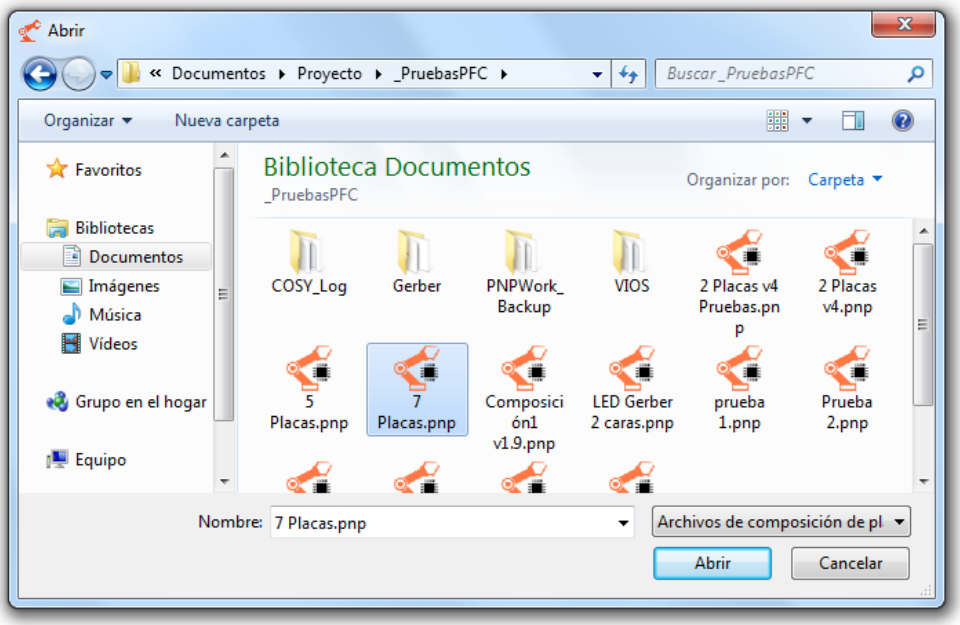


Figura 192. Cuadro de diálogo *Abrir*.

Al igual que sucedía en la opción *Nuevo*, si hay datos en memoria aún sin guardar en disco, se mostrará el cuadro de diálogo de la figura 191 para que confirme el borrado de datos.

Anexar

Esta opción le permitirá añadir datos de trabajos anteriores guardados en disco a los que están actualmente en memoria. Cuando se selecciona esta opción primero se muestra el cuadro

de diálogo de la figura 193 donde elegirá qué datos de los existentes en el archivo desea incorporar.

Puede seleccionar anexar, independientemente o en conjunto, los alimentadores y las placas. En caso de que decida anexar las placas, podrá elegir si anexar los componentes o no. Si elige anexar los componentes puede elegir entre actualizar los alimentadores en los componentes cargados (sólo si también ha marcado para anexar los alimentadores) o dejar los mismos números de alimentador que tenían en el archivo original.

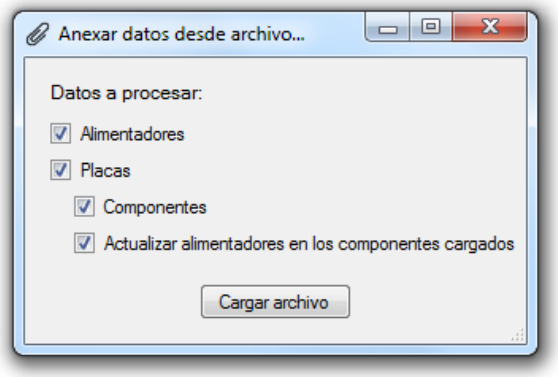


Figura 193. Cuadro de diálogo para anexar datos.

Esto último le permitirá, por ejemplo, reutilizar un conjunto de alimentadores cargados en memoria para utilizarlos en varias placas, iguales o diferentes, que cargará posteriormente. Sin embargo, debe ser cuidadoso cuando desactiva estas casillas, ya que los alimentadores deberán tener los mismos números en ambos diseños para evitar incongruencias.

Tras seleccionar los elementos que desea anexar, se abrirá un cuadro de diálogo *Abrir* como el de la figura 192 donde podrá elegir el archivo desde el que desea adquirir los datos.

Si desea crear una matriz de placas iguales, cree una placa completa con sus alimentadores y guarde el archivo. A continuación anexe el archivo guardado sobre sí mismo, seleccionando únicamente placas y componentes y dejando sin seleccionar los alimentadores. Con esto obtendrá dos placas iguales y una única definición de alimentadores. El tener los alimentadores duplicados no representa inconveniencia alguna por parte del programa, salvo por ocupar más memoria. Sin embargo, le será más conveniente no duplicarlos para poder hacer cambios en el componente que desee de todas las placas editando un único alimentador.

Si graba el archivo y nuevamente lo anexa sobre sí mismo de la misma manera, obtendrá cuatro placas. Si repite la operación obtendrá 8 placas, 16, 32 y así sucesivamente, el doble en cada ocasión. En caso de que necesite un número de placas diferente a los que puede conseguir

mediante este sistema, borre las que no sean necesarias mediante el editor de placas y componentes.

Posteriormente deberá indicar las coordenadas del origen para cada placa duplicada. Podrá hacerlo gráficamente utilizando la cámara desde el editor de placas y componentes o acelerar el trabajo haciéndolo numéricamente si están agrupadas en un panel con distancias fijas y conocidas entre placa y placa.

Importar Altium

Esta opción le permitirá importar datos desde un archivo pick & place de Altium. Al abrirse la ventana que le ayudará a procesar los datos, se muestra automáticamente el cuadro de diálogo *Abrir* con el que podrá especificar el archivo pick & place de Altium desde el que se desea importar los datos. A continuación se muestra la ventana Importar Altium, desde la que puede ver tanto el contenido del archivo de Altium original como el listado de los datos que se van a extraer.

Antes de exportar el archivo de Altium, asegúrese de que ha puesto el origen de coordenadas en la esquina superior izquierda o en la esquina inferior izquierda. El programa detectará automáticamente cuál de las dos opciones ha escogido. Además, en caso de que se vaya a montar con la Cosy, y para mayor comodidad, no olvide incluir dos marcas de referencia lo más separadas posible y que tracen una línea horizontal, pues le servirán para determinar el ángulo de giro de la placa sobre el panel con gran exactitud. La mejor posición para estas dos marcas de referencia son las esquinas superior izquierda y superior derecha de la placa.

El archivo de Altium debe exportarse en formato CSV e incluir al menos los campos *Designator, Center-X(mm), Center-Y(mm), Rotation, Layer, y Footprint*. Este último, se usa por defecto para asociar la geometría del componente a su alimentador correspondiente de forma automática. Si desea usar otro campo para asociar la geometría en lugar de *Footprint*, desde la ventana "*Importar datos desde archivos Altium*" podrá seleccionar cualquier campo para hacer la asociación.

El programa descubre automáticamente el orden en se encuentran los datos que necesita, por lo que no es necesario que los incorpore en un orden en particular. Además, todos los datos irrelevantes que se hayan incluido al generar el archivo desde Altium serán ignorados automáticamente. Si el archivo no contiene al menos las columnas de datos necesarias para importar el archivo exitosamente, el programa mostrará el cuadro de diálogo de la figura 194, avisándole para que lo genere nuevamente con todos los campos necesarios.

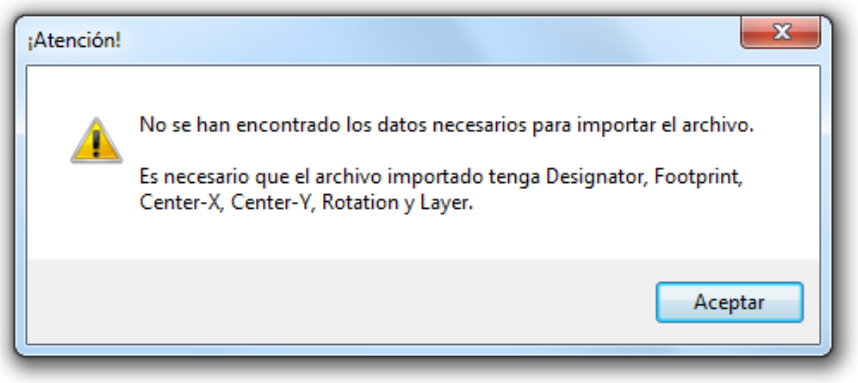


Figura 194. Cuadro de aviso de fichero Altium con datos insuficientes.

Los nombres de los campos de Altium desde donde se capturan los datos se especifican en las propiedades de configuración de la aplicación. De esta forma, si usa una versión de Altium que nombre los campos de forma diferente o usa otra aplicación de diseño, podrá configurar PnPWork para adaptarlo al cambio. Debido a esto, puede que los campos solicitados en el cuadro de diálogo sean diferentes a los mostrados en la imagen, adaptándose a los que usted haya configurado.

En caso de que se detecten líneas con datos incorrectos o insuficientes (por ejemplo, porque no se hayan rellenado los campos de Altium correspondientes en algunos componentes) el programa le avisará de los números de línea con errores con el cuadro de diálogo que se muestra en el ejemplo de la figura 195. Estas líneas no se van a incluir en la conversión, puesto que no se puede garantizar la corrección de los datos obtenidos.

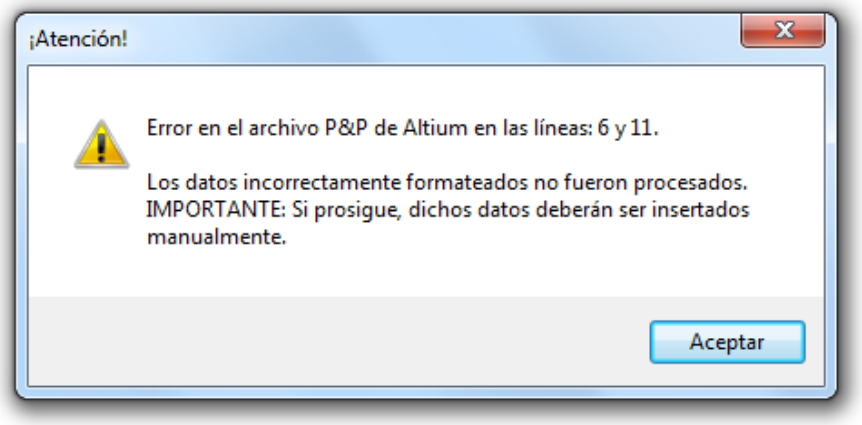


Figura 195. Cuadro de aviso de error en líneas de datos.

Por tanto, puede generar de nuevo el archivo desde Altium, incluyendo todos los datos, o ignorar el error y seguir adelante. Si sigue adelante deberá tener en cuenta las líneas con errores para incluir manualmente estos componentes desde el editor de placas y componentes. Para

ayudarle en la identificación de los errores, en la ventana de importación se muestra íntegramente el archivo de Altium, con las líneas erróneas escritas en rojo, como puede ver en el ejemplo de la figura 196.

Como se puede ver en la figura, todas las líneas del fichero de Altium son numeradas a la izquierda para facilitar la lectura del archivo. A continuación, entre paréntesis se indica el número de campos no vacíos detectado en cada línea. La línea de encabezamientos que incluye Altium se marca en verde para localizar fácilmente los nombres de los campos. El resto de líneas, que se han interpretado correctamente, aparecerán escritas en color negro.

Puede ver que, en la línea 5, el campo *Comment* del componente C299 se dejó vacío en Altium. Por otra parte, en la línea 8 se ha introducido manualmente un campo adicional. En ambos casos, el número de campos utilizables difiere del resto de líneas, con los datos correctos.

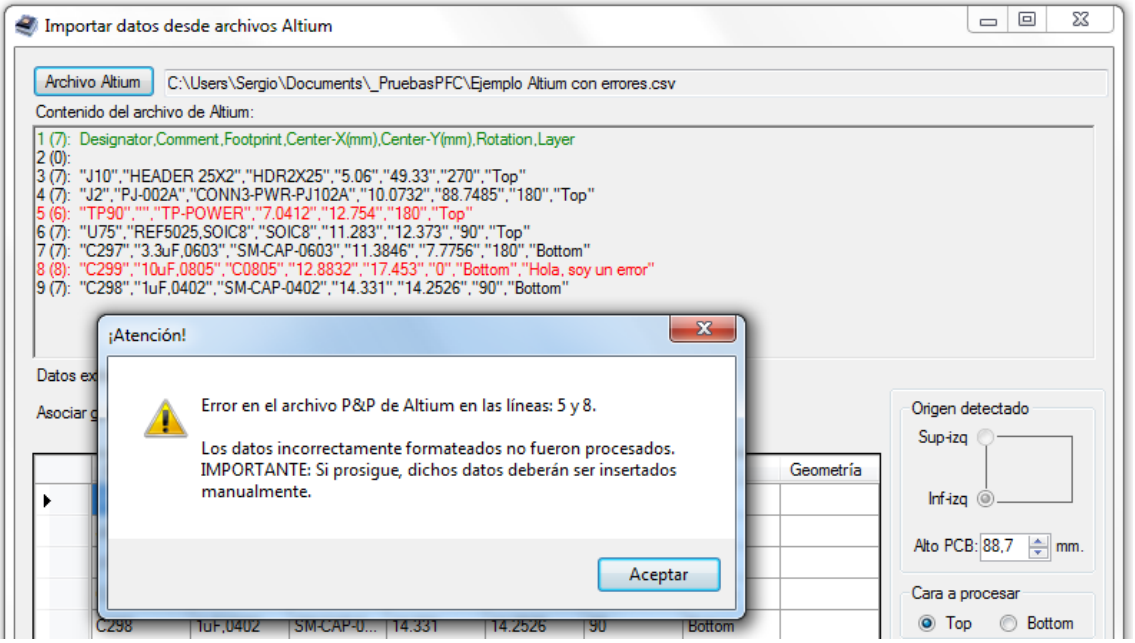


Figura 196. Marcado de líneas erróneas.

Debe tener en cuenta que la elección del punto tomado como origen en Altium es fundamental para un montaje correcto, pues todas las coordenadas de los componentes se calculan a partir de éste.

A la derecha de la ventana se muestra el origen de la placa que se ha detectado. Si ha colocado el origen de coordenadas en un punto interior de la placa, el programa lo detectará y le avisará con el cuadro de diálogo de la figura 197, para que lo corrija en Altium antes de proseguir.

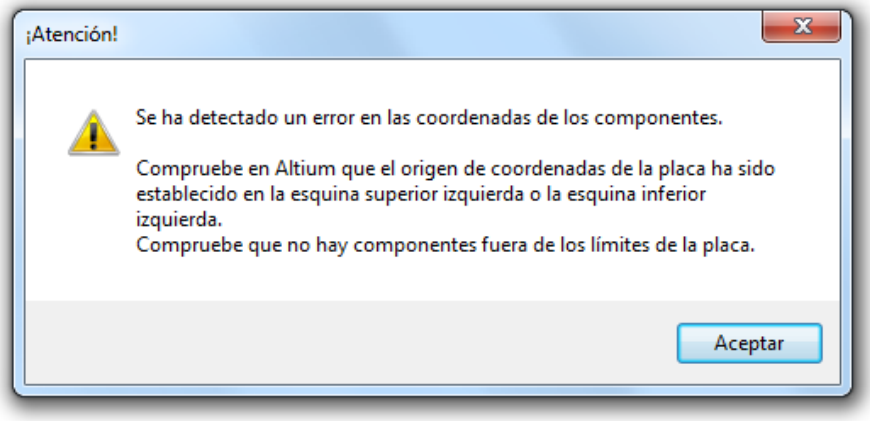


Figura 197. Cuadro de aviso de error en coordenadas en Altium.

También se mostrará el mensaje anterior si se ha dejado olvidado un componente fuera de los límites de la placa. Si el origen detectado no coincide con el origen que ha establecido en el archivo Altium, revise de nuevo el origen para asegurarse de que está en una de las dos esquinas de la izquierda.

Una vez cargado el archivo de Altium, se muestran los datos detectados en éste. En la parte inferior de la ventana se muestra el listado de componentes detectados correctamente con todas sus características, como puede observar en el ejemplo de la figura 198.

También puede ver que el programa estima automáticamente el alto mínimo que tendrá la placa. Desde esta ventana, no podrá establecer un valor inferior al que indique dicho control, puesto que este valor se determina por la posición de los componentes detectados, pero si podrá aumentarlo. Se recomienda que escriba en la casilla “Alto PCB” el tamaño exacto de la placa en el eje y, ya que este valor es fundamental para que las coordenadas del componente se establezcan correctamente.

Como alternativa, puede no escribir el alto en esta casilla, dejándola con el valor que aparece por defecto y especificar todos los parámetros de la placa siguiendo las instrucciones que se detallan más adelante en esta sección.

Inicialmente, sólo se muestran los datos de los componentes que están en la cara “Top”. Si desea trabajar con los componentes que están en la cara “*Bottom*”, pulse el botón de selección “*Bottom*” en el panel de controles que está a la derecha.

En el listado se muestran todos los datos encontrados, no solo los que son útiles para el montaje. Los datos se muestran en el mismo orden en que se encontraban en su archivo Altium,

pero puede reordenar las columnas, arrastrándolas por su encabezado, para trabajar con comodidad.

Si los datos de una casilla exceden su ancho, el texto aparecerá cortado, con unos puntos suspensivos a la derecha. Puede cambiar el ancho de las columnas para ver completos los datos demasiado largos. Para cambiar el tamaño de una columna, lleve el cursor del ratón hasta el borde entre el encabezado de una columna y su adyacente, haga clic y arrastre dicho borde. Si esto no es suficiente, también puede agrandar el tamaño de la ventana, arrastrando uno de sus bordes o maximizándola. El tamaño de la tabla se adaptará al tamaño de la ventana automáticamente, permitiéndole ver los datos completos.

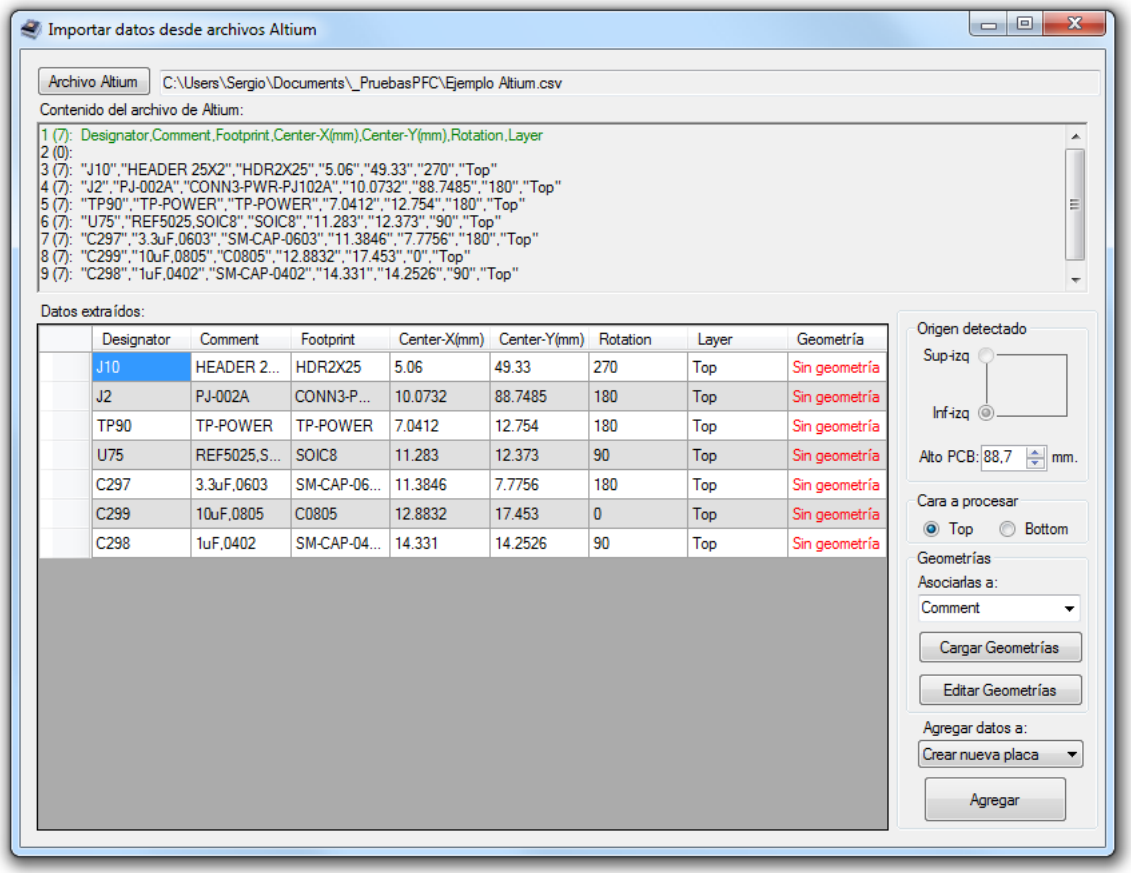


Figura 198. Ventana *Importar datos desde archivos Altium*.

Si desea reordenar las filas, puede pulsar sobre el encabezado de la columna por la que desee que queden ordenadas. Se ordenarán automáticamente por orden alfabético o de menor a mayor si son datos numéricos. Si desea el orden inverso, pulse de nuevo sobre el mismo encabezado.

La última columna de la derecha indica si hay cargada en memoria una geometría para el componente. Si la hay, mostrará su nombre en color negro, pero si no la hay, tendrá el texto "Sin geometría" en color rojo.

De forma predeterminada, las geometrías se asocian al campo "*Footprint*", pero en el panel de controles de la derecha puede cambiar la asociación. Si despliega la lista podrá asociarlas a cualquiera de los campos incluidos en el archivo Altium, como puede ver en la figura 199.

Figura 199. Detalle de la asociación de geometrías.

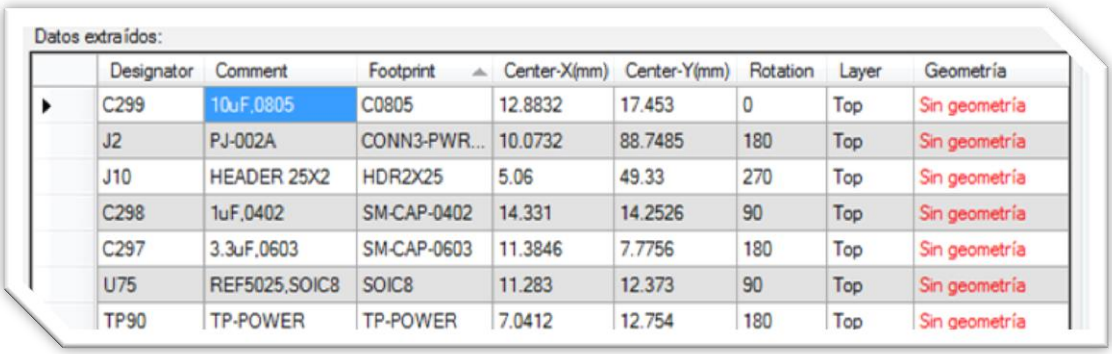
Si no están cargadas en memoria las geometrías de los componentes que va a importar, pero las tiene guardadas en un archivo, puede cargarlas en memoria mediante el botón "Cargar geometrías". En ese caso se le mostrará el cuadro de diálogo Abrir, para que elija el archivo de geometrías que desea cargar, operando de la misma manera que la opción del menú "Abrir geometrías" explicada más adelante.

Si en el menú configuración mantiene activada la opción "*Cargar/guardar geometrías automáticamente*", siempre tendrá cargadas en memoria las geometrías que use habitualmente, acelerando el trabajo al importar desde Altium.

Si en la PCB aparecen componentes nunca usados anteriormente en el programa, posiblemente no disponga de una definición de geometrías. Para crearla puede abrir el editor de geometrías pulsando el botón "Editar geometrías", que mostrará la ventana del editor de igual forma que si lo abriera desde el menú "Alimentadores".

Se recomienda tener abiertas ambas ventanas simultáneamente cuando se añadan las geometrías necesarias. Haciendo clic sobre la celdilla "*Footprint*" del componente sin geometría (o del campo que haya decidido que será el campo asociado a la geometría), esta se marcará

como muestra la figura 200. Si copia el valor (pulsando Control + C) podrá a continuación pegarlo en la celdilla “Nombre” del editor de geometrías (pulsando Control + V), que es la celdilla marcada en la figura 208. De esta manera se asegura que no comete errores al trasladas los datos que necesite.


Datos extraídos:

	Designator	Comment	Footprint	Center-X(mm)	Center-Y(mm)	Rotation	Layer	Geometría
▶	C299	10uF,0805	C0805	12.8832	17.453	0	Top	Sin geometría
	J2	PJ-002A	CONN3-PWR...	10.0732	88.7485	180	Top	Sin geometría
	J10	HEADER 25X2	HDR2X25	5.06	49.33	270	Top	Sin geometría
	C298	1uF,0402	SM-CAP-0402	14.331	14.2526	90	Top	Sin geometría
	C297	3.3uF,0603	SM-CAP-0603	11.3846	7.7756	180	Top	Sin geometría
	U75	REF5025,SOIC8	SOIC8	11.283	12.373	90	Top	Sin geometría
	TP90	TP-POWER	TP-POWER	7.0412	12.754	180	Top	Sin geometría

Figura 200. Detalle del marcado de las celdas.

Para una explicación más detallada del funcionamiento del editor de geometrías consulte el apartado “Editor de geometrías” en la sección “Menú alimentadores”, más adelante.

En el editor de geometrías, en cuanto pase a la celdilla siguiente se rellenarán las dimensiones con el valor 0. En este momento, la geometría está creada en memoria y automáticamente se actualizará el listado, cambiando el mensaje “Sin geometría” por el nombre de la geometría, que coincidirá con el nombre del campo “*Footprint*” o el que haya elegido en su lugar.

A continuación proporcione unas medidas al componente en el editor de geometrías tal como se explica en su sección correspondiente, más adelante. Repita este proceso hasta que todos los componentes que quiera montar tengan datos de geometría.

Si deja componentes sin geometría, el programa creará un alimentador genérico para ellos y se los asignará durante el proceso de conversión. Para el resto de componentes con geometría, el programa les creará un alimentador con las características especificadas en la geometría y el resto de datos asignados a los valores más comunes. Si necesita hacer correcciones a los alimentadores creados, utilice el editor de alimentadores como se explica más adelante en su sección correspondiente. Los componentes que tengan una geometría con un alimentador existente en memoria se asignarán a dicho alimentador y no se les creará un alimentador nuevo.

Una vez todos los componentes deseados tengan asociada una geometría, puede agregar la placa importada, mediante el botón “Agregar”. De manera predeterminada, se creará una

nueva placa para albergar todos los datos importados. Esta placa recibirá automáticamente el nombre “Placa Importada Altium”, pero posteriormente podrá cambiarlo desde el editor de placas y componentes, explicado más adelante.

Las dimensiones de la placa se ajustarán automáticamente para que todos los componentes queden dentro de sus límites (al menos el centro del componente), pero conviene que posteriormente edite sus características desde el editor de placas y componentes.

Si lo desea, puede agregar los componentes importados a una placa existente en memoria mediante la lista desplegable “Agregar a”, tal como muestra la figura 201. Esta es la opción recomendada, puesto que así usted podrá especificar desde el principio las características físicas de la placa creada.

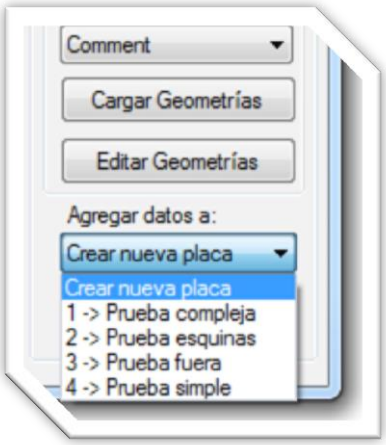


Figura 201. Detalle de la elección de placa.

Si va a usar la máquina Cosy, abra el editor de placas y componentes, coloque la placa a montar sobre el panel de la máquina pick & place y utilice la mira de la cámara integrada en ésta para determinar con precisión los detalles físicos de la placa de la forma que se explica más adelante, en la sección “Editor de placas y componentes”. Puede tener abiertas todas las ventanas simultáneamente para facilitar el trabajo.

Cuando pulse el botón “Agregar” se importarán todos los componentes a la placa elegida y pueden abrirse automáticamente los editores de alimentadores, placas y componentes, según estén configurados en el menú “Configuración”.

Abrir geometrías

Al seleccionar esta opción se muestra un cuadro de diálogo *Abrir* similar al mostrado anteriormente que le permitirá seleccionar el archivo de geometrías que desea cargar en

memoria. El cuadro de diálogo filtrará de forma predeterminada los archivos con extensión .geo, que es la extensión dada automáticamente a los archivos de geometrías del programa cuando se guardan, pero puede cambiar el filtro por “Todos os archivos” para cargar archivos con cualquier extensión.

Al cargar un archivo de geometrías desde disco se eliminarán las geometrías que existiesen previamente en memoria.

Guardar Geometrías

Al seleccionar esta opción muestra un cuadro de diálogo *Guardar como* que le permitirá seleccionar el directorio y el nombre con que va a guardar el archivo de geometrías. Los archivos de geometrías se guardan automáticamente con la extensión .geo.

Si selecciona un nombre de archivo ya existente, le preguntará si realmente desea sobrescribirlo. En caso de que haya que sobrescribir un archivo en disco, el programa creará automáticamente una copia de seguridad del archivo antiguo. Vea más adelante la sección que explica el funcionamiento de las copias de seguridad automáticas.

Importar packages.dim

Al seleccionar esta opción se muestra un cuadro de diálogo *Abrir* similar al mostrado anteriormente que le permitirá seleccionar el archivo de geometrías .dim que desea cargar en memoria. El cuadro de diálogo filtrará de forma predeterminada los archivos con extensión .dim, que es la extensión del archivo utilizado habitualmente con Cad2cad pero puede cambiar el filtro por “Todos os archivos” para cargar archivos con cualquier extensión. Como estos archivos carecen de dimensiones para el eje z, todas las dimensiones importadas tendrán esta dimensión puesta a un valor predeterminado de 2 mm, a menos que las dimensiones x o y sean inferiores a dicho valor, en cuyo caso se utilizará el valor más pequeño.

Guardar

Esta opción permite guardar todos los datos de montaje en el mismo archivo con el que se estuviera trabajando, cuya ruta se muestra en la barra de título de la aplicación, creando antes una copia de seguridad de la versión anterior. Si no se está trabajando con ningún archivo por ser la primera vez que se guardan los datos, esta opción se comportará de idéntica manera que la opción *Guardar como*.

Guardar como

Esta opción permite guardar todos los datos de montaje en disco, especificando un nombre de archivo y una ubicación donde guardar. Para ello muestra el cuadro de diálogo *Guardar Como* que se muestra en la figura 202. Como puede ver, la extensión predeterminada para los archivos guardados es .pnp, reconocibles fácilmente por su icono característico.

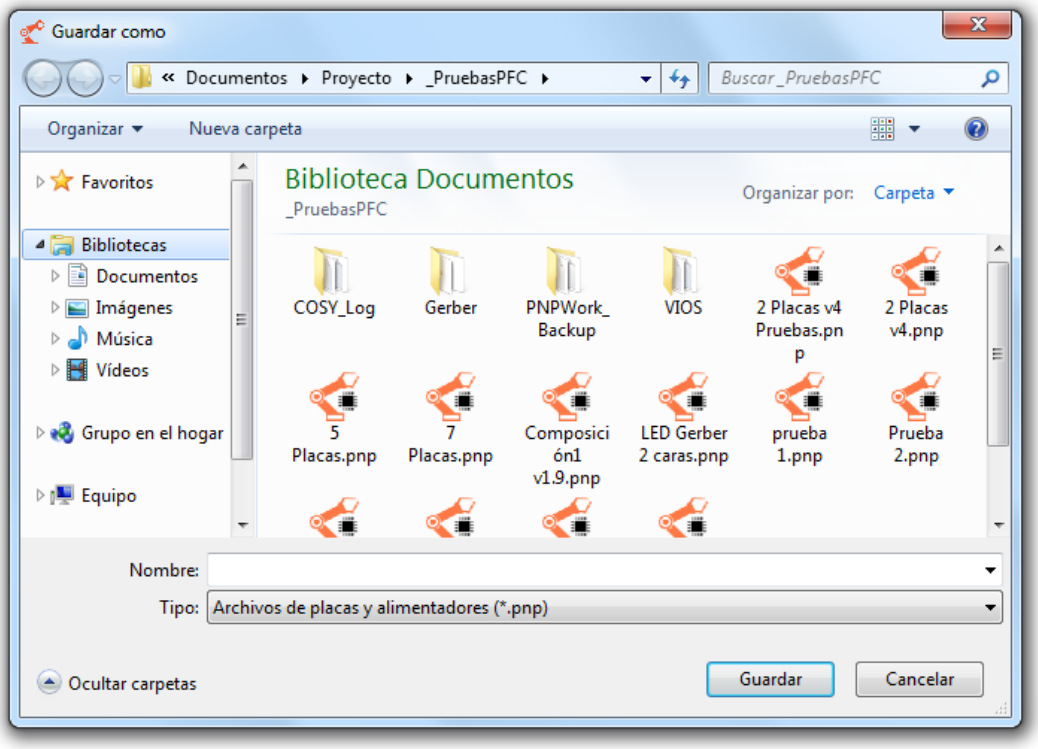


Figura 202. Cuadro de diálogo *Guardar como*.

Si se especifica un archivo ya existente, se pide confirmación al usuario para sobrescribirlo mediante el cuadro de diálogo mostrado en la figura 203.

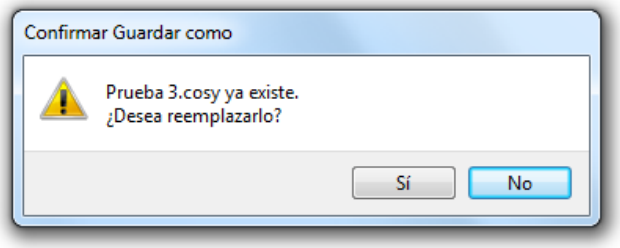


Figura 203. Cuadro de diálogo *Confirmar Guardar como*.

Si sobrescribe un archivo por error, consulte la sección dedicada a las copias de seguridad automáticas de este manual para obtener información sobre cómo recuperar el archivo sobrescrito.

Salir

Con esta opción se cierra el programa. Si hay datos no guardados se muestra el cuadro de diálogo de aviso que se muestra en la para que el usuario confirme la salida del programa sin guardar datos.

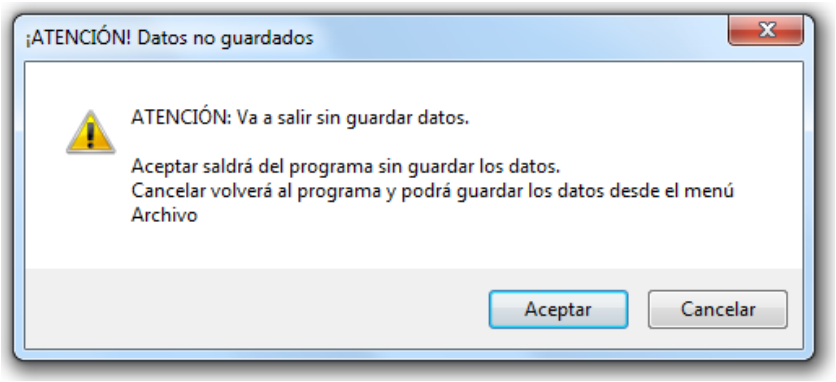


Figura 204. Cuadro de diálogo de aviso al salir de la aplicación.

Menú Alimentadores

Desde este menú puede controlar todos los aspectos relacionados con las características de los componentes implicados en el montaje. También puede establecer las dimensiones físicas de los componentes para automatizar la introducción de algunos datos durante la lectura de los ficheros de Altium.

En la figura 205 puede ver desplegado el menú Alimentadores, mostrando las opciones que contiene y que se explican a continuación.

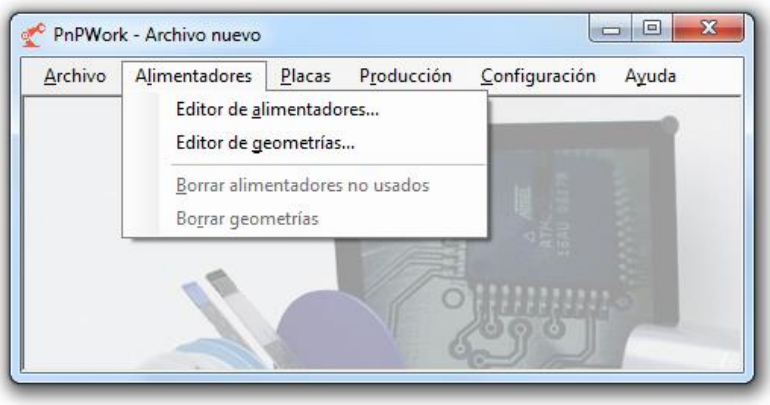


Figura 205. Menú *Alimentadores*.

Editor de alimentadores

Desde esta opción se abre la ventana del editor de alimentadores, desde donde puede editar las características de cualquier alimentador cargado en memoria, crear nuevos alimentadores o borrar alguno de los existentes.

El editor de alimentadores le permite controlar todos los parámetros de los componentes que se van a montar. En la figura 206 se muestra la ventana del editor de alimentadores con algunos componentes de ejemplo. El primer componente está seleccionado en la lista de la izquierda, mostrándose sus parámetros en la parte derecha de la ventana. Si se selecciona otro componente, a la derecha se mostrarán inmediatamente sus parámetros.

La mayor parte de los parámetros de los alimentadores sólo serán útiles en caso de utilizar la máquina pick & place Cosy. Para un uso general, sólo son absolutamente necesarios el componente de que se trata y sus dimensiones. La descripción de los parámetros que se muestran es la siguiente:

- Componente: Es un nombre para el componente que se encuentra en el alimentador. Conviene que sea ilustrativo del tipo de componente exacto de que se trate y que incluya la métrica de su encapsulado. De esta manera, cuando tenga que montarlos físicamente en el tipo de alimentador elegido (carrete, tubo, etc.) tendrá menos probabilidad de cometer un error.
- x pick, y pick: Las coordenadas donde se debe coger el componente. La máquina Cosy dispone de multitud de tipos de alimentador y dentro de estos puede haber variaciones de posición, así que la única manera fiable de asegurar que las coordenadas de captura del componente son correctas es utilizar la cámara para ver que su mira está centrada sobre el primer componente que servirá el alimentador. Para ello dispone del botón, a la derecha de las coordenadas, que tiene dibujado un punto de mira. Este botón hará que aparezca la ventana de control manual de la cámara para que realice los ajustes.
- Banco: El número de banco de trabajo de los tres posibles en la Cosy. El número 1 es el banco que está a la izquierda en el raíl de alimentadores y el número 3 es el banco de la derecha.
- Tamaño x, tamaño y: Tamaño del componente en los ejes x e y de la máquina. El eje x es paralelo al frontal de la máquina y el eje y es paralelo al lateral de la máquina.

- Útil: Número del útil con el que hay que coger el componente. El número 0 indica que se usará la boquilla desnuda. Los números del 1 al 4 montarán el útil correspondiente. El útil número 1 es el más pequeño y el útil número 4 es el más grande. Normalmente sólo se usarán con componentes de tamaño mayor, como los circuitos integrados.
- Cabezal: En esta sección se especificará la altura a la que se debe poner la boquilla para la operación descrita en cada una de las tres secciones siguientes. Se puede escribir el valor numérico en la casilla o utilizar el control deslizante que tiene debajo para especificar el valor deseado. Hay que tener en cuenta que el valor 0 representa el cabezal en su posición más alta, mientras que 200 pone el cabezal en la posición más baja a la que puede llegar. Las ilustraciones que acompañan los valores muestran los límites de la Cosy en cada caso.
    - z pick: Altura a la que se debe poner el cabezal para coger el componente. Normalmente será un valor cercano a 200 menos la altura del componente. Si no se trata de un componente extremadamente delicado, no es un valor crítico, puesto que la boquilla es retráctil, con un muelle que evita que se aplaste el componente o se dañe ésta.
    - z mover: Altura a la que se desea mover el componente. Debe ser un valor inferior a 66, como se puede ver en la ilustración que acompaña al control deslizante. Debe ser una altura suficiente para evitar tropiezos con otros componentes altos ya colocados o las estructuras de la máquina.
    - z place: Altura a la que se debe soltar el componente cuando se vaya a poner sobre la placa. Normalmente será la misma que la altura a la que se debe coger.
- Velocidad X, Y, Z y giro: Son las velocidades a las que se debe transportar el componente en cada eje y la velocidad a la que hay que rotar el componente, en caso de ser necesario. Los límites de la Cosy en cada caso son los escritos a la derecha de cada casilla.
- Gotas Dispensador: Cuántas gotas de pegamento se deben dispensar para este componente.
- Avance del alimentador: Cada vez que se coja un componente se activará el avance magnético del alimentador. Sólo es útil en aquellos alimentadores cuyo sistema de avance lo use.

- Estación de centrado: Especifica si se debe ejecutar la rutina de centrado en la estación de centrado que hay en el área de herramientas de la máquina. Normalmente es útil para circuitos integrados con patillas que se podrían doblar con las pinzas de centrado del cabezal.
- Centrado pinzas X: Especifica si se debe centrar con las pinzas justo antes de soltar el componente en su posición de colocación sobre la placa.
- Centrado pinzas Y: Especifica si se debe centrar el componente antes de ser soltado sobre la placa utilizando las pinzas del cabezal, pero en la dirección del eje Y. Como las pinzas se encuentran a ambos lados de la boquilla, en la dirección del eje x, para efectuar este centrado primero se rota el componente 90° y después de centrarlo se devuelve a su posición original.

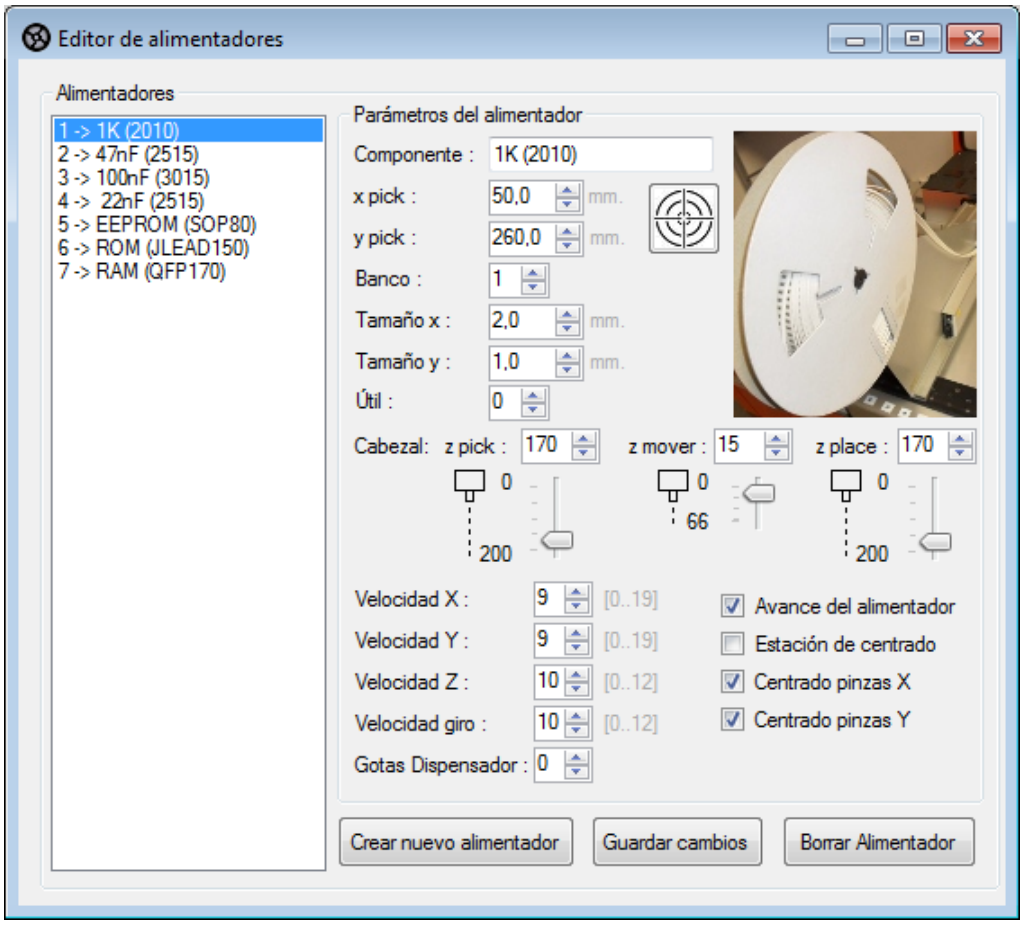


Figura 206. Ventana *Editor de alimentadores*.

Debajo de los parámetros hay 3 botones. El primero le permite crear un nuevo alimentador con los parámetros tal como se muestran. El botón central sirve para guardar los cambios realizados a los parámetros en el mismo componente que está seleccionado en el listado. El botón derecho sirve para eliminar el alimentador seleccionado en el listado.

El programa no le permitirá borrar un alimentador que esté en uso en algún componente de las placas que están en memoria, pues afectaría a la integridad de los datos almacenados. Tampoco le permitirá introducir valores erróneos en ninguno de los parámetros. Si introduce manualmente un valor numérico fuera de rango, el programa lo corregirá automáticamente en cuanto pase a rellenar otra casilla o intente guardar.

El único campo donde el programa le pedirá que corrija los errores es en la casilla *Componente*. Como puede ver en la figura 207, si introduce alguno de los caracteres no permitidos en esta casilla o la deja vacía, el programa le impedirá guardar el componente y le mostrará la casilla en rojo, indicándole así que debe corregir el error antes de guardar.

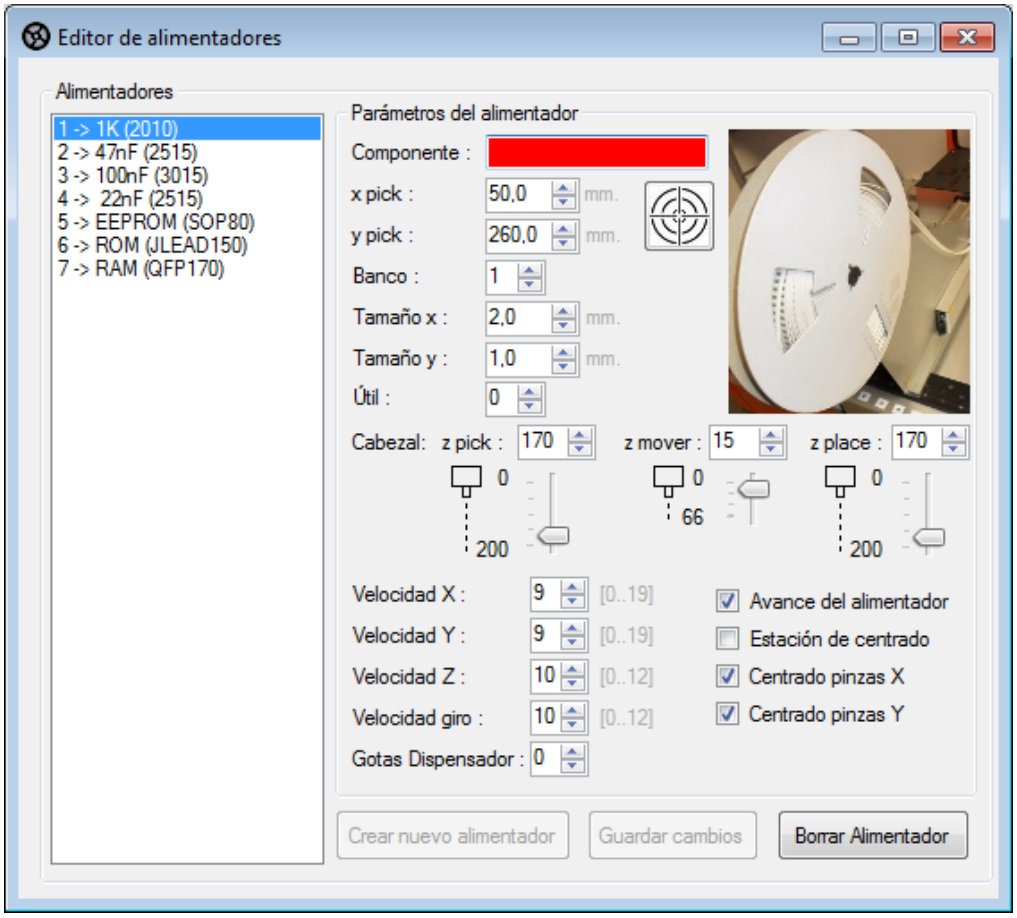


Figura 207. Ventana del editor de alimentadores con un error en el nombre del componente.

Los caracteres no permitidos tienen una función especial en los archivos que lee la aplicación y son las comillas dobles (“) y el carácter separador de campos que se haya configurado mediante las propiedades de la aplicación, por defecto el punto y coma (;).

Editor de geometrías

Desde esta opción se abre la ventana del editor de geometrías, desde la que puede ver y editar las geometrías cargadas en memoria. También puede abrir el editor de geometrías desde la ventana "*Importar datos desde archivos Altium*" a la que se accede desde el menú *Archivo* con la opción "*Importar Altium*".

El editor de geometrías le permite introducir, modificar o eliminar las dimensiones físicas de los componentes que se van a importar desde un fichero de pick & place de Altium. Estos ficheros carecen de detalles sobre las características físicas de los componentes, por lo que es necesario que el usuario las especifique.

El editor de geometrías es muy simple, como puede observar en la figura 208 y consta de una tabla de introducción de datos. En esta tabla, la columna más a la izquierda es la de indicadores y selección, siendo útil para el control de la edición. Las siguientes cuatro columnas contienen los datos y, como se puede ver, la primera columna contiene el nombre de la geometría, y las tres siguientes contienen el tamaño del componente en los ejes x, y, z de la máquina.

El tamaño en el eje x hace referencia al tamaño que tiene el componente en la dirección del eje x de la máquina **cuando está en su alimentador**, listo para ser cogido, como puede ver en el ejemplo sobre la Cosy en la figura 209. De igual manera, el tamaño en el eje y hace referencia al tamaño que tiene el componente en la dirección del eje y de la máquina cuando está en su alimentador, listo para ser cogido. Igualmente, el tamaño en el eje z es el tamaño vertical del componente (eje z de la máquina) cuando está en su alimentador.

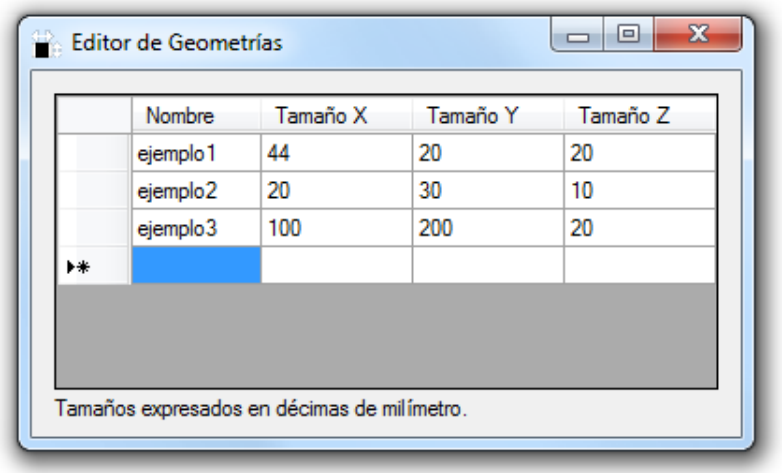


Figura 208. Ventana *Editor de geometrías*.

La casilla seleccionada se marca con un fondo azul, como se puede ver en la imagen y el ancho de las columnas cambia automáticamente para que se pueda ver el contenido. La fila que se indica con un asterisco es la que se debe usar para introducir nuevos registros (siempre será la última).

Si desea añadir un nuevo registro, seleccione la última fila, que tiene el asterisco y siempre estará en blanco y escriba directamente el contenido que necesite. Automáticamente se creará otra fila en blanco al final con el asterisco. La fila que está editando se marca con un pequeño icono de un lápiz en la columna de la izquierda, donde se muestran los indicadores. La fila donde está la casilla seleccionada se marca con una pequeña flecha apuntando hacia la derecha en la columna de indicadores.

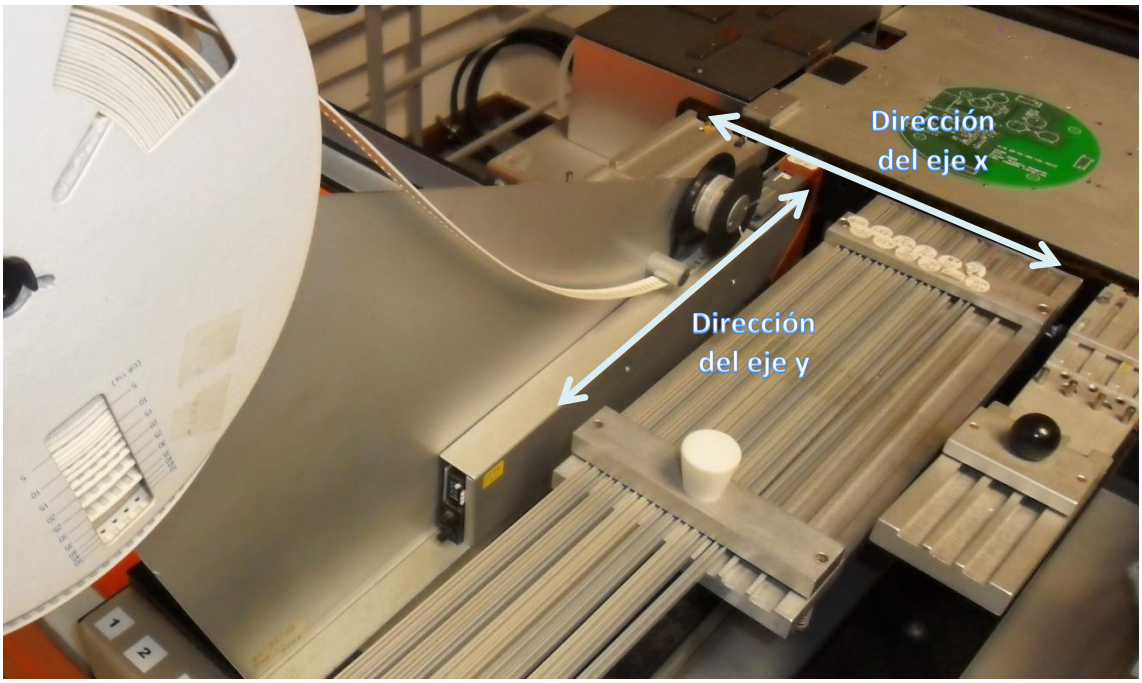


Figura 209. Direcciones de los ejes x e y en los alimentadores colocados en la máquina.

Si desea eliminar un registro, debe seleccionarlo primero. Para ello pulse con el ratón, en la columna de indicadores, en la fila del registro a borrar y toda la fila quedará marcada con el fondo azul. A continuación pulse el botón suprimir del teclado y la fila entera desaparecerá, subiendo las filas siguientes para ocupar su hueco. Si se desea cambiar un valor en una casilla cualquiera, selecciónela y escriba el nuevo valor directamente.

Tenga en cuenta que el nombre que introduzca debe coincidir con el valor del campo adecuado al importar desde Altium, tal como se explica en su sección correspondiente, puesto que es la forma que tiene el programa de saber a qué geometría corresponde cada componente.

Las dimensiones deben introducirse en décimas de milímetro. Por ejemplo: una medida de 1 mm se escribirá en la casilla como 10, mientras que una medida de 18,6 mm se escribirá en la casilla como 186.

Borrar alimentadores no usados

Esta opción comprobará qué alimentadores no están en uso en ningún componente de las placas que se encuentran en memoria y los eliminará, manteniendo los que sí están en uso. La opción sólo se activará si hay alimentadores cargados en memoria.

Borrar geometrías

Esta opción eliminará de memoria todas las geometrías existentes, pero, si se habían cargado desde un archivo, no afectará a éste. Si la opción "Cargar/guardar geometrías automáticamente" estaba marcada y no la desmarca, ni crean o carga nuevas geometrías, la próxima vez que inicie el programa, éste volverá a cargar automáticamente el mismo archivo de geometrías que estaba en uso antes de borrarlas de memoria. La opción sólo se activará si hay geometrías cargadas en memoria.

Menú Placas

Desde este menú puede ver y editar todas las características de las placas que hay sobre el panel. También se pueden editar las características de los componentes numérica o gráficamente. Además puede reordenar automáticamente los componentes de las placas. Si dispone de archivos Gerber de la placa se podrán asociar a ésta para verla en el editor gráfico. En la figura 210 puede ver el menú *Placas* desplegado, mostrando las opciones que contiene.

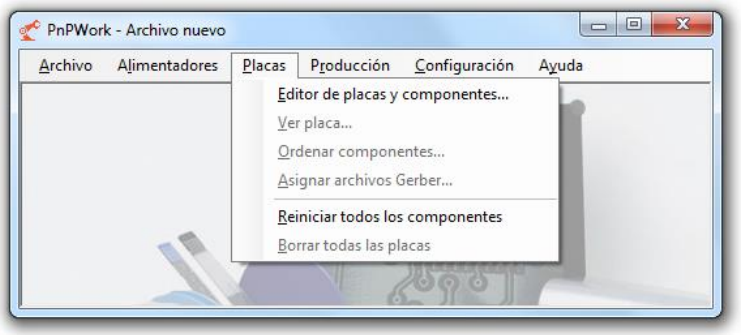


Figura 210. Menú *Placas*.

Editor de placas y componentes

Esta opción mostrará la ventana del editor de placas y componentes, desde el que podrá modificar las características de cualquier placa o componente cargado en memoria. También desde esta ventana podrá añadir o borrar manualmente placas y componentes e iniciar el editor gráfico que le permitirá ver una representación gráfica de los componentes sobre la placa.

El editor de placas y componentes le permite controlar los parámetros de las placas y componentes que son necesarios para una correcta colocación de éstos. En la figura 211 se muestra la ventana del editor de placas y componentes con algunos elementos de ejemplo.

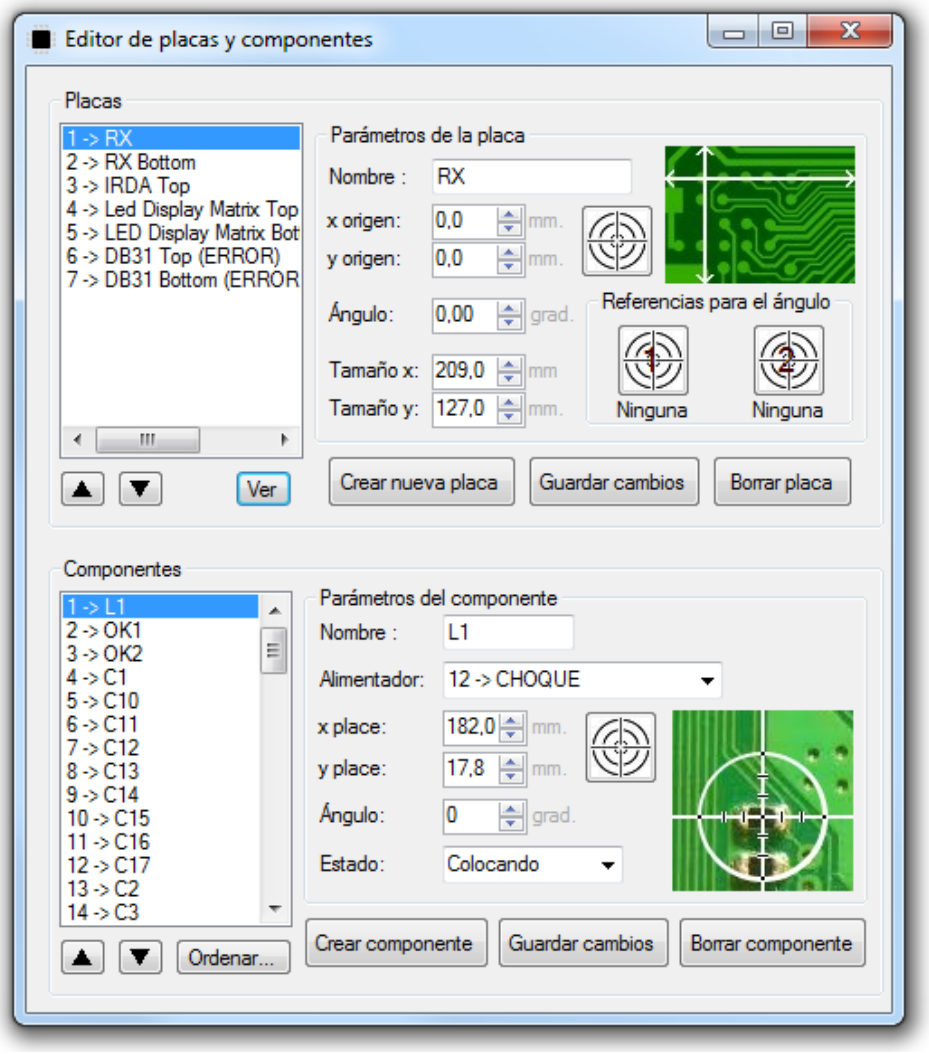


Figura 211. Ventana *Editor de placas y componentes*.

En la parte superior, en la lista de la izquierda se pueden ver todas las placas que están cargadas en memoria actualmente. En el ejemplo de la figura hay cargadas 7 placas, estando seleccionada la primera de ellas. Los parámetros de la placa seleccionada se muestran a la

derecha del listado. Debajo se muestra un listado con todos los componentes de la placa seleccionada y dentro de esta lista está seleccionado el primero, cuyos parámetros aparecen a la derecha de la lista.

Si selecciona otra placa, se mostrarán a la derecha sus parámetros y debajo su listado de componentes, pasando a verse los parámetros del primer componente en la placa seleccionada.

Los parámetros de la placa tienen el siguiente significado:

- Nombre: El nombre dado a la placa para identificarla más fácilmente.
- x, y origen: Coordenadas del origen de la placa sobre el panel de la Cosy. Corresponden a la esquina superior izquierda de la placa. Las coordenadas de todos los componentes de la placa se especifican de forma relativa a este origen. Aunque se pueden especificar numéricamente es mucho más fiable utilizar la cámara de la máquina. Ponga una marca de referencia en la esquina superior izquierda mientras diseña su PCB y utilice la cámara para localizarla y centrarla en la mira. De esta forma el programa introducirá las coordenadas automáticamente. Para mostrar la ventana del control manual de la cámara pulse el botón que está a la derecha de las casillas de las coordenadas (con el dibujo de una mira).
- Ángulo: El ángulo de giro de la placa con respecto al eje x de la máquina. El ángulo se debe especificar en grados centesimales (gradianes) con hasta dos decimales de precisión. Aunque se puede especificar numéricamente, es mucho más fiable utilizar la cámara de la máquina para que el programa calcule el ángulo automáticamente como se explica más adelante.
- Tamaño x, y: Las dimensiones x e y de la placa.

Debajo de las casillas de los parámetros hay 3 botones. El de la izquierda le permite crear una nueva placa con los parámetros mostrados que se añadirá al final del listado de placas. El botón central le permitirá guardar los cambios realizados en los parámetros de la placa que está seleccionada en el listado. El botón de la derecha le permitirá borrar la placa seleccionada en el listado. Tenga en cuenta que al eliminar una placa se eliminan también todos los componentes que contiene.

Debajo del listado hay tres botones. Los dos primeros le permiten cambiar el orden de las placas en el listado, subiendo o bajando la placa seleccionada. El orden de las placas en este listado será el orden en que se monten al iniciar la producción. El tercer botón le permite ver la distribución de componentes sobre la placa, abriendo una ventana del visor/editor gráfico para

mostrarla. De esta forma podrá comprobar visualmente si todo parece estar bien o hacer cambios en los componentes moviéndolos, rotándolos o cambiándolos de estado con el ratón.

Cuando se desea que el programa calcule el ángulo de giro de la placa se deben tomar dos puntos de referencia. Para simplificar la tarea, estos dos puntos de referencia deben estar en la misma línea horizontal en el diseño original de la PCB (o sea, ambos con el mismo valor de coordenada y). Obtendrá una mayor precisión en el cálculo del ángulo si los dos puntos de referencia están muy separados. Cuando coloque la placa sobre el panel de la máquina, en un ángulo arbitrario, utilice estos dos puntos de referencia para que el programa calcule automáticamente el ángulo que se ha girado la placa.

Para ello pulse el botón de la mira con un 1 en el centro y obtenga las coordenadas de la referencia que en el diseño de la PCB estaba a la izquierda. Bajo el botón se mostrarán las coordenadas de la primera referencia donde antes ponía "*Ninguna*". A continuación pulse el botón de la mira marcado con un 2 en el centro y apunte con la cámara a la referencia que en el diseño original de la PCB se encontraba a la derecha. Bajo el botón se mostrarán las coordenadas de la segunda referencia y la casilla del ángulo mostrará el ángulo de giro de la placa ya calculado.

Aunque puede escoger dos puntos cualesquiera de su PCB que formen una línea horizontal, la mejor forma de especificar el ángulo es dejar dos marcas de referencia en el diseño de la PCB. El orden en que se indican las referencias es irrelevante, así que si lo desea puede hacerlo primero con la derecha (usando el botón de la mira marcado con un 2) y luego la izquierda (usando el botón de la mira marcado con un 1). Lo único que importa es que la posición de la marca de referencia número 1 esté más a la izquierda que la 2 en el diseño original de la PCB.

Para simplificar y no tener que dejar tres marcas de referencia sobre el diseño de la PCB puede usar la marca del origen como primera marca de referencia para el ángulo. En estas condiciones, la posición óptima de la segunda marca de referencia será la esquina superior derecha de la PCB en el diseño original, pues ofrecerá la máxima distancia de separación y, por tanto, la mayor precisión posible para el cálculo del ángulo.

Para una mayor precisión en la colocación de los componentes, se aconseja que utilice siempre este método, incluso para las placas que se colocarán sin girar (supuestamente a 0°). Si se hace con cuidado se podrá revelar la existencia de giros inapreciables a la vista de hasta dos centésimas de gradián.

La casilla del nombre no debe quedar vacía ni incluir caracteres no permitidos. En caso de que se detecte un error en esta casilla se deshabilitarán los botones que le permiten guardar los datos, como puede apreciar en la figura 212.

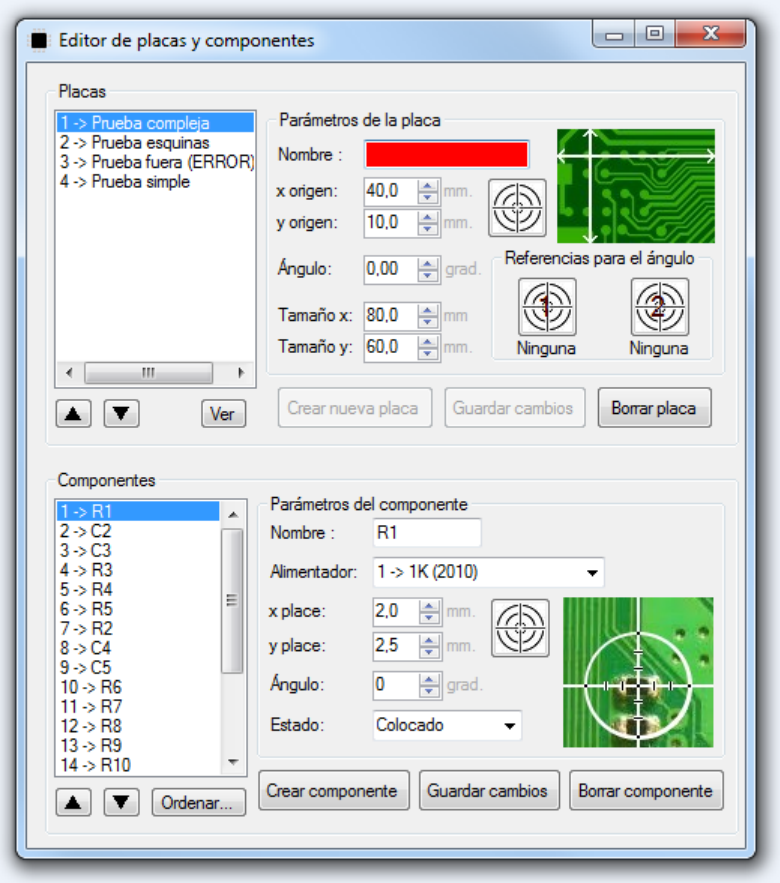


Figura 212. Ejemplo de placa con un error en el nombre.

Ninguna de las casillas numéricas permite la introducción de datos incorrectos ni dejarlas vacías. Si intenta sobrepasar los límites numéricos, el programa corregirá automáticamente los valores a los límites preestablecidos. El límite máximo siempre es el tamaño total de la zona de trabajo de la boquilla de la máquina: 390 mm para las coordenadas o dimensiones horizontales y 261 mm para las coordenadas o dimensiones verticales si está usando los límites de la Cosy (son los predeterminados al instalar el programa). Si desea cambiar los límites del área de trabajo, puede hacerlos desde el menú configuración.

Sin embargo, sí es posible introducir combinaciones de datos que den lugar a situaciones imposibles. Por ejemplo, si introduce una placa cuyo origen está en las coordenadas (300.0, 10.0), y por tanto dentro de los límites, podrá especificar que las dimensiones sean (100.0, 60.0). Esto hará que la suma de las coordenadas de origen y las dimensiones haga salir partes de la placa fuera de los límites de trabajo de la máquina, en este ejemplo, por la derecha.

Se ha permitido explícitamente que guarde datos numéricos con anomalías de esta clase con el fin de flexibilizar la introducción de datos mientras esté haciendo ajustes en la posición y ángulo de las placas sobre el panel, pero, tendrá que corregir cualquier situación anómala en los datos antes de que el programa le permita iniciar el montaje con la Cosy (con el resto de opciones si podrá). Si guarda datos numéricos erróneos el programa se lo indicará marcándolos en rojo y escribiendo "(ERROR)" a la derecha del nombre de la placa en el listado. Además, un mensaje en rojo sobre los parámetros le avisa claramente de que la placa contiene errores. En la figura 213 puede ver un ejemplo del aspecto de la ventana cuando se guarda intencionadamente un error.

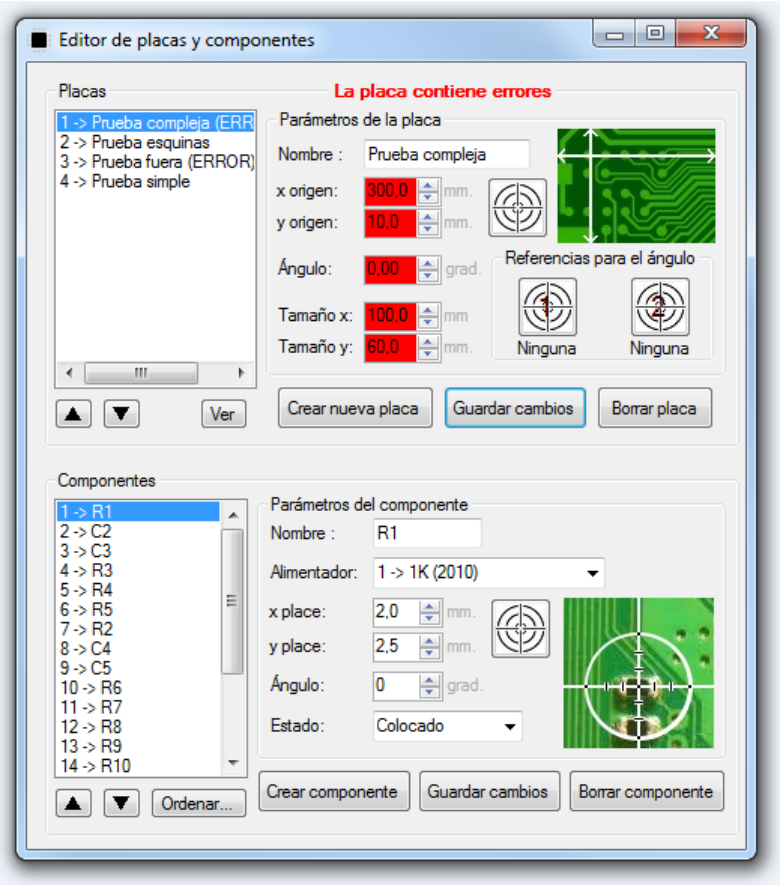


Figura 213. Ejemplo de placa con errores en los datos numéricos.

De esta forma, un error en la placa seleccionada nunca podrá pasar desapercibido. Si el error está en una placa no seleccionada también podrá darse cuenta de la condición de error observando su nombre en el listado de placas. Como puede apreciar en la figura 211, se indica claramente que existe un error en la placa número 3, aunque en la imagen está seleccionada la placa número 1 que no contiene errores.

En la parte inferior de la ventana de la figura 211 está el editor de componentes. Desde esta ventana sólo se pueden editar los componentes de la placa que está seleccionada. Si desea

editar los componentes de una placa diferente a la que está viendo, primero tendrá que seleccionarla en el listado de placas. El funcionamiento del editor de componentes es prácticamente el mismo que en el caso del editor de placas.

Cuando selecciona un componente en el listado de la izquierda, a la derecha se muestran sus parámetros. El significado de los parámetros de los componentes es el siguiente:

- Nombre: El nombre que se le ha dado al componente en el diseño.
- Alimentador: Es un cuadro desplegable que contiene todos los alimentadores que hay cargados en memoria. Se muestra el número del alimentador y el nombre del componente que contiene. Normalmente el nombre del componente debería ser el valor si es pasivo o el código de referencia y además la métrica o el encapsulado. De esta forma disminuirá la probabilidad de cometer errores.
- x, y place: Coordenadas donde se va a colocar el componente sobre la placa. Son coordenadas relativas al origen de la placa, que por defecto está en la esquina superior izquierda de ésta. La coordenada x crece hacia la derecha y la coordenada y crece hacia abajo. La forma más fiable para la introducción de coordenadas cuando se está creando los datos de montaje de forma manual es utilizar la cámara utilizando el botón con la mira que se encuentra a la derecha de las casillas de las coordenadas.
- Ángulo: El ángulo de rotación del componente, expresado en grados centesimales (gradianes) enteros. El límite de giro son 399 gradianes, puesto que 400 gradianes equivalen a una vuelta completa.
- Estado: Refleja el estado en que está el componente. Se puede seleccionar entre los valores posibles mediante el desplegable, pero normalmente debería dejarlo en estado "*No tratado*", a menos que tenga una buena razón para cambiarlo. Durante el montaje de los componentes el programa va cambiando el estado para que vaya reflejando en qué parte del proceso se encuentra. El componente terminará la sesión de montaje en estado "*Colocado*" si todo ha ido bien o en uno de los estados de error. Si no se ordena inicializar el estado de los componentes, el comando de producción ignorará los componentes que ya estén en estado "*Colocado*".

Bajo los parámetros se encuentran tres botones, Al igual que en caso del editor de placas, el primero sirve para crear un nuevo componente con los parámetros mostrados, el segundo

guarda el valor de los parámetros en el componente seleccionado y el tercero borra el componente seleccionado.

Al crear un nuevo componente se agrega al final del listado, pero se puede cambiar el orden moviendo el componente seleccionado arriba o abajo con los botones que se encuentran bajo el listado de componentes u ordenarlos automáticamente con el botón “ordenar”. El orden en que estén en este listado será el orden en que se monten cuando se inicie la producción.

Si la placa está girada, no se puede usar la mira de la cámara para determinar las coordenadas de colocación de un componente, por lo que el botón de la mira se desactivará. Si desea crear manualmente los componentes de una placa, hágalo con la placa sin girar y establezca las coordenadas del origen de la placa en esta posición temporal. Cuando termine de posicionar todos los componentes ayudándose de la mira, coloque la placa en su ubicación y ángulo de giro definitivos para el montaje y establezca de nuevo origen y ángulo de la placa.

Al igual que sucede con el resto de editores, la casilla del nombre no puede quedar vacía ni incluir ninguno de los caracteres no permitidos. En ese caso se desactivan los botones que permiten guardar, tal como se muestra en la figura 214.

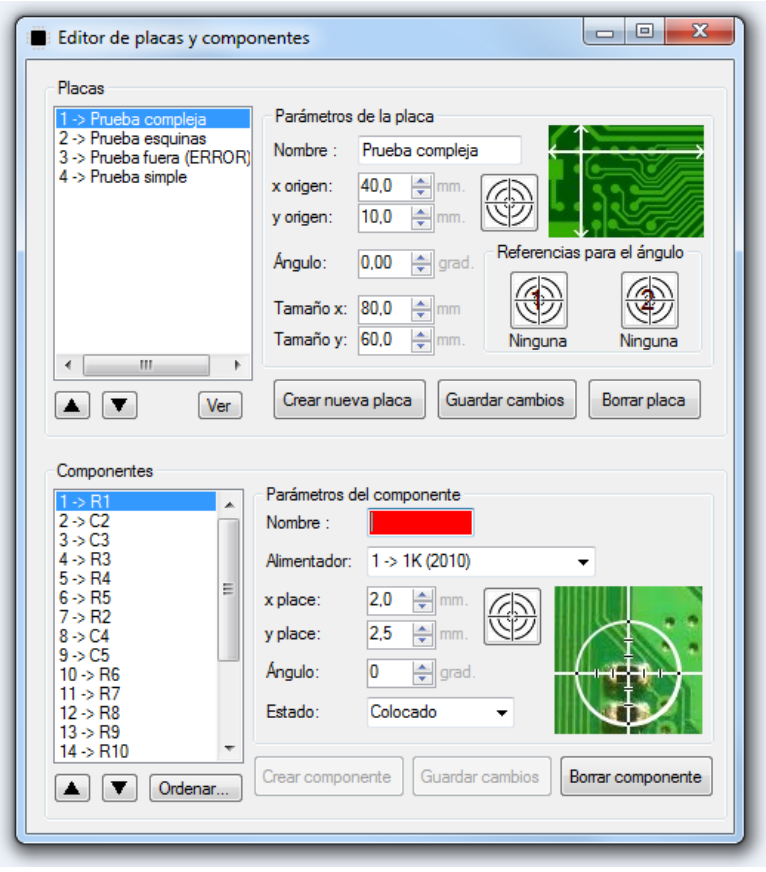


Figura 214. Ejemplo de componente con un error en el nombre.

De la misma manera que ocurría en el editor de placas, se permite introducir y guardar datos que hagan salir un componente fuera de los límites de su placa (recordemos que las dimensiones de un componente se especifican en su alimentador correspondiente). En caso de que se hayan guardado datos que lleven al componente a salirse de los límites de su placa se indicará señalando en rojo los datos numéricos de colocación, añadiendo el texto "(ERROR)" a la derecha de su nombre en el listado de componentes y con un mensaje en rojo sobre los parámetros del componente.

Además, puesto que la placa contiene un componente erróneo, el programa indicará que ella también es errónea, utilizando para ello los mensajes de error propios de la placa, mencionados anteriormente al hablar del editor de placas. De esta forma, un error en un componente no puede quedar oculto por el hecho de no estar seleccionada su placa en el listado de placas.

En la figura 215 se puede ver el aspecto del editor mostrando un componente con un error numérico guardado.

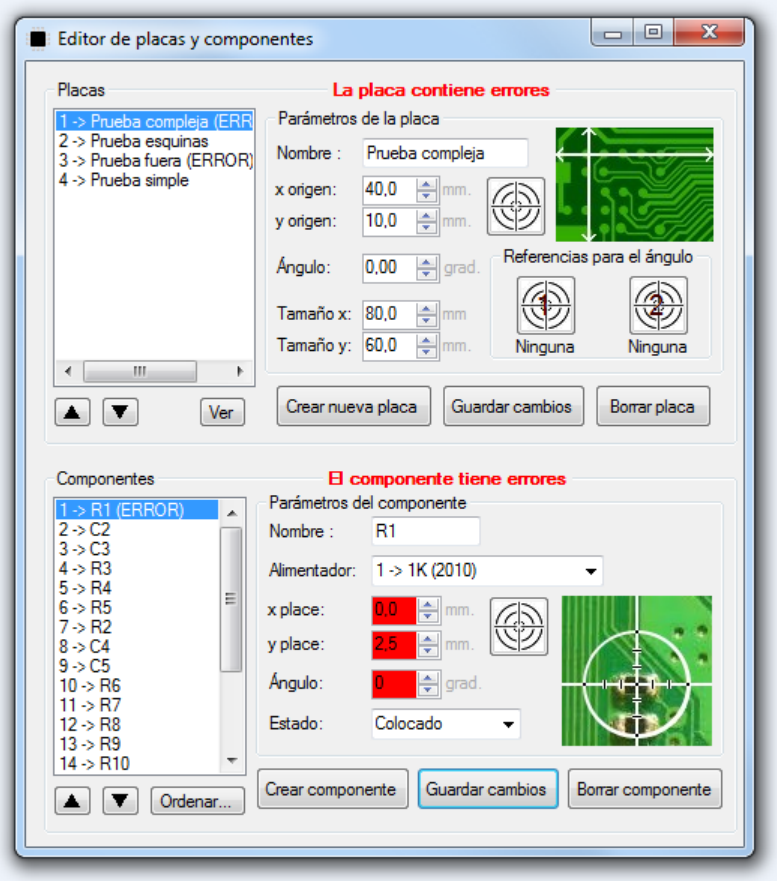


Figura 215. Ejemplo de un componente con errores en los datos de colocación.

Ver placa

Desde esta opción se puede ver la distribución de los componentes sobre la superficie de la placa que se elija, mostrando una representación gráfica de éstos, delimitada por unas escalas de medida que muestran la distancia hasta el origen de la placa.

Esta opción sólo está activa tras cargar o crear alguna placa en memoria. Se trata de un submenú dinámico, que muestra una opción por cada placa alojada en memoria. En la figura 216 puede ver un ejemplo en el que se muestra cómo han aparecido cuatro opciones en este submenú al cargar un archivo que contenía estas siete placas de prueba. Junto al número de orden de la placa aparece su nombre para identificarla más fácilmente.

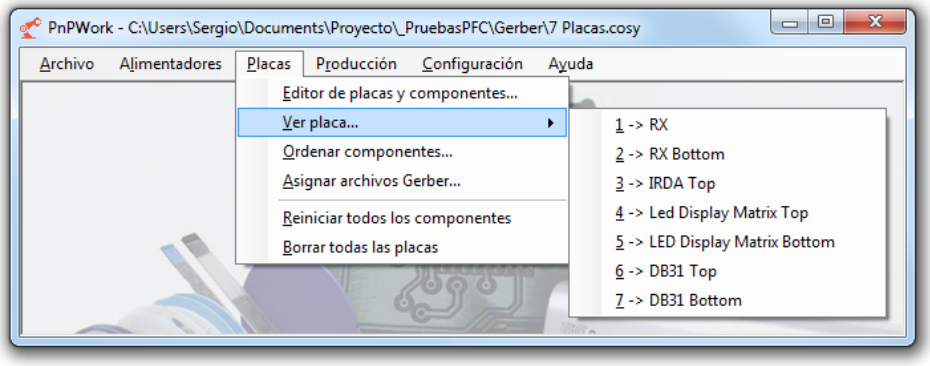


Figura 216. Menú *Placas* desplegado, con algunas placas de ejemplo.

Como se trata de un submenú dinámico, el número de opciones crecerá con el número de placas alojadas en memoria. En caso de que el submenú crezca hasta superar el tamaño vertical del monitor, aparecerán unas flechas en la parte superior e inferior del submenú que le permitirán desplazar la lista de opciones (scroll) hasta encontrar la placa que desea ver.

Visor/Editor gráfico

El visor gráfico de placas le permitirá ver rápidamente la distribución de componentes sobre la placa, su tamaño y el estado en que se encuentran. Además, podrá trasladar los componentes, rotarlos y cambiarlos de estado desde esta ventana. Puede abrir un visor/editor gráfico por cada placa existente en memoria, con lo que podrá ver varias placas simultáneamente.

En la figura 217 se muestra una placa de ejemplo en el visor/editor gráfico que muestra componentes de diverso tipo y con estados diferentes. En la barra de título de la ventana se muestra el número de la placa, su nombre y sus dimensiones. En la barra de estado de la ventana se ven las coordenadas de la placa sobre las que se encuentra el ratón. Estas coordenadas están

expresadas en milímetros. Las barras de desplazamiento aparecerán únicamente si las dimensiones de la placa son mayores que la parte mostrada en la ventana.

En la ventana siempre aparecerán reglas graduadas en los cuatro bordes para ayudarle a orientarse y saber qué parte de la placa está viendo, en caso de que no quepa entera en la ventana. Las reglas están graduadas en milímetros, con una marca numérica cada 5 milímetros. Cuando arrastre la placa para ver partes ocultas, las reglas la seguirán, de forma que siempre muestran la ubicación exacta de los componentes.

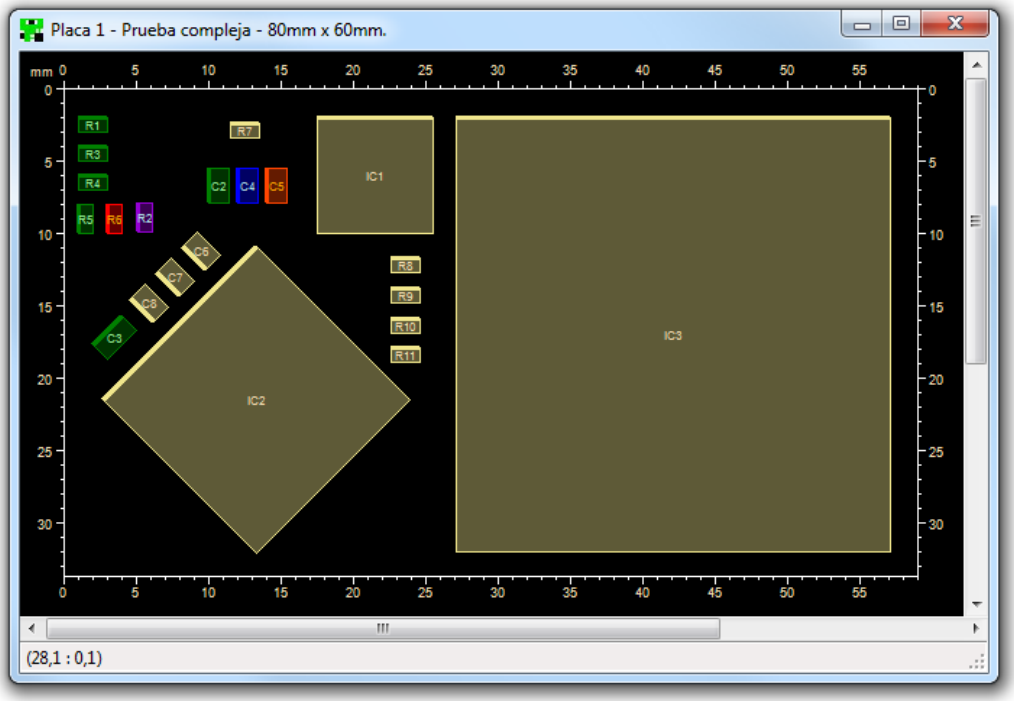


Figura 217. Ventana del visor/editor gráfico con una placa de ejemplo.

Cada componente se muestra como un rectángulo que tiene el ancho y alto especificado en su alimentador correspondiente. Las coordenadas de su centro y el ángulo de giro es el que se especifica en sus propiedades de montaje, visibles también en el editor de placas y componentes. En el centro del componente está el nombre especificado en sus propiedades de montaje. El lado norte del componente se muestra con una línea de borde más gruesa.

Los colores del rectángulo que representa al componente y del texto de su nombre dependen del estado en que se encuentra el componente, visible también en sus propiedades de montaje. Algunos colores sólo se usan cuando se está realizando el montaje mediante la Cosy. En los componentes mostrados en la figura 217 se han editado sus parámetros, mediante el editor de placas y componentes, para mostrar todas las posibilidades artificialmente, aunque en el uso cotidiano de la aplicación nunca se verán todos simultáneamente.

La correspondencia entre colores y el estado de los componentes es la siguiente:

- Amarillo claro: El componente aún no se ha tratado.
- Azul: El componente se está cogiendo o se están haciendo los preparativos previos para hacerlo.
- Violeta: El componente ha sido cogido del alimentador con éxito y se está colocando o se están haciendo los preparativos previos para hacerlo.
- Verde: El componente han sido colocado correctamente.
- Rojo: El componente ha fallado y, tras un número de reintentos fallidos, ha dado error al cogerlo y se desistió de tratarlo.
- Naranja: El componente ha fallado y, tras un número de reintentos fallidos, ha dado error al colocarlo y se desistió de tratarlo.

Lo normal es que en una placa sin procesar todos los componentes se muestren de color amarillo claro. En una placa procesada sin errores todos los componentes serán verdes. Si la placa procesada tuvo errores, además del verde se pueden ver componentes de color rojo o naranja. Si se ha detenido el programa en mitad del proceso puede aparecer el componente que se procesaba en cualquier color. La máquina Cosy procesa los componentes de uno en uno.

Podrá desplazar la placa en la ventana utilizando las barras de desplazamiento, de la forma habitual en el entorno de Windows, o también pulsando sobre la placa que se muestra en la ventana y arrastrándola con el ratón. La acción de arrastrar con el ratón sirve tanto para desplazar un componente como para desplazar la vista sobre la placa, dependiendo de si al arrastrar se hace clic sobre el nombre de un componente o sobre cualquier otra parte del contenido de la ventana respectivamente.

Para ayudarle a evitar errores, cuando el cursor del ratón está encima del nombre de un componente, su imagen cambia a una mano (normalmente es una flecha). De esta manera le indica que arrastrándolo va a alterar las propiedades de dicho componente. Mientras está desplazando la placa, el cursor del ratón normal cambia por otro con cuatro flechas hacia arriba, abajo, izquierda y derecha, que indica que está moviendo la vista de la placa en la ventana y no un componente.

Además, puede desplazar la vista utilizando la rueda del ratón, que en esta ventana pasa a tener triple función. Si gira la rueda del ratón estando el cursor sobre el nombre de un componente (cursor mano), girará el componente. Si la gira sobre cualquier otra parte de la placa, se desplazará la placa verticalmente, pero si no hay barra de desplazamiento vertical porque el

alto de la placa se ve por completo en la ventana, entonces la rueda pasará automáticamente a utilizarse para el desplazamiento horizontal.

Cuando se usa el editor gráfico de componentes para mover o girar un componente, los cambios se reflejan inmediatamente en el editor de placas y componentes, si está abierto. En ese caso se actualizan los datos escritos en las casillas de posición y ángulo del componente y se activan los avisos en caso de error, tanto en el editor de componentes como en el editor de placas.

Además, en caso de que se desplace un componente fuera de los límites de la placa se mostrará un mensaje en la barra de título de esta ventana, a la derecha del texto existente, que avisa de que la placa tiene errores. Este mensaje también se mostrará si la placa tiene errores de posición o dimensiones que la hagan salirse de los límites del panel de trabajo.

Cuando detenga el cursor del ratón sobre el nombre de un componente, además de cambiar su imagen a una mano (indicando que ahora el botón izquierdo y la rueda del ratón actuarán sobre los parámetros del componente), se mostrará un pequeño bocadillo (tooltip) con los parámetros del componente, como puede ver en la figura 218.

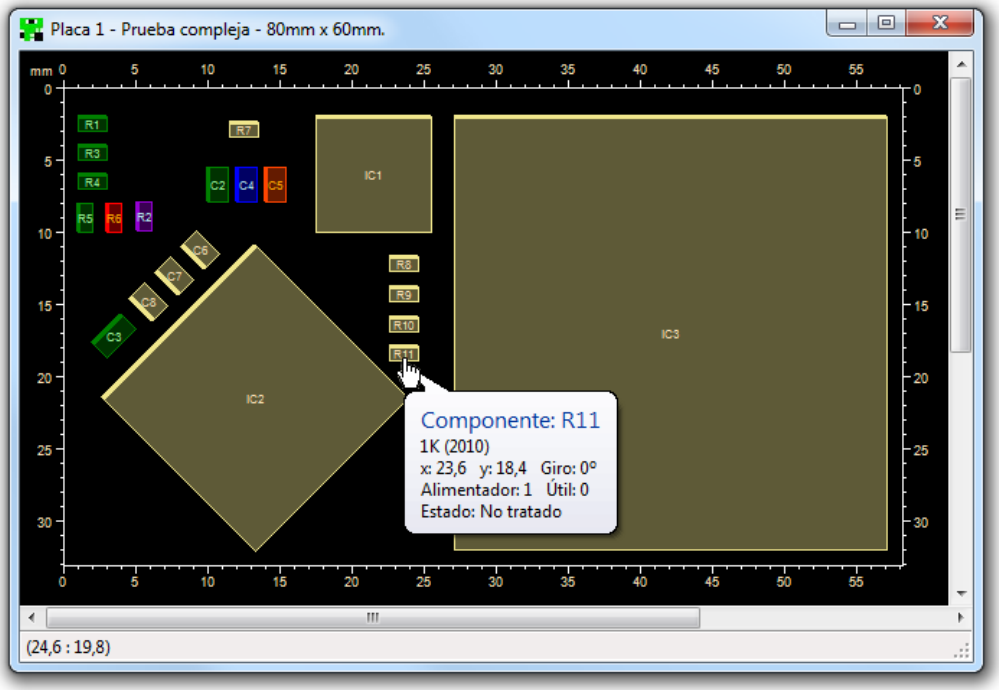


Figura 218. Ventana del editor gráfico mostrando los parámetros de un componente.

Con el botón derecho puede cambiar el estado de un componente. El estado en que quedará depende del estado anterior: Si el componente estaba “No tratado” pasará al estado “Colocado”, en cualquier otro caso, pasará al estado “No tratado”.

Con el botón central se puede desplazar la imagen Gerber de la placa que se muestra como fondo. De esta forma, si la aplicación no la hubiese ubicado automáticamente en la posición adecuada, podrá ajustarla manualmente. La posición de la imagen Gerber se grabará en disco junto al resto de datos de la placa cuando selecciones una de las opciones de guardado del menú archivo.

Ordenar componentes

Esta opción permite ordenar los componentes atendiendo a uno o varios criterios simultáneamente. Esto le permitirá definir tanto el orden en que aparecen en los listados como el orden en que van a ser montados. En la figura 219 se puede ver la ventana que realiza esta función.

En la parte superior de la ventana elija la placa que desea ordenar, si no es la placa que está seleccionada. A continuación puede seleccionar hasta tres criterios simultáneos para ordenar los componentes de la placa elegida en las tres líneas siguientes. Elija el criterio en el control de selección de la izquierda. Puede seleccionar cualquiera de los parámetros de un componente como criterio de ordenación. En "Orden:" elija si quiere que el orden sea ascendente o descendente.

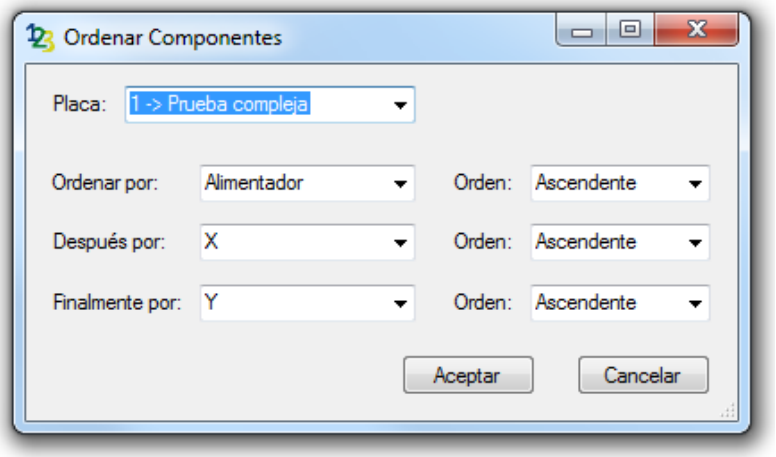


Figura 219. Ventana Ordenar Componentes.

Los componentes se ordenarán según el primer criterio elegido. Cuando dos o más componentes tengan el mismo valor en el campo del primer criterio, éstos se ordenarán según el segundo criterio elegido (si se eligió alguno). Si se selecciona un tercer criterio, cuando dos componentes coincidan tanto en el valor del primer criterio como en el del segundo, se ordenarán según el tercero.

Asignar archivos Gerber

Utilice esta opción para asociar una serie de archivos Gerber a una placa. De esta forma, el programa mostrará la imagen de la placa debajo de los componentes en las ventanas de los editores gráficos y de montaje manual. En la figura 220 se puede ver la ventana de esta opción mostrando una imagen de ejemplo.

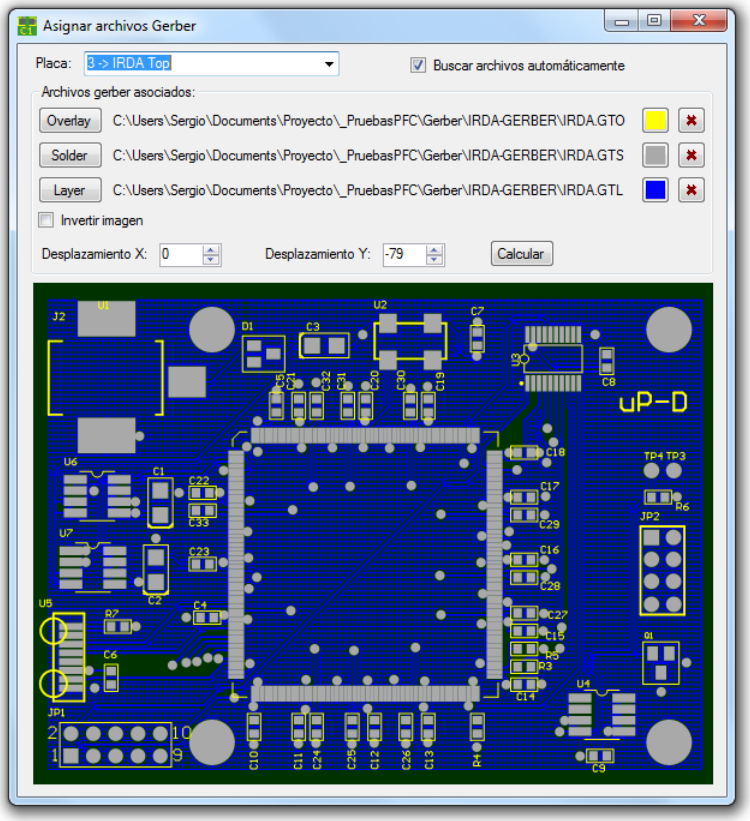


Figura 220. Ventana Asignar archivos Gerber.

El primer control de selección le permite elegir la placa a la que va a asignar los archivos Gerber. Debajo se encuentran tres botones llamados Overlay, Solder y Layer que definen las tres capas que se usarán para generar la imagen. Al pulsar cualquiera de ellos se abrirá un cuadro de diálogo que le permitirá elegir cualquier archivo Gerber para mostrar en esa capa. Al seleccionar un archivo, su ruta en disco aparecerá al lado del botón.

Si está seleccionada la opción "Buscar archivos automáticamente", al seleccionar un archivo para una capa busca archivos adecuados para las otras dos, comprobando las extensiones más habituales para estos tipos de archivo.

Las tres capas se mostrarán en el orden en que están en pantalla, es decir, debajo irá la capa que se seleccione como layer, sobre ésta se dibujará la capa que se seleccione como solder y

por último se dibujará la capa que se seleccione como overlay. Los colores predeterminados con que se dibujará cada capa son los que se pueden ver en el botón que se encuentra a la derecha de su ruta en disco. Pulsando el botón del color se puede cambiar éste por el que se desee, como muestra la figura 221. Con el botón del aspa se puede eliminar la asociación que se ha realizado para esa capa.

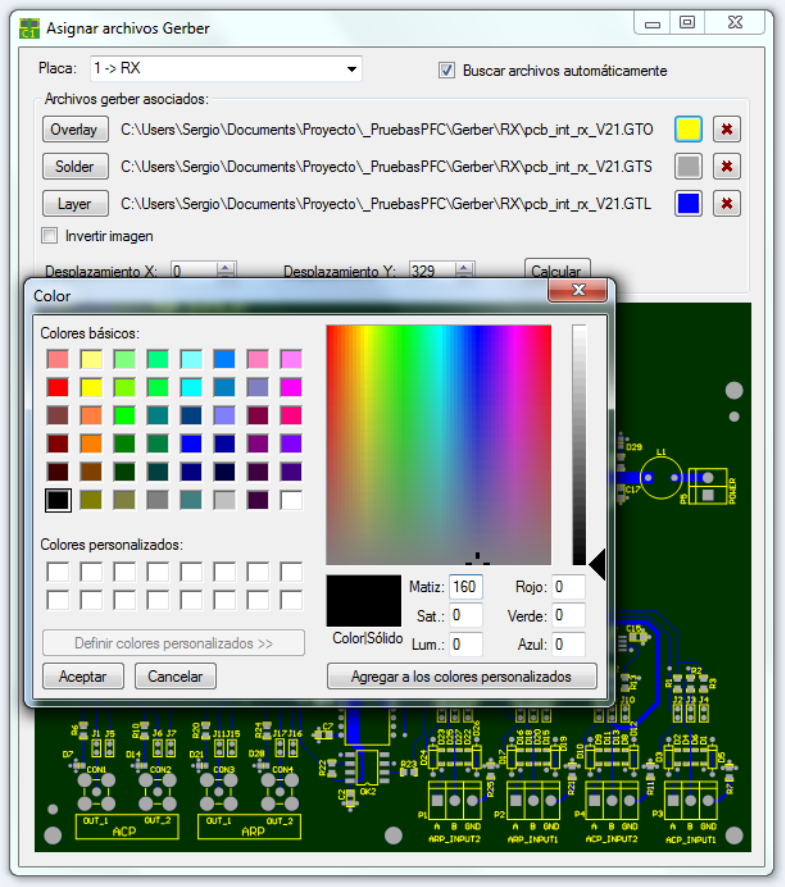


Figura 221. Ventana de selección de colores Gerber.

La opción “Invertir imagen” se usa cuando la placa representa la cara Bottom de una placa. De esta forma se invertirá la imagen y la posición de los componentes para que coincidan con la imagen de la placa dada la vuelta, como es habitual.

Los controles de Desplazamiento x e y, permiten definir la cantidad de píxeles de desplazamiento que tendrá la imagen con respecto al origen de la placa. En algunas situaciones la imagen Gerber puede tener unas dimensiones ligeramente distintas a las dimensiones detectadas para la placa (en caso de no definir manualmente las dimensiones correctas), por lo que puede ser necesario ajustar su posición en la representación visual junto a los componentes de la placa. Pulsando el botón Calcular, se realizará el ajuste automáticamente. Sin embargo, es posible

hacerlo manualmente de forma numérica desde esta ventana y de forma gráfica desde la ventana del editor gráfico, utilizando el botón central para desplazarla.

Los archivos Gerber asociados y las características de representación que elija se guardarán en disco con el resto de los datos de la placa cuando se grabe.

Reiniciar todos los componentes

Con esta opción se restaurará el estado de todos los componentes al estado "No colocado".

Borrar todas las placas

Esta opción elimina de la memoria todos los datos de las placas que estén actualmente cargadas y sus componentes. Si hay editores gráficos de placas abiertos, se cerrarán al ser borrada su placa correspondiente. El resto de datos del usuario, como los alimentadores o las geometrías no se modifican. La opción sólo estará activa si hay placas cargadas en memoria.

Menú Producción

Desde este menú puede iniciar la producción de las placas que están cargadas en memoria y que no contengan errores detectables por el programa. En la figura 222 puede ver el menú producción desplegado.

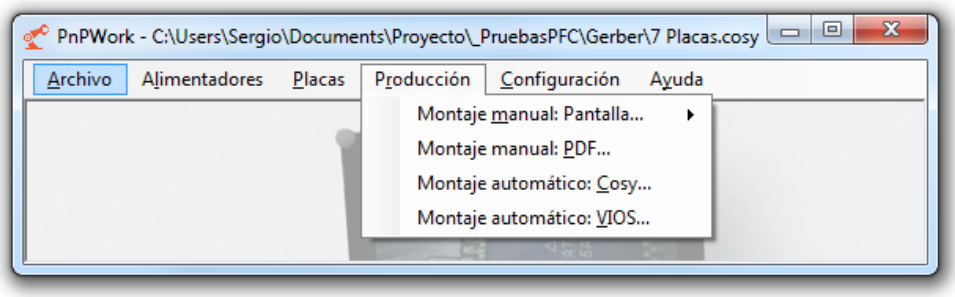


Figura 222. Menú *Producción*.

Montaje manual: Pantalla

Con esta opción podrá ayudarse del programa para realizar un montaje manual interactivo. Esta opción es un submenú dinámico igual que ocurría con el editor gráfico, por lo que

aparecerá una opción de menú por cada placa cargada en memoria, como se muestra en la figura 223. El funcionamiento de este menú es idéntico a lo explicado anteriormente.

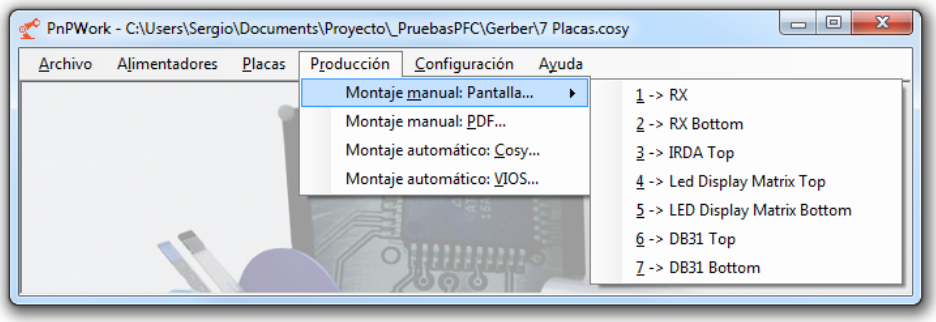


Figura 223. Submenú dinámico Producción manual: Pantalla.

Cuando se selecciona una opción del submenú, se abre la placa seleccionada en una ventana de montaje manual como la que se muestra en la figura 224. Esta ventana es muy parecida a la del editor gráfico, pero no es posible modificar los datos de componentes desde ella, salvo el estado.

Como se puede ver, a la izquierda se muestra el listado de los componentes según se hayan ordenado en etapas anteriores. En este caso, los botones que se encuentran debajo del listado sirven para cambiar el foco al componente siguiente o anterior y no para reordenarlos.

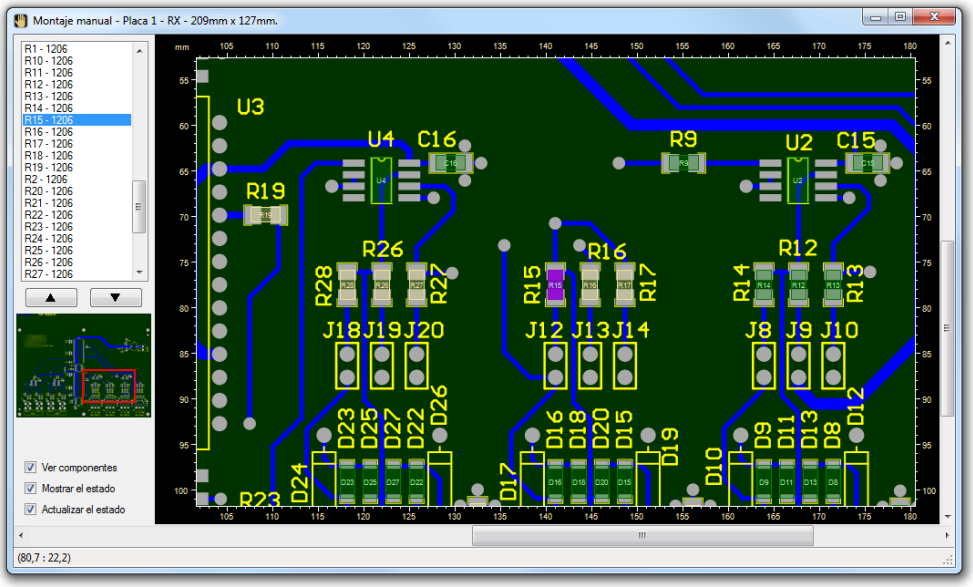


Figura 224. Ventana de montaje manual interactivo.

Debajo de estos botones se muestra una vista en miniatura de toda la placa, marcando con un rectángulo rojo qué parte es la visible en el lado derecho de la ventana. Si se hace clic

sobre dicha imagen, se mostrará ampliada en el lado derecho para poder distinguir mejor sus detalles. En este estado, haciendo clic de nuevo se regresa a la vista normal.

Bajo la vista en miniatura se encuentran tres opciones que se pueden seleccionar individualmente. La opción “Ver componentes” permite ver todos los componentes o sólo el que se está colocando (el que está seleccionado en el listado) en este momento. La opción mostrar el estado hace que se vea cada componente del color que le corresponde según su estado. Si no se encuentra seleccionada, todos los componentes se muestran de color amarillo claro. La opción “Actualizar el estado” hace que cada vez que se pase de un componente a otro, el programa marque el anterior como “Colocado”. Si no se selecciona, el estado de los componentes no cambiará.

En el lado derecho de la pantalla se muestra la placa de una forma muy parecida al editor gráfico, siendo el funcionamiento del ratón similar al comentado anteriormente, salvo que no está permitido trasladas o girar los componentes.

Aunque en esta pantalla se puede ajustar la posición de la imagen Gerber de fondo de la misma manera que en el editor gráfico, en este caso este cambio no quedará grabado.

Montaje manual: PDF

Esta opción le permitirá generar un archivo PDF con la información gráfica de montaje de los componentes de forma que pueda imprimirla en papel o enviarla por medios electrónicos. Al seleccionarla se muestra la ventana de la figura 225.

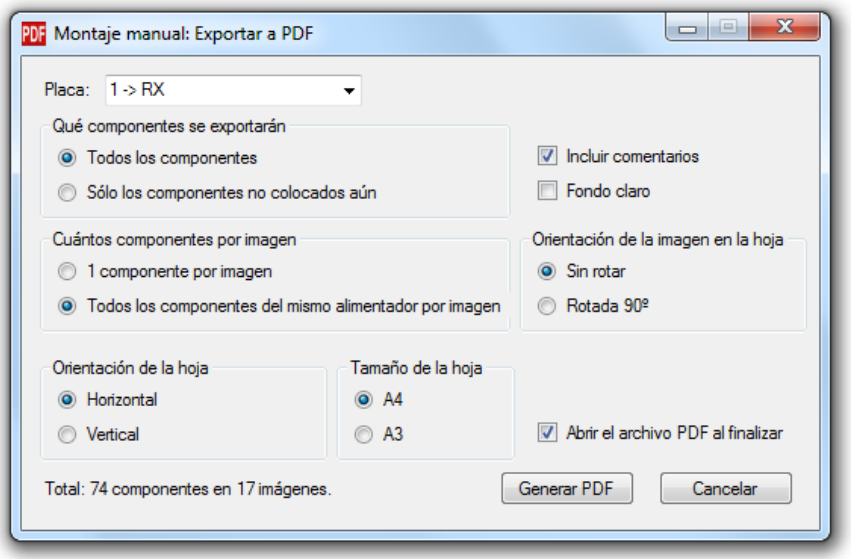


Figura 225. Ventana de generación de archivo PDF.

La primera opción le permite seleccionar la placa para la que va a generar el archivo PDF. A continuación puede seleccionar si deben aparecer todos los componentes o sólo los marcados como no colocados. También debe elegir si desea generar una imagen por cada componente o poner en la misma imagen todos los componentes del mismo alimentador.

Incluir comentarios permite que se añada la numeración y el tipo del componente a cada imagen. Si va a imprimir el PDF, la opción "Fondo claro" le permitirá ahorrar tinta. Si la imagen es más ancha que alta, la opción de rotarla 90° permitirá aprovechar más la hoja generando una imagen más grande.

Por último, se puede seleccionar el formato de impresión en la hoja, apaisada o vertical y el tamaño del papel. El botón Generar PDF inicia el proceso y al terminar se mostrará el PDF resultante si se ha marcado la casilla "Abrir el PDF al finalizar".

Montaje automático: Cosy

Con esta opción se mostrará la ventana de producción que le permitirá seleccionar las placas a montar, las opciones finales de montaje e iniciar el montaje con la máquina Cosy. Esta opción sólo estará activa si hay placas y alimentadores en memoria, puesto que ambos son imprescindibles para realizar el montaje.

Montaje de componentes

Puede iniciar el montaje de componentes desde la ventana de producción, mostrada en la figura 226. Como puede ver en la imagen, esta ventana le permite seleccionar qué placas deben ser procesadas. Para ello, en el listado de la izquierda, muestra las placas con una casilla de selección para cada una.

Si hay placas con errores, no se mostrarán en el listado de selección de placas, avisando en ese caso con un mensaje en rojo. En el ejemplo de la imagen se ha introducido un error en la placa número 3 para que muestre el mensaje de aviso y, como se puede observar, la placa número 3 no aparece en el listado.

En cada elemento del listado se muestra, además del número de la placa, su nombre, el número de componentes pendientes y el número de componentes total de la placa. En caso de que sea necesario, aparecerán barras de desplazamiento. Los componentes pendientes son aquellos que tienen su estado distinto de "*Colocado*" (ver editor de placas y componentes).

Al abrir la ventana, todas las placas con componentes pendientes aparecen marcadas para ser procesadas, mientras que las placas que no tienen componentes pendientes no están marcadas.

A la derecha del listado, una casilla de selección le permite elegir si se reinicia el estado de los componentes al estado "*No tratado*". En caso de que la marque, se montarán todos los componentes de las placas seleccionadas. Si marca para procesar una placa sin componentes pendientes, sus componentes no se montarán a menos que marque la casilla "*Reiniciar componentes*".

En caso de que vaya a continuar una producción detenida anteriormente, no debe marcar la casilla "*Reiniciar componentes*" pues volvería a colocar componentes sobre los ya puestos. En su lugar, déjela sin marcar para que el programa procese sólo el resto de componentes, los que aún no habían sido tratados y los que no se pudieron montar por cualquier error.

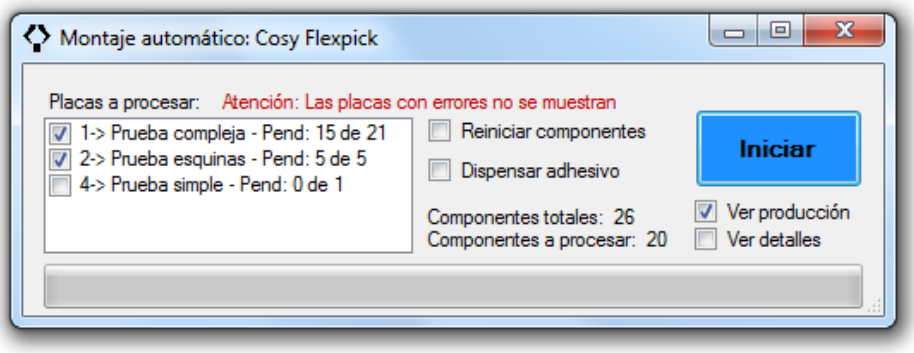


Figura 226. Ventana *Producción*.

Si desea dispensar adhesivo para fijar los componentes a la placa, marque la opción "*Dispensar adhesivo*". Recuerde que la cantidad de adhesivo que se dispensa al componente debe haberla especificado en el editor de alimentadores. Recuerde también que la cantidad de gotas de adhesivo que se debe dispensar debería ser proporcional a su tamaño y a la distancia de separación entre la placa y el componente.

Tenga en cuenta que los componentes tratados anteriormente, pero no colocados por errores, ya tienen dispensado el adhesivo. Por tanto, no debería marcar esta casilla si está haciendo una segunda pasada para procesar componentes pendientes de montar por errores en una producción anterior.

En la ventana se muestra el número total de componentes que hay en las placas seleccionadas y el número total de componentes a procesar. Si no hay componentes ya

colocados, ambos números serán iguales, pero si han quedado componentes por montar en una producción anterior, estas cifras reflejarán la cantidad de trabajo pendiente del total.

El editor gráfico de componentes se puede usar como representación gráfica del proceso de montaje de la placa. Si se activa la opción "*Ver producción*" en la ventana de producción, se mostrará en primer plano la representación gráfica de la placa que se está montando y se van cambiando los colores de los componentes, en tiempo real mientras la máquina los va tratando, para reflejar el estado en que están en cada momento.

Durante el montaje, la edición queda deshabilitada, por lo que no se pueden hacer cambios y el editor gráfico queda como un simple visor de lo que le está pasando a los componentes sobre la placa. La casilla de selección "*Ver producción*" se puede activar o desactivar en cualquier momento, antes de empezar o durante el montaje.

Puede comenzar el montaje pulsando el botón "*Iniciar*". En cuanto se inicia se desactiva el listado y la casilla "*Reiniciar componentes*", por lo que no podrá cambiar estas opciones durante las operaciones de montaje. Además, como se muestra en la figura 227, el botón iniciar cambia, pasando a tener la función de detener el montaje.

Al detener una producción mediante el botón "*Detener*" de la ventana, ésta no se para inmediatamente, sino que antes se termina de colocar el componente que se está procesando y luego se aparcan los útiles, el cabezal y el banco de alimentadores. Si necesita detener la producción con urgencia, la máquina dispone de botones de seguridad en su frontal y en el lateral izquierdo, que son más apropiados que el botón "*Detener*" de la ventana porque la detienen inmediatamente.

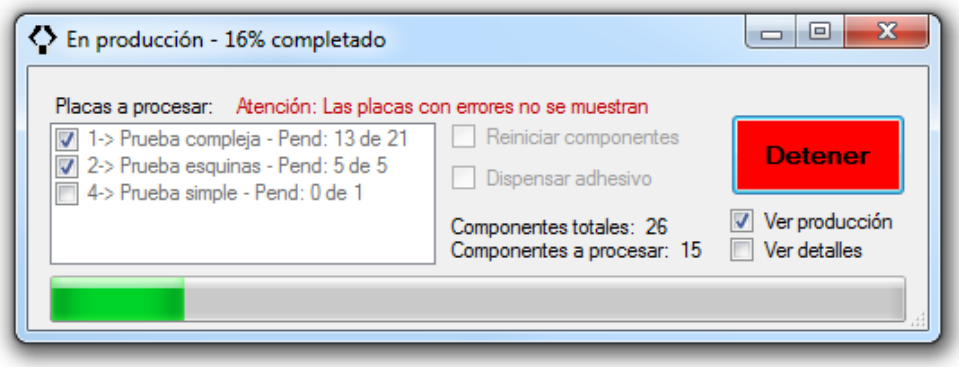


Figura 227. Ventana de montaje automático con un montaje en curso.

La ventana de producción no se puede cerrar mientras se está ejecutando un montaje. Si desea cerrar la ventana, antes debe detener la producción. Además, el resto de menús y ventanas del programa permanecerán bloqueados mientras esté abierta la ventana de producción.

Durante la producción, una barra de progreso en la parte inferior de la ventana le permite ver la cantidad de trabajo realizado hasta el momento.

El programa produce un informe detallado de lo que sucede durante la producción, pero, como lo normal es que no necesite ver el informe detallado, el formulario se inicia contraído para evitar que dicho informe sea visible. Si desea ver dicho informe, marque la casilla de selección "*Ver detalles*" o simplemente estire la ventana arrastrando desde su borde con el ratón. En la parte inferior de la ventana se mostrará un cuadro de texto con el informe detallado, como se puede ver en la figura 228.

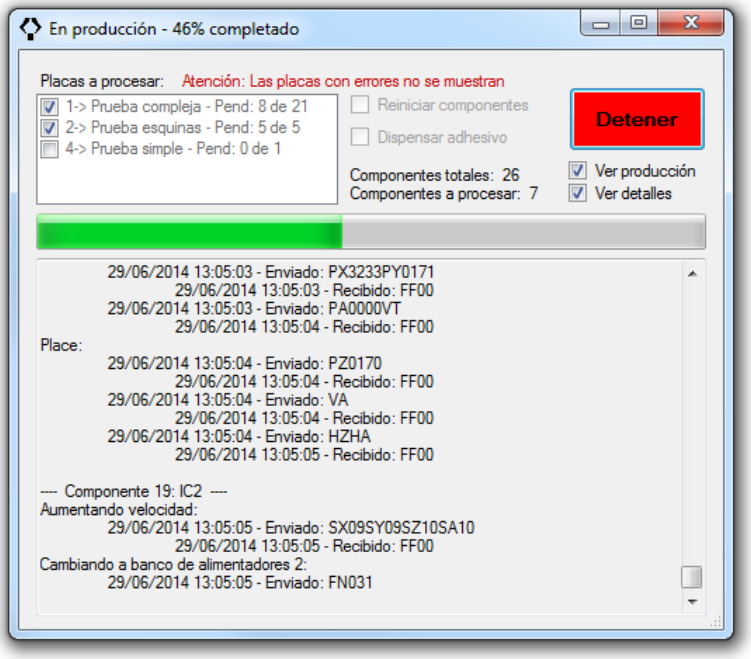


Figura 228. Ventana de montaje automático mostrando un informe detallado.

Cuando termine el montaje, el informe de producción se guardará automáticamente en disco en una carpeta llamada COSYLog (aunque puede cambiarlo por otro nombre en las propiedades de configuración de la aplicación), en el mismo directorio donde se encuentra el archivo .pnp con el que está trabajando. El nombre del archivo de informe será el mismo que el del archivo .pnp, añadiendo al final "Producción_", un número y la extensión .log. Éste número será igual al número de veces que ha iniciado la producción del trabajo actual.

El informe de producción indica cada vez que inicia el proceso de una placa nueva y cada vez que inicia el proceso de un componente. Además, indica en la primera columna la operación que se ha iniciado, en el primer tabulador los mensajes enviados a la máquina a través del puerto serie y en el segundo tabulador los mensajes recibidos de ésta. Todos los mensajes transmitidos o recibidos del puerto serie tienen una marca horaria para ayudar en la depuración, en caso de error.

Montaje automático: VIOS

Con esta opción podrá generar una archivo VIOS para cargar el diseño en cualquier máquina pick & place compatible con este formato de archivos. Al pulsarla se muestra la ventana de la figura 229.

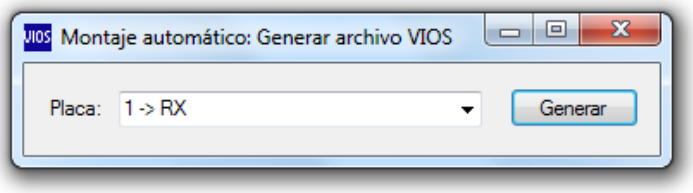


Figura 229. Ventana de generación de archivo VIOS.

Tras seleccionar basta pulsar el botón “Generar” para que se abra el cuadro de diálogo de grabar archivo para que elija la ubicación donde va a guardar el archivo VIOS generado.

Menú Configuración

Desde este menú puede seleccionar el puerto serie y la cámara con que se va a comunicar con la máquina. También puede cambiar algunas características del comportamiento de la aplicación que le facilitarán su uso cotidiano.

Todas las opciones de configuración se guardan automáticamente en las propiedades de configuración de la aplicación, al igual que el estado de la ventana principal (sus dimensiones o si está maximizada/minimizada). Por tanto, cualquier opción que escoja en este menú permanecerá activa hasta que la cambie, en la misma o en otra sesión de trabajo.

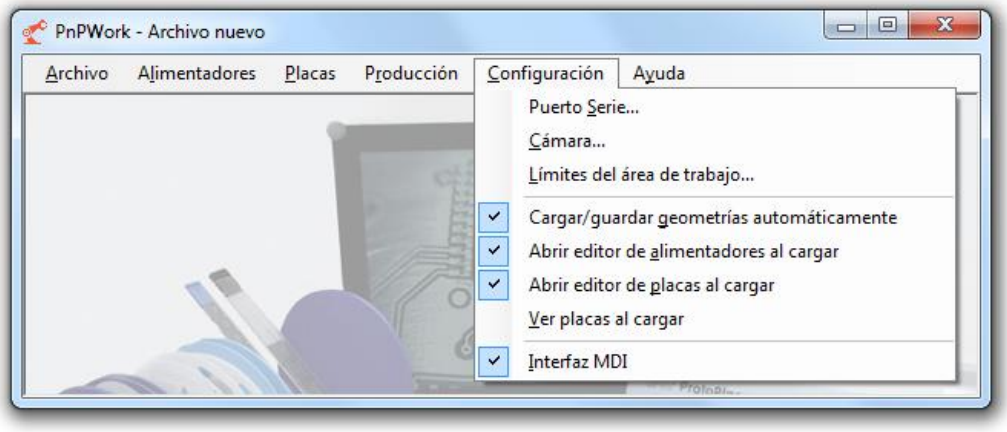


Figura 230. Menú *Configuración*.

Puerto Serie

Con esta opción se abre la ventana de selección del puerto serie, mostrado en la figura 185, desde la que podrá elegir, de entre los puertos serie presentes en el ordenador, aquel al que haya conectado la máquina.

El resto de parámetros del puerto serie también son configurables, pero, como en condiciones normales no debe cambiarlos, no se mostrarán en la ventana para evitar errores accidentales. En su lugar, acuda a las propiedades de configuración de la aplicación si necesita cambiarlos debido a una situación excepcional.

Cámara

Desde esta opción se abre la ventana de configuración de la cámara, que puede ver en la figura 231 y le permitirá elegir qué cámara de las conectadas actualmente al ordenador se desea usar.

El programa escanea constantemente la conexión de nuevas cámaras de vídeo desde esta ventana, por lo que, si no la había conectado aún, conéctela en este momento y espere a que aparezca la imagen de la cámara. La imagen podría tardar en aparecer algunos segundos, dependiendo del tiempo que tarde en inicializarse su driver correspondiente en Windows.

Sobre la imagen de la cámara aparecerán sobreimpresas las características de la imagen recibida. Si no desea que aparezca esta información, desmarque la casilla de selección “Mostrar Datos” que está la derecha. Bajo esta casilla se activará un botón para seleccionar cada cámara detectada, mostrando el nombre con el que ésta se identifica al sistema operativo.

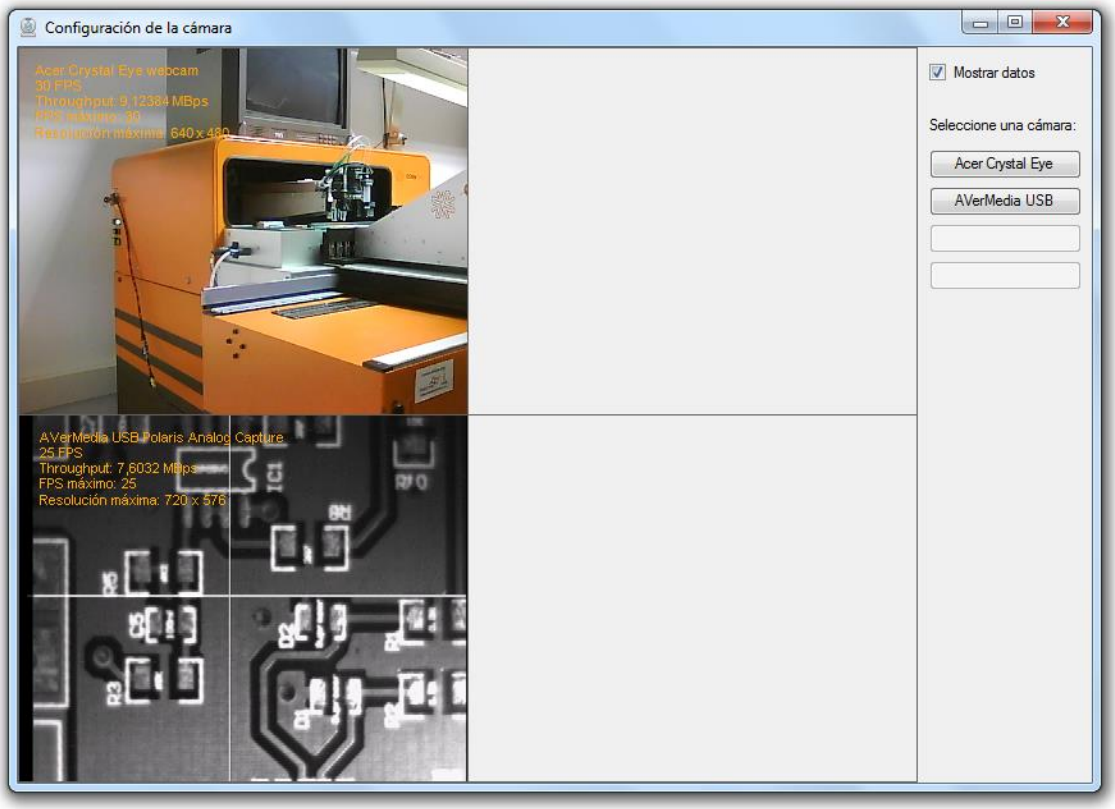


Figura 231. Ventana Configuración de la cámara.

También desde esta ventana podrá realizar ajustes sobre la imagen recibida desde la fuente de vídeo, como el nivel de brillo o contraste de la imagen, entre otros. Para ajustar la imagen, haga clic con el ratón sobre la imagen de vídeo que desea ajustar y se mostrará la ventana de propiedades de la fuente de vídeo que le corresponde.

Estas propiedades suelen estar divididas en varios paneles de configuración, como puede ver en el ejemplo de la figura 232. En este ejemplo puede ver que el primer panel permite elegir el tipo de señal de vídeo que proporciona la fuente de vídeo, mientras que el segundo panel permite ajustar el brillo, contraste, matiz, saturación y nitidez de la imagen.

Los cambios que realice se aplicarán normalmente en tiempo real a la imagen de vídeo que se esté mostrando en pantalla en ese momento, permitiéndole un ajuste óptimo.

Recuerde que la cámara montada en la máquina pick & place dispone de anillos de enfoque y diafragma manuales que también puede ajustar para obtener una mejor imagen. Ajústelos si la imagen recibida está desenfocada o se muestra muy clara o muy oscura antes de cambiar los parámetros de configuración en el software.

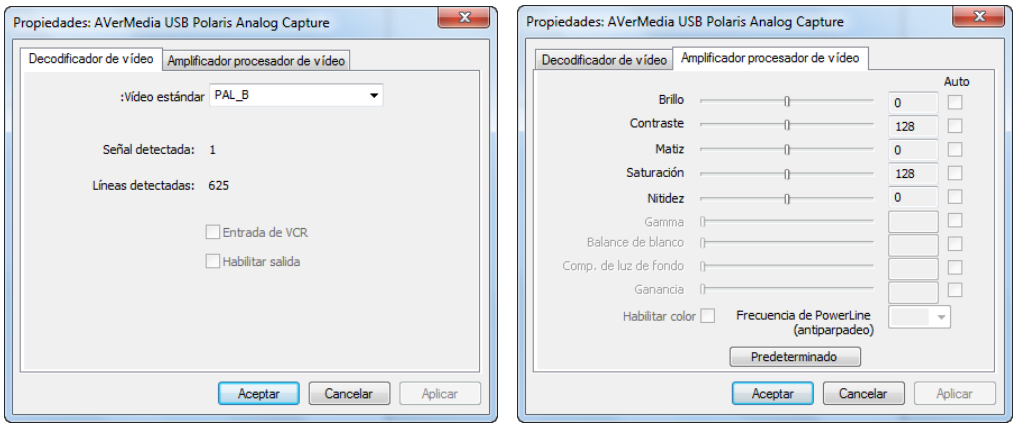


Figura 232. Ventana *Propiedades* de la cámara.

La cantidad de ajustes disponibles depende del software controlador de la cámara, instalado en el sistema operativo del ordenador. Como ejemplo, en la figura 233 se muestran las ventanas de propiedades de otra fuente de vídeo existente en el equipo.

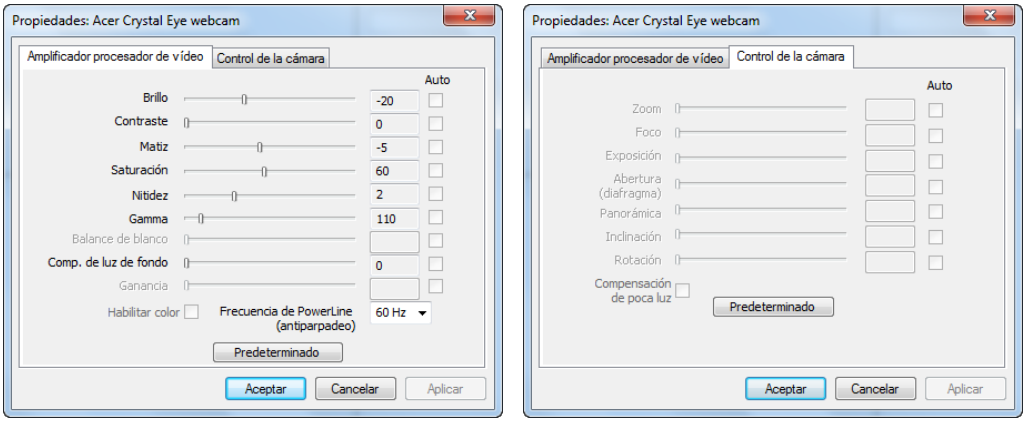


Figura 233. Ventana *Propiedades* de una segunda cámara.

Cuando tenga la imagen de la cámara ajustada según sus necesidades, pulse el botón "Aceptar" para cerrar la ventana "Propiedades".

Finalmente, pulse sobre el botón de la cámara (figura 231) para elegirla. La ventana "Configuración de la cámara" se cerrará automáticamente en cuanto elija la cámara con la que va a trabajar.

Establecer los límites del área de trabajo

Desde esta ventana puede establecer los límites del área de trabajo según la máquina que vaya a emplear o bien imponer unos valores personalizados, como se puede observar en la figura 234. Se pueden escribir directamente los valores para el límite en la dimensión x y la dimensión y

o bien pulsar el botón correspondiente a la máquina deseada para que se rellenen los campos automáticamente.

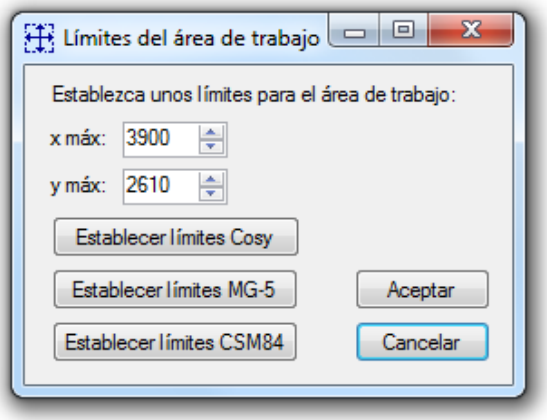


Figura 234. Ventana para establecer los límites del área de trabajo.

Pulsando Aceptar se hace efectivo el cambio, mientras que Cancelar deja los límites que hubiese establecidos anteriormente.

Cargar/guardar geometrías automáticamente

Esta opción permite automatizar el uso habitual de un archivo de geometrías para ayudar en el proceso de conversión de ficheros de pick & place de Altium. Lo normal es que se use siempre el mismo archivo de geometrías, donde se tendrán descritas las dimensiones físicas de los componentes que se usan en los diseños electrónicos.

Activando esta opción hará que el programa, al iniciarse, cargue el último archivo de geometrías que se usó y que, al salir del programa, grabe sobre este archivo las geometrías que actualmente están en memoria. De esta forma se actualizará el archivo de geometrías habitual con las que se hayan añadido o eliminado durante la sesión de trabajo con el programa.

Esta forma de operar le facilitará el trabajo, puesto que cuando vaya a convertir un archivo de Altium ya tendrá en memoria todas las geometrías que haya introducido en trabajos anteriores, debiendo introducir únicamente las dimensiones de los componentes nunca usados anteriormente.

Esta opción (y todas las que le siguen) indica en el menú configuración si actualmente está activada o no mediante una marca a su izquierda.

Abrir editor de alimentadores al cargar

Esta opción hace que se abra el editor de alimentadores al cargar datos o empezar de cero desde la opción Archivo → Nuevo. Manténgala activada si al cargar archivos o iniciar un trabajo nuevo suele abrir el editor de alimentadores, así ahorrará tiempo.

Abrir editor de placas al cargar

Esta opción hace que se abra el editor de placas y componentes al cargar datos o empezar de cero mediante Archivo → Nuevo. Manténgala activada si al cargar archivos o iniciar un trabajo nuevo suele abrir el editor de placas y componentes, así ahorrará tiempo.

Ver placas al cargar

Esta opción hace que se muestren gráficamente todas las placas al cargarlas desde disco. Manténgala activada si al cargar archivos suele abrir el editor gráfico, así ahorrará tiempo.

Tenga en cuenta que el programa abrirá todas las placas existentes en el archivo, por lo que un archivo con muchas placas podría provocar que se llenase la pantalla de ventanas, ya que se abre una nueva ventana para representar cada placa cargada.

Interfaz MDI

Con esta opción puede cambiar la forma en que se distribuyen las ventanas del programa. Si está marcada, la aplicación le presentará una interfaz MDI, en la que la ventana principal contiene en su interior al resto de ventanas, siempre que sean ventanas comunes “no-modales” (las ventanas modales —como por ejemplo los cuadros de diálogo Abrir, Guardar como, etc. — siempre se pueden sacar fuera de los límites de la ventana principal).

En la figura 235 puede ver un ejemplo de la aplicación, mostrando un interfaz MDI, con la ventana principal y tres ventanas en su interior. Con este tipo de interfaz, se fuerza a las ventanas “hija” normales a permanecer dentro de su ventana “padre” que es la ventana principal de la aplicación.

Si alguna de las ventanas hija no queda completamente visible, como ocurre en el ejemplo con la ventana del editor de alimentadores, la ventana padre mostrará barras de desplazamiento para poder desplazar su contenido hasta que quede visible.

Si se pulsa el botón de la aplicación en la barra de tareas, la ventana principal, se minimiza (o se restaura si previamente había sido minimizada), llevándose consigo todas las ventanas hija que contiene en su interior.

Si desmarca la opción "*Interfaz MDI*" del menú configuración, la aplicación reducirá la ventana principal al tamaño del menú, dejando de contener al resto de ventanas.

Si habían ventanas hija abiertas, como se muestra en la figura 235, al desmarcar esta opción la aplicación las dejará abiertas, pero las sacará de la ventana principal, que dejará de ser la ventana padre del resto de ventanas de la aplicación.

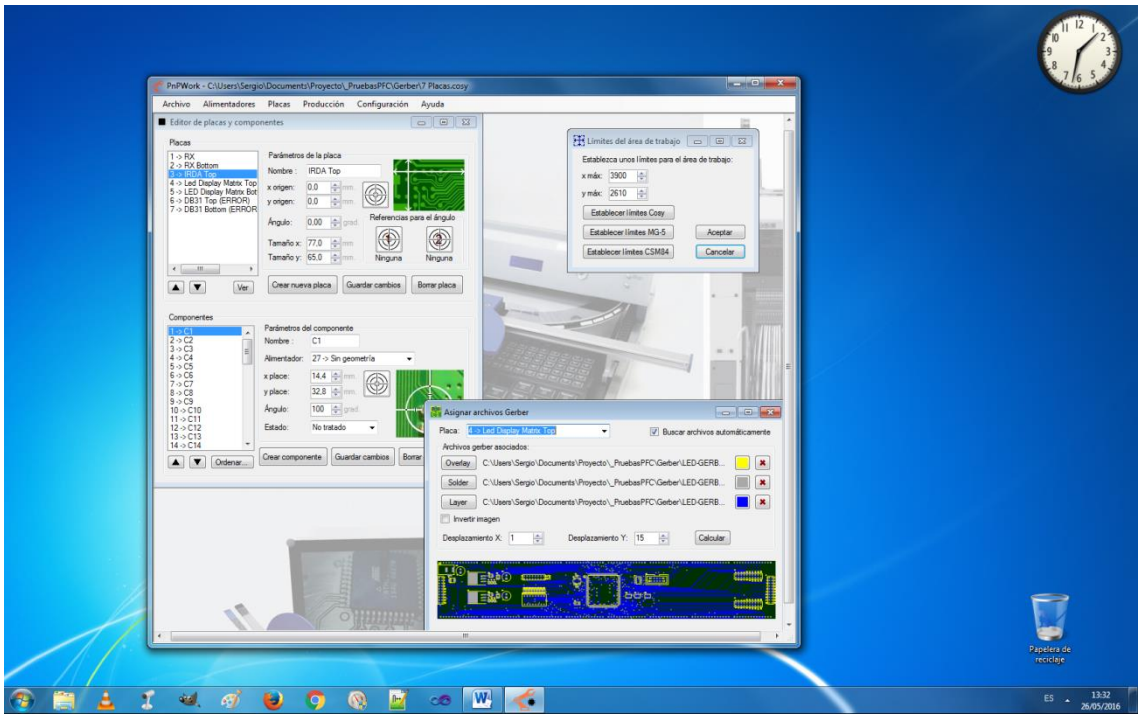


Figura 235. Aplicación con interfaz MDI.

Esto le permitirá distribuir las ventanas en la pantalla como desee, intercalándolas con otras ventanas de aplicaciones diferentes y minimizándolas o maximizándolas independientemente.

Como se puede comprobar en la imagen, ahora el botón de la aplicación en la barra de tareas muestra que se trata de una aplicación multiventana y al pulsarlo nos permite seleccionar cuál de las ventanas abiertas se va a minimizar o restaurar como ventana independiente del resto, tal como puede ver en la figura 236.

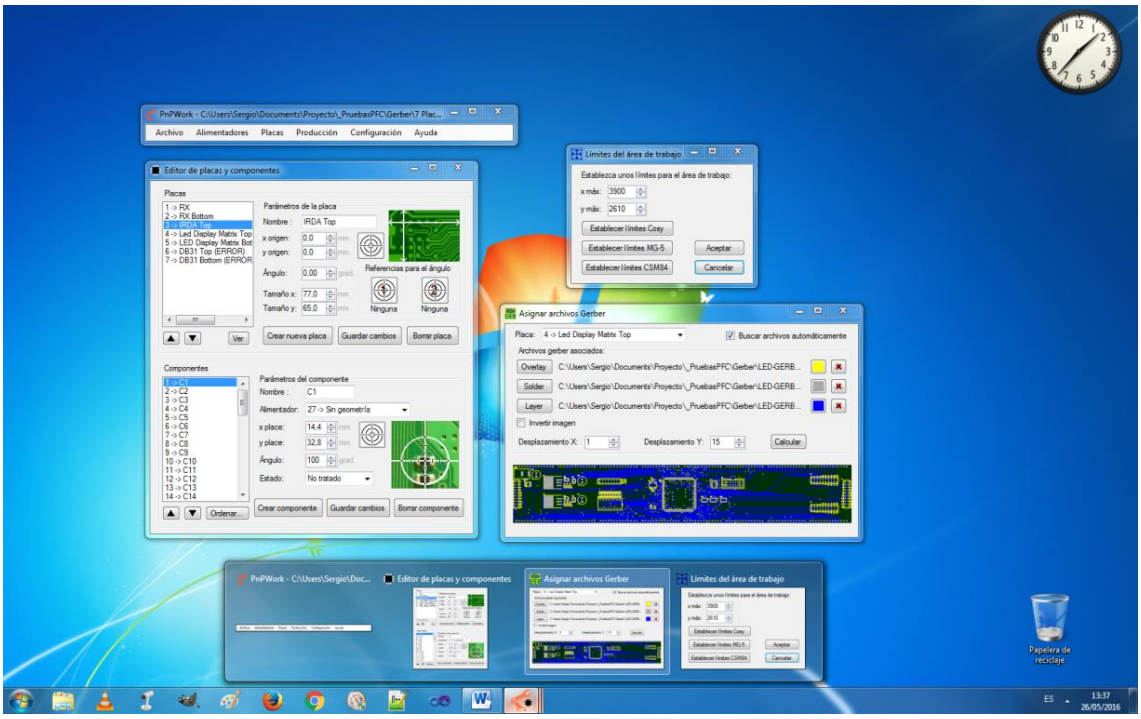


Figura 236. Aplicación con interfaz no MDI.

Menú Ayuda

Desde este menú, mostrado en la figura 237, podrá acceder al manual del usuario y a la ventana "Acerca de" de la aplicación, como suele ser habitual en las aplicaciones con interfaz gráfica de usuario.

Figura 237. Menú *Ayuda*.

Manual del usuario

Con esta opción se abrirá este manual del usuario de la aplicación en formato PDF. Necesitará tener instalado en el ordenador un visor de archivos PDF.

Guía rápida

Con esta opción abrirá la guía rápida para el montaje de placas en formato PDF. Esta guía le mostrará listas de acciones ordenadas para realizar todo el proceso hasta culminar con el

montaje automatizado de la placa. Dispone de listas de acciones para montar una placa partiendo de un archivo de Altium y también partiendo de cero, es decir, especificando todos los datos de alimentadores, placa y componentes.

Acerca de

Esta opción muestra la ventana de datos sobre la aplicación y el autor que se puede ver en la figura 238.

Figura 238. Ventana *Acerca de*.

CONTROL MANUAL DE LA COSY

El control manual le permite mover el brazo que tiene montada la cámara para obtener las coordenadas de un punto sobre el panel de trabajo. Si conecta la cámara de la máquina al ordenador mediante una digitalizadora de vídeo, el programa podrá mostrar la imagen de la cámara con la mira que ésta proporciona para que apunte al objetivo deseado.

Esto le será útil para tener una mayor precisión a la hora de especificar las coordenadas desde donde coger un componente en su alimentador. También para especificar las coordenadas donde colocarlo en el editor de componentes (en el caso de que se esté creando manualmente los datos de colocación sobre la placa).

Cuando coloque una placa sobre el panel de trabajo para montarla, puede usar el control manual para especificar dónde está el origen de la placa sobre el panel y la rotación que tiene ésta (con un error, en el mejor caso, de aproximadamente 2 centésimas de grado).

El acceso al control manual se realiza mediante los botones que tienen el dibujo de una mira en las ventanas del editor de alimentadores y del editor de placas y componentes, como el que se muestra en la figura 239.

Figura 239. Botón para acceder al control manual de la máquina.

¡Atención! Tome precauciones para que no haya personas u objetos en el área de movimiento del brazo de la máquina antes de pulsar el botón.

Cuando se pulsa este botón, aparece la ventana “Seleccionar coordenadas” que se muestra en la figura 240. Desde esta ventana se puede mover el brazo de la máquina de distintas formas.

Normalmente, la ventana que llama al control manual le indica las coordenadas iniciales a las que debe desplazarse, moviendo así el brazo a la zona de interés para la tarea que se está haciendo en la ventana llamante. Estas coordenadas también aparecen en las casillas que están en el recuadro “Posicionar”, en los controles de la parte izquierda de la ventana. Si escribe otras coordenadas y pulsa el botón “IR”, se desplazará el brazo automáticamente hasta la posición que especifique.

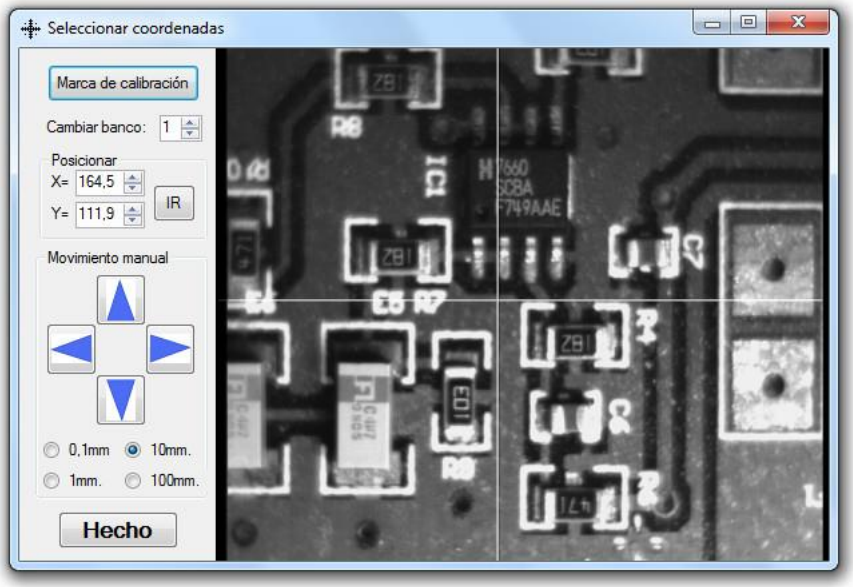


Figura 240. Ventana *Seleccionar coordenadas*.

Debajo de estas casillas se encuentra el cuadro "Movimiento manual" que tiene cuatro flechas y un grupo de botones de selección. Con las cuatro flechas puede mover el brazo en la dirección a la que apunta, mientras que con los botones de selección indica cuanto avanzará el brazo cada vez que se pulse un botón de dirección. Las posibilidades disponibles son pasos de 0,1, 1, 10 y 100 milímetros.

Otra posibilidad para mover el brazo es hacer clic con el ratón directamente sobre la imagen recibida de la cámara. La máquina se moverá para centrar el punto exacto en el que usted haga clic en la mira de la cámara.

La forma óptima de usar esta ventana es moverse con las flechas hacia la zona de interés usando el tamaño de paso adecuado a la distancia a la que se encuentra de su objetivo. Si tiene que cruzar todo el panel, elija pasos de 100 mm, en lugar de pasos de 0,1 mm. Una vez vea en la imagen el punto exacto al que quiere ir, haga clic en él con el ratón y el programa completará el movimiento hasta dejar la mira exactamente sobre el lugar que eligió.

Como podrá observar, mientras mueve el brazo de la máquina, las coordenadas en la ventana llamante se actualizan constantemente. Cuando tenga la mira sobre las coordenadas que necesita, pulse el botón "Hecho" o cierre la ventana directamente. Las coordenadas ya estarán escritas en el punto en que se necesitaban.

El botón "Marca de calibración" de la parte superior izquierda de la ventana centra automáticamente la mira en la marca de calibración de la máquina. Esto le servirá para comprobar que la máquina funciona adecuadamente.

La casilla “Cambiar banco” le permite mover la bandeja de alimentadores sobre su raíl para acceder a los alimentadores de los otros bancos. Escriba el número de banco (de 1 a 3) o pulse las pequeñas flechas para cambiarlo.

¡Atención! Tome precauciones para que no haya personas u objetos en el área de movimiento de la bandeja de alimentadores antes de cambiar de banco.

EDICIÓN DE LAS PROPIEDADES DE CONFIGURACIÓN DE LA APLICACIÓN

La aplicación PnPWork funciona según una serie de parámetros almacenados en las propiedades de configuración de la aplicación. Estas propiedades de configuración de la aplicación son almacenadas en un archivo de configuración con formato XML, permitiendo la edición del mismo con cualquier editor de texto plano o usando una aplicación específica. El archivo se guarda automáticamente en el mismo directorio que el archivo ejecutable de la aplicación y con el mismo nombre que dicho ejecutable, añadiendo la extensión *.config*.

El hecho de incluir en este archivo los parámetros usados para configurar apropiadamente la aplicación, le permite tener acceso a éstos, pudiendo cambiar sus valores por los que le resulten más convenientes. La figura 241 muestra un extracto parcial del archivo de configuración, donde se puede ver la estructura general de éste, que se explica a continuación.

Tras las primeras líneas, que contienen datos necesarios para el correcto uso de este fichero por parte del programa, se llega hasta la sección `<PnPWork.Properties.Settings>`, donde está la información de configuración. El orden en que aparecen los parámetros de configuración, dentro de esta sección, carece de importancia y puede ser diferente al mostrado en la figura 241. En dicha figura se ha dejado una línea de separación entre propiedades para una mayor claridad. Además, los nombres elegidos para las propiedades son bastante descriptivos de la función que controlan.

En esta sección, cada etiqueta *<setting>* incluye un atributo *name*, cuyo valor (puesto entre comillas) especifica el nombre de la propiedad almacenada, mientras que el contenido de la etiqueta *<value>* (hija de la etiqueta *<setting>*) es el valor que se desea asignar a dicha propiedad en nuestra aplicación.

Cada etiqueta *<setting>* termina en su correspondiente etiqueta de cierre *</setting>* y cada etiqueta *<value>* termina en su correspondiente etiqueta de cierre *</value>*. Además cada etiqueta hija está completamente anidada dentro de su etiqueta padre, tal como prescriben las reglas XML.

```
        <?xml version="1.0" encoding="utf-8" ?>
        <configuration>
            <configSections>
                <sectionGroup name="userSettings"
type="System.Configuration.UserSettingsGroup, …
                    <section name="Cosy.Properties.Settings"
type="System.Configuration.Client …
                </sectionGroup>
            </configSections>
            <userSettings>
                <Cosy.Properties.Settings>

                    <setting name="DirectorioBackups" serializeAs="String">
                        <value>PnPWork_Backup</value>
                    </setting>

                    <setting name="NumArchivosBackup" serializeAs="String">
                        <value>10</value>
                    </setting>

                    <setting name="DirectorioLogs" serializeAs="String">
                        <value>COSY_Log</value>
                    </setting>

                    <setting name="SeparadorDatosFicheros" serializeAs="String">
                        <value>;</value>
                    </setting>

                    <setting name="MarcaPlaca" serializeAs="String">
                        <value>Placa</value>
                    </setting>

                    <setting name="MarcaComponente" serializeAs="String">
                        <value>Componente</value>
                    </setting>

                    <setting name="MarcaAlimentador" serializeAs="String">
                        <value>Alimentador</value>
                    </setting>

                    <setting name="ReintentosColocaciónComponente" serializeAs="String">
                        <value>2</value>
                    </setting>

                    <setting name="NombrePuertoSerie" serializeAs="String">
                        <value>COM1</value>
                    </setting>

                    <setting name="VelocidadPuertoSerie" serializeAs="String">
                        <value>4800</value>
                    </setting>

                    <setting name="BitsDatosPuertoSerie" serializeAs="String">
                        <value>8</value>
                    </setting>

                    <setting name="ParidadPuertoSerie" serializeAs="String">
                        <value>None</value>
                    </setting>
```

Figura 241. Extracto parcial del archivo de propiedades de configuración de la aplicación.

Con los valores de configuración predeterminados, y para las máquinas con las que opera el programa, no será necesario que edite este archivo. Sin embargo, se le proporciona esta posibilidad para otorgarle la máxima flexibilidad posible en su uso si desea cambiar alguno de los parámetros por otro valor que le resulte más conveniente

Además de guardar los parámetros que definen el funcionamiento de la aplicación, se ha guardado en las *propiedades de la configuración de la aplicación* las opciones que haya elegido en el menú *Configuración*. De esta forma, cuando se pone en marcha la aplicación, ésta carga los valores guardados en este archivo para configurarla como la dejó la última vez que la usó.

En caso de que desee cambiar alguno de los parámetros que no se han incluido en el menú configuración, siga las siguientes instrucciones:

1. Abra el archivo PnPWork.exe.config. Puede hacerlo con cualquier editor de texto, pero es recomendable uno que coloree la sintaxis del archivo XML.
2. Busque la sección `<PnPWork.Properties.Settings>` (normalmente empezará en la novena línea del archivo, como se muestra en la figura 241). Dentro de esta sección se encuentran todas las propiedades de configuración de la aplicación.
3. Localice el nombre de la propiedad que desea cambiar. Por ejemplo, si desea cambiar la cantidad de archivos de *backup* que realiza el programa, localice la propiedad *NumArchivosBackup*.
4. Localice el valor de la propiedad. Este valor se encuentra en la línea siguiente a la del nombre de la propiedad. En la figura 241, los valores aparecen siempre en color negro. En el caso del ejemplo anterior, sería el valor 10.
5. Cambie el valor por el nuevo valor deseado. En el caso de nuestro ejemplo, si, por ejemplo deseamos que el nuevo máximo sean 20 copias, cambiaremos el 10 por 20.
6. Guarde el archivo y cierre el editor.

Tenga en cuenta que los cambios surtirán efecto cuando inicie de nuevo el programa. Además, antes de salir de PnPWork, se guarda el estado de la aplicación, por lo que se recomienda que haga las modificaciones con el programa cerrado.

FUNCIONAMIENTO DE LAS COPIAS DE SEGURIDAD AUTOMATIZADAS

Cada vez que se graba un archivo desde la aplicación sobre otro archivo existente en el disco, se crea una copia de seguridad automáticamente. Las copias de seguridad se guardan de manera predeterminada dentro de una carpeta llamada "*PnP_Backup*" (aunque puede ser cambiado por el usuario) creada en el mismo directorio en que está el fichero que se ha sobrescrito.

Para evitar que una segunda grabación elimine la primera copia de seguridad, el programa va renombrando las copias de seguridad anteriores. Las copias de seguridad se llaman de igual forma que el archivo original añadiendo la extensión .backup seguida de un número de orden.

El número de orden de los archivos de backup indica la antigüedad del archivo. Así, el archivo con extensión .backup1 será siempre la última copia de seguridad realizada. La anterior a esta (y por tanto más antigua) tendrá la extensión .backup2. La siguiente tendrá la extensión .backup3, y así sucesivamente.

Si desea recuperar un archivo de copia de seguridad, simplemente muévalo a la carpeta que desee y edite el nombre del archivo, eliminando la extensión .backup en caso de ser necesario.

De manera predeterminada se han establecido 10 copias de seguridad por archivo para evitar un número excesivo de copias de seguridad que saturen el disco. Sin embargo, éste valor puede ser cambiado si desea que el programa mantenga un número de copias diferente. Para cambiar el número de copias de seguridad por cada archivo que el programa mantiene o el nombre predeterminado para las carpetas de copias de seguridad, vea la sección dedicada a la edición de las propiedades de la aplicación.

INSTALACIÓN DE LA APLICACIÓN

La instalación de la aplicación se puede realizar de la forma habitual en Windows mediante un programa instalador o se pueden copiar los archivos de que consta a una carpeta para su ejecución sin necesidad de ser instalada (portable). Para ello se dispone de dos versiones, una con un programa instalador y otra sin él, que sólo contiene los archivos necesarios para ejecutar el programa.

Si elige instalar la aplicación desde su programa instalador, éste comprobará que el equipo de destino tiene instalada la versión 4 del *framework* .NET, necesario para la ejecución de la aplicación. En caso de que no esté instalado, el programa instalador se encargará de instalarlo. La instalación mediante el programa instalador dejará un icono de la aplicación en su menú Inicio de Windows y en su escritorio. Además, asociará los archivos .pnp a la aplicación.

Si elige instalar la aplicación copiando usted mismo los archivos a una carpeta, descomprímalos desde el archivo .zip que los contiene y cópielos en la ubicación que desee. Tenga la precaución de descomprimir todos los archivos de la aplicación y dejarlos juntos (en la misma carpeta).

En este caso, tenga en cuenta que si su equipo no tiene instalado el *framework* .NET versión 4.0, tendrá que instalarlo antes de poder ejecutar la aplicación. Puede descargarlo desde www.microsoft.com o utilizar el fichero incluido “dotNetFx40_Full_x86_x64.exe”. Este archivo es el instalador del *framework* proporcionado por Microsoft y redistribuible, para ser instalado sin necesidad de una conexión a internet.

ANEXO II. GUÍA RÁPIDA

En este anexo se ha incluido una copia de la guía rápida para el montaje de placas que acompaña a la aplicación desarrollada. En esta guía se proporciona al usuario novel de la aplicación una serie de pasos con las acciones a realizar para llevar a cabo la tarea, dependiendo de si va a partir de un fichero de BOM de Altium o no. Esto le permitirá trabajar con la aplicación sin casi conocerla, acelerando el proceso de aprendizaje.

El archivo proporcionado al usuario se puede encontrar en la carpeta de la aplicación con el nombre "*Guía rápida.pdf*".

Guía rápida para el montaje de placas

Pasos para montar una placa partiendo de un archivo de Altium

Para crear una placa partiendo de un archivo BOM de Altium, puede seguir los pasos de la fig. 1, listados a continuación:

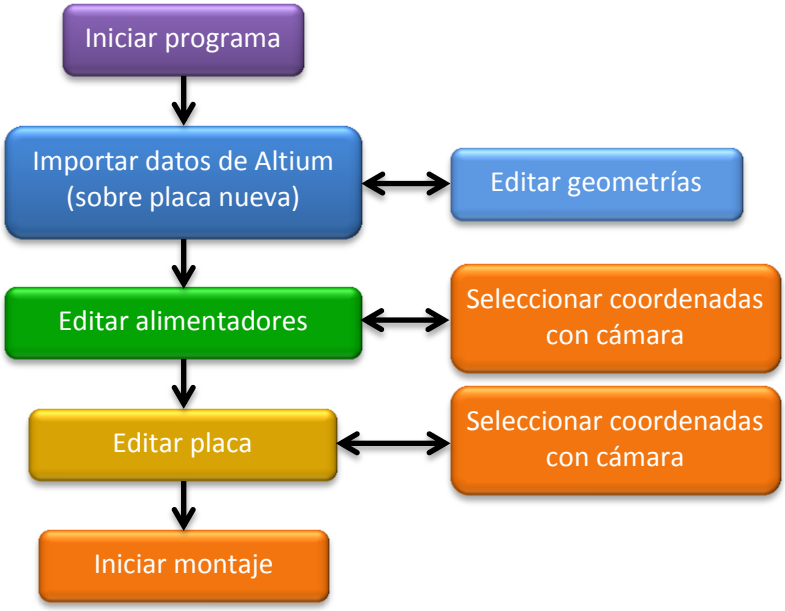


Fig. 1. Esquema de uso importando desde Altium sobre placa nueva.

1. *Archivo* → *Importar Altium*.
2. Elija el archivo de Altium en el cuadro de diálogo *Abrir Archivo* y luego pulse *Abrir*.
3. Elegir la cara (*Top* o *Bottom*) y el campo que se asociará a las geometrías (si no es *Footprint*). Especifique el alto de la placa si el origen detectado es la esquina inferior izquierda.
4. Si las geometrías de este diseño estuviesen en un archivo diferente al archivo de geometrías habitual, pulse *Cargar Geometrías* en la ventana *Importar Altium* y elija el archivo de geometrías en el cuadro de diálogo *Abrir*.
5. Edite las geometrías que faltan con el editor de geometrías que sale automáticamente. Si cierra el editor de geometrías, para abrirlo de nuevo pulse el botón *Editar geometrías* o bien en *Alimentadores* → *Editor de Geometrías*. Cuando cree una nueva geometría, automáticamente se actualizará el listado de geometrías detectadas en la ventana *Importar Altium*.
6. Pulsar *Agregar* en la ventana *Importar Altium*. Se abrirá el editor gráfico de la placa, mostrando sus componentes y puede que también el editor de alimentadores y el editor de placas y componentes, si tiene configurada su apertura automática en el menú configuración. Si lo desea, puede utilizar el editor gráfico para comprobar que los componentes están ubicados correctamente.
7. *Alimentadores* → *Editor de Alimentadores*.
   Si todos los componentes tienen geometría, cargue los alimentadores necesarios en la plataforma de alimentadores de la máquina con los componentes que se van a montar y

especifique en el programa la posición en que se ha de coger ese tipo de componente (valores *x pick* e *y pick*). Para esto se recomienda usar la cámara de la máquina, pulsando el botón de la mira, que rellenará las coordenadas automáticamente. También hay que especificar el banco en que está colocado el alimentador en caso de que no sea el primero, con qué útil hay que coger el componente, en caso de que sea necesario (depende del tamaño del componente), y si se debe activar el avance del alimentador tras coger un componente.

En los componentes que lo requieran será necesario especificar también el valor de la altura a la que debe ser cogido, transportado y puesto sobre la placa (puede usar los controles deslizantes), además de la velocidad del movimiento del cabezal en los cuatro ejes (especialmente para componentes a los que les pueda afectar la inercia por su peso) y el centrado que pudiesen necesitar.

Si hay componentes sin geometría, el programa les habrá asignado a todos un alimentador genérico común, creado con valores predeterminados. Tendrá que crear un alimentador por cada tipo de componente sin geometría y editar los componentes de la placa para asignando a cada uno de estos componentes su alimentador.

8. *Placas → Editor de Placas y componentes*.
   Hay que especificar las coordenadas del origen de la placa (se recomienda hacerlo señalando el punto de origen con la cámara de la máquina, que sale pulsando el botón de la mira, y rellenará las coordenadas automáticamente).. Si el tamaño de la placa calculado por el programa a partir de la posición y tamaño de los componentes no coincide con el real, se puede corregir su tamaño desde aquí, pero no es imprescindible. Si la placa está girada hay que especificar el ángulo de giro (también se recomienda que lo rellene el programa automáticamente, usando la cámara). Pulse el botón *Guardar cambios.* Para poder usar la cámara hay que fijar previamente la placa a montar sobre el panel de trabajo de la máquina.

   En caso de que la placa contenga componentes que no deban ser montados por la máquina será preciso borrarlos desde el editor de componentes, en la parte inferior de esta ventana, o asignarles el estado colocado.

9. *Producción → Iniciar montaje*.
   Elija las opciones que desee y pulse el botón *Iniciar*. Si cometió errores en algún paso anterior, probablemente serán detectados y esta ventana no le permitirá montar la placa hasta que vuelva atrás y los subsane.

   En caso de que desee dispensar adhesivo a los componentes, marque la opción *Dispensar adhesivo* en esta ventana. Recuerde que la cantidad de adhesivo a dispensar para cada componente debe estar especificada en su alimentador correspondiente.

Nota: Se recomienda crear primero la placa donde va a importar los componentes del archivo Altium, así especificará usted desde el principio las características de la placa. Vea la siguiente sección.

Pasos para montar una placa partiendo de un fichero de Altium (Versión recomendada)

Para crear una placa partiendo de un fichero BOM de Altium, especificando desde el principio las características de la placa, puede seguir los pasos de la fig. 2, listados a continuación:

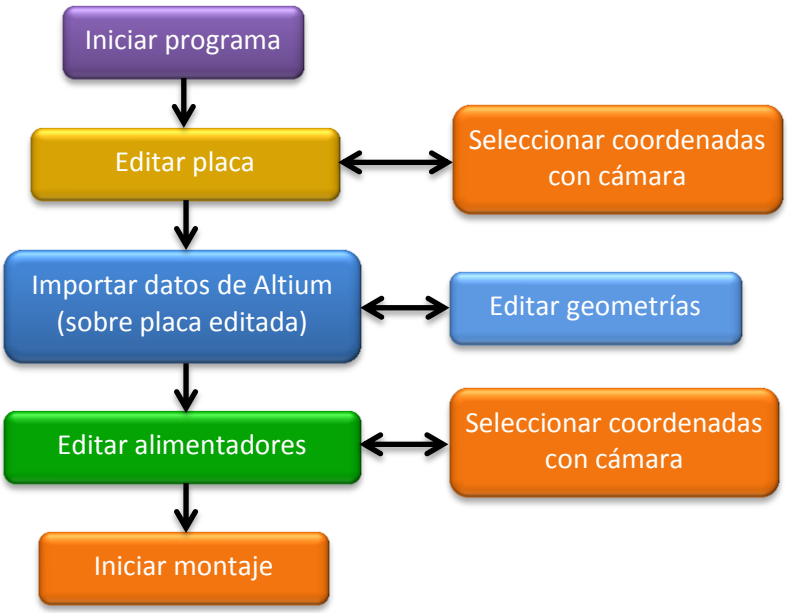


Fig. 2. Esquema de uso importando desde Altium sobre placa creada previamente.

1. *Placas* → *Editor de Placas y componentes*.
   Hay que especificar las coordenadas del origen de la placa (se recomienda hacerlo señalando el punto de origen con la cámara de la máquina, que sale pulsando el botón de la mira, y rellenará las coordenadas automáticamente). Especificar también las dimensiones de la placa. Si la placa está girada debe especificar el ángulo de giro (también se recomienda que lo rellene el programa automáticamente, usando la cámara). Pulse el botón *Crear nueva placa.* Para poder usar la cámara hay que fijar previamente la placa a montar sobre el panel de trabajo de la máquina.

2. *Archivo* → *Importar Altium*.
3. Elija el archivo de Altium en el cuadro de diálogo *Abrir Archivo* y luego pulse *Abrir*.
4. Elija la cara (*Top* o *Bottom*) y el campo que se asociará a las geometrías (si no es *Footprint*).
5. Si las geometrías de este diseño estuviesen en un archivo diferente al archivo de geometrías habitual, pulse *Cargar Geometrías* en la ventana *Importar Altium* y elija el archivo de geometrías en el cuadro de diálogo *Abrir*.
6. Edite las geometrías que falten con el editor de geometrías que sale automáticamente. Si cierra el editor de geometrías, para abrirlo de nuevo pulse el botón *Editar geometrías* o bien en *Alimentadores* → *Editor de Geometrías*. Cuando cree una nueva geometría automáticamente se actualizará el listado de geometrías detectadas en la ventana *Importar Altium*.

7. Seleccione la placa que creó en el paso 1 en la lista desplegable de la ventana *Importar Altium* y pulse *Agregar*. Se abrirá el editor gráfico de la placa, mostrando sus componentes y puede que también el editor de alimentadores y el editor de placas y componentes, si tiene configurada su apertura automática en el menú configuración. Si lo desea, puede utilizar el editor gráfico para comprobar que los componentes están ubicados correctamente.
8. *Alimentadores → Editor de Alimentadores*.
   Si todos los componentes tienen geometría, cargue los alimentadores necesarios en la plataforma de alimentadores de la máquina con los componentes que se van a montar y especifique en el programa la posición en que se ha de coger ese tipo de componente (valores *x pick* e *y pick*). Para esto se recomienda usar la cámara de la máquina, pulsando el botón de la mira, que rellenará las coordenadas automáticamente. También hay que especificar el banco en que está colocado el alimentador en caso de que no sea el primero, con qué útil hay que coger el componente, en caso de que sea necesario (depende del tamaño del componente), y si se debe activar el avance del alimentador tras coger un componente.

   En los componentes que lo requieran será necesario especificar también el valor de la altura a la que debe ser cogido, transportado y puesto sobre la placa (puede usar los controles deslizantes), además de la velocidad del movimiento del cabezal en los cuatro ejes (especialmente para componentes a los que les pueda afectar la inercia por su peso) y el centrado que pudiesen necesitar.

   Si hay componentes sin geometría, el programa les habrá asignado a todos un alimentador genérico común, creado con valores predeterminados. Tendrá que crear un alimentador por cada tipo de componente sin geometría y editar los componentes de la placa para asignando a cada uno de estos componentes su alimentador.

   Nota: En caso de que la placa contenga componentes que no deban ser montados por la máquina será preciso borrarlos desde el editor de placas y componentes o asignarles el estado *colocado*.

9. *Producción → Iniciar montaje*.
   Elija las opciones que desee y pulse el botón *Iniciar*. Si cometió errores en algún paso anterior, probablemente serán detectados y esta ventana no le permitirá montar la placa hasta que vuelva atrás y los subsane.

   En caso de que desee dispensar adhesivo a los componentes, marque la opción *Dispensar adhesivo* en esta ventana. Recuerde que la cantidad de adhesivo a dispensar para cada componente debe estar especificada en su alimentador correspondiente.

Pasos para montar una placa partiendo de cero

Para crear una placa sin partir de un fichero de Altium, debe especificar las características de los alimentadores usados y de la placa y también los datos de colocación de los componentes. Puede seguir los pasos de la fig. 3, listados a continuación:

Fig. 3. Esquema de uso partiendo de cero.

1. *Archivo → Nuevo*.
   Para vaciar la memoria de trabajos anteriores. Si se acaba de abrir el programa no es necesario. Si ya tiene suficiente experiencia con el programa y la máquina, y va a montar simultáneamente otras placas que también están en memoria, tampoco es necesario.
2. *Alimentadores → Editor de alimentadores*.
   Debe crear un alimentador por cada tipo de componente que lleva la placa. Cargue los alimentadores necesarios en la plataforma de alimentadores de la máquina con los componentes que se van a montar y especifique en el programa la posición en que se ha de coger ese tipo de componente (valores x pick e y pick). Para esto se recomienda usar la cámara de la máquina, pulsando el botón de la mira, que rellenará las coordenadas automáticamente. También hay que especificar el banco en que está colocado el alimentador en caso de que no sea el primero, con qué útil hay que coger el componente, si es necesario (depende del tamaño del componente) y si se debe activar el avance del alimentador tras coger un componente.

   En los componentes que lo requieran será necesario especificar también el valor de la altura a la que debe ser cogido, transportado y puesto sobre la placa (puede usar los controles deslizantes), además de la velocidad del movimiento del cabezal en los cuatro ejes (especialmente para componentes a los que les pueda afectar la inercia por su peso) y el centrado que pudiesen necesitar.

3. *Placas → Editor de placas y componentes*.

Debe crear una placa, especificando las coordenadas del origen (se recomienda hacerlo señalando el punto de origen con la cámara de la máquina, que sale pulsando el botón de la mira, y rellenará las coordenadas automáticamente). Especifique también las dimensiones de la placa. Si la placa está girada debe especificar el ángulo de giro (también se recomienda que lo rellene el programa automáticamente, usando la cámara). Para poder usar la cámara hay que fijar previamente la placa a montar sobre el panel de trabajo de la máquina.

A continuación debe crear todos los componentes de la placa que van a ser montados, especificando las coordenadas donde se deben poner (se puede usar la cámara para que lo haga automáticamente, pulsando el botón de la mira), el ángulo de giro y de qué alimentador hay que cogerlo.

El nombre le ayudará a localizarlo, así que se recomienda nombrarlo con su designación en el diseño, pero no es imprescindible para el montaje. Deje el estado en *no tratado*. Ante todo no debe ponerlo en el valor *colocado*, a menos que no quiera montar ese componente.

4. *Producción → Iniciar montaje*.
   Elija las opciones que desee y pulse el botón *Iniciar*. Si cometió errores en algún paso anterior, probablemente serán detectados y esta ventana no le permitirá montar la placa hasta que vuelva atrás y los subsane.
   En caso de que desee dispensar adhesivo a los componentes, marque la opción *Dispensar adhesivo* en esta ventana. Recuerde que la cantidad de adhesivo a dispensar para cada componente debe estar especificada en su alimentador correspondiente.

ANEXO III. CÓDIGO FUENTE DE LA APLICACIÓN

En este anexo se puede ver el código fuente creado por el autor para la aplicación objeto de este Proyecto Fin de Carrera. El código fuente está dividido en varios archivos, cuyo nombre se especifica al principio de cada sección de código. En el disco que acompaña a esta memoria está disponible la carpeta de la solución de Visual Studio, donde se tiene acceso a todo este código, además de los archivos generados automáticamente por el entorno de desarrollo.

```
/*_____________________________________________________________________________
 *
 * UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
 *
 * ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA
 *
 * PROYECTO FIN DE CARRERA
 *
 * HERRAMIENTA SOFTWARE DE GESTIÓN PARA LA PRODUCCIÓN DE SISTEMAS ELECTRÓNICOS CON MÁQUINAS PICK & PLACE
 *
 * Titulación: Ingeniero de Telecomunicación
 * Autor: Sergio Jesús Felipe Delgado
 * Tutor: D. Aurelio Vega Martínez
 *
 * Fecha: Junio 2016
 * _____________________________________________________________________________*/

// Program.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Diagnostics;

namespace PnPWork
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            if (aplicaciónYaEjecutándose())
            {
                if (MessageBox.Show(" Sólo una instancia de la aplicación puede controlar cada puerto
                          serie y cámara. ¿Desea continuar?", "Ya se está ejecutando la aplicación",
                          MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation) ==
                          System.Windows.Forms.DialogResult.Cancel)
                {
                    return;
                }
            }
            if (args.Length > 0) datos.rutaFichero = args[0];
            Application.Run(new FormMDI());
        }

      static bool aplicaciónYaEjecutándose()
```

```
        {
            //Comprueba si la aplicación ya estaba ejecutándose

            // Proceso actual
            Process procesoActual = Process.GetCurrentProcess();

            // Matriz de procesos
            Process[] procesosEnEjecución = Process.GetProcesses();

            // Recorre los procesos en ejecución, buscando otro con el mismo nombre
            foreach (Process proceso in procesosEnEjecución)
            {
                if ((proceso.Id != procesoActual.Id) && (proceso.ProcessName ==
                          procesoActual.ProcessName))
                {
                    return true;
                }
            }
            return false;
        }
    }
}


// FormConfiguraciónLímites
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using PnPWork.Properties;

namespace PnPWork
{
    public partial class FormConfiguraciónLímites : Form
    {
        public FormConfiguraciónLímites()
        {
            InitializeComponent();
            numericUpDownXMax.Value = datos.xMax;
            numericUpDownYMax.Value = datos.yMax;
        }

        private void buttonCosy_Click(object sender, EventArgs e)
        {
            numericUpDownXMax.Value = 3900;
            numericUpDownYMax.Value = 2610;
        }

        private void buttonMG5_Click(object sender, EventArgs e)
        {
            numericUpDownXMax.Value = 5750;
            numericUpDownYMax.Value = 4600;
        }

        private void buttonCSM84_Click(object sender, EventArgs e)
        {
            numericUpDownXMax.Value = 4570;
            numericUpDownYMax.Value = 4070;
        }

        private void buttonAceptar_Click(object sender, EventArgs e)
        {
            datos.xMax = (int)numericUpDownXMax.Value;
            datos.yMax = (int)numericUpDownYMax.Value;
            Settings.Default.xMax = datos.xMax;
            Settings.Default.yMax = datos.yMax;
            this.Close();
        }

        private void buttonCancelar_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

```
// FormMDI.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Diagnostics;
using PnPWork.Properties;

using System.Runtime.InteropServices;   //TEST

namespace PnPWork
{
    public partial class FormMDI : Form
    {
        #region Referencias a los formularios

        FormImportarAltium fia;         // Referencia para el formulario Importar Altium
        FormEditorAlimentadores fea;    // Referencia para el formulario Editor de alimentadores
        FormGeometrías fg;              // Referencia para el formulario Editor de geometrías.
        public FormEditorPlacas fep;    // Referencia para el formulario Editor de placas
        FormConfiguracionPuerto fc;     // Referencia para el formulario Configuración del puerto serie.
        FormAcercaDe fad;               // Referencia para el formulario Acerca de...

        #endregion

        public FormMDI()
        {
            InitializeComponent();

            datos.formPadre = this;
            datos.inicializar();

            // Carga las opciones predeterminadas del usuario.
            cargarGeometríasAlInicioToolStripMenuItem.Checked = Settings.Default.CargarGeometríasAlInicio;
            abrirEditorDeAlimentadoresAlCargarToolStripMenuItem.Checked =
                        Settings.Default.AbrirEditorDeAlimentadoresAlCargar;
            abrirEditorDePlacasAlCargarToolStripMenuItem.Checked =
                        Settings.Default.AbrirEditorDePlacasAlCargar;
            verPlacasAlCargarToolStripMenuItem.Checked = Settings.Default.VerPlacasAlCargar;
            interfazMDIToolStripMenuItem.Checked = !Settings.Default.InterfazMDI;
            interfazMDIToolStripMenuItem.PerformClick();

            // Carga el archivo que haya pasado Windows
            if (datos.rutaFichero != null) cargarArchivo(datos.rutaFichero);

            actualizaInterfaz();
        }

        // MENÚ ARCHIVO
        private void nuevoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (!datos.modificados || (MessageBox.Show("Se perderán los datos actuales. ¿Está seguro?",
                 "¡ATENCIÓN! Datos no guardados", MessageBoxButtons.OKCancel,
                 MessageBoxIcon.Exclamation) == System.Windows.Forms.DialogResult.OK))
            {
                // Cierra las ventanas FormPlaca abiertas.
                for (int n = 0; n < datos.placas.Length; n++)
                {
                    if (datos.placas[n].form != null) datos.placas[n].form.Close();
                }

                datos.placas = new placa[0];
                datos.alimentadores = new alimentador[0];
                datos.geometrías = new geometría[0];
                datos.rutaFichero = null;
                this.Text = "PnPWork P&P - Archivo nuevo";
                datos.modificados = false;
                actualizaInterfaz();
                mostrarOActualizarAlimentadoresSegúnConfiguración();
                mostrarOActualizarPlacasSegúnConfiguración();
```

```
        }
    }

    private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if ((datos.modificados) &&
             !(MessageBox.Show("Se perderán los datos actuales. ¿Está seguro?", "¡ATENCIÓN! Datos no
             guardados", MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation) ==
             System.Windows.Forms.DialogResult.OK))
        {
            return;
        }
        else
        {
            // Cierra las ventanas FormPlaca abiertas.
            for (int n = 0; n < datos.placas.Length; n++)
            {
                if (datos.placas[n].form != null) datos.placas[n].form.Close();
            }

            datos.placas = new placa[0];
            datos.alimentadores = new alimentador[0];
            cargarArchivo();
            datos.modificados = false;
            actualizaInterfaz();
        }
    }

    private void anexarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormAnexar fan = new FormAnexar();
        fan.ShowDialog();
        if (!((fea == null) || (fea.IsDisposed))) fea.actualizarInterfazYSeleccionar(0);
        if (!((fep == null) || (fep.IsDisposed))) fep.actualizarDatos();

        actualizaInterfaz();
    }

    private void importarAltiumToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if ((fia == null) || (fia.IsDisposed)) fia = new FormImportarAltium();
        if (interfazMDIToolStripMenuItem.Checked) fia.MdiParent = this;
        fia.Show();

        actualizaInterfaz();
    }

    private void importarGeometríasToolStripMenuItem_Click(object sender, EventArgs e)
    {
        cargarGeometrías();
        actualizaInterfaz();
    }

    private void guardarGeometríasToolStripMenuItem_Click(object sender, EventArgs e)
    {
        // Muestra el cuadro de diálogo para que el usuario elija el nombre con que se guardará.
        SaveFileDialog saveFileDialog1 = new SaveFileDialog();
        saveFileDialog1.Filter = "Archivos de geometrías (*.geo)|*.geo|Todos los archivos (*.*)|*.*";
        if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
        {
            datos.rutaGeometrías = saveFileDialog1.FileName;
            datos.guardarGeometrías(datos.rutaGeometrías);
        }
    }

    private void guardarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (datos.rutaFichero != null)
        {
            // Si el fichero tenía nombre, lo graba.
            datos.guardarFichero(datos.rutaFichero);
        }
        else
        {
            // Si no tenía nombre, llama a Guardar como...
            guardarComoToolStripMenuItem.PerformClick();
        }
    }
```

```
private void guardarComoToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Muestra el cuadro de diálogo para que el usuario elija el nombre con que se guardará.
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "Archivos de placas y alimentadores (*.pnp)|*.pnp|Todos los
                archivos (*.*)|*.*";
    if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
    {
        datos.rutaFichero = saveFileDialog1.FileName;
        this.Text = "PnPWork - " + datos.rutaFichero;
        datos.guardarFichero(datos.rutaFichero);
    }
}

private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}

// MENÚ ALIMENTADORES
private void editorDeAlimentadoresToolStripMenuItem_ClickToolStripMenuItem_Click(object sender,
         EventArgs e)
{
    if ((fea == null) || (fea.IsDisposed)) fea = new FormEditorAlimentadores();
    if (interfazMDIToolStripMenuItem.Checked) fea.MdiParent = this;
    fea.Show();
}

private void editorDeGeometríasToolStripMenuItem_Click(object sender, EventArgs e)
{
    editarGeometrías();
}

private void borrarAlimentadoresNoUsadosToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Se borrarán todos los alimentadores no usados actualmente. ¿Está
                seguro?", "Borrar alimentadores no usados", MessageBoxButtons.OKCancel,
         MessageBoxIcon.Exclamation) == System.Windows.Forms.DialogResult.OK)
    {
        for (int a = datos.alimentadores.Length - 1; a >= 0 ; a--)
        {
            if (datos.quitarAlimentadorSiNoUsadoYReajustarComponentes(a))
                datos.modificados = true;
        }

        if (!((fea == null) || (fea.IsDisposed)))
         fea.actualizarInterfazYSeleccionar(0);
    }
    actualizaInterfaz();
}

private void borrarGeometríasToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Se borrarán todas las geometrías. ¿Está seguro?",
            "Borrar geometrías", MessageBoxButtons.OKCancel,
            MessageBoxIcon.Exclamation) == System.Windows.Forms.DialogResult.OK)
    {
        datos.geometrías = new geometría[0];
        datos.rutaGeometrías = "";
    }
    actualizaInterfaz();
}

// MENÚ PLACAS
private void editorDePlacasToolStripMenuItem_Click(object sender, EventArgs e)
{
    if ((fep == null) || (fep.IsDisposed)) fep = new FormEditorPlacas();
    if (interfazMDIToolStripMenuItem.Checked) fep.MdiParent = this;
    fep.Show();
    fep.actualizarDatos();
}

private void verPlaca_Click(object sender, EventArgs e)
{
    // Comprueba qué placa se eligió para mostrarla.
    for (int n = 0; n < datos.placas.Length; n++)
    {
        if (verPlacaToolStripMenuItem.DropDownItems[n].Pressed)
        {
```

```
                mostrarPlaca(n);
                break;
            }
        }
    }

    private void borrarPlacasToolStripMenuItem1_Click(object sender, EventArgs e)
    {
        // Se eliminan las placas a través de su método
        while (datos.placas.Length > 0)
        {
            datos.quitarPlaca(0);
        }

        // Si el editor de placas está abierto, se actualiza para evitar inconsistencias.
        if (!((fep == null) || (fep.IsDisposed))) fep.actualizarDatos();

        actualizaInterfaz();
    }

    // MENÚ PRODUCCIÓN
    private void iniciarProducciónToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormProduccion fpr = new FormProduccion();
        fpr.ShowDialog();       // Muestra el formulario de forma modal
    }

    // MENÚ CONFIGURACIÓN
    private void puertoSerieToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if ((fc == null) || (fc.IsDisposed)) fc = new FormConfiguracionPuerto();
        if (interfazMDIToolStripMenuItem.Checked) fc.MdiParent = this;
        fc.Show();
    }

    private void configuraciónCámaraToolStripMenuItem_Click(object sender, EventArgs e)
    {
        FormCámara fcam = new FormCámara();
        if (interfazMDIToolStripMenuItem.Checked) fcam.MdiParent = this;
        fcam.Show();
    }

    private void geometríasAutomáticasToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (cargarGeometríasAlInicioToolStripMenuItem.Checked)
        {
            cargarGeometríasAlInicioToolStripMenuItem.Checked = false;
        }
        else
        {
            cargarGeometríasAlInicioToolStripMenuItem.Checked = true;
        }
        Settings.Default.CargarGeometríasAlInicio = cargarGeometríasAlInicioToolStripMenuItem.Checked;
    }

    private void abrirEditorDeAlimentadoresAlCargarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (abrirEditorDeAlimentadoresAlCargarToolStripMenuItem.Checked)
        {
            abrirEditorDeAlimentadoresAlCargarToolStripMenuItem.Checked = false;
        }
        else
        {
            abrirEditorDeAlimentadoresAlCargarToolStripMenuItem.Checked = true;
        }
        Settings.Default.AbrirEditorDeAlimentadoresAlCargar =
                    abrirEditorDeAlimentadoresAlCargarToolStripMenuItem.Checked;
    }

    private void abrirEditorDePlacasAlCargarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (abrirEditorDePlacasAlCargarToolStripMenuItem.Checked)
        {
            abrirEditorDePlacasAlCargarToolStripMenuItem.Checked = false;
        }
        else
        {
            abrirEditorDePlacasAlCargarToolStripMenuItem.Checked = true;
        }
```

```
        Settings.Default.AbrirEditorDePlacasAlCargar =
                                abrirEditorDePlacasAlCargarToolStripMenuItem.Checked;
    }

    private void verPlacasAlCargarToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (verPlacasAlCargarToolStripMenuItem.Checked)
        {
            verPlacasAlCargarToolStripMenuItem.Checked = false;
        }
        else
        {
            verPlacasAlCargarToolStripMenuItem.Checked = true;
        }
        Settings.Default.VerPlacasAlCargar = verPlacasAlCargarToolStripMenuItem.Checked;
    }

    private void interfazMDIToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (interfazMDIToolStripMenuItem.Checked)
        {
            // Configura la interfaz con ventanas independientes.
            interfazMDIToolStripMenuItem.Checked = false;
            Settings.Default.EstadoInterfaz = this.WindowState;
            this.WindowState = FormWindowState.Normal;
            Settings.Default.AltoInterfaz = this.Height;
            Settings.Default.AnchoInterfaz = this.Width;
            this.Height = 62;
            this.Width = 560;
        }
        else
        {
            // Configura la interfaz MDI.
            interfazMDIToolStripMenuItem.Checked = true;
            this.Height = Settings.Default.AltoInterfaz;
            this.Width = Settings.Default.AnchoInterfaz;
            this.WindowState = Settings.Default.EstadoInterfaz;
        }

        // Actualiza las ventanas hija.
        Form padre = interfazMDIToolStripMenuItem.Checked ? this : null;
        for (int n = 1; n < Application.OpenForms.Count; n++)
            while (Application.OpenForms[n].MdiParent != padre)
               Application.OpenForms[n].MdiParent = padre;
    }

    // MENÚ AYUDA

    private void acercaDeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if ((fad == null) || (fad.IsDisposed)) fad = new FormAcercaDe();
        if (interfazMDIToolStripMenuItem.Checked) fad.MdiParent = this;
        fad.Show();
    }

    private void manualToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Process.Start(Path.Combine(Application.StartupPath, "Manual del usuario.pdf"));
    }

    private void guíaRápidaToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Process.Start(Path.Combine(Application.StartupPath, "Guía rápida.pdf"));
    }

    // OTROS
    private void cargarArchivo()
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Archivos de composición de placas (*.pnp)|*.pnp|Todos los archivos
                    (*.*)|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            datos.rutaFichero = openFileDialog.FileName;
            cargarArchivo(datos.rutaFichero);
        }
    }

    private int cargarArchivo(string rutaFichero)
```

```
{
    int error = datos.cargarFichero(rutaFichero);
    if (error != 0)
    {
        if (error == -1) MessageBox.Show("No se pudo leer el archivo" + error, "Información",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        else MessageBox.Show("Error en la línea" + error, "Información", MessageBoxButtons.OK,
                MessageBoxIcon.Exclamation);
        rutaFichero = null;
    }
    else
    {
        this.Text = "PnPWork - " + rutaFichero;

        mostrarOActualizarAlimentadoresSegúnConfiguración();
        mostrarOActualizarPlacasSegúnConfiguración();
        mostrarPlacasSegúnConfiguración();
    }

    actualizaInterfaz();
    return error;
}

public void mostrarOActualizarAlimentadoresSegúnConfiguración()
{
    if (Settings.Default.AbrirEditorDeAlimentadoresAlCargar)
    {
        if ((fea == null) || (fea.IsDisposed)) fea = new FormEditorAlimentadores();
        if (interfazMDIToolStripMenuItem.Checked) fea.MdiParent = this;
        fea.Show();
    }
    if (!((fea == null) || (fea.IsDisposed))) fea.actualizarInterfazYSeleccionar(0);
}

public void mostrarOActualizarPlacasSegúnConfiguración()
{
    if (Settings.Default.AbrirEditorDePlacasAlCargar)
    {
        if ((fep == null) || (fep.IsDisposed)) fep = new FormEditorPlacas();
        if (interfazMDIToolStripMenuItem.Checked) fep.MdiParent = this;
        fep.Show();
    }
    if (!((fep == null) || (fep.IsDisposed))) fep.actualizarInterfazDePlacasYSeleccionar(0);
}

private void mostrarPlacasSegúnConfiguración()
{
    if (Settings.Default.VerPlacasAlCargar)
    {
        for (int n = 0; n < datos.placas.Length; n++)
        {
            mostrarPlaca(n);
        }
    }
}

public void cargarGeometrías()
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Archivos de geometrías (*.geo)|*.geo|Todos los archivos (*.*)|*.*";
    if (openFileDialog.ShowDialog(this) == DialogResult.OK)
    {
        datos.rutaGeometrías = openFileDialog.FileName;
        int error = datos.cargarGeometrías(datos.rutaGeometrías);
        if (error != 0)
        {
            if (error == -1) MessageBox.Show("No se pudo leer el archivo.", "Error",
                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            else MessageBox.Show("Error en la línea " + error, "Error al leer el archivo",
                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            datos.rutaFichero = null;
        }
        else
        {
            editarGeometrías();
        }
    }

}
```

```
public void editarGeometrías()
{
    if ((fg == null) || (fg.IsDisposed)) fg = new FormGeometrías();
    if (interfazMDIToolStripMenuItem.Checked) fg.MdiParent = this;
    fg.Show();
    fg.Refrescar();
}

public void actualizaInterfaz()
{
    // Actualiza el menú Alimentadores.
    if (datos.alimentadores.Length > 0) borrarAlimentadoresToolStripMenuItem.Enabled = true;
    else borrarAlimentadoresToolStripMenuItem.Enabled = false;

    if (datos.geometrías.Length > 0) borrarGeometríasToolStripMenuItem.Enabled = true;
    else borrarGeometríasToolStripMenuItem.Enabled = false;

    // Actualiza el menú Placas.
    if (datos.placas.Length > 0)
    {
        verPlacaToolStripMenuItem.Enabled = true;
        borrarPlacasToolStripMenuItem.Enabled = true;
        iniciarProducciónToolStripMenuItem.Enabled = true;
    }
    else
    {
        verPlacaToolStripMenuItem.Enabled = false;
        borrarPlacasToolStripMenuItem.Enabled = false;
        iniciarProducciónToolStripMenuItem.Enabled = false;
    }

    // Actualiza las opciones del submenú Ver Placa.
    verPlacaToolStripMenuItem.DropDownItems.Clear();
    for (int n = 0; n < datos.placas.Length; n++)
    {
        ToolStripMenuItem verPlaca = new ToolStripMenuItem();
        verPlaca.Text = "&" + (n + 1).ToString() + " -> " + datos.placas[n].nombre;
        verPlaca.Click += new EventHandler(verPlaca_Click);
        verPlacaToolStripMenuItem.DropDownItems.Add(verPlaca);
    }

    // Actualiza el menú producción.
    if ((datos.placas.Length > 0) && (datos.alimentadores.Length > 0))
    {
        iniciarProducciónToolStripMenuItem.Enabled = true;
    }
    else
    {
        iniciarProducciónToolStripMenuItem.Enabled = false;
    }
}

public void actualizaAltium()
{
    if (!((fia == null) || (fia.IsDisposed))) fia.mostrarDatos();
}

public void mostrarPlaca(int n)
{
    if (datos.placas.Length > n)
    {
        if ((datos.placas[n].form == null) || (datos.placas[n].form.IsDisposed))
                datos.placas[n].form = new FormVerPlaca();
        datos.placas[n].form.MdiParent = datos.formPadre;
        datos.placas[n].form.Show();
        datos.placas[n].form.representarPlaca(n);
        datos.placas[n].form.BringToFront();
    }
}

private void FormMDI_FormClosing(object sender, FormClosingEventArgs e)
{
    if ((datos.modificados) &&
         !(MessageBox.Show("ATENCIÓN: Va a salir sin guardar datos.\n\n" +
         "Aceptar saldrá del programa sin guardar los datos.\n" +
         "Cancelar volverá al programa y podrá guardar los datos desde el menú Archivo",
         "¡ATENCIÓN! Datos no guardados", MessageBoxButtons.OKCancel,
         MessageBoxIcon.Warning) == System.Windows.Forms.DialogResult.OK))
```

```
            {
                e.Cancel = true;
            }
            else
            {
                if ((datos.puertoSerie != null) && datos.puertoSerie.IsOpen) datos.puertoSerie.Close();
            }

            if (datos.cámara != null) datos.cámara.SignalToStop();
            Settings.Default.NombreCámara = datos.nombreCámara;

            // Guarda la configuración del programa en las propiedades de la aplicación.
            if (Settings.Default.CargarGeometríasAlInicio)
            {
                if ((datos.rutaGeometrías != null) && (datos.rutaGeometrías != ""))
                {
                    Settings.Default.RutaGeometrías = datos.rutaGeometrías;
                    datos.guardarGeometrías(datos.rutaGeometrías);
                }
            }
            Settings.Default.NombrePuertoSerie = datos.nombrePuertoSerie;
            Settings.Default.InterfazMDI = interfazMDIToolStripMenuItem.Checked;
            // Si el interfaz está en estado normal, guarda las dimensiones.
            // Así, si se cierra maximizado o minimizado, no se borran las últimas dimensiones normales.
            if (this.WindowState == FormWindowState.Normal)
            {
                Settings.Default.AltoInterfaz = this.Height;
                Settings.Default.AnchoInterfaz = this.Width;
            }
            Settings.Default.EstadoInterfaz = this.WindowState;
            Settings.Default.Save();
        }
    }
}


// FormAnexar.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormAnexar : Form
    {
        public FormAnexar()
        {
            InitializeComponent();
        }

        private void checkBoxAlimentadores_CheckedChanged(object sender, EventArgs e)
        {
            checkBoxActualizar.Enabled = checkBoxPlacas.Checked && checkBoxAlimentadores.Checked;
        }

        private void checkBoxPlacas_CheckedChanged(object sender, EventArgs e)
        {
            checkBoxComponentes.Enabled = checkBoxPlacas.Checked;
            checkBoxActualizar.Enabled = checkBoxPlacas.Checked && checkBoxAlimentadores.Checked;
        }

        private void buttonCargar_Click(object sender, EventArgs e)
        {
            if (checkBoxAlimentadores.Checked || checkBoxPlacas.Checked)
            {
                OpenFileDialog openFileDialog = new OpenFileDialog();
                openFileDialog.Filter = "Archivos de composición de placas (*.pnp)|*.pnp|Todos los
                          archivos (*.*)|*.*";
                if (openFileDialog.ShowDialog(this) == DialogResult.OK)
                {
                    int error = datos.cargarFichero(openFileDialog.FileName,
                          checkBoxAlimentadores.Checked, checkBoxPlacas.Checked,
                          checkBoxComponentes.Checked, checkBoxActualizar.Checked);
                    if (error != 0)
```

```
                    {
                        MessageBox.Show("Error en el archivo en la línea " + error, "Error en el archivo",
                                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                    }
                    else
                    {
                        datos.modificados = true;
                        this.Close();
                    }
                }
            }
            else
            {
                MessageBox.Show("Seleccione antes algún dato que cargar", "Información",
                         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
        }
    }
}


// FormImportarAltium
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.IO;
using PnPWork.Properties;

namespace PnPWork
{
    public partial class FormImportarAltium : Form
    {
        struct componenteAltium
        {
            public string nombre;       // Nombre del componente.
            public string geometría;    // Elemento elegido para localizar la geometría (Comment)
            public int x;               // Coordenadas del centro del componente con respecto a la placa
            public int y;               //
            public int t;               // Ángulo cegesimal de giro levógiro del componente.
            public bool caraTop;        // Cara en la que está el componente True=Top, False=Bottom.
        }

        #region Campos

        componenteAltium[] componentesAltium;
        int alimentadorBase;
        int índiceNombre = 0;
        int índiceGeometría = 0;
        int índiceX = 0;
        int índiceY = 0;
        int índiceGiro = 0;
        int índiceCara = 0;
        bool coordenadasYNegativas = false;
        int altoInterfaz;
        int anchoInterfaz;
        DataGridViewCellStyle celdaResaltada = new DataGridViewCellStyle();
        DataGridViewCellStyle celdaBottom = new DataGridViewCellStyle();
        DataGridViewCellStyle celdaNormal = new DataGridViewCellStyle();

        #endregion

        public FormImportarAltium()
        {
            InitializeComponent();
            altoInterfaz = this.Height;
            anchoInterfaz = this.Width;
            celdaResaltada.ForeColor = Color.Red;
            celdaBottom.ForeColor = Color.Blue;
        }

        private void buttonAbrirFichero_Click(object sender, EventArgs e)
        {
            // Carga ficheros de Altium y los interpreta
```

```
OpenFileDialog openFileDialog = new OpenFileDialog();
componentesAltium = new componenteAltium[0];
openFileDialog.Filter = "Archivos Pick & Place de Altium (*.csv)|*.csv|Todos los archivos
            (*.*)|*.*";
if (openFileDialog.ShowDialog(this) == DialogResult.OK)
{
    string rutaFichero = openFileDialog.FileName;
    ArrayList líneasConErrores = new ArrayList();
    FileInfo fi = new FileInfo(rutaFichero);
    textBoxRutaFichero.Text = rutaFichero;

    if (fi.Exists)
    {
        StreamReader sr = fi.OpenText();        // Abre el fichero como archivo de texto
        string linea;
        char[] separador = { '"', ','};
        bool encabezadoEncontrado = false;
        int numLinea = 0;
        string[] elementos = new string[20];
        int numElementosEncabezado = 0;
        richTextBox1.Clear();
        richTextBox1.SelectionColor = Color.Black;
        dataGridViewComponentes.Rows.Clear();
        dataGridViewComponentes.Columns.Clear();

        // Busca el principio de los datos útiles
        while (!sr.EndOfStream && !encabezadoEncontrado)
        {
            linea = sr.ReadLine();
            numLinea++;
            elementos = linea.Split(separador, 50, StringSplitOptions.RemoveEmptyEntries);
            if (localizaElementosEncabezadoAltium(elementos))
            {
                encabezadoEncontrado = true;
                numElementosEncabezado = elementos.Length;
                inicializaGrid(elementos);
                inicializaComboBoxGeometría(elementos);

                separador = new char[] { '"' };

                richTextBox1.SelectionColor = Color.Green;
                richTextBox1.AppendText(numLinea + " (" + (elementos.Length).ToString() +
                      "): " + linea + "\n");
                richTextBox1.SelectionColor = Color.Black;
            }
            else
            {
                richTextBox1.AppendText(numLinea + ":  " + linea + "\n");
            }
        }

        // Lee los datos de los componentes
        while (!sr.EndOfStream)
        {
            linea = sr.ReadLine();
            numLinea++;
            // Separa los elementos por las comillas
            elementos = linea.Split(separador, 20, StringSplitOptions.RemoveEmptyEntries);
            // Quita las comas
            elementos = Array.FindAll(elementos, noEsComa).ToArray();
            if (elementos.Length == numElementosEncabezado)
            {

                componenteAltium c;    // Crea una variable local para este componente.
                c.nombre = elementos[índiceNombre];
                c.geometría = elementos[índiceGeometría];
                c.x = (int)(Math.Round(convierteNumero(elementos[índiceX]) * 10));
                c.y = (int)(Math.Round(convierteNumero(elementos[índiceY]) * 10));
                c.t = (int)(Math.Round((convierteNumero(elementos[índiceGiro]) * 400 / 360) %
                      400));// Convierte el ángulo a centesimal.
                c.caraTop = elementos[índiceCara][0] == 'T'

                // Agrega el componente al array:
                componenteAltium[] c2 = new componenteAltium[componentesAltium.Length + 1];
                componentesAltium.CopyTo(c2, 0);
                c2[componentesAltium.Length] = c;
                componentesAltium = (componenteAltium[])c2.Clone();

                AñadirLíneaAlDataGrid(elementos, c.caraTop);
```

```
                    richTextBox1.SelectionColor = Color.Black;
                }
                else
                {
                    if (elementos.Length > 0)
                    {
                        líneasConErrores.Add(numLinea);
                        richTextBox1.SelectionColor = Color.Red;
                    }
                }
                richTextBox1.AppendText(numLinea + " (" + (elementos.Length).ToString() + "):  " +
                          linea + "\n");
            }
            sr.Close();
        }

        if (líneasConErrores.Count > 0)
        {
            string texto = "Error en el archivo P&P de Altium en la";
            if (líneasConErrores.Count == 1)
            {
                texto += " línea: ";
            }
            else
            {
                texto += "s líneas: ";
            }
            for (int n = 0; n < líneasConErrores.Count; n++)
            {
                texto += líneasConErrores[n];
                if (n < líneasConErrores.Count - 2) texto += ", ";
                if (n == líneasConErrores.Count - 2) texto += " y ";
            }
            texto += ".";
            MessageBox.Show(texto + "\n\nLos datos incorrectamente formateados no fueron
                  procesados.\nIMPORTANTE: Si prosigue, dichos datos deberán ser insertados
                  manualmente.", "¡Atención!", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            rutaFichero = null;
        }
    }

    if (componentesAltium.Length > 0 && coordenadasAltiumCorrectas())
    {
        numericUpDownAltoPlaca.Minimum = 0;
        numericUpDownAltoPlaca.Value = 0;
        radioButtonOrigenArriba.Checked = coordenadasYNegativas;
        mostrarDatos();
        buttonAgregar.Enabled = true;
    }
    else
    {
        buttonAgregar.Enabled = false;
        if (!coordenadasAltiumCorrectas())
        {
            MessageBox.Show("Se ha detectado un error en las coordenadas de los
                  componentes.\n\nCompruebe en Altium que el origen de coordenadas de la placa ha
                  sido establecido en la esquina superior izquierda o la esquina inferior
                  izquierda.\nCompruebe que no hay componentes fuera de los límites de la placa.",
                  "¡Atención!", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
        else
        {
            MessageBox.Show("No se han encontrado los datos necesarios para importar el
                  archivo.\n\nEs necesario que el archivo importado tenga " +
                  Settings.Default.EncabezadoAltiumParaNombre + ", " +
                  Settings.Default.EncabezadoAltiumParaGeometría + ", " +
                  Settings.Default.EncabezadoAltiumParaX + ", " +
                  Settings.Default.EncabezadoAltiumParaY + ", " +
                  Settings.Default.EncabezadoAltiumParaGiro + " y " +
                  Settings.Default.EncabezadoAltiumParaCapa + ".",
                  "¡Atención!", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
}

bool noEsComa(string cadena)
{
    return cadena != ",";
}
```

```
bool localizaElementosEncabezadoAltium(string[] arrayDondeBuscar)
{
    índiceNombre = índiceCadena(Settings.Default.EncabezadoAltiumParaNombre, arrayDondeBuscar);
    índiceGeometría = índiceCadena(Settings.Default.EncabezadoAltiumParaGeometría,
                arrayDondeBuscar);
    índiceX = índiceCadena(Settings.Default.EncabezadoAltiumParaX, arrayDondeBuscar);
    índiceY = índiceCadena(Settings.Default.EncabezadoAltiumParaY, arrayDondeBuscar);
    índiceGiro = índiceCadena(Settings.Default.EncabezadoAltiumParaGiro, arrayDondeBuscar);
    índiceCara = índiceCadena(Settings.Default.EncabezadoAltiumParaCapa, arrayDondeBuscar);
    if ((índiceNombre == -1) || (índiceGeometría == -1) || (índiceX == -1) || (índiceY == -1) ||
                (índiceGiro == -1) || (índiceCara == -1)) return false;
    else return true;
}

int índiceCadena(string cadenaBuscada, string[] arrayDondeBuscar)
{
    for (int n = 0; n < arrayDondeBuscar.Length; n++)
    {
        if (arrayDondeBuscar[n].IndexOf(cadenaBuscada) >= 0) return n;
    }
    return -1;
}

void inicializaGrid(string[] elementos)
{
    dataGridViewComponentes.ColumnCount = elementos.Length + 2;
    for (int n = 0; n < elementos.Length; n++)
    {
        dataGridViewComponentes.Columns[n].Name = elementos[n];
    }
    dataGridViewComponentes.Columns[elementos.Length].Name = "Geometría";
    dataGridViewComponentes.Columns[elementos.Length + 1].Name = "OrdenCosy";
    dataGridViewComponentes.Columns["OrdenCosy"].Visible = false;
}

void AñadirLíneaAlDataGrid(string[] elementos, bool caraTop)
{
    dataGridViewComponentes.Rows.Add(elementos);
    dataGridViewComponentes.Rows[dataGridViewComponentes.RowCount - 1].Cells["OrdenCosy"].Value =
                dataGridViewComponentes.RowCount - 1;
    if (!caraTop) dataGridViewComponentes.Rows[dataGridViewComponentes.RowCount –
                1].DefaultCellStyle = celdaBottom;
}

private double convierteNumero(string dato)
{
    double factorConversión = 1;
    int cifras = dato.Length;

    // Comprueba si contiene unidades métricas para cortarlas:
    if (dato.IndexOf("mm") != -1)
    {
        cifras = dato.IndexOf("mm");
    }

    // Comprueba si contiene unidades imperiales para cortarlas y convertirlas:
    if (dato.IndexOf("mil") != -1)
    {
        cifras = dato.IndexOf("mil");
        factorConversión = 0.0254;
    }

    // Elimina la unidades si las había y cambia el punto decimal por coma:
    string dígitos = dato.Substring(0, cifras).Replace('.', ',');

    // Convierte el texto a valor numérico y lo devuelve:
    double resultado;
    try
    {
        resultado = Convert.ToDouble(dígitos) * factorConversión;
    }
    catch
    {
        resultado = 0;
    }
    return resultado;
}
```

```
bool coordenadasAltiumCorrectas()
{
    // Busca el primer elemento con coordenada 'y' no cero
    int n = 0;
    while (n < componentesAltium.Length)
    {
        if (componentesAltium[n].x < 0) return false;   // De paso va comprobando también la 'x'.
        if (Math.Sign(componentesAltium[n].y) != 0) break;
        n++;
    }

    // Guarda el primer signo que encontró
    int signoAnterior = 0;
    if (n < componentesAltium.Length)  signoAnterior = Math.Sign(componentesAltium[n].y);

    // Comprueba que el resto de signos sean iguales
    while (n < componentesAltium.Length)
    {
        if (componentesAltium[n].x < 0) return false;   // De paso va comprobando también la 'x'.
        if (Math.Sign(componentesAltium[n].y) != signoAnterior) return false;
        n++;
    }

    // Actualiza el signo de las coordenadas
    coordenadasYNegativas = signoAnterior == -1 ? true : false;
    // Si no encontró ninguna 'x' negativa ni 'y' con diferente signo, devuelve OK
    return true;
}


public void mostrarDatos()
{
    int geometríaActual;

    if (componentesAltium.Length == 0) return;

    // Actualiza el alto de la placa
    decimal alturaMínima = Math.Min((decimal)yMáxima() / 10, numericUpDownAltoPlaca.Maximum);
    numericUpDownAltoPlaca.Minimum = 0;
    numericUpDownAltoPlaca.Value = Math.Max(alturaMínima, numericUpDownAltoPlaca.Value);
    numericUpDownAltoPlaca.Minimum = alturaMínima;

    rellenaComboPlacas();

    // Actualiza las geometrías
    string campoGeometría;
    if (comboBoxAsociarGeometrías.Items.Count > 0) campoGeometría =
                comboBoxAsociarGeometrías.SelectedItem.ToString();
    else return;

    for (int n = 0; n < componentesAltium.Length; n++)
    {
        int fila = líneaGrid(n);
        if (componentesAltium[n].caraTop == radioButtonTop.Checked)
        {
            dataGridViewComponentes.Rows[fila].Visible = true;
            // Recupera el valor de geometría para este componente del dataGrid
            componentesAltium[n].geometría =
                  dataGridViewComponentes.Rows[fila].Cells[campoGeometría].Value.ToString();
            // Busca el campo asociado a las geometrías del componente en el array de geometrías.
            geometríaActual = buscaGeometría(componentesAltium[n].geometría);
            if (geometríaActual == -1)
            {
                dataGridViewComponentes.Rows[fila].Cells["Geometría"].Value = "Sin geometría";
                dataGridViewComponentes.Rows[fila].Cells["Geometría"].Style = celdaResaltada;
            }
            else
            {
                dataGridViewComponentes.Rows[fila].Cells["Geometría"].Value =
                          datos.geometrías[geometríaActual].nombre;
                dataGridViewComponentes.Rows[fila].Cells["Geometría"].Style =
                          dataGridViewComponentes.DefaultCellStyle;
            }
        }
        else dataGridViewComponentes.Rows[fila].Visible = false;
    }

    // Alterna el color de fondo de las líneas visibles
    bool colorNormal = true;
```

```
        for (int n = 0; n < componentesAltium.Length; n++)
        {
            if (dataGridViewComponentes.Rows[n].Visible)
            {
                if (colorNormal)
                {
                    dataGridViewComponentes.Rows[n].DefaultCellStyle =
                              dataGridViewComponentes.DefaultCellStyle;
                }
                else
                {
                    dataGridViewComponentes.Rows[n].DefaultCellStyle =
                              dataGridViewComponentes.AlternatingRowsDefaultCellStyle;
                }
                colorNormal = !colorNormal;
            }
        }
    }

    private void rellenaComboPlacas()
    {
        int elecciónAnterior = 0;
        if (comboBoxPlaca.SelectedIndex > 0) elecciónAnterior = comboBoxPlaca.SelectedIndex;
        comboBoxPlaca.Items.Clear();
        comboBoxPlaca.Items.Add("Crear nueva placa");
        for (int n = 0; n < datos.placas.Length; n++)
        {
            comboBoxPlaca.Items.Add((n + 1).ToString() + " -> " + datos.placas[n].nombre);
        }
        comboBoxPlaca.SelectedIndex = elecciónAnterior;
    }

    int yMáxima()
    {
        int máximo = 0;
        for (int n = 0; n < componentesAltium.Length; n++)
        {
            máximo = Math.Max(máximo, Math.Abs(componentesAltium[n].y));
        }
        return máximo;
    }

    void inicializaComboBoxGeometría(string[] elementos)
    {
        for (int n = 0; n < elementos.Length; n++)
        {
            comboBoxAsociarGeometrías.Items.Add(elementos[n]);
            if (elementos[n] == Settings.Default.EncabezadoAltiumParaGeometría)
                      comboBoxAsociarGeometrías.SelectedIndex = n;
        }
    }

    int líneaGrid(int líneaOriginalBuscada)
    {
        for (int n = 0; n < dataGridViewComponentes.RowCount; n++)
        {
            if ((int)dataGridViewComponentes.Rows[n].Cells["OrdenCosy"].Value == líneaOriginalBuscada)
                      return n;
        }
        return -1;
    }

    private int buscaGeometría(string datoBuscado)
    {
        for (int n = 0; n < datos.geometrías.Length; n++)
        {
            if (datos.geometrías[n].nombre == datoBuscado) return n;
        }
        return -1;
    }


    private void buttonAgregar_Click(object sender, EventArgs e)
    {
        // Comprueba si hay componentes en la cara elegida.
        bool hayComponentes = false;
        for (int n = 0; n < componentesAltium.Length; n++)
        {
            if (componentesAltium[n].caraTop == radioButtonTop.Checked)
```

```
            {
                hayComponentes = true;
                break;
            }
        }

        if (hayComponentes)
        {
            // Si hay componentes, los procesa.
            int pl;

            if ((comboBoxPlaca.SelectedIndex > 0) && (comboBoxPlaca.SelectedIndex <=
                      datos.placas.Length))
            {
                pl = comboBoxPlaca.SelectedIndex - 1;   // Selecciona una placa existente
            }
            else
            {
                placa p;    //Crea una nueva placa
                p.nombre = "Placa Importada Altium";
                p.x = 0;
                p.y = 0;
                p.t = 0;
                p.anchoX = 0;
                p.anchoY = (int)(numericUpDownAltoPlaca.Value * 10);
                p.componentes = new componente[0];  // Crea un vector de componentes vacío.
                p.form = null;
                datos.añadirPlaca(p);
                pl = datos.placas.Length - 1;   // Selecciona la nueva placa
            }

            alimentador a = new alimentador("Sin geometría", 1000, 2600, 20, 20, 190);
                         datos.añadirAlimentador(a);
            alimentadorBase = datos.alimentadores.Length - 1;
            int origen = radioButtonOrigenArriba.Checked ? 0 : datos.placas[pl].anchoY;

            for (int n = 0; n < componentesAltium.Length; n++)
            {
                //Comprueba si el componente está en la cara elegida para procesarlo.
                if (componentesAltium[n].caraTop == radioButtonTop.Checked)
                {
                    // Crea un nuevo componente para la placa con los datos del componente actual.
                    componente c = new componente();
                    c.x = componentesAltium[n].x;
                    c.y = origen - componentesAltium[n].y;
                    c.t = componentesAltium[n].t;
                    c.nombre = componentesAltium[n].nombre;

                    // Busca la geometría del componente
                    int g = buscaGeometría(componentesAltium[n].geometría);
                    if (g == -1)
                    {
                        // Si no tiene geometría, le asigna el alimentador genérico
                        c.alimentador = alimentadorBase;
                    }
                    else
                    {
                        // Si tiene geometría, comprobamos si ya hay alimentador para ella.
                        int al = buscaAlimentador(datos.geometrías[g].nombre);
                        if (al == -1)
                        {
                            // Si aún no se le había creado un alimentador, se le crea y se asigna.
                            a = new alimentador(datos.geometrías[g].nombre, 1000, 2600,
                                                datos.geometrías[g].dimX, datos.geometrías[g].dimY,
                                                datos.zMax - datos.geometrías[g].dimZ);
                            datos.añadirAlimentador(a);
                            c.alimentador = datos.alimentadores.Length - 1;
                        }
                        else
                        {
                            // Si ya se le había creado un alimentador, lo reutiliza
                            c.alimentador = al;
                        }
                    }

                    //Amplía la placa para que quepa el componente si es necesario.
                    if (c.x + datos.alimentadores[c.alimentador].anchoX > datos.placas[pl].anchoX)
                        datos.placas[pl].anchoX = c.x + datos.alimentadores[c.alimentador].anchoX * 2;
                    if (c.y + datos.alimentadores[c.alimentador].anchoY > datos.placas[pl].anchoY)
```

```
                    datos.placas[pl].anchoY = c.y + datos.alimentadores[c.alimentador].anchoY * 2;

                datos.placas[pl].añadirComponente(c);
            }
        }

        // Actualiza el interfaz y muestra los editores.
        datos.formPadre.actualizaInterfaz();
        datos.formPadre.mostrarOActualizarAlimentadoresSegúnConfiguración();
        datos.formPadre.mostrarOActualizarPlacasSegúnConfiguración();
        datos.formPadre.mostrarPlaca(pl);
        this.Close();
    }
    else
    {
        // Si no hay componentes, avisa al usuario.
        MessageBox.Show("No hay componentes en la cara elegida.", "Información",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

private int buscaAlimentador(string datoBuscado)
{
    for (int n = alimentadorBase; n < datos.alimentadores.Length; n++)
    {
        if (datos.alimentadores[n].tipoComponente == datoBuscado) return n;
    }
    return -1;
}

private void FormAltium_Shown(object sender, EventArgs e)
{
    buttonAbrirFichero.PerformClick();
}

private void FormAltium_Resize(object sender, EventArgs e)
{
    colocarControles();
}

private void colocarControles()
{
    int difX = this.Width - anchoInterfaz;
    int difY = this.Height - altoInterfaz;
    textBoxRutaFichero.Width += difX;
    richTextBox1.Width += difX;
    dataGridViewComponentes.Width += difX;
    dataGridViewComponentes.Height += difY;
    groupBoxControles.Left += difX;
    anchoInterfaz += difX;
    altoInterfaz += difY;
}

private void radioButtonTop_CheckedChanged(object sender, EventArgs e)
{
    mostrarDatos();
}

private void radioButtonBottom_CheckedChanged(object sender, EventArgs e)
{
    mostrarDatos();
}

private void comboBoxAsociarGeometrías_SelectedIndexChanged(object sender, EventArgs e)
{
    mostrarDatos();
}

private void buttonCargarGeometrías_Click(object sender, EventArgs e)
{
    datos.formPadre.cargarGeometrías();
    mostrarDatos();
}

private void buttonEditarGeometrías_Click(object sender, EventArgs e)
{
    datos.formPadre.editarGeometrías();
}
```

```
        private void dataGridViewComponentes_ColumnHeaderMouseClick(object sender,
                DataGridViewCellMouseEventArgs e)
        {
            mostrarDatos();
        }

        private void comboBoxPlaca_Click(object sender, EventArgs e)
        {
            rellenaComboPlacas();
        }
    }
}


// FormEditorAlimentadores.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormEditorAlimentadores : Form
    {

        public FormEditorAlimentadores()
        {
            InitializeComponent();
            actualizarInterfazYSeleccionar(0);
        }

        private void botonAñadir_Click(object sender, EventArgs e)
        {
            // Crea un unevo alimentador con los datos de las cajas de texto
            alimentador nuevoAlimentador;
            nuevoAlimentador.tipoComponente = textBoxTipoComponente.Text;
            nuevoAlimentador.x = (int)(numericUpDownX.Value * 10);
            nuevoAlimentador.y = (int)(numericUpDownY.Value * 10);
            nuevoAlimentador.banco = (int)numericUpDownBanco.Value;
            nuevoAlimentador.anchoX = (int)(numericUpDownAnchoX.Value * 10);
            nuevoAlimentador.anchoY = (int)(numericUpDownAnchoY.Value * 10);
            nuevoAlimentador.útil = (int)numericUpDownÚtil.Value;
            nuevoAlimentador.zPick = (int)numericUpDownZPick.Value;
            nuevoAlimentador.zMover = (int)numericUpDownZMover.Value;
            nuevoAlimentador.zPlace = (int)numericUpDownZPlace.Value;
            nuevoAlimentador.velocidadX = (int)numericUpDownVelocidadX.Value;
            nuevoAlimentador.velocidadY = (int)numericUpDownVelocidadY.Value;
            nuevoAlimentador.velocidadZ = (int)numericUpDownVelocidadZ.Value;
            nuevoAlimentador.velocidadA = (int)numericUpDownVelocidadA.Value;
            nuevoAlimentador.gotasDispensador = (int)numericUpDownGotasDispensador.Value;
            nuevoAlimentador.avanceAlimentador = checkBoxAvanceAlimentador.Checked;
            nuevoAlimentador.centradoEstación = checkBoxCentradoEstación.Checked;
            nuevoAlimentador.centradoPinzasX = checkBoxCentradoPinzasX.Checked;
            nuevoAlimentador.centradoPinzasY = checkBoxCentradoPinzasY.Checked;
            datos.añadirAlimentador(nuevoAlimentador);

            actualizarInterfazYSeleccionar(datos.alimentadores.Length - 1);
            datos.modificados = true;
            datos.formPadre.actualizaInterfaz();
        }

        private void botonModificar_Click(object sender, EventArgs e)
        {
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].tipoComponente =
                 textBoxTipoComponente.Text;
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].x = (int)(numericUpDownX.Value * 10);
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].y = (int)(numericUpDownY.Value * 10);
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].banco = (int)numericUpDownBanco.Value;
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].anchoX =
                 (int)(numericUpDownAnchoX.Value * 10);
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].anchoY =
                 (int)(numericUpDownAnchoY.Value * 10);
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].útil = (int)numericUpDownÚtil.Value;
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPick = (int)numericUpDownZPick.Value;
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].zMover =
```

```
            (int)numericUpDownZMover.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPlace =
            (int)numericUpDownZPlace.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadX =
            (int)numericUpDownVelocidadX.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadY =
            (int)numericUpDownVelocidadY.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadZ =
            (int)numericUpDownVelocidadZ.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadA =
            (int)numericUpDownVelocidadA.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].gotasDispensador =
            (int)numericUpDownGotasDispensador.Value;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].avanceAlimentador =
            checkBoxAvanceAlimentador.Checked;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoEstación =
            checkBoxCentradoEstación.Checked;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoPinzasX =
            checkBoxCentradoPinzasX.Checked;
        datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoPinzasY =
            checkBoxCentradoPinzasY.Checked;

        actualizarInterfazYSeleccionar(listBoxAlimentadores.SelectedIndex);
        datos.modificados = true;
        datos.formPadre.actualizaInterfaz();
    }

    private void botonQuitar_Click(object sender, EventArgs e)
    {
        if (datos.quitarAlimentadorSiNoUsadoYReajustarComponentes(listBoxAlimentadores.SelectedIndex))
        {
            actualizarInterfazYSeleccionar(listBoxAlimentadores.SelectedIndex);
            datos.modificados = true;
        }
        else
        {
            MessageBox.Show("El alimentador está en uso, no se puede borrar.\r\n",
                    "No se puede borrar el alimentador", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        datos.formPadre.actualizaInterfaz();
    }

    public void actualizarInterfazYSeleccionar(int ítemSeleccionado)
    {
        listBoxAlimentadores.Items.Clear();
        if (datos.alimentadores.Length == 0)
        {
            listBoxAlimentadores.Enabled = false;
            botonQuitar.Enabled = false;
            botonModificar.Enabled = false;
        }
        else
        {
            listBoxAlimentadores.Enabled = true;
            botonQuitar.Enabled = true;
            botonModificar.Enabled = true;

            //Rellena el listbox de alimentadores.
            for (int a = 0; a < datos.alimentadores.Length; a++)
                listBoxAlimentadores.Items.Add((a + 1).ToString() + " -> " +
                      datos.alimentadores[a].tipoComponente);
            if (ítemSeleccionado >= datos.alimentadores.Length) ítemSeleccionado =
                      datos.alimentadores.Length - 1;
            listBoxAlimentadores.SelectedIndex = ítemSeleccionado;
        }
        if (!((datos.formPadre.fep == null) || (datos.formPadre.fep.IsDisposed)))
        {
            datos.formPadre.fep.actualizarDatos();
        }
    }

    private void listBoxAlimentadores_SelectedIndexChanged(object sender, EventArgs e)
    {
        rellenarCamposAlimentadorSeleccionado();
    }

    private void rellenarCamposAlimentadorSeleccionado()
    {
        textBoxTipoComponente.Text =
```

```
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].tipoComponente;
        numericUpDownX.Value = (decimal)datos.alimentadores[listBoxAlimentadores.SelectedIndex].x /
            10;
        numericUpDownY.Value = (decimal)datos.alimentadores[listBoxAlimentadores.SelectedIndex].y /
            10;
        numericUpDownBanco.Value = datos.alimentadores[listBoxAlimentadores.SelectedIndex].banco;
        numericUpDownAnchoX.Value =
            (decimal)datos.alimentadores[listBoxAlimentadores.SelectedIndex].anchoX / 10;
        numericUpDownAnchoY.Value =
            (decimal)datos.alimentadores[listBoxAlimentadores.SelectedIndex].anchoY / 10;
        numericUpDownÚtil.Value = datos.alimentadores[listBoxAlimentadores.SelectedIndex].útil;
        numericUpDownZPick.Value = datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPick;
        numericUpDownZMover.Value = datos.alimentadores[listBoxAlimentadores.SelectedIndex].zMover;
        numericUpDownZPlace.Value = datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPlace;
        trackBarZPick.Value = 200 - datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPick;
        trackBarZMover.Value = 200 - datos.alimentadores[listBoxAlimentadores.SelectedIndex].zMover;
        trackBarZPlace.Value = 200 - datos.alimentadores[listBoxAlimentadores.SelectedIndex].zPlace;
        numericUpDownVelocidadX.Value =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadX;
        numericUpDownVelocidadY.Value =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadY;
        numericUpDownVelocidadZ.Value =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadZ;
        numericUpDownVelocidadA.Value =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].velocidadA;
        numericUpDownGotasDispensador.Value =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].gotasDispensador;
        checkBoxAvanceAlimentador.Checked =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].avanceAlimentador;
        checkBoxCentradoEstación.Checked =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoEstación;
        checkBoxCentradoPinzasX.Checked =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoPinzasX;
        checkBoxCentradoPinzasY.Checked =
            datos.alimentadores[listBoxAlimentadores.SelectedIndex].centradoPinzasY;
    }

    private void textBoxTipoComponente_TextChanged(object sender, EventArgs e)
    {
        if (datos.cadenaCorrecta(textBoxTipoComponente.Text))
        {
            textBoxTipoComponente.BackColor = SystemColors.Window;
            botonAñadir.Enabled = true;
            if (datos.alimentadores.Length > 0) botonModificar.Enabled = true;
        }
        else
        {
            textBoxTipoComponente.BackColor = Color.Red;
            botonAñadir.Enabled = false;
            botonModificar.Enabled = false;
        }
    }

    private void numericUpDownZPick_ValueChanged(object sender, EventArgs e)
    {
        trackBarZPick.Value = (int)(200 - numericUpDownZPick.Value);
    }

    private void numericUpDownZMover_ValueChanged(object sender, EventArgs e)
    {
        trackBarZMover.Value = (int)(200 - numericUpDownZMover.Value);
    }

    private void numericUpDownZPlace_ValueChanged(object sender, EventArgs e)
    {
        trackBarZPlace.Value = (int)(200 - numericUpDownZPlace.Value);
    }

    private void trackBarZPick_Scroll(object sender, EventArgs e)
    {
        numericUpDownZPick.Value = 200 - trackBarZPick.Value;
    }

    private void trackBarZMover_Scroll(object sender, EventArgs e)
    {
        numericUpDownZMover.Value = 200 - trackBarZMover.Value;
    }

    private void trackBarZPlace_Scroll(object sender, EventArgs e)
```

```
        {
            numericUpDownZPlace.Value = 200 - trackBarZPlace.Value;
        }

        private void buttonCámara_Click(object sender, EventArgs e)
        {
            FormControlManualConCámara fc = new FormControlManualConCámara();
            fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasCámara));
            fc.IrACoordenadas(numericUpDownX.Value, numericUpDownY.Value);
            fc.Show();
        }

        public void recuperarCoordenadasCámara(int x, int y)
        {
            numericUpDownX.Value = (decimal)x / 10;
            numericUpDownY.Value = (decimal)y / 10;
        }
    }
}


// FormGeometrías.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormGeometrías : Form
    {
        public FormGeometrías()
        {
            InitializeComponent();

            Refrescar();
        }

        public void Refrescar()
        {
            dataGridViewGeometrías.Rows.Clear();
            dataGridViewGeometrías.Columns.Clear();
            dataGridViewGeometrías.ColumnCount = 4;
            dataGridViewGeometrías.Columns[0].Name = "Nombre";
            dataGridViewGeometrías.Columns[1].Name = "Tamaño X";
            dataGridViewGeometrías.Columns[2].Name = "Tamaño Y";
            dataGridViewGeometrías.Columns[3].Name = "Tamaño Z";
            for (int n = 0; n < datos.geometrías.Length; n++)
            {
                dataGridViewGeometrías.Rows.Add(datos.geometrías[n].nombre,
                                        datos.geometrías[n].dimX,
                                        datos.geometrías[n].dimY,
                                        datos.geometrías[n].dimZ);
            }
        }

        private void FormGeometrías_Resize(object sender, EventArgs e)
        {
            dataGridViewGeometrías.Width = this.Width - 40;
            dataGridViewGeometrías.Height = this.Height - 75;
            label1.Top = this.Height - 60;
        }

        private void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
        {
            object valorAnterior = 0;

            // Si se creó una fila nueva, crea una nueva geometría inicializada y muestra los valores
            if (e.RowIndex > datos.geometrías.Length - 1) datos.añadirGeometría(new
                        geometría("Nueva",0,0,0));
            if (dataGridViewGeometrías.Rows[e.RowIndex].Cells[0].Value == null)
                dataGridViewGeometrías.Rows[e.RowIndex].Cells[0].Value = "Nueva";
            if (dataGridViewGeometrías.Rows[e.RowIndex].Cells[1].Value == null)
                dataGridViewGeometrías.Rows[e.RowIndex].Cells[1].Value = 0;
            if (dataGridViewGeometrías.Rows[e.RowIndex].Cells[2].Value == null)
```

```
                dataGridViewGeometrías.Rows[e.RowIndex].Cells[2].Value = 0;
            if (dataGridViewGeometrías.Rows[e.RowIndex].Cells[3].Value == null)
                dataGridViewGeometrías.Rows[e.RowIndex].Cells[3].Value = 0;

            // Copia en la geometría el valor que se acaba de editar.
            try
            {
                switch (e.ColumnIndex)
                {
                    case 0:
                        valorAnterior = datos.geometrías[e.RowIndex].nombre;
                        datos.geometrías[e.RowIndex].nombre =
                                (string)dataGridViewGeometrías.Rows[e.RowIndex].Cells[0].Value;
                        break;
                    case 1:
                        valorAnterior = datos.geometrías[e.RowIndex].dimX;
                        datos.geometrías[e.RowIndex].dimX = (int)uint.Parse(dataGridViewGeometrías.
                                Rows[e.RowIndex].Cells[1].Value.ToString());
                        break;
                    case 2:
                        valorAnterior = datos.geometrías[e.RowIndex].dimY;
                        datos.geometrías[e.RowIndex].dimY = (int)uint.Parse(dataGridViewGeometrías.
                                Rows[e.RowIndex].Cells[2].Value.ToString());
                        break;
                    case 3:
                        valorAnterior = datos.geometrías[e.RowIndex].dimZ;
                        datos.geometrías[e.RowIndex].dimZ = (int)uint.Parse(dataGridViewGeometrías.
                                Rows[e.RowIndex].Cells[3].Value.ToString());
                        break;
                }
                datos.formPadre.actualizaAltium();
                datos.geometríasModificadas = true;
            }
            catch
            {
                // Si el dato era incorrecto, pone en la celda el valor anterior que tuviese la geometría.
                dataGridViewGeometrías.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = valorAnterior;
            }
        }

        private void dataGridView1_RowsRemoved(object sender, DataGridViewRowsRemovedEventArgs e)
        {
            // Si se eliminan filas de la cuadrícula, actualiza la matriz de geometrías.
            if (dataGridViewGeometrías.Rows.Count > 0)
            {
                // Crea de nuevo la matriz con tantos elementos como filas con datos en la cuadrícula.
                datos.geometrías = new geometría[dataGridViewGeometrías.Rows.Count - 1];
                for (int n = 0; n < datos.geometrías.Length; n++)
                {
                    datos.geometrías[n].nombre = (string)dataGridViewGeometrías.Rows[n].Cells[0].Value;
                    datos.geometrías[n].dimX =
                          (int)uint.Parse(dataGridViewGeometrías.Rows[n].Cells[1].Value.ToString());
                    datos.geometrías[n].dimY =
                          (int)uint.Parse(dataGridViewGeometrías.Rows[n].Cells[2].Value.ToString());
                    datos.geometrías[n].dimZ =
                          (int)uint.Parse(dataGridViewGeometrías.Rows[n].Cells[3].Value.ToString());
                }
            }
            datos.formPadre.actualizaAltium();
            datos.geometríasModificadas = true;
        }
    }
}


// FormEditorPlacas.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormEditorPlacas : Form
    {
```

```
Point referencia1;
Point referencia2;
Point coordenadasAnteriores;

public FormEditorPlacas()
{
    InitializeComponent();
}

// Editor de placas

private void buttonAñadirPlaca_Click(object sender, EventArgs e)
{
    // Crea una nueva placa con los datos introducidos y la añade al array de placas.
    placa p;
    p.nombre = textBoxNombrePlaca.Text;
    p.x = (int)numericUpDownXPlaca.Value * 10;
    p.y = (int)numericUpDownYPlaca.Value * 10;
    p.t = (float)numericUpDownÁnguloPlaca.Value;
    p.anchoX = (int)numericUpDownTamañoXPlaca.Value * 10;
    p.anchoY = (int)numericUpDownTamañoYPlaca.Value * 10;
    p.componentes = new componente[0];
    p.form = null;
    datos.añadirPlaca(p);

    datos.modificados = true;
    actualizarInterfazDePlacasYSeleccionar(datos.placas.Length - 1);
    datos.formPadre.actualizaInterfaz();
}

private void buttonModificarPlaca_Click(object sender, EventArgs e)
{
    if (listBoxPlacas.Items.Count > 0)
    {
        datos.placas[listBoxPlacas.SelectedIndex].nombre = textBoxNombrePlaca.Text;
        datos.placas[listBoxPlacas.SelectedIndex].x = (int)numericUpDownXPlaca.Value * 10;
        datos.placas[listBoxPlacas.SelectedIndex].y = (int)numericUpDownYPlaca.Value * 10;
        datos.placas[listBoxPlacas.SelectedIndex].t = (float)numericUpDownÁnguloPlaca.Value;
        datos.placas[listBoxPlacas.SelectedIndex].anchoX = (int)numericUpDownTamañoXPlaca.Value *
                10;
        datos.placas[listBoxPlacas.SelectedIndex].anchoY = (int)numericUpDownTamañoYPlaca.Value *
                10;

        datos.modificados = true;
        actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);
        datos.formPadre.actualizaInterfaz();
        datos.actualizarVistaDePlaca(listBoxPlacas.SelectedIndex);
    }
}

private void buttonQuitarPlaca_Click(object sender, EventArgs e)
{
    datos.quitarPlaca(listBoxPlacas.SelectedIndex);
    numericUpDownXPlaca.BackColor = SystemColors.Window;
    numericUpDownYPlaca.BackColor = SystemColors.Window;
    numericUpDownÁnguloPlaca.BackColor = SystemColors.Window;
    numericUpDownTamañoXPlaca.BackColor = SystemColors.Window;
    numericUpDownTamañoYPlaca.BackColor = SystemColors.Window;
    actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);
    datos.modificados = true;
    datos.formPadre.actualizaInterfaz();          // Actualiza los menús
}

private void buttonSubirPlaca_Click(object sender, EventArgs e)
{
    if (listBoxPlacas.SelectedIndex > 0)
    {
        int indiceP = listBoxPlacas.SelectedIndex;
        placa p = datos.placas[indiceP];
        datos.placas[indiceP] = datos.placas[indiceP - 1];
        datos.placas[indiceP - 1] = p;
        listBoxPlacas.SelectedIndex = indiceP - 1;
        actualizarInterfazDePlacasYSeleccionar(indiceP - 1);
    }
}

private void buttonBajarPlaca_Click(object sender, EventArgs e)
{
```

```
        if (listBoxPlacas.SelectedIndex < listBoxPlacas.Items.Count-1)
        {
            int indiceP = listBoxPlacas.SelectedIndex;
            placa p = datos.placas[indiceP];
            datos.placas[indiceP] = datos.placas[indiceP + 1];
            datos.placas[indiceP + 1] = p;
            listBoxPlacas.SelectedIndex = indiceP + 1;
            actualizarInterfazDePlacasYSeleccionar(indiceP + 1);
        }
    }

    private void buttonVerPlaca_Click(object sender, EventArgs e)
    {
        int p = listBoxPlacas.SelectedIndex;
        if (datos.placas.Length > p)
        {
            if ((datos.placas[p].form == null) || (datos.placas[p].form.IsDisposed))
                    datos.placas[p].form = new FormVerPlaca();
            datos.placas[p].form.MdiParent = datos.formPadre;
            datos.placas[p].form.Show();
            datos.placas[p].form.representarPlaca(p);
            datos.placas[p].form.BringToFront();
        }
    }

    private void textBoxNombrePlaca_TextChanged(object sender, EventArgs e)
    {
        if (datos.cadenaCorrecta(textBoxNombrePlaca.Text))
        {
            textBoxNombrePlaca.BackColor = SystemColors.Window;
            buttonAñadirPlaca.Enabled = true;
            if (datos.placas.Length > 0) buttonModificarPlaca.Enabled = true;
        }
        else
        {
            textBoxNombrePlaca.BackColor = Color.Red;
            buttonAñadirPlaca.Enabled = false;
            buttonModificarPlaca.Enabled = false;
        }
    }

    private void listBoxPlacas_SelectedIndexChanged(object sender, EventArgs e)
    {
        referencia1.X = 0;
        referencia1.Y = 0;
        referencia2.X = 0;
        referencia2.Y = 0;
        labelReferencia1.Text = "Ninguna";
        labelReferencia2.Text = "Ninguna";

        rellenarCamposPlacaSeleccionada();
        actualizarInterfazDeComponentesYSeleccionar(listBoxComponentes.SelectedIndex);
    }

    private void rellenarCamposPlacaSeleccionada()
    {
        if (listBoxPlacas.SelectedIndex >= 0)
        {
            textBoxNombrePlaca.Text = datos.placas[listBoxPlacas.SelectedIndex].nombre;
            numericUpDownXPlaca.Value = datos.placas[listBoxPlacas.SelectedIndex].x / 10;
            numericUpDownYPlaca.Value = datos.placas[listBoxPlacas.SelectedIndex].y / 10;
            numericUpDownÁnguloPlaca.Value = (decimal)datos.placas[listBoxPlacas.SelectedIndex].t;
            numericUpDownTamañoXPlaca.Value = datos.placas[listBoxPlacas.SelectedIndex].anchoX / 10;
            numericUpDownTamañoYPlaca.Value = datos.placas[listBoxPlacas.SelectedIndex].anchoY / 10;
            labelPlacaConErrores.Visible = !datos.placaOK(listBoxPlacas.SelectedIndex);
            if (datos.placaEnPanel(listBoxPlacas.SelectedIndex))
            {
                numericUpDownXPlaca.BackColor = SystemColors.Window;
                numericUpDownYPlaca.BackColor = SystemColors.Window;
                numericUpDownÁnguloPlaca.BackColor = SystemColors.Window;
                numericUpDownTamañoXPlaca.BackColor = SystemColors.Window;
                numericUpDownTamañoYPlaca.BackColor = SystemColors.Window;
            }
            else
            {
                numericUpDownXPlaca.BackColor = Color.Red;
                numericUpDownYPlaca.BackColor = Color.Red;
                numericUpDownÁnguloPlaca.BackColor = Color.Red;
                numericUpDownTamañoXPlaca.BackColor = Color.Red;
```

```
                numericUpDownTamañoYPlaca.BackColor = Color.Red;
            }
        }
    }

    public void actualizarInterfazDePlacasYSeleccionar(int ítemSeleccionado)
    {
        listBoxPlacas.Items.Clear();
        if (datos.placas.Length == 0)
        {
            listBoxPlacas.Enabled = false;
            buttonBorrarPlaca.Enabled = false;
            buttonModificarPlaca.Enabled = false;
            buttonSubirPlaca.Enabled = false;
            buttonBajarPlaca.Enabled = false;
            buttonVerPlaca.Enabled = false;
        }
        else
        {
            listBoxPlacas.Enabled = true;
            buttonBorrarPlaca.Enabled = true;
            buttonModificarPlaca.Enabled = true;
            buttonSubirPlaca.Enabled = true;
            buttonBajarPlaca.Enabled = true;
            buttonVerPlaca.Enabled = true;

            //Rellena el listbox de placas.
            for (int p = 0; p < datos.placas.Length; p++)
            {
                listBoxPlacas.Items.Add((p + 1).ToString() + " -> " + datos.placas[p].nombre +
                    (datos.placaOK(p)?"":" (ERROR)"));
            }
            ítemSeleccionado = datos.limitar(ítemSeleccionado, 0, datos.placas.Length - 1);
            listBoxPlacas.SelectedIndex = ítemSeleccionado;
        }
        actualizarInterfazDeComponentesYSeleccionar(listBoxComponentes.SelectedIndex);
    }

    private void buttonCámaraOrigenPlaca_Click(object sender, EventArgs e)
    {
        FormControlManualConCámara fc = new FormControlManualConCámara();
        fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasOrigenPlacaCámara));
        fc.IrACoordenadas(numericUpDownXPlaca.Value, numericUpDownYPlaca.Value);
        fc.ShowDialog();
    }

    public void recuperarCoordenadasOrigenPlacaCámara(int x, int y)
    {
        numericUpDownXPlaca.Value = (decimal)x / 10;
        numericUpDownYPlaca.Value = (decimal)y / 10;
        coordenadasAnteriores = new Point(x, y);
    }

    private void buttonCámaraReferencia1_Click(object sender, EventArgs e)
    {
        FormControlManualConCámara fc = new FormControlManualConCámara();
        fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasReferencia1Cámara));
        fc.IrACoordenadas(numericUpDownXPlaca.Value, numericUpDownYPlaca.Value);
        fc.ShowDialog();
    }

    public void recuperarCoordenadasReferencia1Cámara(int x, int y)
    {
        referencia1.X = x;
        referencia1.Y = y;
        labelReferencia1.Text = x + "," + y;
        if ((referencia2.X != 0) || (referencia2.Y != 0)) calcularÁnguloPlaca();
    }

    private void buttonCámaraReferencia2_Click(object sender, EventArgs e)
    {
        FormControlManualConCámara fc = new FormControlManualConCámara();
        fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasReferencia2Cámara));
        fc.IrACoordenadas(numericUpDownXPlaca.Value, numericUpDownYPlaca.Value);
        fc.ShowDialog();
    }

    public void recuperarCoordenadasReferencia2Cámara(int x, int y)
    {
```

```
        referencia2.X = x;
        referencia2.Y = y;
        labelReferencia2.Text = x + "," + y;
        if ((referencia1.X != 0) || (referencia1.Y != 0)) calcularÁnguloPlaca();
    }

    private void calcularÁnguloPlaca()
    {
        double deltaX = referencia2.X - referencia1.X;
        double deltaY = referencia2.Y - referencia1.Y;
        double ánguloEnGradianes = -200 * Math.Atan2(deltaY, deltaX) / Math.PI;
        if (ánguloEnGradianes < 0) ánguloEnGradianes += 400;
        numericUpDownÁnguloPlaca.Value = (decimal)ánguloEnGradianes;
    }

    public int placaActiva()
    {
        return listBoxPlacas.SelectedIndex;
    }

    // Editor de componentes

    private void buttonAñadirComponente_Click(object sender, EventArgs e)
    {
        int indice = datos.placas[listBoxPlacas.SelectedIndex].componentes.Length;

        //Crea un unevo componente con los datos introducidos
        componente c;
        c.x = (int)numericUpDownXComponente.Value * 10;
        c.y = (int)numericUpDownYComponente.Value * 10;
        c.t = (int)numericUpDownÁnguloComponente.Value;
        c.alimentador = Math.Max(0, Math.Min(datos.alimentadores.Length - 1,
                    comboBoxAlimentador.SelectedIndex));
        c.nombre = textBoxNombreComponente.Text;
        c.estado = (estadoComponente)Math.Max(0, comboBoxEstado.SelectedIndex);
        datos.placas[listBoxPlacas.SelectedIndex].añadirComponente(c);

        datos.modificados = true;
        actualizarInterfazDeComponentesYSeleccionar(datos.placas[listBoxPlacas.SelectedIndex].
             componentes.Length - 1);
        actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);
        datos.actualizarVistaDePlaca(listBoxPlacas.SelectedIndex);
    }

    private void buttonModificarComponente_Click(object sender, EventArgs e)
    {
        if (listBoxComponentes.Items.Count > 0)
        {
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.
                      SelectedIndex].nombre = textBoxNombreComponente.Text;
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.
                      SelectedIndex].alimentador = comboBoxAlimentador.SelectedIndex;
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.SelectedIndex].x
                      = (int)numericUpDownXComponente.Value * 10;
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.SelectedIndex].y
                      = (int)numericUpDownYComponente.Value * 10;
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.SelectedIndex].t
                      = (int)numericUpDownÁnguloComponente.Value;
            datos.placas[listBoxPlacas.SelectedIndex].componentes[listBoxComponentes.
                      SelectedIndex].estado = (estadoComponente)comboBoxEstado.SelectedIndex;

            datos.modificados = true;
            actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);
            datos.actualizarVistaDePlaca(listBoxPlacas.SelectedIndex);
        }
    }

    private void buttonQuitarComponente_Click(object sender, EventArgs e)
    {
        int indice = listBoxComponentes.SelectedIndex;
        datos.placas[listBoxPlacas.SelectedIndex].quitarComponente(indice);    //Quita el componente

        datos.modificados = true;
        actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);

        numericUpDownXComponente.BackColor = SystemColors.Window;
        numericUpDownYComponente.BackColor = SystemColors.Window;
        numericUpDownÁnguloComponente.BackColor = SystemColors.Window;
        datos.actualizarVistaDePlaca(listBoxPlacas.SelectedIndex);
```

```
}

private void buttonSubirComponente_Click(object sender, EventArgs e)
{
    if (listBoxComponentes.SelectedIndex > 0)
    {
        int indiceC = listBoxComponentes.SelectedIndex;
        int indiceP = listBoxPlacas.SelectedIndex;
        componente componenteASubir = datos.placas[indiceP].componentes[indiceC];
        datos.placas[indiceP].componentes[indiceC] = datos.placas[indiceP].componentes[indiceC –
                1];
        datos.placas[indiceP].componentes[indiceC - 1] = componenteASubir;
        listBoxComponentes.SelectedIndex = indiceC - 1;
        actualizarInterfazDeComponentesYSeleccionar(indiceC - 1);
    }
}

private void buttonBajarComponente_Click(object sender, EventArgs e)
{
    if (listBoxComponentes.SelectedIndex < listBoxComponentes.Items.Count - 1)
    {
        int indiceC = listBoxComponentes.SelectedIndex;
        int indiceP = listBoxPlacas.SelectedIndex;
        componente componenteASubir = datos.placas[indiceP].componentes[indiceC];
        datos.placas[indiceP].componentes[indiceC] = datos.placas[indiceP].componentes[indiceC +
                1];
        datos.placas[indiceP].componentes[indiceC + 1] = componenteASubir;
        listBoxComponentes.SelectedIndex = indiceC + 1;
        actualizarInterfazDeComponentesYSeleccionar(indiceC + 1);
    }
}

private void textBoxNombreComponente_TextChanged(object sender, EventArgs e)
{
    if (datos.cadenaCorrecta(textBoxNombreComponente.Text))
    {
        textBoxNombreComponente.BackColor = SystemColors.Window;
        buttonAñadirComponente.Enabled = true;
        if (datos.placas[listBoxPlacas.SelectedIndex].componentes.Length > 0)
                buttonModificarComponente.Enabled = true;
    }
    else
    {
        textBoxNombreComponente.BackColor = Color.Red;
        buttonAñadirComponente.Enabled = false;
        buttonModificarComponente.Enabled = false;
    }
}

private void listBoxComponentes_SelectedIndexChanged(object sender, EventArgs e)
{
    rellenarCamposComponenteSeleccionado();
}

public void rellenarCamposComponenteSeleccionado()
{
    comboBoxEstado.SelectedIndex = 0;
    comboBoxAlimentador.Items.Clear();
    for (int a = 0; a < datos.alimentadores.Length; a++)
    {
        comboBoxAlimentador.Items.Add((a + 1).ToString() + " -> " +
                datos.alimentadores[a].tipoComponente);
    }
    if (comboBoxAlimentador.Items.Count > 0) comboBoxAlimentador.SelectedIndex = 0;

    int indiceP = listBoxPlacas.SelectedIndex;
    int indiceC = listBoxComponentes.SelectedIndex;
    if ((indiceC >= 0) && (indiceP >= 0))
    {
        textBoxNombreComponente.Text = datos.placas[indiceP].componentes[indiceC].nombre;
        if (datos.alimentadores.Length > 0) comboBoxAlimentador.SelectedIndex =
                datos.placas[indiceP].componentes[indiceC].alimentador;
        numericUpDownXComponente.Value = (decimal)datos.placas[indiceP].componentes[indiceC].x /
                10;
        numericUpDownYComponente.Value = (decimal)datos.placas[indiceP].componentes[indiceC].y /
                10;
        numericUpDownÁnguloComponente.Value = datos.placas[indiceP].componentes[indiceC].t;

        labelComponenteConErrores.Visible = !datos.componenteOK(indiceP, indiceC);
```

```
            if (datos.componenteEnPlaca(indiceP, indiceC))
            {
                numericUpDownXComponente.BackColor = SystemColors.Window;
                numericUpDownYComponente.BackColor = SystemColors.Window;
                numericUpDownÁnguloComponente.BackColor = SystemColors.Window;
            }
            else
            {
                numericUpDownXComponente.BackColor = Color.Red;
                numericUpDownYComponente.BackColor = Color.Red;
                numericUpDownÁnguloComponente.BackColor = Color.Red;
            }
        }
    }

    public void actualizarInterfazDeComponentesYSeleccionar(int ítemSeleccionado)
    {
        listBoxComponentes.Items.Clear();
        buttonAñadirComponente.Enabled = datos.alimentadores.Length != 0;
        if (datos.placas.Length > 0)
        {
            groupBoxComponentes.Enabled = true;
            buttonCámaraComponente.Enabled = datos.placas[listBoxPlacas.SelectedIndex].t == 0;
            if ((datos.placas[listBoxPlacas.SelectedIndex].componentes.Length == 0) ||
                      (datos.alimentadores.Length == 0))
            {
                listBoxComponentes.Enabled = false;
                buttonBorrarComponente.Enabled = false;
                buttonModificarComponente.Enabled = false;
                buttonSubirComponente.Enabled = false;
                buttonBajarComponente.Enabled = false;
            }
            else
            {
                listBoxComponentes.Enabled = true;
                buttonBorrarComponente.Enabled = true;
                buttonModificarComponente.Enabled = true;
                buttonSubirComponente.Enabled = true;
                buttonBajarComponente.Enabled = true;

                //Rellena el listbox de componentes.
                for (int c = 0; c < datos.placas[listBoxPlacas.SelectedIndex].componentes.Length; c++)
                    listBoxComponentes.Items.Add((c + 1).ToString() + " -> " +
                        datos.placas[listBoxPlacas.SelectedIndex].componentes[c].nombre +
                        (datos.componenteOK(listBoxPlacas.SelectedIndex, c)?"":" (ERROR)"));
                ítemSeleccionado = datos.limitar(ítemSeleccionado, 0,
                     datos.placas[listBoxPlacas.SelectedIndex].componentes.Length - 1);
                listBoxComponentes.SelectedIndex = ítemSeleccionado;
            }
        }
        else
        {
            groupBoxComponentes.Enabled = false;
            numericUpDownXComponente.BackColor = SystemColors.Window; ;
            numericUpDownYComponente.BackColor = SystemColors.Window; ;
            numericUpDownÁnguloComponente.BackColor = SystemColors.Window; ;
        }
    }

    private void buttonCámaraComponente_Click(object sender, EventArgs e)
    {
        FormControlManualConCámara fc = new FormControlManualConCámara();
        fc.registrarCallback(new devolverCoordenadasCallback(recuperarCoordenadasComponenteCámara));
        fc.IrACoordenadas(numericUpDownXComponente.Value, numericUpDownYComponente.Value);
        fc.ShowDialog();
    }

    public void recuperarCoordenadasComponenteCámara(int x, int y)
    {
        // Cambia las coordenadas recibidas de la cámara para que sean relativas al origen de la placa
        numericUpDownXComponente.Value = datos.limitar((decimal)x / 10 - numericUpDownXPlaca.Value,
                                  0, datos.placas[listBoxPlacas.SelectedIndex].anchoX / 10);
        numericUpDownYComponente.Value = datos.limitar((decimal)y / 10 - numericUpDownYPlaca.Value,
                                  0, datos.placas[listBoxPlacas.SelectedIndex].anchoY / 10);
    }

    public void actualizarDatos()
    {
        actualizarInterfazDePlacasYSeleccionar(listBoxPlacas.SelectedIndex);
```

```
            rellenarCamposPlacaSeleccionada();
            actualizarInterfazDeComponentesYSeleccionar(listBoxComponentes.SelectedIndex);
            rellenarCamposComponenteSeleccionado();
        }
    }
}


// FormPlaca
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormVerPlaca : Form
    {
        #region Constantes y campos

        const int margenX = 95;
        const int margenY = 130;
        const int x0 = 30;              // Coordenadas del origen del área de dibujo de
        const int y0 = 25;              // la placa en la pantalla.
        const int distanciaCaptura = 5;

        int xp0 = 0;
        int yp0 = 0;
        int anchoVisibleX;
        int anchoVisibleY;
        int componente = -1;
        int numplaca;
        bool mostrandoToolTip = false;
        bool moviendoComponente = false;
        Point posiciónAnteriorRatón;
        SizeF tamañoTexto = new SizeF();
        Color colorFondo = Color.Black;
        Font fuenteComponentes = new Font("Arial", 5);
        Font fuenteEscalas = new Font("Arial", 7);
        Pen plumaLíneasEscalas = new Pen(Color.White);     // Pluma para dibujar las líneas.
        Brush pincelRótulosEscalas = Brushes.Wheat;        // Pincel para escribir los rótulos.
        placa p;

        #endregion

        public FormVerPlaca()
        {
            InitializeComponent();
            this.MouseWheel += new MouseEventHandler(FormPlaca_MouseWheel);
        }

        public bool representarPlaca(int n)
        {
            if (n >= datos.placas.Length) return false;

            p = datos.placas[n];
            numplaca = n;

            if ((this.MdiParent != null) && (datos.formPadre.ClientSize.Width < (p.anchoX + margenX)))
            {
                this.Left = 0;
                this.Width = datos.formPadre.ClientSize.Width - 10;
            }
            else
            {
                this.Width = p.anchoX + margenX;
            }

            if ((this.MdiParent != null) && (datos.formPadre.ClientSize.Height < (p.anchoY + margenY)))
            {
                this.Top = 0;
                this.Height = datos.formPadre.ClientSize.Height - 32;
            }
            else
            {
```

```
            this.Height = p.anchoY + margenY;
        }

        dibujarPlaca();
        return true;
    }

    private void dibujarPlaca()
    {
        if (this.IsDisposed) return;
        Color colorComponente;
        System.Drawing.Brush pincelTexto;

        // Pone el texto de la barra de título.
        this.Text = "Placa " + (numplaca + 1).ToString() + " - " + datos.placas[numplaca].nombre + " -
             " + Convert.ToString(p.anchoX / 10) + "mm x " + Convert.ToString(p.anchoY / 10) + "mm.";
        if (!datos.placaOK(numplaca))
        {
            this.Text += "  ATENCIÓN: Placa con errores";
        }

        // Ajusta las barras de desplazamiento.
        if ((xp0 + anchoVisibleX) > p.anchoX) xp0 = p.anchoX - anchoVisibleX;
        if ((yp0 + anchoVisibleY) > p.anchoY) yp0 = p.anchoY - anchoVisibleY;
        ajusteScrollBars();

        // Crea un lienzo buffer para dibujar.
        Bitmap imagen = new Bitmap(this.Width, this.Height);
        Graphics lienzo = Graphics.FromImage(imagen);
        lienzo.Clear(colorFondo);

        // Dibuja los componentes con su nombre.
        for (int i = 0; i < p.componentes.Length; i++)
        {
            switch (p.componentes[i].estado)
            {
                case estadoComponente.Cogiendo:
                    colorComponente = Color.Blue;
                    pincelTexto = Brushes.LightBlue;
                    break;
                case estadoComponente.Colocando:
                    colorComponente = Color.DarkViolet;
                    pincelTexto = Brushes.LightBlue;
                    break;
                case estadoComponente.Colocado:
                    colorComponente = Color.Green;
                    pincelTexto = Brushes.LightGreen;
                    break;
                case estadoComponente.Error_Cogiendo:
                    colorComponente = Color.OrangeRed;
                    pincelTexto = Brushes.Orange;
                    break;
                case estadoComponente.Error_Colocando:
                    colorComponente = Color.Red;
                    pincelTexto = Brushes.Orange;
                    break;
                default:    // Componente no tratado
                    colorComponente = Color.Khaki;
                    pincelTexto = Brushes.Wheat;
                    break;
            }
            lienzo.DrawPolygon(new Pen(colorComponente), p.componentes[i].vértices(x0 - xp0, y0 –
                      yp0));
            tamañoTexto = lienzo.MeasureString(p.componentes[i].nombre, fuenteComponentes);
            lienzo.DrawString(p.componentes[i].nombre, fuenteComponentes, pincelTexto,
                              new Point((int)(p.componentes[i].x + x0 - xp0 - tamañoTexto.Width /
                              3), (int)(p.componentes[i].y + y0 - yp0 - tamañoTexto.Height / 3)));
        }

        // Borra los bordes para que no se vean lo que sobresale.
        lienzo.FillRectangle(Brushes.Black, 0, 0, this.Width, y0);
        lienzo.FillRectangle(Brushes.Black, 0, y0, x0, anchoVisibleY);
        lienzo.FillRectangle(Brushes.Black, 0, y0 + anchoVisibleY, this.Width, this.Height - y0 -
                      anchoVisibleY);
        lienzo.FillRectangle(Brushes.Black, x0 + anchoVisibleX, y0, this.Width - x0 - anchoVisibleX,
                      anchoVisibleY);

        // Dibuja las 4 escalas y el marco.
        dibujarEscala(lienzo, x0, y0, anchoVisibleX, xp0, 'u');
```

```
        dibujarEscala(lienzo, x0, y0, anchoVisibleY, yp0, 'l');
        dibujarEscala(lienzo, x0, y0 + anchoVisibleY, anchoVisibleX, xp0, 'd');
        dibujarEscala(lienzo, x0 + anchoVisibleX, y0, anchoVisibleY, yp0, 'r');
        lienzo.DrawRectangle(Pens.White, x0, y0, anchoVisibleX, anchoVisibleY);
        tamañoTexto = lienzo.MeasureString("mm", fuenteEscalas);
        lienzo.DrawString("mm", fuenteEscalas, pincelRótulosEscalas, x0 - tamañoTexto.Width - 5, y0 –
                    tamañoTexto.Height - 5);

        // Copia el dibujo hecho en el buffer en la ventana para que se vea.
        Graphics g = this.CreateGraphics();
        g.DrawImage(imagen, 0, 0);
        lienzo.Dispose();
        imagen.Dispose();
        g.Dispose();
    }

    private void dibujarEscala(Graphics lienzo, int xr0, int yr0, int longitud, int valorInicial,
            char posición)
    {
        // Dibuja las líneas de graduación de las escalas y las rotula.
        // La posición de las líneas y el texto será:
        //  u = encima
        //  d = debajo
        //  l = izquierda
        //  r = derecha

        const int divisionMenor = 10;       // Separación entre divisiones menores de la escala.
        const int divisionMayor = 50;       // Separación entre divisiones numeradas de la escala.

        int x, y;
        float xRótulo, yRótulo;
        string rótulo;

        // Desplazamiento de la primera división. Valor en el rango [0..divisionMenor-1]
        int desplazamientoInicial = divisionMenor - valorInicial % divisionMenor;
        // Incrementador horizontal. Vale 1 para escalas horizontales.
        int incX = (posición == 'u' || posición == 'd') ? 1 : 0;
        // Incrementador vertical. Vale 1 para escalas verticales.
        int incY = (posición == 'l' || posición == 'r') ? 1 : 0;
        // Posición de los rótulos encima (-1) debajo (1) o a nivel (0) de la escala.
        int arrAba = posición == 'u' ? -1 : posición == 'd' ? 1 : 0;
        // Posición de los rótulos a izquierda (-1), derecha (1) o a nivel (0) de la escala.
        int izqDer = posición == 'l' ? -1 : posición == 'r' ? 1 : 0;

        for (int n = desplazamientoInicial ; n <= longitud ; n += divisionMenor)
        {
            // Calcula las coordenadas de la siguiente línea de la escala
            x = xr0 + n * incX;
            y = yr0 + n * incY;

            if (((valorInicial+n) % divisionMayor) != 0)
            {
                // Dibuja la línea de las divisiones menores.
                lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 2, y + arrAba * 2);
            }
            else
            {
                // Dibuja la línea de las divisiones mayores.
                lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 5, y + arrAba * 5);
                // Crea el rótulo de las divisiones mayores.
                rótulo = Convert.ToString((xp0 * incX + yp0 * incY + n) / 10);
                // Mide el tamaño que tendrá el rótulo.
                tamañoTexto = lienzo.MeasureString(rótulo, fuenteEscalas);
                // Calcula las coordenadas de escritura del rótulo.
                xRótulo = x - incX * tamañoTexto.Width / 2 + incY * izqDer * 6 - (posición == 'l' ?
                      tamañoTexto.Width : 0);
                yRótulo = y - incY * tamañoTexto.Height / 3 + incX * arrAba * 6 - (posición == 'u' ?
                      tamañoTexto.Height : 0);
                // Escribe el rótulo.
                lienzo.DrawString(rótulo, fuenteEscalas, pincelRótulosEscalas, xRótulo, yRótulo);
            }
        }
    }

    private void ajusteScrollBars()
    {
        if (this.Width < (p.anchoX + margenX))
        {
            hScrollBar1.Visible = true;
```

```
            hScrollBar1.Maximum = p.anchoX;
            hScrollBar1.LargeChange = Math.Max(0,anchoVisibleX); // Evita error al minimizar
            hScrollBar1.Value = xp0;
        }
        else
        {
            hScrollBar1.Visible = false;
            hScrollBar1.Value = 0;
        }

        if (this.Height < (p.anchoY + margenY))
        {
            vScrollBar1.Visible = true;
            vScrollBar1.Maximum = p.anchoY;
            vScrollBar1.LargeChange = Math.Max(0, anchoVisibleY); // Evita error al minimizar
            vScrollBar1.Value = yp0;
        }
        else
        {
            vScrollBar1.Visible = false;
            vScrollBar1.Value = 0;
        }
    }

    private void FormPlaca_Paint(object sender, PaintEventArgs e)
    {
        dibujarPlaca();
    }

    protected override void OnPaintBackground(PaintEventArgs e)
    {
        return;
    }

    private void FormPlaca_Resize(object sender, EventArgs e)
    {
        if (this.Width < (p.anchoX + margenX))
        {
            anchoVisibleX = this.Width - margenX;
        }
        else
        {
            anchoVisibleX = p.anchoX;
            this.Width = p.anchoX + margenX;
        }

        if (this.Height < (p.anchoY + margenY))
        {
            anchoVisibleY = this.Height - margenY;
        }
        else
        {
            anchoVisibleY = p.anchoY;
            this.Height = p.anchoY + margenY;
        }

        dibujarPlaca();
    }

    private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
    {
        if (e.NewValue != e.OldValue)
        {
            xp0 = e.NewValue;
            dibujarPlaca();
        }
    }

    private void vScrollBar1_Scroll(object sender, ScrollEventArgs e)
    {
        if (e.NewValue != e.OldValue)
        {
            yp0 = e.NewValue;
            dibujarPlaca();
        }
    }

    private void FormPlaca_MouseDown(object sender, MouseEventArgs e)
    {
```

```
    if ((e.Button == System.Windows.Forms.MouseButtons.Left) && (componente != -1))
    {
        moviendoComponente = true;
    }
}

private void FormPlaca_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
{
    if ((e.Button == System.Windows.Forms.MouseButtons.Left) && moviendoComponente)
    {
        // Actualiza la posición del componente que se estuviera moviendo
        int desplazamientoX = e.X - posiciónAnteriorRatón.X;
        int desplazamientoY = e.Y - posiciónAnteriorRatón.Y;
        p.componentes[componente].x += desplazamientoX;
        p.componentes[componente].y += desplazamientoY;

        // Si se está moviendo un componente se quita el ToolTip y se muestran sus coordenadas
        toolTipComponente.Hide(this);
        mostrandoToolTip = false;
        dibujarPlaca();
        toolStripStatusLabel1.Text = "(" +
            Convert.ToString(Math.Round((float)(p.componentes[componente].x / 10)))
            + "," + Convert.ToString((float)(p.componentes[componente].x % 10))
            + " : " + Convert.ToString(Math.Round((float)p.componentes[componente].y / 10))
            + "," + Convert.ToString(p.componentes[componente].y % 10) + ") " +
            Convert.ToString(p.componentes[componente].t) + "º";
        datos.modificados = true;

        // Si se ha movido, actualiza la ventana del editor de componentes.
        if (((desplazamientoX != 0) || (desplazamientoY != 0)) && !((datos.formPadre.fep == null)
                || (datos.formPadre.fep.IsDisposed)))
        {
            datos.formPadre.fep.actualizarInterfazDePlacasYSeleccionar(numplaca);
            datos.formPadre.fep.actualizarInterfazDeComponentesYSeleccionar(componente);
        }
    }
    else
    {
        if ((e.X >= x0) && (e.X < x0 + p.anchoX) && (e.Y >= y0) && (e.Y <= y0 + p.anchoY))
        {
            toolStripStatusLabel1.Text = "(" + Convert.ToString(Math.Round((float)(e.X - x0) /
                  10)) + "," + Convert.ToString((float)(e.X - x0) % 10)
                  + " : " + Convert.ToString(Math.Round((float)(e.Y - y0) / 10)) + "," +
                  Convert.ToString((e.Y - y0) % 10) + ")";
            componente = -1;
            for (int i = 0; i < p.componentes.Length; i++)
            {
                if (((Math.Abs(e.X - (p.componentes[i].x + x0 - xp0))) < distanciaCaptura) &&
                          (Math.Abs(e.Y - (p.componentes[i].y + y0 - yp0))) < distanciaCaptura)
                {
                    componente = i;
                }
            }

            // Actualiza el ToolTip
            if (componente == -1)
            {
                this.Cursor = System.Windows.Forms.Cursors.Default;
                toolTipComponente.Hide(this);
                mostrandoToolTip = false;
            }
            else
            {
                if (!mostrandoToolTip)
                {
                    this.Cursor = System.Windows.Forms.Cursors.Hand;
                    string texto = datos.alimentadores[p.componentes[componente].alimentador].
                        tipoComponente + "\r"
                        + "x: " + Math.Round((float)(p.componentes[componente].x / 10)).ToString()
                        + "," + ((float)(p.componentes[componente].x % 10)).ToString()
                        + "   y: " + Math.Round((float)p.componentes[componente].y /
                        10).ToString()
                        + "," + (p.componentes[componente].y % 10).ToString()
                        + "   Giro: " + p.componentes[componente].t.ToString()
                        + "º\rAlimentador: " + p.componentes[componente].alimentador.ToString()
                        + "   Útil: " +
                        datos.alimentadores[p.componentes[componente].alimentador].útil.ToString()
                        + "\rEstado: ";
                    switch (p.componentes[componente].estado)
```

```
                    {
                        case estadoComponente.Cogiendo:
                            texto += "Cogiendo";
                            break;
                        case estadoComponente.Colocando:
                            texto += "Colocando";
                            break;
                        case estadoComponente.Colocado:
                            texto += "Colocado";
                            break;
                        case estadoComponente.Error_Cogiendo:
                            texto += "Error al coger";
                            break;
                        case estadoComponente.Error_Colocando:
                            texto += "Error al colocar";
                            break;
                        default:
                            texto += "No tratado";
                            break;
                    }
                    mostrandoToolTip = true;
                    toolTipComponente.ToolTipTitle = "Componente: " +
                          p.componentes[componente].nombre;

                    // Apunta el tooltip hacia un label oculto en la posición del componente
                    labelObjetivo.Left = p.componentes[componente].x + x0 - xp0;
                    labelObjetivo.Top = p.componentes[componente].y + y0 - yp0;

                    toolTipComponente.Show(string.Empty, labelObjetivo, 0);
                    toolTipComponente.Show(texto, labelObjetivo);
                }
            }
        }
        if (e.Button == System.Windows.Forms.MouseButtons.Left) // Hace scroll                {
            if (anchoVisibleX < p.anchoX)
            {
                xp0 -= e.X - posiciónAnteriorRatón.X;
                if (xp0 < 0) xp0 = 0;
            }
            if (anchoVisibleY < p.anchoY)
            {
                yp0 -= e.Y - posiciónAnteriorRatón.Y;
                if (yp0 < 0) yp0 = 0;
            }
            this.Cursor = System.Windows.Forms.Cursors.SizeAll;
            dibujarPlaca();
        }
    }
    posiciónAnteriorRatón = e.Location;
}

private void FormPlaca_MouseUp(object sender, MouseEventArgs e)
{
    moviendoComponente = false;
}

private void FormPlaca_MouseWheel(object sender, MouseEventArgs e)
{
    if (componente != -1)
    {
        p.componentes[componente].t = p.componentes[componente].t + Math.Sign(e.Delta);
        if (p.componentes[componente].t < 0) p.componentes[componente].t = 400 +
                p.componentes[componente].t;
        if (p.componentes[componente].t >= 400) p.componentes[componente].t =
                p.componentes[componente].t - 400;
        toolTipComponente.Hide(this);
        mostrandoToolTip = false;
        dibujarPlaca();
        datos.modificados = true;

        // Actualiza la ventana del editor de componentes.
        if (!((datos.formPadre.fep == null) || (datos.formPadre.fep.IsDisposed)))
        {
            datos.formPadre.fep.actualizarInterfazDePlacasYSeleccionar(numplaca);
            datos.formPadre.fep.actualizarInterfazDeComponentesYSeleccionar(componente);
        }
    }
    else
    {
```

```
                if (vScrollBar1.Visible)
                {
                    yp0 -= 10 * Math.Sign(e.Delta);
                    if (yp0 < 0) yp0 = 0;
                    vScrollBar1.Value = yp0;
                    dibujarPlaca();
                }
                else
                {
                    if (hScrollBar1.Visible)
                    {
                        xp0 -= 10 * Math.Sign(e.Delta);
                        if (xp0 < 0) xp0 = 0;
                        hScrollBar1.Value = xp0;
                        dibujarPlaca();
                    }
                }
            }
        }
    }
}


// FormProduccion.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.IO.Ports;

namespace PnPWork
{
    public partial class FormProduccion : Form
    {
        #region Campos

        placa placaActual;
        int útilActual = 0;                 // Siempre se supondrá que la máquina arranca sin útil.
        int bancoActual = 0;                // 0 Obliga a inicializar el banco en la producción
        int velocidadXActual = 15;
        int velocidadYActual = 15;
        int velocidadZActual = 12;
        int velocidadAActual = 12;
        int totalComponentes = 0;           // Total de componentes de las placas en producción.
        int totalPendientes = 0;            // Total de componentes pendientes de procesar.
        int componentesTratados;
        string registro;
        int componentesFallados;
        Size ajusteTextBox;                 // Para redimensionar la caja de texto.
        BackgroundWorker worker;            // Referencia al hilo de producción.
        static ArrayList índicesDePlacasMostradas;

        #endregion

        // Métodos ejecutados en el hilo principal de la aplicación

        public FormProduccion()
        {
            InitializeComponent();

            ajusteTextBox = this.Size - textBoxRegistro.Size;
            this.Height = this.MinimumSize.Height;
            actualizaListaPlacas();
        }

        private void checkedListBoxPlacas_ItemCheck(object sender, ItemCheckEventArgs e)
        {
            // Cuenta el número de componentes antes del cambio
            cuentaComponentes();

            // Añade o resta los componentes de la placa añadida o quitada
            int índicePlaca = (int)índicesDePlacasMostradas[e.Index];
            switch (e.NewValue)
```

```
        {
            case CheckState.Indeterminate:
                break;
            case CheckState.Checked:
                totalComponentes += datos.placas[índicePlaca].componentes.Length;
                if (checkBoxReiniciar.Checked) totalPendientes +=
                      datos.placas[índicePlaca].componentes.Length;
                else totalPendientes += datos.placas[índicePlaca].componentesPendientes();
                break;
            case CheckState.Unchecked:
                totalComponentes -= datos.placas[índicePlaca].componentes.Length;
                if (checkBoxReiniciar.Checked) totalPendientes -=
                      datos.placas[índicePlaca].componentes.Length;
                else totalPendientes -= datos.placas[índicePlaca].componentesPendientes();
                break;
        }

        // Actualiza la interfaz gráfica:
        labelPendientes.Text = totalPendientes.ToString();
        labelTotal.Text = totalComponentes.ToString();
    }

    private void checkBoxReiniciar_CheckedChanged(object sender, EventArgs e)
    {
        cuentaComponentes();

        // Actualiza la interfaz gráfica:
        labelPendientes.Text = totalPendientes.ToString();
        labelTotal.Text = totalComponentes.ToString();
    }

    private void checkBoxVerDetalles_CheckedChanged(object sender, EventArgs e)
    {
        if (checkBoxVerDetalles.Checked)
        {
            if (this.Height == this.MinimumSize.Height) this.Height = 450;
        }
        else
        {
            this.WindowState = FormWindowState.Normal;
            this.Height = this.MinimumSize.Height;
        }
    }

    private void FormProduccion_Resize(object sender, EventArgs e)
    {
        textBoxRegistro.Height = this.Height - ajusteTextBox.Height;
        textBoxRegistro.Width = this.Width - ajusteTextBox.Width;
        progressBarProducción.Width = this.Width - ajusteTextBox.Width;

        if (this.Height > this.MinimumSize.Height) checkBoxVerDetalles.Checked = true;
        else checkBoxVerDetalles.Checked = false;
    }

    private void FormProduccion_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (backgroundWorkerProducir.IsBusy) e.Cancel = true;
    }

    private void cuentaComponentes()
    {
        totalComponentes = 0;
        totalPendientes = 0;

        for (int n = 0; n <= checkedListBoxPlacas.CheckedIndices.Count - 1; n++)
        {
            int índicePlaca = (int)índicesDePlacasMostradas[checkedListBoxPlacas.CheckedIndices[n]];
            totalComponentes += datos.placas[índicePlaca].componentes.Length;
            if (checkBoxReiniciar.Checked) totalPendientes +=
                      datos.placas[índicePlaca].componentes.Length;
            else totalPendientes += datos.placas[índicePlaca].componentesPendientes();
        }
    }

    private void actualizaListaPlacas()
    {
        // Rellena la lista de placas con los componentes que tienen pendientes
        checkedListBoxPlacas.Items.Clear();
        índicesDePlacasMostradas = new ArrayList();
```

```
        labelPlacasErrores.Visible = false;
        for (int p = 0; p < datos.placas.Length; p++)
        {
            if (datos.placaOK(p))
            {
                índicesDePlacasMostradas.Add(p);
                checkedListBoxPlacas.Items.Add((p + 1) + "-> " + datos.placas[p].nombre + " - Pend: "
                      + datos.placas[p].componentesPendientes() + " de " +
                      datos.placas[p].componentes.Length);
                if (datos.placas[p].componentesPendientes() > 0)
                      checkedListBoxPlacas.SetItemChecked(checkedListBoxPlacas.Items.Count - 1, true);
            }
            else
            {
                labelPlacasErrores.Visible = true;
            }
        }
    }

    private void GrabarRegistroYFinalizarlo()
    {
        try
        {
            string directorioDeTrabajo = Path.GetDirectoryName(datos.rutaFichero);
            string nombreFichero = Path.GetFileNameWithoutExtension(datos.rutaFichero);
            string directorioLog = Path.Combine(directorioDeTrabajo, datos.directorioDeLogs);
            string rutaficheroLog;
            string nombreFicheroLog;

            if (!Directory.Exists(directorioLog)) Directory.CreateDirectory(directorioLog);

            //Busca un nombre libre
            int n = 1;
            do
            {
                nombreFicheroLog = nombreFichero + " Producción_" + n + ".log";
                rutaficheroLog = Path.Combine(directorioLog, nombreFicheroLog);
                n++;
            }
            while (File.Exists(rutaficheroLog));

            // Graba el archivo de registro
            StreamWriter fichero = new StreamWriter(rutaficheroLog, false);
            fichero.Write(registro);
            fichero.Close();
            registro += "\r\nRegistro guardado.\r\n";
        }
        catch
        {
            registro += "\r\nNo se pudo guardar el registro.\r\n";
        }
        finally
        {
            textBoxRegistro.Text = registro;
            textBoxRegistro.SelectionStart = textBoxRegistro.TextLength;
            textBoxRegistro.ScrollToCaret();
        }
    }

    private void buttonIniciar_Click(object sender, EventArgs e)
    {
        // Comprueba si hay una producción en curso para saber si se va a cancelar o iniciar
        if (backgroundWorkerProducir.IsBusy)
        {
            buttonIniciar.Enabled = false;
            buttonIniciar.Text = "Espere";
            buttonIniciar.BackColor = Color.Gray;

            // Solicita cancelar la producción
            backgroundWorkerProducir.CancelAsync();
        }
        else
        {
            if (totalPendientes < 1)
            {
                MessageBox.Show("No hay componentes pendientes de montar.", "No hay componentes
                      pendientes", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
            else
```

```
        {
            // Comprueba si el puerto serie está disponible para empezar
            if (!datos.puertoSerieDisponible)
            {
                MessageBox.Show("No se ha seleccionado un puerto serie.\nLa comunicación con la
                          máquina Pick & Place no será posible mientras no se configure un puerto
                          serie válido (menú Configuración).", "Puerto serie no disponible",
                          MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            else
            {
                // Bloquea los controles para evitar cambios durante la producción
                checkBoxReiniciar.Enabled = false;
                checkBoxDispensarAdhesivo.Enabled = false;
                checkedListBoxPlacas.Enabled = false;

                buttonIniciar.Text = "Detener";
                buttonIniciar.BackColor = Color.Red;
                textBoxRegistro.Text = "";

                // Inicia el hilo de producción
                backgroundWorkerProducir.RunWorkerAsync();
            }
        }
    }
}

private void backgroundWorkerProducir_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    progressBarProducción.Value = Math.Min(e.ProgressPercentage, progressBarProducción.Maximum);
    this.Text = "En producción - " + e.ProgressPercentage.ToString() + "% completado";
    labelPendientes.Text = (totalPendientes - componentesTratados).ToString();
    textBoxRegistro.Text = registro;
    textBoxRegistro.SelectionStart = textBoxRegistro.TextLength;
    textBoxRegistro.ScrollToCaret();
    if (e.UserState != null) datos.mostrarPlaca((int)e.UserState);
}

private void backgroundWorkerProducir_RunWorkerCompleted(object sender,
         RunWorkerCompletedEventArgs e)
{
    buttonIniciar.Text = "Iniciar";
    buttonIniciar.BackColor = Color.DodgerBlue;
    buttonIniciar.Enabled = true;
    this.Text = "Producción";

    // Desbloquea los controles
    checkBoxReiniciar.Enabled = true;
    checkBoxDispensarAdhesivo.Enabled = true;
    checkedListBoxPlacas.Enabled = true;
    actualizaListaPlacas();

    if (e.Error != null)
    {
        // Se produjo una excepción no gestionada o relanzada en el DoWork
        registro += "---------- ERROR IRRECUPERABLE. PRODUCCIÓN INTERRUMPIDA ----------\r\n";
        GrabarRegistroYFinalizarlo();
        MessageBox.Show("Error: " + e.Error.Message +
                  "\r\n\r\nEs necesario revisar la máquina, puede haber sufrido una parada de
                  seguridad." + "\r\nEn ese caso, reiníciela.", "Error", MessageBoxButtons.OK,
                  MessageBoxIcon.Error);
    }
    else if (e.Cancelled)
    {
        // Se canceló la producción
        registro += "------- Producción cancelada antes de ser completada -------\r\n";
        GrabarRegistroYFinalizarlo();
        MessageBox.Show("Se cancelo la producción antes de ser completada", "Producción
                  cancelada", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    else
    {
        // La producción terminó normalmente
        if (componentesFallados == 0)
        {
            registro += "------- Producción completada sin errores -------\r\n";
            GrabarRegistroYFinalizarlo();
            MessageBox.Show("Se ha completado la producción. Se han montado todos los
```

```
                componentes.", "Producción completada exitosamente", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
        else
        {
            registro += "------- Producción completada. " + componentesFallados + " componentes no
                se han podido montar -------\r\n";
            GrabarRegistroYFinalizarlo();
            MessageBox.Show("Producción terminada. " + componentesFallados + " componentes no se
                han podido montar.", "Producción terminada", MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
        }
    }
}

// Métodos ejecutados en el segundo hilo

private void backgroundWorkerProducir_DoWork(object sender, DoWorkEventArgs e)
{
    worker = sender as BackgroundWorker;    // Lo guarda para acceder desde otros métodos
    componentesTratados = 0;
    componentesFallados = 0;
    velocidadXActual = 0;
    velocidadYActual = 0;
    velocidadZActual = 0;
    velocidadAActual = 0;
    registro = DateTime.Now.ToString() + "  -###- Iniciada la producción -###-\r\n\r\n";


    datos.puertoSerie.DiscardOutBuffer();   // Se limpian los buffers de posibles
    datos.puertoSerie.DiscardInBuffer();    // operaciones anteriores.

    try
    {
        // Secuencia de inicialización de la Cosy
        transmitir("OO");
        transmitirYConfirmar("HZHAVATAT10030T20010T30020T40030T50150");
        worker.ReportProgress(0);

        // Produce cada placa en secuencia
        for (int n = 0; n <= checkedListBoxPlacas.CheckedIndices.Count - 1; n++)
        {
            int índicePlaca =
                (int)índicesDePlacasMostradas[checkedListBoxPlacas.CheckedIndices[n]];
            producirPlaca(índicePlaca);

            // Comprueba si se ha pulsado el botón de detener
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                registro += "\r\n" + DateTime.Now.ToString() + " -###- Producción cancelada -###-
                    \r\n\r\n";
                break;
            }
        }

        // Al terminar desmonta el útil si había alguno y pone en la posición inicial el cabezal y
        // los alimentadores
        registro += "\r\nAparcando...\r\n";
        transmitirYConfirmar("LE");
        establecerVelocidadMáxima();
        desmontarÚtilActual();
        transmitirYConfirmar("PZ0000PA0000PX0000PY0000VA");
        transmitirYConfirmar("LA");
    }
    // Captura las excepciones para registrar los errores y las relanza para abortar y mostrar
    // mensaje de error en RunWorkerCompleted
    catch (PeligroException)
    {
        // La máquina se bloquea.
        registro += "\t\t" + DateTime.Now.ToString() + "#### PELIGRO DE INTEGRIDAD ####\r\n";
        throw;
    }
    catch (TimeoutException)
    {
        registro += "\t\t" + DateTime.Now.ToString() + "#### ERROR TIMEOUT PUERTO SERIE ####\r\n";
        throw;
    }
```

```
        if (!e.Cancel) registro += "\r\n" + DateTime.Now.ToString() + " -###- Producción finalizada –
             ###-\r\n\r\n";
        worker.ReportProgress(pocentajeTerminado());
    }

    private void producirPlaca(int p)
    {
        registro += "\r\n## Placa " + p + " en producción ##\r\n";
        placaActual = datos.placas[p];

        // Reinicia el estado de los componentes de la placa si se marcó el checkBox.
        if (checkBoxReiniciar.Checked) datos.reiniciarComponentesPlaca(p);

        // Procesa todos los componentes de la placa en secuencia.
        int reintentosComponenteActual = 0;
        int c = 0;   // Índice del componente
        while (c < placaActual.componentes.Length)
        {
            try
            {
                // Comprueba si este componente ya ha sido colocado.
                if (placaActual.componentes[c].estado == estadoComponente.Colocado)
                {
                    registro += "\r\n---  Componente " + c + ": " + placaActual.componentes[c].nombre
                              + "---\r\n--- Ya estaba colocado. ---\r\n";
                }
                else
                {
                    colocarComponente(c, p, reintentosComponenteActual);
                    componentesTratados++;
                    reintentosComponenteActual = 0;
                }
                worker.ReportProgress(pocentajeTerminado());
                c++;
            }
            catch (PérdidaException)
            {
                // Captura la excepción de pérdida del componente para recuperarse y continuar
                transmitirYConfirmar("LE"); // Activa la lámpara de alarma.
                registro += "#### ERROR DE PÉRDIDA ####\r\n";
                desecharComponente(); // Desecha el componente que dio error de pérdida.
                if (reintentosComponenteActual > datos.reintentosMax)
                {
                    // Si excede el nº de reintentos, lo añade a la cuenta, lo marca y prosigue.
                    if (placaActual.componentes[c].estado == estadoComponente.Cogiendo)
                              placaActual.componentes[c].estado = estadoComponente.Error_Cogiendo;
                    if (placaActual.componentes[c].estado == estadoComponente.Colocando)
                              placaActual.componentes[c].estado = estadoComponente.Error_Colocando;
                    componentesFallados++;
                    componentesTratados++;
                    reintentosComponenteActual = 0;
                    c++;
                }
                else
                {
                    reintentosComponenteActual++;
                }
                transmitirYConfirmar("LA"); // Desactiva la lámpara de alarma.
            }
            if (worker.CancellationPending) return;
        }
    }

    private void colocarComponente(int c, int p, int reintento)
    {
        string bufferTX = "";
        componente componenteActual = placaActual.componentes[c];
        alimentador alimentadorActual = datos.alimentadores[componenteActual.alimentador];
        registro += "\r\n----  Componente " + c + ": " + componenteActual.nombre + "  ----\r\n";
        if (reintento > 0)
        {
            registro += "----  Reintento " + reintento.ToString() + "  ----\r\n";
        }

        //Maximiza velocidades y dispensa las gotas de pegamento que requiera este componente
        establecerVelocidadMáxima();
        if (checkBoxDispensarAdhesivo.Checked && (reintento == 0) &&
             (alimentadorActual.gotasDispensador > 0)) dispensarAdhesivo(componenteActual);
```

```
// Pone el componente en estado "Cogiendo" y lo muestra en pantalla:
placaActual.componentes[c].estado = estadoComponente.Cogiendo;
if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);

// Selecciona banco de alimentadores y coloca  el útil adecuado en el cabezal:
seleccionarBanco(alimentadorActual.banco);
montarÚtil(alimentadorActual.útil);

// Coloca el cabezal sobre el alimentador:
registro += "Posicionando Pick:\r\n";
bufferTX = "PX" + cuatroDígitos(datos.xMax - alimentadorActual.x)
           + "PY" + cuatroDígitos(alimentadorActual.y);
transmitirYConfirmar(bufferTX);

// Establece la velocidad de elevación y coge el componente:
registro += "Pick:\r\n";
bufferTX = "";
if (velocidadZActual > alimentadorActual.velocidadZ)
{
    velocidadZActual = alimentadorActual.velocidadZ;
    bufferTX = "SZ" + dosDígitos(velocidadZActual);
}

bufferTX += "PZ" + cuatroDígitos(alimentadorActual.zPick)
    + "VEPZ" + cuatroDígitos(alimentadorActual.zMover);

// Avance del alimentador:
if (alimentadorActual.avanceAlimentador) bufferTX += "FA001";

// Test de vacío:
bufferTX += "VT";
transmitirYConfirmar(bufferTX);

// Pone el componente en estado "Colocando" y lo muestra en pantalla:
placaActual.componentes[c].estado = estadoComponente.Colocando;
if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);

// Establece la velocidad de transporte y giro del componente:
bufferTX = "";
if (velocidadXActual > alimentadorActual.velocidadX)
{
    velocidadXActual = alimentadorActual.velocidadX;
    bufferTX = "SX" + dosDígitos(velocidadXActual);
}
if (velocidadYActual > alimentadorActual.velocidadY)
{
    velocidadYActual = alimentadorActual.velocidadY;
    bufferTX += "SY" + dosDígitos(velocidadYActual);
}
if (velocidadAActual > alimentadorActual.velocidadA)
{
    velocidadAActual = alimentadorActual.velocidadA;
    bufferTX += "SA" + dosDígitos(velocidadAActual);
}
if (bufferTX != "")
{
    registro += "Reduciendo velocidad:\r\n";
    transmitirYConfirmar(bufferTX);
}

// Centrado en la estación de centrado:
if (alimentadorActual.centradoEstación) centrarEnEstaciónDeCentrado();

// Lleva el cabezal hasta la posición de colocación del componente:
registro += "Posicionando Place:\r\n";
Point coordenadasPlace = calculaCoordenadasPlace(componenteActual);
bufferTX = "PX" + cuatroDígitos(datos.xMax - coordenadasPlace.X) + "PY" +
     cuatroDígitos(coordenadasPlace.Y);
transmitirYConfirmar(bufferTX);

// Centrado con las pinzas del cabezal:
bufferTX = "";
if (alimentadorActual.centradoPinzasX) bufferTX += "ZX";
if (alimentadorActual.centradoPinzasY) bufferTX += "ZY";
if (bufferTX != "")
{
    registro += "Centrando con pinzas:\r\n";
    transmitirYConfirmar(bufferTX);
}
```

```
        // Giro del componente y test de vacío:
        bufferTX = "";
        int anguloDeGiro = (int)placaActual.t + componenteActual.t;
        bufferTX = "PA" + cuatroDígitos(anguloDeGiro);
        bufferTX += "VT";
        transmitirYConfirmar(bufferTX);

        // Pone el componente sobre la placa y retira la aguja:
        registro += "Place:\r\n";
        transmitirYConfirmar("PZ" + cuatroDígitos(alimentadorActual.zPlace));
        transmitirYConfirmar("VA");
        transmitirYConfirmar("HZHA");

        // Pone el componente en estado Colocado y lo muestra en pantalla.
        placaActual.componentes[c].estado = estadoComponente.Colocado;
        if (checkBoxVerProducción.Checked) worker.ReportProgress(pocentajeTerminado(), p);
    }

    private Point calculaCoordenadasPlace(componente componenteActual)
    {
        // Para calcular la posición de colocación se tiene en cuenta el ángulo de giro de la placa.
        double ánguloDelVectorComponenteEnPlacaSinRotarEnRadianes = Math.Atan2(componenteActual.y,
             componenteActual.x);    // alfa
        double ánguloDeLaPlacaEnRadianes = placaActual.t * Math.PI / 200;   // beta
        double ánguloDelVectorComponenteEnPlacaRotadaEnRadianes = ánguloDeLaPlacaEnRadianes –
             ánguloDelVectorComponenteEnPlacaSinRotarEnRadianes;    //beta - alfa
        double móduloVectorComponenteEnPlaca = Math.Sqrt(Math.Pow(componenteActual.x, 2) +
             Math.Pow(componenteActual.y, 2));    // Módulo del vector c
        int coordenadaXFinal = placaActual.x + (int)(móduloVectorComponenteEnPlaca *
             Math.Cos(ánguloDelVectorComponenteEnPlacaRotadaEnRadianes));    // xf
        int coordenadaYFinal = placaActual.y - (int)(móduloVectorComponenteEnPlaca *
             Math.Sin(ánguloDelVectorComponenteEnPlacaRotadaEnRadianes));    // yf
        return new Point(coordenadaXFinal,coordenadaYFinal);
    }

    private void dispensarAdhesivo(componente componenteActual)
    {
        // Posicionar aguja del dispensador
        registro += "Dispensando adhesivo:\r\n";
        Point coordenadasDispensar = calculaCoordenadasPlace(componenteActual);
        coordenadasDispensar.X += datos.offsetDispensadorX;
        coordenadasDispensar.Y += datos.offsetDispensadorY;
        string bufferTX = "PX" + cuatroDígitos(datos.xMax - coordenadasDispensar.X) + "PY" +
             cuatroDígitos(coordenadasDispensar.Y);
        transmitirYConfirmar(bufferTX);

        // Dispensar gotas
        transmitirYConfirmar("LI" +
             dosDígitos(datos.alimentadores[componenteActual.alimentador].gotasDispensador));
    }

    private void seleccionarBanco(int nuevoBanco)
    {
        // Cambia el banco de alimentadores si no está ya puesto.
        if (nuevoBanco != bancoActual)
        {
            registro += "Cambiando a banco de alimentadores " + nuevoBanco + ":\r\n";
            switch (nuevoBanco)
            {
                case 1:
                    transmitir("FN001");
                    break;
                case 2:
                    transmitir("FN031");
                    break;
                case 3:
                    transmitir("FN061");
                    break;
            }
            esperarConfirmación(14);
            bancoActual = nuevoBanco;
        }
    }

    private void establecerVelocidadMáxima()
    {
        // Establece la velocidad máxima de movimientos del cabezal si habían sido cambiadas.
        string bufferTX = "";
```

```
        if (velocidadXActual != datos.velocidadMaxX)
        {
            velocidadXActual = datos.velocidadMaxX;
            bufferTX += "SX" + dosDígitos(velocidadXActual);
        }
        if (velocidadYActual != datos.velocidadMaxY)
        {
            velocidadYActual = datos.velocidadMaxY;
            bufferTX += "SY" + dosDígitos(velocidadYActual);
        }
        if (velocidadZActual != datos.velocidadMaxZ)
        {
            velocidadZActual = datos.velocidadMaxZ;
            bufferTX += "SZ" + dosDígitos(velocidadZActual);
        }
        if (velocidadAActual != datos.velocidadMaxA)
        {
            velocidadAActual = datos.velocidadMaxA;
            bufferTX += "SA" + dosDígitos(velocidadAActual);
        }
        if (bufferTX != "")
        {
            registro += "Aumentando velocidad:\r\n";
            transmitirYConfirmar(bufferTX);
        }
    }

    private void montarÚtil(int útilPedido)
    {
        // Coloca el útil pedido en el cabezal, desmontando el que estuviese montado previamente
        if (útilActual != útilPedido)
        {
            if (útilActual != 0) desmontarÚtilActual();
            if (útilPedido >= 1 && útilPedido <= 4)
            {
                registro += "Montando útil " + útilPedido + ":\r\n";
                transmitirYConfirmar("HZTEHA");
                transmitirYConfirmar("PX3300PY0200");
                transmitirYConfirmar("PX3" + Convert.ToString(33 + útilPedido * 12) +
                      "0PY0316PZ0170HZHATA");
                útilActual = útilPedido;
            }
        }
    }

    private void desmontarÚtilActual()
    {
        if (útilActual >= 1 && útilActual <= 4)
        {
            registro += "Desmontando útil " + útilActual + "\r\n";
            transmitirYConfirmar("TEHZHA");
            transmitirYConfirmar("PX3300PY0200");
            transmitirYConfirmar("PX3" + Convert.ToString(33 + útilActual * 12) +
                    "0PY0316PZ0170TAHZHA");
            útilActual = 0;
        }
    }

    private void centrarEnEstaciónDeCentrado()
    {
        registro += "Estación de centrado:\r\n";
        transmitirYConfirmar("PX3870PY1250PZ0140VA");   // Lleva el componente a la estación.
        transmitirYConfirmar("PZ0045ZSZS");             // Sube la aguja y activa el centrado.
        transmitirYConfirmar("PZ0150VE");               // Baja el cabezal y activa el vacío.
        transmitirYConfirmar("PZ0045");                 // Sube el cabezal.
        transmitirYConfirmar("PX3300PY1250");           // Sale de la estación de centrado.
    }

    private void desecharComponente()
    {
        registro += "Desechando componente:\r\n";
        transmitirYConfirmar("PX0020PY0020VAHZHZ");
    }

    private void transmitirYConfirmar(string bufferTX)
    {
        transmitir(bufferTX);
        esperarConfirmación();
    }
```

```
        private void transmitir(string bufferTX)
        {
            registro += "\t" + DateTime.Now.ToString() + " - Enviado: " + bufferTX + "\r\n";
            datos.puertoSerie.WriteLine(bufferTX);
        }

        private void esperarConfirmación()
        {
            // Calcula el tiempo de espera de OK en función de la velocidad de movimiento del cabezal.
            int tiempoEspera = 22 - Math.Min(velocidadXActual, velocidadYActual);
            esperarConfirmación(tiempoEspera);
        }

        private void esperarConfirmación(int tiempoEspera)
        {
            // Espera hasta recibir el código de OK desde la Cosy durante el tiempo máximo especificado.
            datos.puertoSerie.ReadTimeout = 1000 * tiempoEspera;
            while (true)
            {
                string bufferRX = "";
                bufferRX = datos.puertoSerie.ReadLine();

                registro += "\t\t" + DateTime.Now.ToString() + " - Recibido: " + bufferRX + "\r\n";
                if (bufferRX.Contains("FF00")) return;
                else if (bufferRX.Contains("FF02")) throw new PérdidaException();
                else if (bufferRX.Contains("FF23")) throw new PeligroException();
            }
        }

        private string dosDígitos(int n)
        {
            string texto = Convert.ToString(n);
            switch (texto.Length)
            {
                case 2:
                    break;
                case 1:
                    texto = "0" + texto;
                    break;
                default:
                    texto = "00";
                    break;
            }
            return texto;
        }

        private string cuatroDígitos(int n)
        {
            string texto = Convert.ToString(n);
            switch (texto.Length)
            {
                case 4:
                    break;
                case 3:
                    texto = "0" + texto;
                    break;
                case 2:
                    texto = "00" + texto;
                    break;
                case 1:
                    texto = "000" + texto;
                    break;
                default:
                    texto = "0000";
                    break;
            }
            return texto;
        }

        private int pocentajeTerminado()
        {
            return (int)(componentesTratados * 100 / totalPendientes);
        }
    }
}


// FormConfiguracionPuerto.cs
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace PnPWork
{
    public partial class FormConfiguracionPuerto : Form
    {
        public FormConfiguracionPuerto()
        {
            InitializeComponent();

            // Rellena el comboBox con los nombres de los puertos instalados enel sistema y lo ordena.
            foreach (string puerto in SerialPort.GetPortNames())
            {
                comboBoxPuertoSerie.Items.Add(puerto);
            }
            comboBoxPuertoSerie.Sorted = true;

            // Selecciona el puerto serie preconfigurado. Si no existe selecciona el primero de la lista.
            comboBoxPuertoSerie.SelectedItem = datos.nombrePuertoSerie;
            if ((comboBoxPuertoSerie.Items.Count > 0) && (comboBoxPuertoSerie.SelectedIndex == -1))
            {
                comboBoxPuertoSerie.SelectedIndex = 0;
                datos.nombrePuertoSerie = comboBoxPuertoSerie.Text;
            }
        }

        private void buttonAceptar_Click(object sender, EventArgs e)
        {
            // Intenta abrir el puerto serie seleccionado y si no puede avisa al ususario.
            datos.nombrePuertoSerie = comboBoxPuertoSerie.Text;
            if (!datos.abrirPuertoSerie()) MessageBox.Show("Ha habido un error al abrir el puerto serie "
                 + datos.nombrePuertoSerie, "Puerto serie no disponible", MessageBoxButtons.OK,
                 MessageBoxIcon.Error);
            else this.Close();
        }
    }
}


// FormConfiguraciónCámara.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using AForge.Video;
using AForge.Video.DirectShow;

namespace PnPWork
{
    public partial class FormCámara : Form
    {
        FilterInfoCollection dispositivos;
        VideoCaptureDevice[] cámara = new VideoCaptureDevice[4];
        int numeroCámaras = 0;
        Font tipografía = new Font("Arial", 8);
        Brush pincelTexto = Brushes.Orange;
        string[] nombreCámara = new string[4];
        string[] fpsCámara = new string[4];
        string[] throughputCámara = new string[4];
        string[] fpsMáxCámara = new string[4];
        Size[] resoluciónCámara = new Size[4];

        public FormCámara()
        {
            InitializeComponent();
        }

        private void timer1_Tick(object sender, EventArgs e)
```

```
{
    for (int n = 0; n < cámara.Length; n++)
    {
        if ((cámara[n] != null) && (cámara[n].IsRunning))
        {
            fpsCámara[n] = cámara[n].FramesReceived.ToString();
            throughputCámara[n] = ((float)cámara[n].BytesReceived / 1000000).ToString();
        }
    }
    buscaCámaras();
}

private void buscaCámaras()
{
    dispositivos = new FilterInfoCollection(FilterCategory.VideoInputDevice);
    if (numeroCámaras != dispositivos.Count)
    {
        numeroCámaras = dispositivos.Count;
        for (int n = 0; n < numeroCámaras; n++)
        {
            if ((cámara[n] != null) && (!cámara[n].Equals(new
                  VideoCaptureDevice(dispositivos[n].MonikerString)))) cámara[n].SignalToStop();
            cámara[n] = new VideoCaptureDevice(dispositivos[n].MonikerString);
            nombreCámara[n] = dispositivos[n].Name;
            switch (n)
            {
                case 0:
                    cámara[0].NewFrame += new NewFrameEventHandler(cámara0_NuevoFrame);
                    cámara[0].DesiredFrameSize = pictureBoxCámara0.Size;
                    buttonCámara0.Text = nombreCámara[0];
                    buttonCámara0.Enabled = true;
                    break;
                case 1:
                    cámara[1].NewFrame += new NewFrameEventHandler(cámara1_NuevoFrame);
                    cámara[1].DesiredFrameSize = pictureBoxCámara1.Size;
                    buttonCámara1.Text = nombreCámara[1];
                    buttonCámara1.Enabled = true;
                    break;
                case 2:
                    cámara[2].NewFrame += new NewFrameEventHandler(cámara2_NuevoFrame);
                    cámara[2].DesiredFrameSize = pictureBoxCámara2.Size;
                    buttonCámara2.Text = nombreCámara[2];
                    buttonCámara2.Enabled = true;
                    break;
                case 3:
                    cámara[3].NewFrame += new NewFrameEventHandler(cámara3_NuevoFrame);
                    cámara[3].DesiredFrameSize = pictureBoxCámara3.Size;
                    buttonCámara3.Text = nombreCámara[3];
                    buttonCámara3.Enabled = true;
                    break;
            }
            fpsMáxCámara[n] = cámara[n].VideoCapabilities[0].FrameRate.ToString();
            resoluciónCámara[n] = cámara[n].VideoCapabilities[0].FrameSize;
            cámara[n].DesiredFrameSize = new Size(352, 288);
            cámara[n].Start();
        }
        for (int n = numeroCámaras; n < 4; n++)
        {
            cámara[n] = null;
            switch (n)
            {
                case 0:
                    buttonCámara0.Text = "";
                    buttonCámara0.Enabled = false;
                    break;
                case 1:
                    buttonCámara1.Text = "";
                    buttonCámara1.Enabled = false;
                    break;
                case 2:
                    buttonCámara2.Text = "";
                    buttonCámara2.Enabled = false;
                    break;
                case 3:
                    buttonCámara3.Text = "";
                    buttonCámara3.Enabled = false;
                    break;
            }
        }
```

```
        }
        dispositivos = null;
    }

    private void cámara0_NuevoFrame(object sender, NewFrameEventArgs eventArgs)
    {
        Bitmap imagen = (Bitmap)eventArgs.Frame.Clone();
        if (checkBoxMostrarDatos.Checked)
        {
            Graphics lienzo = Graphics.FromImage(imagen);
            string s = nombreCámara[0] + "\n" + fpsCámara[0] + " FPS\nThroughput: " +
                       throughputCámara[0] + " MBps\nFPS máximo: " + fpsMáxCámara[0] + "\nResolución
                       máxima: " + resoluciónCámara[0].Width + " x " + resoluciónCámara[0].Height;
            lienzo.DrawString(s, (Font)tipografía.Clone(), (Brush)pincelTexto.Clone(),
                       new Point(10, 10));
        }
        pictureBoxCámara0.Image = imagen;
    }

    private void cámara1_NuevoFrame(object sender, NewFrameEventArgs eventArgs)
    {
        Bitmap imagen = (Bitmap)eventArgs.Frame.Clone();
        if (checkBoxMostrarDatos.Checked)
        {
            Graphics lienzo = Graphics.FromImage(imagen);
            string s = nombreCámara[1] + "\n" + fpsCámara[1] + " FPS\nThroughput: " +
                       throughputCámara[1] + " MBps\nFPS máximo: " + fpsMáxCámara[1] + "\nResolución
                       máxima: " + resoluciónCámara[1].Width + " x " + resoluciónCámara[1].Height;
            lienzo.DrawString(s, (Font)tipografía.Clone(), (Brush)pincelTexto.Clone(),
                       new Point(10, 10));
        }
        pictureBoxCámara1.Image = imagen;
    }

    private void cámara2_NuevoFrame(object sender, NewFrameEventArgs eventArgs)
    {
        Bitmap imagen = (Bitmap)eventArgs.Frame.Clone();
        if (checkBoxMostrarDatos.Checked)
        {
            Graphics lienzo = Graphics.FromImage(imagen);
            string s = nombreCámara[2] + "\n" + fpsCámara[2] + " FPS\nThroughput: " +
                       throughputCámara[2] + " MBps\nFPS máximo: " + fpsMáxCámara[2] + "\nResolución
                       máxima: " + resoluciónCámara[2].Width + " x " + resoluciónCámara[2].Height;
            lienzo.DrawString(s, (Font)tipografía.Clone(), (Brush)pincelTexto.Clone(),
                       new Point(10, 10));
        }
        pictureBoxCámara2.Image = imagen;
    }

    private void cámara3_NuevoFrame(object sender, NewFrameEventArgs eventArgs)
    {
        Bitmap imagen = (Bitmap)eventArgs.Frame.Clone();
        if (checkBoxMostrarDatos.Checked)
        {
            Graphics lienzo = Graphics.FromImage(imagen);
            string s = nombreCámara[3] + "\n" + fpsCámara[3] + " FPS\nThroughput: " +
                       throughputCámara[1] + " MBps\nFPS máximo: " + fpsMáxCámara[3] + "\nResolución
                       máxima: " + resoluciónCámara[3].Width + " x " + resoluciónCámara[3].Height;
            lienzo.DrawString(s, (Font)tipografía.Clone(), (Brush)pincelTexto.Clone(), new Point(10,
                       10));
        }
        pictureBoxCámara3.Image = imagen;
    }

    private void pictureBoxCámara0_Click(object sender, EventArgs e)
    {
        if (cámara[0] != null)
        {
            cámara[0].DisplayPropertyPage(this.Handle);
        }
    }

    private void pictureBoxCámara1_Click(object sender, EventArgs e)
    {
        if (cámara[1] != null)
        {
            cámara[1].DisplayPropertyPage(this.Handle);
        }
    }
```

```
        private void pictureBoxCámara2_Click(object sender, EventArgs e)
        {
            if (cámara[2] != null)
            {
                cámara[2].DisplayPropertyPage(this.Handle);
            }
        }

        private void pictureBoxCámara3_Click(object sender, EventArgs e)
        {
            if (cámara[3] != null)
            {
                cámara[3].DisplayPropertyPage(this.Handle);
            }
        }

        private void FormCámara_FormClosing(object sender, FormClosingEventArgs e)
        {
            for (int n = 0; n < cámara.Length; n++)
            {
                if ((cámara[n] != null) && (cámara[n].IsRunning))
                {
                    cámara[n].SignalToStop();
                    cámara[n] = null;
                }
            }
        }

        private void buttonCámara0_Click(object sender, EventArgs e)
        {
            datos.cámara = cámara[0];
            datos.nombreCámara = nombreCámara[0];
            this.Close();
        }

        private void buttonCámara1_Click(object sender, EventArgs e)
        {
            datos.cámara = cámara[1];
            datos.nombreCámara = nombreCámara[1];
            this.Close();
        }

        private void buttonCámara2_Click(object sender, EventArgs e)
        {
            datos.cámara = cámara[2];
            datos.nombreCámara = nombreCámara[2];
            this.Close();
        }

        private void buttonCámara3_Click(object sender, EventArgs e)
        {
            datos.cámara = cámara[3];
            datos.nombreCámara = nombreCámara[3];
            this.Close();
        }

    }
}


// FormAcercaDe.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PnPWork
{
    public partial class FormAcercaDe : Form
    {
        public FormAcercaDe()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}


// FormControlManualConCámara.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using AForge.Video;
using AForge.Video.DirectShow;

namespace PnPWork
{
    public partial class FormControlManualConCámara : Form
    {
        int xCámara = 0;
        int yCámara = 0;
        int incremento = 100;
        Size ajusteImagen;
        devolverCoordenadasCallback devolverCoordenadas = null;
        public VideoCaptureDevice cámaraActual = null;

        public FormControlManualConCámara()
        {
            InitializeComponent();

            ajusteImagen = this.Size - pictureBoxCámara.Size;
            if (!datos.puertoSerieDisponible)
            {
                MessageBox.Show("No se ha seleccionado un puerto serie.\n\nLa comunicación con la máquina
                         pick & place no será posible mientras no se configure un puerto serie válido
                         (menú configuración).", "Puerto serie no disponible", MessageBoxButtons.OK,
                         MessageBoxIcon.Error);
                this.Close();
            }
            else
            {
                // Inicializa la Cosy
                transmitir("OO");
                transmitir("VATAT10030T20010T30020T40030T50150SX09SY09SZ10PZ0000");
            }
        }

        private void FormControlManualConCámara_Load(object sender, EventArgs e)
        {
            cámaraActual = datos.cámara;
            if (cámaraActual != null)
            {
                cámaraActual.NewFrame += new NewFrameEventHandler(video_NewFrame);
                cámaraActual.DesiredFrameSize = pictureBoxCámara.ClientSize;
                cámaraActual.Start();
                pictureBoxCámara.BackgroundImageLayout = ImageLayout.Stretch;
            }
            else
            {
                MessageBox.Show("No se ha seleccionado una cámara.\n\nPara usar esta función correctamente
                         debe seleccionar la cámara de la máquina pick & place (menú configuración).",
                         "Cámara no disponible", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void FormControlManualConCámara_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (!(cámaraActual == null) && cámaraActual.IsRunning)
            {
                cámaraActual.NewFrame -= video_NewFrame;
                cámaraActual = null;
            }
            if (datos.puertoSerieDisponible)
```

```
        {
            datos.puertoSerie.DiscardInBuffer();
            datos.puertoSerie.DiscardOutBuffer();
        }
    }

    private void FormControlManualConCámara_Resize(object sender, EventArgs e)
    {
        pictureBoxCámara.Height = this.Height - ajusteImagen.Height;
        pictureBoxCámara.Width = this.Width - ajusteImagen.Width;
        cámaraActual.DesiredFrameSize = pictureBoxCámara.ClientSize;
    }

    private void video_NewFrame(object sender, NewFrameEventArgs eventArgs)
    {
        pictureBoxCámara.BackgroundImage = (Bitmap)eventArgs.Frame.Clone();
        if (cámaraActual != datos.cámara) cámaraActual = datos.cámara;
    }

    private void buttonMarcaDeCalibración_Click(object sender, EventArgs e)
    {
        IrACoordenadas(datos.marcaCalibraciónX, datos.marcaCalibraciónY);
    }

    private void numericUpDownCambiarBanco_ValueChanged(object sender, EventArgs e)
    {
        switch ((int)numericUpDownCambiarBanco.Value)
        {
            case 1:
                transmitir("FN001");
                break;
            case 2:
                transmitir("FN031");
                break;
            case 3:
                transmitir("FN061");
                break;
        }
    }

    private void botonIR_Click(object sender, EventArgs e)
    {
        IrACoordenadas(numericUpDownX.Value, numericUpDownY.Value);
    }

    private void botonArriba_Click(object sender, EventArgs e)
    {
        IrACoordenadas(xCámara, yCámara - incremento);
    }

    private void botonAbajo_Click(object sender, EventArgs e)
    {
        IrACoordenadas(xCámara, yCámara + incremento);
    }

    private void botonIzquierda_Click(object sender, EventArgs e)
    {
        IrACoordenadas(xCámara - incremento, yCámara);
    }

    private void botonDerecha_Click(object sender, EventArgs e)
    {
        IrACoordenadas(xCámara + incremento, yCámara);
    }

    private void buttonVolver_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void FormControlManualConCámara_KeyDown(object sender, KeyEventArgs e)
    {
        switch (e.KeyData)
        {
            case Keys.Up:
                botonArriba.PerformClick();
                break;
            case Keys.Down:
                botonAbajo.PerformClick();
```

```
                break;
            case Keys.Left:
                botonIzquierda.PerformClick();
                break;
            case Keys.Right:
                botonDerecha.PerformClick();
                break;
        }
    }

    private void radioButton01mm_CheckedChanged(object sender, EventArgs e)
    {
        if (radioButton01mm.Checked) incremento = 1;
    }

    private void radioButton1mm_CheckedChanged(object sender, EventArgs e)
    {
        if (radioButton1mm.Checked) incremento = 10;
    }

    private void radioButton10mm_CheckedChanged(object sender, EventArgs e)
    {
        if (radioButton10mm.Checked) incremento = 100;
    }

    private void radioButton100mm_CheckedChanged(object sender, EventArgs e)
    {
        if (radioButton100mm.Checked) incremento = 1000;
    }

    private void pictureBoxCámara_MouseDown(object sender, MouseEventArgs e)
    {
        if (e.Button == System.Windows.Forms.MouseButtons.Left)
        {
            int saltoX = e.X * 290 / pictureBoxCámara.Width - 130;
            int saltoY = e.Y * 215 / pictureBoxCámara.Height - 105;
            IrACoordenadas(xCámara + saltoX, yCámara + saltoY);
        }
        else if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
            if (cámaraActual != null) cámaraActual.DisplayPropertyPage(this.Handle);
        }
    }

    public void IrACoordenadas(decimal x, decimal y)
    {
        IrACoordenadas((int)(x * 10), (int)(y * 10));
    }

    public void IrACoordenadas(int x, int y)
    {
        // Limita el movimiento del cabezal a valores alcanzables.
        xCámara = Math.Max(datos.offsetCamaraX, Math.Min(datos.xMax, x));
        numericUpDownX.Value = (decimal)xCámara / 10;
        yCámara = Math.Max(datos.offsetCamaraY, Math.Min(datos.yMax, y));
        numericUpDownY.Value = (decimal)yCámara / 10;
        moverCabezalYDevolverCoordenadas();
    }

    private void moverCabezalYDevolverCoordenadas()
    {
        int xCabezal = xCámara - datos.offsetCamaraX;
        int yCabezal = yCámara - datos.offsetCamaraY;
        transmitir("PX" + cuatroDígitos(3900 - xCabezal) + "PY" + cuatroDígitos(yCabezal));
        if (devolverCoordenadas != null) devolverCoordenadas(xCámara, yCámara);
    }

    private void transmitir(string bufferTX)
    {
        if (datos.puertoSerieDisponible) datos.puertoSerie.WriteLine(bufferTX);
    }

    private string cuatroDígitos(int n)
    {
        string texto = Convert.ToString(n);
        switch (texto.Length)
        {
            case 4:
                break;
```

```
                case 3:
                    texto = "0" + texto;
                    break;
                case 2:
                    texto = "00" + texto;
                    break;
                case 1:
                    texto = "000" + texto;
                    break;
                default:
                    texto = "0000";
                    break;
            }
            return texto;
        }

        public void registrarCallback(devolverCoordenadasCallback métodoParaDevolverCoordenadas)
        {
            devolverCoordenadas = métodoParaDevolverCoordenadas;
        }
    }
}


// Gerber.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using System.Text.RegularExpressions;
using System.IO;
using System.Drawing;

namespace PnPWork
{
    class Gerber
    {
        public static float escala = 1;
        public Color colorDeFondo = Color.Transparent;
        public Color colorDeDibujo = Color.Blue;
        public double xmin = 0;
        public double ymin = 0;
        public double xmax = 0;
        public double ymax = 0;

        ArrayList aperturas = new ArrayList();
        ArrayList elementosGráficos = new ArrayList();
        bool leadingZerosOmitidos = true;
        int enteros = 3;
        int decimales = 5;
        bool enPulgadas = true;

        public bool interpretarFichero(string rutaFichero)
        {
            try
            {
                double xi = 0;
                double yi = 0;
                double xf = 0;
                double yf = 0;
                int aperturaActual = 0;
                bool modoRegión = false;
                bool modoDark = true;
                bool interpolaciónLineal = true;
                double signoGiro = -1;   // Dirección del giro de la interpolación circular: 1=cw, -1=ccw

                FileInfo fi = new FileInfo(rutaFichero);
                if (fi.Exists)
                {
                    StreamReader sr = fi.OpenText();
                    string linea;
                    while (!sr.EndOfStream)
                    {
                        linea = sr.ReadLine();

                        // Formato de los datos
                        if (linea.Contains("%MOIN")) enPulgadas = true;
                        if (linea.Contains("%MOMM")) enPulgadas = false;
```

```
if (linea.Contains("%FS"))
{
    int índice = linea.IndexOf("%FS") + 3;
    leadingZerosOmitidos = linea[índice] == 'L';
    Match digitosXEncontrado = Regex.Match(linea.Substring(índice + 2),
          @"X*\d{2}");
    if (digitosXEncontrado.Success)
    {
        enteros = int.Parse((digitosXEncontrado.Value)[1].ToString());
    }
}

// Aperturas
Match adEncontrado = Regex.Match(linea, @"%ADD*\d{1,4}");
if (adEncontrado.Success)
{
    datosApertura ap = new datosApertura();
    switch (linea.Substring(adEncontrado.Index + adEncontrado.Length, 2))
    {
        case "C,":
            ap.tipo = tipoApertura.circular;

            int indiceDatoC = linea.IndexOf(",") + 1;
            int longitudDatoC = linea.IndexOf("*") - indiceDatoC;

            ap.x = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoC,
                  longitudDatoC).Replace('.', ',')));
            ap.y = 0;
            aperturas.Add(ap);
            break;
        case "R,":
        case "O,":
            ap.tipo = tipoApertura.rectangular;

            int indiceDatoX = linea.IndexOf(",") + 1;
            int longitudDatoX = linea.IndexOf("X") - indiceDatoX;
            ap.x = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoX,
                  longitudDatoX).Replace('.', ',')));

            int indiceDatoY = linea.IndexOf("X") + 1;
            int longitudDatoY = linea.IndexOf("*") - indiceDatoY;

            ap.y = convertirAmm(Convert.ToDouble(linea.Substring(indiceDatoY,
                  longitudDatoY).Replace('.', ',')));
            aperturas.Add(ap);
            break;
        default:
            ap.tipo = tipoApertura.desconocida;
            ap.x = 0;
            ap.y = 0;
            aperturas.Add(ap);
            break;
    }
}

// Coordenadas
Match xEncontrado = Regex.Match(linea, @"X-*\d{1," + (enteros +
  decimales).ToString() + "}");
if (xEncontrado.Success)
{
    xi = xf;
    xf = convertirAmm(convertirANúmero(xEncontrado.Value.Substring(1)));
    xmin = xf < xmin ? xf : xmin;
    xmax = xf > xmax ? xf : xmax;
}
Match yEncontrado = Regex.Match(linea, @"Y-*\d{1," + (enteros +
  decimales).ToString() + "}");
if (yEncontrado.Success)
{
    yi = yf;
    yf = convertirAmm(convertirANúmero(yEncontrado.Value.Substring(1)));
    ymin = yf < ymin ? yf : ymin;
    ymax = yf > ymax ? yf : ymax;
}

// Polaridad
if (linea.Contains("%LPC")) modoDark = false;
if (linea.Contains("%LPD")) modoDark = true;
```

```
// Regiones
if (linea.Contains("G36*"))
{
    modoRegión = true;
    datosRegión región = new datosRegión();
    región.x = new ArrayList();
    región.y = new ArrayList();

    datosGráficos gráfico;
    gráfico.tipo = tipoGráfico.región;
    gráfico.datos = región;
    gráfico.colorInvertido = modoDark;
    elementosGráficos.Add(gráfico);
}
if (linea.Contains("G37*")) modoRegión = false;

// Tipo de interpolación
if (linea.Contains("G01*")) interpolaciónLineal = true;
if (linea.Contains("G03*"))
{
    interpolaciónLineal = false;
    signoGiro = -1;
}
if (linea.Contains("G02*"))
{
    interpolaciónLineal = false;
    signoGiro = 1;
}

// Ejecución de pistas / regiones
if (modoRegión)
{
    if (linea.Contains("D01"))
    {
        datosGráficos gráfico =
          (datosGráficos)elementosGráficos[elementosGráficos.Count - 1];
        ((datosRegión)gráfico.datos).x.Add(xf);
        ((datosRegión)gráfico.datos).y.Add(yf);
    }
    if (linea.Contains("D02"))
    {
        datosRegión región = new datosRegión();
        región.x = new ArrayList();
        región.y = new ArrayList();

        datosGráficos gráfico;
        gráfico.tipo = tipoGráfico.región;
        gráfico.datos = región;
        gráfico.colorInvertido = modoDark;
        elementosGráficos.Add(gráfico);
    }
}
else
{
    if ((linea.Contains("D01")) && interpolaciónLineal)
    {
        if (!xEncontrado.Success) xi = xf;
        if (!yEncontrado.Success) yi = yf;
        datosPista pista;
        pista.xi = xi;
        pista.yi = yi;
        pista.xf = xf;
        pista.yf = yf;
        datosApertura aperturaAUsar = (datosApertura)aperturas[aperturaActual];
        pista.ancho = aperturaAUsar.x;

        datosGráficos gráfico;
        gráfico.tipo = tipoGráfico.draw;
        gráfico.datos = pista;
        gráfico.colorInvertido = modoDark;
        elementosGráficos.Add(gráfico);
    }
    if ((linea.Contains("D01")) && !interpolaciónLineal)
    {
        double i = 0;
        double j = 0;
        if (!xEncontrado.Success) xi = xf;
        if (!yEncontrado.Success) yi = yf;
        Match iEncontrado = Regex.Match(linea, @"I-*\d{1," + (enteros +
```

```
                        decimales).ToString() + "}");
                    if (iEncontrado.Success) i =
                        convertirAmm(convertirANúmero(iEncontrado.Value.Substring(1)));
                    Match jEncontrado = Regex.Match(linea, @"J-*\d{1," + (enteros +
                        decimales).ToString() + "}");
                    if (jEncontrado.Success) j =
                        convertirAmm(convertirANúmero(jEncontrado.Value.Substring(1)));
                    datosArco arco;
                    arco.diámetro = Math.Sqrt(Math.Pow(i, 2) + Math.Pow(j, 2)) * 2;
                    arco.xOrigen = xi + i - arco.diámetro / 2;
                    arco.yOrigen = yi + j + arco.diámetro / 2;
                    arco.ánguloInicial = - Math.Atan2(-j, -i) * (180 / Math.PI);
                    arco.ánguloABarrer = Math.Abs(Math.Atan2(yf - (yi + j), xf - (xi + i)) *
                        (180 / Math.PI) - arco.ánguloInicial);
                    arco.ánguloABarrer = signoGiro * Math.Min((arco.ánguloABarrer < 0.5) ? 360
                        : arco.ánguloABarrer, 360);
                    datosApertura aperturaAUsar = (datosApertura)aperturas[aperturaActual];
                    arco.ancho = aperturaAUsar.x;

                    datosGráficos gráfico;
                    gráfico.tipo = tipoGráfico.arco;
                    gráfico.datos = arco;
                    gráfico.colorInvertido = modoDark;
                    elementosGráficos.Add(gráfico);
                }
            }

            // Ejecución de pads
            if (linea.Contains("D03"))
            {
                datosFlash flash;
                flash.x = xf;
                flash.y = yf;
                flash.apertura = (datosApertura)aperturas[aperturaActual];

                // Actualiza dimensiones de la placa
                datosApertura aperturaUsada = (datosApertura)aperturas[aperturaActual];
                double ancho = aperturaUsada.x;// *2;
                double alto = aperturaUsada.y;// *2;
                xmin = xf - ancho < xmin ? xf - ancho : xmin;
                xmax = xf + ancho > xmax ? xf + ancho : xmax;
                ymin = yf - alto < ymin ? yf - alto : ymin;
                ymax = yf + alto > ymax ? yf + alto : ymax;

                datosGráficos gráfico;
                gráfico.tipo = tipoGráfico.flash;
                gráfico.datos = flash;
                gráfico.colorInvertido = modoDark;
                elementosGráficos.Add(gráfico);
            }

            // Cambio de apertura
            for (int n = 0; n < aperturas.Count; n++)
            {
                string aperturaBuscada = "D" + (n + 10).ToString();
                if (linea.Contains(aperturaBuscada)) aperturaActual = n;
            }
        }
        return true;
    }
    else
    {
        return false;
    }
}
catch
{
    return false;
}
}

private double convertirANúmero(string cadena)
{
    double signo = cadena[0] == '-' ? -1 : 1;
    cadena = (cadena[0] == '-' || cadena[0] == '+') ? cadena.Substring(1) : cadena;
    if (leadingZerosOmitidos) cadena = cadena.PadLeft(enteros + decimales, '0');
    else cadena = cadena.PadRight(enteros + decimales, '0');
    cadena = cadena.Insert(enteros, ",");
    return Convert.ToDouble(cadena) * signo;
```

```
}

private double convertirAmm(double cantidad)
{
    if (enPulgadas) return cantidad * 25.4;
    return cantidad;
}

public void dibujarImagen(ref Bitmap imgBuffer)
{
    Graphics buffer = Graphics.FromImage(imgBuffer);
    buffer.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
    Pen pluma = new System.Drawing.Pen(colorDeDibujo);
    pluma.EndCap = System.Drawing.Drawing2D.LineCap.Round;
    pluma.StartCap = System.Drawing.Drawing2D.LineCap.Round;
    SolidBrush rellenoDibujo = new SolidBrush(colorDeDibujo);
    SolidBrush rellenoFondo = new SolidBrush(colorDeFondo);
    SolidBrush relleno;

    // Dibujar objetos gráficos
    for (int n = 0; n < elementosGráficos.Count; n++)
    {
        datosGráficos gráfico = (datosGráficos)elementosGráficos[n];
        switch (gráfico.tipo)
        {
            case tipoGráfico.flash:
                datosFlash flash = (datosFlash)gráfico.datos;
                relleno = gráfico.colorInvertido ? rellenoDibujo : rellenoFondo;
                switch (flash.apertura.tipo)
                {
                    case tipoApertura.circular:
                        double radio = flash.apertura.x;
                        buffer.FillEllipse(relleno, (float)((flash.x - radio / 2) * escala),
                          (float)(imgBuffer.Height - (flash.y + radio / 2) * escala),
                          (float)(radio * escala), (float)(radio * escala));
                        break;
                    case tipoApertura.rectangular:
                        double ancho = flash.apertura.x;
                        double alto = flash.apertura.y;
                        buffer.FillRectangle(relleno, (float)((flash.x - ancho / 2) * escala),
                          (float)(imgBuffer.Height - (flash.y + alto / 2) * escala), (float)(ancho
                          * escala), (float)(alto * escala));
                        break;
                }
                break;
            case tipoGráfico.draw:
                datosPista pista = (datosPista)gráfico.datos;
                pluma.Width = (float)(pista.ancho * escala);
                pluma.Color = gráfico.colorInvertido ? colorDeDibujo : colorDeFondo;
                buffer.DrawLine(pluma, (float)pista.xi * escala, imgBuffer.Height –
                  (float)pista.yi * escala, (float)pista.xf * escala, imgBuffer.Height –
                  (float)pista.yf * escala);
                break;
            case tipoGráfico.región:
                datosRegión región = (datosRegión)gráfico.datos;
                PointF[] puntos = new PointF[región.x.Count];
                for (int i = 0; i < región.x.Count; i++)
                {
                    puntos[i].X = (float)((double)región.x[i] * escala);
                    puntos[i].Y = (float)(imgBuffer.Height - (double)región.y[i] * escala);
                }
                if (puntos.Length > 1)
                {
                    relleno = gráfico.colorInvertido ? rellenoDibujo : rellenoFondo;
                    buffer.FillPolygon(relleno, puntos);
                }
                break;
            case tipoGráfico.arco:
                datosArco arco = (datosArco)gráfico.datos;
                pluma.Width = (float)(arco.ancho * escala);
                pluma.Color = gráfico.colorInvertido ? colorDeDibujo : colorDeFondo;
                buffer.DrawArc(pluma, (float)arco.xOrigen * escala, imgBuffer.Height –
                  (float)arco.yOrigen * escala, (float)arco.diámetro * escala,
                  (float)arco.diámetro * escala, (float)arco.ánguloInicial,
                  (float)arco.ánguloABarrer);
                break;
        }
    }
}
```

```
    }
}


// FormAsignarGerbers.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace PnPWork
{
    public partial class FormAsignarGerbers : Form
    {
        Gerber overlay;
        Gerber solder;
        Gerber layer;
        Color colorFondo = Color.FromArgb(0, 50, 0);
        Point origenPictureBox = new Point();
        Size tamañoPictureBox = new Size();
        Size separaciónPictureBox;
        int separaciónButtonX;
        int separaciónButtonColor;
        int anchoLabel;
        double altoDibujo;

        public FormAsignarGerbers()
        {
            InitializeComponent();
            origenPictureBox = pictureBoxGerber.Location;
            tamañoPictureBox = pictureBoxGerber.Size;
            separaciónPictureBox.Width = this.Width - pictureBoxGerber.Width;
            separaciónPictureBox.Height = this.Height - pictureBoxGerber.Height;
            separaciónButtonColor = groupBoxGerber.Width - buttonColorOverlay.Left;
            separaciónButtonX = groupBoxGerber.Width - buttonQuitarOverlay.Left;
            anchoLabel = groupBoxGerber.Width - labelOverlay.Width;

            // Rellena el comboBox con las placas existentes y selecciona la primera
            rellenarCombo();
            if (comboBoxPlacas.Items.Count > 0)
            {
                comboBoxPlacas.SelectedIndex = 0;
                rellenarCampos();
            }
            else
            {
                this.Close();   // Cierra la ventana si no hay placas.
            }
        }

        private void rellenarCombo()
        {
            comboBoxPlacas.Items.Clear();
            for (int p = 0; p < datos.placas.Length; p++)
            {
                comboBoxPlacas.Items.Add((p + 1).ToString() + " -> " + datos.placas[p].nombre);
            }
        }

        private void ajustarPictureBox()
        {
            double anchoDibujo = 0;
            altoDibujo = 0;

            if (overlay == null)
            {
                overlay = new Gerber();
                overlay.interpretarFichero(datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero);
            }
            anchoDibujo = overlay.xmax;
            altoDibujo = overlay.ymax;

            if (solder == null)
```

```
        {
            solder = new Gerber();
            solder.interpretarFichero(datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero);
        }
        anchoDibujo = Math.Max(anchoDibujo, solder.xmax);
        altoDibujo = Math.Max(altoDibujo, solder.ymax);

        if (layer == null)
        {
            layer = new Gerber();
            layer.interpretarFichero(datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero);
        }
        anchoDibujo = Math.Max(anchoDibujo, layer.xmax);
        altoDibujo = Math.Max(altoDibujo, layer.ymax);

        int ancho = tamañoPictureBox.Width;
        int alto = tamañoPictureBox.Height;
        if ((anchoDibujo > 0) && (altoDibujo > 0))
        {
            double relaciónAspectoPictureBox = (double)tamañoPictureBox.Width /
                      (double)tamañoPictureBox.Height;
            double relaciónAspectoDibujo = anchoDibujo / altoDibujo;
            if (relaciónAspectoPictureBox > relaciónAspectoDibujo)
            {
                Gerber.escala = (float)(tamañoPictureBox.Height / altoDibujo);
                ancho = (int)(anchoDibujo * Gerber.escala);
                alto = tamañoPictureBox.Height;
            }
            else
            {
                Gerber.escala = (float)(tamañoPictureBox.Width / anchoDibujo);
                ancho = tamañoPictureBox.Width;
                alto = (int)(altoDibujo * Gerber.escala);
            }
            pictureBoxGerber.Left = origenPictureBox.X + (tamañoPictureBox.Width - ancho) / 2;
            pictureBoxGerber.Top = origenPictureBox.Y + (tamañoPictureBox.Height - alto) / 2;
        }
        pictureBoxGerber.Height = alto;
        pictureBoxGerber.Width = ancho;

    }

    public void rellenarCampos()
    {
        ajustarPictureBox();

        // Rellena los campos con los datos gerber de la placa seleccionada y dibuja la placa
        Bitmap imagen = new Bitmap(pictureBoxGerber.Width, pictureBoxGerber.Height);

        if (File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero) ||
            File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero) ||
            File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero))
        {
            Graphics lienzo = Graphics.FromImage(imagen);
            lienzo.Clear(colorFondo);
        }

        if (File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero))
        {
            labelLayer.Text = datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero;
            buttonColorLayer.BackColor = datos.placas[comboBoxPlacas.SelectedIndex].layer.color;
            buttonQuitarLayer.Enabled = true;
            layer.colorDeDibujo = datos.placas[comboBoxPlacas.SelectedIndex].layer.color;
            layer.dibujarImagen(ref imagen);
        }
        else
        {
            labelLayer.Text = "";
        }

        if (File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero))
        {
            labelSolder.Text = datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero;
            buttonColorSolder.BackColor = datos.placas[comboBoxPlacas.SelectedIndex].solder.color;
            buttonQuitarSolder.Enabled = true;
            solder.colorDeDibujo = datos.placas[comboBoxPlacas.SelectedIndex].solder.color;
            solder.dibujarImagen(ref imagen);
        }
        else
```

```
        {
            labelSolder.Text = "";
        }

        if (File.Exists(datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero))
        {
            labelOverlay.Text = datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero;
            buttonColorOverlay.BackColor = datos.placas[comboBoxPlacas.SelectedIndex].overlay.color;
            buttonQuitarOverlay.Enabled = true;
            overlay.colorDeDibujo = datos.placas[comboBoxPlacas.SelectedIndex].overlay.color;
            overlay.dibujarImagen(ref imagen);
        }
        else
        {
            labelOverlay.Text = "";
        }

        checkBoxInvertir.Checked = datos.placas[comboBoxPlacas.SelectedIndex].invertir;

        numericUpDownDesplazamientoX.Value =
                    datos.placas[comboBoxPlacas.SelectedIndex].desplazamientoGerber.X;
        numericUpDownDesplazamientoY.Value =
                    datos.placas[comboBoxPlacas.SelectedIndex].desplazamientoGerber.Y;

        if (datos.placas[comboBoxPlacas.SelectedIndex].invertir)
        {
            imagen.RotateFlip(RotateFlipType.RotateNoneFlipX);
        }

        pictureBoxGerber.Image = imagen;
    }

    private void buttonOverlay_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Archivos Gerber|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero = openFileDialog.FileName;
            overlay = null;
            if (checkBoxRellenarAutomáticamente.Checked)
                    buscarFicherosSimilares(openFileDialog.FileName);
            rellenarCampos();
            datos.modificados = true;
            numericUpDownDesplazamientoY.Value = (int)(altoDibujo * 10) -
                    datos.placas[comboBoxPlacas.SelectedIndex].anchoY;
        }
    }

    private void buttonSolder_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Archivos Gerber|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero = openFileDialog.FileName;
            solder = null;
            if (checkBoxRellenarAutomáticamente.Checked)
                    buscarFicherosSimilares(openFileDialog.FileName);
            rellenarCampos();
            datos.modificados = true;
            numericUpDownDesplazamientoY.Value = (int)(altoDibujo * 10) -
                    datos.placas[comboBoxPlacas.SelectedIndex].anchoY;
        }
    }

    private void buttonLayer_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Archivos Gerber|*.*";
        if (openFileDialog.ShowDialog(this) == DialogResult.OK)
        {
            datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero = openFileDialog.FileName;
            layer = null;
            if (checkBoxRellenarAutomáticamente.Checked)
             buscarFicherosSimilares(openFileDialog.FileName);
            rellenarCampos();
            datos.modificados = true;
            numericUpDownDesplazamientoY.Value = (int)(altoDibujo * 10) -
```

```
            datos.placas[comboBoxPlacas.SelectedIndex].anchoY;
        }
    }

    private void buttonColorOverlay_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)
        {
            buttonColorOverlay.BackColor = colorDialog1.Color;
            datos.placas[comboBoxPlacas.SelectedIndex].overlay.color = colorDialog1.Color;
            rellenarCampos();
            datos.modificados = true;
        }
    }

    private void buttonColorSolder_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)
        {
            buttonColorSolder.BackColor = colorDialog1.Color;
            datos.placas[comboBoxPlacas.SelectedIndex].solder.color = colorDialog1.Color;
            rellenarCampos();
            datos.modificados = true;
        }
    }

    private void buttonColorLayer_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)
        {
            buttonColorLayer.BackColor = colorDialog1.Color;
            datos.placas[comboBoxPlacas.SelectedIndex].layer.color = colorDialog1.Color;
            rellenarCampos();
            datos.modificados = true;
        }
    }

    private void buttonQuitarOverlay_Click(object sender, EventArgs e)
    {
        datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero = null;
        labelOverlay.Text = "Ninguno";
        buttonQuitarOverlay.Enabled = false;
        overlay = null;
        rellenarCampos();
        datos.modificados = true;
    }

    private void buttonQuitarSolder_Click(object sender, EventArgs e)
    {
        datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero = null;
        labelSolder.Text = "Ninguno";
        buttonQuitarSolder.Enabled = false;
        solder = null;
        rellenarCampos();
        datos.modificados = true;
    }

    private void buttonQuitarLayer_Click(object sender, EventArgs e)
    {
        datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero = null;
        labelLayer.Text = "Ninguno";
        buttonQuitarLayer.Enabled = false;
        layer = null;
        rellenarCampos();
        datos.modificados = true;
    }

    private void buscarFicherosSimilares(string rutafichero)
    {
        // Listas de extensiones registradas.
        ArrayList extensionesOverlayTop = new ArrayList();
        ArrayList extensionesSolderTop = new ArrayList();
        ArrayList extensionesLayerTop = new ArrayList();
        ArrayList extensionesOverlayBottom = new ArrayList();
        ArrayList extensionesSolderBottom = new ArrayList();
        ArrayList extensionesLayerBottom = new ArrayList();
        extensionesOverlayTop.Add(".GTO");
        extensionesSolderTop.Add(".GTS");
        extensionesLayerTop.Add(".GTL");
```

```
extensionesOverlayBottom.Add(".GBO");
extensionesSolderBottom.Add(".GBS");
extensionesLayerBottom.Add(".GBL");

ArrayList extensionesTop = new ArrayList();
extensionesTop.Add(extensionesOverlayTop);
extensionesTop.Add(extensionesSolderTop);
extensionesTop.Add(extensionesLayerTop);
ArrayList extensionesBottom = new ArrayList();
extensionesBottom.Add(extensionesOverlayBottom);
extensionesBottom.Add(extensionesSolderBottom);
extensionesBottom.Add(extensionesLayerBottom);

string directorioDeTrabajo = Path.GetDirectoryName(rutafichero);
string nombreFichero = Path.GetFileNameWithoutExtension(rutafichero);
string extensiónFichero = Path.GetExtension(rutafichero);

bool encontradoTop = false;
bool encontradoBottom = false;
foreach (ArrayList grupoExt in extensionesTop)
{
    if (grupoExt.Contains(extensiónFichero))
    {
        encontradoTop = true;
        extensionesTop.Remove(grupoExt);
        break;
    }
}
if (!encontradoTop)
{
    foreach (ArrayList grupoExt in extensionesBottom)
    {
        if (grupoExt.Contains(extensiónFichero))
        {
            encontradoBottom = true;
            extensionesBottom.Remove(grupoExt);
            break;
        }
    }
}
if (encontradoTop)
{
    foreach (ArrayList grupoExt in extensionesTop)
    {
        foreach (string ext in grupoExt)
        {
            string fichero = Path.Combine(directorioDeTrabajo, nombreFichero + ext);
            if (File.Exists(fichero))
            {
                if (extensionesOverlayTop.Contains(ext))
                {
                    datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero = fichero;
                    overlay = null;
                }
                else if (extensionesSolderTop.Contains(ext))
                {
                    datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero = fichero;
                    solder = null;
                }
                else if (extensionesLayerTop.Contains(ext))
                {
                    datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero = fichero;
                    layer = null;
                }
            }
        }
    }
}
else if (encontradoBottom)
{
    foreach (ArrayList grupoExt in extensionesBottom)
    {
        foreach (string ext in grupoExt)
        {
            string fichero = Path.Combine(directorioDeTrabajo, nombreFichero + ext);
            if (File.Exists(fichero))
            {
                if (extensionesOverlayBottom.Contains(ext))
                {
```

```
                        datos.placas[comboBoxPlacas.SelectedIndex].overlay.fichero = fichero;
                        overlay = null;
                    }
                    else if (extensionesSolderBottom.Contains(ext))
                    {
                        datos.placas[comboBoxPlacas.SelectedIndex].solder.fichero = fichero;
                        solder = null;
                    }
                    else if (extensionesLayerBottom.Contains(ext))
                    {
                        datos.placas[comboBoxPlacas.SelectedIndex].layer.fichero = fichero;
                        layer = null;
                    }
                }
            }
        }
    }
}

private void FormAsignarGerber_Resize(object sender, EventArgs e)
{
    tamañoPictureBox.Width = this.Width - separaciónPictureBox.Width;
    tamañoPictureBox.Height = this.Height - separaciónPictureBox.Height;
    groupBoxGerber.Width = tamañoPictureBox.Width;
    buttonQuitarOverlay.Left = groupBoxGerber.Width - separaciónButtonX;
    buttonQuitarSolder.Left = groupBoxGerber.Width - separaciónButtonX;
    buttonQuitarLayer.Left = groupBoxGerber.Width - separaciónButtonX;
    buttonColorOverlay.Left = groupBoxGerber.Width - separaciónButtonColor;
    buttonColorSolder.Left = groupBoxGerber.Width - separaciónButtonColor;
    buttonColorLayer.Left = groupBoxGerber.Width - separaciónButtonColor;
    labelOverlay.Width = groupBoxGerber.Width - anchoLabel;
    labelSolder.Width = groupBoxGerber.Width - anchoLabel;
    labelLayer.Width = groupBoxGerber.Width - anchoLabel;
    rellenarCampos();
}

private void comboBoxPlacas_SelectedIndexChanged(object sender, EventArgs e)
{
    overlay = null;
    solder = null;
    layer = null;
    rellenarCampos();
}

private void numericUpDownDesplazamientoX_ValueChanged(object sender, EventArgs e)
{
    datos.placas[comboBoxPlacas.SelectedIndex].desplazamientoGerber.X =
         (int)numericUpDownDesplazamientoX.Value;
    datos.modificados = true;
}

private void numericUpDownDesplazamientoY_ValueChanged(object sender, EventArgs e)
{
    datos.placas[comboBoxPlacas.SelectedIndex].desplazamientoGerber.Y =
         (int)numericUpDownDesplazamientoY.Value;
    datos.modificados = true;
}

private void buttonCalcularDesplazmiento_Click(object sender, EventArgs e)
{
    numericUpDownDesplazamientoY.Value = (int)(altoDibujo * 10) -
         datos.placas[comboBoxPlacas.SelectedIndex].anchoY;
}

public void seleccionarPlaca(int n)
{
    if (n < datos.placas.Length)
    {
        rellenarCombo();
        comboBoxPlacas.SelectedIndex = n;
    }
}

private void checkBoxInvertir_CheckedChanged(object sender, EventArgs e)
{
    datos.placas[comboBoxPlacas.SelectedIndex].invertir = checkBoxInvertir.Checked;
    for (int n = 0; n< datos.placas[comboBoxPlacas.SelectedIndex].componentes.Length;n++)
    {
        datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].x =
```

```
                    datos.placas[comboBoxPlacas.SelectedIndex].anchoX -
                    datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].x;
                datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].t = 200 -
                    datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].t;
                if (datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].t < 0)
                {
                    datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].t += 400;
                }
            }
            rellenarCampos();
        }
    }
}
```

```
// FormMontajeManual.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace PnPWork
{
    public partial class FormMontajeManual : Form
    {
        #region Constantes y campos

        const int transparenciaRelleno = 80;
        const int transparenciaRellenoSeleccionado = 230;
        int margenX = 95;
        const int margenY = 130;
        const int x0 = 200;              // Coordenadas del origen del área de dibujo de
        const int y0 = 25;               // la placa en la pantalla.
        const int distanciaCaptura = 6;

        int xp0 = 0;                     // Coordenadas de la placa que están en el origen
        int yp0 = 0;                     // de la zona visible en pantalla.
        int anchoVisibleX;
        int anchoVisibleY;
        int componente = -1;
        public int numplaca;
        bool mostrandoToolTip = false;
        Point posiciónAnteriorRatón;
        SizeF tamañoTexto = new SizeF();
        Color colorFondo = Color.Black;
        Font fuenteComponentes = new Font("Arial", 6);
        Font fuenteEscalas = new Font("Arial", 7);
        Pen plumaLíneasEscalas = new Pen(Color.White);  // Pluma para dibujar las líneas.
        Brush pincelRótulosEscalas = Brushes.Wheat;     // Pincel para escribir los rótulos.
        Gerber overlay = new Gerber();
        Gerber solder = new Gerber();
        Gerber layer = new Gerber();
        Bitmap imagenGerber;
        Point desplazamientoGerber = new Point(0, 0);
        Bitmap imagenMiniatura;
        float escalaMiniatura;
        placa p;
        int componenteAnterior = -1;
        estadoComponente estadoAnterior;

        #endregion

        public FormMontajeManual()
        {
            InitializeComponent();
            this.MouseWheel += new MouseEventHandler(FormMontajeManual_MouseWheel);
            margenX += panelIzquierdo.Width + 10;
        }

        public bool representarPlaca(int n)
        {
            if (n >= datos.placas.Length) return false;
```

```
        p = datos.placas[n];
        numplaca = n;

        // Establece posición y dimensiones del form
        if ((this.MdiParent != null) && (datos.formPadre.ClientSize.Width < (p.anchoX + margenX)))
        {
            this.Left = 0;
            this.Width = datos.formPadre.ClientSize.Width - 10;
        }
        else
        {
            this.Width = p.anchoX + margenX;
        }

        if ((this.MdiParent != null) && (datos.formPadre.ClientSize.Height < (p.anchoY + margenY)))
        {
            this.Top = 0;
            this.Height = datos.formPadre.ClientSize.Height - 32;
        }
        else
        {
            this.Height = p.anchoY + margenY;
        }

        // Rellena la lista de componentes
        listBoxComponentes.Items.Clear();
        for (int i = 0; i < p.componentes.Length; i++)
        {
            listBoxComponentes.Items.Add(p.componentes[i].nombre + " - " +
                      datos.alimentadores[p.componentes[i].alimentador].tipoComponente);
        }

        // Dibuja la placa en pantalla
        generarImagenesGerber();
        if ((imagenGerber == null) && (MessageBox.Show("La placa seleccionada no tiene asociados
              archivos Gerber.\r\n¿Desea asociarlos ahora?", "No hay imagen Gerber",
             MessageBoxButtons.OKCancel, MessageBoxIcon.Warning) ==
             System.Windows.Forms.DialogResult.OK))
        {
            datos.formPadre.asignarGerberANuevaPlaca(numplaca);
        }
        dibujarPlaca();
        listBoxComponentes.SelectedIndex = 0;
        componenteAnterior = 0;
        return true;
    }

    public void dibujarPlaca()
    {
        if (this.IsDisposed) return;

        try
        {
            // Pone el texto de la barra de título.
            this.Text = "Montaje manual - Placa " + (numplaca + 1).ToString() + " - " +
             datos.placas[numplaca].nombre + " - " + Convert.ToString(p.anchoX / 10) + "mm x " +
             Convert.ToString(p.anchoY / 10) + "mm.";
            if (!datos.placaOK(numplaca)) this.Text += "  ATENCIÓN: Placa con errores";

            // Ajusta las barras de desplazamiento.
            if ((xp0 + anchoVisibleX) > p.anchoX) xp0 = p.anchoX - anchoVisibleX;
            if ((yp0 + anchoVisibleY) > p.anchoY) yp0 = p.anchoY - anchoVisibleY;
            ajusteScrollBars();

            // Crea un lienzo buffer para dibujar antes de enviar a la ventana.
            Bitmap imagen = new Bitmap(this.Width, this.Height);
            Graphics lienzo = Graphics.FromImage(imagen);
            lienzo.Clear(colorFondo);
            if (imagenGerber != null)
            {
                lienzo.DrawImage(imagenGerber, x0 - xp0 -
                      datos.placas[numplaca].desplazamientoGerber.X, y0 - yp0 -
                      datos.placas[numplaca].desplazamientoGerber.Y);
                dibujarMiniatura();
            }

            // Dibuja los componentes.
            for (int i = 0; i < p.componentes.Length; i++)
            {
```

```
                if (i == listBoxComponentes.SelectedIndex) dibujarComponente(i, lienzo);
                else if (checkBoxVerComponentes.Checked)
                {
                    dibujarComponente(i, lienzo);
                }
            }

            // Borra los bordes para que no se vean los componentes que sobresalen.
            lienzo.FillRectangle(Brushes.Black, 0, 0, this.Width, y0);
            lienzo.FillRectangle(Brushes.Black, 0, y0, x0, anchoVisibleY);
            lienzo.FillRectangle(Brushes.Black, 0, y0 + anchoVisibleY, this.Width, this.Height - y0 –
                      anchoVisibleY);
            lienzo.FillRectangle(Brushes.Black, x0 + anchoVisibleX, y0, this.Width - x0 –
                      anchoVisibleX, anchoVisibleY);

            // Dibuja las 4 escalas y el marco.
            dibujarEscala(lienzo, x0, y0, anchoVisibleX, xp0, 'u');
            dibujarEscala(lienzo, x0, y0, anchoVisibleY, yp0, 'l');
            dibujarEscala(lienzo, x0, y0 + anchoVisibleY, anchoVisibleX, xp0, 'd');
            dibujarEscala(lienzo, x0 + anchoVisibleX, y0, anchoVisibleY, yp0, 'r');
            lienzo.DrawRectangle(Pens.White, x0, y0, anchoVisibleX, anchoVisibleY);
            tamañoTexto = lienzo.MeasureString("mm", fuenteEscalas);
            lienzo.DrawString("mm", fuenteEscalas, pincelRótulosEscalas, x0 - tamañoTexto.Width - 5,
                      y0 - tamañoTexto.Height - 5);

            // Copia el dibujo hecho del buffer a la ventana.
            Graphics g = this.CreateGraphics();
            g.DrawImage(imagen, 0, 0);
            lienzo.Dispose();
            imagen.Dispose();
            g.Dispose();
        }
        catch
        {
            this.Close();
            MessageBox.Show("No se ha podido generar la imagen. Asegúrese de que se han asignado
             archivos gerber a la placa.",
             "Error en la generación de imagen", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }

    private void dibujarComponente(int i, Graphics lienzo)
    {
        Color colorComponente = Color.Khaki;
        Brush pincelTexto = Brushes.Wheat;

        if (checkBoxMostrarEstado.Checked)
        {
            switch (p.componentes[i].estado)
            {
                case estadoComponente.Cogiendo:
                    colorComponente = Color.Blue;
                    pincelTexto = Brushes.LightBlue;
                    break;
                case estadoComponente.Colocando:
                    colorComponente = Color.DarkViolet;
                    pincelTexto = Brushes.LightBlue;
                    break;
                case estadoComponente.Colocado:
                    colorComponente = Color.Green;
                    pincelTexto = Brushes.LightGreen;
                    break;
                case estadoComponente.Error_Cogiendo:
                    colorComponente = Color.OrangeRed;
                    pincelTexto = Brushes.Orange;
                    break;
                case estadoComponente.Error_Colocando:
                    colorComponente = Color.Red;
                    pincelTexto = Brushes.Orange;
                    break;
            }
        }

        lienzo.DrawPolygon(new Pen(colorComponente), p.componentes[i].vértices(x0 - xp0, y0 - yp0));
        Color colorRelleno;
        if (i == listBoxComponentes.SelectedIndex)
        {
            colorRelleno = Color.FromArgb(transparenciaRellenoSeleccionado, colorComponente);
            if (!checkBoxMostrarEstado.Checked) pincelTexto = Brushes.Chocolate;
```

```
        }
        else colorRelleno = Color.FromArgb(transparenciaRelleno, colorComponente);
        lienzo.FillPolygon(new SolidBrush(colorRelleno), p.componentes[i].vértices(x0 - xp0, y0 –
             yp0));
        tamañoTexto = lienzo.MeasureString(p.componentes[i].nombre, fuenteComponentes);
        lienzo.DrawString(p.componentes[i].nombre, fuenteComponentes, pincelTexto,
                          new Point((int)(p.componentes[i].x + x0 - xp0 - tamañoTexto.Width / 2.5),
                          (int)(p.componentes[i].y + y0 - yp0 - tamañoTexto.Height / 3)));
    }

    private void dibujarEscala(Graphics lienzo, int xr0, int yr0, int longitud, int valorInicial, char
              posición)
    {
        // Dibuja las líneas de graduación de las escalas y las rotula.
        //
        // La posición de las líneas y el texto será:
        //  u = encima
        //  d = debajo
        //  l = izquierda
        //  r = derecha

        const int divisionMenor = 10;      // Separación entre divisiones menores de la escala.
        const int divisionMayor = 50;      // Separación entre divisiones numeradas de la escala.
        int x, y;
        float xRótulo, yRótulo;
        string rótulo;

        int desplazamientoInicial;
        if (valorInicial == 0) desplazamientoInicial = 0;
        else desplazamientoInicial = divisionMenor - valorInicial % divisionMenor;
        int incX = (posición == 'u' || posición == 'd') ? 1 : 0;
        int incY = (posición == 'l' || posición == 'r') ? 1 : 0;
        int arrAba = posición == 'u' ? -1 : posición == 'd' ? 1 : 0;
        int izqDer = posición == 'l' ? -1 : posición == 'r' ? 1 : 0;

        for (int n = desplazamientoInicial; n <= longitud; n += divisionMenor)
        {
            // Calcula las coordenadas de la siguiente línea de la escala
            x = xr0 + n * incX;
            y = yr0 + n * incY;

            if (((valorInicial + n) % divisionMayor) != 0)
            {
                // Dibuja la línea de las divisiones menores.
                lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 2, y + arrAba * 2);
            }
            else
            {
                lienzo.DrawLine(plumaLíneasEscalas, x, y, x + izqDer * 5, y + arrAba * 5);
                rótulo = Convert.ToString((xp0 * incX + yp0 * incY + n) / 10);
                tamañoTexto = lienzo.MeasureString(rótulo, fuenteEscalas);
                xRótulo = x - incX * tamañoTexto.Width / 2 + incY * izqDer * 6 - (posición == 'l' ?
                      tamañoTexto.Width : 0);
                yRótulo = y - incY * tamañoTexto.Height / 3 + incX * arrAba * 6 - (posición == 'u' ?
                      tamañoTexto.Height : 0);
                lienzo.DrawString(rótulo, fuenteEscalas, pincelRótulosEscalas, xRótulo, yRótulo);
            }
        }
    }

    private void ajusteScrollBars()
    {
        if (this.Width < (p.anchoX + margenX))
        {
            hScrollBar1.Visible = true;
            hScrollBar1.Maximum = p.anchoX;
            hScrollBar1.LargeChange = Math.Max(0, anchoVisibleX);
            hScrollBar1.Value = xp0;
        }
        else
        {
            hScrollBar1.Visible = false;
            hScrollBar1.Value = 0;
        }

        if (this.Height < (p.anchoY + margenY))
        {
            vScrollBar1.Visible = true;
            vScrollBar1.Maximum = p.anchoY;
```

```
            vScrollBar1.LargeChange = Math.Max(0, anchoVisibleY);
            vScrollBar1.Value = yp0;
        }
        else
        {
            vScrollBar1.Visible = false;
            vScrollBar1.Value = 0;
        }
    }

    private void generarImagenesGerber()
    {
        Gerber overlay = new Gerber();
        Gerber solder = new Gerber();
        Gerber layer = new Gerber();
        Gerber.escala = 10F;
        bool hayLayer = layer.interpretarFichero(p.layer.fichero);
        bool haySolder = solder.interpretarFichero(p.solder.fichero);
        bool hayOverlay = overlay.interpretarFichero(p.overlay.fichero);

        if (hayLayer || haySolder || hayOverlay)
        {
            int anchoGerber = (int)(Math.Max(solder.xmax, Math.Max(layer.xmax, overlay.xmax)) *
                      Gerber.escala);
            int altoGerber = (int)(Math.Max(solder.ymax, Math.Max(layer.ymax, overlay.ymax)) *
                     Gerber.escala);
            imagenGerber = new Bitmap(anchoGerber, altoGerber);
            Graphics lienzo = Graphics.FromImage(imagenGerber);
            lienzo.Clear(Color.FromArgb(0, 50, 0));
            layer.colorDeDibujo = p.layer.color;
            solder.colorDeDibujo = p.solder.color;
            overlay.colorDeDibujo = p.overlay.color;
            layer.dibujarImagen(ref imagenGerber);
            solder.dibujarImagen(ref imagenGerber);
            overlay.dibujarImagen(ref imagenGerber);
            desplazamientoGerber.X = datos.placas[numplaca].desplazamientoGerber.X; // Copia local
            desplazamientoGerber.Y = datos.placas[numplaca].desplazamientoGerber.Y; //
            if (desplazamientoGerber.Y == 0)
            {
                datos.placas[numplaca].desplazamientoGerber.Y = imagenGerber.Height - p.anchoY;
                datos.modificados = true;
            }

            escalaMiniatura = Math.Min((float)(pictureBoxGerber.ClientSize.Width) /
             imagenGerber.Width, (float)(pictureBoxGerber.ClientSize.Height) / imagenGerber.Height);
            imagenMiniatura = escalarImagen(imagenGerber, (int)(imagenGerber.Width * escalaMiniatura),
             (int)(imagenGerber.Height * escalaMiniatura));

            if (p.invertir)
            {
                imagenGerber.RotateFlip(RotateFlipType.RotateNoneFlipX);
                imagenMiniatura.RotateFlip(RotateFlipType.RotateNoneFlipX);
            }
        }
        else
        {
            imagenGerber = null;
            imagenMiniatura = null;
        }
    }

    public static Bitmap escalarImagen(Image imagen, int ancho, int alto)
    {
        var rectánguloImagen = new Rectangle(0, 0, ancho, alto);
        var imagenRedimensionada = new Bitmap(ancho, alto);

        imagenRedimensionada.SetResolution(imagen.HorizontalResolution, imagen.VerticalResolution);

        using (var graphics = Graphics.FromImage(imagenRedimensionada))
        {
            graphics.CompositingMode = System.Drawing.Drawing2D.CompositingMode.SourceCopy;
            graphics.CompositingQuality = System.Drawing.Drawing2D.CompositingQuality.HighQuality;
            graphics.InterpolationMode =
                      System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
            graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
            graphics.PixelOffsetMode = System.Drawing.Drawing2D.PixelOffsetMode.HighQuality;

            using (var wrapMode = new System.Drawing.Imaging.ImageAttributes())
            {
```

```
                wrapMode.SetWrapMode(System.Drawing.Drawing2D.WrapMode.TileFlipXY);
                graphics.DrawImage(imagen, rectánguloImagen, 0, 0, imagen.Width, imagen.Height,
                      GraphicsUnit.Pixel, wrapMode);
            }
        }
        return imagenRedimensionada;
    }

    private void dibujarMiniatura()
    {
        Bitmap imagen = new Bitmap(pictureBoxGerber.ClientSize.Width,
             pictureBoxGerber.ClientSize.Height);
        Graphics lienzo = Graphics.FromImage(imagen);
        lienzo.DrawImage(imagenMiniatura, 0, 0);

        lienzo.DrawRectangle(new Pen(Color.Red, 2),
                             (xp0 + desplazamientoGerber.X) * escalaMiniatura,
                             (yp0 + desplazamientoGerber.Y) * escalaMiniatura,
                             anchoVisibleX * escalaMiniatura,
                             anchoVisibleY * escalaMiniatura);
        pictureBoxGerber.Image = imagen;
    }

    private void centrarComponente(int n)
    {
        if (anchoVisibleX < p.anchoX)
        {
            xp0 = p.componentes[n].x - anchoVisibleX / 2;
            if (xp0 < 0) xp0 = 0;
        }
        if (anchoVisibleY < p.anchoY)
        {
            yp0 = p.componentes[n].y - anchoVisibleY / 2;
            if (yp0 < 0) yp0 = 0;
        }
        dibujarPlaca();
    }


    private void FormMontajeManual_Paint(object sender, PaintEventArgs e)
    {
        dibujarPlaca();
    }

    protected override void OnPaintBackground(PaintEventArgs e)
    {
        return;
    }

    private void FormMontajeManual_Resize(object sender, EventArgs e)
    {
        if (this.Width < (p.anchoX + margenX))
        {
            anchoVisibleX = this.Width - margenX;
        }
        else
        {
            anchoVisibleX = p.anchoX;
            this.Width = p.anchoX + margenX;
        }

        if (this.Height < (p.anchoY + margenY))
        {
            anchoVisibleY = this.Height - margenY;
        }
        else
        {
            anchoVisibleY = p.anchoY;
            this.Height = p.anchoY + margenY;
        }

        panelIzquierdo.Height = vScrollBar1.Height;

        dibujarPlaca();
    }

    private void FormMontajeManual_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
    {
        if ((e.X >= x0) && (e.X < x0 + p.anchoX) && (e.Y >= y0) && (e.Y <= y0 + p.anchoY))
```

```
{
    toolStripStatusLabel1.Text = "(" + Convert.ToString(Math.Round((float)(e.X - x0) / 10)) +
              "," + Convert.ToString((float)(e.X - x0) % 10)
             + " : " + Convert.ToString(Math.Round((float)(e.Y - y0) / 10)) + "," +
             Convert.ToString((e.Y - y0) % 10) + ")";
    componente = -1;
    for (int i = 0; i < p.componentes.Length; i++)
    {
        if (((Math.Abs(e.X - (p.componentes[i].x + x0 - xp0))) < distanciaCaptura) &&
              (Math.Abs(e.Y - (p.componentes[i].y + y0 - yp0))) < distanciaCaptura)
        {
            componente = i;
        }
    }

    // Actualiza el ToolTip
    if (componente == -1)
    {
        this.Cursor = System.Windows.Forms.Cursors.Default;
        toolTipComponente.Hide(this);
        mostrandoToolTip = false;
    }
    else
    {
        if (!mostrandoToolTip)
        {
            this.Cursor = System.Windows.Forms.Cursors.Hand;
            string texto = "Tipo: " +
              datos.alimentadores[p.componentes[componente].alimentador].tipoComponente + "\r"
                + "x: " + Math.Round((float)(p.componentes[componente].x / 10)).ToString()
                + "," + ((float)(p.componentes[componente].x % 10)).ToString()
                + "   y: " + Math.Round((float)p.componentes[componente].y / 10).ToString()
                + "," + (p.componentes[componente].y % 10).ToString()
                + "   Giro: " + p.componentes[componente].t.ToString()
                + "\rEstado: ";
            switch (p.componentes[componente].estado)
            {
                case estadoComponente.Cogiendo:
                    texto += "Cogiendo";
                    break;
                case estadoComponente.Colocando:
                    texto += "Colocando";
                    break;
                case estadoComponente.Colocado:
                    texto += "Colocado";
                    break;
                case estadoComponente.Error_Cogiendo:
                    texto += "Error al coger";
                    break;
                case estadoComponente.Error_Colocando:
                    texto += "Error al colocar";
                    break;
                default:
                    texto += "No tratado";
                    break;
            }
            mostrandoToolTip = true;
            toolTipComponente.ToolTipTitle = "Componente: " +
              p.componentes[componente].nombre;

            // Apunta el tooltip hacia un label oculto en la posición del componente
            labelObjetivo.Left = p.componentes[componente].x + x0 - xp0;
            labelObjetivo.Top = p.componentes[componente].y + y0 - yp0;

            toolTipComponente.Show(string.Empty, labelObjetivo, 0);
            toolTipComponente.Show(texto, labelObjetivo);
        }
    }
}

// Hace scroll si la ventana lo permite
if (e.Button == System.Windows.Forms.MouseButtons.Left)
{
    if (anchoVisibleX < p.anchoX)
    {
        xp0 -= e.X - posiciónAnteriorRatón.X;
        if (xp0 < 0) xp0 = 0;
    }
    if (anchoVisibleY < p.anchoY)
```

```
            {
                yp0 -= e.Y - posiciónAnteriorRatón.Y;
                if (yp0 < 0) yp0 = 0;
            }
            this.Cursor = System.Windows.Forms.Cursors.SizeAll;
            dibujarPlaca();
        }

        if (e.Button == System.Windows.Forms.MouseButtons.Middle)
        {
            desplazamientoGerber.X -= e.X - posiciónAnteriorRatón.X;
            desplazamientoGerber.Y -= e.Y - posiciónAnteriorRatón.Y;
            datos.modificados = true;
            dibujarPlaca();
        }

        posiciónAnteriorRatón = e.Location;
    }

    private void FormPlaca_MouseDown(object sender, MouseEventArgs e)
    {
        // Si se hace click izquierdo, se selecciona el componente para centrarlo
        if ((e.Button == System.Windows.Forms.MouseButtons.Left) && (componente != -1))
        {
            listBoxComponentes.SelectedIndex = componente;
        }

        // Si se hace clic derecho se cambia de estado
        if ((e.Button == System.Windows.Forms.MouseButtons.Right) && (componente != -1))
        {
            p.componentes[componente].estado = p.componentes[componente].estado ==
             estadoComponente.No_Tratado ? estadoComponente.Colocado : estadoComponente.No_Tratado;
            dibujarPlaca();
        }
    }

    private void FormMontajeManual_MouseWheel(object sender, MouseEventArgs e)
    {
        if (vScrollBar1.Visible)
        {
            yp0 -= 10 * Math.Sign(e.Delta);
            if (yp0 < 0) yp0 = 0;
            vScrollBar1.Value = yp0;
            dibujarPlaca();
        }
        else
        {
            if (hScrollBar1.Visible)
            {
                xp0 -= 10 * Math.Sign(e.Delta);
                if (xp0 < 0) xp0 = 0;
                hScrollBar1.Value = xp0;
                dibujarPlaca();
            }
        }
    }

    private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
    {
        if (e.NewValue != e.OldValue)
        {
            xp0 = e.NewValue;
            dibujarPlaca();
        }
    }

    private void vScrollBar1_Scroll(object sender, ScrollEventArgs e)
    {
        if (e.NewValue != e.OldValue)
        {
            yp0 = e.NewValue;
            dibujarPlaca();
        }
    }

    private void listBoxComponentes_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (checkBoxActualizarEstado.Checked)
        {
```

```
            estadoAnterior = p.componentes[listBoxComponentes.SelectedIndex].estado;
            p.componentes[listBoxComponentes.SelectedIndex].estado = estadoComponente.Colocando;
            if (componenteAnterior >= 0) p.componentes[componenteAnterior].estado =
                    estadoComponente.Colocado;
        }
        componenteAnterior = listBoxComponentes.SelectedIndex;
        centrarComponente(listBoxComponentes.SelectedIndex);
    }

    private void listBoxComponentes_KeyPress(object sender, KeyPressEventArgs e)
    {
        toolTipComponente.Hide(this);
        mostrandoToolTip = false;
        if (e.KeyChar == Convert.ToChar(8))
        {
            if (listBoxComponentes.SelectedIndex > 0) listBoxComponentes.SelectedIndex--;
        }
        else
        {
            if (listBoxComponentes.SelectedIndex < p.componentes.Length - 1)
                    listBoxComponentes.SelectedIndex++;
        }
    }

    private void buttonSiguiente_Click(object sender, EventArgs e)
    {
        if (listBoxComponentes.SelectedIndex < p.componentes.Length - 1)
             listBoxComponentes.SelectedIndex++;
        listBoxComponentes.Focus();
    }

    private void buttonAnterior_Click(object sender, EventArgs e)
    {
        if (listBoxComponentes.SelectedIndex > 0) listBoxComponentes.SelectedIndex--;
        listBoxComponentes.Focus();
    }

    private void checkBoxVerComponentes_CheckedChanged(object sender, EventArgs e)
    {
        dibujarPlaca();
        listBoxComponentes.Focus();
    }

    private void checkBoxActualizarEstado_CheckedChanged(object sender, EventArgs e)
    {
        if (!checkBoxActualizarEstado.Checked) p.componentes[listBoxComponentes.SelectedIndex].estado
             = estadoAnterior;
        else p.componentes[listBoxComponentes.SelectedIndex].estado = estadoComponente.Colocando;
        dibujarPlaca();
        listBoxComponentes.Focus();
    }

    private void checkBoxMostrarEstado_CheckedChanged(object sender, EventArgs e)
    {
        dibujarPlaca();
        listBoxComponentes.Focus();
    }

    private void pictureBoxGerber_Click(object sender, EventArgs e)
    {
        if (!pictureBoxZoom.Visible)
        {
            float ffImagen = (float)imagenGerber.Width / (float)imagenGerber.Height;
            float anchoVentana = this.Width - panelIzquierdo.Width;
            float altoVentana = this.Height - hScrollBar1.Height - statusStrip1.Height;
            float ffventana = anchoVentana / altoVentana;
            float escala;
            if (ffImagen > ffventana) escala = anchoVentana / (float)imagenGerber.Width;
            else escala = altoVentana / (float)imagenGerber.Height;

            pictureBoxZoom.Location = new Point(panelIzquierdo.Width, 0);
            pictureBoxZoom.Size = new Size((int)anchoVentana, (int)altoVentana);
            pictureBoxZoom.SizeMode = PictureBoxSizeMode.CenterImage;
            pictureBoxZoom.BringToFront();

            Bitmap imagen = escalarImagen(imagenGerber, (int)(imagenGerber.Width * escala),
                    (int)(imagenGerber.Height * escala));
            Graphics lienzo = Graphics.FromImage(imagen);
            lienzo.DrawRectangle(new Pen(Color.Red, 4),
```

```
                    (xp0 + desplazamientoGerber.X) * escala,
                    (yp0 + desplazamientoGerber.Y) * escala,
                    anchoVisibleX * escala,
                    anchoVisibleY * escala);

                pictureBoxZoom.Image = imagen;
                pictureBoxZoom.Visible = true;
            }
            else pictureBoxZoom.Visible = false;
        }

        private void pictureBoxZoom_Click(object sender, EventArgs e)
        {
            pictureBoxZoom.Visible = false;
        }
    }
}

//FormMontajeVIOS.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Globalization;
using System.Collections;

namespace PnPWork
{
    public partial class FormMontajeVIOS : Form
    {
        public FormMontajeVIOS()
        {
            InitializeComponent();

            // Rellena el comboBox con las placas existentes y selecciona la primera
            comboBoxPlacas.Items.Clear();
            for (int p = 0; p < datos.placas.Length; p++)
            {
                comboBoxPlacas.Items.Add((p + 1).ToString() + " -> " + datos.placas[p].nombre);
            }
            if (comboBoxPlacas.Items.Count > 0)
            {
                comboBoxPlacas.SelectedIndex = 0;
            }
            else
            {
                this.Close();   // Cierra la ventana si no hay placas.
            }
        }

        private void buttonGenerar_Click(object sender, EventArgs e)
        {
            ArrayList alimentadores = new ArrayList();
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "Archivos VIOS (*.txt)|*.txt|Todos los archivos (*.*)|*.*";
            if (saveFileDialog1.ShowDialog(this) != DialogResult.OK) return;
            FileInfo fi = new FileInfo(saveFileDialog1.FileName);
            StreamWriter sw = fi.CreateText();
            if (sw != null)
            {
                sw.WriteLine(("PCBNAME=" + (datos.placas[comboBoxPlacas.SelectedIndex].nombre.Length <= 20
                          ? datos.placas[comboBoxPlacas.SelectedIndex].nombre :
                          datos.placas[comboBoxPlacas.SelectedIndex].nombre.Substring(0, 20))));
                sw.WriteLine("&I.TYPE=");
                sw.WriteLine("&I.COMMENT=");
                sw.WriteLine("&I.EXEC=00000001000000000000000000002C");
                sw.WriteLine("&I.TRNS=0100000000000000000050F03000000");
                sw.WriteLine("&I.COND=00000000000000000010101010100");
                sw.WriteLine("&I.NZL =0000000000000000000000000000000000000000000000000000000000000000000000");
                sw.WriteLine("End_of_ID");
                sw.WriteLine("&B.BAD");
                sw.WriteLine("&B.DSP");
                sw.WriteLine("&B.DSM");
                sw.WriteLine("&B.DSL");
```

```
                sw.WriteLine("&B.DSC");
                sw.WriteLine("&B.FID");
                sw.WriteLine("&B.MNT");
                for (int n = 0; n < datos.placas[comboBoxPlacas.SelectedIndex].componentes.Length; n++)
                {
                    // Genera el número de alimentador consecutivamente
                    if (!alimentadores.Contains(
                          datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].alimentador))
                          alimentadores.Add(datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].
                          alimentador);

                    sw.WriteLine((((float)datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].x /
                          10).ToString("F3", CultureInfo.InvariantCulture).PadLeft(8) + " " +
                          ((float)datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].y /
                          10).ToString("F3", CultureInfo.InvariantCulture).PadLeft(8) + "    0.000 " +
                          (datos.placas[comboBoxPlacas.SelectedIndex].componentes[n].t * 0.9).ToString(
                          "F3", CultureInfo.InvariantCulture).PadLeft(8) + " 0A0000FFFF0101000064FFFFFFFF"
                          + alimentadores.IndexOf(datos.placas[comboBoxPlacas.SelectedIndex.omponentes[n].
                          alimentador).ToString("X4")+ " " + datos.placas[comboBoxPlacas.SelectedIndex].
                          componentes[n]. nombre.ToString()));
                }

                sw.WriteLine("&B.OPT");
                sw.WriteLine("&B.ORG");
                sw.WriteLine("   0.000    0.000    0.000    0.000 0A0000FFFF0001000000FFFFFFFF0000");
                sw.WriteLine("&B.BRD");
                sw.WriteLine("   0.000    0.000    0.000    0.000 0A0000000000001000000FFFFFFFF0000");
                sw.WriteLine("&B.QED");
                sw.WriteLine("End_of_BD");
                for (int n = 0; n < alimentadores.Count; n++)
                {
                    sw.WriteLine(("&F"));
                    sw.WriteLine(datos.alimentadores[(int)alimentadores[n]].tipoComponente);
                    sw.WriteLine(datos.alimentadores[(int)alimentadores[n]].tipoComponente);
                    sw.WriteLine("0000000000030000070100000000000000001E1E\r\n
                                  00000000000002000000000100000F1E1E000000");
                    sw.WriteLine(("        1 " + n.ToString().PadLeft(8) + "       90        0"));
                    sw.WriteLine("    0.000     0.00     0.00     0.00");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine((((float)datos.alimentadores[(int)alimentadores[n]].anchoX /
                          10).ToString("F3", CultureInfo.InvariantCulture).PadLeft(9) +
                          ((float)datos.alimentadores[(int)alimentadores[n]].anchoY / 10).ToString("F3",
                          CultureInfo.InvariantCulture).PadLeft(9) + "     0.50     0.00"));
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("        0        0        0        0");
                    sw.WriteLine("     0.00     0.00     0.00     0.00");
                }
                sw.WriteLine("End_of_FD");
                sw.WriteLine("&M   0=0000000000C800001E00000000000000");
                sw.WriteLine("1");
                sw.WriteLine("REFLECT");
                sw.WriteLine("010100018D1E0000000000000000007010000");
                sw.WriteLine("      153        0        0        0");
                sw.WriteLine("    2.000    1.000    0.785    3.142");
                sw.WriteLine("    0.000    1.500    0.000    0.000");
                sw.WriteLine("        0        0        0      214");
                sw.WriteLine("        0        0        0        0");
                sw.WriteLine("");
                sw.WriteLine("End_of_MD");
                sw.Close();
            }
            this.Close();
        }
    }
}
```