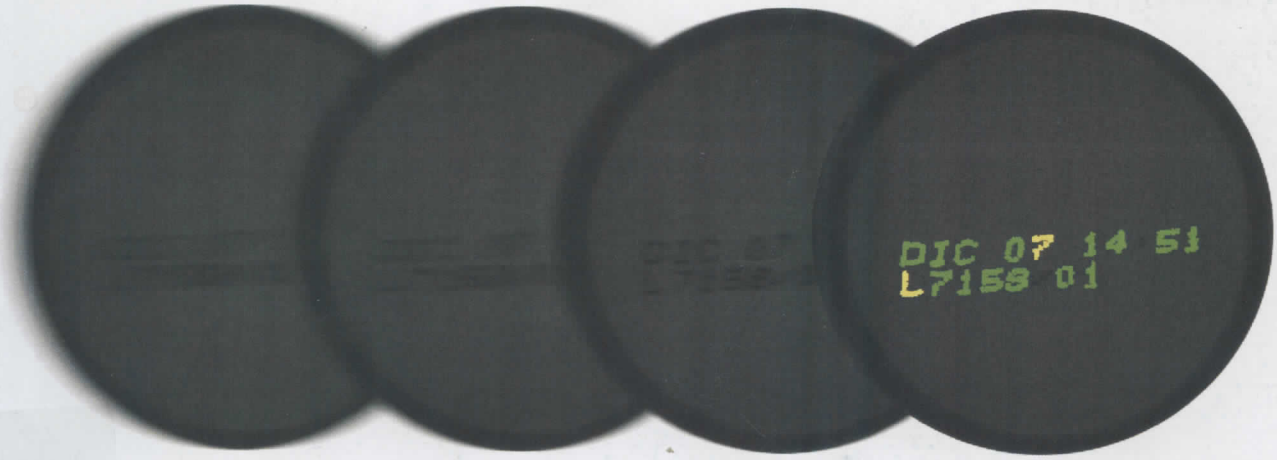
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Ciencias
y Tecnologías Cibernéticas

Tesis doctoral

MONICOD: Supervisión Visual a Muy Alta Velocidad de Codificación Impresa Industrial



Jose Carlos Rodríguez Rodríguez

Las Palmas de Gran Canaria, Noviembre 2012

MONICOD: Supervisión Visual a Muy Alta Velocidad de Codificación Impresa Industrial

"La perfección se alcanza, no cuando no hay nada más que añadir,
sino cuando ya no queda nada más que quitar."

Antoine de Saint-Exupéry

Agradecimientos

Pido perdón.

Un texto tan voluminoso como este, que goza de una historia tan larga como repleta de vicisitudes, ha quedado en deuda con mucha gente. Demasiada en realidad. Es verdad que no recuerdo todos los nombres y ponernos exhaustivos no evitará que termine siendo injusto. Pero pienso que eso es algo normal en apartados como este y solicitar perdón sólo porque se supone que debería hacerse no me parece correcto. No. Pediré perdón porque siento que en estos agradecimientos no me sinceraré por entero. Diré la verdad y seré objetivo, pero no lo diré todo ni sonaré tan auténtico como merecería la ocasión. Y por esto es por lo que suplico vuestro perdón. Mientras escribo estas líneas se me agolpan tantísimos recuerdos que me siento agotado, impotente, indefenso. Incluso incómodo. Estoy embargado y sé que si me dejo arrastrar no podré escribir algo adecuado. Por tanto me alejaré unos pasos, esperaré a que se me pase, dejaré morir ese estado alterado de conciencia, recuperaré el ánimo, apartaré esas cosas que me debilitan y regresaré para terminar lo que he empezado...

Podrá ser un tópico, pero no por eso es menos cierto: Sin **mi familia** este trabajo no podría existir de ninguna manera. Y ningún agradecimiento les hará justicia. Mi madre. Mi padre. Mi hermana. ME HAN SOPORTADO Y ME HAN DADO SOPORTE. Y difícilmente estas palabras pueden ser más ciertas. Me han dado un techo y me han alimentado y me han acompañado antes y durante la gestación de este trabajo. Pero sé que esa ha sido la parte fácil. Un recuerdo emocionado para mi padre que ya no está entre nosotros, y un beso cálido para mi madre que me ha llevado en volandas hasta aquí. Y un tirón de moño para mi hermana que es quien mejor me comprende. Que sepan que de ellos sólo me estoy quedando con los recuerdos buenos.

Roberto Moreno (jr) es la primera persona que debo mencionar. En un momento dado él me inspiró. Me inspiró, se ocupó de mi y por él llegué a esto de los doctorados. Fue a través de él como conocí a **Alexis Quesada** y tomé contacto con toda la familia del IUCTC. Él puso en marcha todo el engranaje que me ha llevado hasta este preciso instante. Lo único que lamento es no haber podido estar más tiempo con él, aunque no podré agradecerle lo suficiente haberme dejado en las manos adecuadas. ¡Gracias **Roberto**!

Existen numerosas notas de agradecimiento que mencionan a **Alexis Quesada** calurosamente. Y seguro que aún hay muchas que están por escribir. Sé

de primera mano que la mención está justificada. A estas alturas, muchos (probablemente tu mismo, lector) ya han concluido que el simple hecho de conocerle ha sido un enorme golpe de buena suerte personal y profesional. Sé que gente mucho más válida que yo le brindará el reconocimiento y tributo que se merece. Por mi parte, en lugar de escribir una nota elogiosa sobre él, me limitaré a citar los hechos: Horas y horas acumuladas. Ese es el tiempo que me ha dedicado. Puede decirse que hemos viajado juntos. Yo habré hecho la contribución más prominente a este trabajo pero la más decisiva procede de **Alexis** con su liderazgo. Infinitas veces no he sabido qué hacer y él ha estado ahí con su instrucción e ilimitada paciencia. Así, su mano está presente en todas partes. Agregado a este mérito, también es de justicia reconocer de forma explícita su papel como intermediario absoluto entre el resto del mundo y yo en el desarrollo de este trabajo. Y con el mundo me refiero a empresas, técnicos, organismos, etcétera. Yo soy un tipo desastroso pero **Alexis** ha hecho lo que ha estado en su mano para ocultar ese detalle.

Mientras trataba de sacar adelante este trabajo me he topado con gente que, con sus ideas, me ha influido. En ese sentido la contribución individual de **David González González** en su PFC ([1]) ha sido especialmente importante. Mientras yo me peleaba con el problema de una detección de lata eficiente, suya fue la idea de un agente separador (que él denominó de manera cariñosa "hormiga") para separar las líneas de texto. También introdujo el concepto de "solapamiento vertical" como un criterio para agrupar fragmentos. Llegado el momento lo intuitivo y simple de aquellas ideas me sedujo tanto que traté de formalizarlas, generalizarlas y extenderlas derivando en algunos de los métodos que aquí se describen.

Por fin, pero no menos importante, toda mi gratitud y reconocimiento a la **Compañía Embotelladora de Canarias** y la **Compañía Cervecera de Canarias**. Su predisposición ha sido fantástica. Impresionante. Tanto una como otra. Me han provisto de todas las facilidades de acceso, tiempo y recursos. No he echado en falta nada de nada. Absolutamente nada. He contraído una gran deuda con ellos que en este momento aún no he saldado debidamente. Sobre esto quisiera destacar en una nota especial a **Sergio** de la Compañía Embotelladora de Canarias que nos ha arropado estupendamente y a los **dos talleres mecánicos** que diseñaron y construyeron los versátiles soportes físicos[1] sobre los que se asienta MONICOD.

Así mismo creo justo dedicarle una mención especial a la empresa distribuidora de cámaras y fuentes de iluminación **Infaimon.SL** que han probado, de forma consistente, cortesía y excelente trato al responder a nuestras preguntas y permitirnos experimentar con su material.

Hay muchos, muchos más que merecerían aparecer en este apartado. Demasiados. Pero sencillamente no puedo hacerlo. Sólo tenía dos páginas destinadas a esto. Y ahora sí puedo decirlo.: ¡Qué sepan perdonarme!

[1] Pueden verse en las figuras 5.16 y 5.17.

Resumen

La legislación indica que todo producto perecedero destinado a la alimentación humana debe exhibir la fecha a partir de la cual se considera que su consumo deja de ser seguro. Esta es la fecha de caducidad o expiración. Las latas de bebidas no son una excepción.

Es responsabilidad del productor que la fecha de caducidad esté presente y sea visible, legible e interpretable sin ambigüedad por el consumidor final. En propiedad la única forma segura de garantizar esto es la supervisión unidad por unidad dentro de la cadena de producción. La velocidad de la producción y su carácter masivo hace esta labor inasumible para un operario humano. Ahora bien; durante las últimas décadas se ha conseguido delegar esta supervisión a una máquina apoyándose en los avances en visión artificial (en particular en reconocimiento de caracteres) y en el desarrollo de cámaras de alta velocidad. La tarea es particularmente complicada porque el entorno es hostil. Para empezar la lata es validada en movimiento (no se detiene para ser supervisada) y la superficie metálica (hojalata o aluminio) donde se imprime la fecha de caducidad responde de manera compleja ante la iluminación, generando brillos y sombras muy pronunciadas. Estas circunstancias no se presentan en el clásico reconocimiento de caracteres sobre papel.

MONICOD, la herramienta presentada en este documento de tesis, es un esfuerzo más en esta área de aplicación de validadores industriales de fechas de caducidad. Su particularidad es su alta velocidad de operación. Esto es; trata de maximizar el número de latas por unidad de tiempo. Para lograrlo combina varias estrategias. En primer lugar se ha construido una plataforma física que amortigua algunos efectos indeseables del entorno de operación. Su elemento más prominente es una cámara oscura. En segundo lugar los algoritmos que se encadenan en la operación de validación gozan de una gran simplicidad, especialmente respecto al tipo de operaciones empleadas (sumas) y el tipo de datos (enteros). Esto los hace muy asequibles al cómputo. Por otra parte se ha evitado depender de un hardware específico y así MONICOD no depende de ningún fabricante en particular y puede correr en un computador doméstico.

Prefacio

"Citius, altius, fortuis"

Más rápido, más alto, más fuerte. Esto es lo que declara el lema olímpico[2]. Véase figura 1. Otra traducción más adecuada podría ser: *Citius*-más rápidamente-, *altius*-a mayor altura-, *fortius*-más fuertemente, con más fuerza-.

Es interesante apreciar que la primera de las consignas es *Citius* -más rápidamente-. Desde tiempos pretéritos[3] la velocidad ha sido importante para el hombre. Por tierra, mar o aire alcanzar velocidades cada vez más elevadas ha sido una constante ya sea por razones prácticas, científicas o lúdicas.

Claro que en un sentido más amplio la velocidad no se reduce a minimizar tiempos para cubrir distancias. Hay muchas cosas en las que se puede ser rápido[4]. Es necesario tiempo para ejercer casi cualquier actividad. Y cuando se trata de actividades productivas suele ocurrir que el tiempo es valioso y limitado. Gestionar tiempo es importante para la producción, y aprovecharlo bien es la primera consideración para una buena gestión del tiempo. A partir de ahí la necesitad de ser rápidos, esto es, hacer más en menos tiempo, surge con naturalidad como la medida por defecto para aprovechar mejor el tiempo.

[2] El lema de las olimpiadas modernas(1896-2???)

[3] Asociada probablemente con la depredación. Caza y huida.

[4] No hay más que acudir a una edición actualizada del libro Guinnes de los Récords para una buena colección de récords de velocidad en todo tipo de actividades y escenarios[2]

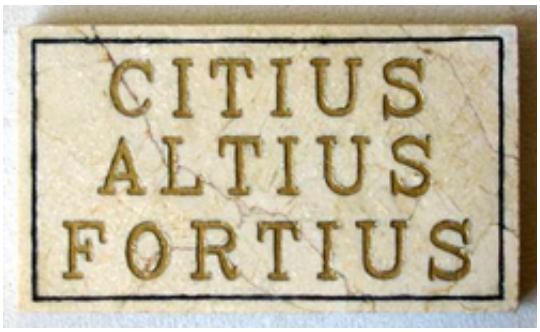


Figura 1: Lema olímpico.

Velocidad y MONICOD MONICOD-MONItorizador de CÓDigo- es una plataforma hadware/software que busca **ser rápida en una tarea de validación-de-códigos-impresos-en-superficies-de-latas-que-se-desplazan-encima-de-una-cinta-transportada-industrial**.

- Es una plataforma Hardware porque existe un componente físico irrenunciable que le permite acometer la tarea. Este componente va más allá de una máquina de computo. Y su importancia no debe ser despreciada. El capítulo 5 profundizará sobre ello.
- Es una plataforma Software porque existe un componente algorítmico implementado en forma de programa computable que es el corazón de MONICOD. El capítulo 6 versa principalmente sobre eso.

Velocidad y precio Si examinamos con detenimiento como se alcanza velocidad en cualquier actividad no tardaremos en darnos cuenta de la presencia de una constante interesante: El precio. Conseguir 'velocidad' ineludiblemente implica pagar un precio.

También este documento, a parte de velocidad, trata sobre el precio. Y, sin embargo, no se trata de velocidad a cualquier precio. En general la velocidad a cualquier precio no resulta práctica[5]. En MONICOD existe la pretensión de ser viables en un entorno real plagado de consideraciones prácticas. Es necesario alcanzar una suerte de compromiso que resulte aceptable entre la velocidad y el precio. Puede ocurrir que sea necesario renunciar a ser todo lo "rápido" que se es posible ser porque el precio a pagar sea demasiado alto. Ese documento también trata de ese compromiso.

¿En qué se es rápido? Conocer en qué tenemos que ser rápidos es imprescindible no sólo para saber en qué competimos, sino también para tomar decisiones correctas y decidir prioridades. Hablamos de validar, de códigos, de impresión, de superficies, de latas, de desplazamiento (o movimiento), de cintas transportadoras, y de un entorno industrial. ¡Esto es mucho! Como se verá lo largo de este documento, MONICOD es sólo una parte de la cuestión. Habrá que hacerse preguntas y detallar respuestas sobre cada una de las cosas que le rodean. Porque cada cosa tiene un papel relevante en nuestra búsqueda de mejores velocidades y contribuye a decidir qué podemos y qué no podemos hacer.

Para ser rápido en algo, cualquier cosa, todo conocimiento previo sobre ese "algo" ayuda.

¿Ser rápido para qué? A efectos del trabajo aquí presentado se necesita ser rápido porque no serlo es inaceptable. Como quedará explicado más adelante, debemos ser, al menos, lo suficientemente rápidos para validar al ritmo con que cada lata impresa llega al sistema si queremos garantizar que toda lata sea

[5] Una excepción de esto es la carrera por un nuevo récord Guinnes.

supervisada[6]. Una lata sin supervisión significa incumplir la tarea de validación. Por otra parte existe un límite de velocidad por encima del cual ser más rápido no comporta ventaja alguna.

Sobre la estructura de este documento.

Este documento consta de doce capítulos.

El capítulo 1 ***La iniciativa*** introducirá el objeto de estudio del documento, a saber, *La Supervisión Visual A Muy Alta Velocidad De Codificación Impresa Industrial*. La introducción será desarrollada desde el contexto en el que surge la problemática discutida en el documento. Así se ofrece una retrospectiva de las circunstancias que hacen derivar el interés por un objeto de investigación tal como MONICOD.

El capítulo 2 ***MONICOD y el problema clásico del reconocimiento óptico de caracteres*** considerará la teoría preexistente sobre este campo y el modo con que esta se ajusta a las imposiciones reservadas a MONICOD.

El capítulo 3 ***El problema: Estudio preliminar*** indagará sobre el dominio. Examinaremos, procurando ser exhaustivos, los elementos que juegan algún rol en el escenario.

El capítulo *4* ***Obstáculos y contra-medidas*** servirá para poner de manifiesto los obstáculos y barreras que complican alcanzar nuestro objetivo, así como las soluciones inmediatas adoptadas para algunos de ellos.

El capítulo 5 ***La plataforma física y herramientas software*** expone los elementos físicos y programáticos que dan soporte al desarrollo y funcionamiento de la solución planteada. La solución MONICOD necesita ineludiblemente de la plataforma para proceder.

El capítulo 6 ***MONICOD: Preparativos*** presenta la filosofía y algunos conceptos que maneja MONICOD y colecciona hechos relevantes en su universo. También considera el protocolo de calibrado del sistema.

El capítulo 7 ***MONICOD: Extracción de caracteres*** es probablemente el más extenso del documento, y describe el procedimiento de extracción de caracteres desde la imagen adquirida. Es en realidad una colección de métodos dispuestos secuencialmente. Así detectamos lata, preprocesamos, segmentamos fragmentos de tinta, asociamos dichos fragmentos a líneas de código, y los agrupamos en caracteres.

El capítulo 8 ***MONICOD: Validación y aprendizaje*** se divide en dos partes. La primera parte trata ***La validación*** efectiva de los caracteres segmentados en el capítulo previo. Para ello se introduce una base de caracteres almacenados que llamamos BFM. Estos caracteres almacenados nos servirán de referencia frente a los caracteres adquiridos. También se presentará una medida

[6] Esta exigencia puede suprimirse trivialmente manteniendo varias réplicas de validación en paralelo para una única línea de impresión. Para los propósitos de este documento ignoraremos la solución trivial.

de distancia entre caracteres y una metodología para usarla. La segunda parte versa sobre ***El aprendizaje.*** Allí se explica el procedimiento para adquirir conocimiento nuevo a través de una estrategia mixta supervisada y no supervisada que gestiona la inyección de nuevos datos y la supresión de datos obsoletos (adaptándose así a variaciones en los elementos validados) en la mencionada BFM.

El capítulo 9 ***Resultados*** recopila estadísticas sobre MONICOD. Además compara algunas de sus partes con otros algoritmos contrastados. Sólo se consideran dos índices: Calidad y Tiempo.

El capítulo 10 ***Conclusiones y extensiones*** se compone en dos partes. La primera parte, ***Conclusiones***, estudia la excelencia -fortalezas y debilidades- alcanzada en el cumplimiento de la tarea. A saber, clasificar latas (una a una) en dos categorías: 'Código de caducidad legible' y 'código de caducidad ilegible'. La segunda parte***, Extensiones,*** versará sobre las adiciones y cambios a la solución alcanzada que previsiblemente mejorarían el cumplimiento de los objetivos.

Finalmente en los ***Anexos*** se comentan tópicos de interés relativos al trabajo presentado pero que por razones de diversa índole no forman parte del cuerpo central del documento.

Sobre las herramientas de documentación.

LyX Este documento ha sido escrito con la ayuda de LyX[3]. LyX es un procesador de documentos multiplataforma de código abierto y libre que combina el poder y flexibilidad de TeX/LaTeX con la facilidad de uso de una interfaz gráfica. LyX sigue la filosofía WYSIWYM[7] en oposición a WYSIWYG[8].

TeX/LaTeX es un lenguaje de creación de documentos que permite concentrarse en la estructura del documento, y no simplemente en su apariencia.

Jabref Jabref[4] es un gestor de referencias bibliográficas de código abierto, libre y multiplataforma.

Doxygen Doxygen[5] es un generador de documentación a partir de código fuente C/C++ de código abierto y libre.

Smartdraw Smartdraw[6] es un "procesador visual" comercial con el que es posible generar elegantes figuras e ilustraciones que complementen un texto de forma rápida y sencilla.

MATLAB MATLAB[7] es un entorno de programación para el desarrollo de algoritmos, el análisis de datos, la visualización y el cálculo numérico. En

[7] What You See Is What You Mean. Lo que ves es lo que significa.
[8] What You See Is What You Get. Lo que ves es lo que obtienes.

este documento, ha sido utilizado específicamente para generar[9] las gráficas que ilustran el capítulo de ***Resultados***.

A ser tenido en cuenta

Sobre las figuras

En este documento hay numerosas imágenes particularmente oscuras que aparecen ennegrecidas y sin detalle al ser impresas. Para evitar en lo posible que quede comprometido su valor como apoyo a la explicación se ha optado por aplicarles un tratamiento (un valor de gamma de 1.25) que mejore su visibilidad a costa de perder fidelidad a la muestra original y ocultando en parte al lector la dificultad la falta de contraste. Las figuras así modificadas son indicadas con el símbolo (☼).

Sobre el pseudocódigo

El pseudocódigo de los algoritmos mostrados en este documento es fiel a su descripción, pero podría no encontrarse codificado tal cual en el código fuente original. En realidad podría diferir bastante. Esta aparente contradicción responde a la imperiosa necesidad de ser claros cuando procede.

1. *Abstracción allí donde es posible y conveniente, para evitar profundizar más allá de lo necesario. Existen detalles y operaciones que se han estimado no merecedoras de mayor detalle.*

2. *El código mostrado se presenta NO optimizado. Si bien no son raras las ocasiones en que un código simple y claro también resulta ser el más óptimo, también hay situaciones en que las maniobras de optimización oscurecen la presentación del procedimiento.*

3. *Las consideraciones relativas a la gestión de la memoria se han suprimido. La creación, asignación, supresión de áreas de memoria y condiciones necesarias para hacer buen uso de ellas se omiten. En general se asume memoria infinita en el código presentado.*

4. *En multitud de ocasiones se omiten detalles de la implementación asociadas a la idiosincrasia del lenguaje, y cuya razón de ser puede ser localizada en un buen texto de referencia de C/C++ como [8] o [9] para una visión más exhaustiva.*

[9] Otras herramientas como Freemat u Octave, de código abierto y libre, podrían haberlo substituido perfectamente.

Índice general

Capítulo 1

La iniciativa

Que trata del origen de la materia sobre la que trata este documento y donde se exponen las circunstancias que dieron lugar al proyecto

En este capítulo colocamos, literalmente, a MONICOD dentro de la cadena. Tratemos:

- La industria en general y peculiaridades que de algún modo tienen que ver con MONICOD
- La escritura impresa
- La cuestión legal
- Génesis de MONICOD

1.1. Humanos, máquinas y cadenas de montaje

Industria puede definirse como el conjunto de procesos y actividades que tienen como finalidad transformar las materias primas en productos elaborados. Las máquinas son la herramienta que hace posible la transformación **masiva** de las materias primas a productos elaborados. La industrialización está intrincadamente unida al desarrollo tecnológico. Fue el desarrollo tecnológico lo que hizo posible la industrialización, y la industrialización ha demandado desde entonces constantes progresos tecnológicos.

La perpetua necesidad de mejorar la producción a menor coste, acarrearían a la industria sus propias revoluciones internas. En particular en el campo de la "ingeniería de procesos". La producción en cadena abanderada por Fred Taylor y Henry Ford (Fordismo) (principios del siglo XX) fue una de ellas. Ver figura 1.2. La división del trabajo en una serie de tareas específicas delegadas en obreros

Figura 1.1: Antes de que se estandarizase el conector de Edison los siguientes conectores para lámparas incandescentes estaban en uso en los Estados Unidos. De arriba abajo, izquierda a derecha: Siemens-Halske, Hawkeye, Italian, Westinghouse, Ediswan de contacto simple, Brush-Swan, Schaefer, Ediswan de contacto doble, Westinghouse, Perkins (o Mather), Weston, United States, Fort Wayne "Jenney", Edison, Edison, Thomson-Houston.

especializados y máquinas desarrolladas, colocadas en sucesión o cadena, combinada con el uso efectivo de un sistema de turnos de trabajo, supuso una mejor relación entre tiempo/costo y número de unidades producidas.

Sin embargo el factor clave detrás de la producción en cadena fue la estandarización. Componentes estandarizados, procesos de manufactura estandarizados y un producto estándar simple y fácil de manufacturar (y de reparar). Ver figura 1.1.

Estandarizar implica que las partes sean intercambiables. Esto es, iguales e idénticas entre sí. La cadena de montaje o ensamblaje es posible gracias a la estandarización. Y la propia cadena favorece la estandarización. Innovaciones en el desarrollo de las herramientas y en particular el motor eléctrico -que permite distribuir las máquinas en una línea de ensamblado sin estar incrustadas en un sistema de alimentación central como ocurría antes- facilitaron la re-configuración del flujo de trabajo. El resultado de todo esto era que *"estandarizar un producto y manufacturarlo en masa disminuye su precio hasta el punto que el hombre corriente puede comprarlo*" [10]

Un caso especial (y probablemente el más importante) de producción en cadena es la cadena de montaje o ensamblaje. Ver figura 1.2.

Las cadenas de montaje (como por ejemplo las cadenas de envasado) favorecen la mecanización. Es decir, la reducción del factor humano mediante un proceso de intensa mecanización que reemplaza el puesto de trabajo de un obrero por máquinas capaces de cumplir con la tarea delegada de manera continua y autónoma sin casi intervención humana que no sea supervisión y mantenimiento. El carácter rutinario o el muy escaso número de decisiones (si alguna) a adoptar en la línea de montaje resulta apto para máquinas que, con la tecnología disponible, se les da mucho mejor que al ser humano común la actividad rutinaria o

Figura 1.2: Producción en cadena en una fábrica de Ford (1913)

"mecánica". Eso explica porque la robótica ha tenido una rápida implantación en cadenas de montaje. Ver figura 1.3.

Figura 1.3: Robots industriales en una línea de automoción

Por otra parte, hoy por hoy, a las máquinas no se les da bien tomar decisiones "inteligentes". Sus decisiones deben ser gobernadas por políticas que sigan reglas muy bien definidas. Si no existe un esquema así para esa decisión probablemente deba ser asumida por un ser humano. Además, mas allá de la tarea de control hay otras actividades donde la substitución humana por máquinas no siempre es posible. Tal es el caso de las actividades que envuelven procesamiento visual de alto nivel.

1.2. Cadena y eficiencia

Reflexionemos ahora con mayor detalle sobre algunos fundamentos del modelo de cadena de montaje - un tipo especial de producción en cadena -. Sería ingenuo suponer que toda manufactura que podamos imaginar es susceptible de ser descompuesta en tareas consecutivas (es decir en una cadena de montaje) donde podamos manipular la velocidad del conjunto a voluntad. Hay tareas (indivisibles e inevitables) que, bien porque son especialmente complejas, porque requieren tiempos físicos mínimos ineludibles, o por otras razones, exigen más tiempo que otras. Si esto ocurre, y dado el principio de secuencialidad entre tareas (donde unas tareas dependen del cumplimiento de otras para iniciarse) existirán problemas en el balance de la eficiencia de la cadena. Esto es, hay peligro de que hayan tareas obligadas a esperar. Y así todos los elementos asociados a ellas - obrero, protorobot, herramientas-, se mantienen ociosos durante periodos de tiempo. En otras palabras; parece inevitable que las tareas indivisibles que más tiempo exigen para su ejecución sean las que condicionan la velocidad máxima de la cadena de montaje. Esto es indeseable en términos de productividad/recursos invertidos.

Estas consideraciones son ya clásicas y conocidas por ciencia, ingeniería y matemáticas. Áreas de conocimiento como la teoría de la planificación, teoría de colas o la logística lidian con este asunto aceptando las restricciones de entrada y reorganizando la secuencialidad, concurrencia o simultaneidad de las tareas. En este dominio se asume que es imposible (o prohibitivo) mejorar la ejecución de la tarea problemática y así, se la trata como una caja negra con un costo asociado inevitable. Ver figura 1.4

No es nuestro propósito profundizar sobre esta cuestión apasionante pero merece la pena subrayar que, dentro de una cadena de ensamblado industrial, un importante conjunto de estas tareas problemáticas tienen que ver con el factor humano. Esto es, el ser humano tiene gran parte de responsabilidad en la ineficiencia de la cadena de montaje. No es sorprendente porque, después de todo, los organismos biológicos NO han sido concebidos en origen como autómatas dentro de una eficiente cadena de montaje. Y, optimizar la actividad humana tiene límites. Adicionalmente el factor humano introduce una componente de falibilidad desproporcionada respecto al margen de error de las máquinas (siempre que su mantenimiento y empleo sea el adecuado) porque, para un ser humano común, mantener una atención exhaustiva y prudente sobre una tarea monótona durante demasiado tiempo resulta muy difícil.

Como veremos en la sección 1.3 una de estas tareas problemáticas es la lectura de códigos industriales,

1.3. Procesamiento visual y cadenas de montaje

El procesamiento visual es un logro evolutivo. Una habilidad sofisticada que

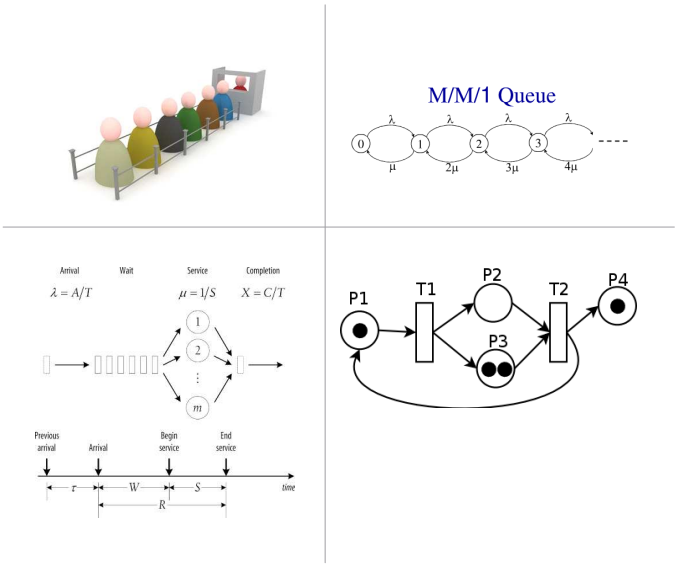


Figura 1.4: La teoría de colas, una rama de las matemáticas, hace uso de las cadenas de Markov para explicar el comportamiento de sistemas con colas, mientras que otros modelos matemáticos como las redes de Petri se emplean para enfrentar la problemática del control de flujo.

le supone a nuestro cerebro considerable espacio y esfuerzo.

La necesidad de procesamiento visual aparece con frecuencia en las cadenas de montaje. Así, ocurre con frecuencia que es necesario obtener una medida visual para tomar algún tipo de decisión crítica. Hay muchos niveles de complejidad detrás del término "procesamiento visual". Detectar simplemente la presencia o no de luz puede entenderse como una forma de procesamiento visual para la cual existen, afortunadamente, soluciones fotosensibles bien conocidas, bastante rápidas, relativamente baratas y suficientemente precisas.Ver figura 3.12.

Dada la especificidad de la tarea, en ocasiones es posible esquivar fácilmente el "procesamiento visual" con ingenios. Una sonda de nivel puede decir el nivel de líquido en un depósito mejor incluso que una apreciación visual. Y ciertamente hay ocasiones en que resultan muy convenientes otras formas de procesamiento visual alternativas más próximas a la de algunos animales - como la visión ultravioleta o infrarroja - o inéditas para la naturaleza conocida -como los rayos X-. En cualquier caso el objetivo es siempre discernir (de una forma manejable) características significantes [1] en el objeto de estudio[2] en el contexto de operación de la cadena de montaje o cadena de envasado.

[1] Pej. Tinta negra sobre hojalata o aluminio
[2] Pej. Bases de lata

Por fin, la lectura de códigos alfabético-numéricos (una aplicación de la escritura en la industria) en una cadena de montaje sin otro recurso que la percepción visual es un problema significativo dentro de cadenas de montaje, y sobre él versa el trabajo aquí presentado.

1.4. Escritura e industria

En ocasiones existe cierto tipo de información individualizada que cada unidad de producto debe comunicar al consumidor de forma segura y durable en orden a ser utilizado correctamente. Cuando esto ocurre entran en consideración varios factores.

- La importancia de la información. En general será obligatoria, crítica o muy importante porque de otra manera no se plantearía el esfuerzo extra de hacerla llegar de forma segura y permanente. La información de carácter publicitario, si bien importante para la marca, su degradación suele resultar tolerable.
- La naturaleza de la impresión. Escrita o no, individual o no, durabilidad, acceso,...
- La naturaleza del mensaje. Significado, destinatario(puede haber varios tipos de usuario), cómo se genera el mensaje, ...
- El encarecimiento del producto por el costo de incluir esa información.

La tarea de dotar al producto con la capacidad de comunicar un mensaje de cierta relevancia/complejidad está incluida en la cadena de montaje/embasado y por tanto forma parte de la problemática descrita en la sección 1.2.

1.4.1. El procedimiento

En escritura industrial podemos clasificar los procedimientos según estén basados en pigmentación o no.

- El ejemplo más típico de escritura no basada en pigmentación es la escritura en relieve. Dicha escritura goza de un ilustre pasado. La escritura cuneiforme mesopotámica es el primer ejemplo de escritura del que se tiene constancia. Tiene interesantes ventajas. Es económica (limitado al costo de la superficie) y puede ser apreciada por el tacto además de por la vista (característica que aprovecha la escritura braille). También goza de una excelente durabilidad dependiendo de la superficie y la calidad de su conservación. Infortunadamente no ofrece un buen contraste visual perjudicando su legibilidad. El grabado y especialmente el estampado son dos técnicas para generar escritura en relieve que también se emplean en la industria.

- El procedimiento de impresión de texto más extendido se basa en el empleo de pigmentos a los que comúnmente se les conoce como tinta. La tinta suele asociarse a bolígrafos y pinceles aunque gran parte de la producción tintada existente proviene de imprentas e impresoras. El principio que hace funcionar a la pigmentación es el contraste visual. Es necesario un fuerte contraste de color entre el pigmento o tinta y la superficie sobre la que se aplica. Es preferible una superficie con un color homogéneo siendo el blanco y el negro los más extendidos para superficies y pigmentos. En el proceso es crítico que la superficie sea capaz de absorber el pigmento allí donde es aplicado sin redistribuirlo. El papel es, con diferencia, la superficie para texto tintado más extendida. Diferentes colores para la tinta son posibles dependiendo su elección del contraste y el objetivo que se persiga.

Como se verá más adelante MONICOD opera sobre texto tintado pero la superficie no será papel sino metal (aluminio/hojalata -acero con una capa de estaño-).

1.4.2. El aspecto

En el aspecto de la impresión de texto juega siempre un papel trascendental las tipografías de texto. Una de las definiciones más socorridas de tipografía nos viene del creador de la famosa fuente *Times New Roman:*

> *"Arte de disponer correctamente el material de imprimir, de acuerdo con un propósito específico: el de colocar las letras, repartir el espacio y organizar los tipos con vistas a prestar al lector la máxima ayuda para la comprensión del texto*"[11]

La historia de las fuentes de texto empieza mucho antes de la era digital y los computadores. Anterior incluso a la propia industria. Y, por sorprendente que pueda parecer anterior a la propia imprenta. Habrá que retroceder al medievo para vislumbrar su orígenes. La palabra fuente[3] procede del término francés '*fonte*', que viene a significar "algo que estuvo derretido". El término hacía alusión al procedimiento con el que se creaban los caracteres: Se empleaban moldes sobre los que se vertía metal fundido que al solidificarse forjaban los caracteres (Ver figura 1.5).

Con la aparición de la imprenta se aceleró la búsqueda de nuevos tipos de letra, estilos de letra (negrita, itálica,...), con diferentes tamaños... etcétera.

El término "fuente" fue utilizado por los impresores ingleses durante siglos para referirse a las piezas de metal que eran ensambladas para obtener un determinado tamaño, estilo o tipo. Durante el siglo XIX, la composición tipográfica pudo ser automatizada. El procedimiento tuvo tanto éxito que hasta los años setenta del siglo XX se utilizó ampliamente.

En los cincuenta, con el desarrollo de la técnica de foto-composición tipográfica, fue posible producir fuentes almacenables en rollos de film. Una de las

[3] 'Font' en inglés

Figura 1.5: Arco de Constantino (315 D.C). Los romanos ya disponían de sus propias fuentes estándar de texto.La inscripción en la imagen, son en realidad huecos dejados por las piezas de bronce que eran las letras.

ventajas de esta técnica es que el escalado era trivial con el empleo de técnicas de óptica. Por tanto era muy fácil a partir de una fuente producir todas su variaciones de tamaño.

A mediados de los setenta (siglo XX) hubo un breve periodo de convivencia entre las técnicas tipográficas tradicionales y las digitales. Esto fue así porque las imprentas digitales de la época eran enormes con unidades de computación muy pequeñas no diferenciándose demasiado de los otros tipos de imprenta. Fue en los ochenta (siglo XX) cuando la tipografía digital se impuso, se extendió, y el uso de fuentes se diversificó.

Algunas de esas fuentes, como 'Verdana', fueron diseñadas primariamente para su visualizado a través de una pantalla de computador mientras que otras permiten un extenso rango de aplicaciones. Impresión, diseño web, escritura... Las fuentes digitales facilitan enormemente el proceso de crear y cambiar la fuente a voluntad. Sin embargo el objetivo con las fuentes siguen siendo el mismo: Elegir la fuente correcta para cada propósito. Ver figura 1.6.

La fuentes que nos interesarán deberá tener soporte para caracteres alfabético-numéricos anglosajones que son los usados en la codificación de los códigos a ser validados por MONICOD.

1.4.3. El mensaje

El mensaje de naturaleza industrial más habitual corresponde a un código con una función identificativa. Es conveniente el empleo de un conjunto limitado de caracteres perfectamente identificables: Los caracteres numéricos y los

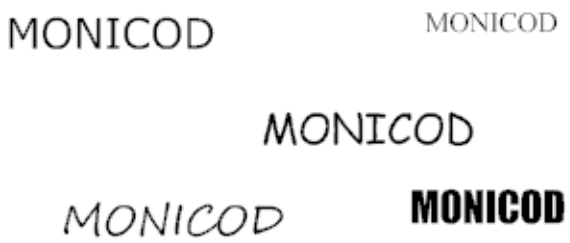


Figura 1.6: Diversas fuentes

caracteres alfabéticos latinos en mayúsculas son una elección común. El código se construye como una serie de caracteres dispuestos en línea y separados entre ellos por un espacio de no-tinta suficiente para darles independencia. La capacidad identificadora del mensaje depende del reconocimiento fiable de todos y cada uno de los caracteres[4]

1.5. Máquinas que leen

En general, hasta muy recientemente (a partir de los años cincuenta del siglo XX) la lectura "clásica" era una actividad restringida a los seres humanos. Y sigue siéndolo si nos circunscribimos a la actividad de la lectura "clásica" comprensiva.

Pero las demandas de mecanización en la clasificación y tratamiento de documentación escrita despertaron el interés de la ingeniería[5]. Así, ya en 1929 existe una patente de OCR[6] en Alemania y otra en USA en 1933[12]. Es muy interesante mencionar que en estos arcaicos OCRs de naturaleza electro-mecánica (ver figura 1.7) ya se vislumbraba claramente la primera técnica de reconocimiento: Comparación de mascaras o plantillas. La comparación era posible por aplicación del principio euclidiano de superposición .

Con los primeros computadores uno de los primeros problemas de reconocimiento que fueron abordados fue el OCR. Esto llevó a la creación y, en ocasiones, comercialización, de máquinas especializadas en el reconocimiento OCR. El procedimiento dominante era superponer una plantilla con el carácter de entrada, y medir el grado de coincidencia. Para lograr cierta invarianza podían utilizarse proyecciones verticales y/o horizontales.

[4] Aunque no sería imposible el uso de códigos detectores, o incluso correctores, de error que proporcionen cierta redundancia al código no son deseables porque incrementan la longitud de código.

[5] Y por supuesto también estaba la conocida fascinación por conseguir que las máquinas mimeticen actividades humanas.

[6] Acrónimo que deriva de ***O**ptical **C**haracter **R**ecognition*. Reconocedor óptico de caracteres.

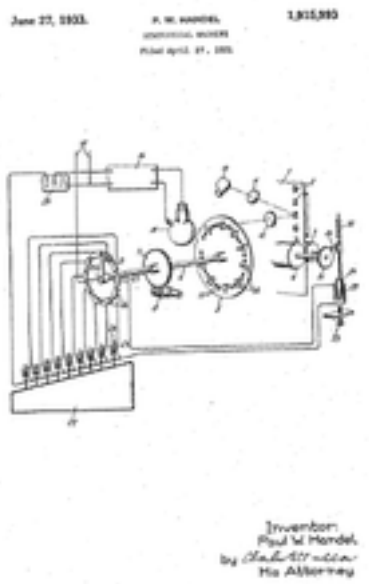


Figura 1.7: Patente USA de un OCR en 1933 [12]

El proyecto MONICOD es identificable inmediatamente como un esfuerzo más dentro de este dominio restringido, y cómo veremos en la sección 6 también guarda fuertes similitudes con el principio de funcionamiento descrito.

1.6. El reconocimiento de caracteres industrial (y no industrial)

El primer OCR práctico apareció en Estados Unidos en los cincuenta. La misma década en que apareció el UNIVAC, el primer computador comercial.

A principio de los sesenta, IBM produjo sus primeros modelos de lectores ópticos: El IBM 1418 (1960) y el IBM 1428 (1962). Estos estaban orientados a reconocer números desde una fuente de impresión o escritos a mano. Eran capaces de tratar texto impreso desde un catálogo de doscientas fuentes de impresión.

En lo que se refiere a reconocimiento de caracteres propiamente industrial sus antecedentes pueden hallarse en el servicio postal. En los sesenta, las operaciones postales fueron automatizadas proveyendo una ordenación mecánica por letras mediante OCR. Con esto la lectura automática de códigos postales para determinar el destino del correo fue posible. Así tenemos que en 1965, el servicio postal de los Estados Unidos fue el primero en introducir la lectura de direcciones de destinatario en formato Ciudad/Estado/Zip en sobres impresos.

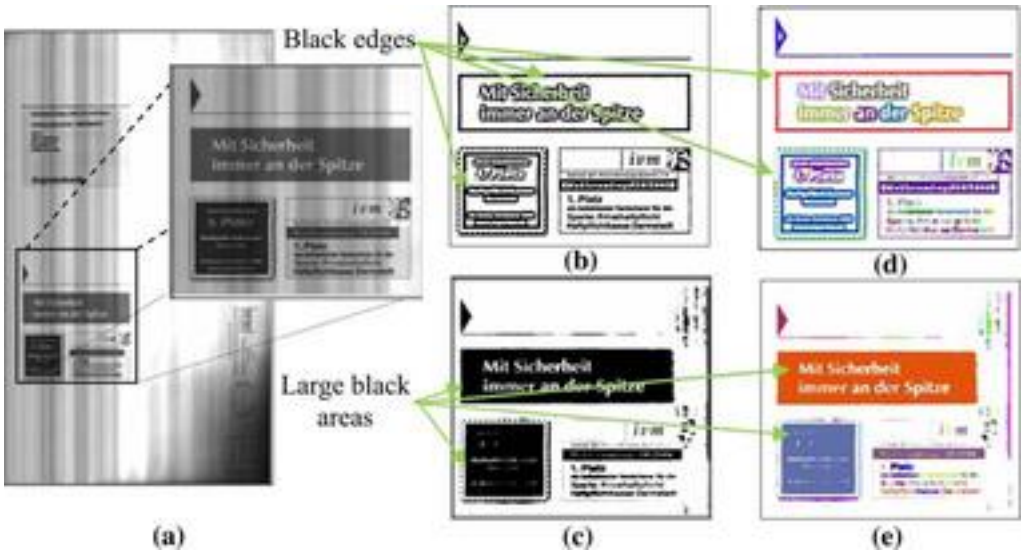


Figura 1.8: Estudio de la extracción de campos de un bloque de dirección (2008) [13].

Ver figura 1.8. En Japón, Toshiba y NEC desarrollaron la lectura de números (0-9) manuscritos para reconocimiento de código postal en 1968. El caso europeo es irónico. Fue en Alemania donde, en 1961 se introdujo por primera vez en el mundo el sistema del código postal que revolucionó el sistema postal. Pero no fue hasta 1973 (después de EEUU y Japón) cuando se implantó en Italia un sistema de lectura de códigos postales que permitía automatizar la ordenación del correo. En 1978 la implantación llegó a Alemania.

En lo que al OCR común se refiere, Hitachi produjo su primer reconocedor óptico de caracteres para texto alfabético-numérico impreso en 1968. Y el primer reconocedor óptico para números escritos a mano en 1972. Ver figura 1.7. Ahora bien, la problemática de la lengua escrita japonesa (emparentada con la china), ya fuera manuscrita o no, presentaba sus propios desafíos . NEC, Toshiba y Fujitsu trabajaron en el problema durante la década de los setenta. Ver figura 1.9.

En el lado industrial puede decirse que la banca fue un nuevo usuario de los servicios OCR. IBM introdujo en 1977 un sistema de procesamiento de depósitos (IBM 3895). Este tenía la habilidad de reconocer la cantidad de dinero depositada sin necesidad de que esta estuviera constreñida en el campo reservado para ese efecto. Parece que el 50 % de los depósitos eran tratados con éxito, donde el resto tenía inevitablemente que ser tratado a mano.

La década de los ochenta contempló avances significativos en dispositivos basados en tecnología de semiconductores. Sensores de Imagen CCD, microprocesadores, memorias dinámicas de acceso aleatorio (DRAM) y la habilidad de 'personalizar' circuitos integrados.

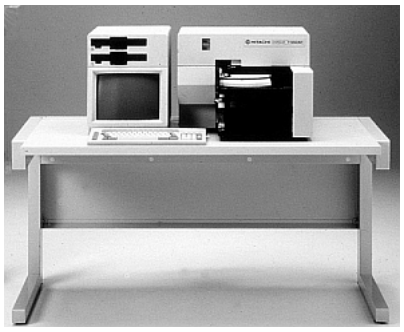


Figura 1.9: La unidad HT-560. Primer OCR de sobremesa de Hitachi (1982)

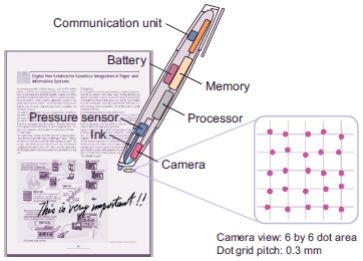


Figura 1.10: Bolígrafo digital usando funcionalidad "Anoto"

Estos progresos incidieron directamente en la tecnología OCR. Por ejemplo, ahora, con el abaratamiento de la memoria era posible escanear una imagen completa y almacenarla en la memoria para posterior procesamiento. También era posible llevar a cabo un procesamiento más sofisticado y hacer uso de técnicas más avanzadas.

Un recorrido detallado y extensivo de la historia de los OCR puede encontrarse en [14].

1.7. Legislación

MONICOD tratará con códigos de lata que codifican esencialmente códigos de caducidad.

Según el Instituto Nacional de Consumo[15]:

> Los requisitos sobre la indicación de fechas en los productos alimenticios vienen recogidos con carácter general en la Norma general de etiquetado, presentación y publicidad de los productos alimenticios, aprobada por el RD 1334/1999, de 31 de julio. Además, en

> un futuro próximo, cuando transcurra el plazo transitorio previsto en el Reglamento UE 1169/2011 sobre la información alimentaria facilitada al consumidor, las reglas para indicar las fechas límites de consumo serán las establecidas en esta norma.

En dicha legislación se indica en su artículo 5 que:

> ***Artículo 5***. *Información obligatoria de etiquetado.*
>
> *El etiquetado de los productos alimenticios requerirá solamente, salvo las excepciones previstas en este capítulo, las indicaciones obligatorias siguientes:*
>
> *a. La denominación de venta del producto.*
>
> *b. La lista de ingredientes.*
>
> *c. La cantidad de determinados ingredientes o categoría de ingredientes.*
>
> *d. El grado alcohólico en las bebidas con una graduación superior en volumen al 1,2 %.*
>
> *e. La cantidad neta, para productos envasados.*
>
> *f.* ***La fecha de duración mínima o la fecha de caducidad.***
>
> *g. Las condiciones especiales de conservación y de utilización.*
>
> *h. El modo de empleo, cuando su indicación sea necesaria para hacer un uso adecuado del producto alimenticio.*
>
> *i. Identificación de la empresa: el nombre, la razón social o la denominación del fabricante o el envasador o de un vendedor establecido dentro de la Unión Europea y, en todo caso, su domicilio.*
>
> *j. El lote.*
>
> *k .El lugar de origen o procedencia.*

Y en el artículo 11 se refuerza:

> ***Artículo 11***. *Marcado de fechas.*
> *En el etiquetado de todo producto alimenticio figurará la fecha de duración mínima o en su caso, la fecha de caducidad.*

En la misma normativa, en su artículo 3 se aporta también algunas definiciones interesantes para MONICOD:

> ***Artículo 3***. *Definiciones.*
> *A los efectos de esta Norma, se entiende por:*
>
> - *Etiquetado: las menciones, indicaciones, marcas de fábrica o comerciales, dibujos o signos relacionados con un producto alimenticio que figuren en cualquier envase, documento, rótulo, etiqueta, faja o collarín que acompañen o se refieran a dicho producto alimenticio.*

- *Producto alimenticio envasado: la unidad de venta destinada a ser presentada sin ulterior transformación al consumidor final y a las colectividades, constituida por un producto alimenticio y el envase en el que haya sido acondicionado antes de ser puesto a la venta, ya recubra el envase al producto por entero o sólo parcialmente, pero de forma que no pueda modificarse el contenido sin abrir o modificar dicho envase.*
- *Ingrediente: cualquier sustancia, incluidos los aditivos y los enzimas, utilizada en la fabricación o en la preparación de un producto alimenticio y que todavía se encuentra presente en el producto acabado, aunque sea en una forma modificada [...]. [Redacción según Real Decreto 890/2011, de 24 de junio]*
- *Lote: conjunto de unidades de venta de un producto alimenticio producido, fabricado o envasado en circunstancias prácticamente idénticas.*
- *Fecha de duración mínima: fecha hasta la cual el producto alimenticio mantiene sus propiedades específicas en condiciones de conservación apropiadas.*

1.8. MONICOD

MONICOD es un bot[7] de supervisión. Supervisa la cadena de producción siendo capaz de llevar a cabo acciones sobre esta.

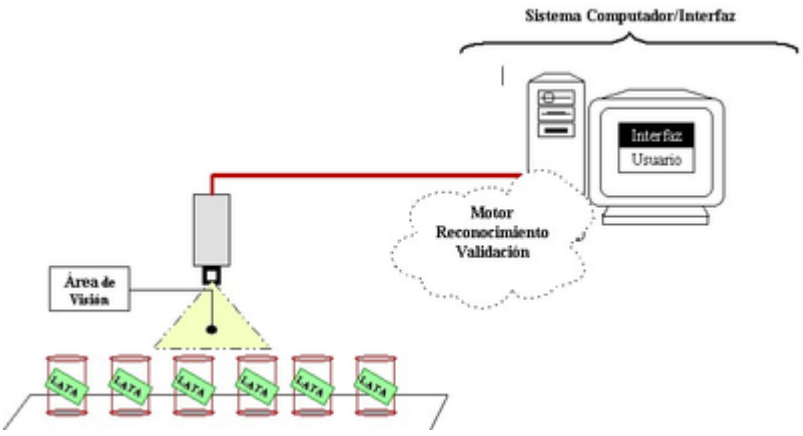


Figura 1.11: La primera representación gráfica del concepto de MONICOD.

[7] La palabra "bot" deriva de ro-bot y es parte de la jerga informática. Define a cierto tipo de programas informático orientados a imitar el comportamiento humano.

Este documento presenta la construcción de una solución de validación basada en reconocimiento de caracteres de códigos de caducidad en bases de lata de aluminio y hojalata (ver esquema descriptivo 1.11).

Las características críticas que lo definen son:

- La alta velocidad de ejecución para asegurar una cadencia de validación acorde a las imposiciones de la cadena de montaje.
- La no dependencia de un hardware específico (que no sea un sistema de adquisición de imágenes capaz de alcanzar un ratio de captura de *frames* por unidad de tiempo muy elevado con calidad suficiente)
- Siempre que sea posible, evitar dependencias con software que no sea libre y de código abierto (Free Open Source). Desgraciadamente se depende de los drivers y SDK suministrados por el fabricante.

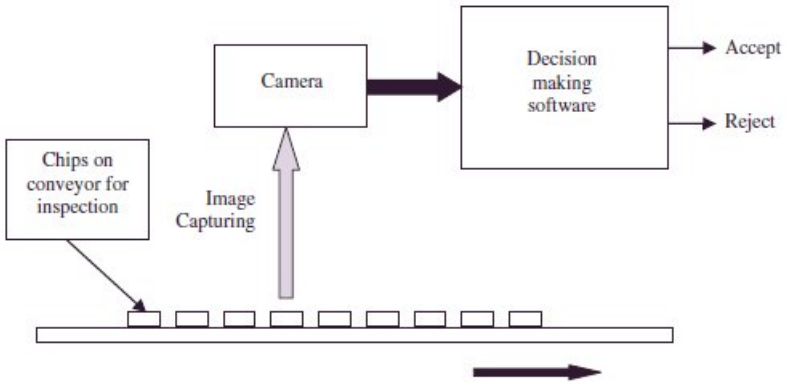


Figura 1.12: Diagrama esquemático de un sistema de inspección de marcado en *Texas Instruments* tal como aparece en [16]

El proyecto MONICOD arranca a partir de la iniciativa compartida de la Compañía Cervecera de Canarias y la Compañía Embotelladora de Canarias que derivó en un convenio entre la ULPGC (Universidad de Las Palmas de Gran Canaria), representada por el IUCTC (Instituto Universitario de Ciencias y Tecnologías Cibernéticas) como unidad de investigación ejecutora del proyecto, las empresas citadas y ASINCA (Asociación de Empresarios de Canarias).

Capítulo 2

MONICOD y el problema clásico del reconocimiento óptico de caracteres

"Donde se nos habla de la noble estirpe a la que pertenece MONICOD"

El enfoque de este capítulo no es, de ninguna manera, desarrollar exhaustivamente los tópicos de Reconocimiento Óptico de Caracteres o de Reconocimiento y Clasificación de Patrones. Intentarlo siquiera sería insensato y fútil. La cantidad apabullante de biografía sobre esta temática prueba la importancia, nivel de investigación, esfuerzo desarrollado y carácter casi inabarcable de la materia. Por otra parte, la naturaleza no didáctica de este documento impide una aproximación sistemática a la manera de un texto fundamental sobre la materia.

En lugar de eso el capítulo actual se enfoca desde el contexto del objeto de estudio central. A saber; Un Validador óptico de caracteres destinado a operar a muy altas velocidades en un entorno industrial exigente (MONICOD). Con este espíritu, el capítulo expondrá principios universalmente aceptados sobre el Reconocimiento Óptico de Caracteres o de Reconocimiento y Clasificación de Patrones que subyacen en la mayoría de los desarrollos con cierto éxito, y cómo estos principios se relacionan con MONICOD[1].

Así pues la cuestión será abordada en dos partes:

- Arquitectura de un reconocedor de caracteres clásico.
- Presentación de algunos métodos representativos de cada uno de sus fases.

[1] De especial interés serán los argumentos de no aplicabilidad de ciertas operaciones habituales en el dominio en un sistema de MONICOD

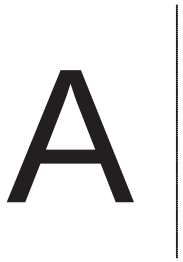


Figura 2.1: Distinguir entre "A" y "B". Un problema de OCR.

2.1. Reconocedores de patrones y reconocedores ópticos de caracteres

2.1.1. La importancia de reconocer patrones

MONICOD, el objeto de estudio de este documento, es un Reconocedor Óptico de Caracteres (OCR). Ver 2.1.

Los reconocedores ópticos de caracteres son una instancia particular de un problema mucho más general que tiene suficiente importancia como para apropiarse de su propia disciplina científica: El *Reconocimiento de Patrones*. Ver figura 2.2.

La tarea central del *Reconocimiento de Patrones* es la asignación de etiquetas a valores de entrada. El paradigma de esto es la clasificación de objetos en un conjunto dado de categorías o clases. Sin embargo el reconocimiento de patrones es más general que el problema de la clasificación. Así por ejemplo, el análisis de regresión[2] o el análisis sintáctico[3] ('parser') también son problemas de reconocimiento de patrones que no envuelven clasificación.

Un componente de MONICOD es un clasificador, de manera que nos centraremos en el aspecto específico de clasificación de patrones.

En la naturaleza reconocer patrones es crucial para la supervivencia, de ahí que haya desarrollado sofisticados mecanismos de base neuronal para tratar con esta actividad. Sobrevivir depende de decisiones y tomar una decisión acertada depende en gran medida de una interpretación correcta. Parte de este esfuerzo por interpretar o evaluar correctamente pasa por un reconocimiento acertado. Por si esto fuera poco, en el caso de la clasificación, determinar a que categoría o clase corresponde un objeto puede ser interpretado como una decisión. Es tal la importancia capital del aspecto clasificador dentro de la decisión que podemos

[2] El análisis de regresión persigue el entendimiento de la relación de dependencia entre una variable dependiente y las variables independientes de las que depende.

[3] El análisis sintáctico es el proceso de analizar los elementos del texto para determinar la estructura gramatical, que es paso previo para su correcta interpretación.

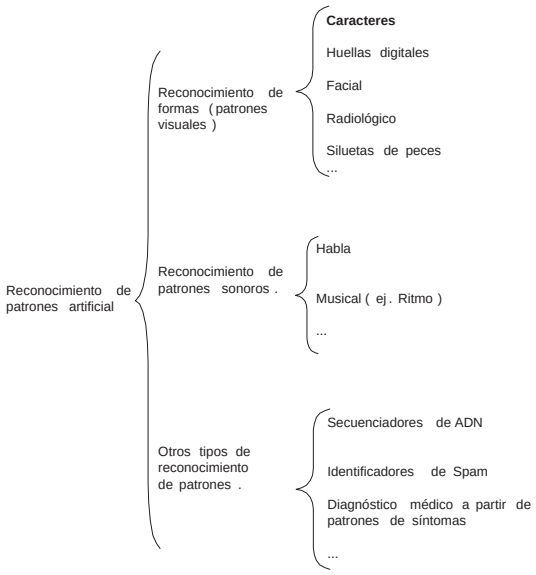


Figura 2.2: Los reconocedores de patrones clasificados por la naturaleza sensorial de las muestras.

ver a esta disciplina como una subárea dentro de La Teoría de la Decisión[4].

El reconocimiento artificial de patrones deriva del interés por máquinas que tomen decisiones acertadas. Al comparar los reconocedores artificiales actuales de patrones con sus homólogos naturales en ocasiones cunde cierta desazón. Algunos problemas que se nos antojan elementales parecen inaccesibles para las máquinas. Pero la realidad es que los reconocedores artificiales de patrones puede clasificar mucho mejor que un ser humano en una gran cantidad de dominios. Y cuando clasifican, lo hacen "industrialmente": Ofrecen mayor rapidez y la calidad del trabajo no se resiente por el número de clasificaciones o por el tiempo salvo falla en la maquinaria. Además, es habitual que se fundamenten en una sólida base matemática que respalde sus decisiones[5].

2.1.2. Reconocimiento y aprendizaje

La fortaleza de la habilidad -natural- del reconocimiento de patrones (de la que los humanos somos un portentoso ejemplo) no se basa tanto en proporcionar fantásticos ratios de acierto en un problema de clasificación concreto, sino en la flexibilidad para enfrentar problemas de clasificación de naturaleza muy diversa.

[4]Un aspecto importante en la teoría de la decisión es la idea de que toda decisión implica un costo y resulta muy conveniente encontrar reglas de decisión que minimicen ese costo.

[5]Claro que existen reconocimientos de patrones en que la lógica determinista no es suficiente. En esos casos complicadas lógicas estocásticas e inextricables heurísticas pueden hacerse necesarias o bien el problema espera a ser abordado con el avance de la ciencia.

Identificar cuando un correo es "spam" o no o cuando una seta es o no venenosa son problemas de clasificación susceptibles de ser aprendidos.

La percepción de la muestra. Los patrones. Las clases. La propia clasificación. Las decisiones asociadas a las clases... todos esos mecanismos son plásticos y moldeables en alguna medida. Bien puede decirse que nuestra habilidad no reside en el reconocimiento de patrones en si, sino en la habilidad de implementar diferentes reconocedores de patrones adaptados a nuestras necesidades.

En el reconocimiento artificial, uno de sus componentes- el clasificador- presenta en su ciclo de trabajo una fase denominada entrenamiento que captura esta característica natural. Sin embargo otros componentes (como el extractor de características) no suelen compartir esta destreza. Esta es una de las desventajas que lastran al reconocimiento artificial frente al reconocimiento natural.

2.1.3. Muestras y patrones

Muestra Ejemplos susceptibles de clasificación. Pueden ser de naturaleza visual, auditiva, olfativa, somato-sensorial o de otra naturaleza. Ver figura 2.3.

Patrón Las muestras pueden compartir de forma recurrente elementos constitutivos mas o menos evidentes que pueden derivar en una organización o modelo. La presencia de estas características o patrones desemboca en generalizaciones que permiten ordenar el universo de muestras, presuntamente enorme, de acuerdo a las clases. Identificar estos patrones es esencial. Ver figura 2.3.

2.1.4. Arquitectura de un reconocedor de patrones

El núcleo de la arquitectura de un reconocedor de patrones reside en dos entidades. El extractor de características y el clasificador (Ver figura 2.4).

Extractor de características El extractor de características obtiene del objeto un conjunto de propiedades con las que construye el denominado vector de características. El vector de características caracteriza al objeto ante el clasificador.

Selección de características Para que la clasificación sea efectiva es condición imprescindible que el vector citado contenga la información necesaria para determinar la pertenencia del objeto a una clase u otra. De otro modo la clasificación será imperfecta. Por otro lado condensar de forma efectiva el patrón original en un vector de características reducido resulta bastante conveniente porque evita al clasificador lidiar con demasiada información. La redundancia o presencia de datos irrelevantes pueden ser causas para un vector de características injustificadamente grande.

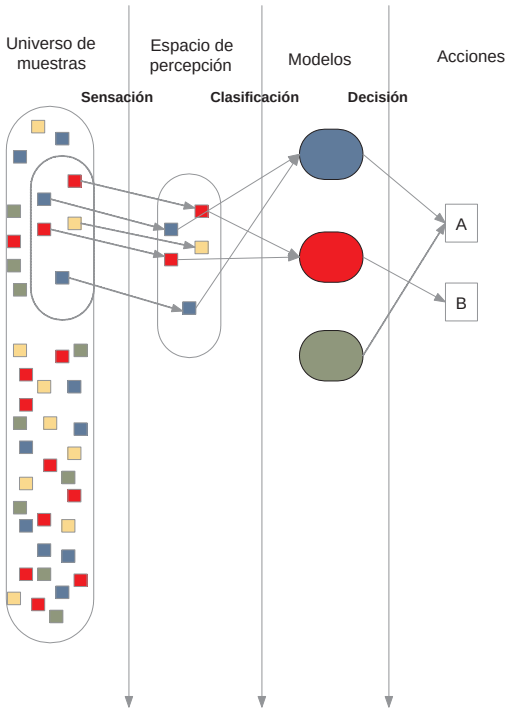


Figura 2.3: Relaciones entre espacios

Características cuidadosamente elegidas podrían mejorar las oportunidades de generalización. Este es el caso de las características invariantes a transformaciones que no alteran la condición de pertenencia del objeto[6]. Un indicativo de una buena selección suele ser que las propiedades extraídas que la comprenden sean similares entre objetos de una misma categoría pero difieran entre objetos que pertenecen a categorías distintas.

En general, la selección de las características pertinente es una tarea bastante delicada y es llevada a cabo por el diseñador del reconocedor. En este proceso de selección el conocimiento previo en el dominio del problema tiende a jugar un rol importante.

Resulta sorprendente que existiendo una sólida base matemática para buena parte de los clasificadores, el ensayo y error juega un importante papel para obtener buenos resultados en muchos casos de clasificación de patrones. Un ejemplo significativo de esto es la construcción del vector de características: La elección de las características adecuadas no siempre es evidente y puede significar una gran diferencia para el éxito o el fracaso.

[6]Una 'A' al final o al principio de la página sigue siendo una 'A'. Es independiente de la posición.

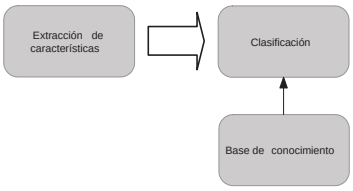


Figura 2.4: El núcleo de un clasificador de patrones

Clasificador El clasificador es posiblemente la parte más importante del reconocedor de patrones[7]. Es aquí donde el objeto, representado por el vector de características, es etiquetado o asignado en una clase u otra. Este problema tampoco es trivial.

Si las entradas son muy parecidas entre sí pero sin embargo pertenecen a clases diferentes, separarlas puede ser bastante difícil. Esta es una situación indeseable, y puede ser síntoma de una mala selección en las características a extraer. Por otra parte ser muy estrictos con la frontera de decisión puede ser contraproducente si se requiere una buena capacidad de generalización.

Entrenamiento Muchos clasificadores tienen la habilidad de aprender los términos de la clasificación y por ello requieren un entrenamiento previo para construir su base de conocimiento. A grandes rasgos el entrenamiento puede ser supervisado (con asistencia externa) o no supervisado (sin asistencia externa). La selección de un buen conjunto de entrenamiento es determinante y de nuevo se exige un esfuerzo previo por parte del diseñador para hacer una buena selección.

Una regla de oro es que el conjunto de entrenamiento tenga ejemplos de todas las clases a separar.

No sólo la selección del conjunto de entrenamiento es importante sino que, dependiendo de la naturaleza del clasificador, también importa la manera en la que se lleva a cabo el entrenamiento. En general es indeseable mantener un orden en la exposición de las muestras porque condiciona el aprendizaje. El clasificador, también sufre procesos de olvido y refuerzo durante el entrenamiento, y un determinado orden puede favorecer que se clasifique mejor entre unas muestras de entrada que entre otras.

Calidad de la clasificación Los criterios para medir la calidad de la clasificación son variados. El simple recuento de aciertos y fallos es bastante directo. Sin embargo esto no tiene en cuenta el impacto o costo de fallar o acertar en un determinado caso, que no tiene porque ser el mismo que en otro. De cualquier modo siempre hace falta una selección realista de muestras en el conjunto de test para obtener una medida de calidad realista.

[7] Aunque para el éxito global son igualmente determinantes otras etapas como la extracción de características.

Preprocesamiento y postprocesamiento Habitualmente el núcleo del reconocedor de patrones precisa de un preprocesamiento y postprocesamiento. Ver 2.5.

- El preprocesamiento recibe las muestras del mecanismo de adquisición de muestras (**Obtención de entrada**) y se ocupa de hacerlas "digeribles" al núcleo de procesamiento. Cumple principalmente esta función suprimiendo o amortiguando la presencia de artefactos extraños o ajenos que pueden distraer al núcleo de la arquitectura. El ejemplo más evidente de esta clase de artefactos es el ruido en la entrada.

  Un esfuerzo de preprocesamiento menos general y más dirigido ocurre cuando el objeto de entrada es en realidad un contenedor de objetos. En tales casos es conveniente separar el(los) objeto(s) que interesa(n) clasificar de todo lo demás. Es la segmentación. Este problema suele ser no trivial y tiene suficiente entidad para que algunos autores lo traten de forma separada al preprocesamiento más genérico.

- El postprocesamiento responde a la necesidad de actuar en consecuencia dada la clase a la que ha resultado pertenecer el patrón de entrada. Muchas veces el postprocesamiento es un mecanismo de toma de decisión que lanza la rutina de actuación que corresponde (**Toma de decisiones**). La necesidad de tomar esta decisión suele ser la razón de la existencia de todo el clasificador de patrones.

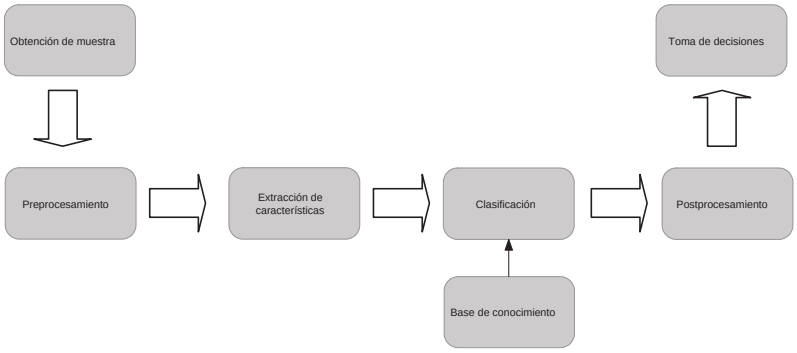


Figura 2.5: Arquitectura de tratamiento de una muestra

Retroalimentación El resultado de la clasificación de muestras puede alimentar a etapas previas además de al postprocesamiento.

Si el sistema detecta dificultades o no clasifica correctamente el problema debe estar o en el sistema o en la propia muestra o en ambos. Si es capaz de discernir entre situaciones e identificar el problema quizás sea posible reclasificar la muestra con configuraciones alternativas de parámetros. Cambios en la iluminación o en el grosor del texto que se reconoce, pueden ser absorbidos con

ajustes puntuales a los parámetros. También puede aprovecharse para incluir las adaptaciones en una base de conocimiento. De esta forma hacemos un sistema de reconocimiento más adaptable.

La arquitectura global es expuesta en la figura 2.6. Obsérvese además que en la figura se separa el preprocesamiento genérico de la segmentación de patrones.

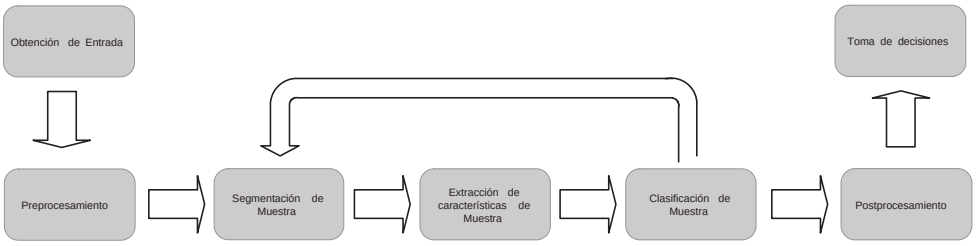


Figura 2.6: Arquitectura global de tratamiento de múltiples muestras en una captura

2.1.5. Construcción de un reconocedor de patrones

Clasificar es un problema abierto, no existe (aún) un clasificador artificial universal ni existen (aún) procedimientos sistemáticos para construir clasificadores de una determinada familia de patrones. En el excelente [17] se plantea un razonable esquema a seguir en el diseño de reconocedores de patrones. Ver 2.7. En este esquema la elección de características y la elección de modelo vienen dadas por el conocimiento previo con que se cuente del problema, pero también de la evaluación de la calidad de los resultados conseguidos. Así, de forma más o menos iterativa, tratando de maximizar la calidad del resultado, se seleccionan características y modelos.

2.1.6. ¿Qué es un OCR?

Un OCR o reconocedor óptico de caracteres se ajusta modélicamente a la idea de reconocedor de patrones.

- Es un tipo de reconocedor.
- Los elementos a reconocer son caracteres.
- Es 'óptico' porque la fuente de entrada es una captura de luz (niveles de intensidad lumínica y, eventualmente, color). Una imagen o fotografía que ha sido digitalizada para ser tratada.

He aquí la definición de OCR en una de las bases de conocimiento más consultadas:

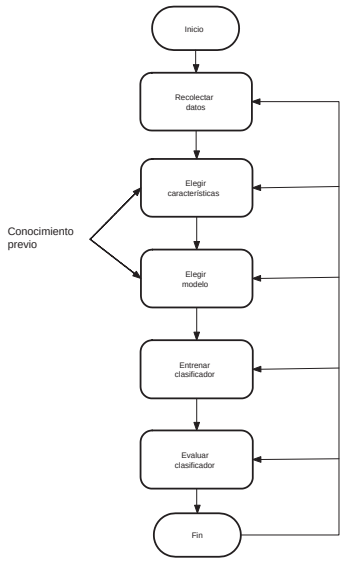


Figura 2.7: Flujo de diseño de un clasificador de patrones según [17].

> *Es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos, así podremos interactuar con estos mediante un programa de edición de texto o similar*[18][8].

Con el advenimiento de lo digital, transducir desde el universo no-digital al digital las pequeñas unidades que conforman mensajes textuales escritos en medios analógicos tales como una hoja de papel o la superficie de una lata, y hacerlo de un modo automatizado, es una necesidad casi elemental.

Los OCR (vistos como los reconocedores de patrones que son), clasifican objetos (bidimensionales formas de color sobre un plano de otro color) en categorías (signos o símbolos que tienen o contribuyen a un significado).

Los OCRs tienen el estatus de campo tradicional de aplicación del reconocimiento artificial de patrones. Y sus más antiguos ejemplos son anteriores a la computación moderna. Ver a este respecto la sección 1.5.

Objetos a reconocer Los patrones a reconocer (sobre todo en sistemas de escritura alfabética) están perfectamente definidos.

Grafema Grafema es el término técnico que se refiere a las unidades indivisibles de un determinado sistema de escritura. Son elementos de significado

[8] Wikipedia NO es una fuente primaria de información. La inclusión de la cita aquí se debe a su inmensa popularidad en la comunidad hispano-hablante.

mínimo que componen los bloques de construcción con los que se construyen textos. Los términos glifo, signo y carácter son usados como sinónimos de grafema. El alfabeto, los números y signos de puntuación son grafemas.

Alógrafo Alógrafo es el término técnico para referirse a una de las formas visuales/gráficas distintas con las que puede representarse un determinado grafema. Cada alógrafo de un grafema es una variación visualmente distinta respecto a los otros alógrafos del grafema pero se refieren al mismo, luego resultan equivalentes. Así por ejemplo el grafema 'j' tiene diferentes alógrafos según el estilo (negrita (**j**), cursiva(*j*),...), la fuente de texto, la caligrafía del escritor, etcétera.

Carácter Aunque carácter es usado como sinónimo de grafema en lingüística, la literatura del reconocimiento de caracteres utiliza el término 'carácter' para referirse tanto a grafemas como a alógrafos. Procedemos en este documento de igual forma. Desafortunadamente, esto puede llevar a confusión entre la representación gráfica y la clase o categoría a la que corresponde.

2.1.6.1. El futuro de los reconocedores ópticos de caracteres.

La documentación digital no precisa reconocimiento óptico alguno, y cada vez está más presenta. Se genera y se lee texto digital con cada vez más frecuencia y teclados y pantallas pueden encontrarse por doquier. El papel declina frente al soporte digital[9]. Ahora bien, aunque hacer predicciones en tecnología siempre es arriesgado, no parece que en el sector de la industria, que es donde MONICOD se ubica, el texto impreso corra riesgo de ser reemplazado por el texto digital. Se necesita precisamente un texto con existencia física útil para etiquetar objetos que son físicos. Un código de barras (ilegible para lectura directa de un humano sin ayuda de dispositivos) o un chip (caro y además también ilegible para lectura directo por un humano) no parece que puedan competir con la vieja y fácil solución del texto escrito.

2.1.7. Reconocedores y validadores

MONICOD **NO** ha sido concebido como un reconocedor óptico de caracteres aunque en gran medida lo sea.

En la sección 1.8 del capítulo 1 se explicó que MONICOD debía "validar" códigos de caducidad impresos sobre superficies de latas de aluminio u hojalata. "Validar" (ver figura 2.8) no implica necesariamente "reconocer". Al validar sabemos de antemano y a ciencia cierta qué es lo que se espera que esté impreso allí. De otra manera no podríamos validar nada. En cambio un OCR común NO valida texto (en su modo de funcionamiento normal). Su propósito es "reconocer" el texto con tanta certeza como sea posible, y trasladarlo a un dominio

[9]Si bien, paradójicamente, la informática, y lo asequible de los sistemas de impresión, ha disparado el consumo de papel.

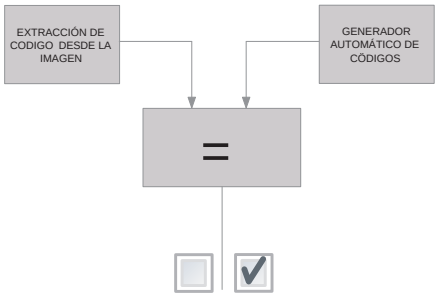


Figura 2.8: Validador como un comparador

digital. Allí cada carácter corresponderá a un número binario según alguna tabla de códigos, y no estará almacenado en una superficie analógica, tal como una fotografía, sino en un contenedor de datos digitales. Si es necesario llevar a cabo alguna tarea de validación, será una tarea ulterior obedeciendo a lo designios específicos de un cliente del OCR. Ciertamente hay situaciones en que el OCR conoce el texto de antemano y lo utiliza. Tal es el caso del entrenamiento automatizado. Pero en estos casos la meta no es validar la calidad de la impresión en la imagen de entrada (no directamente al menos) sino validar su propia destreza en el reconocimiento de texto con el fin de reajustar sus propios parámetros y/o adquirir nuevo conocimiento.

2.2. Fases del reconocimiento/validación

Las fases de un reconocedor/validador típico se ajustan bastante bien a la arquitectura descrita en 2.1.4. Ver figura 2.9.

1. Preproceso
2. Segmentación
3. Extracción de características
4. Clasificador
5. Postproceso

2.3. La entrada: La adquisición

El primer paso es la obtención de imágenes digitalizadas destinadas a ser procesadas, y en las que presumíblemente tiene que haber contenido texto. Un dispositivo denominado escáner es el medio de adquisición por excelencia porque

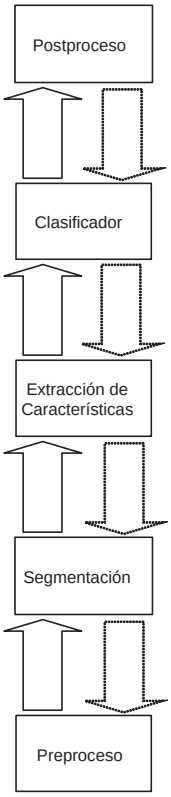


Figura 2.9: Un esquema por fases de un reconocedor/clasificador de formas: Preproceso →Segmentación →Extracción de Características →Clasificador →Postproceso. Sin embargo una fase posterior puede retroalimentar una fase previa para ajustar sus parámetros.

está concebido para la digitalización de papel que es el soporte más común de texto escrito fuera del computador.

> En MONICOD la fuente utilizada será una cámara de alta velocidad que proveerá capturas estáticas (*frames*) al sistema de la cadena de llenado. El sistema queda descrito en el capítulo 5.

Una adquisición bien ejecutada es esencial para el tratamiento, y condiciona las oportunidades de éxito en cualquier sistema OCR.

En los capítulos 4 y 5 se examina las consideraciones e instrumentos que la hacen posible.

2.4. Preprocesamiento

En 2.1.4 queda indicado que el preprocesamiento acondiciona a la imagen para las fases posteriores del reconocimiento. Por ejemplo la supresión de ruido. Es una fase crítica en tanto que su éxito o fracaso condiciona el resultado global. Hay un gran número de algoritmos de preprocesado, muchos de ellos de carácter general y no específicos para reconocedores de caracteres. Aquí nos limitaremos a describir los relacionados con MONICOD.

2.4.1. Suavizado y eliminación de ruido

Esta fase trata de suprimir o al menos reducir efectos no deseados en la imagen original. La procedencia de estos efectos es variada: Partículas de polvo en las ópticas, perturbaciones en la adquisición, muestreo, cuantización y transmisión... . Las técnica más común para enfrentar este problema es el filtrado en se encuentra desarrollado en cualquier texto básico de procesamiento de imágenes.

El filtrado consiste en aplicar un algoritmo a la vecindad de un píxel dado en la imagen de entrada obteniéndose un valor asignable al píxel correspondiente en la imagen de salida. El concepto de vecindad es examinado en A.3.2.1. Hay dos tipos de filtrado: Lineal y no lineal. Aquí nos centraremos en el filtrado lineal donde el valor de salida es obtenido como combinación lineal de la vecindad correspondiente en la entrada.

Un filtro o máscara se define como una matriz con una serie de valores o pesos. Dada la imagen de entrada E y la imagen de salida S, la multiplicación escalar del filtro y la vecindad de $E(i,j)$ -que incluye al propio píxel $E(i,j)$- proporciona el valor de $S(i,j)$. S se construye repitiendo esta operación para casa uno de los píxeles (i,j) de la imagen de entrada. Esta es la operación de convolución descrita con un poco más de detalle en A.3.6.

Algunos ejemplos de núcleos de filtros paso bajo 3×3 muy utilizados:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.1}$$

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

Estos filtros son útiles para suprimir ruido del tipo sal y pimienta ("salt and pepper"). También pueden atenuar detalles indeseables. Es posible aplicar el filtrado de forma reiterada pero limitada porque se corre el riesgo de atenuar o perder características importantes que dificulten o imposibiliten el reconocimiento[19].

La multiplicación escalar es común pero es sólo una posibilidad. Por ejemplo en:

$$\begin{bmatrix} = & = & = \\ = & T & = \\ X & X & X \end{bmatrix} \begin{bmatrix} X & = & = \\ X & T & = \\ X & = & = \end{bmatrix} \begin{bmatrix} X & X & X \\ = & T & = \\ = & = & = \end{bmatrix} \begin{bmatrix} = & = & X \\ = & T & X \\ = & = & X \end{bmatrix} \tag{2.3}$$

tenemos ejemplos de filtros con la leyenda:

T es el píxel objetivo.

X es un valor que no será tenido en cuenta.

$=$ es uno de los valores de la vecindad que, de ser todos iguales, su valor será asignado al píxel objetivo de salida. De lo contrario se le asigna al valor de salida el valor del píxel objetivo de entrada.

Con estas máscaras, que son versiones rotadas entre sí, pueden rellenar o suprimir hueco y ruido de dimensión 1 píxel.

Una mención especial merece el suavizado gaussiano cuyo núcleo es una gaussiana 2-D de media 0 y desviación estándar σ :

$$G(i,j) = e^{\frac{-(i^2+j^2)}{2\sigma^2}} \tag{2.4}$$

Para MONICOD el problema es el consumo de tiempo que supone aplicar una convolución. Ver al respecto de esto en este mismo documento 9.3. Es por esta razón por lo que no profundizaremos en esta área.

2.4.2. Realzado

El realzado trata de acentuar detalles subyacentes en la imagen original, pero que no se aprecian con la suficiente nitidez. Este problema está presente en el dominio de MONICOD como quedará patente en los siguientes capítulos.

Los métodos disponibles para mejorar la visibilidad de la imagen son numerosos. No existe, desgraciadamente, una teoría general para determinar como de buena es una mejora particular a la imagen en términos de percepción humana[10]([20]). Sin embargo cuando el realzado es una etapa de preproceso para otros procesamientos de imágenes medidas cuantitativas pueden decidir cuando es mejor.

La mayoría pueden clasificarse en dos categorías atendiendo a su naturaleza matemática

- Métodos en el dominio del espacio.
  - Se aplican operaciones directas en los píxeles de la imagen.
- Métodos en el dominio de la frecuencia.
  - Se opera sobre la transformada de Fourier de la imagen.

Otra clasificación más cualitativa ([21]) divide los métodos en:

[10] Si la calidad parece buena, ¡es buena!

- Técnicas basadas en intensidad.
  - Estas técnicas afectan a los niveles de intensidad de grises presentes en la imagen. Serán tratadas en este apartado.
- Técnicas basadas en características.
  - Estas técnicas tratan de realzar aspectos característicos de la imagen tales como los bordes de los objetos. En este documento estas técnicas serán presentadas como componentes de la segmentación.

La elección de una técnica depende de la tarea específica, contenido de la imagen, características del observador y condiciones de adquisición. En nuestro caso, además, juega un rol crítico el hecho de tratarse de una aplicación de tiempo real[22].

Las técnicas basadas en intensidad pueden expresarse en la forma:

$$Io(x,y) = f(I(x,y)) \tag{2.5}$$

donde I es la imagen original, f es alguna clase de transformación que afecta a las intensidades e Io es la imagen realzada.

En MONICOD, nos basaremos en la operación clásica de histograma de la imagen para conseguir realce. Ver A.3.4 para una discusión sobre histogramas. El histograma tiene la ventaja de ser una operación particularmente rápida lo que la hace muy conveniente. MONICOD computa histogramas sólo de una parte de la imagen -El área de interés- pero lo que sigue es igualmente aplicable.

2.4.2.1. Contracción del histograma

La contracción del histograma construye una nueva imagen $g(i,j)$ de salida a partir de la imagen de entrada $f(i,j)$ utilizando el histograma para redistribuir los niveles de gris entre los valores C_{Max} y C_{Min} deseados.

$$g(i,j) = \left[\frac{C_{Max} - C_{Min}}{f(i,j)_{Max} - f(i,j)_{Min}}\right] \times [f(i,j) - f(i,j)_{Min}] + C_{Min} \tag{2.6}$$

$f(i,j)_{Max}$ y $f(i,j)_{Min}$ son los niveles de gris máximo y mínimo presentes en la imagen de entrada. Esta técnica resulta contraproducente para el contraste porque lo disminuye, sin embargo resulta interesante presentarla de cara a la técnica inversa (la expansión del histograma)

2.4.2.2. Expansión del histograma

Esta operación es opuesta a la contracción del histograma. Expande el histograma resultando muy efectiva para incrementar el contraste de una imagen. En este caso se construye una nueva imagen $g(i,j)$ de salida a partir de la imagen de entrada $f(i,j)$ utilizando el histograma para redistribuir los niveles de

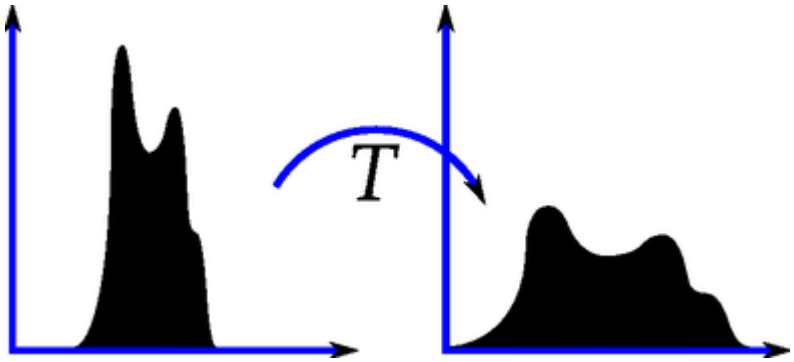


Figura 2.10: El principio del ecualizado del histograma.

gris entre los valores Max y Min siendo estos los niveles máximos y mínimos posibles de gris. En nuestra notación estos son $L-1$ y 0.

$$g(i,j) = \left[\frac{f(i,j) - f(i,j)_{Min}}{f(i,j)_{Max} - f(i,j)_{Min}}\right] \times [Max - Min] + Min \qquad (2.7)$$

2.4.2.3. Ecualizado del histograma

El ecualizado del histograma es un técnica muy popular([21, 23, 24]) capaz de revelar detalles ocultos en imágenes con pocos contrastes. Intuitivamente la idea es redistribuir los niveles de grises en el histograma de manera que se "aprovechen" todos los niveles de gris disponibles. Ver 2.10. Como resultado el contraste se incrementa. Esta técnica es muy rápida con la ayuda de tablas LUT[11]. La razón para querer obtener un histograma plano está motivada por la teoría de la información donde se sabe que función de densidad de probabilidad uniforme contiene la mayor cantidad de información([25]).

Como se ha dicho, el propósito es encontrar una función $F(g)$ que realce el contraste general en la imagen original expandiendo la distribución de niveles de gris. Dicha expansión debe ser lo más suave posible en el sentido que debería haber el mismo número de píxeles por niveles de gris.

Dado que el número de píxeles en una dimensión $N \times M$ es precisamente $N \times M$ y el número de niveles de gris sobre el cual se va a a realizar la expansión es N_g, el histograma ideal sería plano, con el mismo número de píxeles en cada nivel de gris esto es, el número ideal de píxeles en cada nivel de gris corresponde a $\frac{N \times M}{N_g}$.

Por otra parte podemos definir la función de probabilidad a partir del histograma . El procedimiento es el que sigue([26]):

1. Suponiendo una imagen con n niveles de gris en el rango $[0, L-1]$ se cumple que:

[11] Look Up Table.Ver entrada en glosario.

$$\sum_{g=0}^{g=L-1} N(g) = N \times M \tag{2.8}$$

2. La probabilidad por cada nivel de gris g viene dada por:

$$p(g) = \frac{N(g)}{N \times M}; g = 0, ..., L-1 \tag{2.9}$$

3. De lo anterior se deduce fácilmente que:

$$\sum_{g=0}^{g=L-1} p(g) = 1 \tag{2.10}$$

4. La función de densidad de probabilidad resulta ser:

$$P_x(x) \cong \sum_{g=0}^{g=x} p(g) \tag{2.11}$$

5. A partir de aquí lo que requerimos es hacer una transformación entre funciones de densidad de probabilidad $P_x(x)$ y $P_y(y)$, imponiendo la condición de que la función de transformación sea monótona creciente. Así para cada valor de x e y se cumple que:

$$\int_0^{F(g)} P_y(y)dy = \int_0^{g} P_x(x)dx = \sum_{g=0}^{g=x} p(g) \tag{2.12}$$

6. A partir de aquí hay varios ecualizados posibles. La ecualización uniforme y la ecualización exponencial son de uso común. MONICOD utiliza ecualización uniforme. Para ella la función de distribución deseada tiene la forma:

$$P_y(y) = \frac{1}{L}\ 0 \leq y \leq (L-1) \tag{2.13}$$

7. Aplicando la expresión 2.12:

$$\int_0^{F(g)} P_y(y)dy = \int_0^{F(g)} \frac{1}{L}dy = \frac{1}{L}\int_0^{F(g)} dy = \frac{1}{L}F(g) \tag{2.14}$$

$$\int_0^{g} P_x(x)dx = \sum_{g=0}^{g=x} p(g) \tag{2.15}$$

$$\frac{1}{L}F(g) = \sum_{g=0}^{g=x} p(g) \tag{2.16}$$

$$F(g) = L\sum_{g=0}^{g} p(g) \tag{2.17}$$

Aunque el ecualizado es una técnica simple y efectiva de mejora de imagen produce usualmente demasiado contraste resultando en imágenes poco naturales incluso con artefactos visuales. En [27] se presenta un estudio de variantes para el ecualizado(HE) tales como BBHE[12], QBHE[13] , DSIHE[14] , MMBEBHE[15] , RMSHE[16], BPDHE[17] y WMSHE[18].

Una ventaja importante del ecualizado del histograma global (empleado en MONICOD) sobre otras técnicas es que es completamente automático[22].

Los resultados sobre las imágenes tratadas por MONICOD puede apreciarse en 7.2.2.

2.4.3. Inclinación de texto y carácter

En general los métodos de reconocimiento de caracteres toleran mal la inclinación de texto. Suele combatirse en dos etapas: Detección de la inclinación y corrección de la inclinación.

Detección de la inclinación Quizás los métodos más representativos para la detección de inclinación sean:

- Búsqueda de caracteres/ fragmentos de carácter, y calcular el ángulo promedio conectando sus centroides. Esta estrategia es sensible al ruido: Falsos caracteres o fragmentos de carácter pueden desviar el ángulo de inclinación.
- Empleo de análisis de proyecciones. Con esta técnica el texto es proyectado con diferentes ángulos, y las varianzas del número de píxeles por linea proyectada es computado. Es de esperar que la máxima varianza corresponda a la proyección con el ángulo al que corresponde al texto. La racionalización de este método es sencilla: Las líneas de proyección con el ángulo adecuado atraviesa más texto o más líneas que ninguna otra. Ver figura 2.11.

[12] Brightness preserving Bi-Histogram Equalization.
[13] Quantized Bi-Histogram Equalization.
[14] Dual Sub-Image Histogram Equalization.
[15] Minimum Mean Brightness Error Bi-HE.
[16] Recursive Mean-Separate Histogram Equalization.
[17] Brightness preserving dynamic histogram equalization.
[18] Weighting mean-separated sub-histogram equalization.

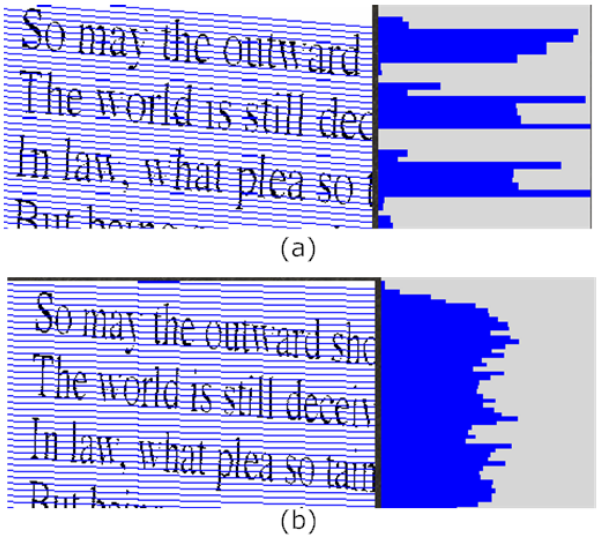


Figura 2.11: Estudio de proyecciones extraído de `http://www.eecs.berkeley.edu/~fateman/kathey/skew.html`. El ángulo de proyección en (a) está más próximo del ángulo de proyección de (b) a la inclinación real que presenta el texto.

Corrección de la inclinación Ahora hay que rotar el texto el ángulo detectado para corregir la inclinación. Esto puede hacerse aplicando a cada píxel que comprenden la imagen la transformación:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.18}$$

donde θ es el ángulo de inclinación.

Trabajos más sofisticados de detección y corrección pueden hallarse por ejemplo en [28].

Letra cursiva Un tipo especial de inclinación es la letra cursiva (inclinación de carácter y no de texto). De nuevo se requiere algún tipo de corrección para reducir la variación en el carácter en el paso previo a la extracción de características. Una forma de hacer esto también comprende dos pasos: Estimación del ángulo de inclinación de la letra, y corrección de este. Ver sobre esto los dos trabajos recopilados [29] en que enfrentan el problema utilizando la transformada de Radon.

MONICOD En MONICOD se ha optado por ignorar esta dificultad porque se tiene control sobre la inclinación de texto durante la adquisición, y puede suprimir. Eso es interesante porque hace innecesario el preprocesamiento que deriva de este problema. Con todo en la sección 9.2 operamos en condiciones

de inclinación de texto obteniéndose resultados aceptables gracias al sistema de validación de caracteres MONICOD.

2.4.4. Normalización de caracteres

La normalización de caracteres es considerado el más importante preprocesamiento para reconocimiento de caracteres. La meta de la normalización es por un lado reducir la variabilidad dentro de los ejemplares de una misma clase y por otro tratar de garantizar que las clases se ajustan a la misma "norma". Esto es importante para facilitar la extracción de características y la calidad acierto de la clasificación. Existen aquí de nuevo dos aproximaciones: Lineal y no lineal.

Probablemente un método representativo y bastante conocido sea la normalización adaptativa de la relación de aspecto.

Dada la relación de aspecto original del carácter R_1:

$$R_1 = \frac{min(W_1, H_1)}{max(W_1, H_1)} \tag{2.19}$$

el método la transforma en una nueva relación de aspecto R_2:

$$R_2 = \frac{min(W_2, H_2)}{max(W_2, H_2)} \tag{2.20}$$

usando una función asociativa como las presentadas en el cuadro 2.1.

W_1 y H_1 representan al ancho y el alto de la región y W_2 y H_2 el ancho y el alto del plano estándar.

La posición de cada píxel en la región original se ubica en una nueva posición en el plano estándar usando una función de conversión de coordenadas como las descritas en el cuadro 2.2. En dicho cuadro:

$$\alpha = \frac{W_2}{W_1} \tag{2.21}$$

$$\beta = \frac{H_2}{H_1} \tag{2.22}$$

Para la conversión basada en momentos tenemos que x_c y y_c denota el centro de gravedad del carácter original que puede calcularse como:

$$x_c = \frac{m_{10}}{m_{00}} \tag{2.23}$$

$$y_c = \frac{m_{01}}{m_{00}} \tag{2.24}$$

Ver más sobre momentos en 2.6.2.1.

$$x'_c = \frac{W_2}{2} \tag{2.25}$$

$$y'_c = \frac{H_2}{2} \tag{2.26}$$

Método	Función
Relación de aspecto constante	$R_2 = 1$
Relación de aspecto conservador	$R_2 = R_1$
Raíz cuadrada de la relación de aspecto original	$R_2 = \sqrt{R_1}$
Raíz cuadrada de la relación de aspecto original	$R_2 = \sqrt[3]{R_1}$
Seno de la relación de aspecto original	$R_2 = \sqrt[3]{sin\frac{\pi}{2}R_1}$

Cuadro 2.1: Transformaciones de la relación de aspecto

Método	Mapeo
Lineal	$x = \alpha x'$ $y = \beta y'$
En base a momentos	$x = \alpha(x - x_c) + x'_c$ $y = \beta(y - y_c) + y'_c$
No lineal	$x' = W_2 h_x(x)$ $y' = H_2 h_y(y)$

Cuadro 2.2: Funciones de conversión de coordenadas para normalizar una región.

Por último para la normalización no lineal (que ha demostrado considerable éxito al tratar con caracteres asiáticos tales como kanjis) se ecualizan las proyecciones horizontal y vertical de la imagen de carácter. Denotamos las funciones de densidad vertical de la imagen $f(x, y)$ como $d_x(x, y)$ y $d_y(x, y)$. Las proyecciones en el eje horizontal y vertical son calculadas como:

$$p_x(x) = \frac{\sum_x d_x(x, y)}{\sum_x \sum_y d_x(x, y)} \tag{2.27}$$

$$p_y(x) = \frac{\sum_y d_y(x, y)}{\sum_x \sum_y d_x(x, y)} \tag{2.28}$$

Las proyecciones normalizadas, también vistas como histogramas, son acumuladas para generar las funciones de conversión de coordenadas:

$$h_x(x) = \sum_{u=0}^{x} p_x(u) \tag{2.29}$$

$$h_y(y) = \sum_{u=0}^{y} p_y(u) \tag{2.30}$$

MONICOD En MONICOD no se aplicará proceso de normalización de caracteres porque se trabaja con la presunción de que los caracteres tratados durante la jornada cumplen especificaciones de homogeneidad constante (por ejemplo en escalado), al menos durante un periodo de varias horas. Para el sistema de clasificación que MONICOD está usando en la redacción del presente documento la normalización del carácter(TM) no debería jugar un papel decisivo.

2.5. Segmentación

La idea de segmentación, aplicada al tratamiento de imágenes, se apoya en que el espacio de entrada (la imagen) es susceptible de ser descompuesta en elementos que tengan una significación relevante por si mismos siendo susceptibles a análisis por separado. Dichos elementos son más complejos que los píxeles (la unidad mínima de información en la imagen) pero más simples de tratar (o eso se espera) e interpretar que la imagen completa.

2.5.1. Definiendo la segmentación

La segmentación es una operación ampliamente extendida en el reconocimiento de formas. Podemos considerarla como una partición de una imagen en un conjunto de regiones que no se solapan y en las que resulta deseable que obedezcan algunas reglas (ver [30]):

- Deben ser uniformes y homogéneas con respecto a algunas características.
- Sus interiores deberían ser simples y sin demasiados agujeros/huecos.
- Regiones adyacentes deberían tener diferencias significativas en los valores con respecto a las características en las cuales son uniformes.
- Fronteras de cada segmento deberían ser simples, no confusas y aproximadas espacialmente.

Quizás la formalización más conocida sea la presentada en [31]. En ella, la imagen I queda descrita como una colección de píxeles (1 píxel: 1 valor de intensidad) localizado sobre un dominio espacial bidimensional R. Dicho dominio o región R está constituido por una o más subregiones $R_1, R_2.., R_n$. Cada región R_i se caracteriza porque los píxeles que la conforman comparten una cierta propiedad.

En dicho dominio definiremos un predicado P que se aplicará sobre un conjunto de píxeles Y. Dicho predicado cumplirá que:

- $P(Y)$ tendrá necesariamente los valores VERDADERO o FALSO (mutuamente excluyentes). El valor en cuestión dependerá sólo de las propiedades de la región o segmento de I.
- Siempre que es Z sea un subconjunto no nulo de Y ($Z \subseteq Y$) y se cumpla que $P(Y) = VERDADERO$ entonces también se cumplirá que $P(Z) = VERDADERO$. Este es el principio de uniformidad.
- Con la ayuda de P construimos la definición formal de segmentación de una imagen:

$$R = \bigcup_{i=1,2...n} R_i \tag{2.31}$$

$$R_i \text{ es una región conectada, } i = 1, 2, ..., n \tag{2.32}$$

$$R_i \bigcap R_j = 0 \tag{2.33}$$

$$P(R_i) = VERDADERO \ \forall i \tag{2.34}$$

$$P(R_i \bigcup R_j) = FALSO \ \forall i, j \ adjacentes \ donde \ i \neq j \tag{2.35}$$

La condición 2.31 indica que la unión de todas las regiones resultado de la segmentación debe formar la imagen completa.

La condición 2.32 señala que los puntos que conforman una región están conectados entre sí por medio de alguna característica que define la propia segmentación. Ver A.3.2.3.

La condición 2.33 implica que las regiones son conjuntos disjuntos, esto es, no comparten puntos.

La condición 2.34 establece que los puntos de una región satisfacen una o varias propiedades y por tanto comparten alguna característica, y así guardan cierta homogeneidad.

La condición 2.35 dispone que puntos de dos regiones distintas satisfacen propiedades distintas.

Los métodos con que los píxeles de una imagen son asociados a la región que corresponden son variados y constituyen soluciones en el campo de la segmentación. Sin embargo, un principio inherente de la segmentación es que NO existe un procedimiento de segmentación que sea al mismo tiempo ideal y universal. Hay muchas posibles particiones R que cumplan los principios citados para una misma imagen. En realidad incluso imágenes triviales pueden ser segmentadas de diferentes formas sin que ninguna partición sea netamente superior a otra, salvo consideraciones sobre el uso final que queremos darle. La calidad del resultado de la segmentación siempre depende del resultado que se aspira a obtener. Y este depende del campo de aplicación.

Existe una extensa bibliografía que trata con el problema de la segmentación. No hay teoría en segmentación de imágenes. En lugar de eso las técnicas de segmentación de imágenes son básicamente *ad hoc* y difieren en la manera en la que enfatizan una o más de las propiedades deseadas en un segmentador ideal y el compromiso y balance de unas frente a otras [32].

[33] constata que, hasta el año 2006, se habían propuesto sobre cuatro mil algoritmos de segmentación.

Basándonos en el ya clásico estudio de [34] clasificaremos en este documento las técnicas de segmentación en:

- Métodos de histogramas, umbralización y clustering(una multidimensional extensión de la umbralización)
- Métodos basados en regiones
- Métodos basados en contornos y bordes
- Métodos basados en transformada de Watershed.

El problema con esta clasificación es que el umbralizado es un caso particular de aproximación basada en regiones. Una clasificación alternativa puede encontrarse en [35] que divide los algoritmos de segmentación en seis grupos:

1. Umbralizado
2. Clasificación de píxeles
3. Segmentación de imágenes por rango
4. Segmentación de imágenes por color
5. Detección de bordes
6. Métodos basados en teoría 'fuzzy' o difusa(incluyendo umbralizado difuso, clustering difuso y detección de bordes difusa).

Pero también aquí encontramos solapamiento entre grupos de técnicas. La segmentación de imágenes por rango o color enfatiza cómo se lleva a cabo la segmentación pero pueden estar a su vez basados en umbralizado, detección de bordes o clasificación de píxeles).

Más recientemente [36] clasifica los métodos en:

1. Métodos basados en la forma del histograma (donde picos, valles curvaturas de la forma del histograma son analizados)
2. Métodos de clustering(donde las muestras de niveles de grises son agrupadas en dos partes como fondo y no-fondo)
3. Métodos basadas en entropía(donde la entropía de las regiones del fondo y no-fondo , la entropía cruzada de la imagen original y segmentada, etcétera son calculadas)
4. Métodos basados en atributos del objeto (donde una medida de la similaridad entre los niveles de grises e imágenes segmentadas, tales como forma, bordas, número de objetos son investigadas)
5. Métodos basados en la relación espacial(empleando distribuciones de probabilidad de un alto ordem y/o correlación entre píxeles)

Un estudio exhaustivo de las posibles formas de clasificación de los métodos de segmentación puede encontrarse en [37, 33].

2.5.2. Histogramas y umbralización

Las técnicas de umbralización se revelan efectivas cuando los objetos a segmentar y el fondo tienen una superficie, textura o distribución de grises/colores uniforme/homogénea pero diferente entre sí. Este tipo de imágenes se revelan en el histograma cuando se reconocen dos picos claramente, de ahí que se les denomine "imágenes bimodales". En este escenario ideal, podemos valernos de

esta característica en lugar de la forma de los objetos a segmentar como criterio de separabilidad. La idea es buscar una frontera o umbral que separe el fondo de los objetos a segmentar atendiendo atendiendo a la superficie, textura o distribución de grises/color. Dado que en el marco de los reconocedores de caracteres la cualidad mencionada es susceptible de cumplirse parece adecuado extendernos en la exposición de estos métodos.

Al aplicar un umbral T la imagen en escala de grises$f(x)$ puede etiquetarse píxel a píxel, tal como ocurre en una operación de binarizado.

$$g(x,y) = \begin{cases} 1 \Leftrightarrow f(x,y) > T \\ 0 \Leftrightarrow f(x,y) \leqslant T \end{cases} \tag{2.36}$$

o bien:

$$g(x,y) = \begin{cases} 1 \Leftrightarrow f(x,y) < T \\ 0 \Leftrightarrow f(x,y) \geqslant T \end{cases} \tag{2.37}$$

El problema es que varios factores como histogramas planos(imágenes no suficientemente bimodales, ruido no estacionario o correlacionado, iluminación ambiente, contraste inadecuado, violaciones locales en la separabilidad descrita entre regiones del fondo o de los objetos a segmentar) hacen que la elección del umbral sea compleja. Estas situaciones son comunes en imágenes reales. De ahí la presencia de una amplia variedad de algoritmos de umbralizado.

Habitualmente la selección del umbral puede depender de una o varias características:

$$T = T(f(x,y), p(x,y), x, y) \tag{2.38}$$

- $f(x,y)$. Si sólo depende de $f(x)$, el umbral es global.
- Alguna propiedad local del píxel $p(x,y)$ como por ejemplo el valor medio de sus vecinos. El umbral será local.
- La posición (x,y) del píxel. El umbral será dinámico.

Según [38] pueden distinguirse seis categorías de algoritmos de umbralizado:

1. Métodos basados en las propiedades de la forma del histograma donde, por ejemplo, picos, valles y curvaturas son examinados.

   Cabe destacar el método de Rosenfeld([39]) que analiza la "colina convexa" del histograma calculando las concavidades más profundas, llegando a ser estas candidatas a umbral.

   Sezan([40]) analiza picos y valles del histograma convolucionando el histograma con máscaras de diferenciado y suavizado. Ajustando la apertura de suavizado se fusionan picos y el histograma se reduce a una función de dos lóbulos. La operación de diferenciado extrae donde los picos empiezan, terminan y alcanzan su máximo.

Ramesh([41]) usan una aproximación funcional al histograma. Para ello minimizan la suma de errores cuadrados entre una función de dos niveles y el histograma o, alternativamente, minimizan la varianza del histograma aproximado. El umbral optimo ($Topt$) se obtiene por una búsqueda iterativa.

2. Agrupamiento/clustering en el espacio de medida donde las muestras de nivel de grises son agrupadas en dos partes correspondientes al fondo o al objeto a segmentar, o alternativamente son modelados como la mezcla de dos gaussianas. Dada su importancia en MONICOD dos ejemplos representativos de esta categoría son examinados en detalle en 2.5.2.2 y 2.5.2.3.

3. Información de la entropía del histograma que usan la entropía del fondo y del objeto, la entropía cruzada entre el original y la imagen binarizada, etcétera

4. Métodos basados en atributos del objeto donde se busca una medida de la similaridad entre el nivel de gris y las imagen binarizada, tales como coincidencias en los bordes del objeto, similaridades en las borrosidades de la forma,...

   Así Tsai([42]) utiliza los momentos para determinar la mejor umbralización posible entre la imagen original y la imagen umbralizada.

   Hertz([43]) usa una técnica de umbralización múltiple donde se extraen los bordes de la imagen original (utilizando el operador de Sobel, ver a este respecto 2.5.4). en escala de grises y se comparan con una imagen binarizada. El umbral que maximice las coincidencias entre ambos bordes debe ser el mejor.

   Murthy([44]) propone un umbralizado con un umbral borroso. Huang([45]) propone un factor fuzzy o borrosidad que se obtiene midiendo la similitud entre la imagen(en escala de grises) y la imagen binarizada. El propósito es minimizar el factor de borrosidad en términos de medias o medianas de fondo y objeto. Ramar([46]) ha evaluado varias medidas fuzzy (índice lineal y cuadrático de borrosidad, entropía logarítmica y entropía exponencial) concluyendo que el índice lineal de borrosidad es el mejor.

   Este tipo de técnicas son empleadas sobre todo en la restauración de documentos o sistemas no-destructivos de segmentación como la segmentación de imágenes de rayos-x, ultrasonidos o endoscópicas.

5. Métodos de información espacial basados en distribuciones de probabilidad de orden elevado y/o correlación entre píxeles

6. Métodos basados en características locales que adaptan el valor de umbral de cada píxel dependiendo de las características cercanas al píxel en la imagen(umbralizaciones adaptativas). Una selección de estas técnicas es presentada en 2.5.2.1.

El citado [38] lleva a cabo un estudio comparativo, con la calidad de la segmentación como único criterio, de estas seis categorías aplicadas al análisis de dos tipos de imágenes: Imágenes de documentos o procedentes de alguna modalidad de testeo no destructivo/invasivo (radiografías, imágenes de ultrasonidos, imágenes térmicas, etcétera...). En el citado estudio parece confirmarse que ningún algoritmo por si solo alcanza la excelencia en la calidad para cualquier tipo de imagen. No se considera una iluminación compleja o restricciones en el tiempo de computo. Circunstancias presentes en MONICOD.

2.5.2.1. Una comparativa de métodos de Umbralizado

Una aproximación para determinar cual es el mejor sistema de umbralizado para un problema es probar todos los disponibles con una imagen característica del problema y compararlos de forma cualitativa.

Dada la imágenes de entrada de la figura 2.12 aplicaremos sobre ellas métodos de umbralizado global y local[19].

Métodos de umbralizado global Los métodos de umbralizado global proporcionan un umbral global que se aplica a toda la imagen o área de interés. Nos centraremos aquí en métodos no paramétricos y que no precisen de información del contexto. En la figura 2.13 y 2.14 se comparan cualitativamente 16 métodos aplicados sobre una lata de hojalata y aluminio respectivamente:

- Variación de IsoData([47])
- Huang([48])
- Intermodes([49])
- IsoData([47])
- Li([50, 51])
- MaxEntropy([52])
- Mean([53])
- MinError(variación iterativa del método de Kittler-Illingworth)([54])
- Minimum([49])
- Moments([42])
- Otsu([55])
- Percentile([56])

[19] Y la impresión en papel agrava este problema. Sin embargo, por fidelidad se ha decidido mantenerlas en su forma original y no aplicarles ningún tratamiento de mejora como en otros casos.

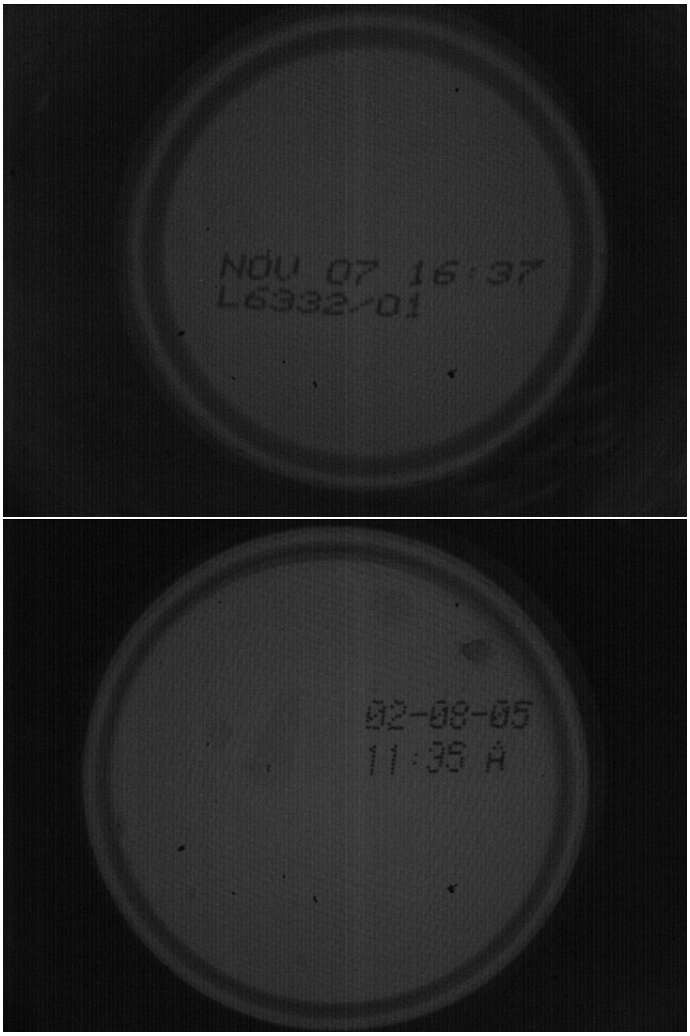


Figura 2.12: Figuras de entrada para umbralizado. Arriba hojalata. Abajo aluminio. Se trata de imágenes bastante oscuras.

Figura 2.13: Umbrales globales para hojalata. De izquierda a derecha y de arriba a abajo: Variación de IsoData, Huang, Intermodes, IsoData, Li, MaxEntropy, Mean, MinError(variación iterativa del método de Kittler-Illingworth), Minimum, Moments, Otsu, Percentile, RenyiEntropy, Shanbhagm, Triangle y Yen. Imagen generada con ImageJ([60])

- RenyiEntropy(similar a MaxEntropy pero usando entropía de Renyi)([52])
- Shanbhagm([57])
- Triangle([58])
- Yen([59])

El método de Otsu y el método de Kittler-Illingworth son examinados con más detalle en 2.5.2.2 y en 2.5.2.3 en atención a su popularidad y éxito.

Métodos de umbralizado local Los métodos de umbralizado local proporcionan umbrales locales que son calculados a partir de los valores de la vecindad o ventana local. A modo de ejemplo aquí contemplamos los siguientes métodos locales:

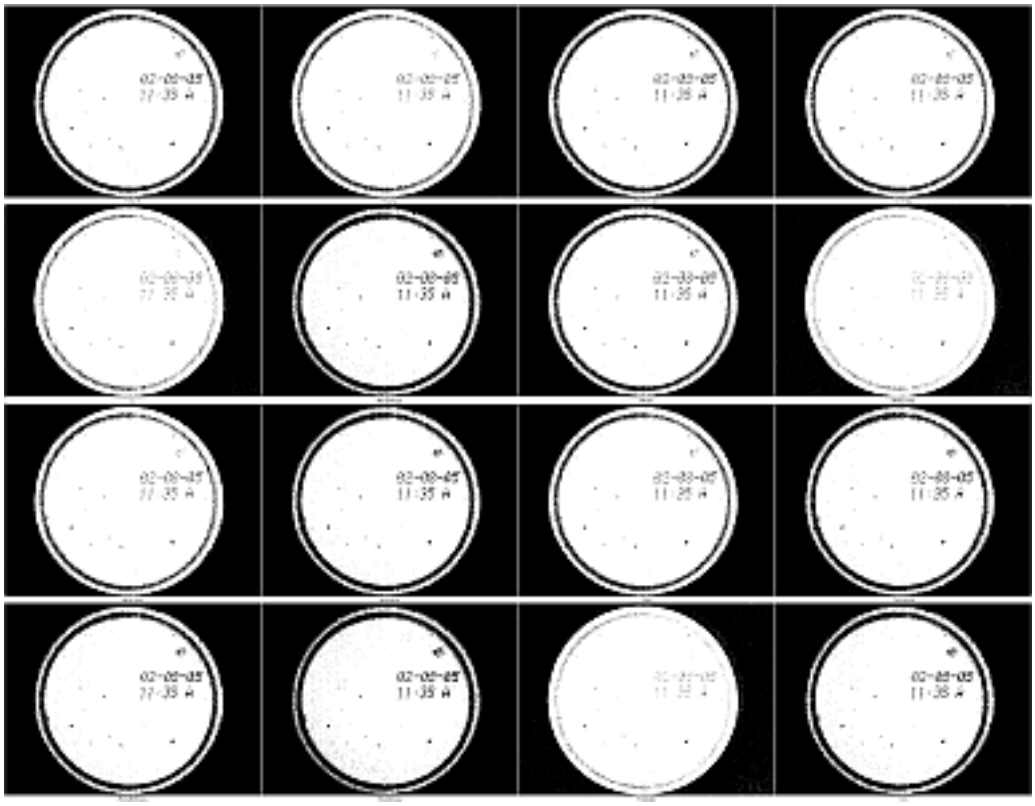


Figura 2.14: Umbrales globales para aluminio. De izquierda a derecha y de arriba a abajo: Variación de IsoData, Huang, intermodes, IsoData, Li, MaxEntropy, Mean, MinError(variación iterativa del método de Kittler-Illingworth), Minimum, Moments, Otsu, Percentile, RenyiEntropy, Shanbhagm, Triangle y Yen. Imagen generada con ImageJ([60])

Algoritmo 2.1 Algoritmo de Bernsen aplicado sobre la ventana local

si *contrastelocal* es menor o igual que *contrasteumbral* **entonces**
  si *grismedio* es menor o igual que 128 **entonces**
    $pixel \leftarrow objeto$
  sino
    $pixel \leftarrow fondo$
  fin si
sino
  si *pixel* es menor o igual que *grismedio* **entonces**
    $pixel \leftarrow objeto$
  sino
    $pixel \leftarrow fondo$
  fin si
fin si

Algoritmo 2.2 Algoritmo de la Media aplicado sobre la ventana local.

si *pixel* es menor o igual que (*media* - C) **entonces**
  $pixel \leftarrow objeto$
sino
  $pixel \leftarrow fondo$
fin si

Bernsen Discutido en [61], el proceso que tiene lugar en la ventana local corresponde al algoritmo 2.1 donde *contrastelocal* es un parámetro y *grismedio* corresponde a la expresión 2.39.

$$grisMedio = \frac{grisMax + grisMin}{2} \tag{2.39}$$

Media Se toma como umbral local la media aritmética de los valores de la ventana y se aplica el algoritmo 2.2.

Mediana Se toma como umbral local la mediana de valores de la ventana y se aplica el algoritmo 2.3 . Es especialmente intensivo porque exige ordenar los valores (de menor a mayor o de mayor a menor), operación costosa en cómputo.

Algoritmo 2.3 Algoritmo de la *Mediana* aplicado sobre la ventana local.

si *pixel* es menor o igual que (*mediana* - C) **entonces**
  $pixel \leftarrow objeto$
sino
  $pixel \leftarrow fondo$
fin si

Algoritmo 2.4 Algoritmo del *gris medio* aplicado sobre la ventana local.

si *pixel* es menor o igual que (*grismedio* - C) **entonces**
    $pixel \leftarrow objeto$
sino
    $pixel \leftarrow fondo$
fin si

Algoritmo 2.5 Algoritmo de NiBlack aplicado sobre la ventana local.

si *pixel* es menor o igual que (*media*+k**desviacionestandar* - C) **entonces**
    $pixel \leftarrow objeto$
sino
    $pixel \leftarrow fondo$
fin si

Gris Medio Se toma como umbral local el valor de gris medio de valores de la ventana y se aplica el algoritmo 2.4 . Este valor de gris medio se calcula como en 2.39.

NiBlack Discutido en [62]. Se toma como umbral local y se aplica el algoritmo 2.5.

Sauvola Discutido en [63]. Es una variación del método de NiBlack. Se toma como umbral local y se aplica el algoritmo2.6.

En las figuras 2.15 y 2.16 se aprecian los resultados de estos métodos sobre una lata de hojalata y aluminio respectivamente.

2.5.2.2. Método de Otsu

El método de Otsu[55] correspondería a un método de agrupamiento/clustering y es uno de los métodos de umbralizado más referenciados, considerado casi un estándar. Durante bastante tiempo fue evaluado como el mejor y más rápido en la búsqueda de un umbral global[64, 65]. Ver también a este respecto la comparativa de Sahoo[66]. Proporciona resultados satisfactorios cuando el número de píxeles de cada clase es similar. En el método, los umbrales se obtienen de una forma óptima de acuerdo al criterio de maximizar la variancia[20] entre clases

[20] La variancia es una medida de la dispersión de valores, en este caso de niveles de gris.

Algoritmo 2.6 Algoritmo de Sauvola aplicado sobre la ventana local.

si *pixel* es menor o igual que 1+k*(*desviacionestandar*/r-1) **entonces**
    $pixel \leftarrow objeto$
sino
    $pixel \leftarrow fondo$
fin si

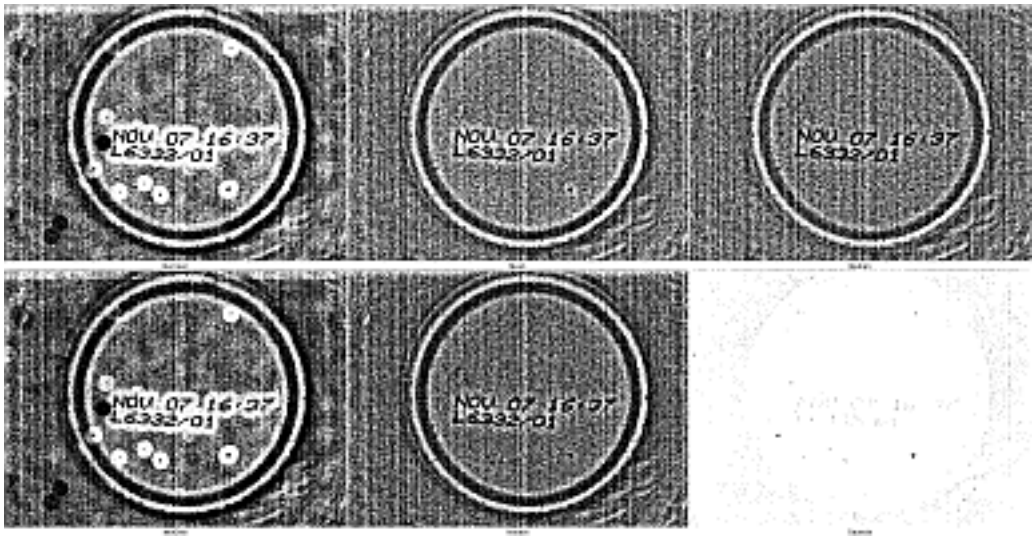


Figura 2.15: Umbrales locales para hojalata con ventana circular de radio 15. De izquierda a derecha y de arriba a abajo: Bernsen, Mean, Median, MidGrey, Niblack y Sauvola. Imagen generada con ImageJ([60])

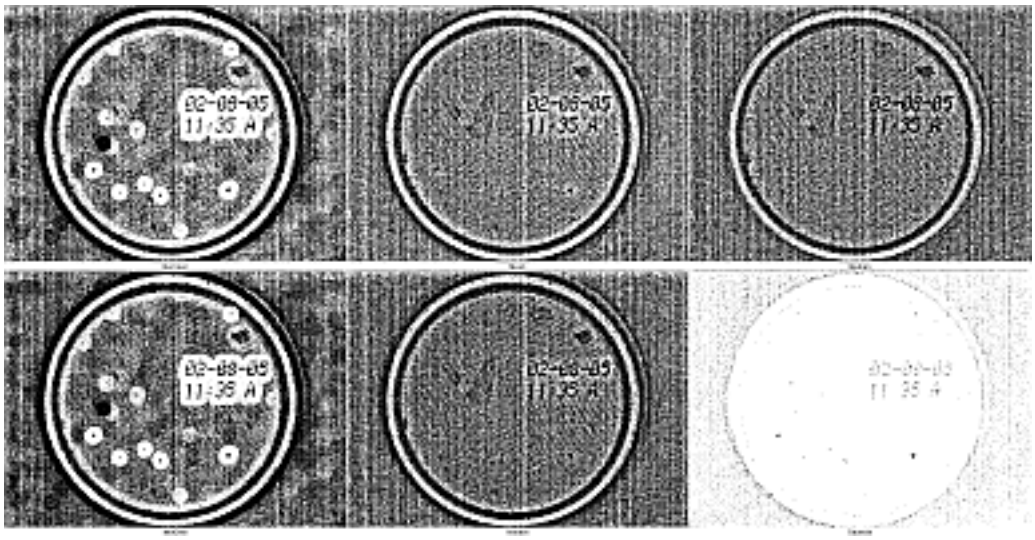


Figura 2.16: Umbrales locales para aluminio con ventana circular de radio 15. De izquierda a derecha y de arriba a abajo: Bernsen, Mean, Median, MidGrey, Niblack y Sauvola. Imagen generada con ImageJ([60])

mediante una búsqueda exhaustiva.

Sin perder generalidad supondremos objetos oscuros (caracteres) contra un fondo mas claro. La operación de umbralizado corresponde a particionar los píxeles de una imagen con $G : [1, ..., L]$ niveles de grises en dos clases $C_0 : [1, ..., k]$ y $C_1 : [k+1, ..., L]$ siendo el umbral k. El Método de Otsu busca el umbral óptimo k^* minimizando el error de clasificar un píxel del fondo como píxel del objeto o viceversa.

Para ello conocidas las probabilidad de ocurrencia definidas en A.2 partimos de la distribución de probabilidad para ambas clases:

$$C_0 : \frac{p_1}{w_1(t)}, ..., \frac{p_t}{w_1(t)} \tag{2.40}$$

$$C_1 : \frac{p_1}{w_1(t)}, ..., \frac{p_t}{w_1(t)} \tag{2.41}$$

donde:

$$w_0 = Pr(C_0) = \sum_{i=1}^{k} p_i = w(k) \tag{2.42}$$

$$w_1 = Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - w(k) \tag{2.43}$$

Debe resultar evidente que se cumple:

$$w_1 + w_2 = 1 \tag{2.44}$$

El cómputo de las variancias se obtiene con:

$$\mu_0 = \sum_{i=1}^{k} \frac{ip_i}{w_0} = \frac{\mu(k)}{w(k)} \tag{2.45}$$

$$\mu_1 = \sum_{i=k+1}^{L} \frac{ip_i}{w_1} = \frac{\mu_T - \mu(k)}{1 - w(k)} \tag{2.46}$$

donde:

$$\mu(k) = \sum_{i=1}^{k} ip_i \tag{2.47}$$

y:

$$\mu_T = \mu(L) = \sum_{i=1}^{L} ip_i \tag{2.48}$$

Es fácil demostrar que:

$$w_0 \cdot \mu_0 + w_1 \cdot \mu_1 = \mu_T \tag{2.49}$$

Otsu definió la variancia entre clases de una imagen umbralizada como:

$$\sigma_B^2 = w_1 \cdot (\mu_1 - \mu_T)^2 + w_2 \cdot (\mu_2 - \mu_T)^2 \tag{2.50}$$

Para una umbralización de dos niveles, Otsu verificó que el umbral óptimo t^* se elige de manera que σ_B^2 sea máxima. Esto es:

$$k^* = arg\ max(\sigma_B^2)\ con\ 1 \leq k \leq L \tag{2.51}$$

Este valor umbral hace que la dispersión dentro de cada clase sea lo más pequeña posible, pero al mismo tiempo la dispersión sea la mayor posible entre clases diferentes.

El método puede extenderse fácilmente a múltiples umbrales. Asumiendo que hay $M-1$ umbrales, $\{k_1, k, ..., k_{M-1}\}$ los cuales dividen a la imagen en M clases $C_1 : [1, ..., k_1]$, $C_2 : [k_1 + 1, ..., k_2]$,..., $C_M : [k_{M-1}, ..., L]$, los umbrales óptimos $\{k_1^*, k_2^*, ..., k_{M-1}^*\}$ se eligen maximizando σ_B^2:

$$\{k_1^*, k_2^*, ..., k_{M-1}^*\} = arg\ max(\sigma_B^2)\ con\ 1 \leq k_1 < ... < k_{M-1} \leq L \tag{2.52}$$

2.5.2.3. Método de Kittler-Illingworth o mínimo error de clasificación

El método de Kittler-Illingworth[54] obtuvo el mejor resultado global en el citado Sezgin[38], y es similar al método de Otsu ya que se basa en el cálculo del umbral óptimo para las k funciones gaussianas que definen en el histograma cada uno de los objetos a segmentar y que vienen caracterizadas por sus medias μ_k y sus desviaciones típicas σ_k. Una vez más el umbralizado se convierte en un problema de clasificación entre dos o más clases. La noción que subyace es que, con frecuencia, resulta realista asumir que las poblaciones respectivas que corresponden a cada clase están distribuidas normalmente con distintas medias y desviaciones típicas. Como en Otsu se busca una función criterio a optimizar relacionada con el ratio de error promedio en la clasificación de los píxeles en clases.

Consideremos una imagen g en niveles de grises en el intervalo $[0, L]$. De nuevo tenemos un histograma $h(g)$ que proporciona la frecuencia de ocurrencia para cada nivel de gris de g. El histograma puede verse como una estimación de la función de densidad de probabilidad que comprime los niveles de grises de objetos y fondo. Suponer ahora que umbralizamos los datos de grises en cierto umbral T, y modelamos cada una de las dos poblaciones de píxeles resultantes se caracterizan como $h(g|i, T)$ con una densidad normal, con parámetros $\mu_i(T)$ y $\sigma_i(T)$, y una probabilidad *a priori* $P_i(T)$.

Así haciendo $a = \begin{cases} 0 & i = 1 \\ T+1 & i = 2 \end{cases}$ y $b = \begin{cases} T & i = 1 \\ L & i = 2 \end{cases}$, tenemos:

$$P_i(T) = \sum_{g=a}^{b} h(g) \tag{2.53}$$

$$\mu_i(T) = \frac{\left[\sum_{g=a}^{b} h(g)g\right]}{P_i(T)} \tag{2.54}$$

$$\sigma_i^2(T) = \frac{\left[\sum_{g=a}^{b} [g-\mu_i(T)]^2 h(g)\right]}{P_i(T)} \tag{2.55}$$

Ahora usando el modelo $h(g|i,T)$, la probabilidad condicional $e(g,T)$ de que un nivel de gris g sea reemplazada en la imagen por el valor binario correcto (como si de un binarizado se tratase) considerando $i = \begin{cases} 1 & g \leqslant T \\ 2 & g > T \end{cases}$.

$$e(g,T) = \frac{h(g|i,T)\cdot P_i(T)}{h(g)} \tag{2.56}$$

En el cociente sólo el numerador $h(g|i,T)\cdot P_i(T)$ depende de i y T. Desarrollando el logaritmo:

$$\begin{aligned}
\ln\left[h(g|i,T)\cdot P_i(T)\right] &= \ln\left[h(g|i,T)\right] + \ln\left[P_i(T)\right] = \ln\left(\frac{1}{\sqrt{2\pi}\sigma_i} e^{-\left(\frac{(g-\mu_i)^2}{2\sigma_i^2}\right)}\right) + \ln P_i(T) = \\
&= \ln\left(\frac{1}{\sqrt{2\pi}\sigma_i(T)}\right) + \ln\left(e^{-\left(\frac{(g-\mu_i)^2}{2\sigma_i^2}\right)}\right) + \ln P_i(T) = \\
&= ln(1) - \ln\left(\sqrt{2\pi}\sigma_i(T)\right) - \left(\frac{(g-\mu_i)^2}{2\sigma_i^2}\right) ln(e) + \ln P_i(T) = \\
&= -\ln\left(\sqrt{2\pi}\right) - \ln\sigma_i(T) - \left(\frac{(g-\mu_i)^2}{2\sigma_i^2}\right) + \ln P_i(T)
\end{aligned} \tag{2.57}$$

Multiplicando por -2 y suprimiendo el elemento constante $-\ln\left(\sqrt{2\pi}\right)$:

$$\varepsilon(g,T) = \left(\frac{(g-\mu_i)^2}{\sigma_i^2}\right) + 2\ln\sigma_i(T) - 2\ln P_i(T) \tag{2.58}$$

$$J(T) = \sum_{g} h(g)\cdot\varepsilon(g,T) \tag{2.59}$$

El umbral óptimo corresponderá al valor mínimo de J

$$J(\tau) = \min_{T} J(T) \tag{2.60}$$

Para calcular J desplegamos la expresión 2.59 con 2.58.

$$J(T) = \sum_{g=0}^{T} h(g) \times \left\{ \left[\frac{g - \mu_1(T)}{\sigma_1(T)} \right]^2 + 2\log\sigma_1(T) - 2\log P_1(T) \right\} + \\ + \sum_{g=T+1}^{L} h(g) \times \left\{ \left[\frac{g - \mu_2(T)}{\sigma_2(T)} \right]^2 + 2\log\sigma_2(T) - 2\log P_2(T) \right\} \tag{2.61}$$

Reemplazando en 2.61 la expresión 2.53 tenemos:

$$J(T) = 1 + 2\left[P_1(T)\log\sigma_1(T) + P_2(T)\log\sigma_2(T)\right] - 2\left[P_1(T)\log P_1(T) + P_2(T)\log P_2(T)\right] \tag{2.62}$$

2.5.3. Regiones

La segmentación se orienta a la búsqueda de similitudes sin necesidad explícita de métodos adicionales como la determinación de umbrales o la detección de bordes. Los esquemas más extendidos son el 'crecimiento de regiones' y la 'División y unión de regiones'. Ambos esquemas utilizan un predicado P que determina el nivel de homogeneidad de una región I (ver 2.5.1) . En otras palabras y en su forma más simple, el criterio de homogeneidad P que comparten los píxeles los cohesiona en una región I(ver 2.5.1). La diferencia estriba en la dirección de aplicación (crecimiento o división[21]).

Otras técnicas son más directas y se basan en el etiquetado de los píxeles de forma individualizada y aislada. Etiquetados todos los píxeles los agrupamos utilizando la propiedad de adyacencia. Es decir píxeles adyacentes con la misma etiqueta son agrupados.

2.5.3.1. Crecimiento de regiones

Este método es probablemente uno de los más populares para la segmentación. La idea es agrupar regiones progresivamente según algún criterio hasta que ya no sea posible. Ese agrupamiento será una región a su vez. La operación fundamental es la adición de regiones. En su forma más simple las regiones son píxeles.

Empezamos partiendo de un conjunto inicial de píxeles seleccionados mediante algún procedimiento. A estos se les llama semillas. Para gestar una región debemos contar inicialmente al menos con una semilla para esa región. De otra manera esa región permanecerá oculta.

Para cada semilla se da pie a una región y a un procedimiento de crecimiento. Esto es, buscamos píxeles adyacentes que tengan propiedades similares a la

[21] Aunque el método de 'división y unión' también contempla el crecimiento de regiones

semilla (por ejemplo la luminancia), que son agregados a la región de la semilla. En la determinación de las propiedades es esencial el conocimiento previo sobre el dominio. Repetimos el proceso con los píxeles de cada región y píxeles externos adyacentes. Si la similitud es suficiente son agregados a la región. Cuando no hay nuevas adiciones el proceso se detiene.

Los problemas de esta técnica tienen que ver con la elección de semillas. Habitualmente habrá que utilizar nuestro conocimiento de la escena para aplicar alguna heurística que haga la selección.

2.5.3.2. División y unión de regiones

En este caso se trata de un proceso de subdivisión. Se comprueba si cada región disponible $I_1, I_2, ...I_n$, si es suficientemente homogénea. Si no lo es se lleva a cabo una partición regular y predeterminada sobre esa región (por ejemplo en cuatro cuadrantes) y las subregiones resultantes se añaden a la lista de regiones disponibles.

Además, puede ocurrir que durante el proceso de división dos regiones diferentes pero adyacentes sean suficientemente homogéneas respecto al predicado como P y se procede a unirlas.

Se repite el proceso hasta que no hayan nuevas particiones. Cuando esto ocurre las regiones resultantes son uniformes respecto al predicado P. Una división completa de la imagen puede ser representada como un nivel de un árbol (en el caso de los cuatro cuadrantes sería un árbol cuaternario)

Dos regiones suficientemente homogéneas son susceptibles de fusionarse.

El número de regiones finales será menor o igual a número de regiones iniciales porque se puede contemplar la posibilidad que regiones que corresponden a semillas diferentes se fusionen.

2.5.4. Contornos

Un contorno es definido como la frontera entre dos regiones con diferentes propiedades de nivel de gris[67]. Es posible detectar la frontera porque las propiedades de nivel de gris sufren un cambio drástico a ambos lados de esta. La segmentación se orienta a la búsqueda de estas discontinuidades.

2.5.4.1. Aproximación por gradiente

La aproximación por gradiente es fácil de intuir. La idea es encontrar cambios bruscos en las intensidades. Y para encontrarlos usamos la primera derivada que queda representada por el vector gradiente. El vector gradiente es definido como la variación de luminancia en magnitud y dirección.

Durante décadas se han descrito varios operadores basadas en el concepto de vector gradiente específicamente utilizados para la detección de bordes en imágenes. Los más famosos son las máscaras de gradiente de Roberts, Prewitt, Sobel y Frei-Chen.

Operador de Roberts Es un operador no lineal simple de calcular, ya que sólo usa una ventana (también llamada filtro, plantilla, máscara, kernel) de tamaño 2x2. Ver [68]

$$G_x = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}; G_y = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} \tag{2.63}$$

$$G(f(i,j)) = |f(i,j) - f(i+1,j+1)| + |f(i,j+1) - f(i+1,j)| \tag{2.64}$$

Ver figuras 2.17 y 2.18 para conocer los efectos de esta máscara sobre latas de aluminio y hojalata.

Operador de Prewitt, Sobel y Frei-Chen

$$\begin{vmatrix} A_0 & A_1 & A_2 \\ A_7 & (i,j) & A_3 \\ A_6 & A_5 & A_4 \end{vmatrix} \tag{2.65}$$

$$\frac{1}{2+k}\begin{vmatrix} +1 & 0 & -1 \\ +K & 0 & -K \\ +1 & 0 & -1 \end{vmatrix} \quad \frac{1}{2+\mathrm{k}}\begin{vmatrix} -1 & -K & -1 \\ 0 & 0 & 0 \\ +1 & +K & +1 \end{vmatrix} \tag{2.66}$$

Prewitt

- Convolución de la imagen con la máscara 2.66 con valor de $K = 1$. Ver [49].
- Es una diferencia de píxeles donde se involucran a los vecinos de filas/columnas adyacentes para proporcionar mayor inmunidad al ruido.
- Ver figuras 2.17 y 2.18 para conocer los efectos de esta máscara sobre latas de aluminio y hojalata.

Sobel

- Convolución de la imagen con la máscara 2.66 con valor de $K = 2$.
- Se duplican los valores al norte, sur, este y oeste del píxel que estamos procesado. Con ello hacemos más sensible al operador a los bordes diagonales con respecto a Prewitt, aunque debilitamos su sensibilidad a los bordes horizontales y verticales. Con todo no se aprecia gran diferencia.
- Ver figuras 2.17 y 2.18 para conocer los efectos de esta máscara sobre latas de aluminio y hojalata.

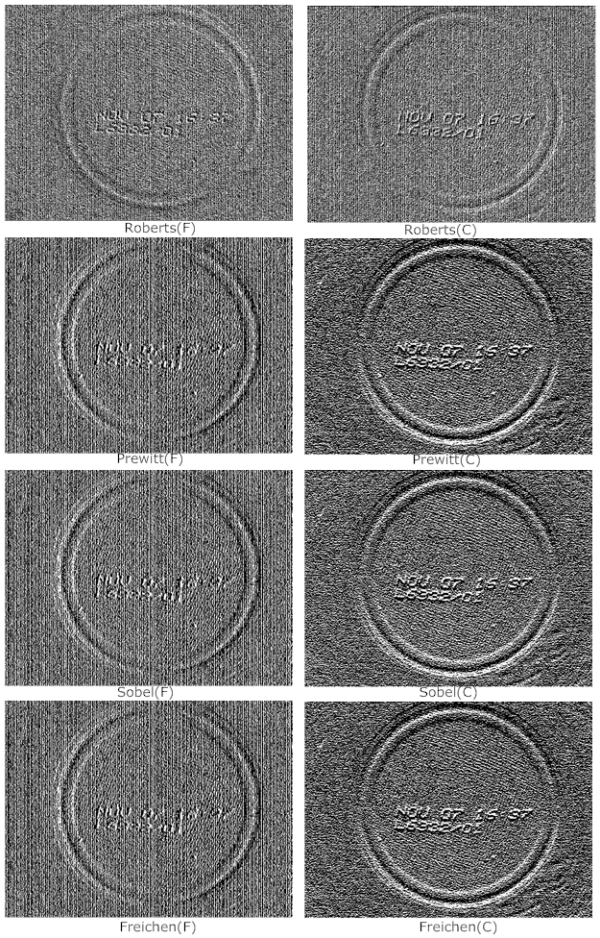


Figura 2.17: Sobre una lata de hojalata diferentes máscaras de convolución

Frei-Chen

- Convolución de la imagen con la máscara 2.66 con valor de $K = \sqrt{2}$.
- Se consigue que el gradiente sea igual para bordes horizontales, verticales y diagonales.
- Ver figuras 2.17 y 2.18 para conocer los efectos de esta máscara sobre latas de aluminio y hojalata.

La teoría matemática del gradiente, basada en el concepto de la primera derivada, se aplica satisfactoriamente para la extracción de bordes. Otros operadores a los citados son las máscaras direcciones como las máscaras de brújula de Kirsh([69]) o de Robinson, pero su alto costo computacional y escasa precisión en la determinación del gradiente desaconsejan su utilización ([24]).

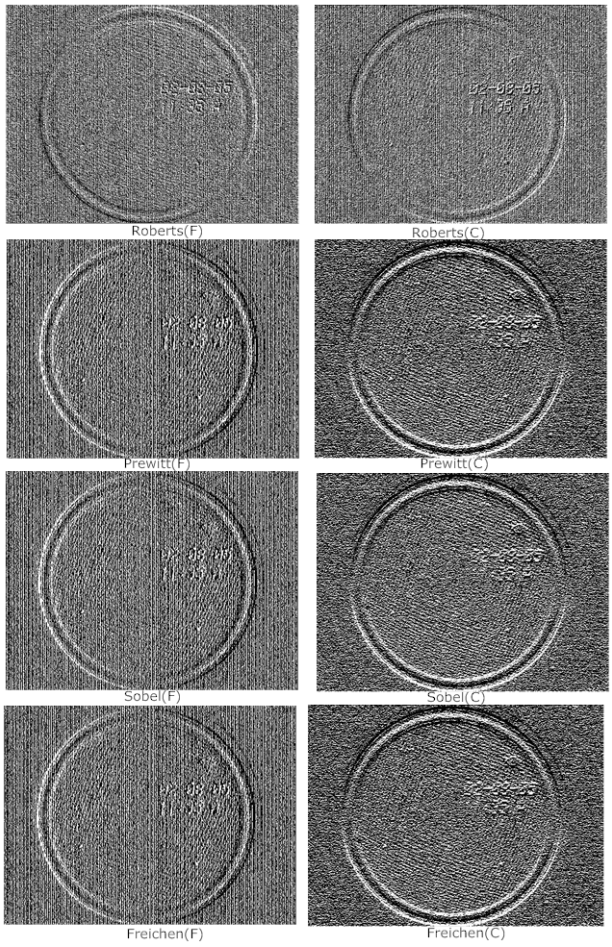


Figura 2.18: Sobre una lata de aluminio diferentes máscaras de convolución.

Detector de Canny El algoritmo de Canny está considerado como uno de los mejores métodos de detección de contornos, mediante el empleo de máscaras de convolución, optimizando los tres criterios de detección, localización y respuesta única/ausencia de ambigüedad ante un borde. Se busca minimizar la distancia entre píxeles señalados como bordes y los bordes reales. Es necesario detectar todos los bordes y únicamente ellos. Y por último sólo existe una única respuesta para cada borde.

El algoritmo consiste básicamente en cinco fases:

1. Suavizado de la imagen mediante una gaussiana para reducir los efectos del ruido.
2. Convolución de la imagen por operadores de gradiente direccionales.
3. Supresión de los gradientes que no son máximos en la dirección del contorno.
4. Aplicación de un umbral adaptativo con histéresis[22] para distinguir los contornos reales del ruido.
5. Integración de los contornos detectados a diferentes escalas en un solo mapa de contornos consistentes.

Este operador es la combinación de un filtro gaussiano (del tipo descrito en 2.4.1) con una aproximación del gradiente , la cual es sensible al borde en la dirección del máximo cambio. El detector de bordes de Canny es la primera derivada de una gaussiana.

Canny calcula el módulo y dirección del gradiente, y compara el valor de la magnitud del gradiente para cada píxel con el de sus vecinos a lo largo de la dirección del gradiente. Así tiene lugar un proceso que elimina los píxeles cuyo gradiente no es máximo y en consecuencia no pertenecen al borde.

2.5.4.2. Aproximación por laplaciana[23]

El principio de funcionamiento las técnicas basadas en aproximaciones por Laplaciana es que es que los cambios de luminancia se representan por un máximo en la primera derivada a lo largo de dicha dirección, y por un paso por cero en cualquier dirección en la segunda derivada. Un borde será óptimo cuando la magnitud del gradiente sea máxima y la segunda derivada (la laplaciana) sea cero.

La laplaciana de una función 2-D es un operador de segunda derivada definido como:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.67}$$

[22] inercia; tendencia a conservar sus propiedades en ausencia del estímulo que la ha generado.

[23] Tiene ese nombre en reconocimiento a Pierre-Simon Laplace que estudió soluciones de ecuaciones diferenciales en derivadas parciales en las que aparecía dicho operador

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial G_x}{\partial x} = \frac{\partial f((i+1,j) - f(i,j))}{\partial x} = f(i+2,j) - 2f(i+1,j) + f(i,j) \quad (2.68)$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial G_y}{\partial y} = \frac{\partial f((i,j+1) - f(i,j))}{\partial y} = f(i,j+2) - 2f(i,j+1) + f(i,j) \quad (2.69)$$

Hay varias formas de definir la laplaciana digital. El requisito básico para definir la laplaciana digital es que los coeficientes asociados con el píxel central sean positivos y los asociados con los coeficientes con el resto de los píxeles sean de píxeles sean negativos. Las tres máscaras laplacianas representan diferentes aproximaciones del operador laplaciano. Como la laplaciana es una derivada la suma de los coeficientes es cero:

$$\left|\begin{array}{ccc} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{array}\right| \left|\begin{array}{ccc} 1 & -2 & 1 \\ -2 & +4 & -2 \\ 1 & -2 & 1 \end{array}\right| \left|\begin{array}{ccc} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{array}\right| \quad (2.70)$$

Como novedad frente a las máscaras primera derivada es que estas son simétricas rotacionalmente. Esto se traduce en que son capaces de detectar bordes en todas las direcciones espaciales[26]. Se aplican seleccionando una máscara y realizando una operación de convolución sobre la imagen. El signo del resultado (positivo o negativo) de dos píxeles adyacentes proporciona información direccional y nos dice que lado del borde es más o menos oscuro.

Sin embargo los contornos de los objetos aparecen inconexos: Seccionados o cortados. El ruido en la imagen (ya sea por la adquisición o por otra razón) es el responsable. Por esta misma razón pueden aparecer falsos contornos. Así que no queda otro remedio que aplicar un postprocesamiento que nos permita encontrar continuidades basados en modelos parametrizados. De estos métodos de parametrización de bordes destaca la transformada de Hough HT y sus variantes para la detección de líneas y círculos descrita en 2.6.2.3.

2.5.5. Segmentación "watershed[24]"

El algoritmo "watershed" (o de división de aguas) parte de una intuición de naturaleza geográfica. Interpreta la imagen como un mapa topológico donde las intensidades de iluminación representan elevaciones[70, 71]. En esta interpretación los mínimos locales son "depresiones" sobre las que convergen los puntos circundantes a esos mínimos.

En otras palabras, en una imagen en grises el nivel de gris de un píxel indica la altitud a la que está la posición de ese píxel. Así, la imagen se torna en un mapa con "valles" separados por cordilleras. Imaginemos que lluvias torrenciales,

[24] Watershed puede traducirse como: Línea divisoria de las aguas. Por ejemplo la frontera nautica de una nación con salida al mar.

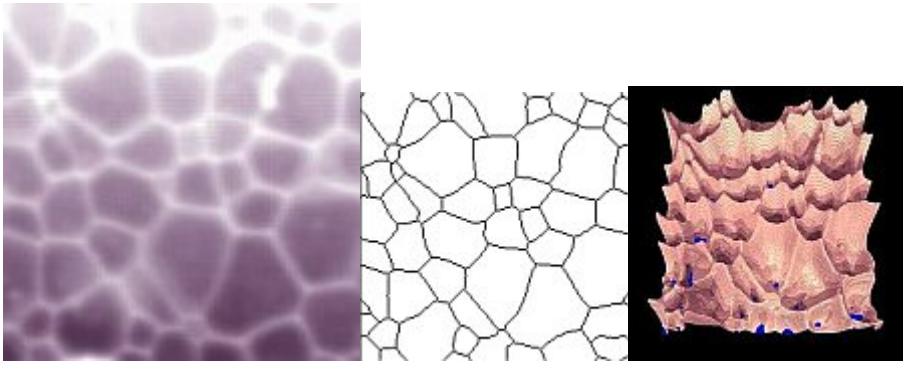


Figura 2.19: Tres etapas en la segmentación "watershed".

a modo de diluvio, asuelan el lugar que representa la imagen. Las primeras posiciones en inundarse serian los mínimos locales en cada uno de los valles presentes. La lluvia no escampa y empiezan a formarse charcos en cada valle. Estos charcos derivan en lagos. El nivel de agua sube y sube hasta que desborda a las cordilleras que rodean los valles. Las aguas de uno y otro valle "chocan". Estas líneas de desbordamiento son la división de aguas a la que hace alusión el nombre del método. La analogía progresa hasta que el nivel de agua cubre el punto más alto del mapa, donde se detiene. Ver figura 2.19.

Esta aproximación gráfica permite comprender el principio que sustenta al método pero hay varios algoritmos posibles que lo implementan. El principal problema del procedimiento es que, en su forma original, tiende a sobresegmentar la imagen. Esto es, dividir la imagen en demasiadas regiones de segmentación. Para impedirlo, existen técnicas como jerarquizar las divisiones según su importancia.

El accesible [72] resulta adecuado para iniciarse en la transformada "Watershed" y en sus definiciones de campo.

2.6. Extracción de características

Un conjunto de características que describe un caso es denominado 'vector de características'. El vector se construye seleccionando y extrayendo dichas características de casos. El vector de características es instrumental para la clasificación porque es el elemento de entrada para los clasificadores de patrones que veremos en la sección 2.7.

La elección de un conjunto de características apropiado para describir un caso es denominada *selección de características.* Ver 2.20. Como norma general, un conjunto de características es más apropiado en tanto maximice los postulados de Niemann de **representatividad**, **discriminabilidad**, **compacidad** y **separación** descritos en 2.6.1. La calidad de la selección de las características

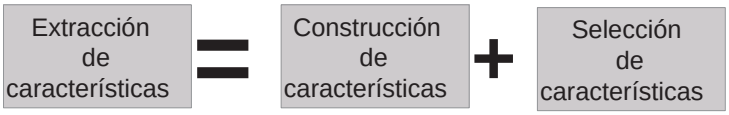


Figura 2.20: La extracción de características comprende su construcción y su selección.

determina en buena medida la efectividad global del sistema de clasificación.Y una buena elección es una tarea complicada.

En muchas ocasiones la búsqueda de un conjunto de características para extraer depende de una observación empírica de un conjunto de muestras y también de un conocimiento experto del dominio sobre el que estamos operando. Recientemente han irrumpido nuevas técnicas dirigidas a automatizar la fabricación o la exploración de un conjunto de características adecuado.

Ni siquiera en la ya antigua materia del reconocimiento de caracteres existe una selección de características definitiva. De hecho existe una enorme cantidad de propuestas que, lógicamente, trataremos aquí una pequeña parte.

Las características seleccionadas o creadas ideales deben maximizar la efectividad del reconocedor, esto es el número de acierto, y al mismo tiempo resulta muy conveniente que sea un conjunto mínimo para simplificar la tarea del clasificador. Suele ser importante que muestren cierta invarianza a la señal de entrada. Esto es; que su valor se mantenga constante a pesar traslaciones, rotaciones, cambios de escala... Y es deseable que puedan ser computadas con suficiente eficiencia para una especificación dada. En el caso de MONICOD, esto es una exigencia.

En este documento enfrentaremos algunos ejemplos de conjuntos de características consolidados que se han revelado útiles.

2.6.1. Postulados de Niemann

Los postulados clásicos de Nienmann [73] son un referente en la problemática del Reconocimiento de Formas

1. **Representatividad**: El diseño de un proceso de reconocimiento de formas dado requiere de una muestra representativa de formas.

2. **Discriminabilidad**: Una forma simple tiene siempre características simples que permiten determinar el grado de pertenencia a una clase, esto es, siempre es posible encontrar un conjunto de características que discriminen una forma simple en algún tipo de espacio.

3. **Compacidad y separación**: Las características de las formas de una clase ocupan un dominio compacto de la representación, y los dominios ocupados por clases diferentes están separados. Este postulado es *condición necesaria* para que pueda llevarse a cabo un proceso de reconocimiento de formas.

Si las formas son complejas podríamos hablar de partes. Así tendremos además dos nuevos postulados cuando tratamos con formas complejas.

1. Una forma compleja se compone de constituyentes más simples, entre los cuales existen ciertas relaciones de estructura.

2. A un conjunto de componentes más unas relaciones entre ellos le corresponde una y sólo una forma compleja.

2.6.2. Descriptores de líneas, contornos y regiones

2.6.2.1. Momentos

Momentos de monomios Los momentos de monomios presentan un bajo costo computacional, pero son muy sensibles al ruido. Además la reconstrucción es extremadamente difícil. Ver 2.21. Su implementación es muy directa pero su inestabilidad numérica, debida a un rango dinámico muy amplio, llega a ser un problema.

Momentos geométricos Son propiedades numéricas que se pueden obtener de una determinada imagen teniendo en cuenta todos los píxeles de la imagen y no sólo los bordes de las formas. Los momentos geométricos han probado ser una herramienta eficiente para análisis de imagen lo que los ha hecho especialmente populares.

Podemos clasificarlos en momento simples, momentos centrales y momentos centrales normalizados.

Los momentos simples se emplean sobre todo para obtener otros momentos pero también suministran información concreta por si mismos. Ver 2.3.

Para una función de densidad continua $p(x, y)$ el momento geométrico de orden $(p + q)$-nésimo m_{pq} se define como:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q p(x, y) dx dy \tag{2.71}$$

Aunque originalmente descrita en forma continua, por razones prácticas se trabaja con su versión continua. De tal manera que dada una imagen considerada como una función discreta de intensidades $f(x, y)$ donde $x = 0, 1, ..., M$ y $y = 0, 1, ..., N$, el momento geométrico queda descrito como:

$$m_{pq} = \sum_{x=0}^{M} \sum_{y=0}^{N} x^p y^q f(x, y) \tag{2.72}$$

Así por ejemplo es fácil ver que el momento simple de orden 0 representa el área de la figura en imágenes binarias. Así, es la suma de los valores de todos los píxeles:

Momento	Significado	Descripción
m_{00}	Área de la región	
m_{10}/m_{00}	Coordenada X del centroide	
m_{01}/m_{00}	Coordenada Y del centroide	
m_{20}, m_{02}	Variancia	Es un índice de la dispersión de la distribución a ambos lados del eje de la media.
m_{30}, m_{03}	Asimetría	Es un índice de la simetría de la distribución de probabilidad respecto al eje de la media.
m_{40}, m_{04}	Curtosis	Es un índice de lo afilado o "picudo" de la distribución de probabilidad en torno a la media.

Cuadro 2.3: Significado de algunos momentos

$$m_{oo} = \sum_{x=0}^{M}\sum_{y=0}^{N} f(x,y) \tag{2.73}$$

Los momentos centrales ofrecen la ventaja de reconocer figuras dentro de una imagen independientemente de su posición. Su versión original continua:

$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\widehat{x})^p (y-\widehat{y})^q f(x,y)dxdy \tag{2.74}$$

En su versión discreta:

$$\mu_{pq} = \sum_{x=0}^{M}\sum_{y=0}^{N} (x-\widehat{x})^p (y-\widehat{y})^q f(x,y) \tag{2.75}$$

Siendo el par $\widehat{x}$,$\widehat{y}$ el centroide de la forma de la que se pretende extraer sus características. Es decir el área de la figura que queda por encima o por debajo de $\widehat{x}$ es la misma, y el área de la figura que queda a la derecha e izquierda de $\widehat{y}$ es la misma.

Es posible poner los momentos centrales en función de los momentos simples considerando que $\widehat{x} = \frac{m_{10}}{m_{00}}$ e $\widehat{y} = \frac{m_{01}}{m_{00}}$, esto es, aprovechando los momentos simples de orden 0 y orden 1.

Así tendremos:

$$\mu_{00} = \sum_{x=0}^{M}\sum_{y=0}^{N} (x-\widehat{x})^0 (y-\widehat{y})^0 f(x,y) = \sum_{x=0}^{M}\sum_{y=0}^{N} f(x,y) = m_{00} \tag{2.76}$$

$$\mu_{10} = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})^1(y-\widehat{y})^0 f(x,y) = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})\, f(x,y) = m_{10} - \frac{m_{10}}{m_{00}} m_{00} = 0 \tag{2.77}$$

$$\mu_{11} = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})^1(y-\widehat{y})^1 f(x,y) = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})\,(y-\widehat{y})\, f(x,y) = m_{11} - \frac{m_{10}}{m_{00}} m_{01} \tag{2.78}$$

$$\mu_{20} = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})^2(y-\widehat{y})^0 f(x,y) = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})^2 f(x,y) = \\ = m_{20} - \frac{2{m^2}_{10}}{m_{00}} + \frac{{m^2}_{10}}{m_{00}} = m_{20} - \frac{{m^2}_{10}}{m_{00}} \tag{2.79}$$

$$\mu_{02} = \sum_{x=0}^{M}\sum_{y=0}^{N}(x-\widehat{x})^0(y-\widehat{y})^2 f(x,y) = m_{02} - \frac{{m^2}_{01}}{m_{00}} \tag{2.80}$$

Los momentos centrales normalizados permiten reconocer figuras dentro de una imagen independientemente de su tamaño.

Se pueden obtener a partir de los momentos centrales normalizandolos con el momento central de orden 0:

$$\eta_{pq} = \frac{\mu_{pq}}{{\mu^{\gamma}}_{00}} \tag{2.81}$$

siendo $\gamma = \frac{p+q}{2} + 1$ para $p + q = 2, 3, ...$

Los momentos centrales normalizados de orden 2 y 3 pueden utilizarse para construir los siete momentos invariantes, también conocidos como momentos de Hu ([74]). Este conjunto de momentos tiene la ventaja de ser invariante a la traslación, rotación y cambio de escala de una figura.

$$\phi_1 = \eta_{20} + \eta_{02} \tag{2.82}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4{\eta^2}_{11} \tag{2.83}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{2.84}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{2.85}$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2\right] + \\ + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{12} + \eta_{03})^2\right] \quad (2.86)$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + \\ + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \quad (2.87)$$

$$\phi_7 = (3\eta_{21} - \eta_{30})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + \\ + (3\eta_{21} - \eta_{30})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \quad (2.88)$$

Su mayor inconveniente se refiere a los valores muy grandes que pueden alcanzar los momentos geométricos lo cual lleva a inestabilidad numérica y sensibilidad al ruido. Sin embargo se ha utilizado con éxito en el reconocimiento de caracteres anglosajones, por ejemplo en [75] se aplica una mejora donde se le asigna a los momentos invariantes un conjunto de pesos óptimo computado automáticamente.

Es una referencia habitual en los extractores de características(un ejemplo de esto es el trabajo de [76] en la clasificación de la planta de bambú). Sin embargo, en [77, 78]el autor establece que el sistema de Hu no es un buen conjunto de características al demostrar que es dependiente e incompleto.

Momentos complejos Los momentos complejos[79, 80] proveen nuevos descriptores adicionales invariantes de forma nativa. Así, son invariantes a rotaciones, traslaciones y escalado, usándose también extensamente.

Se describen de forma parecida a los momentos geométricos. Así dada una imagen considerada como una función discreta de intensidades $f(x, y)$ donde $x = 0, 1, ..., M$ y $y = 0, 1, ..., N$, el momento complejo queda descrito como:

$$m_{pq} = \sum_{x=0}^{M} \sum_{y=0}^{N} (x + yi)^p (x - yi)^q f(x, y) \quad (2.89)$$

Se calculan a partir de los primeros diez momentos geométricos. Como con los momentos de Hu, su mayor inconveniente se refiere a los valores muy grandes que pueden alcanzar los momentos geométricos lo cual lleva a inestabilidad y sensibilidad al ruido. De hecho, ha sido probado que los momentos de Hu son en realidad un caso especial de momentos complejos.

Momentos de base ortogonal continua Los momentos de base ortogonal continua (introducidos originalmente en [81]) han probado ser menos sensitivos al ruido, invariantes de forma nativa, y efectivos para reconstrucción de imagen. Sin embargo su complejidad computacional los hace inadecuados, al menos, para codificación software en aplicaciones en tiempo real, tales como MONICOD. Ver 2.21.

Legendre Los polinomios de Legendre forman la base de los momentos de Legendre:

$$L_{pq} = \frac{(2p+1)(2q+1)}{4} \int_{-1}^{+1} \int_{-1}^{+1} P_p(x) P_q(y) f(x,y) dxdy \tag{2.90}$$

siendo el p-esimo polinomio de Legendre:

$$P_p(x) = \frac{1}{2^p p!} \frac{d^p}{dx^p}(x^2-1)^p, x \in [-1,+1] \tag{2.91}$$

donde p, q son enteros entre (0,∞) y x e y son las coordenadas de la imagen.

La aproximación discreta con la que se opera en una imagen digital de dimensiones $N \times N$ corresponde a:

$$L_{pq} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} P_p(m_N) P_q(n_N) f(m,n) \tag{2.92}$$

donde m_N y n_N son la coordenada de la imagen en el intervalo $[-1,+1]$ que es donde operan los polinomios de Legendre:

$$m_N = \frac{2m - N + 1}{N-1} \tag{2.93}$$

En [82] se llevó a cabo un extensivo análisis y comparación de las definiciones más comunes de momentos. Los autores examinaron la sensibilidad al ruido, redundancia de información y representación de la imagen. Concluyeron que momentos de orden más alto son más vulnerables al ruido. Con ello a partir de un conjunto de momentos por debajo de cierto orden bajo condiciones de ruido se verificó que Zernique superaba a los otros momentos considerados(que incluían geométricos, complejos, Legendre...) en calidad general. Respecto a la redundancia de información los momentos ortogonales (Legendre y Zernique) resultaron ser mejores que otros tipos de momentos. Respecto a los momentos de Legendre en [83] se expone un método más eficiente que el método directo aplicable a imágenes de grises e imágenes binarias.

Zernike También los polinomios de Zernique[84, 81] ofrecen una base muy útil para construir momentos. Presentan invarianza rotacional nativa y son bastante robustos frente al ruido. Escala y traslación pueden ser implementadas usando normalización de momentos.

Los momentos de Zernique de orden p con repetición q se definen como:

$$Z_{pq} = \frac{p+1}{\pi} \sum_X \sum_Y f(x,y) W_{pq}(r,\theta) \tag{2.94}$$

donde

$$W_{pq}(r,\theta) = R_{pq} e^{iq\theta} \tag{2.95}$$

$$R_{p,\pm q}(r) = \sum_{k=q, p-|k|=par}^{p} B_{pqk} r^k \tag{2.96}$$

$$B_{pqk} = \frac{(-1)^{(p-k)/2} ((p+k)/2)!}{((p-k)/2)!((q+k)/2)!((k-q)/2)!} \tag{2.97}$$

En [85] podemos encontrar un análisis de rendimiento y proponen una nueva técnica (que denominan método *q-recursivo*) que mejora el rendimiento de los momentos de Zernike. La mejora de velocidad en la computación se obtiene empleando polinomios de alto orden q para derivar polinomios de bajo orden q y no usa términos factoriales.

Momentos de base ortogonal discreta Los problemas relacionados con el uso de una base ortogonal continua son purgados utilizando una base ortogonal discreta como son los polinomios de Tchebichef.

Los momentos de base ortogonal discreta aventajan los momentos de base ortogonal continua en velocidad y mejor capacidad de presentación de imagen, compartiendo algunas características positivas como la baja sensibilidad a ruido y efectividad para la reconstrucción de imagen. Sin embargo no son invariantes a transformaciones de forma nativa, obligando a normalizar la imagen como paso previo a la extracción de características para obtener cierta invarianza. Ver 2.21.

Tchebichef Los momentos de Tchebichef pueden ser vistos como un caso especial de momentos de Hahn descritos en 2.6.2.1.

Son ortogonales en el dominio del espacio de coordenadas de la imagen. Esa característica elimina la necesidad de una aproximación discreta en su implementación numérica. Con ello, a parte de hacerla más aproximada, se reduce el costo computacional [86].

Para describirlos procedamos desde los momentos de Tchebichef unidimensionales donde tendremos una función discreta de intensidades $f(x)$ donde $x = 0, 1, ..., N$.

$$T_p = \frac{1}{\widetilde{\rho}(r,N)} \sum_{x=0}^{N-1} \widetilde{t}_p(x) f(x) \tag{2.98}$$

en donde:

$$\widetilde{t}_n(x) = \frac{t_n(x)}{\beta(n,N)} = \frac{1}{\beta(n,N)} n! \sum_{k=0}^{n} (-1)^{n-k} \left(\begin{array}{c} N-1-k \\ n-k \end{array} \right) \left(\begin{array}{c} n+k \\ n \end{array} \right) \left(\begin{array}{c} x \\ k \end{array} \right) \tag{2.99}$$

La versión bidimensional se obtiene con:

$$T_p = \frac{1}{\widetilde{\rho}(p,N)} \frac{1}{\widetilde{\rho}(q,N)} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \widetilde{t}_p(x) \widetilde{t}_p(y) f(x) \tag{2.100}$$

De momento no existen demasiados algoritmos de computación rápida para los momentos de base ortogonal discreta debido a su introducción relativamente reciente. Sin embargo para esta versión bidimensional ya existen propuestas de algoritmos de computación rápida. En [87] se emplea la representación por bloques de la imagen para imágenes binarias de [88] y la representación por rebanadas de intensidad de [89] y se derivan algunas propiedades de los polinomios de Tchebichef que pueden aplicarse para cada bloque o rebanada mejorando la eficiencia en el computo.

Hahn Se han propuesto los momentos polinómicos de Hahn como una alternativa más aproximada que los momentos de Tchebichef, o los momentos no discutidos aquí de Krawtchouk, dado que son una generalización de estos con la adecuada configuración de parámetros. [90][91]. Ofrecen exactitud incrementada en la reconstrucción sin costo computacional adicional respecto a los momentos de Tchebichef.

Los polinomios de Hahn son polinomios discretos clásicos con medida discreta:

$$d\lambda(t) = \sum_{k=0}^{N} w_k d(t-k) dt \tag{2.101}$$

con

$$w_k = \left(\begin{array}{c} a+k \\ k \end{array} \right) \left(\begin{array}{c} b+N-k \\ N-k \end{array} \right), \; a > -1, b > -1 \tag{2.102}$$

De la citada figura 2.21 puede concluirse que los momentos de monomios son la solución más rápida dada su baja complejidad computacional, aunque ofrece debilidades importantes como la sensibilidad al ruido. Por esta razón se ha hecho un esfuerzo en hallar alternativas hardware para la implementación de los momentos de base ortogonal tratando de aliviar su baja velocidad, y aprovechando así sus características ventajosas.

Otro camino ha sido desarrollar algoritmos alternativos a la computación directa de los momentos geométricos para hacerlos aún más rápidos. La intención es computar otros tipos de momento(Legendre, Zernike) a través de su relación con los momentos geométricos(Ver a este respecto [93]). En este sentido el trabajo de [94] (que llegó a ser el estado del arte en lo que a computación de momentos se refiere) ofrecía un filtro 2-D eficaz para computar momentos

	Métodos	Sensibilidad al ruido	Complejidad computacional	Exactitud en la reconstrucción	Invarianza nativa
Momentos de monomios	Momentos geométricos	Alta	Baja	No aplicable	No
	Momentos complejos	Alta	Baja	No aplicable	Si
Momentos de base ortogonal continua	Legendre	Baja	Alta	Media	Si
	Zernike	Baja	Muy Alta	Alta	Si
Momentos de base ortogonal discreta	Tchebichef	Baja	Media	Muy Alta	No
	Hahn	Baja	Media	Muy Alta	No

Figura 2.21: Comparativa de momentos [92]

geométricos hasta tercer orden, e incluso su implementación hardware en un único chip VLSI.

En [95] se hace una comparativa intensiva de estos algoritmos alternativos para la computación de momentos geométricos, que está gobernada por el tipo de imagen (escala de grises o binaria), el espacio de dimensiones en el que se representa la imagen (2-D, 3D, o de mayor orden, y el orden de los momentos a ser computados.

Es posible combinar diferentes momentos para resolver un mismo problema de clasificación. En [96] se muestra una interesante aproximación que combina momentos y descriptores genéricos de Fourier. Para ello primero hace una partición de la imagen en clases usando propiedades morfológicas, y a continuación extrae las características para cada clase utilizando la técnica (momento) que mejor reconoce las imágenes de esa clase.

2.6.2.2. Descriptores de Fourier

Dado un contorno digital cerrado de N puntos en el plano, partimos de un punto arbitrario (x_0, y_0) y recorremos el contorno en el sentido de las agujas del reloj (por ejemplo) obteniendo los pares de coordenadas (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3) ,...,(x_{N-1}, y_{N-1}). Al final regresaremos al mismo punto, dado que es una curva cerrada. Se trata por tanto de una sucesión de pares periódica.

Para expresar esta notación con mayor claridad haremos $x(k) = x_k$ y $y(k) = y_k$ de manera que la serie podemos expresarla como $s(k) = [x(k), y(k)]$ para $k = 0, 1, 2, 3, ..., N-1$. Por otro lado cada par de coordenadas puede ser expresado como un número complejo:

$$s(k) = x(k) + iy(k) \tag{2.103}$$

Esta representación tiene la gran ventaja de que convierte un problema bidimensional en uno unidimensional[67].

La transformada discreta de Fourier (DFT) de $s(k)$ es:

$$a(u) = \frac{1}{N} \sum_{k=0}^{N-1} s(k) e^{\frac{-i2\pi uk}{N}} \tag{2.104}$$

siendo $u = 0, 1, 2, ..., N-1$. Son estos coeficientes complejos $a(u)$ a los que se les llama "descriptores complejos" y dan nombre a esta representación.

La transformada de Fourier inversa de estos $a(u)$ restaura $s(k)$.

$$s(k) = \sum_{u=0}^{N-1} a(u) e^{\frac{i2\pi uk}{N}} \tag{2.105}$$

siendo $k = 0, 1, 2, ..., N-1$.

Supongamos sin embargo que en vez de todos los $a(u)$ utilizamos los primeros M coeficientes. Esto equivale a hacer $a(u) = 0$ para $u > M-1$ en la ecuación 2.105. El resultado es la aproximación:

$$\widehat{s}(k) = \sum_{u=0}^{M-1} a(u) e^{\frac{i2\pi uk}{N}} \tag{2.106}$$

para $k = 0, 1, 2, ..., M-1$. Debe subrayarse que aunque sólo se utilicen los M primeros términos para obtener cada componente de $\widehat{s}(k)$, k permanece en el intervalo de 0 a $N-1$.

Esto significa que existe el mismo número de puntos en el contorno aproximado, sólo que no se utilizan tantos puntos en la reconstrucción de cada punto. Si el número de puntos de contorno es grande se suele utilizar un valor de M múltiplo de dos(2) para aprovechar las ventajas que ofrece la FFT[25].

Los componentes de altas frecuencias responden al detalle fino, y los de baja frecuencia determinan la forma global. Como consecuencia entre menor sea el valor de M más detalle del contorno se pierde. Por tanto si bien algunos coeficientes de orden inferior son capaces de capturar la forma en grueso, se necesitan muchos más términos de orden superior para definir con precisión características más acusadas como esquinas y líneas rectas[67]. Lo interesante de esta aproximación es que bastan las componentes de baja frecuencia (los primeros términos) para quedarse con la esencia de la forma. Y pueden utilizarse como base para diferenciar entre dos formas de contorno distintas.

Como siempre, nos interesa que los descriptores sean tan insensibles como sea posible a traslación, rotación, y cambios de escala. Para este método también convendría insensibilidad al punto de partida desde el cual empezamos a coger

[25] Fast Fourier Transform

Transformación	Borde	Descriptor de Fourier
Identidad	$s(k)$	$a(u)$
Rotación	$s_r(k) = s(k)e^{i\theta}$	$a_r(u) = a(u)e^{i\theta}$
Traslación	$s_t(k) = s(k) + \Delta_{xy}$	$a_t(u) = a(u) + \Delta_{xy}\delta(u)$
Escalado	$s_s(k) = \alpha s(k)$	$a_s(u) = \alpha a(u)$
Punto de comienzo	$s_p(k) = s(k - k_0)$	$a_p(u) = a(u)e^{\frac{-2j\pi k_0 u}{N}}$

Figura 2.22: Una selección de las propiedades básicas de los descriptores de Fourier

muestras. Los descriptores no son insensibles a estos cambios pero resulta relativamente sencillo aplicarles transformaciones para recrear esos cambios y así ajustarlos a diferentes escalas y rotaciones. Aquí por ejemplo la transformación necesaria para recrear la rotación:

$$a_r(u) = \frac{1}{N}\sum_{k=0}^{N-1} s(k)e^{i\theta}e^{-\frac{i2\pi uk}{N}} = a(u)e^{i\theta} \tag{2.107}$$

para $u = 0, 1, 2, ..., N-1$.

En el cuadro 2.22 tenemos una selección de estas transformaciones. En ella:

$$\Delta_{xy} = \Delta x + i\Delta y \tag{2.108}$$

y así la notación $s_t(k) = s(k) + \Delta_{xy}$ indica la re-definición de la serie como:

$$s_t(k) = [x(k) + \Delta x] + i\,[y(k) + \Delta y] \tag{2.109}$$

Esto es, la traslación consiste en sumar el desplazamiento constante a todas las coordenadas del contorno. Por otro lado la expresión $s_p(k) = s(k - k_0)$ significa re-definir la secuencia como:

$$s_p(k) = x(k - k_0) + iy(k - k_0) \tag{2.110}$$

que no es mas que cambiar el punto de partida desde $k = 0$ a $k = k_0$

2.6.2.3. Transformada de Hough

En 1962 Hough propuso un método para encontrar la ecuación de una línea que pase por un conjunto de n puntos en el plano xy. Este trabajo seminal fue generalizado a otras formas e introducido en el área de visión por computador en el influyente trabajo *"Use of the Hough Transformation to Detect Lines And Curves in Pictures"[97]* de Richard O. Duda y Peter E. Hart. El método se hecho muy popular pero tiene un especial interés en el sector industrial porque muchos objetos manufacturados contienen formas circulares o elípticas como es, precisamente, nuestro caso.

Para presentar el método partamos de su concepción inicial. En el plano xy una línea que une una secuencia de puntos o píxeles puede expresarse como:

$$y = ax + b \tag{2.111}$$

La meta es evaluar a y b.

Dado un punto cualquiera (x_i, y_i), existe un número infinito de líneas que pasan por dicho punto y por tanto cumplen:

$$y_i = ax_i + b \tag{2.112}$$

para infinitos pares de parámetros (a, b).

Ahora bien, si reescribimos la ecuación como:

$$b = -x_i a + y_i \tag{2.113}$$

y pasamos a considerar el plano (espacio de parámetros) ab tenemos la ecuación de una sola línea para un valor fijo (x_i, y_i).

También un segundo punto (x_j, y_j) tendrá asociada una única línea en el espacio de parámetros. Esa segunda línea interceptará con la primera en algún punto (a', b'). Este punto corresponde precisamente a los parámetros de la línea en el eje xy que une a (x_i, y_i) y a (x_j, y_j). En realidad todos los puntos (x, y) por los que pasa esa línea tienen asociadas líneas en el espacio de parámetros ab que interceptan en ese punto.

A partir de esta evidencia la transformada de Hough opera como sigue:

1. Se subdivide el espacio de parámetros en secciones o celdas, donde cada celda está en el rango (a_{min}, a_{max}) y (b_{min}, b_{max}). La celda (i, j) la haremos corresponder con el par (a_i, b_j). Por supuesto es una inexactitud que puede reducirse aumentando la densidad (las subdivisiones) de nuestro particionado. Estas celdas son acumuladores iniciados a cero.

2. Para cada uno de los n puntos del conjunto que tenemos en el plano xy, cogemos todos los a asociado a cada celda y calculamos sus respectivos b con 2.113. Para cada par (a_i, b_j) obtenido de este modo redondeamos y acumulamos un voto en la celda asociada.

3. Acabado este proceso la célula con mayor número de votos nos ofrece el par (a, b) buscado.

Aquí ya puede apreciarse el fuerte costo computacional del método: Si se divide el espacio de parámetros en K partes y tenemos n puntos en la colección de puntos, habrá que hacer $K \times n$ veces la operación descrita. Recordemos que el espacio de parámetros ab es real, al contrario de xy que es discreto.

Un problema que aparece al obrar como se ha descrito es que al utilizar la ecuación 2.112 para representar la recta es que tanto la pendiente como la ordenada en el origen se acercan a infinito cuando la recta se aproxima a posiciones verticales. Por esto una forma mucho mejor para representar la recta es usar coordenadas polares:

$$x \cos\theta + y \sin\theta = \rho \tag{2.114}$$

donde el espacio de parámetros viene dado por θ y ρ.

Ahora bien, lo que se aplica a rectas puede extenderse a otras formas que puedan describirse con expresiones de la forma $g(x, c)$ donde x es un vector de coordenadas y c es un vector de coeficientes. Este es caso de las circunferencias donde el espacio geométrico es:

$$(x-a)^2 + (y-b)^2 = c^2 \tag{2.115}$$

Claro está que ahora el espacio de parámetros es tridimensional abc y por ende las células son cúbicas y acumuladores de la forma $A(i, j, k)$. Ahora el procedimiento sería rastrear todos los posibles pares (a, b) para calcular c y actualizar el acumulador correspondiente. Como puede verse la complejidad de la transformada de Hough depende en gran medida del número de coordenadas y coeficientes de la función a la que se quiere aplicar[26]. Así, aunque ha sido probada en presencia de ruido y cuando hay partes ocultas o no presentes del objeto, no puede soportar la cadencia exigida en un entorno como MONICOD. Sin embargo se apunta a su empleo en tiempo de calibrado con latas estáticas para re-calcular ciertos parámetros relacionados con los contornos de la lata contribuyendo a automatizar la labor de parametrización.

2.6.2.4. Ajuste de Elipses

Dado el interés particular de MONICOD en procesar latas, la forma más recurrente es la superficie de la lata que es circular. Debido al efecto de deformación óptica casi siempre aparecen con formas ligeramente elípticas en la imagen. A modo de muestra en este apartado se busca la ecuación de la elipse que mejor se adapta a una colección de N puntos previamente extraídos $p_1, p_2, p_3, ..., p_N$ donde $p_i = [x_i, y_i]$.

Sea $x = [x^2, xy, y^2, x, y, 1]$, $p = [x, y]$ y :

$$f(p, a) = x^t a = ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{2.116}$$

la ecuación explícita de la elipse genérica caracterizada por los parámetros $a = [a, b, c, d, e, f, g]^t$. El propósito es tratar de encontrar el vector de parámetros a_0 asociado a la elipse que mejor ajusta $p_1, p_2, p_3, ..., p_N$ en el sentido de mínimos cuadrados, como una solución de:

$$\min_a \sum_{i=1}^{N} |D(p_i, a)|^2 \tag{2.117}$$

Aquí $D(p_i, a)$ es una distancia.

Si establecemos como distancia la euclidiana tenemos que nuestro problema será:

$$\min_a \sum_{i=1}^{N} \|p - p_i\|^2 \tag{2.118}$$

bajo la restricción de que$p$ pertenece a la elipse. Esto es, $f(p,a)=0$.

Lo más efectivo es resolver el problema mediante los multiplicadores de Lagrange.

$$L=\sum_{i=1}^{N}\|p-p_i\|^2-2\lambda f(p,a) \tag{2.119}$$

y hacemos $\frac{\partial L}{\partial x}=\frac{\partial L}{\partial y}=0$ lo que produce:

$$p-p_i=\lambda\nabla f(p,a) \tag{2.120}$$

No conocemos p pero podemos expresarlo como una función de cantidades computables. Para hacer esto se introducen dos aproximaciones:

1. Se considera una aproximación de la curva de primer orden:

$$0=f(p,a)\approx f(p_i,a)+[p-p_i]^t\nabla f(p_i,a) \tag{2.121}$$

2. Se supone que los p_i están suficientemente próximos a la curva, de modo que:

$$\nabla f(p)\approx\nabla f(p_i) \tag{2.122}$$

La condición 2.122 nos permite reescribir la expresión 2.120 como:

$$p-p_i=\lambda\nabla f(p_i,a) \tag{2.123}$$

que insertada en 2.121 nos ofrece:

$$\lambda=\frac{-f(p_i,a)}{\|\nabla f(p_i,a)\|^2} \tag{2.124}$$

y sustituyéndola finalmente en 2.120 encontramos:

$$\|p-p_i\|=\frac{|f(p_i,a)|}{\|\nabla f(p_i,a)\|} \tag{2.125}$$

que nos permite la cantidad desconocida $\|p-p_i\|$ por una función que puede obtenerse.

Así, dado el conjunto de puntos $p_1,p_2,p_3,...,p_N$ se fija a a un valor inicial a_0 de forma que utilizando a_0 como punto inicial, se realiza la minimización numérica 2.118 para encontrar la mejor solución a_m que describe la elipse.

2.6.2.5. Representación basada en líneas

Hay varias formas de representar un carácter como un conjunto de segmentos y curvas.

Se puede utilidad el conjunto de puntos frontera que conforman el contorno. Trazar un contorno envuelve una detección de bordes como las vistas en2.5.4.

Algoritmo 2.7 Esquema general del algoritmo de adelgazamiento.

repetir
Convertir a fondo todos los píxeles que puedan ser eliminados.
hasta Ningún píxel sea eliminado

En lugar de trazar el contorno podemos trazar el esqueleto del carácter. La idea de esqueleto de un carácter es bastante intuitiva. Una forma de pensar en él es como una colección de puntos que corresponden a los centros de todos los círculos máximos inscritos en el carácter. Un círculo inscrito es máximo cuando no está contenido por entero en otro círculo inscrito. El esqueleto es útil porque provee una representación simple y compacta que preserva las dimensiones originales del carácter y muchas de las características topológicas (ver 2.6.2.6).

Hay varias formas de obtener el esqueleto *(esqueletización).* Así el esqueleto de una región se puede definir mediante la transformación del eje medio(MAT); La MAT de una región R con borde B es la siguiente: Para cada punto p de R, se encuentra su vecino más próximo en$B$. Si p tiene más de un vecino de estos se dice que pertenece al eje medio (el esqueleto de R. El concepto de "más próximo" depende de la definición de distancia. Para nuestro caso R es el interior del carácter y B su contorno. Desgraciadamente esta técnica aunque directa es computacionalmente costosa.

Una alternativa es someter a los caracteres a un proceso de adelgazamiento hasta "dejarlos en los huesos". Esto es; de forma progresiva vamos suprimiendo tantos píxeles como sea posible del carácter sin que quede afectado la forma general del patrón. El esqueleto obtenido debe tener las siguientes propiedades:

1. Debe ser tan delgado como sea posible.
2. Debe estar conectado.
3. Debe estar centrado.

Satisfechas estas cualidades debe detenerse el proceso de adelgazamiento. La idea es siempre no eliminar puntos extremos, ni romper la continuidad, ni causar excesiva erosión en la región. En general todo algoritmo de adelgazamiento para imágenes binarias donde negro es tinta y blanco es fondo puede ser resumido como indica 2.7.

Algoritmo de Zhang y Suen A modo de ejemplo, describimos el algoritmo de Zhang y Suen [67], que es prácticamente clásico en este área:

Este algoritmo supone que los píxeles del objeto(el carácter) están marcados como valor uno mientras que el resto(el fondo) lo está con el valor cero. Ver ilustración 2.23. Los píxeles del contorno son aquellos etiquetados como uno, que tienen al menos uno de los ocho vecinos a cero.

1. Un píxel del contorno se marca para borrado si cumple las siguientes cuatro condiciones:

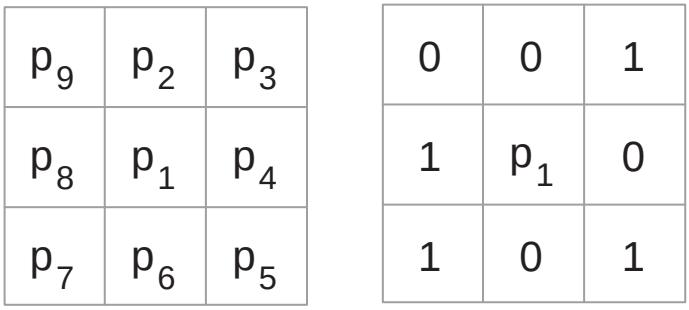


Figura 2.23: Representación extraída de [67]. En el ejemplo $N(p_1) = 4$ y $S(p_1) = 3$

   a) Si la vecindad del píxel $N(p_1)$ tiene dos o más píxeles no nulos pero seis o menos píxeles no nulos. Es decir: $2 \leq N(p_1) \leq 6$ con $N(p_1) = p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9$
   b) Si el número de transiciones (0-1) (y no 1-0) al recorrer los vecinos en el orden p_2, p_3, p_4, p_5, p_6, p_7, p_8 y p_9 es igual a uno. Esto es; $S(p_1) = 1$
   c) $p_2 \cdot p_4 \cdot p_6 = 0$
   d) $p_4 \cdot p_6 \cdot p_8 = 0$

2. Los píxeles marcados se borran.

3. Un píxel del contorno se marca para borrado si cumple las siguientes cuatro condiciones:

   a) Si la vecindad del píxel $N(p_1)$ tiene dos o más píxeles no nulos pero seis o menos píxeles no nulos. Es decir: $2 \leq N(p_1) \leq 6$
   b) Si el número de transiciones (0-1) (y no 1-0) al recorrer los vecinos en el orden p_2, p_3, p_4, p_5, p_6, p_7, p_8 y p_9 es igual a uno. Esto es; $S(p_1) = 1$
   c) $p_2 \cdot p_4 \cdot p_8 = 0$
   d) $p_2 \cdot p_6 \cdot p_8 = 0$

4. Los píxeles marcados se borran.

5. Si no ha habido supresiones el algoritmo termina. En caso contrario repetir el proceso.

Es interesante resaltar la separación entre marcar para borrado y el propio borrado. De esta manera se mantiene intacta la representación mientras se procesan todos los puntos.

2.6.2.6. Propiedades topológicas

Las propiedades topológicas proporcionan descriptores invulnerables ante deformaciones. Estos no se ven afectado por la rotación o por un estiramiento, aunque si por cortes y roturas en la imagen. Cualquier descriptor apto no puede depender de ninguna medida de distancia.

Una propiedad topológica apta para la descripción de regiones es el número de componentes conectadas. Dada una región de la imagen una componente conectada de dicho conjunto es un subconjunto de tamaño máximo en el cual todos los puntos se pueden interconectar mediante una curva si rupturas que se pueda incluir en el subconjunto[24].

Otro ejemplo de descriptor topológico es el número de huecos de una región. Formalmente el número de huecos en la figura es uno menos que el número de componentes conexas en el complemento de la figura

De las anteriores propiedades topológicas pueden derivarse nuevos descriptores. Por ejemplo la diferencia entre el número de regiones conectadas C y el número de huecos H es denominado número de Euler:

$$E = C - H \tag{2.126}$$

Estas propiedades topológicas pueden resultar muy convenientes para la validación de caracteres como un descriptor preliminar. Por ejemplo, si el número de Euler no coincide con un determinado patrón el carácter ya no puede ser comparado[26]. Cómo se verá más adelante esta interesante aproximación se ve comprometida porque los caracteres con los que trata MONICOD pueden adolecer de fuerte fragmentación.

2.7. Clasificación de patrones

Esta fase es crucial dentro del reconocimiento. Con las características extraídas (representadas en el vector de características) debemos identificar la clase a la que pertenece el objeto(el carácter).

Los clasificadores pueden ignorar la naturaleza de los patrones que clasifican, por eso pueden aplicarse a una inmensa variedad de dominios de reconocimiento. La premisa es que debe subyacer en los vectores de características que son entregados al clasificador alguna clase de estructura o patrón que haga posible la clasificación. Por eso, tal como se ha explicado, es crítica la selección de características de la fase previa(las características deben ser "características") así como suprimir, en tanto sea posible, el ruido y la varianza (a rotaciones, traslaciones, escalado, iluminación,...) de las características.

Es habitual que el clasificador esté sujeto a una etapa de 'aprendizaje' donde el clasificador aprende a discernir como se relacionan los vectores de características con las clases generalizando ese conocimiento. La palabra clave aquí es 'generalizar'. Esta característica asocia a los clasificadores con la disciplina de la inteligencia artificial y su área de aprendizaje de máquinas[26]. Un correcto

[26] Machine Learning

aprendizaje resulta crítico para una buena clasificación.

La complejidad de la clasificación es el otro factor involucrado en el éxito. Una separabilidad entre clases difícil obliga al clasificador a equiparar esa complejidad.

Un aspecto interesante de los clasificadores es su habilidad para adquirir nuevo conocimiento mediante el proceso del aprendizaje. Hay varios esquemas de aprendizaje: Aprendizaje supervisado, no supervisado, mediante refuerzo... En este documento nos centraremos en esquemas de aprendizaje supervisado al tratarse de la rama más desarrollada y es compatible con las especificaciones de MONICOD.

Hay una vasta literatura sobre clasificadores dada su enorme importancia en cualquier proceso de reconocimiento. En este documento nos centraremos en cinco tipos de clasificadores que se han mostrado especialmente efectivos.

- Los métodos estadísticos y en particular el método de los K-vecinos-mas-cercanos (KNN).
- Las redes neuronales y en particular:
  - El perceptrón multicapa con algoritmo de aprendizaje por retropropagación
  - Redes de función radial(RBF).
- Las máquinas de vectores soporte.
- La clasificación por comparación de plantillas (Template Matching), utilizada en MONICOD.

2.7.1. Métodos estadísticos

Los métodos estadísticos de clasificación están basados en la teoría de decisión de Bayes. Así, dada la matriz de perdidas asociada a clasificaciones erróneas y las probabilidades estimadas se trata de minimizar las clasificaciones erróneas.

Asumamos un vector de características con d características $\boldsymbol{x} = [x_1, x_2, ..., x_d]^T$. El vector es construido a partir de un conjunto de d características $\{x_1, x_2, ..., x_d\}$ extraídas de un patrón de entrada[27] . Se establece que $\boldsymbol{x}$ pertenece a una de M clases predefinidas[28] $\{w_1, w_2, ..., w_M\}$.

Así, dadas las probabilidades *a priori* $P(w_i)$ y las distribuciones de probabilidad condicional a la clase $p(x|w_i), i = 1, ..., M$ las probabilidades *a posteriori* se computan con la expresión 2.127.

$$P(w_i|x) = \frac{P(w_i)p(x|w_i)}{p(x)} = \frac{P(w_i)p(x|w_i)}{\sum_{j=1}^{M} P(w_j)p(x|w_j)} \tag{2.127}$$

[27] Por ejemplo un carácter.

[28] El alfabeto es un posible conjunto de clases para un carácter donde cada letra corresponde a una clase

Ahora, dada la matriz de perdidas $[c_{ij}]$ donde c_{ij} es la perdida ocasionada por clasificar erróneamente un patrón de la clase w_j como de la clase w_i, la perdida esperada o riesgo condicional de clasificar el patrón $\boldsymbol{x}$ en la clase w_i se formula en la expresión 2.128.

$$R_i(x) = \sum_{j=1}^{M} c_{ij} P(w_j|x) \tag{2.128}$$

Así, clasificando el patrón $\boldsymbol{x}$ en la clase i que produce la menor pérdida $R_i(x)$ minimizamos la perdida esperada.

Frecuentemente se asume que la perdida por clasificación errónea es la misma para cualquier par de clases y que una clasificación correcta produce una perdida cero. Así se deriva la expresión 2.129.

$$c_{ij} = \begin{cases} 1, i \neq j \\ 0, i = j \end{cases} \tag{2.129}$$

De este modo la expresión 2.128 se completa en 2.130.

$$R_i(x) = \sum_{j \neq i}^{M} c_{ij} P(w_j|x) = 1 - P(w_i|x) \tag{2.130}$$

Es inmediato deducir de 2.130 que la clase a seleccionar es aquella que maximiza la probabilidad condicional, porque minimiza el ratio de error que resulta ser $1 - \text{máx}_i P(w_i|x)$. Este ratio de error se le conoce como error bayesiano.

Con esto, regresamos al cociente 2.127 donde se aprecia que el denominador no depende de la clase, por lo que la clase a seleccionar es aquella que maximiza el numerador tal como se indica en 2.131 o su logaritmo indicado en 2.132.

$$\max_i g(x, w_i) = \max_i P(w_i) p(x|w_i), i = 1, ..., M \tag{2.131}$$

$$\max_i g(x, w_i) = \max_i \log P(w_i) p(x|w_i), i = 1, ..., M \tag{2.132}$$

A las funciones 2.131 y 2.132 se les denomina funciones discriminante. Para un par de clases w_i y w_j el conjunto de puntos en el espacio de características con igual valor de discriminante $g(x, w_i) = g(x, w_j)$ es la frontera de decisión.

Métodos de clasificación paramétrica asumen formas funcionales(por ejemplo gaussianas) para funciones de densidad de probabilidad condicional y estiman los parámetros por máxima probabilidad. Por otra parte los métodos de clasificación no paramétrica no asumen forma alguno sino que estiman distribuciones arbitrarias adaptadas a las muestras de entrenamiento. Nos centraremos en estos métodos no paramétricos en 2.7.1.1.

Los métodos estadísticos están presentes en innumerables textos sobre clasificadores. En [17] se profundiza en ellos (sobre todo en sus aspectos matemáticos), pero el espléndido [19] parece adecuado para iniciarse.

2.7.1.1. Métodos no paramétricos : K-vecinos-mas-proximos

Los clasificadores no paramétricos no asumen ninguna forma funcional para la distribución condicional. En su lugar, la densidad puede tener cualquier forma arbitraria dependiendo de la información de entrenamiento. Una aproximación general está basada en la definición de densidad de probabilidad: La frecuencia de puntos de datos por unidad de volumen de espacio de características.

Asumiendo que la densidad $p(x)$ es continua y no varía considerablemente en una región local $\Re$, la probabilidad que un vector aleatorio x, gobernado por la distribución de $p(x)$, caiga en la región $\Re$ puede ser aproximada por:

$$P = \int_{\Re} p(x')dx' \approx p(x) \times V \tag{2.133}$$

donde V es el volumen de $\Re$.

Por otra parte si hay N puntos de datos(N es suficientemente grande) y k de ellos caen en la región local $\Re$ alrededor de x , la probabilidad puede aproximarse con:

$$P \approx \frac{k}{N} \tag{2.134}$$

Combinando las ecuaciones 2.133 y 2.134 tenemos:

$$p(x) \approx \frac{k}{N \times V} \tag{2.135}$$

Para aproximarse a una estimación práctica de la densidad usando 2.135 podemos hacer dos cosas: Fijar V y determinar el valor$k$ desde los datos (es el método de la ventana Parzen). O bien, fijar k y determinar el valor de V desde los datos (es el método K-vecinos-mas-próximos).

El método de los K-vecinos-mas-próximos sobresale por su relativa simplicidad y su considerable popularidad. Se basa en localizar el conjunto de muestras de entrenamiento más próximas a la muestra a clasificar. Es un ejemplo simple de clasificador de aprendizaje basado en ejemplos. La familia de los clasificadores con aprendizaje basado en ejemplos se caracterizan por basar su clasificación en la comparación más o menos directa de las nuevas instancias del problema con instancias almacenadas en memoria durante el periodo de entrenamiento. Al proceder así el entrenamiento tiende a ser muy rápido porque se limita a registrar en memoria las instancias de entrenamiento. Sin embargo la clasificación tiende a ser lenta porque el esfuerzo de generalización se lleva a cabo durante las búsquedas en oposición a otras aproximaciones que generalizan durante la fase de entrenamiento.

Así dado N puntos de entrenamiento de M clases, al tratar de clasificar un nuevo vector $\boldsymbol{x}$ se construye una ventana centrada en $\boldsymbol{x}$ suficientemente grande para que caiga en ella exactamente k instancias de entrenamiento. Estos serán los k vecinos más próximos a los que hace alusión el nombre del método. Asumiendo que los k vecinos más próximos incluyen k_i puntos de la clase w_i, donde $i = 1, .., M$ y $\sum_{i=1}^{M} k_i = k$, la densidad condicional de la clase alrededor

de $\boldsymbol{x}$ se estima desde la expresión 2.136 dondeN_i es el número de puntos de entrenamiento de la clase w_i presentes en la base de entrenamiento.

$$\hat{p}(x|w_i) = \frac{k_i}{N_i V}, i = 1, ..., M \tag{2.136}$$

donde N_i es el número de muestras de entrenamiento de la clase w_i . Las probabilidades *a posteriori* se computan con la fórmula de Bayes cómo se indica en 2.137.

$$p(w_i|x) = \frac{P(w_i)\hat{p}(x|w_i)}{\sum_{j=1}^{M} P(w_j)\hat{p}(w_i|x)} = \frac{P(w_i)k_i/N_i}{\sum_{j=1}^{M} P(w_j)k_j/N_j}, i = 1, ..., M \tag{2.137}$$

Considerando $\hat{P}(w_i) = \frac{N_i}{N}$, la computación de las probabilidades se simplifica a 2.138.

$$p(w_i|x) = \frac{k_i}{k}, i = 1, ..., M \tag{2.138}$$

Esto es, la probabilidad *a posteriori* de una clase es, sencillamente, su fracción de vecinos más próximos en las muestras de entrenamiento y la decisión es seleccionar la clase con mayor número de vecinos-más-cercanos al patrón de entrada. Es la regla K-NN.

Por ejemplo, cuando $k = 1$ clasificamos el patrón de entrada a la clase con la muestra de entrenamiento más cercana al patrón de entrada. Se muestra en el texto ya citado de [17] que, para este caso 1-NN, cuando el número de muestras de entrenamiento tiende a infinito el error que se comete está acotado según la expresión:

$$P^* \leqslant P \leqslant P^* \left[2 - \frac{M}{M-1} P^*\right] \tag{2.139}$$

donde P^* es el error bayesiano ya descrito y M el número de clases.

La calidad del resultado se ve degradada severamente por la presencia de ruido y características irrelevantes, o un escalado de las características que no es consistente con su importancia. Por eso se ha hecho un considerable esfuerzo en la selección y el correcto escalado de características para mejorar la clasificación *K-NN* (por ejemplo utilizando algoritmos evolutivos). En cualquier caso la calidad del resultado de los K-vecinos-mas-proximos lo hace propicio para usarse como método de referencia en comparativas, tal como haremos en este documento.

Hasta el momento los métodos no paramétricos no resultan adecuados para aplicaciones en tiempo real debido a su alto costo computacional, aunque puede mitigarse seleccionando cuidadosamente o construyendo prototipos más sintéticos de las muestras de entrenamiento para ahorrar computo. Además se ha demostrado que, si es aceptable tolerar cierto error (retornando un punto que podría no ser el vecino más próximo, pero que no está excesivamente lejos

del vecino efectivamente más próximo), el tiempo de cómputo puede mejorarse considerablemente con alternativas que aproximan el método[98, 99]. Una implementación práctica de estas ideas puede encontrarse en [100].

2.7.2. Redes neuronales artificiales

Originalmente las redes neuronales artificiales se desarrollaron con la esperanza de recrear las capacidades perceptivas y cognitivas del cerebro humano mediante la simulación de la estructura física de sus homólogas naturales. Si bien hasta el momento su éxito ha sido limitado en su propósito original, con el tiempo sus principios y algoritmos se han revelado útiles para la resolución de problemas asociados a procesamiento de señales y al reconocimiento de patrones. Será este último aspecto donde nos centraremos aquí. Para una descripción exhaustiva remitirse a [101].

2.7.2.1. Redes y neuronas

Una red neuronal artificial está compuesta de un número de neuronas artificiales(también denominadas unidades o nodos) interconectadas. La forma en la que están interconectadas varía dependiendo del modelo que se trate, Así tenemos redes de propagación hacia adelante, redes recurrentes, redes autoorganizativas, etcétera. Las conexiones entre los nodos no solamente muestran la relación entre los nodos sino que también transmiten datos e información, llamadas señales o impulsos.

2.7.2.2. Perceptrón: Un modelo simple de neurona

Diseccionando una de estas neuronas artificiales encontramos un conjunto de señales de entradas con un conjunto de pesos asociados a cada entrada, una unidad sumadora y finalmente una función de activación que proporciona la salida(qué es individual) de la neurona. Ver a este respecto la figura 2.24.

Así la salida de una neurona se obtendrá como la suma pesada de una serie de entradas o características a las que suele aplicarse una entrada interna o bias. Ver 2.140.

$$v = \sum_{i=1}^{d} w_i x_i + b \tag{2.140}$$

En general, esta suma pesada no alimenta directamente la salida de la neurona sino que es modulada a través de una función de activación que generará finalmente la salida.

La función de activación puede ser lineal o no lineal. Una popular función de activación es la sigmoide o función logística. Ver 2.141.

$$g(a) = \frac{1}{1 + e^{-a}} \tag{2.141}$$

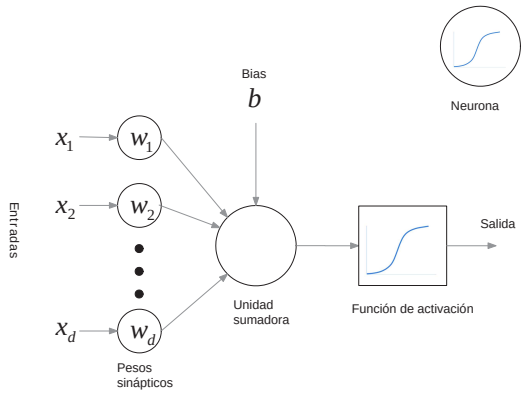


Figura 2.24: Modelo de una neurona

Los méritos que la hacen apropiada son:

1. Es continua y derivable de tal manera que el aprendizaje de gradiente es posible.
2. El valor de la sigmoide actúa como una probabilidad posterior lo que facilita la integración contextual de múltiples salidas de neurona.

La derivada de la función sigmoide es expresada en 2.142

$$g'(a) = g(a)\,[1 - g(a)] \tag{2.142}$$

Con la unidad básica de la neuronas construimos las estructuras que llamamos redes.

2.7.2.3. Red neuronal artificial de una sola capa

La estructura de una red neuronal artificial de una capa para clasificación multiclase se muestra en la figura 2.25.Las señales de entrada o características están conectadas a todas las neuronas de salida donde cada una corresponde a una clase. En el caso de funciones de activación lineales, la salida de cada neurona es una función discriminante lineal descrita en 2.143 donde w_{k0} es el término bias el cual puede ser visto como el peso a una señal fija $x_0 = 1$.

$$y_k(x) = v_k = \sum_{i=1}^{d} w_{ki}x_i + w_{k0} = \sum_{i=0}^{d} w_{ki}x_i = \boldsymbol{w}_k^T\boldsymbol{x}', k = 1, ..., M \tag{2.143}$$

Los pesos de cada neurona forman el vector de pesos (d+1)-dimensional $\boldsymbol{w}_k$ y las entradas junto con la señal fija conforman el vector $\boldsymbol{x}'$, también denominado vector mejorado (porque incluye la entrada interna de bias).

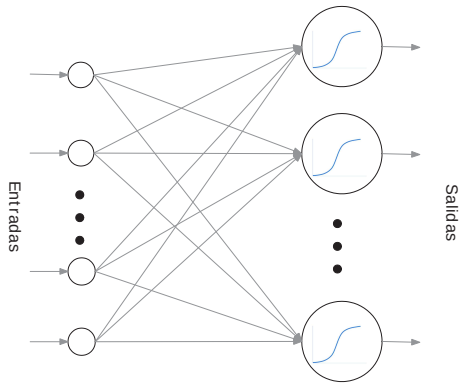


Figura 2.25: Red neuronal de una sola capa. Bias y pesos no son mostrados.

En este caso de función de activación lineal el patrón de entrada es clasificado a la clase de salida máxima. Así, usar la función de activación sigmoide no afecta la decisión de clasificación porque el orden de los valores de salida no cambia. Sin embargo la función de activación sigmoide puede explicarse como una probabilidad *a posteriori* y puede mejorar la eficiencia del aprendizaje, de ahí que sea preferible.

Descenso de gradiente Aunque la red neuronal artificial de una sola capa sólo puede proveer clasificaciones efectivas entre dos clases separables por un hiperplano, lo que limita su aplicabilidad, presentamos aquí el método de descenso de gradiente como introducción al clásico aprendizaje de retropropagación.

Si las salidas de las neuronas de la red son moduladas por la función de activación sigmoide tendremos como salida 2.144

$$y_k(x) = g(v_k) = \frac{1}{1 + e^{w_k^T x'}}, k = 1, ..., M \tag{2.144}$$

Dado un conjunto de muestras de entrenamiento $\aleph = \{(x^n, c^n) \,|n = 1, ..., N\}$ donde c^n denota la etiqueta de clase que corresponde con la muestra x^n,los valores de salida de cada muestra son computadas por la expresión 2.144 mientras que las salidas esperadas son expresadas con 2.145.

$$t_k^n = \begin{cases} 1, \mathrm{k} = \mathrm{c^n} \\ 0, \text{en caso contrario} \end{cases} \tag{2.145}$$

En este escenario la suma de errores cuadráticos del conjunto de entrenamiento es 2.146.

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{M} [y_k(x^n) - t_k^n]^2 \tag{2.146}$$

Los pesos de la red son optimizados tratando de minimizar esta suma de errores cuadráticos. Para hacerlo los pesos y bias son ajustados iterativamente (t). Con el descenso de gradiente se busca minimizar la función buscando en la dirección negativa del gradiente tal como se muestra en la expresión 2.147. En esta expresión $\eta(t)$ es el ratio de aprendizaje que puede variar de una iteración a otra.

$$w_k(t+1) = w_k(t) - \eta(t)\frac{\partial E}{\partial w_k},\ k = 1, ..., M \tag{2.147}$$

En la expresión 2.148 se deduce la computación de las derivadas parciales derivando 2.146 que contiene 2.144 aprovechando 2.142 y la regla de la cadena.

$$\frac{\partial E}{\partial w_k} = \sum_{n=1}^{N}\sum_{k=1}^{M}\left[y_k(x^n) - t_k^n\right] y_k(x^n)\left[1 - y_k(x^n)\right] x'^n, k = 1, ..., M \tag{2.148}$$

donde x'^n es el vector mejorado ya mencionado.
De esta forma el procedimiento de aprendizaje de gradiente será como sigue:

1. Inicialmente los parámetros son inicializados a valores aleatorios muy pequeños con lo que tendremos $w_{ki}(0)$ y $w_{k0}(0)$ para cada neurona i.
2. La red es alimentada con las muestras del conjunto de entrenamiento de una en una.
3. Se computan las derivadas parciales según 2.148 y se actualizan los pesos como está descrito en 2.147.
4. Se repite los pasos 2(alimentar muestras) y 3(actualizar pesos) hasta que la función objetivo no cambie.

La convergencia del aprendizaje puede ser acelerada con la aproximación estocástica[102] la cual actualiza los parámetros cada vez que la red es alimentada con una muestra en lugar de actualizar sólo tras alimentar a la red con todo el conjunto de entrenamiento.

Con la aproximación estocástica el error cuadrático es calculado con cada patrón de entrada tal como se ve en 2.149.

$$E^n = \frac{1}{2}\sum_{k=1}^{M}\left[y_k(x^n) - t_k^n\right]^2 \tag{2.149}$$

De acuerdo con esto, las derivadas parciales se computan con la expresión 2.150.

$$\frac{\partial E^n}{\partial w_k} = \sum_{k=1}^{M}\left[y_k(x^n) - t_k^n\right] y_k(x^n)\left[1 - y_k(x^n)\right] x'^n \tag{2.150}$$

Ahora los pesos(cada k) son actualizados con cada patrón de entrada (cada n) con la expresión 2.151 donde $\delta_k(x^n) = [t_k^n - y_k(x^n)]\, y_k(x^n)\, [1 - y_k(x^n)]$ es una señal de error generalizado.

$$w_k(t+1) = w_k(t) - \eta(t)\frac{\partial E^n}{\partial w_k} = w_k(t) + \eta(t)\delta_k(x^n)x'^n \tag{2.151}$$

Se ha probado que este algoritmo de aproximación estocástica converge a un mínimo local de E bajo las condiciones exhibidas en 2.152.

$$\begin{cases} \lim_{t\to\infty}\eta(t) = 0 \\ \Sigma_{t=1}^{\infty}\eta(t) = \infty \\ \Sigma_{t=1}^{\infty}\eta(t)^2 < \infty \end{cases} \tag{2.152}$$

Una elección que cumple estas condiciones es 2.153.

$$\eta(t) = \frac{1}{t} \tag{2.153}$$

2.7.2.4. Perceptrón multicapa

La red neuronal de una sólo capa descrita en 2.7.2.3 opera como una función discriminante lineal. Esto es; no importa como los pesos son aprendidos, sólo podremos obtener de esta red como frontera de decisión un hiperplano lineal entre dos clases. Así es incapaz de separar clases con distribuciones complicadas porque para esos casos la frontera decisión es generalmente NO lineal.

Un modo de mejorar la habilidad 'separatoria' de la red es usar múltiples capas. En este esquema los patrones de características alimentan la primera de las capas ocultas. Por otro lado las salidas de la red (el resultado de la clasificación) son proporcionadas por la capa de salida de la red. Puede haber cero mas de una capa oculta dispuestas una después de otra de tal manera que la anterior alimenta a la posterior.

Asumir una red perceptrón multicapa de dos capas como la mostrada en la figura 2.26 con d señales de entrada, m unidades ocultas y M unidades de salida. Denotaremos los pesos de la j-esima unidad oculta por w_{ji} y su salida con h_j . Así mismo denotaremos los pesos de la k-esima unidad de salida con w_{kj} y su salida con o_k .

$$x_0 = 1 \tag{2.154}$$

$$h_0 = 1 \tag{2.155}$$

$$u_j(x) = \sum_{i=1}^{d} w_{ji}x_i + w_{j0} \tag{2.156}$$

$$v_k(x) = \sum_{j=1}^{m} w_{kj}h_j + w_{k0} \tag{2.157}$$

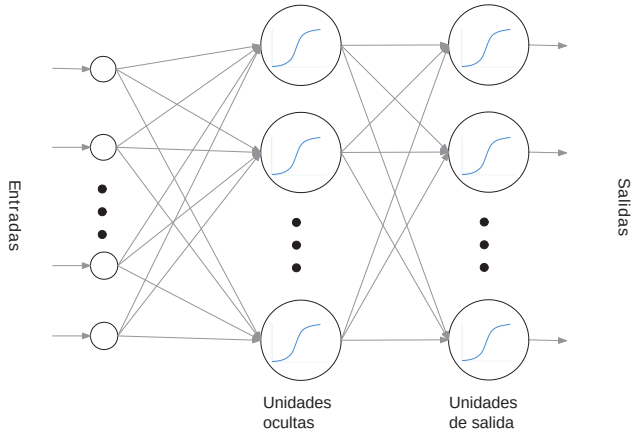


Figura 2.26: Red neuronal de más de una capa. Bias y pesos no son mostrados.

$$h_j(x) = g\left[u_j(x)\right] = g\left[\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right] \tag{2.158}$$

$$o_k(x) = g\left[v_k(x)\right] = g\left[\sum_{j=1}^{m} w_{kj}h_j + w_{k0}\right] \tag{2.159}$$

$$\begin{aligned} y_k(x) = o_k(x) &= g\left[\sum_{j=1}^{m} w_{kj}h_j + w_{k0}\right] = \\ &= g\left[\sum_{j=1}^{m} w_{kj}\left(g\left[\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right]\right) + w_{k0}\right], k = 1, ..., M \end{aligned} \tag{2.160}$$

Retropropagación Recordando el descenso de gradiente estocástico ya comentado, la red es alimentada con las muestras de forma iterativa ajustando los pesos para cada patrón de entrada en cada iteración tal como se indica en 2.161 tratando de minimizar el error cuadrático[29] con la salida esperada.

$$\begin{aligned} w_i(t+1) &= w_i(t) + \Delta w_i(t) \\ \Delta w_i(t) &= -\eta(t)\frac{\partial E^n}{\partial w_i} \end{aligned} \tag{2.161}$$

donde $w_i$representa a cualquiera de los pesos o bias.

[29] U otro criterio de error.

Para computar las derivadas parciales con respecto a los pesos ajustables ya hemos visto la expresión 2.150 en el descenso por gradiente para una red de una sola capa con función de activación sigmoide (como también ocurre aquí)

$$\frac{\partial E^n}{\partial w_i} = \sum_{k=1}^{M} [y_k(x^n) - t_k^n] \frac{\partial y_k(x^n)}{\partial w_i} = \\ = \sum [y_k(x^n) - t_k^n] y_k(x^n) [1 - y_k(x^n)] \frac{\partial v_k(x^n)}{\partial w_i} = \\ = -\sum_{k=1}^{M} \delta_k(x^n) \frac{\partial v_k(x^n)}{\partial w_i} \quad (2.162)$$

Ahora para calcular cada $\frac{\partial v_k(x^n)}{\partial w_i}$, derivamos 2.157 respecto a w_{kj} obteniendo la expresión 2.163.

$$\frac{\partial v_k(x)}{\partial w_{kj}} = h_j, j = 0, ..., m \quad (2.163)$$

Derivando respecto a w_{ji} la expresión 2.157 tenemos 2.164.

$$\frac{\partial v_k(x)}{\partial w_{ji}} = w_{kj} \frac{\partial h_j}{\partial w_{ji}} = w_{kj} h_j(x) [1 - h_j(x)] x_i, j = 1, ..., m; i = 0, ..., d \quad (2.164)$$

Combinando las ecuaciones

$$\Delta w_{kj}(t) = \eta(t) \delta_k(x) h_j, k = 1, ..., M; j = 0, ..., m; \quad (2.165)$$

$$\Delta w_{kj}(t) = \eta(t) \sum_{k=1}^{M} w_{kj} \delta_k(x) h_j(x) [1 - h_j(x)] x_i, ... = \\ = \eta(t) \delta_j(x) x_i, j = 1, ..., m; i = 0, ..., d \quad (2.166)$$

donde $\delta_j(x) = \sum_{k=1}^{M} w_{kj} \delta_k(x) h_j(x) [1 - h_j(x)]$ es la señal de error que se retropropaga desde la capa salida a la capa oculta.

Para una red con más de una capa oculta el error se retropropaga de esta misma manera entre capa y capa y las actualizaciones de los pesos se llevarán a cabo como se ha descrito.

Acelerando la retropropagación Aunque se garantiza que los pesos de la red converjan a un mínimo local, la búsqueda puede complicarse por la aparición de mesetas, que demoran la convergencia, o mínimos demasiado locales que den resultados suboptimos.

Una forma de enfrentar el problema de los mínimos suboptimos es entrenar la red múltiples veces con conjuntos de pesos iniciales diferentes. Es de esperar que converjan a diferentes mínimos locales así que se puede escoger el mejor. Esto es, el que hace el error cuadrático más pequeño.

En el caso de las mesetas, los pesos apenan cambian de iteración a iteración porque la actualización es prácticamente nula. Esto es así porque el gradiente en la superficie de error es cero, o casi cero. Para sortear este problema resulta efectivo aplicar un *momentum* en la actualización de pesos lo que también ayuda a sortear mínimos insignificantes. La idea es aprovechar alguna fracción (γ) de la inercia de pasadas actualizaciones aunque el gradiente sea prácticamente cero en este momento. Con un coeficiente de *momentum* $0 < \gamma < 1$ se modifica la actualización de pesos a la expresión 2.167.

$$\Delta w_{kj}(t) = \eta(t)\frac{\partial E^n}{\partial w_i} + \gamma \Delta w_i(t-1) \tag{2.167}$$

Por otro lado el descenso de gradiente estocástico puede requerir muchas iteraciones de alimentación de muestras de entrenamiento y actualización de pesos para alcanzar una convergencia aceptable a un mínimo local suficientemente bueno. Para conseguir un entrenamiento más rápido reduciendo significativamente el número de iteraciones puede utilizarse las derivadas de segundo orden en la actualización de pesos. Para ello ver 2.168 donde H es la matriz Hessiana construida con las derivadas de segundo orden de la función de error con respecto a los pesos $H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$.

$$w(t+1) = w(t) - \eta(t) H^{-1} \nabla E \tag{2.168}$$

$$\Delta w_i = -\eta \left(\frac{\partial^2 E}{\partial w_i^2} + \lambda \right)^{-1} \frac{\partial E}{\partial w_i} \tag{2.169}$$

2.7.2.5. Redes de funciones de base radial (RBF)

Las redes de funciones de base radial tienen una sola capa oculta, donde cada nodo ejecuta un procesamiento local no lineal, comúnmente una función gaussiana. Las salidas de los nodos ocultos son combinadas linealmente por los nodos de salida. Dichos nodos de salida pueden tener una función de activación lineal o sigmoide. La sigmoide facilita el entrenamiento por descenso de gradiente ya descrito, y hace que los valores de salida se aproximen a probabilidades *a posteriori*. Ver 2.27.

Para una red RBF con d señales de entrada, m unidades ocultas y M unidades de salida, asumamos 2.170como función en los nodos ocultos.

$$\phi_j(x) = \exp\left(\frac{\|x - \mu_j\|^2}{2\sigma_j^2} \right), j = 1, ..., m \tag{2.170}$$

donde $\mu_j$es el centro de la gaussiana y σ_j^2 es la variancia.

Siguiendo la descripción, la salida se obtiene con la combinación lineal de los nodos ocultos (con $\phi_j(0) = 1$) tal que así:

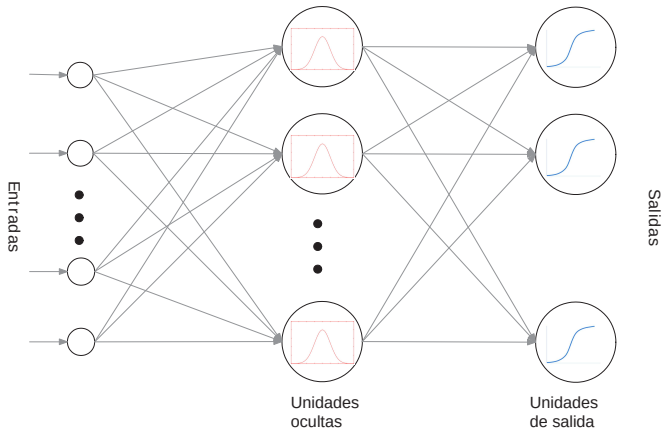


Figura 2.27: Red neuronal RBF. Neuronas ocultas con función de activación gaussiana y neuronas de salida con función de activación sigmoide. Bias y pesos no son mostrados

$$y_k(x) = v_k(x) = \sum_{j=1}^{m} w_{kj}\phi_j(x) + w_{k0} = \sum_{j=0}^{m} w_{kj}\phi_j(x), k = 1, ..., M \qquad (2.171)$$

Para nuestra exposición nos inclinamos por aplicar sobre esta combinación lineal una función de activación sigmoide cuya salida, sería, esta vez sí, la salida de la red:

$$y_k(x) = g(v_k(x)) = \frac{1}{1 + e^{-v_k(x)}} \qquad (2.172)$$

El procedimiento sería tal que así:

1. Se lleva a cabo un proceso de agrupamiento o clustering de las muestras de entrenamiento.

2. El centro de cada agrupamiento o clustering será el centro de una gaussiana y su variancia es promediada de las muestras que componen el agrupamiento. Estos valores se mantienen fijos.

3. Los pesos de la capa de salida son estimados por minimización del error cuadrático.

Fijadas las funciones gaussianas, los pesos de este nivel pueden ser estimados por la técnica ya descrita de descenso de gradiente:

$$w_{kj}(t+1) = w_{kj}(t) - \eta(t)\frac{\partial E^n}{\partial w_{kj}} \qquad (2.173)$$

donde:

$$\frac{\partial E^n}{\partial w_{kj}} = [y_k(x^n) - t_k^n]\, y_k(x^n)\, [1 - y_k(x^n)]\, \phi_j(x^n) = -\delta_k(x^n)\phi_j(x^n) \qquad (2.174)$$

Este entrenamiento en dos etapas de las redes RBF es rápido pero no lleva necesariamente a una calidad elevada en la clasificación porque la estimación de los centros (por clustering) no considera la separabilidad de los patrones de diferentes clases. Una alternativa es ajustar todos los parámetros simultáneamente minimizando el error bajo entrenamiento supervisado. Al proceder así las redes RBF pueden mejorar considerablemente la calidad de la clasificación con muchos menos nodos ocultos. Así las redes RBF pueden competir o incluso superar a las redes MLP.

Como en la retropropagación(2.7.2.4) los parámetros de la RBF son actualizados simultáneamente para minimizar el error cuadrático en el conjunto de muestras de entrenamiento.

El procedimiento sería tal que así:

1. Se lleva a cabo un proceso de agrupamiento o clustering de las muestras de entrenamiento (tal como hacíamos antes en el entrenamiento de dos etapas).

2. El centro de cada agrupamiento o clustering será el centro de una gaussiana y su variancia es promediada de las muestras que componen el agrupamiento (tal como hacíamos antes en el entrenamiento de dos etapas, sólo que ahora los valores no permanecerán fijos).

3. Los pesos de la capa de salida son actualizados tal como quedó descrito en la expresión 2.173.

4. Al mismo tiempo los centros y variancias($\tau_j = \sigma_j^2$) de las gaussianas se actualizan con:

$$\mu_j(t+1) = \mu_j(t) - \eta(t)\frac{\partial E^n}{\partial \mu_j} \qquad (2.175)$$

$$\tau_j(t+1) = \tau_j(t) - \eta(t)\frac{\partial E^n}{\partial \tau_j} \qquad (2.176)$$

donde:

$$\frac{\partial E^n}{\partial \mu_j} = -\sum_{k=1}^{M} w_{kj}\delta_k(x^n)\frac{\partial \phi_j(x^n)}{\partial \mu_j} \qquad (2.177)$$

$$\frac{\partial E^n}{\partial \tau_j} = -\sum_{k=1}^{M} w_{kj}\delta_k(x^n)\frac{\partial \phi_j(x^n)}{\partial \tau_j} \qquad (2.178)$$

siendo:

$$\frac{\partial \phi_j(x^n)}{\partial \mu_j} = \frac{\phi_j(x^n)}{2\tau_j}(x^n - \mu_j) \tag{2.179}$$

$$\frac{\partial \phi_j(x^n)}{\partial \tau_j} = \frac{\phi_j(x^n)}{2\tau_j^2}\|x^n - \mu_j\|^2 \tag{2.180}$$

2.7.2.6. Consideraciones finales

Las redes neuronales parecen haber sido desplazadas en los últimos años por técnicas como las máquinas de vectores soporte(SVM). Con todo, en algunos escenarios de clasificación ellas pueden ofrecer niveles de calidad comparables a técnicas más recientes con mucha menos complejidad (en entrenamiento y operación).

2.7.3. Máquinas de vectores soporte[30]

Una máquina de vectores soporte (SVM)[103, 104] clasifica construyendo un hiperplano n-dimensional que separa los datos en dos categorías.

En otras palabras, dado un universo de casos susceptibles a una categorización objetivo, la meta de un SVM es encontrar un hiperplano óptimo que separe vectores de características asociados a cada caso de tal modo que queden a un lado del hiperplano los vectores que corresponden a casos de una categoría y al otro lado aquellos vectores que correspondan a casos de la otra categoría.

El modelo de máquinas de vectores soporte es un pariente cercano al perceptrón multicapa de las redes neuronales artificiales clásicas. De hecho un modelo SVM usando una función kernel sigmoide es equivalente a una red neural perceptrón de dos capas.

2.7.3.1. Margen máximo

Para ilustrar esto gráficamente consideremos un simple caso de dos dimensiones donde los casos son caracterizados con vectores características de dos componentes. En este ejemplo operamos con casos con una caracterización objetivo de dos categorías.

En este ejemplo (gráficamente expuesto con 2.28) podemos trazar un número infinito de líneas candidatas (planos 1-dimensionales) que separarían las dos categorías con igual éxito. La cuestión es cual línea es mejor, y como definir la línea optima. Es posible extender esta misma idea a espacios de características d-dimensionales (Ver 2.29, 2.30 y 2.30) donde las clases pueden ser separadas linealmente.

La función lineal de decisión queda determinada en 2.140donde w es el vector de pesos y b es un *bias* o umbral.

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{2.181}$$

[30]Las máquinas de vectores soporte pueden ser utilizadas para clasificación y regresión. Nosotros consideraremos aquí sólo el aspecto de la clasificación.

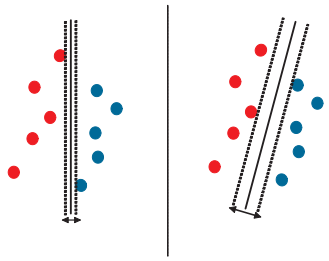


Figura 2.28: Un ejemplo en dos dimensiones, donde la separación en dos categorías es posible realizarla con una línea (un plano 1-dimensional). A izquierda y derecha dos posibles separaciones igualmente validas expresadas por una línea continua. La distancia entre las líneas discontinuas es denominada *margen*.

Figura 2.29: Un ejemplo en dos dimensiones, donde la separación en dos categorías no es posible realizarla con una línea (un plano 1-dimensional). Una mala representación deriva fácilmente en una clasificación difícil. No todos los análisis consisten en categorizaciones objetivo de dos categorías basadas en vectores características de dos componentes sino que podemos precisar más componentes. En estos espacios d-dimensionales las clases podrían ser separables linealmente(hiperplanos).

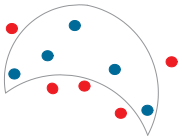


Figura 2.30: Separación compleja en dimensiones bajas. Tratar de separar clases puede ser bastante complicado si el espacio de características usado no es adecuado. Por ejemplo, si no considera todas las características oportunas.

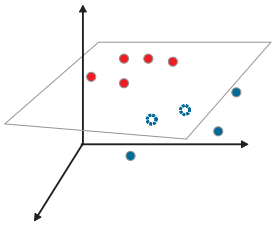


Figura 2.31: Separación simple en dimensiones altas. Un espacio de características adecuado puede favorecer enormemente la clasificación al considerar TODAS las características oportunas.

La clasificación es dada por $sgn[f(\mathbf{x})]$ donde $+1$ y -1 corresponden a dos clases. En el caso linealmente separable la función de decisión especifica un hiperplano que separa los puntos[31] de las dos clases. A este respecto ver 2.28 y 2.31. Para obtener una forma canónica de un hiperplano $\boldsymbol{w}$ y b son reescalados de tal manera que los puntos más cercanos al hiperplano de ambas clases satisfacen $|\mathbf{w} \cdot \mathbf{x} + b| = 1$. Así para todos los puntos $\boldsymbol{x}_i, i = 1, ..., l$ con etiquetas $y_i \in \{+1, -1\}$ se cumplirá que $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1$. La distancia entre los puntos de las dos clases más próximas al hiperplano será $2/\|\boldsymbol{w}\|$ y es conocida como el margen de el hiperplano.

La función de decisión de SVM estimada desde un conjunto de entrenamiento $\{(\boldsymbol{x}_i, y_i)|i = 1, ..., l\}$es formulada en 2.182

$$f(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \tag{2.182}$$

donde $k(\boldsymbol{x}, \boldsymbol{x}_i)$ es una función de kernel que implícitamente define un espacio de características expandido:

$$k(\boldsymbol{x}, \boldsymbol{x}_i) = \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{x}_i) \tag{2.183}$$

donde $\Phi(x)$ es un vector de características en el espacio de características expandido y podría tener dimensionalidad infinita. En nuestro espacio lineal de características lineal tenemos $k(\boldsymbol{x}, \boldsymbol{x}_i) = \boldsymbol{x} \cdot \boldsymbol{x}_i = \boldsymbol{x}^T \cdot \boldsymbol{x}_i$ y la ecuación 2.182 es equivalente a 2.184.

$$f(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i (\mathbf{x} \cdot \boldsymbol{x}_i) + b = \left(\sum_{i=1}^{l} y_i \alpha_i \boldsymbol{x}_i \right) \cdot \boldsymbol{x} + b = \mathbf{w} \cdot \mathbf{x} + b$$

$$, \text{ donde } \mathbf{w} = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{x}_i \tag{2.184}$$

[31] Muestras o ejemplos objetos de clasificación.

Los coeficientes $\alpha_i, 1, ..., l$ son estimados desde las muestras de entrenamiento con el objetivo de maximizar el margen con restricciones de separación lineal. Esto es; 2.185.

$$\text{Minimizar } \tau(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{, sujeto a } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1,\ i = 1, ..., l. \quad (2.185)$$

Esto corresponde a un problema de programación cuadrática(QP)[32] en su forma primitiva que puede ser convertido al problema dual expresado en 2.186 introduciendo multiplicadores de Lagrange.

$$\text{Maximizar } W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$$
$$\text{, sujeto a } \alpha_i \geq 0,\ i = 1, ..., l \text{ y } \sum_{i=1}^{l} \alpha_i y_i = 0 \quad (2.186)$$

Los coeficientes(multiplicadores) $\alpha_i, 1, ..., l$ pueden ser obtenidos resolviendo el problema QP citado. El bias b se determina satisfaciendo las condiciones complementarias Karush-Kuhn-Tucker(KKT) expresadas en 2.187.

$$\alpha_i \left[y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 \right] = 0,\ i = 1, ..., l \quad (2.187)$$

De estas condiciones los coeficientes α_i deben ser 0 para esos puntos con $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) > 1$(puntos no fronterizos). Los puntos de frontera, con $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$ y $\alpha_i > 0$ son los llamados vectores soporte que dan nombre a este método de clasificación. En realidad el bias es calculado precisamente por las condiciones KKT en estos vectores soporte.

Ver más sobre esto en [19].

2.7.3.2. Márgenes relajados y 'kernels'.

La presunción de que todos los puntos/muestras/ejemplos de entrenamiento son separables es frecuentemente violada en la práctica. La restricción de margen descrita en la ecuación 2.185es relajada a la expresión 2.188.

$$\text{sujeto a } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0,\ i = 1, ..., l \quad (2.188)$$

La variable ξ_i es no cero para puntos no separables. Ahora el objetivo de maximizar es modificado de acuerdo a este margen "relajado" en 2.189

$$\text{Minimizar } \tau(\boldsymbol{w}, \xi) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{l} \xi_i \quad (2.189)$$

[32] La 'programación cuadrática' es un tipo especial de problema de optimización matemática: Es el problema de optimizar (maximizando o minimizando) una función cuadrática de varias variables sujeta a restricciones lineales en esas variables.

Figura 2.32: Separación muy difícil con un determinado kernel

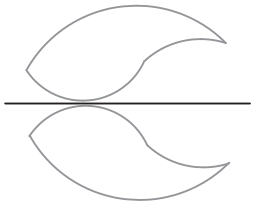


Figura 2.33: Separación trivial seleccionando el kernel adecuado

Por otra parte para clasificación en un espacio de características no lineal, usamos funciones de kernel no lineales $\boldsymbol{k(x, x_i)}$ que reemplazan el producto $\boldsymbol{x} \cdot \boldsymbol{x}_i$. Ver figuras 2.32 y 2.33 .El problema de aprendizaje de una máquina de vector soporte (SVM) en un espacio de características inducido por un kernel general es entonces formulado como:

$$\begin{aligned} \text{Minimizar } \tau(\boldsymbol{w}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i, \\ \text{sujeto a } y_i(\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_i + b) &\geq 1 - \xi_i, \xi_i \geq 0,\ i = 1, ..., l \\ \text{donde } \boldsymbol{w} &= \sum_{i=1}^{l} y_i\alpha_i\Phi(\boldsymbol{x}_i) \end{aligned} \tag{2.190}$$

Introduciendo multiplicadores de Lagrange este problema QP queda convertido en el problema dual 2.191.

$$\begin{aligned} \text{Maximizar } W(\alpha) = \sum\nolimits_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{, sujeto a } 0 \leq \alpha_i \leq C, i = 1, ..., l \text{ y } \sum\nolimits_{i=1}^{l}\alpha_i y_i = 0 \end{aligned} \tag{2.191}$$

En esta expresión 2.191la constante C , también llamada 'parámetro de ruido' sirve como frontera superior de los multiplicadores y controla la tolerancia al error del aprendizaje.

Para aprendizaje SVM desde ejemplos, el parámetro C y la función de kernel $k(\boldsymbol{x}, \boldsymbol{x}_i)$ son especificados a priori, y los multiplicadores $\alpha_i, 1, ..., l$ son estimados

mediante la resolución del problema de optimización descrito en 2.191. El valor de b es entonces determinado desde las condiciones KKT de los vectores soporte. La selección acertada del parámetro C y la función de kernel no es una cuestión cerrada. La experiencia y el ensayo y error juegan un papel preponderante.

Las máquinas de vectores soporte tienen algunas propiedades atractivas:

1. Los multiplicadores de los puntos de entrenamiento son estimados en QP, lo cual garantiza encontrar el óptimo global porque la función objetivo cuadrática convexa de 2.191, restringida en la región del espacio de parámetros, no tiene óptimos locales.

2. Las SVM puede enfrentar además de la clasificación lineal (producto escalar de vectores como kernel lineal) clasificaciones no lineales introduciendo funciones de kernel diferentes, las cuales pueden ser diseñadas con cierta flexibilidad y pueden ser generalizas a representaciones de patrones no vectoriales. Algunas frecuentemente usados son el kernel sigmoide (ver 2.192), el kernel polinómico(ver 2.193) y el kernel RBF(ver 2.194).

3. La optimización QP de SVM resulta en una pobre representación de patrones porque sólo una fracción de puntos de entrenamiento tienen multiplicadores no nulos y son usados en la decisión. Más aún el número de vectores soporte es adaptable a la dificultad de separación de los puntos de entrenamiento. Esta propiedad lleva a buenas generalizaciones en la clasificación.

$$k(\boldsymbol{x}, \boldsymbol{x}_i) = \tanh(k(\boldsymbol{x} \cdot \boldsymbol{x}_i) + \theta) \tag{2.192}$$

$$k(\boldsymbol{x}, \boldsymbol{x}_i) = (k(\boldsymbol{x} \cdot \boldsymbol{x}_i) + 1)^P \tag{2.193}$$

$$k(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{2\sigma^2}\right) \tag{2.194}$$

2.7.4. Comparación de plantillas

2.7.5. El uso de diccionarios de palabras

Esta estrategia NO es aplicable en un monitor de validación. La filosofía en el uso de los diccionarios de palabras por parte de los OCR se fundamenta en el reconocimiento de la 'palabra' allí donde existe incertidumbre sobre uno o varios de sus caracteres que la componen. Pero un código de caducidad NO es una palabra. Y aunque lo fuera, no nos sirve de nada deducir, gracias a un diccionario, cual es. No queremos 'descubrirla'. Queremos validar que efectivamente está ahí.

2.8. Postproceso

La tercera fase, que en los últimos tiempos ha cobrado especial relevancia, gravita en torno a la necesidad de almacenar (y mostrar) correctamente los resultados. Por ejemplo aprovechando las características del uso de texto enriquecido digital. Hay cabida para variopintas políticas en esta linea y claramente la fidelidad al formato original puede ser una de ellas. Así empieza a cobrar importancia el propósito de respetar e imitar en la medida de lo posible los formatos originales de la imagen de texto. Esto supone un esfuerzo adicional de localización, identificación de fuentes, espacios sin texto (cómo espacio reservado a imágenes explicativas de texto), dobles columnas, etcétera[33]

En MONICOD el resultado del sistema por reconocimiento se limita a salida booleana: Validación Positiva/ Validación Negativa. Esta información bien puede engrosar alguna clase de estadística de positivos y negativos o servir como entrada a un algoritmo de decisión que desencadene acciones, pero en cualquier caso, queda fuera del ámbito principal de trabajo de MONICOD.

[33] Dado que MONICOD es un sistema de verificación de códigos y es prescindible cualquier salida formateada, esta tercera fase es irrelevante y será obviada.

Capítulo 3

El problema: Estudio preliminar

Donde se explica con algún detalle en qué consiste el problema

Nuestro estudio preliminar de proyecto comprende los siguientes objetivos:

- Descripción del universo del proyecto.
- Definición precisa de los requisitos de usuario.
- Definición precisa de los requisitos de la aplicación.

3.1. El universo de proyecto

El universo del proyecto es de naturaleza cerrada.

Se trata de un entorno industrial. Una **cadena de envasado** de **latas** donde una **cinta transportadora** transporta latas. El recorrido de la cinta pasa bajo una cabeza impresora que estampa (con **tinta**) un **código** en la **superficie** de la base de la lata. El código proporciona la fecha de caducidad del **producto** contenido en la lata. Un **operario** supervisa -en intervalos de tiempo mas o menos regulares- si la impresión se está llevando a cabo correctamente.

En el universo del proyecto aparecen varios elementos: lata, código de caducidad, tinta. sistema impresor, producto, cinta transportadora, operario.

Estos elementos (Ver figura 3.1) juegan un papel en la cadena de envasado y en MONICOD de manera que en los siguientes apartados se procederá a un estudio de cada uno de ellos.

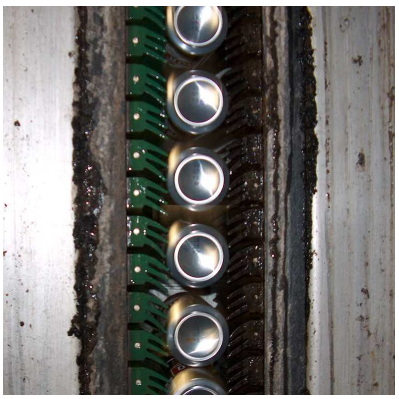



Figura 3.1: Cadena de envasado. Cinta Transportadora. Lata. Código de lata.

3.2. La lata[1]

3.2.1. Historia

Existen antecedentes de latas orientadas al almacenamiento de comida desde el siglo XIX. Nicolas Appert ("el padre del enlatado") presentó un método para preservar comida por esterilización en 1809. El propósito original -en el contexto de las guerras napoleónicas- era enfrentar el clásico problema logístico del transporte de alimentos para una tropa que debía cubrir grandes distancias. Múltiples patentes surgieron desde entonces. En 1880-1890 aparecieron las primeras máquinas capaces de fabricar latas (para enlatado de comida) de manera automatizada. Eso supuso un impulso masivo a la producción de comida enlatada[105].

Sin embargo, las primeras experiencias con éxito de producción de latas como recipiente contenedor de bebidas (particularmente cerveza) para el consumo no doméstico se remontan al periodo 1931-1933. Fueron llevadas a cabo por la compañía American Can (USA) que se anticipaba así al fin de La Prohibición. Ver figura 3.2.

No fue hasta 1952 cuando la idea terminó siendo adoptada universalmente por las cerveceras que a la postre se tornaron con el tiempo en las principales consumidores de latas. En 1957 se introdujo el aluminio en la fabricación de latas. En 1967 se desarrolló en USA, por la compañía Metal Box, un nuevo mecanismo de apertura -la que reconocemos hoy como apertura de anilla (Ver figura 3.3)- que incrementó sobremanera las ventas de latas frente a otros formatos: Ya no era necesario depender de ninguna clase de herramienta para abrir una lata que no fueran las manos[106].

[1] En este documento nos centramos en las latas destinadas a bebidas. Siempre que usemos el término "lata" nos referimos específicamente a esa "familia" de latas. Pero en propiedad, la industria manufacturadora de latas no distingue entre estas y las latas destinadas a alimentos o a otros usos.

Figura 3.2: Latas históricas

Figura 3.3: La apertura de anilla marcó un hito en la historia de las latas.

3.2.2. ¿Por qué latas?

La lata tenía muchas ventajas sobre la botella de cristal, al menos hasta la llegada de la botella de plástico. Era mas ligera, ocupaba menos espacio, no tenía que ser "retornada" y no se rompía fácilmente por impacto Además es inviolable, resistente a roturas y favorece la refrigeración del contenido .

3.2.3. Especificaciones

Los principales consideraciones que han llevado al diseño actual cilíndrico en la fabricación de latas son[107]:

1. La razón volumen/superficie es óptima en una esfera.
2. La forma óptima de un contenedor susceptible de ser transportado es una caja o un cubo (buena utilización del espacio).
3. La típica forma de vasija es apropiada para la presión interna.
4. Un contenedor de dos piezas requiere menos material que un contenedor de tres piezas.

5. Es posible reducir el grosor de la lata de manera efectiva y segura (lo que conlleva un ahorro de material) desplazando la tarea de pintado después de haber forjado las bobinas de metal con las que fabricarán las latas. Esto implica desplazar tareas desde el fabricante de material al productor de latas.

6. El ratio volumen/superficie óptimo de un cilindro matemático es obtenido de igualar altura (h) y diámetro($D = 2r$) para tratar de obtener un cilindro que se "parezca" tanto como sea posible a un cubo, por la consideración 2. Por tanto forzamos $h = D = 2r$.

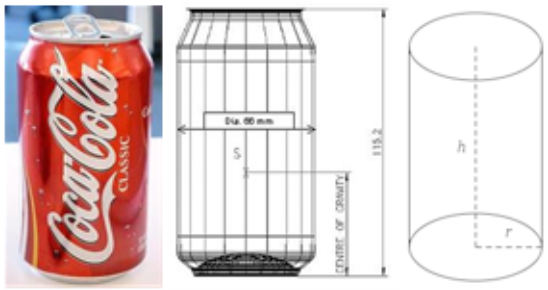


Figura 3.4: Latas y cilindros

El volumen y superficie de un cilindro viene dadas por las expresiones respectivas 3.1 y 3.2. Ver figura 3.4:
El volumen de un cilindro viene dado por la expresión:

$$V = \pi r^2 h \tag{3.1}$$

El área de superficie de un cilindro viene dado por la expresión.

$$S = A_t = A_l + 2 \times A_b = 2\pi r \cdot h + 2 \times (\pi r^2) = 2\pi r(h + r) \tag{3.2}$$

Siendo A_t el área total, A_l el área lateral y A_b el área de la base.
Forzamos:

$$h = D = 2r \tag{3.3}$$

Y obtenemos:

$$V = \pi r^2 h = \pi r^2 (2r) = 2\pi r^3 \tag{3.4}$$

Si queremos que contenga *330 cl* simplemente debemos despejar de la expresión:

$$2\pi r^3 = \frac{\pi}{4} D^3 = 330 \tag{3.5}$$

Resolvamos el problema de la superficie mínima de un recipiente con volumen de 1 litro:

$$V = \pi r^2 h = 1 \tag{3.6}$$

Con lo que:

$$h = \frac{1}{\pi r^2} \tag{3.7}$$

Ya sabemos que la superficie es:

$$S = 2\pi r h + 2\pi r^2 \tag{3.8}$$

Así que:

$$S = 2\pi r^2 + \frac{2}{r} \tag{3.9}$$

Para obtener la superficie óptima igualamos la derivada de superficie a cero y obtenemos r:

$$\frac{dS}{dr} = 4\pi r - \frac{2}{r^2} = 0 \tag{3.10}$$

$$4\pi r = \frac{2}{r^2} \tag{3.11}$$

$$r^3 = \frac{1}{2\pi} \tag{3.12}$$

Y con esto obtenemos r y h:

$$r = \left(\frac{1}{2\pi}\right)^{\frac{1}{3}} \tag{3.13}$$

$$h = \frac{1}{\pi r^2} \tag{3.14}$$

Estos sencillos cálculos son útiles sólo para construir los contornos básicos de la lata y no toman en cuenta importantes consideraciones como las fluctuaciones en el grosor del material o la presencia de junturas para unir las piezas de la lata.

Hay que considerar aparte otros factores tales como:

- El cuello de lata apertura de anilla.
- La geometría del extremo de la lata (que no es plana)
- Estándares internacionales aplicables para máquinas expendedoras de latas, paletizado para transporte, envoltorios de plástico,...

Figura 3.5: Las latas que MONICOD debe tratar.

Otro criterio de diseño importante no considerado aquí es la minimización del consumo de material.

En lo que se refiere a MONICOD es deseable evitar que el sistema sea demasiado dependiente de las características particulares de las latas en aras de la generalización. Ver figura 3.5 y 3.6.

Por otra parte, la adaptabilidad a las variaciones de la lata implica aumentar la complejidad del sistema (que presumíblemente redundará negativamente en la velocidad de proceso). Ver a este respecto el capítulo 4. Por fortuna la forma de la lata parece mantenerse estable durante los últimos años y no hay razones para sospechar cambios inminentes en ella.

3.2.4. Material

Las latas de bebidas están fabricadas con hojalata (delgada capa de acero de bajo contenido en carbono recubierta de estaño) o con aluminio. Ambos materiales son resistentes a la oxidación si bien la hojalata resulta más barata que el aluminio. Una vez se dispone del material listo para usarse, aluminio u hojalata, el proceso de fabricación de la lata es prácticamente idéntico. Son envases muy ligeros cuya composición garantiza la protección del contenido durante un largo periodo ante la entrada de aire y luz. Son resistentes y por su peso y tamaño manejables y fáciles de transportar y almacenar. Asimismo, el material permite que el contenido se enfríe rápidamente.

Tanto el aluminio como el acero son metales que al fundirse recuperan todas sus propiedades y pueden reutilizarse casi ilimitadamente mediante su recicla-

Figura 3.6: Lata de 220 cl (más alta) frente a lata de 330 cl (más baja)

do[2]. Se pueden convertir de nuevo en envases o en cualquier producto metálico destinado a cualquier sector industrial, maquinaria, transporte, construcción, etc.

El material determina el color de la superficie donde el código de caducidad es impreso. La información de color es probablemente la más valiosa de la adquisición porque, como quedará descrito más adelante, los caracteres no son más que manchas de color sin relieve en la base de la lata. Hay dos colores en particular que redundan muy directamente en el sistema: El color de fondo de la lata (que vendrá dado por el material), y el color de la tinta utilizada en la impresión del código.

3.2.5. Ciclo de vida

El ciclo de vida de la lata (ver 3.7) empieza con la producción del metal con el cual se fabrican las latas. Ese metal (ya sea aluminio u hojalata) es embobinado para facilitar su uso y transporte a las manufacturadoras de latas. La lata se produce, se almacena y se transporta a la fábrica de envasado. Esta se ocupa de llenar la lata con el producto en cuestión y, en ocasiones, también se ocupa de la decoración externa de la lata. A partir de aquí las latas pasan a la red de venta para el consumo. En los últimos años el reciclado de latas ha cobrado suficiente importancia para ser considerado parte activa del ciclo de vida de una lata común. Durante el tratamiento del reciclado las latas se clasifican según material y se transforman en materia prima destinada a usos variopintos como

[2]Una lata de acero puede tardar en descomponerse 10 años. Y una de aluminio de 200 a 300 años.

Figura 3.7: El ciclo de vida de las latas.

Tipo de lata	Altura	Radio	Radio Base
Lata de 330cl Hojalata	11,5cm	3,25cm	2,5cm
Lata de 220cl Hojalata	?	?	?
Lata de 330cl Aluminio	11,5cm	3,25cm	2,5cm
Lata de 220cl Aluminio	13,5cm	2,7cm	2,4cm

Cuadro 3.1: Especificaciones de latas

la fabricación de llantas de bicicleta[3], ceniceros o de nuevo... ¡latas!

La impresión del código de caducidad es una fase más dentro de la etapa de llenado, pero a una lata le ocurren muchas otras cosas. También puede ocurrir que la lata pase de nuevo bajo el sistema de impresión si la primera vez el código no fue correctamente impreso.

3.2.6. Instancias de latas sobre las que opera MONICOD

En el cuadro 3.1 se especifican las dimensiones cilíndricas de las latas sobre las que MONICOD ha sido aplicado.

Obsérvese que aparecen dos radios: 'Radio' y 'Radio Base'. Corresponde a dos circunferencias concéntricas en la base de la lata. En la figura 3.8 queda ilustrado. La circunferencia a la que corresponde el radio de base resulta ser la superficie de material que está en contacto con la superficie plana sobre la que suelen estar las latas. El área de impresión está libre de contacto y eso contribuye a ralentizar la degradación de la impresión.

No se ha contando con ningún ejemplo de lata de 220cl de hojalata.

[3] Así, con 80 latas de bebidas se puede fabricar una llanta de bicicleta.

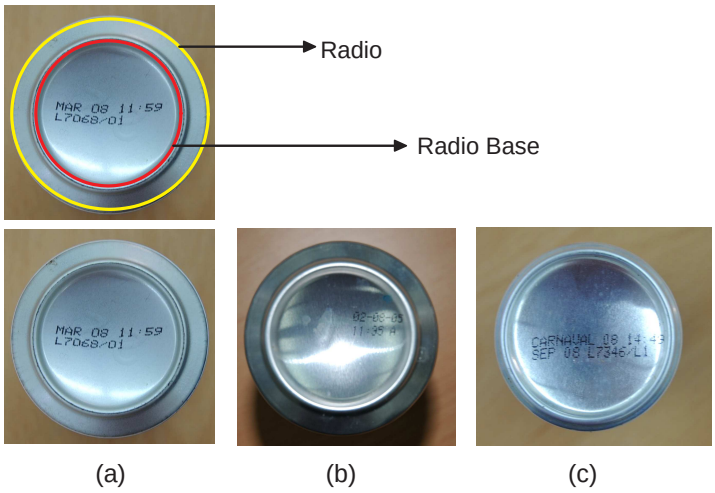


Figura 3.8: Radios de base de lata. (a) Lata de hojalata de 330cl. (b) Lata de aluminio de 330cl. (c) Lata de aluminio de 220cl.

3.3. ¿Qué es el código de caducidad[4]?

El "código de caducidad" es la denominación habitual para un conjunto de caracteres (números y letras) impresos en un recipiente con el propósito de alertar al consumidor sobre cuando concluye la vida útil del producto contenido. Dicho tiempo límite se calcula bajo la suposición de que el recipiente se mantiene correctamente almacenado en todo momento. Ver a este respecto la legislación vigente en 1.7.

La necesidad de un código de caducidad en la industria alimentaria resulta bastante natural (ver figura 3.9): Todos los alimentos orgánicos (y algunos no orgánicos) están sujetos a un factor de degradación. Cuando un alimento se deteriora en exceso es contraproducente su consumo. Puede incluso tornarse tóxico llegando a ser peligroso[5].

El código de caducidad hace accesible[6] al usuario final información sobre el momento a partir del cual consumir el producto deja de ser seguro. De otra forma el consumidor tendría serios problemas para saber algo tan elemental. El código de caducidad debe estar situado en un lugar visible, y ser legible durante periodos muy prolongados de tiempo. Se preserva así el derecho del consumidor a conocer cuando el producto deja de cumplir las especificaciones por las que

[4]El código de caducidad (*expiration date*) es diferente al código de frescura o fecha max. recomendada de consumisión (*shelf life*). El primero es una referencia a la seguridad, el segunda una referencia a la calidad. Un producto que ha sobrepasado la fecha recomendada puede seguir siendo seguro, pero su calidad ya no está garantizada. El marketing juega un papel importante en la presencia de una fecha de consumisión recomendada.

[5]En el caso de las bebidas los conservantes, el cierre hermético y el tratamiento térmico contribuyen a ampliar el plazo límite para el consumo.

[6]De manera inmediata y puntualmente.

Producto	Lugar	Duración
Leche (una vez abierta)	Frigorífico	3 – 4 días
Yogures y postres lácteos	Frigorífico	24 días
Queso	Frigorífico	10 – 12 días
Productos cárnicos	Frigorífico	3 días
Pescado fresco	Frigorífico	3 días
Zumos de fruta abiertos	Frigorífico	3 – 4 días
Pasta fresca y pizzas	Frigorífico	3 – 4 días
Helados	Congelador	Más de 6 meses
Galletas	Fresco y seco	Más de 6 meses
Chocolate	Fresco y seco	Más de 6 meses
Arroz, pasta, té, cacao	Seco y sin luz	Más de 6 meses
Latas y conservas	Seco sin luz y envase cristal	Más de 6 meses

Figura 3.9: Fechas de caducidad.

fue adquirido, y se previene un escenario de insalubridad.

Los códigos de caducidad se evalúan en el momento del envasado. Ver 3.10. La principal variable para el cálculo es la fecha de envasado. El factor de degradación ha sido previamente calculado[7] en un laboratorio para el producto en cuestión. A partir de ambos datos puede determinarse una fecha límite para el consumo del contenido.

En general, en una economía sensible hacia la salud de los consumidores, todo producto perecedero debe estar debidamente etiquetado por un código de caducidad. Las propiedades deseables de los códigos de caducidad son:

- Visibilidad
- Legibilidad
- Simplicidad
- Durabilidad
- No ambigüedad
- Estándar
- Independencia del lenguaje

La **visibilidad** obedece a la necesidad de acceder al código fácil y rápidamente para su lectura. Esto debe ser independiente de si se conoce o no el lugar de la impresión previamente.

[7] Se trata de una estimación a partir de un modelo matemático desarrollado a través de experiencias concretas. Las condiciones en las que el consumidor almacena el producto influyen decisivamente y así suele indicarse.

Figura 3.10: Laboratorio de control de caducidad de una empresa de alimentos.

La **legibilidad** hace referencia a la calidad de la impresión, que debe ser suficiente para obtener sin esfuerzo la información requerida del código de caducidad.

La **simplicidad** define cuan fácil es interpretar la información requerida desde la lectura del código de caducidad. Esto es, debe ser suficientemente simple como para permitir una lectura con éxito sin entrenamiento ni esfuerzo.

La **durabilidad** determina la perdurabilidad de la legibilidad de la impresión del código en condiciones normales. Es muy deseable que el código sea indeleble durante toda la vida útil del recipiente.

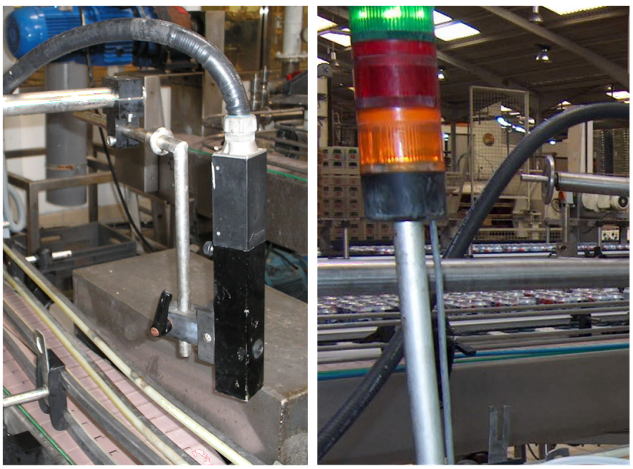
La **no ambigüedad** asegura que el código de caducidad esté expresado de tal manera que no sea posible errar al interpretar la lectura de caducidad. Es decir; no deben darse situaciones que toleren más de una interpretación.

El uso de un **estándar** favorece una lectura ágil.

Un código de caducidad **independiente del lenguaje** es una propiedad cada vez más relevante en los actuales tiempos de globalización creciente. El código de caducidad debe evitar elementos cifrados idiomáticamente. Así por ejemplo, un identificador castellanizado del primer mes del año podría ser 'ENE'. Su cifrado en inglés sería 'JAN'. Pero para ambas culturas lingüísticas el mes es numéricamente el mismo: '1'. Numerar los meses resulta más conveniente en la síntesis de una fecha de caducidad.

En cuanto a la visibilidad, la ubicación del código ya ha sido designada. Y no es el propósito de este trabajo evaluar las propiedades de simplicidad, durabilidad, no ambigüedad o el grado de estandarizado. La legibilidad es la característica que se espera medir.

En el apartado 3.8 se describe la construcción de los códigos de caducidad a



(a) (b)

Figura 3.11: Izquierda cabezal impresor. Derecha sistema 'discreto' de aviso de estado de impresora.

tratar por MONICOD.

3.4. El sistema impresor

El sistema impresor es el dispositivo que imprime los códigos de caducidad en las latas.

La tarea de impresión forma parte de la cadena de llenado. En las plantas de operación actuales para MONICOD es habitual que apenas pasen unos minutos entre el llenado efectivo de la lata y la impresión correspondiente en el envase con un código de caducidad "fresco".

La tarea del sistema impresor no es sencilla (ver figura 3.11): Debe hacer impresiones continuas sobre superficies irregulares más o menos húmedas que se desplazan en una cinta transportadora a velocidades más o menos considerables. Dichas impresiones son solicitadas por un dispositivo sensor (una célula fotoeléctrica en la figura 3.12) que detecta cuando hay una lata presente frente a la cabeza impresora.

El sistema de impresión también suele ejecutar el cálculo del código de caducidad en tiempo real. Dicho cálculo se lleva a cabo continuamente, de manera que dos latas inmediatamente consecutivas bien pudieran tener códigos de caducidad diferentes. Esto es así porque, tal como es descrito en la sección 3.3 el código de caducidad depende de la fecha y hora actual que, naturalmente, cambia constantemente. Los sistemas de impresión actuales ofrecen bastantes

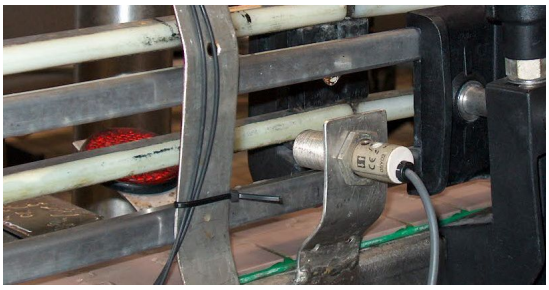


Figura 3.12: Célula fotoeléctrica detectora de latas.

Figura 3.13: Consola de Sistema impresor

posibilidades de configuración que incluyen opciones de aspecto como fuente de letra, estilo, tamaño... Ver figura 3.13.

3.5. La tinta

La tinta es la substancia que se utiliza para imprimir el código de caducidad. Ver 3.14.

Para asegurar legibilidad debe tener propiedades adherentes de larga duración, ofrecer un buen contraste, y resistencia a la humedad, a la temperatura y a la fricción.

El color de la tinta es su propiedad más valiosa. Es negra, color que es casi un estándar dentro de la industria de etiquetado.

La rapidez de asentamiento - y secado - de la tinta en la superficie de la lata

Figura 3.14: Deposito de tinta a la derecha.

una vez que se ha llevado a cabo la impresión es otro factor importante. En impresoras de inyección de tinta resulta casi instantáneo.

Tan importante como la tinta es el estado de almacenamiento para su uso. Después de una carga completa en el sistema de impresión, la calidad de la impresión es diferente a cuando la tinta está casi agotada.

MONICOD se limita a verificar la legibilidad del código instantes después de la impresión. Sera la calidad de la tinta, las propiedades de la superficie de la lata, y el entorno donde esté almacenada la lata las que determinen la durabilidad de esa legibilidad.

3.6. El producto

Los productos susceptibles a ser almacenados en latas son muy variados. MONICOD tratará latas de cerveza y refresco.

Las proporciones de los ingredientes (y la ausencia o presencia de estos) varían de refresco en refresco y de cerveza en cerveza.

El producto determina la fecha de caducidad y afecta de forma marginal a la propia impresión: El sistema de impresión está expuesto a vapores que emanan de la cadena envasado. Estos vapores son susceptibles de condensarse sobre la superficie de las latas. Se ha detectado que con sustancias azucaradas (propias de muchas variedades de refresco), la condensación sobre la base de la lata genera una película pegajosa que puede alterar el procedimiento de impresión produciendo resultados irregulares.

3.7. La cinta transportadora/elevadora en la cadena de envasado

La cita transportadora[8] se ocupa de mover las latas a través de la cadena de envasado donde son sometidas a diversos tratamientos. La cadena de envasado es el último viaje del recipiente en su manufactura. En el transcurso de ese viaje será tornado en un producto listo para el consumo final.

A grandes rasgos la cadena de envasado comprende diferentes etapas:

1. La lata se le aplicará una decoración que responde a la necesidad de identificación fácil por parte del usuario de la clase de contenido almacenado, la marca, etcétera...
2. La lata será enjuagada, llenada con el tipo de producto asignado y sometida a un tratamiento térmico para suprimir elementos patógenos.
3. La lata sera adecuadamente marcada con un código de caducidad.
4. Las lata serán empaquetadas en lotes para facilidad su transporte.

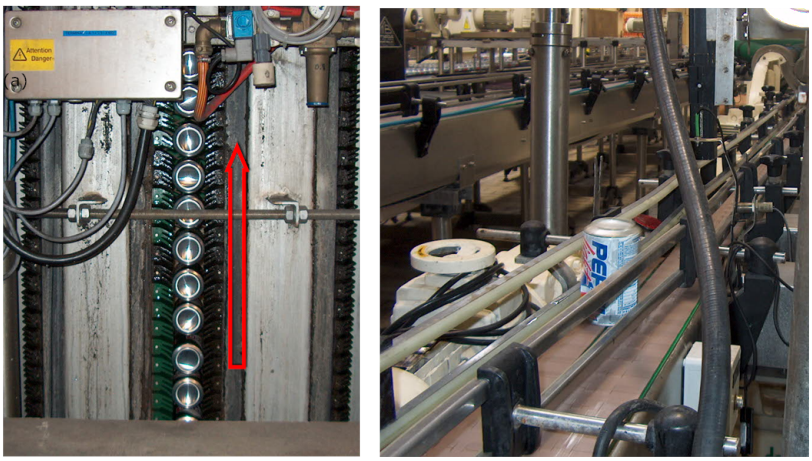
El orden específico de las etapas puede variar de una compañía a otra: 1, 2, 3 y 4 en la Compañía Embotelladora de Canarias y 3, 1, 2, 4 en la Compañía Cervecera de Canarias.

La aproximación habitual es que todo el trayecto se divida en secciones y dependiendo de la naturaleza de la sección se emplee un tipo de cinta transportadora. Nos limitaremos aquí a la sección de cinta transportadora/elevadora sobre la que opera MONICOD. Lógicamente dicha sección se sitúa siempre después de la sección donde se lleva a cabo la impresión, y tan próxima a esta como sea posible por razones que serán expuestas en 4.

En la figura 3.15 se aprecian los dos tipos de cinta transportadora con las que trata MONICOD:

- Cinta elevadora de lata aprisionada
  - Son en realidad dos bandas que aprisionan las latas por ambos lados a modo de pinza. Esta sujeción posibilita vencer la gravedad y desplazarlas 'en volandas' sobre el eje vertical. Las latas están vacías de manera que la carga es tolerable.
- Cinta transportadora horizontal de lata encarrilada
  - La lata descansa sobre la banda y es transportada por esta. La lata sólo está unida a la banda por la gravedad sin ninguna otra sujeción salvo barreras a ambos lados que impiden que la lata abandone los límites de la banda. Este sistema aprovecha que las latas están llenas y por tanto son relativamente pesadas.

[8] En el siglo XIX ya se usaban las cintas transportadoras siendo su principal campo de aplicación la minería. Las primeras cintas transportadoras modernas, hechas de acero, vieron la luz en 1901, pero no fue hasta 1913 cuando Henry Ford introdujo la cinta transportadora en las cadenas de montaje.



(a) (b)

Figura 3.15: Cintas transportadoras: (a) Cinta elevadora de lata aprisionada(la flecha roja indica el sentido de la marcha habitual). (b)Cinta transportadora de lata encarrilada.

3.8. Validación: Código esperado versus código real

3.8.1. Validando la legibilidad

Nos referimos aquí a la legibilidad de la impresión. La definición de diccionario de "legibilidad" es "posibilidad de ser leído especialmente por tener la suficiente claridad". En MONICOD legibilidad tiene un significado diferente aunque, en general, un carácter legible para MONICOD también es un carácter que puede ser leído con la suficiente claridad.

En MONICOD la legibilidad se determina a nivel de carácter. Viene dada por el grado de similitud de la forma del carácter analizado con la forma esperada. Así, es un proceso de validación y no de reconocimiento: Existen formas esperadas para cada posición. Debe subrayarse que el sistema NO impone que estas formas esperadas gocen de la cualidad de ser "leídas especialmente por tener la suficiente claridad", aunque lo normal es que así sea.

3.8.2. Generación del código

El código de caducidad puede ser visto fácilmente como una composición de caracteres que guardan un determinado orden de tal manera que cada carácter tiene significado individual y su interpretación es determinada por su posición dentro del código.

3.8.2.1. Clasificación de caracteres

Prescindible e imprescindible No todos los caracteres son igual de importantes. El código de caducidad está orientado al usuario final de la lata principalmente. Él es su principal lector. Pero también existen caracteres de uso interno, esto es, que sólo tienen significado para la compañía llenadora de latas[9]. El número de lectores potenciales interesados en los caracteres internos es mucho menor y su ámbito de utilización es mucho más restringido. En consecuencia aunque un carácter particular haya sido etiquetado como no legible por el sistema no quiere decir necesariamente que el código devenga en inadmisible.

Extendiendo esta idea aceptamos que en los códigos de caducidad pueden existir caracteres prescindibles cuya condición de no legibilidad no afecta la condición de admisibilidad del código como un todo. Uno o más caracteres prescindibles ilegibles son irregularidades soportables donde no es necesario tomar ninguna medida, como lanzar una alarma o parar la línea.

La importancia de cada carácter es un atributo de la posición que ocupa. Por tanto sería más apropiado hablar de la importancia de cada posición. La importancia (prescindible/imprescindible) es un parámetro que viene dado por el exterior.

En 3.8.3 se examina como afecta a MONICOD este concepto.

Estáticos y dinámicos Un carácter puede ser estático o dinámico.

Un carácter estático es aquel que NO debería cambiar durante toda la jornada. Un mensaje navideño promocional o un carácter separador fijo tal como el carácter ':' que separa horas y minutos.

Un carácter dinámico es aquel que cambia o podría cambiar durante la jornada. Estos cambios suelen tener lugar a intervalos regulares. Horas y minutos son un buen ejemplo de caracteres dinámicos.

Existen caracteres dinámicos como los que reflejan el año en curso: '09', '98', etc. que no es de esperar que cambien durante la gran mayoría de las jornadas, aunque en teoría podrían hacerlo.

Manuales y automáticos Un carácter manual es introducido directamente por el usuario. Por ejemplo el mencionado texto navideño o un carácter fijo 'L' para designar Lote. Los caracteres manuales son casi siempre estáticos.

[9] Tales como la Compañia Cervecera de Canarias y la Compañia Cervecera de Canarias.

	00	01	02	03	04	05	06	07	08	09	10
00	J	U	L	0	5		1	0	:	4	2
01	L	4	1	9	0	/	0	1			

	00	01	02	03	04	05	06	07
00	0	2	-	0	7	-	0	5
01	1	0	:	1	7		A	

Figura 3.16: Código de Embotelladora de Canarias (Arriba) y de Cervecera de Canarias(Abajo)

Un carácter automático puede ser generado artificialmente siguiendo alguna regla. Los caracteres automáticos tienden a ser, por su propia naturaleza, dinámicos.

Información sensible La información mas sensible está casi siempre contenida en los caracteres dinámicos y automáticos. . Un ejemplo de los códigos en la figura 3.16. La estructura de los códigos de la compañía Embotelladora de Canarias y la compañía Cervecera de Canarias puede verse en las figura 3.17 y 3.19. El significado de cada elemento queda condensado respectivamente en las figura 3.18 y la figura 3.20.

3.8.2.2. Componentes del código

Fecha La fecha de caducidad. Está calculada a partir de la fecha de impresión. Es la principal razón de ser del código de caducidad. Al tratarse, con diferencia, de la información más valiosa, su legibilidad tiene que ser garantizada.

Hora La hora de la impresión. Se trata de información muy poco útil para el usuario final. Su interés es de naturaleza interna.

Código interno Un código que tiene significado interno. No tiene interés alguno para el usuario final. Su legibilidad es deseable pero no crítica

Embotelladora de Canarias

Secuencia conocida de tres Carácteres			dos dígitos			dos dígitos		:	dos dígitos	
'L'	cuatro dígitos				'/'	'0'	'1'			

J	U	L	0	5		1	0	:	4	2
L	4	1	9	0	/	0	1			

Figura 3.17: Estructura del código en Embotelladora de Canarias

Embotelladora de Canarias

	00	01	02	03	04	05	06	07	08	09	10
00	Primer caracter del mes	Segundo caracter del mes	Tercer caracter del mes	Penultimo dígito del año	Último dígito del año	' ' Espacio	Primer dígito de la hora	Segundo dígito de la hora	':' Dos puntos	Primer digito del minutero	Segundo digito del minutero
01	'L' de lote	Último dígito del año	Primer dígito del juliano	Segundo dígito del juliano	Tercer dígito del juliano	'\' barra divisora	Primer dígito de turno	Segundo dígito de turno			

Figura 3.18: Significado del código en Embotelladora de Canarias

Compañia Cervecera de Canarias

dos dígitos	'-'	dos dígitos	'-'	dos dígitos	
dos dígitos	':'	dos dígitos		'A'/'B'	

0	2	-	0	7	-	0	5
1	0	:	1	7		A	

Figura 3.19: Estructura del código en la Compañía Cervecera de Canarias

Código de Cervecera de Canarias

	00	01	02	03	04	05	06	07
00	Primer dígito del día	Segundo dígito del día	'-' guión separador	Primer dígito del mes	Segundo dígito del mes	'-' guión separador	Primer dígito del año	Segundo dígito del año
01	Primer dígito de la hora	Segundo dígito de la hora	':' Dos puntos	Primer dígito del minutero	Segundo dígito del minutero		Letra identificadora del turno	

Figura 3.20: Significado del código en la Compañía Cervecera de Canarias

Mensaje de texto Habitualmente un mensaje con carácter promocional. Si bien dirigido al usuario final, su legibilidad no es demasiado importante.

3.8.3. Importancia de caracteres

Exigir que todos los caracteres del código sean imprescindibles para validar la impresión y admitir la lata es posible, pero existen otras consideraciones.

1. Dado el costo de detener la línea para solventar un problema con la impresión o de localizar una lata para su retirada[10], lanzar una alarma o descartar una lata son operaciones que tienen aparejado un costo.

2. No para todos los caracteres la legibilidad es una propiedad imprescindible. Existen caracteres cuya no legibilidad es tolerable. Por ejemplo, la de aquellos caracteres destinados a uso puntual interno dentro de la planta de envasado.

Es posible reducir las detenciones de planta si aceptamos que haya caracteres ilegibles en el código siempre que para estos la legibilidad no sea imprescindible. Como quedó descrito en 3.8.2.1 habrán dos categorías:

- Caracteres Imprescindibles
- Caracteres Prescindibles

La decisión de qué carácter es importante o no procede del usuario de la herramienta. También el usuario decidirá cómo de estricta será la "política de validación". Ver 3.21.

[10] y quizás una posterior readmisión en la línea (si procede)

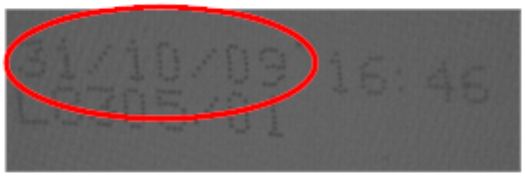


Figura 3.21: El código y la parte que indefectiblemente debe ser legible. Se extrae de este caso que cualquier problema de legibilidad en la segunda línea puede desestimarse si se opta por una política de validación relajada, y no considerarse causa suficiente de descarte. (☼)

3.8.4. Tipología de fallos

La tipología de fallos ayuda a determinar que es "legible" y qué no lo es.

En el esquema 3.22 se recogen los diferentes tipos de defectos que MONICOD debe encarar. Estos tipos de defectos pueden combinarse en una sola impresión. Tampoco es el propósito de MONICOD determinar que tipos de defecto están presentes en la impresión. En lugar de eso MONICOD trata de averiguar si los caracteres imprescindibles están presentes (no están ausentes), son legibles y admisibles. De ser así la lata es validada positivamente. En caso contrario será validada negativamente.

3.8.4.1. Defectos de ausencia

Los defectos de ausencia son una tipología tan particular y extendida que en sí mismos forman su propia subcategoría. Son impresiones incompletas que hacen imposible para el usuario final rescatar la información que se pretendía que figurase. Precisamente dentro de los defectos de ausencia la ausencia completa es, al parecer y con mucho, el error más común de todos los posibles provocados por el sistema de impresión. Ver 3.23 y 3.24. Cualquier defecto de ausencia que afecte a caracteres imprescindibles deberá venir seguido de una alarma de detención y/o descarte.

Todos los caracteres Todos los caracteres están ausentes

Uno o algunos caracteres Al menos uno o más caracteres están ausentes pero no todos. Los caracteres presentes son legibles.

Partes de carácter El carácter está parcialmente presente. En este caso el carácter muy probablemente sea ilegible, por lo que también puede clasificarse como un tipo de defecto ilegible.

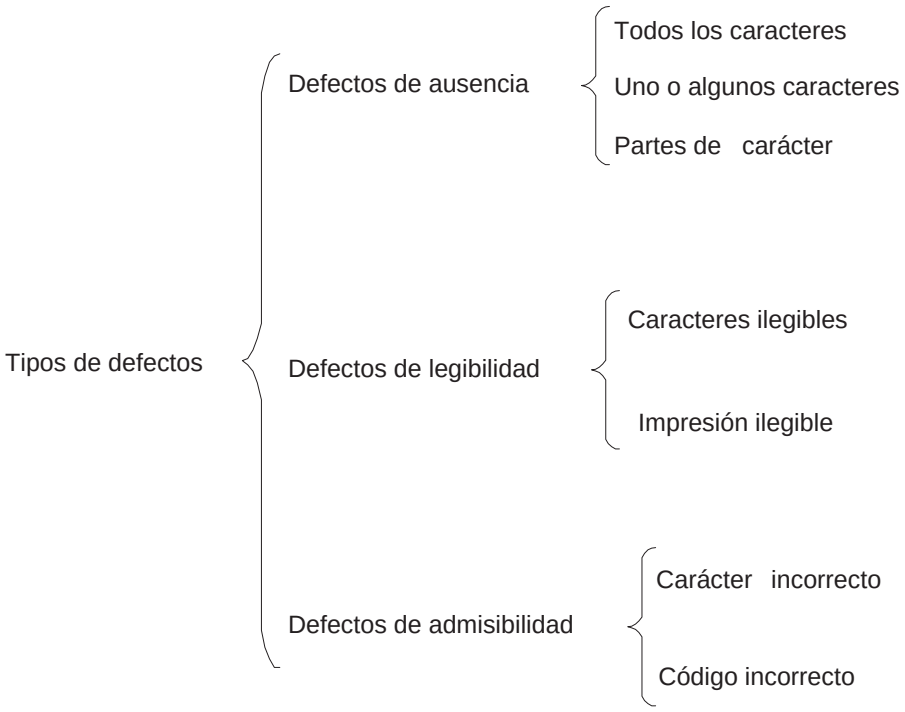


Figura 3.22: Clasificación de defectos de lata. Un carácter puede estar ausente, ser ilegible o ser inadmisible

3.8.4.2. Defectos de legibilidad

Los defectos de legibilidad son aquellos problemas que hacen imposible la identificación de los caracteres, aunque estos están presente (no ausentes) . Emborronamiento, formato incorrecto, caracteres invertidos o mal orientados, etcétera... Si la ilegibilidad afecta, al menos, a un carácter imprescindible, deberá provocar una alarma de detención y/o descarte.

Toda la impresión La ilegibilidad afecta a toda la impresión o a gran parte de ella de tal manera que aunque hay tinta presente es imposible identificar caracteres.

Caracteres ilegibles Es una variante del anterior más localizada o que respecta los espacios *intercaracter*. La ilegibilidad no afecta a la individualidad de los caracteres aunque estos no sean reconocible. Puede afectar a un carácter, a una fila entera, o a todos los caracteres.

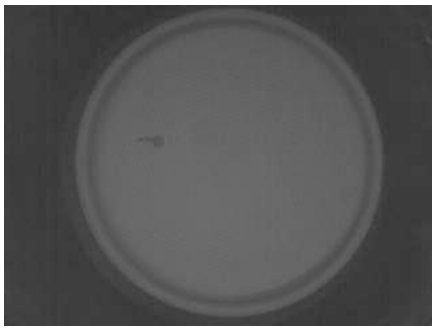


Figura 3.23: Ausencia de código. (☼)

Figura 3.24: Los caracteres prioritarios son legibles pero el código está incompleto. (☼)

3.8.4.3. Defectos de admisibilidad

Los defectos legibles son aquellos que no afectan a la correcta identificación de los caracteres pero el mensaje, como un todo, puede presentar dificultades de interpretación o simplemente ser erróneo. Este tipo de defectos menoscaba la confiabilidad del código. Si la ambigüedad, dobles interpretaciones, presencia de caracteres espurios, o sencillamente equivocados afecta a caracteres imprescindibles puede asumirse como código inaceptable y por tanto crear una alarma de detención/descarte. Si los defectos NO afectan a caracteres imprescindibles y no es posible interpretaciones incorrectas no es necesario lanzar un alarma de detención y/o descarte.

Carácter incorrecto Presencia de caracteres erróneos, ambiguos o ubicados en lugares equivocados. Por ejemplo, no se está imprimiendo la fecha correcta.

Mensaje incorrecto Todos los caracteres presentes son correctos pero aparecen caracteres inesperados que pueden hacer el mensaje ambiguo o reducir su confiabilidad.

3.8.5. Frecuencia de fallos

La frecuencia de fallos de validación es una consideración importante en MONICOD. Es diferente un error que ocurre puntualmente, o cada diez, cinco o incluso dos latas, que un error que se manifiesta con cada lata.

- Un error que se manifiesta con cada lata DEBE ser resuelto tan pronto sea detectado, deteniendo la línea si es necesario. Eso es así porque ninguna lata puede pasar el punto de impresión bajo esas condiciones. Esta es la situación más común.
- Un error que se manifiesta de forma periódica o regular (aunque no con toda lata) debe ser tenido en cuenta tarde o temprano (dependiendo de la frecuencia). Es una situación indeseable que responde a una anomalía peculiar. Es la situación más rara.
- Un error que se manifiesta puntualmente puede ser obviado. No es eficiente malgastar energía en averiguar la causa, que probablemente sea anecdótica y difícil de reproducir.

3.8.6. Posibilidad de diagnóstico

La posibilidad de diagnóstico obedece al hecho de que existe la posibilidad de establecer una relación causa-efecto para los problemas de legibilidad. Ver cuadro 3.2.

3.9. Soluciones existentes

3.9.1. Reconocedores de caracteres estándar

Desde hace ya tiempo los computadores domésticos tienen la suficiente potencia de cálculo para lanzar un reconocedor de carácter con una precisión excelente sobre fuentes de texto populares como por ejemplo Times New Roman con tamaño 12. Ver un ejemplo notable en la figura 3.25. Se centran en texto en imágenes (producto de un proceso de escaneado doméstico) o más recientemente archivos de extensión PDF. Entre sus virtudes más recientes está la adición de algoritmos de reconocimiento de formato (estilos de texto, doble columna, ilustraciones, ...) que puede ser trasladado a la salida.

Sus principales limitaciones tienen que ver con la dificultad de reconocer letra manuscrita (que sigue siendo un reto para cualquier OCR), y la exigencia de un buen contraste entre texto y fondo (por eso su empleo está especialmente indicado para fuentes de papel).

En lo que respecta a este trabajo no son sistemas dirigidos a obtener un rendimiento industrial (una página de texto puede llevar entre 4 a 8 segundos) o al procesamiento de *frames* en tiempo real o sobre superficies conflictivas como el metal. Por tanto no son comparables.

Síntoma	Causa
Código ausente	• El sistema de impresión está apagado. • La tinta de impresión se ha agotado. • La cabeza impresora no está bien situada sobre la cinta transportadora. • El cónducto de la tinta está desconectado o bloqueado.
Parte del código no aparece impreso	• La cabeza impresora no está bien situada sobre la cinta transportadora • La cabeza impresora está parcialmente obstruida.
...	...
El código es demasiado grueso o demasiado debil	• La composición de la solución del depósito de tinta no es idónea para la impresión.
El código es demasiado pequeño o demasiado grande	• La altura de la cabeza de impresora no es idónea.
Error de fecha u hora	• Los relojes de la impresora y de MONICOD no están correctamente sincronizados . • La configuración de MONICOD no es correcta para el actual producto. • La configuración del sistema impresor no es correcta para el actual producto.

Cuadro 3.2: Tabla de diagnósticos

Figura 3.25: Omnipage Professional de Nuance en `http://www.nuance.com/`. Probablemente el producto líder en el sector de los reconocedores de caracteres para uso doméstico o profesional. Aparentemente no le gustan mucho las latas.

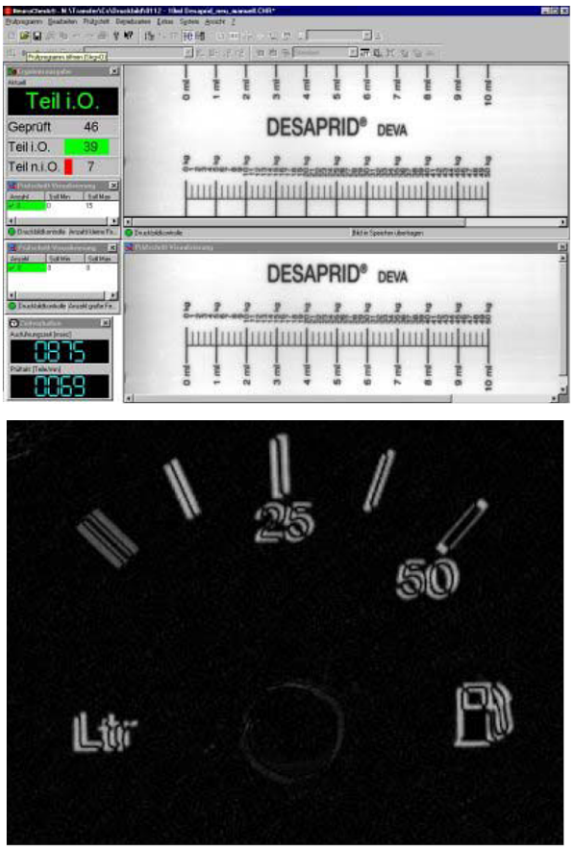


Figura 3.26: Ejemplos de uso de Neurocheck. Arriba dosificador de líquidos. Abajo una imagen diferencia

3.9.2. Neurocheck

Según reza su página web `http://www.neurocheck.com`, Neurocheck es un software de propósito general para control de calidad industrial. Puede chequear la ensamblajes, calidad de superficie y medidas de piezas, y validar inscripciones.

Sobre la validación de inscripciones y a modo de demostración desde su web puede descargarse dos ejemplos: La evaluación de inscripciones en un dosificador de líquidos y de un indicador de combustible de un automóvil. Ver figura 3.26. Parece que el principio que subyace en el mecanismo de funcionamiento es el de la diferencia de imágenes (una plantilla esperada vs una plantilla real). El tiempo de evaluación prometido para texto es de 0.4 a 0.6 segundos. Por lo que puede verse en los ejemplos resulta esencial que la imagen sea tan estándar como sea posible, lo que probablemente implique algún tipo de disparador o trigger, y el uso de áreas de interés muy especializadas.

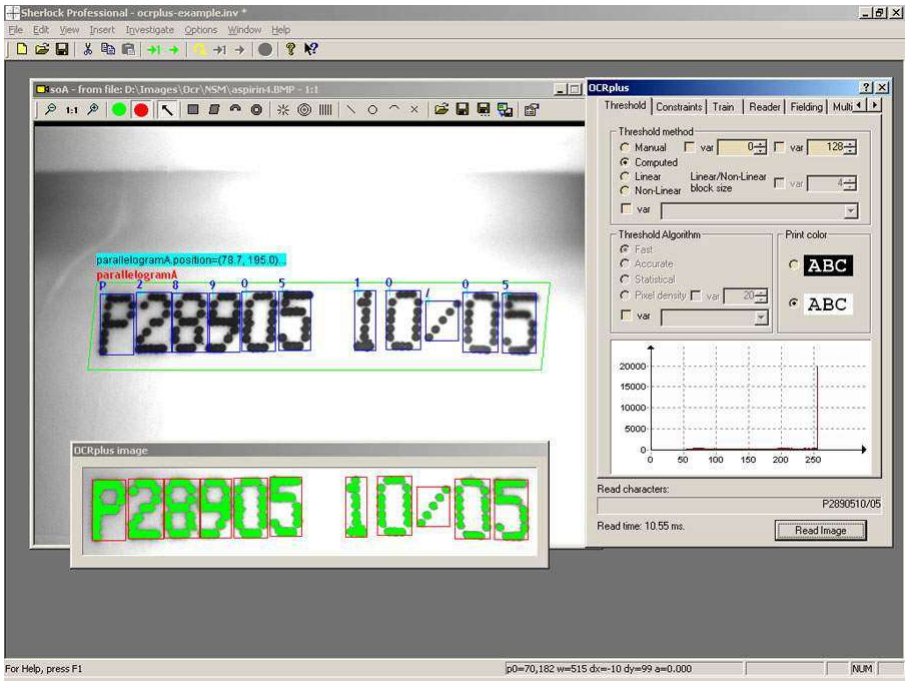


Figura 3.27: Un ejemplo de Sherlock. En esta captura el código mostrado ofrece un buen contraste y está perfectamente alineado. La iluminación no es homogénea, y sus consecuencias pueden apreciarse en la letra "P", más gruesa, respecto al resto de caracteres.

3.9.3. Sherlock

Sherlock es el software distribuido por TELEDYNE DALSA, que es el fabricante de la tarjeta de adquisición utilizada en el desarrollo de MONICOD.

Según puede leerse en su página web en `http://www.teledynedalsa.com/ipd/products/sherlock.aspx`, Sherlock es un inspector que puede ser aplicado a un amplio rango de aplicaciones. Entre ellas la lectura de caracteres. Ver figura 3.27. No ha trascendido información relacionada con su velocidad de operación. Al igual que MONICOD, parece estar basado en un comparador de plantillas que incluye capacidad de aprendizaje. El propósito de esta aproximación parece tener más que ver con la generalidad que con el alto rendimiento. Su precio anunciado son 2500$ mas 495$ por el módulo OCR.

3.9.4. Halcon

Aparentemente Halcon es el estado del arte actual en este campo.

En su web `http://www.mvtec.com/halcon/` se promete gran eficiencia y

soporte para plataformas multinucleo, SSE2 y AVX también como aceleración GPU. Funciona bajo Windows, Linux y MAX OSX, y garantiza independencia del hardware.

Como Sherlock y Neurocheck su funcionalidad es bastante genérica enfocada a la problemática industrial. De su folleto promocional (ver figura 3.28) en el aspecto concreto OCR, declara que entrena, clasifica, o verifica la fuente usando una gama de clasificadores que van desde el KNN al SVM. Se anuncia ratios de reconocimiento con márgenes de error de hasta 0.65 % con fuentes pregeneradas. Se afirma que ofrece una segmentación especializada, eliminación de rotación, clasificadores entrenables para fuentes personalizadas y selección de características. Para validación de caracteres se ofrece un sistema de comparación de plantillas basado en correlación, que es invariante respecto a iluminación, posición y cambios de escala.

Sin embargo no transciende información alguna sobre el *framerate* de adquisición o número de validaciones por segundo, ni se ofrece ejemplos concretos. La versión de prueba que ofrece tiene deshabilitada la adquisición desde cámara. El precio a fecha de este documento rondaba los 5.900€ mas actualizaciones anuales opcionales de 800€ (cada una).

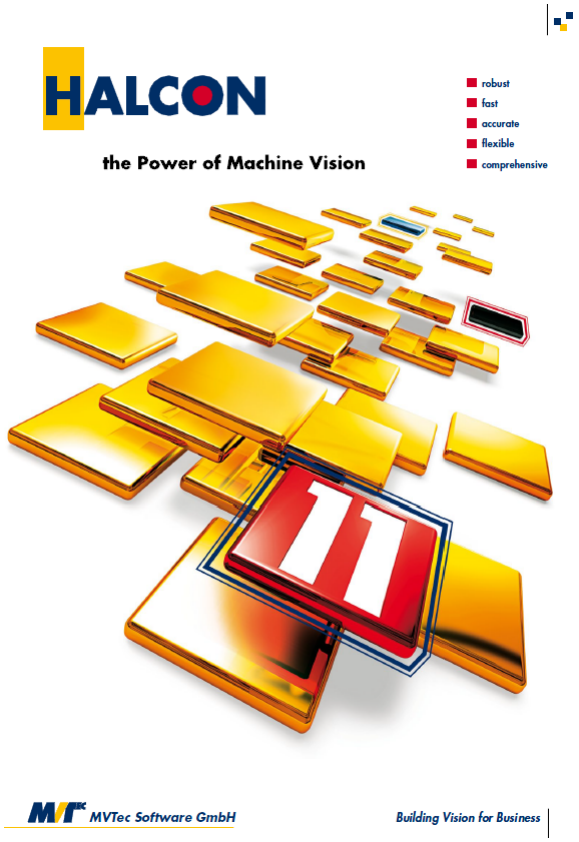


Figura 3.28: Portada del folleto promocional del hermético Halcon.

Capítulo 4

Obstáculos y contramedidas

Donde se explica con algún detalle los adversarios que enfrentamos en nuestro problema.

Este capítulo presenta los problemas a los que hace frente el proyecto así como soluciones y estrategias que los enfrentan. El orden de exposición corresponde más o menos a su importancia

- La velocidad de adquisición
- El rendimiento del procesador
- La impresión
- La superficie
- El entorno
- La línea

4.1. La velocidad de adquisición

El proceso de adquisición digital de imágenes es crítico en MONICOD. El escenario a tratar impone unas condiciones sobre como esa adquisición debe tener lugar. Una de las imposiciones tiene que ver con la velocidad de adquisición, esto es, el número de capturas independientes por unidad de tiempo.

Conceptos recurrentes, como *frame*, imagen, resolución, profundidad y definición de imagen así como la idea de un vídeo como una secuencia de *frames* interelacionados a través del paso del tiempo en anteriores y posteriores son tratados en los apartados A.2, A.3 y A.3.3.

4.1.1. Operar a bajas velocidades lleva a perder latas

Consideremos el problema en términos numéricos. La velocidad que se aspira a alcanzar con MONICOD corresponde a **120.000 latas por hora**.

$$\frac{120000 \text{ latas}}{3600 \text{ segundos en una hora}} \approx 33,33 \approx 35 \text{latas/s} \tag{4.1}$$

Una velocidad de 35 latas por segundo implica que en un segundo pueden pasar como máximo 35 latas diferentes por delante del campo de entrada. Con esto parece lógico que sea suficiente capturar 35 *frames* por segundo. 1 *frame* por lata. Sin embargo se ha observado que al trabajar a bajas velocidades raramente obtenemos *frames* con latas completas en las imágenes. El problema reside en que, al no tener sincronizada la captura con el instante en que la lata que transita ofrece "su mejor perfil a la cámara", no hay garantías de adquisiciones idóneas[1].

Esto lleva irremisiblemente a la pérdida de latas de validación: Se tienen *frames* donde aparece la lata pero no son 'validables'. Esto es inaceptable: el sistema estaría ciego ante cierto volumen de latas que pasarían inadvertidas.

4.1.1.1. Célula fotoeléctrica

La manera más común en la industria para enfrentar este problema es usar un disparador o 'trigger'. El funcionamiento de este esquema es sencillo: La célula fotoeléctrica detecta la entrada de una lata y envía una señal a la cámara para que haga una captura de tal manera que se obtiene una imagen de la lata en una ubicación óptima. De esta manera, fácilmente tenemos sólo los *frames* que interesan.

Las células fotoeléctricas son relativamente baratas. El propio sistema impresor las utiliza. Ver figura 3.12. MONICOD, sin embargo, renuncia a un disparador y prescindirá, en primera instancia, de esa posible solución. Las razones para esta decisión son:

- Generalidad/precisión

  No es siempre posible garantizar, para una instalación concreta, una ubicación específica, un tipo de cinta transportadora, una solución basada en célula fotoeléctrica. Por ejemplo, en un escenario donde las latas pueden estar en contacto entre sí la célula eléctrica tendría problemas porque depende de los huecos entre latas. Para ese ejemplo sería necesario reforzar el sistema con un separador[2] de latas a costa de ralentizar la velocidad de la línea.

  Por otra parte lo que sirve para el sistema impresor no tiene porque servir para el sistema de validación OCR (en este caso MONICOD). Al sistema

[1] Las latas no están distribuidas uniformemente en la cinta transportadora/elevadora. Ver sobre esto 4.7.1 y, especialmente, la figura 4.14 donde puede apreciarse una gran cantidad de capturas poco adecuadas para el reconocimiento/validación.

[2] Un dispositivo que separa las latas dentro de la cinta transportadora forzando una distancia mínima entre ellas.

impresor le basta imprimir el código de caducidad en cualquier parte de la base de lata con tal de que quede completo. Un sistema de validación OCR precisa que la lata quede en una posición muy concreta para que pueda aprovechar recursos como las áreas de interés. Ver 6.5. El simple mecanismo de funcionamiento de célula fotoeléctrica es insuficiente para alcanzar tales niveles de precisión dado el comportamiento complejo de la lata en la línea como queda reflejado en 4.7

- Abierto a incorporación posterior

  MONICOD es compatible y siempre está abierto a la incorporación de la célula fotoeléctrica. La célula fotoeléctrica es un componente hardware que mediatiza las imágenes de entrada al sistema pero no afecta a la plataforma física o al algoritmo.

4.1.1.2. Contramedida

Operaremos con cámaras de muy alta velocidad. *Framerates* más altos posibilitan mayor número de imágenes por unidad de tiempo. Y con esto se mejoran las posibilidades de adquirir varias imágenes del tránsito de una misma lata. Ver figura 4.14. Estas capturas adicionales permiten hacer una selección o incluso aplicar el método de validación en distintos *frames* de la misma lata para contrastar resultados.

La configuración a muy alta velocidad no está al alcance de las cámaras domesticas. Ni siquiera de las cámaras que suelen utilizarse en aplicaciones de visión artificial. En estas la calidad de la imagen prima, y los objetos que aparecen en escena no se mueven a velocidades muy altas. El ojo humano puede ser "engañado" con velocidades entre 25-30 *frames* por segundo para que crea que una sucesión de imágenes estáticas es una secuencia de movimiento. La técnica cinematográfica se fundamenta en ese principio[3].

En el caso que aborda este documento la adquisición de imágenes no tiene como usuario final a un ser humano. De hecho, el objeto de estudio con el que trata MONICOD -el código de caducidad- se mueve a tal velocidad que el ojo humano es incapaz siquiera de verlo. Ver figura **??**. Por ello se utilizará una cámara especializada.

Como la mayoría de los dispositivos de adquisición de imágenes tienen un destinatario humano directo la adquisición a muy altas velocidades, capacidad de hacer un gran número de capturas por unidad de tiempo, corresponde a un grado de especialización dentro de las cámaras, tal como lo supone la adquisición a muy altas resoluciones, o la adquisición en diferentes longitudes de onda. Ver secciones 5.2.2 y 5.2.6 para conocer especificaciones.

[3]Sin embargo el ojo -en determinados escenarios- es suficientemente sensible para percibir que "algo raro ocurre", aún a velocidades de 60 *frames* por segundo. Este "algo raro ocurre" se traduce en cansancio ocular, y sensación de parpadeo de la imagen. La frecuencia de barrido de los monitores es configurable precisamente para tratar con esta eventualidad. Mayor frecuencia de barrido, mejor resultado...

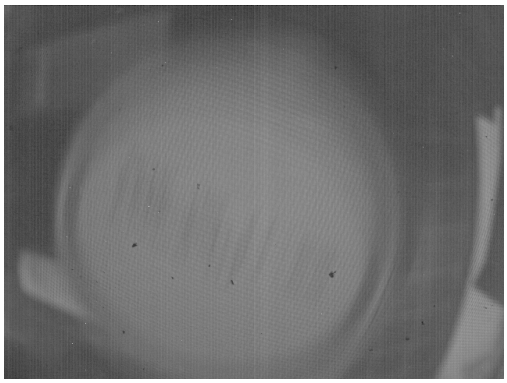


Figura 4.1: Tiempo de exposición demasiado largo. (☼)

4.1.2. Tiempos de exposición demasiado largos

En una cámara el sensor óptico integra en una sola imagen toda la luz recibida durante el tiempo de exposición. Esa integración implica una mejor calidad de imagen final operando bajo el principio de que los objetos retratados permanezcan inmóviles durante la exposición. Bajo esta condición trabajar el mayor tiempo de exposición posible equivale a mejor calidad en la adquisición. El tiempo de exposición es regulado por el obturador de la cámara (ver 5.2.2.1). Por otra parte el tiempo de exposición máximo está limitado por el número de *frames* por unidad de tiempo (*framerate*) que se desea capturar.

En nuestro escenario las latas, y los códigos de caducidad con ellas, se desplazan a través del campo de entrada. Y lo hacen a gran velocidad en relación a la longitud del campo de entrada y el tiempo de adquisición por *frame*. El sensor óptico, al tratar de integrar la escena en una sola imagen generará un *frame* borroso (lo que integra son imágenes diferentes) y las latas, y con ellos los códigos de caducidad, aparecen "movidos". Ver figura 4.1.

4.1.2.1. Contramedida

Es necesario reducir el tiempo de exposición o los códigos de caducidad saldrán borrosos en la imagen haciendo imposible cualquier procesamiento posterior. Las cámaras de alta velocidad ofrecen un gran margen en esta característica. Ver 4.2. Habrá que reducirlo de todas maneras con la contramedida 4.1.1.2: Al trabajar con mayor *framerate* (*frames* por segundo) las capturas ocurrirán más rápido y habrá menos margen para la exposición. Ver 4.3.

4.1.3. Tiempos de exposición demasiado corto

Tiempos de exposición menores reducen la cantidad de luz a la que es expuesto el sensor con lo que la imagen saldrá más oscura. Una imagen más oscura pierde contraste porque se compone de un rango de grises más estrecho. Es im-

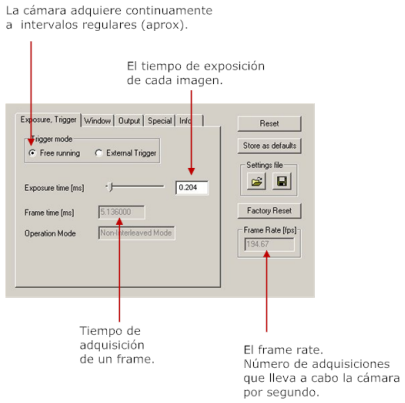


Figura 4.2: Configuración de una cámara de alta velocidad.

Figura 4.3: Aquí una secuencia de tres imágenes con una configuración de alta velocidad. Se agita la lata tan rápido como es posible y aun así resulta legible la fecha al dorso.

prescindible evitar que se pierda información de contraste debido a la oscuridad de la imagen. Técnicas como el ecualizado (ver 2.4.2.3) pueden hasta cierto punto contrarrestar este problema, pero tienen sus límites.

4.1.3.1. Contramedida

La dos únicas medidas posibles ante esto son:

1. Iluminar la escena intensamente.
2. Incrementar el tiempo de exposición.

En MONICOD se aplican las dos medidas aunque con limitaciones.

Es necesario alcanzar un compromiso entre la intensidad de la iluminación y la calidad de iluminación que el mercado nos ofrece. En el apartado 4.5 se expone lo que se quiere decir por calidad de iluminación.

El tiempo de exposición tiene un límite máximo dado por la velocidad de adquisición. Por ejemplo, 200 *frames* por segundo implica 200 capturas por segundo. Por tanto para cada captura dispondremos de un tiempo máximo de exposición de $\frac{1}{200}$ segundos. Ver sobre esto la subsección 4.1.4.

4.1.4. Operando a altas velocidades

Desafortunadamente trabajar con imágenes derivadas de una adquisición a alta velocidad nos deparará desagradables e inesperadas sorpresas.

Aunque el hardware de soporte a una adquisición con una alta velocidad, la calidad de la imagen se ve afectada:

1. La escena es diferente

   Cuando los objetos se desplazan a altas velocidades hay consecuencias visuales. En el caso de latas sobre una cinta transportada, estás se encuentran sometidas a una aceleración que genera temblor, corrientes de aire, desplazamiento de partículas de agua y polvo, brillos inusuales... Además reflejan la luz de forma algo diferente a cuando están inmóviles porque esta incide en ellas en fracciones de tiempo menores.

2. Perdida de contraste

   Una cámara que adquiere a altas velocidades debe abrir y cerrar el obturador (ver 5.2.2.1) a gran velocidad para llevar a cabo la captura. De manera que el tiempo de exposición (ver 5.2.2.1) de la captura es mucho menor. Al disminuir el tiempo de exposición entra menos luz que estimule el sensor (ver figura 5.6) de la cámara[4]. Como resultado la captura se oscurece lo que se traduce en una reducción de contraste significativa: Toda la información queda comprimida en un rango de grises menor.

[4] En oposición a las técnicas de prolongación del tiempo de exposición utilizadas en astronomía para obtener buenas capturas del firmamento. Estas se basan en acumular la escasa luz que llega de los objetos celestes muy lejanos.

3. Definición reducida

   Por otro lado, como la exposición para cada captura no es prolongada es difícil obtener una buena definición. ESTE PROBLEMA ES CRÍTICO. En otras palabras, la calidad de las imágenes de una adquisición a alta velocidad es siempre inferior que una adquisición a velocidades medias. Y el problema se agrava a mayor velocidad de adquisición. Esto es independiente de la velocidad de los objetos en la escena o incluso de que estos permanezca inmóviles.

4. La resolución está limitada

   Por limitaciones del harware a mayor *framerate* las imágenes que pueden obtenerse serán de menor resolución y viceversa: Mayores resoluciones sólo serán disponibles con menores *framerates*. Probablemente este problema esté relacionado con la perdida de tiempo de exposición o con el enorme volumen de datos generado por unidad de tiempo.

4.1.4.1. Contramedidas

1. No existen en MONICOD contramedidas explícitas para la diferencia de escena ni para el límite de resolución. Se espera que su sistema de adaptación pueda asumir esta dificultad.

2. La perdida de contraste puede ser contenida parcialmente lanzado un preproceso de ecualizado estándar.

3. La definición reducida es inevitable. Debe buscarse un compromiso entre *framerate*, tiempo de exposición y resolución.

   a) MONICOD opera a 640x480. Una resolución estándar que ofrece suficiente detalle para la validación de caracteres para el tamaño real de campo de entrada.

   b) A esta resolución la máxima velocidad ofrecida por el hardware escogido es 200 *frames* por segundo (200 fps)

   c) Para este *framerate* se selecciona el máximo tiempo de exposición posible y se determina que es suficiente.

   d) Otras consideraciones:

      1) Trabajar a menores resoluciones permite aumentar el *framerate* pero el máximo tiempo de exposición se volvería inadmisible.
      2) Trabajar a menores resoluciones sin aumentar el *framerate*. No sería posible aumentar el tiempo de exposición
      3) Trabajar a mayores resoluciones implica disminuir el *framerate*. Sería posible aumentar el tiempo de exposición. Peligro de perdida de latas. Volumen de datos a procesar mayor.
      4) Disminuir el *framerate* sin cambiar la resolución. Sería posible aumentar el tiempo de exposición. Peligro de perdida de latas.

5) Existe una resolución mínima que corresponde a la necesaria para identificar caracteres de una sola lata (o parcial tal como un área de ella), y una resolución máxima a partir de la cual el exceso de detalles de los caracteres podrían enturbiar su reconocimiento.

4.2. El rendimiento del procesador

La consideración más grave de todas las posibles es: ¿Es posible llevar a cabo un proceso de verificación en el tiempo mínimo exigido con la tecnología que se dispone? Esto dependerá de los algoritmos que constituyan MONICOD. Es necesario estudiar su **complejidad computacional** y **costo computacional**. En el capítulo 9 se profundiza sobre esto. Baste considerar ahora lo siguiente.

Hay que tener en cuenta varios factores:

- Operaciones involucradas

  En un computador genérico no todas las operaciones cuestan lo mismo. Algunas tienen mayor que costo que otras. El cálculo de una división es más costoso que el de una suma pero menor que el de una función trigonométrica.

- Datos involucrados en las operaciones

  También depende del tipo de datos que se ven afectados entre las operaciones. En general, una suma entera no cuesta lo mismo que una suma entre reales porque afecta a datos más complejos. Por otra parte entre los enteros hay diferencias dependiendo del tamaño que ocupan (enteros cortos, enteros largos,...) Un conocimiento de la arquitectura del procesador y de la memoria resulta determinante para la elección del tipo adecuado.

- Número de ejecuciones (bucles)

  Por sencilla y rápida que sea una operación puede tardar una infinidad si la ejecutamos infinitas veces.

- Orden de ejecución.

4.2.1. Contramedidas: Ejecución secuencial, concurrente y paralela

El primer paso para enfrentar dificultades en el rendimiento obtenido es siempre localizar e identificar el problema (por ejemplo con una herramienta de *profiling).* Una vez localizado la búsqueda de algoritmos alternativos más eficientes, que involucren operaciones más sencillas o que reduzcan el número de instrucciones, usar datos más simples, o realizar menos iteraciones suele ser

la respuesta. Es interesante minimizar el número de instrucciones promedio que se ejecuta. Por eso la aplicación de test que criben o incluso descarten cálculos es una medida atractiva.

Por otro lado también está el recurso de la concurrencia.

Según [108]

> *Dos procesos serán* **concurrentes** *cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última. Es decir, existe un solapamiento en la ejecución de sus instrucciones.*
>
> *Si se ejecutan al mismo tiempo los dos procesos, entonces estamos en una situación de ejecución* ***paralela****. La programación concurrente es un paralelismo potencial. Dependerá del hardware subyacente.*

MONICOD es un sistema multiproceso concurrente que necesita de un sistema que ofrezca ejecución paralela. La ejecución paralela, aunque incrementa la complejidad lógica, mejora considerablemente el aprovechamiento de cada unidad de tiempo. Las arquitecturas de hardware llevan favoreciendo esta aproximación desde hace más de una década.

Tómese como ejemplo evitar la perdida de *frames*. A medida que los frames van llegando al sistema es necesario transferirlos a memoria en el menor tiempo posible e indicar que se puede disponer de ellos. Un proceso debe ocuparse en exclusiva de esto manteniendo una frecuencia al menos tan alta como el *framerate* configurado (en nuestro caso 200fps). Para mantener una frecuencia será necesario descargar a este proceso de cualquier actividad que no sea crítica. Con este fin se habilitan procesos que operan en paralelo (no de manera concurrente) a este y que son los que efectivamente disponen de los *frames*. Ver una descripción en detalle de esto en la sección 6.7.

La tercera vía es simplemente no usar el procesador siempre que sea posible. Con la aparición de extensiones accesibles en el hardware programable tales como las tarjetas de adquisición y, ahora, las tarjetas de vídeo se ofrece una circuiteria especializada muy aprovechable para determinados cómputos (por ejemplo operaciones matriciales).

4.3. La impresión

Para una comprensión más completa de este apartado remito a la sección referida al sistema impresor 3.4 y a la tinta 3.5.

La impresión industrial sobre superficies de lata **NO produce el mismo resultado de una lata a otra**.

Supuesta una impresión continuada sin intervención externa (recarga, recalibrado, reajuste, limpieza del sistema de impresión) la impresión varia progresivamente, de manera que dos latas cualesquiera impresas una inmediatamente después de la otra se parecerán mucho entre si, pero diferirán de forma apreciable de una lata impresa hace un tiempo, o de la lata que se imprimirá al cabo de un tiempo. Todas esas variantes pueden ser legibles sin embargo. De hecho

puede ocurrir que el código alcance su mejor legibilidad en etapas intermedias hasta el punto de resultar más legible que en las primeras etapas. Con todo en las últimas etapas el código siempre se degrada y, por eso denominaremos a este fenómeno *degradación progresiva*. La *degradación progresiva* puede ser fácilmente explicada como la consecuencia de la degradación de los elementos que hacen posible la impresión. Por ejemplo a medida que la reserva de tinta queda exhausta la mezca que se inyecta pierde sus propiedades.

Hay varios factores para ello:

- Estado del cabezal impresor

  El cabezal es clave para la impresión. En un sistema de inyección de tinta, tal como nos ocupa con MONICOD, obstrucciones que puedan haber en el camino de la tinta debido a suciedad, puede motivar que la impresión sea defectuosa, incompleta o inexistente de una lata a otra.

- Posición del cabezal impresor sobre la cinta transportadora

  Un ligero desplazamiento del cabezal impresor puede motivar que la impresión sea defectuosa, incompleta o inexistente de una lata a otra. No sólo desplazamientos laterales sino la distancia del cabezal a la lata pueden causar cambios en la escala de la impresión o en la integridad de los caracteres.

- Estado del conducto entre el cabezal impresor y el depósito del sistema impresor.

- Estado del depósito del sistema impresor.

  La situación del depósito del sistema impresor afecta a la densidad de la tinta con la consiguiente aparición de grumos e impurezas que pueden obstruir al cabezal impresor o al conducto entre cabezal y depósito. Además, afecta a la calidad de la impresión.

- Velocidad de la lata.

  La impresión se lleva a cabo en movimiento. Las latas pueden tener velocidades diferentes en el momento de la impresión y sufrir aceleraciones y desaceleraciones durante la impresión. Ver más sobre esto en 4.7.

- Disposición de las latas entre sí

  La disposición de las latas entre sí en su camino al lugar de la impresión puede afectar la impresión. Así las latas pueden estar tan juntas que la célula fotoeléctrica sólo es capaz de ver una única lata. Además una cadencia de disparo irregular puede afectar al estado del cabezal impresor, el conducto entre cabezal y depósito y el propio depósito. La velocidad con la que el nivel del depósito desciende, la probabilidad con la que el cabeza puede quedar obstruido con tinta seca,... Ver más sobre esto en la sección 4.7.1.

- Vibraciones y otros desplazamientos en el momento de la impresión

  La lata está sujeta a otra clase de movimientos que pueden afectar al momento de la impresión. Ver a este respecto la sección 4.6.2.

- Estado de la superficie de lata

  Las superficies de lata no son iguales en todas las latas. Además, presentan sus propias dificultades combinadas con la humedad ambiente y la propia condensación de agua sobre la superficie. Se tratará específicamente en 4.4.

- Humedad ambiente

- La post-impresión

  El secado rápido de la tinta no es inmediato.

4.3.1. Contramedidas

No hay medidas especiales para enfrentar la impresión. El sistema de impresión, con todos sus inconvenientes, está aceptado por la industria.

4.4. La superficie

Una lata es un objeto cilíndrico. Una de sus bases soporta el mecanismo de apertura exhibido en 3.3. La impresión tiene lugar en la base opuesta[5] a la que denominaremos 'base de lata'. Ver figura 3.8

La superficie de la base de lata es ligeramente cóncava y está fabricada con aluminio u hojalata. Los posibles problemas que presenta la superficie de base de lata son de tres tipos.

- Problemas de iluminación

  Se profundizará sobre ellos en el apartado 4.5.

- Problemas de superficie irregular

  La superficie está pulida pero no es plana, sino cóncava. La concavidad, que es aproximadamente igual de lata a lata, no parece afectar a las condiciones de lectura del código de caducidad de forma reseñable. Es plausible que bajo ciertas circunstancias tenga alguna incidencia en la impresión o en la aparición de determinados brillos pero no hemos podido colegir efectos nocivos a los que atribuirles responsabilidad. En este documento la despreciaremos como factor independiente.

- Presencia de elementos extraños sobre la superficie. Por ejemplo gotas de agua. No parecen producir efectos negativos reseñables.

[5] Informalmente denominada el "culo de la lata" por la industria.

Esta zona para la impresión es conveniente porque la concavidad no es excesiva, es posible imprimir porque el color de la superficie ofrece un contraste suficiente con la tinta negra, existe tinta que ofrece una buena fijación a la superficie, es accesible para la lectura, el texto impreso goza de cierta protección que reduce el desgaste (precisamente por la concavidad) y no colisiona con la decoración de la lata.

4.4.1. Contramedidas

No hay medidas especiales para enfrentar la superficie. El principal problema (con diferencia) que ofrecen las superficies de lata tiene que ver con sus particulares propiedades lumínicas combinadas con que están en movimiento en el momento de la impresión.

4.5. La iluminación

La iluminación es, junto a la velocidad, el problema más importante que afronta MONICOD.

En condiciones de laboratorio, las primeras imágenes que se tomaron de las latas mostraron de manera contundente el primer adversario al que tendríamos que hacer frente: Los brillos. Ver 4.4.

El brillo es en realidad una propiedad subjetiva de la percepción visual. Un objeto "brilla" cuando parece estar emitiendo o reflejando luz. El brillo queda asociado así a la luminancia que es una medida de la intensidad luminosa por unidad de área de luz viajando en una dirección dada. La luminancia describe la cantidad de luz que pasa o es emitida desde un área particular y cae dentro un ángulo sólido dado.

Figura 4.4: Ejemplos de iluminación de laboratorio con la misma iluminación blanca.

Brillo se define como la cantidad de energía que el plano imagen recibe por unidad de área aparente. Por área aparente se entiende la parcela de superficie realmente observada [24].

Figura 4.5: Una iluminación NO homogénea genera problemas formidables en la selección de umbrales.

La conclusión de nuestra observación es que la superficie de lectura no es neutra lumínicamente. Refleja parte de la luz incidente en múltiples direcciones. En definitiva, genera brillos que ocultan y/o dificultan la visión de la superficie. En nuestro caso, los brillos son una consecuencia indeseable al tratar de iluminar superficies metálicas como las que nos ocupan (hojalata/aluminio) .

En algún momento necesitamos separar el texto (píxeles con tinta) del fondo (píxeles sin tinta): un umbralizado. Los factores descritos en 4.5.1 y la forma cóncava de la lata complican la tarea porque hacen que el contraste entre tinta y fondo sea variable de una parcela a otra de la imagen (ver 4.5). Eso limita las posibilidades de aplicar un umbralizado global. Por otra parte el umbralizado local es costoso computacionalmente como queda reflejado en 9.

4.5.1. Factores

El brillo de la imagen de un objeto tridimensional depende de los siguientes factores:

- Distribución de las fuentes de luz. Evidentemente, no es lo mismo observar una escena con unas condiciones de iluminación que con otras; bordes y sombras que no aparecían en una pueden aparecer en la otra, y viceversa.
- Propiedades de reflectancia del objeto, pudiendo éste estar compuesto de diversos materiales con distintas características reflectantes. Ver 4.6.
- Posición y orientación del objeto relativa a la cámara. Normalmente serán necesarios tres grados de libertad para especificar la orientación y tres para la posición.

Tenemos cierto control del primer factor ya que podemos decidir sobre la fuente de iluminación a utilizar como veremos más abajo en 4.5.4. Y escoger una fuente de iluminación de acuerdo con especificaciones severas.

Figura 4.6: Una lata de aluminio (izquierda) y una lata de hojalata (derecha) bajo luz de fluorescente común.

El segundo factor es aquí evidente. El infortunio ha dictado que los materiales primarios usados en la fabricación de latas, muy adecuados por su ligereza, ductilidad, e inocuidad, a saber, el aluminio y, alternativamente, la hojalata, presentan características refulgentes muy particulares. Especialmente en el área que NO se pinta, como es el caso del área de la base de la lata. El brillo creado aun por fuentes de luz débiles, satura a la cámara, sino en toda la superficie de la lata sí en ciertos lugares, lo que dificulta enormemente la lectura y validación del código.

La orientación de la cámara respecto a la lata está fija. Aún así el tercer factor es variable porque la posición del objeto relativa a la cámara cambia a medida que la lata se desplaza en la cinta transportadora. Por esto puede obtenerse imágenes más o menos afectadas por el brillo de un *frame* a otro (!).

4.5.2. Experimentos

Los mejores resultados obtenidos al natural (es decir, sin hacer uso de ningún tipo de artilugio salvo los comentados en el apartado anterior) tuvieron lugar cuando se aplicaba sombra sobre la lata de manera que ninguna luz incidía directamente sobre ella.

Desgraciadamente la aplicación de este método oscurecía sobremanera el área de lectura hasta el punto de hacerse ilegible por la ausencia de luz. Otro "modus operandi" se basaba en disminuir la cantidad de luz que entraba en la cámara rebajando la obturación de esta.

De nuevo resultaba muy difícil encontrar un compromiso entre la ausencia de brillos y el sacrificio de luz de trabajo.

Era tiempo de otras alternativas y optamos por el uso de técnicas fotográficas para la eliminación de brillos. Si bien en la industria existen sofisticadas soluciones para crear escenarios de iluminación complejos, nuestro propósito seguía siendo la construcción de una plataforma de pruebas económica y flexible sobre la que experimentar. El campo de la fotografía es fácilmente accesible, las soluciones que proponía no eran demasiado caras y sin duda, tendrían constancia y respuestas a problemas similares.

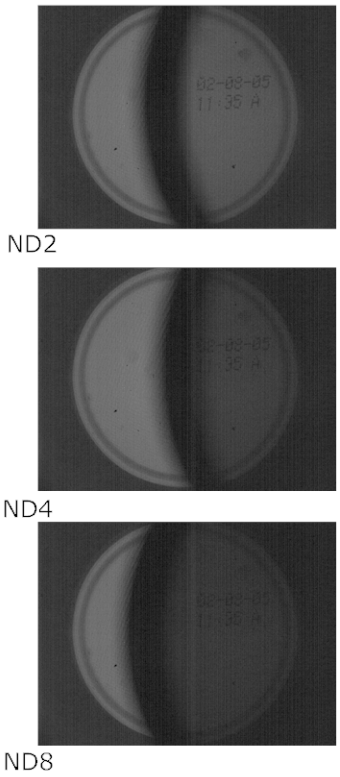


Figura 4.7: Filtros de densidad neutra (izquierda) y resultados obtenidos (derecha). De arriba a abajo filtros N2, N4 y N8. (☼)

Tras varias pruebas los filtros especializados en eliminación de reflejos (conocidos como filtros de densidad neutra[6]) daban los mejores resultados en el espectro probado. Ver figura 4.7. Desafortunadamente funcionan peor con brillos metálicos como es nuestro caso. Ver figura 4.8.

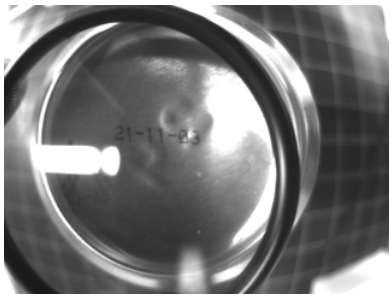


Figura 4.8: Ejemplo con idéntica iluminación, lata y filtro de densidad neutra

Por fin se experimentó con diferentes fuentes de iluminación. Ver 4.5.3.

4.5.3. Fuentes de iluminación

Distinguiremos dos tipos de iluminación:

- Iluminación global que afecta a todo el recinto donde se ubica MONICOD. Una parte de este recinto es el entorno inmediato sobre el que que trabaja MONICOD que denominaremos entorno de operación de MONICOD.
- Iluminación local que afecta muy específicamente al entorno de operación.

4.5.3.1. Iluminación global

Respecto a la iluminación ambiental se ha contrastado empíricamente durante el desarrollo de MONICOD que es imprevisible. No hay garantías sobre ella de ningún modo. Puede estar apagada, parcialmente apagada o encendida o parcialmente encendida, con mayor o menor intensidad. La iluminación suele correr a cargo de una distribución de lámparas fluorescente destinada a cubrir tareas genéricas de iluminación de trabajo. Las lámparas envejecen a velocidad variable. Parpadean. No encienden. O se enciende a voluntad sólo una parte de la distribución

Por otro lado es insuficiente, demasiado débil y/o está demasiado alejada para servir como fuente de iluminación del sistema. Ya se ha hablado del reducido tiempo de exposición con el que tiene que operar las cámaras de alta velocidad.

[6]Los filtros de densidad neutra simplemente reducen la intensidad de la luz, afectando por igual a todo el espectro. De este modo, lo que consiguen es reducir la cantidad de luz que penetra en el objetivo gracias a la interposición de una película neutra que actúa sobre todas las longitudes de onda por igual.

Figura 4.9: Sistema de iluminación ambiental del laboratorio

Si además la luz es escasa, las imágenes que se adquieran no ofrecerán ningún contraste.

Sin embargo tiene la suficiente intensidad para generar artefactos y brillos indeseados en la imagen. En definitiva, ruido lumínico.

De forma que la iluminación global es para MONICOD:

1. Inútil
2. Imprevisible
3. Ruidosa

En la figura 4.9 una imagen del sistema de iluminación presente en el laboratorio: Un fluorescente de luz blanca estándar.

4.5.3.2. Iluminación local

Respecto a la iluminación local, MONICOD debe proveer su propio sistema de iluminación para las latas porque, de otra forma, la cámara no podría adquirir imagen alguna en condiciones. Ver 4.10, 4.11, 4.12, 4.13

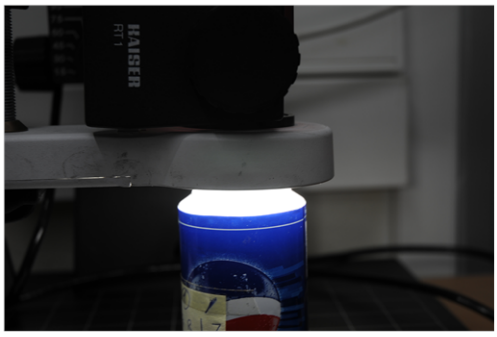
La distancia de la lata a la fuente de iluminación es mínima (en el rango de seguridad para el desplazamiento libre de las latas) para concentrar la luz en el área de observación y aprovechar las cualidades de la iluminación.

4.5.4. Contramedidas

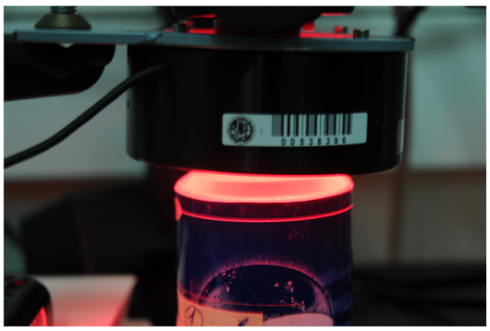
Las contramedidas adoptadas son:

- Se anulará la iluminación global de forma efectiva y completa mediante la construcción de una cámara oscura

  Con la cámara oscura conseguimos aislamiento de las condiciones adversas. También protegemos el entorno inmediato de la cámara, y parte del eje de visión. Es conveniente que pueda ser retirada de forma fácil y segura, y no produzca efectos adversos en el sistema (como un exceso de temperatura). Ver 5.2.5



(a)



(b)

Figura 4.10: (a) Iluminación blanca(fluorescente). (b) Iluminación roja(LED).

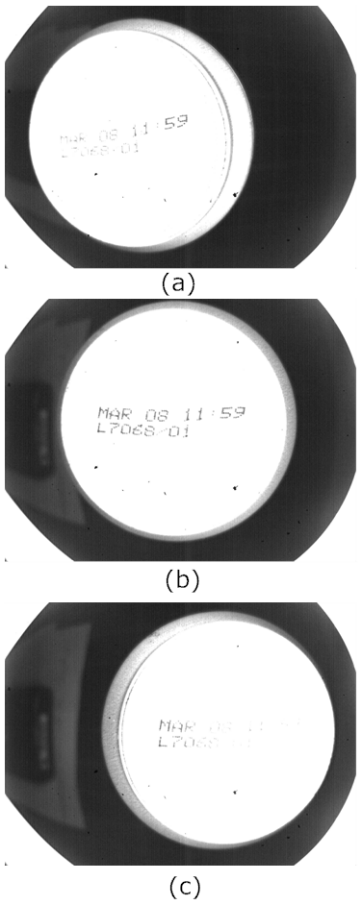


Figura 4.11: Iluminación blanca(fluorescente). (a)lata a la izquierda. (b)lata centrada. (c)lata a la derecha.

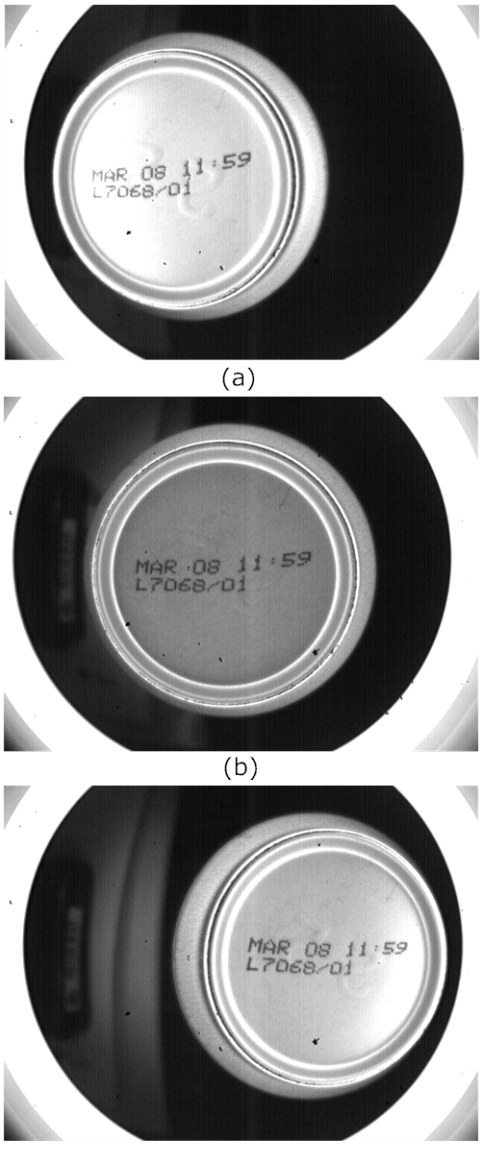


Figura 4.12: Iluminación roja (LED). (a) lata a la izquierda. (b) lata centrada. (c) lata a la derecha

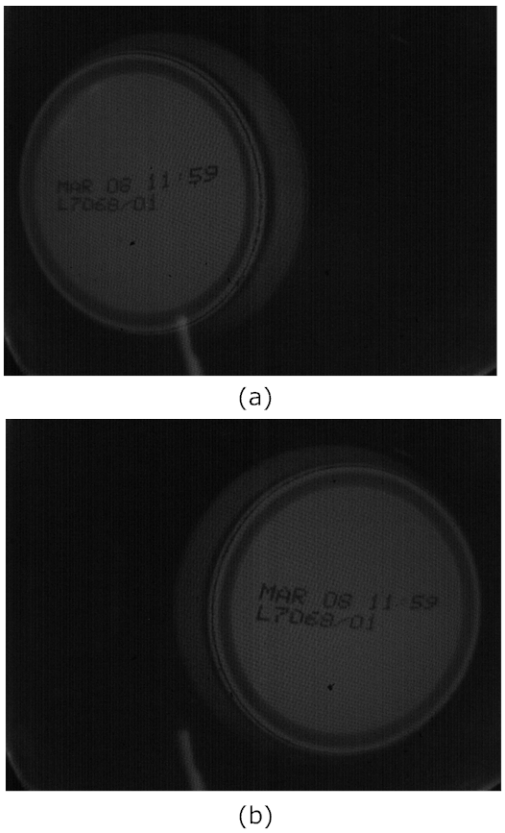


Figura 4.13: Iluminación polarizada. (a) lata a la izquierda. (b) lata a la derecha. En la parte inferior de la imagen puede apreciarse un ejemplo del efecto nocivo de la iluminación global. La cámara oscura resuelve la cuestión-

- Iluminación homogénea específica antibrillos

  El sistema debe contar con su propio de sistema de iluminación independiente y autónomo. Conseguimos así que el sistema sea inmune a un fallo generalizado de iluminación en toda la planta. Por otro lado un fallo en la iluminación local es catastrófico, y puede ser detectado por el operario de manera casi inmediata.

  Para determinar la luz a utilizar es preciso conocer que luces generan menos efectos indeseables al iluminar las superficies de lectura y que, en cierta medida, potencien una lectura correcta. Se experimentó con varias opciones. Ver 4.11,4.12 y 4.13. Como puede apreciarse en las figuras se comprobó que la luz polarizada ofrecía un mejor comportamiento lumínico con diferencia. Especialmente cuando la lata no está centrada. Ofrecía la ventaja adicional de una intensidad de luz configurable. El inconveniente es el precio[7].

4.6. El entorno

4.6.1. Presencia de agua y vapor de agua

La agresión más destacada proviene de la presencia abundante de agua en las plantas de llenado. Al finalizar la jornada, por ejemplo, se aplica, mediante manguera, chorros de agua a presión para asear el entorno de trabajo para la siguiente jornada.

En principio (sin alguna clase de coraza o "chubasquero") la permeabilidad de sistema es muy elevada. La fragilidad de los instrumentos ante el agua (alimentados por electricidad todos ellos) es considerable.

La humedad ambiental es muy alta. El vapor de agua (o su forma semicondensada) es el gas más extendido en el entorno. El problema se ve acentuado por la presencia de azúcares disueltos en la fabricación del refresco que viajan por el aire adheridos a las partículas de vapor de agua. Como resultado el vapor es más grasiento y opaco cuando se condensa.

El factor descrito afecta al proceso de adquisición formando irregularidades ópticas. Bien directamente, interponiéndose en forma de gas que altera la difusión de la luz entre lata y cámara, o bien adhiriéndose a los instrumentos formando una película no jabonosa sobre las lentes del sistema óptico o el sistema de iluminación.

4.6.1.1. Contramedidas

No hay un modo sencillo de evitarlo, salvo hacer una revisión a fondo de posibles emplazamientos y seleccionar una ubicación protegida. Por otra parte

[7] Sin embargo, a la vista de las ventajas, MONICOD lo asume.

otros factores restringen las posibles ubicaciones del sistema: Longitud de cable de datos Camera-Link, toma de corriente, la situación del sistema impresor, espacio suficiente, etcétera...

Afortunadamente, y por su propia naturaleza, el sistema impresor debe ser también ubicado en lugares protegidos de este tipo de incidencias. Como es una prioridad mantener cerca el sistema de validación, compartirá esa protección. A parte de esto la cámara oscura descrita en 4.5.4 también actuará como capuchón protector, retrasando la degradación citada sobre óptica, cámara y sistema de iluminación. Si bien parte del sistema de iluminación queda expuesto.

Será inevitable un mantenimiento periódico de las partes más expuestas del sistema.

4.6.2. Perturbaciones mecánicas del entorno general e inmediato

La cinta transportadora genera un movimiento mecánico que permite el desplazamiento de las latas sobre ella. Desafortunadamente hay efectos colaterales. Se ocasionan perturbaciones mecánicas (incluso temblores en el suelo) que afectan a cualquier elemento fijado a la cinta. Es extremadamente necesario que la plataforma física esté protegida de perturbaciones mecánicas. Ya no solo por los daños evidentes que pudieran sufrir algunos instrumentos especialmente sensibles sino por los errores en las verificaciones(falsos negativos y positivos).

4.6.2.1. Contramedidas

El sistema NO estará fijado bajo ninguna circunstancia a la cinta transportadora ya sea directa o indirectamente. Permanecerá como una ente independiente tanto como sea posible[8].

4.6.3. La intervención del operario

La intervención del operario es necesaria para el mantenimiento del sistema pero también en su papel de actor principal en la interacción con MONICOD.

1. Iniciar, reiniciar y detener el sistema MONICOD
2. Gestionar el calibrado del sistema MONICOD
3. Reiniciar alarmas de validación negativa.
4. Establecimiento de fechas, horas, codificación interna de las latas...
5. Configuración avanzada MONICOD

El sistema es absolutamente dependiente de la buena actuación del operario en el calibrado y al resolver incidencias.

[8] Aunque compartirán suelo probablemente :)

4.6.3.1. Contramedidas

El sistema tendrá una interfaz de complejidad ajustable de tres niveles.

Nivel básico Una única pantalla con acceso a los controles primarios de "Iniciar, reiniciar y detener el sistema MONICOD" y "Reiniciar alarmas de validación negativa". Esta interfaz de perfil bajo está destinada a uso por pantalla táctil. Se mantiene un perfil minimalista y simple.

Nivel avanzado Acceso a las funciones avanzadas de la aplicación. Todos los parámetros que un usuario convencional de MONICOD podría requerir se gestionan desde aquí. 'Gestionar el calibrado del sistema MONICOD', 'Establecimiento de fechas, horas, codificación interna de las latas' y 'Configuración avanzada MONICOD'

Nivel experto o de depuración El nivel experto ofrece capacidades de uso restringido, como la edición de los archivos de configuración.

4.7. La línea

El principal elemento de la línea es la cinta transportadora/elevadora. Ver figura 3.15 Hace circular las latas bajo el sistema de impresión y bajo la cámara del sistema de validación.

4.7.1. El tránsito de la lata

En la figura 4.14 se exhibe la adquisición a muy alta velocidad (200 *frames* por segundo) del tránsito de dos latas. Es interesante observar que a una velocidad de 15 latas por segundo muchos *frames* resultan inútiles. En la primera captura de la secuencia la lata está centrada en el campo de entrada, y puede ser validada. En la última captura de la secuencia aparece la siguiente lata centrada que necesita de validación. Todas las capturas no son aptas para la validación por una o varias de estas razones:

- No necesitan validación porque la lata centrada que hay en ellas ya ha sido validada.
- No se puede aplicar validación porque no hay lata o hay más de una lata.
- No se validará aún porque la lata no está en centrada en el campo de entrada.

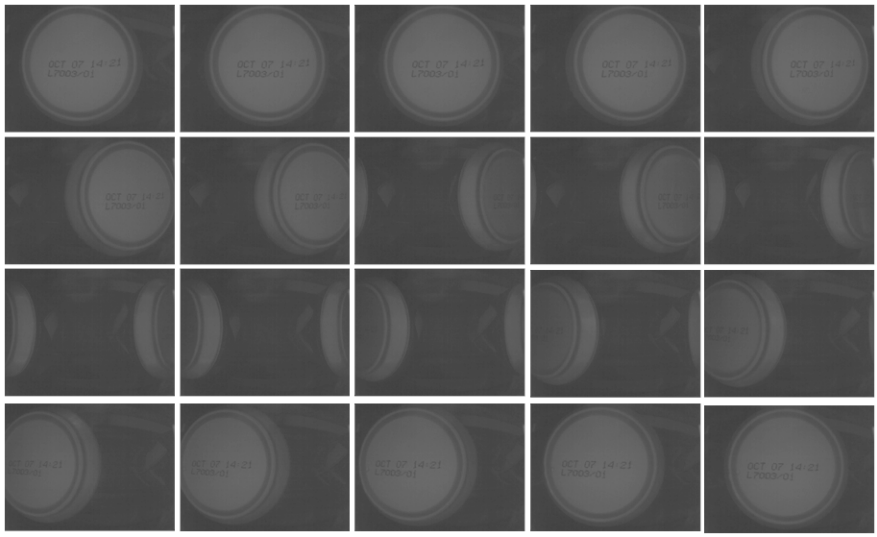


Figura 4.14: Tránsito de la lata como una sucesión de *frames* ininterrumpida. Esquina superior izquierda una lata centrada. Esquina inferior derecha la siguiente lata centrada. Los *frames* entre estos dos son superfluos y pueden ser ignorados desde el punto de la validación sin pérdida de latas.(☼)

4.7.2. Anomalías en el tránsito de la lata

El tránsito de la lata bajo el sistema de impresión y de validación MONICOD se caracteriza por su NO uniformidad.

Esta uniformidad se manifiesta en que :

1. Las latas NO están equidistantes entre sí.
2. La velocidad NO es constante.
3. Las latas NO están alineadas con el eje de desplazamiento (en el caso de la cinta transportadora).
4. Las latas no están alineadas sobre el plano de desplazamiento (en el caso de la cinta elevadora)
5. Las latas giran durante el tránsito.

4.7.2.1. Latas no equidistantes entre sí

La distancia entre latas no es fija de modo que las latas se presentan ante el sistema de impresión y sistema de validación MONICOD a intervalos irregulares.

Esto ocurre tanto en la cinta transportadora de lata encarrilada como en la cinta elevadora de lata aprisionada. Ver figura 4.15.

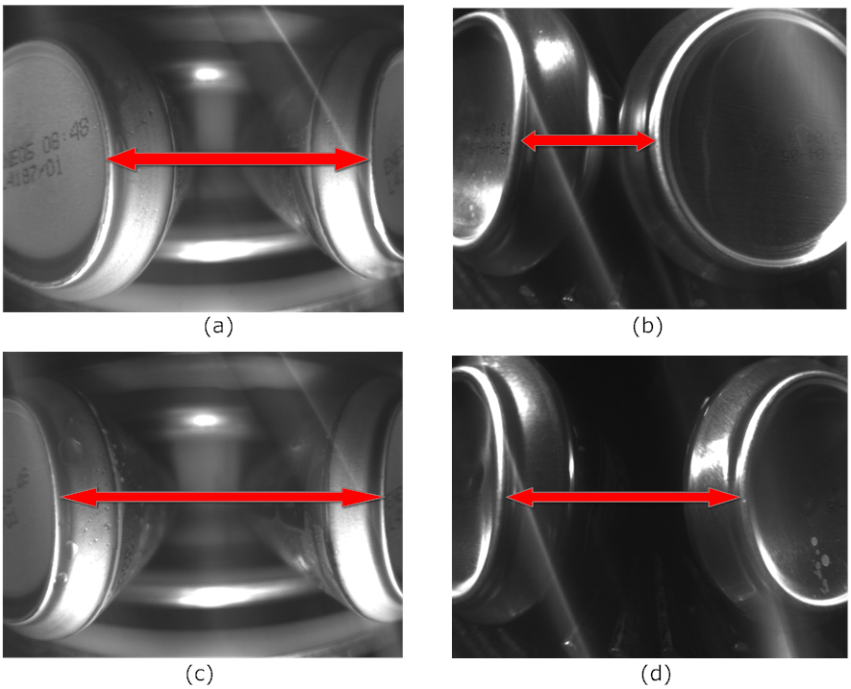


Figura 4.15: Latas no equidistantes. (a) y (c) cinta transportadora en la misma secuencia. (b) y (d) cinta elevadora en la misma secuencia.

El rango de diferencias de distancia van desde lo imperceptible a muchos metros. Es decir entre una lata pueden pasar décimas de segundo a varios minutos. En estos últimos casos no existe diferencia entre la línea parada si una lata en el campo de entrada de MONICOD o una línea en funcionamiento sobre la que no hay latas.

4.7.2.2. Velocidad no constante

Aunque existe una velocidad de operación, que es la habitual de la cinta, la velocidad de desplazamiento de las latas está lejos de ser constante. La cinta debe estar en cierta sincronía con la línea y esta pasa por diferentes estados (máquina de estado). La transición entre estos deriva en cambios de velocidad. Los cambios de estado son MUY frecuentes.

- Línea en reposo

  La cinta está inmóvil. Su velocidad es 0.

- Arranque de la línea

  La línea alcanza la velocidad de operación gradualmente. Aceleración positiva hasta alcanzar la velocidad de operación.

- Detención de la línea

  La línea alcanza la detención gradualmente. Aceleración negativa hasta alcanzar la velocidad 0.

- Cambios de velocidad

  La velocidad de operación fluctúa. La línea puede ver reducida su velocidad por muchos factores externos que tienen que ver con el rozamiento, incidencias en alguna etapas, etcétera.

- Reversa

  Velocidad negativa. La cinta puede avanzar en sentido contrario (marcha atrás) al natural. Este estado no forma parte del ciclo de funcionamiento normal de la línea y podemos ignorarlo de forma segura.

En el caso de la cinta elevadora de lata aprisionada las latas avanzan solidarias a la línea. En el caso de la cinta transportadora de lata encarrilada y dependiendo de la holgura las latas sufren micro-desplazamientos en respuesta a las variaciones de velocidad de la cinta de llenado (chocan o se empujan entre sí)[9].

4.7.2.3. Latas no alineadas con el eje de desplazamiento

Las latas pueden no estar perfectamente alineadas en su camino bajo el sistema de impresión y validación.

En general en la cinta transportadora de lata encarrilada no es deseable que la lata esté en contacto con los dos raíles laterales (que son estáticos y no se

[9] En caso extremos las latas pueden terminar siendo derribadas.

mueven con la cinta) al mismo tiempo porque el rozamiento causaría efectos muy negativos: Primero, la fricción puede dañar la integridad de las latas (deteriorando la decoración, deformándolas, ...). Segundo, la cinta transportadora tendría que vencer la resistencia de la fricción demandando mucha mayor potencia/energía. Tercero. Los raíles inevitablemente quedarían inservibles por el desgaste en muy poco tiempo. Por estas razones es necesario ofrecer holgura a las latas encarrilada que deriva en pequeños desplazamientos perpendiculares al eje de avance de las latas.

Esta inconveniencia NO se aplica a la cinta elevadora de lata aprisionada porque las dos bandas que aprisionan las latas impiden los desplazamientos laterales.

4.7.2.4. Latas no alineadas sobre el plano de desplazamiento

Las latas pueden no estar a la misma distancia de la cámara (distancia en el eje z variable). Las consecuencias de esto son:

- Se tiene que garantizar que las latas no están demasiado cerca suponiendo un riesgo para el sistema de iluminación o la propia cámara. Ver figura 4.16
- Se tiene que garantizar que las diferencias de distancias no entorpecen la validación. Por lo observado las diferencias de distancias son suficientemente pequeñas para no resultar apreciables en la validación.

Esta inconveniencia NO se aplica a la cinta transportadora de lata encarrilada porque las latas descansan directamente sobre el plano de desplazamiento, esto es, la cinta transportadora.

4.7.2.5. Giros de la lata

Además del desplazamiento de traslación lineal las latas pueden girar sobre si mismas una cierta cantidad de grados. Para los procesos de la cadena de envasado, incluida la propia impresión, esto resulta irrelevante. Para MONICOD, y por extensión para casi cualquier sistema OCR, estos giros suponen un gran problema. En 2.4.3 ya se presentó este severo problema.

En la figura 4.17 tratamos de exponer las razones para estos giros.

El giro es un problema cuando se produce en el espacio entre la impresión y la validación y resulta demasiado arbitrario para predecir de cuanta magnitud es para cada lata.

Los códigos con los que trabaja MONICOD tienen un sentido de lectura. Usando como sistema de referencia el eje de sentido de lectura y su perpendicular posicionamos los caracteres dentro del código. Los caracteres son ordenados de izquierda a derecha y de arriba a abajo. Esto es importante porque el significado de cada carácter viene dado por la posición que ocupa en los códigos de caducidad tratados por MONICOD. Algo tan sencillo como determinar cuantas filas de código hay presentes se vuelve bastante complicado con un código rotado una cantidad desconocida de grados.

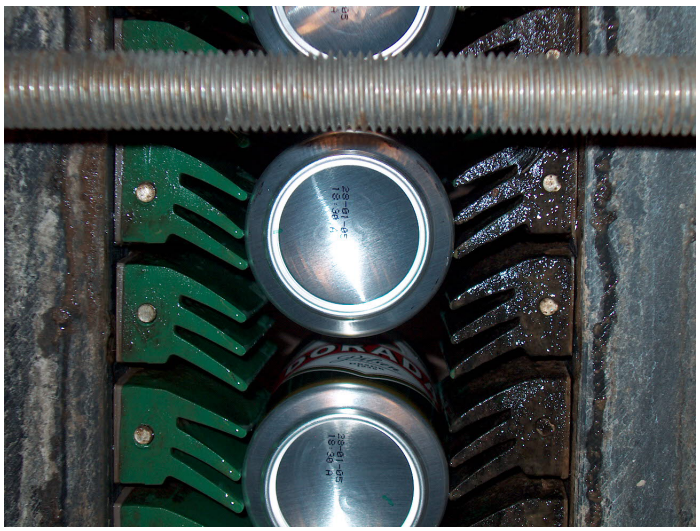


Figura 4.16: Latas en cinta elevadora vertical. Sentido de desplazamiento hacia arriba. La barra que aparece en la imagen evita que las latas sobresalgan demasiado en la cinta elevadora. Las latas están decoradas pero vacías en este punto.

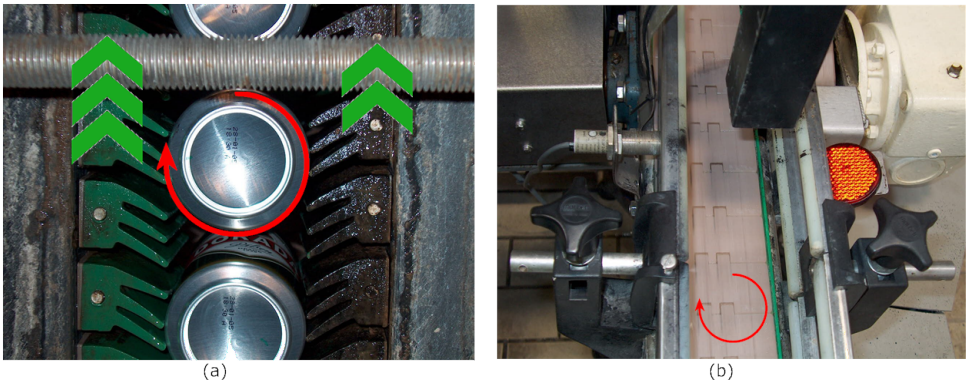


Figura 4.17: Rotación de latas en la cinta transportadora/elevadora (a) y (b) respectivamente. En (a) las bandas que aprisionan latas (que son más o menos independientes) pueden no moverse a la misma velocidad. Esa diferencia de velocidad hace que la lata rote. En (b) las latas giran aprovechando la holgura del carril; El contacto con otras latas o el trasvase de una cinta a otro hace que roten NOTA: El sentido horario de giro que se indica en la figura es sólo con propósitos ilustrativos: también puede ser anti-horario.

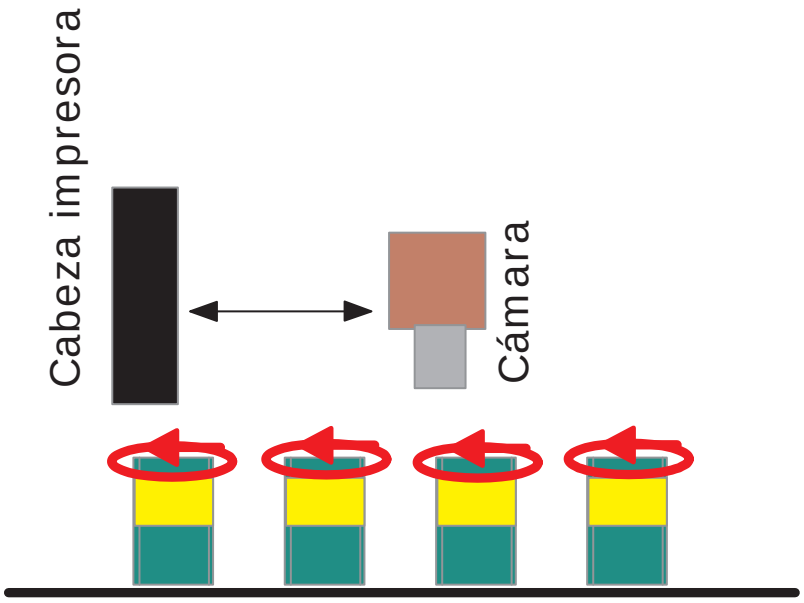


Figura 4.18: Distancia cabeza impresora-cámara

4.7.2.6. Contramedidas

MONICOD y el sistema de impresión comparten las cuatro incidencias descritas.

Para aliviar las consecuencias de las incidencias 1 y 2 el sistema de impresión depende de una célula fotoeléctrica. MONICOD prescinde de ese dispositivo integrando su propio mecanismo de detección de lata en el proceso visual.

La incidencia 3 no se da en el caso de la cinta elevadora, y en el caso de la cinta transportadora se encuentra limitada por los carriles. Afortunadamente es posible aproximar los raíles hasta alcanzar un compromiso entre fricción y holgura. Por ello el sistema impresor se limita a ignorarlo. MONICOD implementa el área de búsqueda del código en toda la base de la lata para absorber esta anomalía dentro de límites 'razonables' (que siempre son ciertos). Así, si el sistema impresor imprime con éxito la lata y MONICOD es capaz de detectarla, el código será localizado con independencia de donde exactamente está. Ver 6.5.

La incidencia 4 no se da en el caso de la cinta transportada, y en el caso de la cita elevadora es posible estimar una distancia de seguridad, así como crear salvaguardas en la plataforma de soporte que escuden la cámara de posibles latas que sobresalgan en exceso y amenacen con colisionar.

La incidencia 5 puede ser minimizada acercando tanto como sea posible la cámara al punto de impresión. Ver a este respecto la figura 4.18.

```
0.00639486 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--114.jpg
0.00635409 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--115.jpg
0.0064559 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--116.jpg
0.00635719 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--117.jpg
0.00653195 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--118.jpg
0.00641108 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--119.jpg
0.00634098 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--120.jpg
0.00635195 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--121.jpg
0.00644112 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--122.jpg
0.00679684 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--123.jpg
0.00643802 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--124.jpg
0.00634408 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--125.jpg
0.00634909 segundos
/home/usuario/Projects/Velocidad2/Salida/Frame--126.jpg
0.00641894 segundos
------------------------------------------------
INFORME 1
------------------------------------------------
Tiempo Transcurrido Medio       ->0.00642473
Tiempo Transcurrido Máximo       ->0.00697088
Tiempo Transcurrido Mínimo       ->0.00626087
Tiempo Trancurrido en Lectura   ->1.39
Tiempo Trancurrido en Escritura ->2.92

------------------------------------------------
Program exited successfully with errcode (0)
Press the Enter key to close this terminal ...
```

Figura 4.19: Captura que muestra una prueba de consumo la lectura y escritura de *frames* desde el disco frente al procesamiento efectivo de estos.

4.8. La imposibilidad del almacenamiento

El almacenamiento de los *frames* adquiridos para, por ejemplo, la creación de un vídeo no es una tarea inofensiva desde el punto de vista de la velocidad y el volumen de datos generado.

Existen dos inconvenientes críticos:

- La demora del almacenamiento respecto a la velocidad de adquisición en una filmación a alta velocidad.
- La enorme cantidad de datos implicada en el almacenamiento de una filmación a alta velocidad.

4.8.1. Velocidad

El primer inconveniente tiene que ver con la velocidad. Ver figura 4.19.

En las cámaras de adquisición a muy alta velocidad en que el proceso de filmación (entendiéndose por este término el almacenamiento de una secuencia de imágenes en algún tipo de medio reproducible) está automatizado existe un hardware muy especializado que da soporte para ello. Ver figura 4.20.

Pero en un PC (u otras máquinas de proceso genéricas) la filmación es una tarea delicada con efectos colaterales imprevistos. Como contrapartida la ven-

Figura 4.20: Una filmación a alta velocidad permite registrar cientos de *frames* en un soporte de vídeo para un análisis posterior. El registro no suele extenderse más de unos segundos.

taja del PC (u otras máquinas de proceso genéricas) sobre los dispositivos automatizados de filmación es que ofrecen la posibilidad de interponer en tiempo de adquisición alguna clase de avanzado tratamiento de imagen personalizado/programables en tiempo real, como es el caso del servicio que ofrece MONICOD.

Almacenar *frames* en un disco duro genérico (ver figura 4.21) implica una serie de tareas muy lentas[10] que se ejecutan secuencialmente sin que entre ellas haya posibilidad de concurrencia o simultaneidad. La demanda de acogida de nuevos *frames* para ser almacenados desbordará muy rápidamente el ritmo con que el hardware es capaz de almacenarlos. Clarifiquemos esta cuestión.

En una filmación la cámara está constantemente adquiriendo nuevos *frames*. Dichos *frames* pasan a la tarjeta de adquisición (si existe, si no se envía directamente) que a su vez los envía a la memoria RAM del PC a través del puerto de conexión (PCI, AGP, conexión firewire, etcétera según el modelo de tarjeta de adquisición) y del BUS de datos. La memoria RAM está destinada a un almacenamiento perecedero de datos (cuando el equipo se apaga, o se resetea, se borra por completo). Además, es un recurso extremadamente valioso en la ejecución de procesos del PC de manera que la ocupación de un porcentaje significativo durante mucho tiempo es gravosa para el funcionamiento y ejecución de procesos en todo el PC. Por otra parte se dispone de una capacidad de almacenamiento de memoria RAM muy limitada debido a su alto costo. Por todas estas razones

[10]Comparadas con el tiempo de un ciclo de CPU (Unidad de control de proceso).

Figura 4.21: Un disco duro

MEMORIA		DISCO DURO
Tiempo de transferencia	>	Tiempo de transferencia
Tiempo de escritura	<<<	Tiempo de escritura

Figura 4.22: Comparativa cualitativa de tiempos de transferencia y escritura cuando se realizan sobre memoria principal o sobre un disco duro.

el almacenamiento debe tener aparejado un volcado de las imágenes que estén en memoria al disco duro del PC, el cual carece de todos esos inconvenientes. Pero esta es una operación significativamente lenta que obligará al procesador a esperar. Ver figura 4.22.

La velocidad de transferencia a un disco duro genérico es, con mucho, bastante menor a la velocidad de transferencia desde la cámara hasta la memoria. Ciertamente el camino entre la memoria y el HD es mucho más corto y menos enrevesado que entre la cámara y la memoria. Pero mientras que en el segundo caso la velocidad es limitada sólo por un ancho de banda de transferencia cada vez más optimizado, en el primer caso tenemos como limitantes el ancho de banda y la velocidad propia de escritura en el HD. Tal diferencia de velocidades se traduce en un cuello de botella que obliga al sistema inevitablemente a ajustar su velocidad a la transferencia más lenta.

En tiempo real la cámara no puede dejar de adquirir imágenes al mismo ritmo y a transferir constantemente. De no hacerse se perderían irremediablemente *frames* y el ratio de *frames* procesados por unidad de tiempo decrecería significativamente. Así, los *frames* que no pueden ser tratados tienen que ser

descartados inmediatamente, y la filmación necesariamente estará incompleta.

4.8.1.1. Contramedidas

Soluciones:

1. La primera solución consiste sencillamente en mejorar el hardware físico. Existen discos duros de alta tasa de transferencia (como los ultra-wide SCSI) que pueden absorber suficientes *frames* para evitar tales cuellos de botellas. Desafortunadamente estos dispositivos son caros y no siendo la filmación una actividad principal en el proyecto queda poco justificado su uso. Además NO resuelven el problema expuesto. Sólo mejoran las estadísticas de perdida de *frame*. La tecnología de disco duro hasta la fecha requiere tiempos de escritura muy elevados respecto al resto de los dispositivos implicados. Respecto al tiempo de transferencia el proceso de filmación obliga a optimizar toda la cadena de transferencia desde la cámara al HD, y siempre estaremos limitados por el eslabón más lento. Como la transferencia es continua (sin intermedios), no es efectivo el uso de buffers (zonas de almacenamiento temporal que den mayor margen a las esperas) ni artimañas parecidas.

2. La segunda solución pasa por "soluciones" software del mercado optimizadas para este propósito. Haciendo uso de principios como la comprensión de imagen, la creación de un vídeo directamente y no de ficheros intermedios de imágenes independientes para cada *frame* (la mayor parte del tiempo consumido en la escritura del HD es para la apertura y cierre del fichero de modo que crear un vídeo sólo implica una única apertura y cierre) y otros refinamientos logra resultados muy aceptables. Sin embargo, de nuevo su elevado costo en forma de licencias, y sobre todo la alta demanda de computo adicional ofrece poca justificación para esto. La compresión de imágenes y el mantenimiento en memoria de un área de síntesis hace que consuma excesivos recursos impidiendo la convivencia con procesos exigentes tales como MONICOD

3. La tercera solución, que es la aplicada en nuestro caso y está anexada en los apéndices, consiste en el desarrollo de un algoritmo de almacenamiento a dos tiempos. Es decir, se restringe la filmación a unos segundos. Lo suficiente para que "quepa" en la memoria (primer tiempo) antes de que el sistema empieza a degradarse. Luego, concluida la adquisición y detenida la cámara, se lleva a cabo el almacenamiento efectivo en el disco duro (segundo tiempo). Este almacenamiento efectivo necesita tiempo del orden minutos para completarse. Como contrapartida esta solución limita severamente el tiempo de adquisición continua.

4.8.2. Volumen de datos

La segunda consideración procede del gran volumen de datos producidos por la cámara por unidad de tiempo. Para empezar a mayor *framerate* mayor

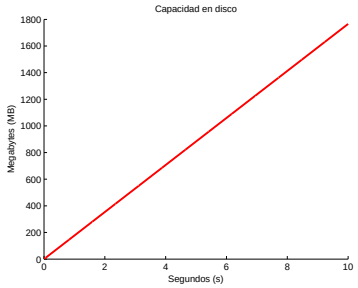


Figura 4.23: Crecimiento de la ocupación del disco en un intervalo de diez segundos. *Framerate*: 200fps. Tamaño *frame*: 925696 bytes. Tiempo: 10 segundos.

número de capturas. Cada captura debe ser registrada ocupando un espacio. Por tanto el espacio requerido para almacenamiento se multiplica por unidad de tiempo.

El espacio es un recurso limitado y se agotará. Será necesario migrar en intervalos regulares el volumen de datos a dispositivos de almacenamiento más baratos. De otra manera el dispositivo de almacenamiento colapsará.

Desgraciadamente el problema descrito se agravará inevitablemente a medida que aumente la velocidad de adquisición. Así, este problema que ya se presenta a 30 *frames* por segundo, a 60 *frames* o a 120 será mucho más acuciante. Ver 4.23.

4.8.2.1. Contramedidas

MONICOD no contempla el proceso de filmación. Al fin y al cabo estamos interesados en una validación sistemática en tiempo real de latas que viajan sobre una cinta transportadora y no en filmar dicho trasiego de latas. En otras palabras, en la MONItorización de CODigo, no es relevante conservar un *frame* ya sea válido o no. Sería lícito suprimirlo inmediatamente, si bien se contempla la posibilidad de almacenar un número limitado de imágenes con códigos inválidos a efectos de traza.

Ahora bien: En tiempo de desarrollo resulta muy conveniente disponer de secuencias de imágenes almacenadas y reproducibles una y otra vez con las que trabajar. De ahí, que sea una característica soportada.

4.9. Otras consideraciones

4.9.1. La ausencia de color

En MONICOD prescindiremos de la información de color. Operaremos en su lugar con intensidades lumínicas representadas con un valor de gris que va desde '0' (reflexión de la luz inexistente ya sea por ausencia de luz o por las propiedades

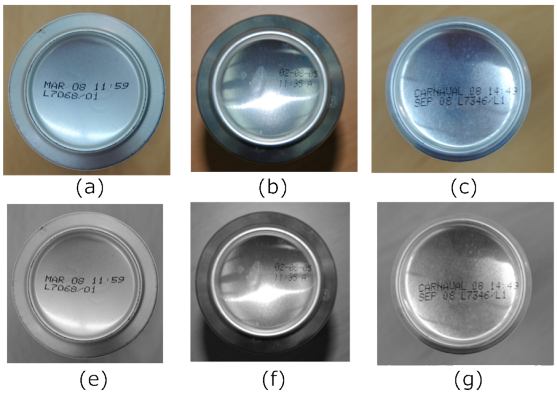


Figura 4.24: Latas en color vs latas en escala de grises

de absorción del material donde esta incide) o '255' o blanco (reflexión de la luz total)

Hay varias razones para ello:

1. Hasta la fecha el hardware de alta velocidad casi universalmente trabaja en escala de grises.

2. Trabajar con canales de color multiplicaría la cantidad de datos a tratar.

3. Un código legible en color también lo es en escala de grises y viceversa. Toda la información que se necesita teóricamente está ahí. Ver figura 4.24.

Capítulo 5

La plataforma física y herramientas software

"Sobre los elementos constitutivos de MONICOD"

Es fácil, e imprescindible, separar MONICOD en dos partes claramente diferenciadas pero con igual protagonismo. La Plataforma Física y el Motor de Validación.

Este capítulo se centrará en la plataforma física y en las herramientas software que han posibilitado la construcción del motor de validación.

5.1. Dos partes, un todo

La Plataforma Física es aquello que podemos tocar con las manos: Cámara, PC, cables de conexión, focos para iluminar el área, cámara oscura para aislar el entorno del campo de visión, monitor y teclado para interactuar el usuario, etcétera...

En un sistema de alta eficiencia donde el tiempo de cómputo debe minimizarse encontrar soluciones físicas, cuando es posible, es altamente deseable cuando no inevitable. De hecho en no pocas ocasiones proceder de otra manera sería imposible. Un ejemplo de esto es la solución adoptada para la iluminación.

Por otra parte el Motor de Validación, con sus mecanismos de reconocimiento, validación y decisión no tiene existencia física. Se trata de procesos de cómputo que vienen dados por uno o más algoritmos[1]. Si bien en temas posterio-

[1] Una analogía burda pero este esquela es la calculadora elemental y su botón raíz cuadrada ($\surd$) (o cualquier otro botón). La operación "raíz cuadrada" no está contenida en la mecánica que hace que funcione el botón de raíz cuadrada. En lugar de eso dicho botón lanza la cadena de instrucciones "programada" que la calcula - tal cadena de instrucciones es sencillamente el algoritmo "Raíz cuadrada". En la analogía se parte de la premisa que la raíz cuadrada no esté integrada en el circuito de la calculadora, es decir, que el algoritmo NO se haya cableado dándole una existencia física.

res se hará una descripción exhaustiva del Motor de Validación aprovecharemos aquí para presentar la herramientas software con las que ha sido construido.

5.2. Los elementos físicos

5.2.1. Plataforma física

La importancia de una buena plataforma física no será nunca sobreestimada.

En los anexos puede encontrarse el estudio y selección de la oferta que nos ofreció el mercado en su momento. Dicho estudio se ha llevado a cabo por secciones, resultado natural de la descomposición del sistema físico en sus partes, que son justamente los apartados que trataremos inmediatamente. Ver 5.1.

De acuerdo al servicio con que cada elemento físico nos provee tenemos:

- Dispositivos físicos que contribuyen en la adquisición de datos.

  1. **Cámara**. Ver sección 5.2.2.
  2. **Ópticas**. Ver sección 5.2.3.
  3. **Sistema de Iluminación**. Ver sección 5.2.4.
  4. **Cámara Oscura**. Ver sección 5.2.5.
  5. **Placa de Adquisición**. Ver sección 5.2.6.
  6. **Soporte.** Ver sección 5.2.7.

- Dispositivos físicos que dan soporte a la computación del motor de reconocimiento/validación.

  1. **Computador**. Ver sección 5.2.8.

- Dispositivos físicos que dan soporte a la entrada/salida de la interfaz de usuario del motor de reconocimiento/validación.

  1. **Tarjeta de vídeo/sonido**
  2. **Monitor**
  3. **Botonera**
  4. **Pantalla táctil**
  5. **Teclado**
  6. **Ratón**

- Dispositivos físicos que dan soporte a la salida principal del motor de validación. Ejemplos de esto pueden ser:

  1. **Sirena luminosa y sonora**
  2. **Ejector expulsor de latas que descarta físicamente la lata**

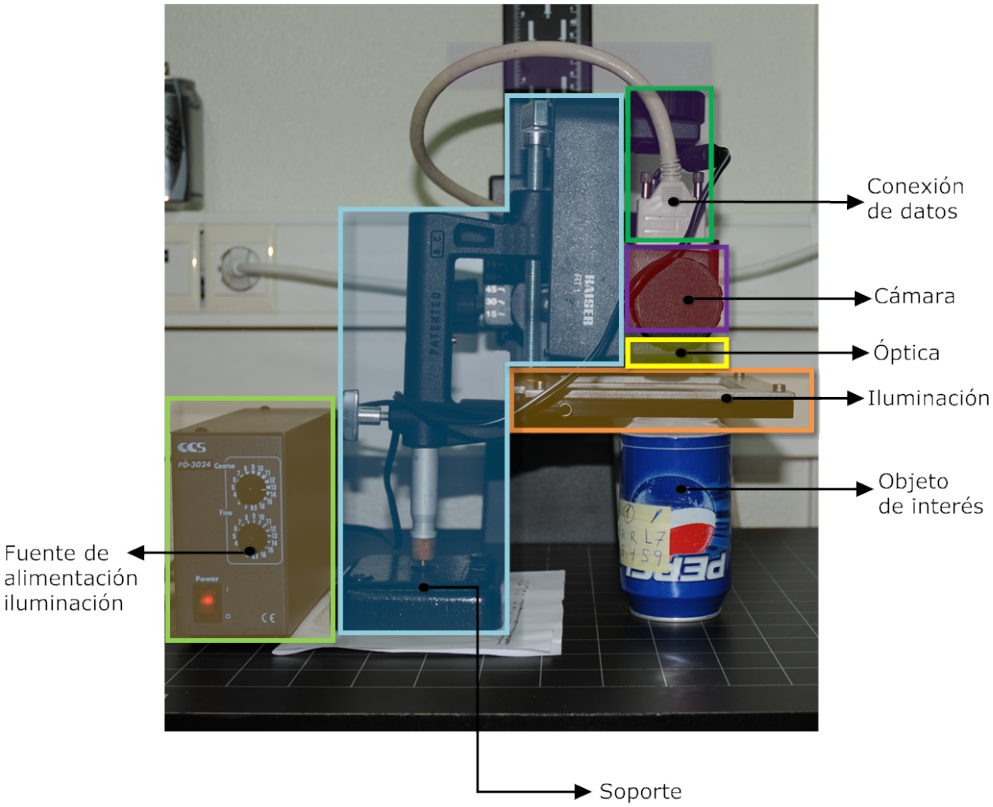


Figura 5.1: Esquema de MONICOD:Plataforma física

3. **Modem GSM**
4. **...**

Esta salida es alternativa a la que podría dar opcionalmente la interfaz MONICOD y persigue dar una visibilidad/utilidad más adecuada en un entorno industrial.

- Otros elementos.

  De especial importancia son aquellos elementos que ofrecen protección general contra un entorno hostil. Un ejemplo de esto es la cámara oscura: Una caja que protege al sistema y mejora las condiciones de la adquisición.

Los seis primeros componentes son **imprescindibles** para ejecutar el objetivo propuesto. Es decir, si bien algunos tienen una mayor relevancia que otros en la construcción del resultado final, la presencia e implementación correcta de cada uno es obligatoria en aras de obtener cualquier resultado.

Figura 5.2: Vista lateral de la cámara

5.2.2. Cámara

La cámara es el dispositivo transductor de la realidad visual a una representación interna sobre la que es posible operar. Representación que puede ser analógica o digital y que será enviada al sistema perceptivo interno (Placa de captura). Objetivamente es un componente decisivo en la tarea de la adquisición. En las figuras 5.2, 5.3, 5.4, 5.5 se muestra diferentes imágenes de la cámara. En la figura 5.6 puede apreciarse el sensor CCD[2] con el que está provisto la cámara. Un componente decisivo para sacar partido de la cámara es el software que tiene asociado. Ver figura 5.7.

La configuración de operación que emplearemos será:

- 200 fps (*frames* por segundo)
- Resolución 640×480
- B/N 8-bits 256 niveles de gris

5.2.2.1. El obturador y el tiempo de exposición

El obturador controla cuanto tiempo se deja pasar la luz a través de las lentes y, por tanto, el tiempo que el sensor es expuesto a la luz. Dicho tiempo es denominado tiempo de exposición. Como referencia en el momento de redacción de este documento el tiempo de exposición utilizado por MONICOD es 0,990 milisegundos.

[2]CCD(Charge-coupled device) es un dispositivo para el movimiento de la carga eléctrica. habitualmente desde el interior del dispositivo a un área donde la carga puede ser manipulada, por ejemplo, para hacer una conversión a valores digitales. Willard Boyle y George E. Smith, inventores del dispositivo CCD, recibieron el premio nobel por este hallazgo en 2009.

Figura 5.3: Vista trasera de la cámara

Figura 5.4: Vista "aérea" de las conexiones de la cámara.

Figura 5.5: Vista transversal de las conexiones de la cámara

Figura 5.6: Vista del sensor CMOS

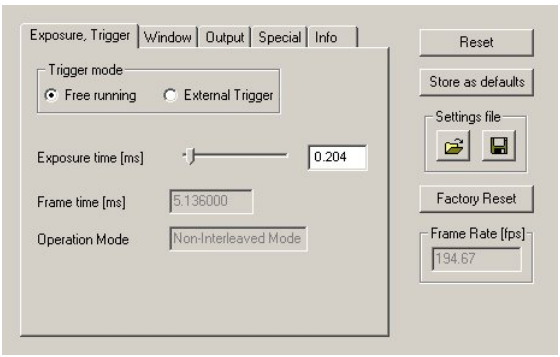


Figura 5.7: PFRemote: Software de configuración de la cámara photonfocus proporcionado por el fabricante.

Figura 5.8: Óptica manual utilizada por MONICOD.

5.2.3. Ópticas

La óptica es el mecanismo que regula la calidad de visionado de los aspectos de la realidad visual que nos interesan antes de que la cámara capture la escena. Esto se hace controlando la manera con la que la luz incide en el sensor de la cámara. Esto es el enfoque, y debe configurarse para el escenario concreto con el que estamos operando. Ver figura 5.8.

Exteriormente, la óptica u objetivo adopta la forma de un cilindro metálico cuya cara anterior es una lente; en su parte posterior (montura) posee un rosca o un sistema de bayoneta que permite su fijación a la cámara. En su interior se dispone de una agrupación de lentes de características diversas, junto con los dispositivos que posibilitan el desplazamiento de las mismas.

Las **ópticas** comerciales **están compuestas por agrupaciones de lentes**

positivas o convergentes y negativas o divergentes. Las lentes son dispositivos ópticos transparentes que, merced a fenómenos de refracción de la luz al pasar a través suya, desvían los haces de luz que reciben. Las lentes positivas hacen converger en un punto todos los rayos paralelos que las atraviesan. Las lentes negativas, por el contrario, se caracterizan porque los rayos de luz paralelos que las atraviesan se apartan de este paralelismo, divergen entre si. Aunque esta agrupación de lentes se comporta, respecto de los rayos luminosos, como lo haría una lente simple, su utilización se justifica en aras de una mejor calidad de la imagen, compensando, en la medida de lo posible, las distintas distorsiones y aberraciones ópticas que se producen[24].

Todas las ópticas, en mayor o menor medida, distorsionan la imagen. En la práctica, la distorsión es tanto mayor cuanto menor es la distancia focal, es decir, cuanto mayor es el ángulo de visión. Efectos de esta distorsión puede ser apreciados si se observa con detenimiento la figura 4.14 Existen ópticas que suprimen la distorsión pero la relación costo/beneficio no es buena si podemos prescindir de el procesamiento cerca de los límites de la imagen y centramos la atención. Eso es lo que hace MONICOD aprovechando que la alta velocidad de adquisición le permite seleccionar un *frame* apto.

Debe considerarse además que la distancia más adecuada entre cámara y lata depende de las necesidades ópticas pero también de las circunstancias medioambientales que podrían afectar al correcto visionado. Así, la lente de la cámara no puede estar excesivamente cerca de la superficie de la lata si se quiere evitar perturbaciones (agua, gases, corrientes de aire,...). Ni tampoco demasiado lejos ya que ello obligaría a resoluciones mayores que requieren un tratamiento computacional más costosos. Además, se ha constando que aumentar la distancia acarrea serios problemas lumínicas (la iluminación controlada pierde eficacia).

5.2.4. Sistema de iluminación

El sistema de iluminación mejora las condiciones de visionado del entorno general limitando el ruido visual, especialmente el que deriva de la luz. Este ruido es muy perjudicial en el proceso de imagen y las estrategias para amortiguarlo por software (si ello fuera posible) serían muy costosas en tiempo, factor que resulta crítico para nuestros propósitos. En las figuras 5.9, 5.10, 5.11 y 5.12 pueden apreciarse los sistemas de iluminación con los que se ha experimentado. Los resultados pueden verse en el apartado 4.5.3.2.

5.2.5. Cámara Oscura

La cámara oscura (ver 5.13) complementa al sistema de iluminación aislando la escena de casi toda iluminación externa a este. Otra función importante de la cámara oscura es proteger la integridad y el delicado equilibrio del sistema del exterior. La cámara oscura en MONICOD no puede ser hermética porque hay que posibilitar el tránsito de las latas.

Figura 5.9: Fuente de iluminación blanca basada en tecnología de fluorescente.

Figura 5.10: Fuente de iluminación blanca basada en tecnología de fluorescente encendida.

Figura 5.11: Fuente de iluminación roja basada en tecnología LED

Figura 5.12: Fuente de iluminación roja encendida basada en tecnología LED

5.2.6. Placa de adquisición

La placa de adquisición es el elemento de interconexión entre la cámara y el sistema computador, de manera que suministra las imágenes a los procesos de reconocimiento y verificación. Ver 5.14.

La principal utilidad que tiene para MONICOD la placa de adquisición es ofrecer puerto para conexión CameraLink. CameraLink es un protocolo de conexión serie capaz de soportar anchos de conexión considerables (del orden de varios Gigabits por segundo). CameraLink es necesario porque otros tipos de conexión digital como USB o FireWire no ofrecen la misma capacidad y estabilidad.

Por otra parte la placa de adquisición ofrece capacidades por hardware de tratamiento. MONICOD renuncia a ellas en aras de la generalidad (no dependencia de un hardware concreto).

5.2.7. Soporte

La plataforma de soporte es la "percha" en la que "cuelga" la cámara, ópticas, el sistema de iluminación y en ocasiones el sistema computador y sistema interfaz. Su propósito es adecuar la plataforma física a los requerimientos, condiciones y restricciones de medio en el que se usa. Además protege a los dispositivos sensibles de cualquier riesgo o inclemencia del medio. Si bien no lleva a cabo ninguna tarea perceptiva es un elemento básico para la percepción. Ver figura 5.15, y dos ejemplos en planta en la figura 5.16 y la figura 5.17.

5.2.8. Computador

El sistema computador se ocupa del procesamiento efectivo de las imágenes realizando la transducción a respuestas del tipo: *Código correctamente impre-*

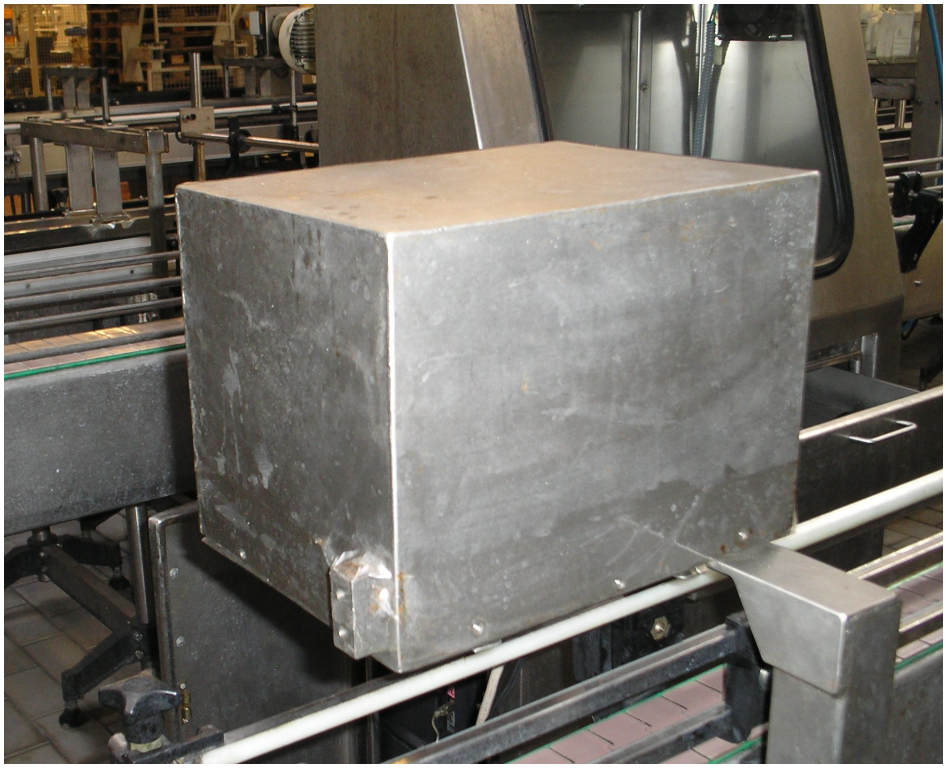


Figura 5.13: Cámara oscura en planta

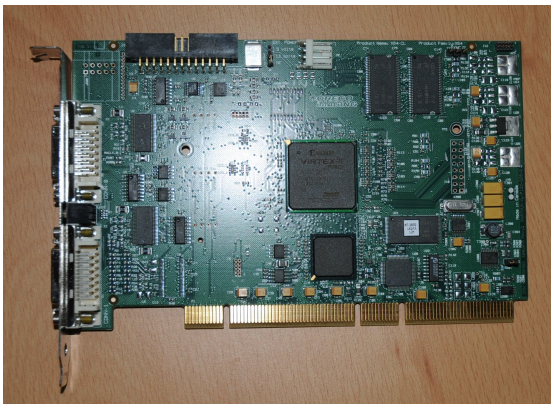


Figura 5.14: Tarjeta de adquisición. X64-iPro

Figura 5.15: Soporte del laboratorio

so/código incorrectamente impreso. No se descartan sin embargo otras posibles salidas. Ejecutar el algoritmo de validación/verificación es su principal tarea aunque otros estados y tareas son también posibles: Cargar nuevos valores de parámetros debido a una recalibración, reconfigurar la placa de adquisición y la cámara, aceptar entradas de usuario o bien sencillamente esperar nuevas adquisiciones retrasadas por una detención de la línea.

El computador utilizado es un Intel(R) Xeon(TM) CPU 380GHz 3.79GHz, 2GB RAM.

5.2.9. Interfaz física

El sistema de interfaz se encarga de todas las comunicaciones entre el usuario humano y el sistema computador. Tanto las entradas del usuario como las salidas del sistema encontrarán su expresión a través de la interfaz. El usuario regirá las configuraciones, calibraciones y demás alteraciones desde aquí. El sistema de proceso informará del resultado de sus resoluciones así como cualquier otro aviso pertinente desde aquí. El modo de interacción vendrá dado, pues, por este sistema. Un monitor, pantalla táctil, MODEM/móvil GSM, ratón, teclado, tarjeta de vídeo/sonido, botonera,... cualquiera de esos dispositivos, o una combinación de estos pudiera ser adecuado para el propósito de interfaz.

No todos han sido utilizados en MONICOD. Por ejemplo no han sido empleados la tarjeta de sonido, la botonera o la pantalla táctil.

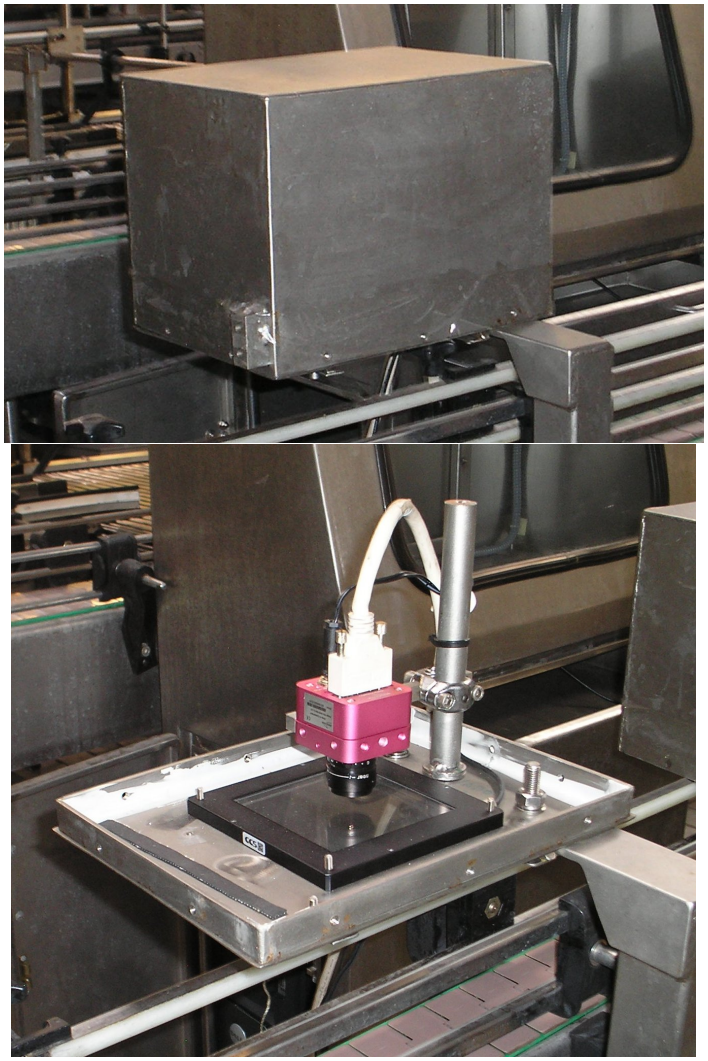


Figura 5.16: Plataforma física en planta sobre cinta transportadora horizontal. Arriba con la cámara oscura. Abajo sin la cámara oscura.

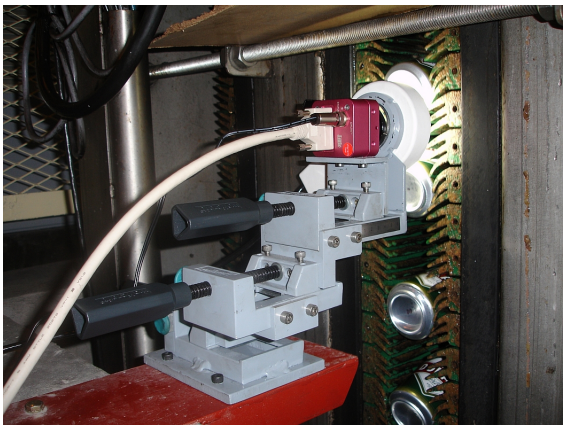


Figura 5.17: Otra plataforma física en planta, esta vez sobre cinta transportadora vertical.

5.2.10. Mecanismo efector

El mecanismo efector transduce la salida/decisión del sistema de proceso a una acción particular. Dependerá de que acción particular se decida finalmente. Un modelo de alarmas sonoras o visuales, un "puf" o dispositivo integrado que descarte físicamente la lata, etcétera.

5.3. Los elementos software

5.3.1. Filosofía

Las metas perseguidas en el desarrollo de la solución así como en el desarrollo de los productos asociados y herramientas propias, definen por si mismas, la filosofía elegida:

- Portabilidad
- Modularidad
- Transparencia
- Eficiencia
- Amigabilidad
- Simplicidad
- Gratuidad

Estas propiedades deberían ser heredadas en mayor o menor grado por el producto final dependiendo de sus especificaciones concretas.

He aquí una prolija descripción de sus características y la justificación para cada una de ellas:

5.3.1.1. Portabilidad

El sistema debe ser portable para permitir evaluar diseños en plataformas muy diferentes. Portabilidad significa independencia de las condiciones del entorno en el que se ejecuta. El entorno está comprendido principalmente por dos aspectos: Sistema Operativo y Máquina/Hardware. La portabilidad nos garantiza poder movernos de unos entornos a otros con relativa facilidad, y con cambios mínimos en el diseño nativo de aplicación.

Así es esperable que la plataforma pueda ser ejecutada en diferentes sistemas operativos[3]. Del mismo modo, también sería posible migrar con un esfuerzo mínimo entre diferentes máquinas u arquitecturas de procesador[4].

Es forzoso reconocer que la idea de portabilidad se puede alcanzar sólo hasta ciertos límites. No todas las conversiones son posibles ni todas las migraciones viables. El esfuerzo de compatibilizar desarrollos requiere un importante esfuerzo de terceros, y dicho esfuerzo viene dado por la demanda. Si no existe tal demanda, simplemente lo puentes necesarios para la portabilidad no se construyen o se construyen insuficientemente. Tampoco podemos olvidar las diferencias de planteamiento entre las organizaciones/empresas propietarias, autoras de los diferentes entornos posibles. En ocasiones esa brecha es demasiado ancha para poder ser ignorada, incluso a este nivel, haciéndose casi imposible la traducción desde un entorno a otro. Y, por último, pero no menos importante, quedan los intereses particulares para asegurar la impermeabilidad de los detalles de entorno, y garantizar diseños exclusivos.

En cualquier caso, el mero propósito de buscar portabilidad cómo objetivo básico ya en los estadios más tempranos del desarrollo software (tal como se ha hecho) proporciona unas bases incontestables destinadas a facilitar las cosas en fases posteriores de cara a este aspecto.

La importancia de este requerimiento viene dada por la necesidad de seleccionar la mejor máquina posible para el proyecto. Esto es, la máquina que proporcione el mayor rendimiento dado el presupuesto disponible. Adicionalmente el hecho de no estar atado a una plataforma concreta garantiza facilidad en futuras actualizaciones. Los sistemas multiprocesadores (múltiples CPUs integradas en un equipo), por ejemplo, parecen prometedores para este fin.

MONICOD es portable salvo por la dependencia con las librerías de adquisición de SAPERA.

5.3.1.2. Modularidad

La modularidad de un sistema viene dada por la capacidad de ser descompuesto en módulos independientes (o pseudos-independientes). Ver figura 5.18.

[3] Léase Windows 98/Windows XP/Windows 2000/Windows 2003/ Distribuciones Linux/UNIX/ MacOS.

[4] Así, i386/i686/ SPARC/ Macintosh.

Tales módulos pueden ser dotados de una interfaz que sigue algún tipo de estándar (front-end) y un motor que lleva a cabo el trabajo que da funcionalidad al módulo (back-end). La idea consiste en que módulos de idéntica funcionalidad aunque con construcciones en detalle distintas o, incluso muy distintas, compartan interfaces idénticas. Esto permite que desde el punto de vista del sistema externo al módulo, sean perfectamente intercambiables los back-end de módulos equivalentes.

Un ejemplo clásico es el que deriva del uso de librerías de cámaras. Una librería de cámara está destinada a ser utilizada con un modelo particular de cámara. Sin embargo la función primaria de cualquier librería de cámara es la adquisición de imágenes para un cliente computador. De este modo las especificaciones de cada cámara son solo detalles de los que es posible abstraerse. Sería factible tener diferentes módulos para cada modelo de cámara siendo todos ellos intercambiables. Si el desarrollo soporta de por sí esta capacidad, añadir nuevos módulos a un sistema ya construido se ve muy simplificado en términos de costo y esfuerzo. Añadir nuevas capacidades, mejorar las existentes o incluso crear nuevas familias de destrezas, aun para desarrolladores externos al sistema modular, se transforma en una tarea sencilla.

Siendo modulares se alarga la longitud de vida del desarrollo, así como su mantenimiento, reutilización, actualización, y versatilidad de cara a nuevos usos prácticos.

Aunque MONICOD es modular con respecto a la entrada o adquisición y la salida (son módulos independientes), se ha primado la eficiencia en el motor de validación. Por ello los algoritmos descritos en este documento, aunque pueden ser descritos en términos de módulos, la implementación llevada a cabo NO es modular existiendo una fuerte acoplamiento.

5.3.1.3. Transparencia

El sistema debe ser transparente. Esto es, debe ocultar detalles técnicos y matices de la implementación allí donde conocerlos no es necesario. Las operaciones no están vinculadas con el corazón del desarrollo, sino más bien con la entrada y salida de datos del sistema, y por ende no relacionado con el proceso crítico de validación de datos. Desgraciadamente en MONICOD la fase de calibrado requiere conocer el significado de parámetros ciertamente oscuros así como de aspectos internos de los algoritmos.

5.3.1.4. Eficiencia

La eficiencia es probablemente el objetivo primario. El motivo es evidente, y es la principal finalidad en la investigación subsiguiente de motores de validación. Muchas elecciones han estado guiadas por ese principio. Si bien la plataforma para pruebas no es en si misma el motor de validación sobre el que debe aplicarse el mayor esfuerzo de optimización, la eficiencia en todo el código es deseable. Primero porque se justifica mejor el esfuerzo en la modularidad y su posterior

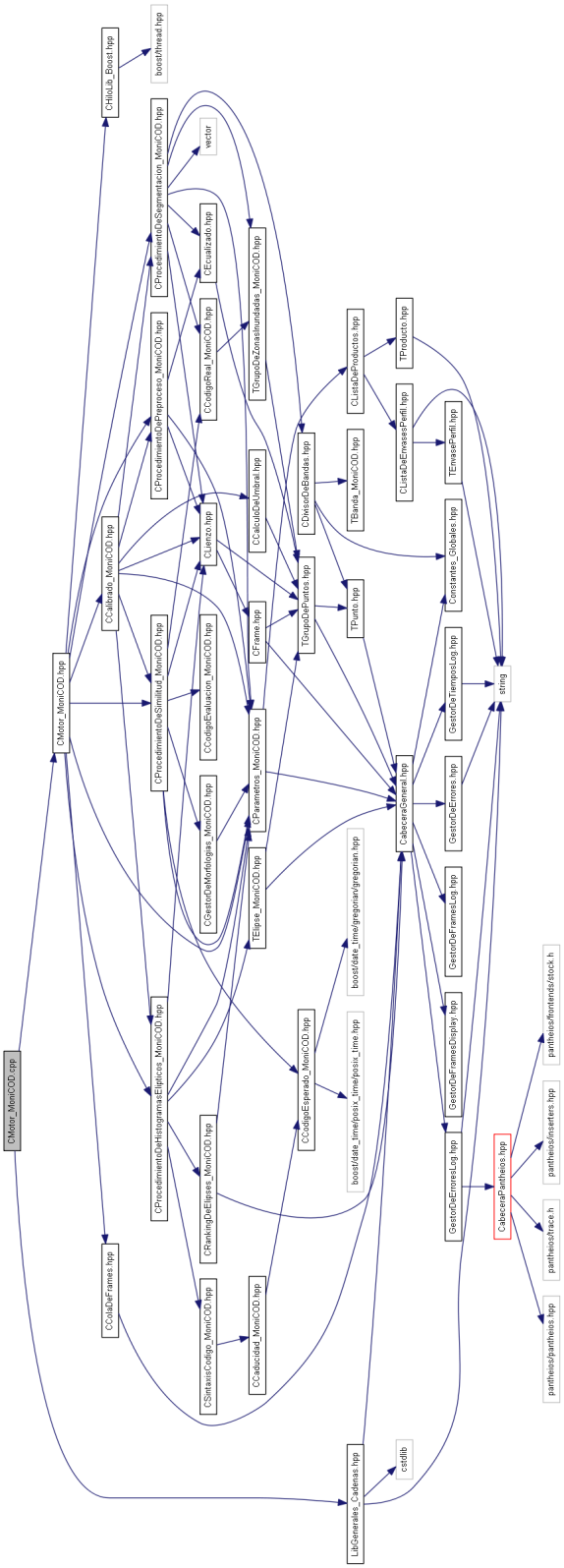


Figura 5.18: Los módulos de MONICOD

reutilización. Segundo porque tendremos un campo de pruebas para prácticas de optimización de código. Tercero porque un principio deseable de cualquier código que se implemente en cualquier área debe de ser la eficiencia, excepto cuando la elegancia y/o la simplicidad y/o la legibilidad del código sean prioritarios (lo que sería el caso de MONICOD).

5.3.1.5. Amigabilidad

La amigabilidad de sistema es una virtud deseable. De la misma manera que se ocultan las actividades necesarias, pero no relacionadas con el motor de validación, para facilitar las cosas al investigador es conveniente que la propia plataforma le sea fácilmente accesible. Esto se traduce en una interfaz visual comprensible, clara y auto-explicativa. También es uno de los retos más importantes de la plataforma. Las herramientas elegidas para su construcción, si bien notables en otros campos, como la portabilidad, no son precisamente las más potentes en la creación de interfaces. El problema reside en la difícil portabilidad de los aspectos visuales pues estos son muy dependientes del entorno en el que estén sumergidos. En definitiva, son difícilmente portables. Estos aspectos son controlados por librerías específicas del entorno, y las librerías genéricas y portables existentes no le sacan el máximo partido a las capacidades disponibles en cada uno de los entornos. En otras palabras, es posible crear interfaces mejores si se renuncian a otras directivas que hemos adoptado como básicas.

El usuario final de MONICOD es el que tiene la última palabra pero creemos que, en esta primera versión, se ha obtenido un resultado satisfactorio y "amigable".

5.3.1.6. Simplicidad

Características como la modularidad contribuyen decisivamente a mantener una estructura ordenada. Tal estructura sería fútil si no se mantuviera simple. Con la simplicidad contribuimos al mantenimiento posterior por terceros (e incluso por primeros), facilitamos la búsqueda y aislamiento de errores que puedan surgir, y aseguramos la legibilidad conceptual y de código para los desarrolladores presentes y futuros que deseen incorporar nuevas características. Somos de la opinión de que resulta erróneo pensar en la complejidad y la simplicidad como términos reñidos. Manteniendo una visión simple, que no simplista, pueden lograrse diseños sofisticados.

5.3.2. El lenguaje

C/C++. Lenguaje de propósito general de alto nivel capaz de operar en tiempo real. Código generado muy ligero con mínimo consumo de recursos.

- GCC.Compilador altamente optimizado. Eficiente, muy probado, amplia comunidad de soporte.

- Libre. Código Abierto (modificable, recompilable, adaptable)
- Multiplataforma Flexibilidad.
- Abundancia de librerías de proceso
- Librerías SAPERA: Lenguaje Natural C/C++
- Plataforma operativa de desarrollo final (funcional en Windows, Linux, MacOS)
- Interfaz preliminar multiplataforma
- Actualmente investigando interacción entre plataforma de desarrollo y SAPERA

5.3.3. Librerías de desarrollo

En la figura 5.19 tenemos una panorámica de las dependencias incidentales de la actual encarnación práctica de MONICOD. El uso de librerías externas para tratar con funciones periféricas como salvar/recuperar imágenes, la interfaz gráfica, manejo multiplataforma del sistema de archivos o la obtención de una fecha y hora por parte del sistema. Todas ellas son soportadas por diferentes sistemas operativos, son código abierto, y con licencias liberales que permiten un uso comercial con pocas restricciones.

Mención aparte merece el kit de desarrollo SAPERA. Es la única dependencia comercial aunque es de libre disponibilidad tan pronto adquieras un producto del fabricante.

5.3.3.1. SAPERA

Ese apartado está dedicado a una pequeña mención introductoria a las librerías de adquisición, esenciales para MONICOD. Probablemente el mejor modo de describir qué es la librería SAPERA es ofrecer la descripción proporcionada por el fabricante.

<<Sapera LT es una suite de librerías C/C++ de software independientes del hardware, para adquisición de imágenes, exhibición en pantalla y control, que es soportada por las plataformas Coreco para el tratamiento de imágenes. Entre sus características más notables se incluye portabilidad, control versátil de cámara y facilidad de uso en el desarrollo de aplicaciones. Sapera LT es compatible con Microsoft Visual Studio C/C++, .NET, Visual Basic 6.0 y Borland C++ Builder y soporte para Windows XP, 2000 y plataformas .NT>>

CORECO IMAGING
EXCELLENCE IN MACHINE VISION

Así pues, las librerías SAPERA son el mecanismo de comunicación que enlaza MONICOD con el sistema de adquisición de imágenes. Ver figura 5.20. En

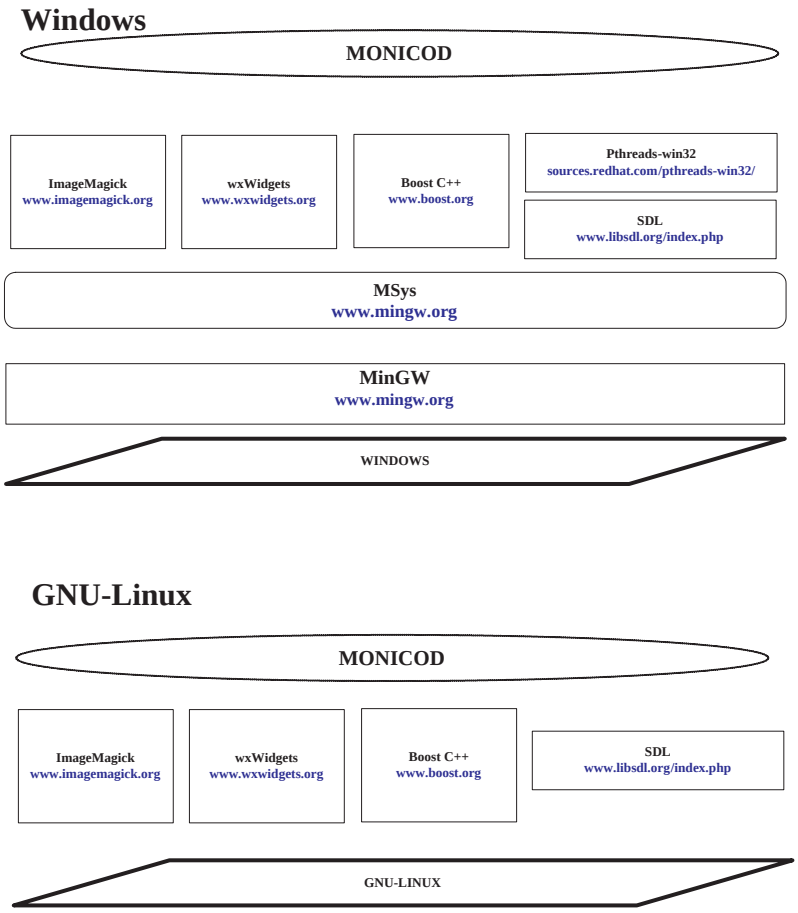


Figura 5.19: Capas de software entre MONICOD y el sistema operativo.

las figuras 5.21 y 5.22 se aprecian algunos detalles internos de la relación del hardware con el SDK Sapera.

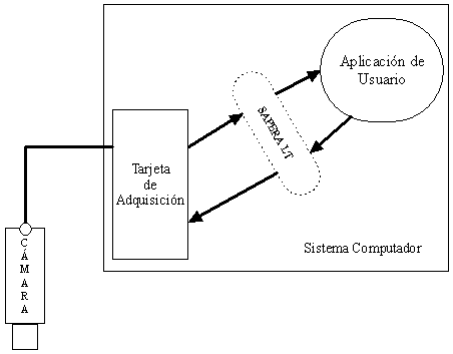


Figura 5.20: Relación de SAPERA con los componentes físicos Cámara y Tarjeta(Placa) de Adquisición y el componente software "Aplicación de Usuario"

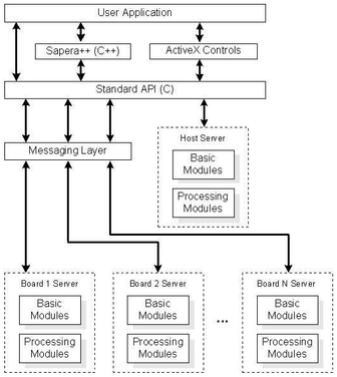


Figura 5.21: Representación en detalle del vínculo entre la aplicación de usuario (User Application) y la tarjeta de adquisición (Board 1 Server)

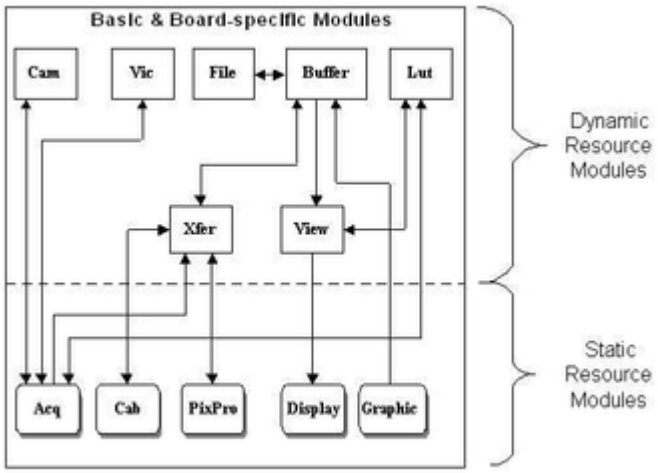


Figura 5.22: Diagrama de clases NO exhaustivo de Sapera

5.3.3.2. PThreads

Para aprovechar las características de los sistemas multiprocesador aprovechamos la gestión de hilos. Los hilos son necesarios para crear tareas paralelas o al menos concurrentes. El modelo de gestión de hilos elegido es Pthreads. Nativo de sistemas Posix (como Linux/Unix), soportado oficialmente por Apple/MacOS.

PThreads32 (ver 5.23) lleva a los sistemas Windows el modelo de gestión de hilos natural de Unix/Linux y otros sistemas POSIX. Desgraciadamente Pthreads32 no cubre por ahora la totalidad del estándar POSIX 1003.1-2001 que define cómo debe ser la API para escribir aplicaciones multihilo pero ha sido suficiente para nuestros propósitos.

Figura 5.23: Logo de Pthreads

5.3.3.3. wxWidgets

wxWidget (ver 5.24) es uno de los más potentes Frameworks disponibles para la creación de interfaces. Es código abierto, busca la portabilidad entre máquinas muy distintas (Windows, Unix/Linux, Macintosh, Palm OS...), y está especialmente dirigido a trabajar con C++. Ver [109].

Figura 5.24: Logo de wxWidgets

5.3.3.4. ImageMagick

ImageMagick (ver 5.25) es una poderosa librería destinada al tratamiento de imágenes estáticas. Probablemente sea una de las más completas y portables, aunando simplicidad con elegancia. ¡Y encima es libre!, mejorándose constantemente gracias a un ciclo de vida repleto de contribuciones. Tiene múltiples interfaces para diferentes compiladores como C, Delphi, Python,... Nuestro interés está dirigido a trabajar con C (cuya interfaz precisamente es el núcleo de la librería) y C++ (Magick++). Ver [110].

Figura 5.25: Logo de ImageMagick

5.3.3.5. Boost

Boost C++ (ver 5.26) es una polifacética colección de librerías que mantienen en común operar muy bien con el C++ Standard. Esto hace que sean portables, aun cuando se ocupen de temas muy dispares. Es usada como librería de complemento. Así guardar archivos o cronometrar tareas son tareas soportadas. Ver [111].

Figura 5.26: Logo de Boost C++

5.3.3.6. SDL

La librería SDL (ver 5.27) es un intento para hacer asequible al desarrollador las capacidades multimedia del sistema operativo y del propio hardware.

Las librerías base cómo DirectX y OpenGL (ver 5.28) tienden a ser demasiado confusas requiriendo una curva de aprendizaje muy pronunciada. Por otra parte DirectX no es una librería portable a sistemas Linux, y por tanto no es una opción. SDL está basada en OpenGL, y si bien no es la opción más rápida resulta adecuada para nuestros intereses.

Históricamente se utilizó para reemplazar el soporte de display (visualizado del vídeo por pantalla en "tiempo real") proporcionado por Sapera Coreco. Esto fue necesario para ofrecer un display que permitiera cargar imágenes y vídeos y no solamente *frames* rescatados de la tarjeta de adquisición. Además, queríamos un display que NO fuera dependiente del sistema operativo. De manera que con la ayuda de SDL se creó dicho display. Sin embargo, recientemente, el display SDL ha sido reemplazado por una versión más rápida de display construido directamente con OpenGL. Ver [112].

Figura 5.27: Logo de SDL

Figura 5.28: Logo de OpenGL

5.3.4. Compiladores y entornos de desarrollo.

5.3.4.1. GCC/gcc

- **GNU g++**
  - Compilador libre y de código abierto de C/C++. Ver 5.29. Maduro e inmensamente popular.

- **Codeblocks**
  - Codeblocks(ver 5.30) es un entorno libre y de código abierto de desarrollo C/C++. Es ambicioso y está diseñado para ser muy extensible y completamente configurable. Ver [113].

Figura 5.29: Logo de la familia de compiladores GCC, entre ellos el compilador g++.

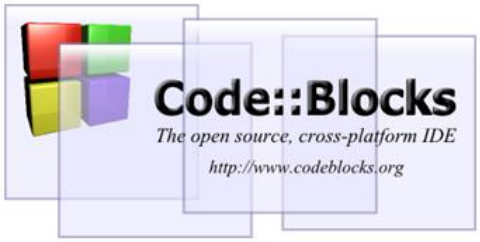


Figura 5.30: Entorno de desarrollo libre y abierto Codeblocks, compatible con g++ o MSVC.

- **MinGW**

MinGW (ver 5.31) es un proyecto libre y de código abierto que traslada a un entorno Microsoft Windows muchas utilidades GNU[5].

5.3.4.2. MSVC

MSVC de Microsoft Corporation (ver 5.32) es el compilador C/C++ por excelencia para entornos Microsoft Windows. Existe una versión gratuita sin interfaz gráfica. Se caracteriza por ser altamente compatible con el sistema operativo Windows pero no es capaz de generar código portable fuera de él. Es

[5]"GNU's Not Unix!" es un sistema operativo que tiene un diseño similar a UNIX, pero ofrece su software con licencias libres y no incluye código UNIX. MONICOD se limita a utilizar el compilador C/C++ y herramientas/utilidades asociadas.

Figura 5.31: Soporte para las utilidades en Windows, entre ellas GCC.

Figura 5.32: Entorno comercial de desarrollo para el compilador MSVC

destacable la madurez alcanzada en algunos de sus componentes, como el depurador. Ver [114]

Se ha trabajado en varias ediciones del producto.

- Visual Studio 2005
- Visual Studio 2008
- Visual Studio 2010

Capítulo 6

MONICOD: Preparativos

"Sobre lo que hay que pensar y hacer antes de empezar"

Este capítulo se divide en tres partes.

La primera parte expone nociones fundamentales y conceptos generales en el universo del motor de validación MONICOD.

La segunda parte recopila una serie de hechos básicos, muchos de ellos evidentes por si mismos, que de algún modo (directo o no) MONICOD aprovecha.

La tercera parte expone un protocolo de calibrado aún en estado incipiente.

6.1. La arquitectura del algoritmo

El algoritmo MONICOD ha sido dispuesto como una serie de test consecutivos (ver figura 6.1) que persiguen verificar diferentes propiedades. Esta serie de test es aplicada (al menos parcialmente) en cada *frame*. En este esquema de test consecutivos no es necesario llevar a cabo todos los test existentes para tomar una decisión de validación negativa. Esto es; si un test no es superado todos los test pendientes de realizarse no cambiarán la decisión adoptada de Validación Negativa[1] y pueden ser obviados porque no existe razón para ser ejecutados.

De lo dicho se sigue que:

- Un código validado como bueno (Validación Positiva) debe haber superado todos los test.
- Si falla cualquiera de los test el código no puede ser dado por bueno (Validación Negativa). Las razones por las que esto es factible serán descritas en la sección 6.4. Las razones por las que hacerlo así resulta especialmente conveniente son descritas en la sección 6.7.

[1] La lata tiene el código incorrecto o no es legible.

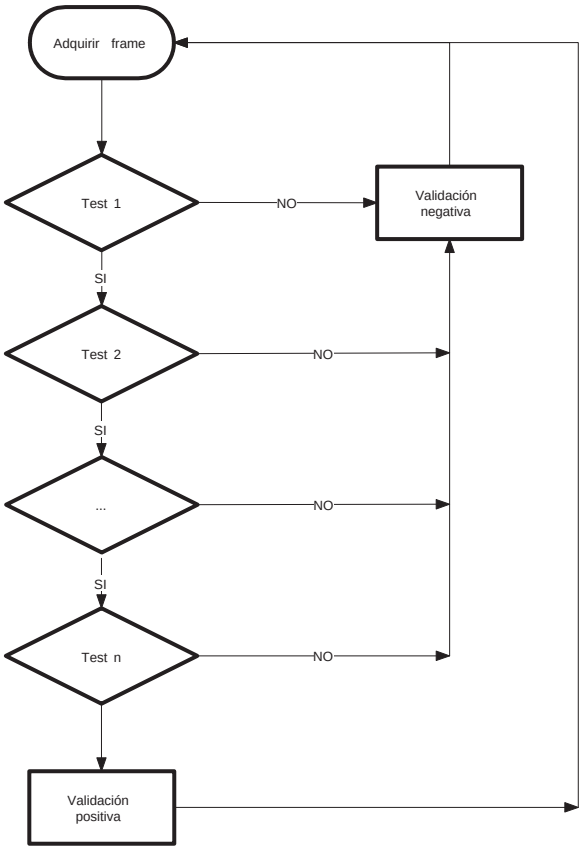


Figura 6.1: Test sucesivos

También ocurre que si un test no se supera los test subsiguientes sean innecesarios o incluso no aplicables. Por ejemplo, si el test de presencia de lata falla - no hay lata en el *frame* que se procesa-, el test de existencia de código es inaplicable porque no hay lugar -la lata- donde el código pueda (o no) estar presente.

De lo dicho se sigue que cada *frame* presenta una dificultad de procesamiento diferente. Es inmediato concluir que el peor caso (en tiempo de proceso) es aquel que nos obliga a realizar todos los test. Esto caso siempre ocurre en el caso de una Validación Positiva[2].

La secuencia de test es importante. Es conveniente que los test más decisivos y rápidos estén primero en la secuencia para intentar determinar cuánto antes si estamos en un caso de validación negativa.

En la sección 6.7 se pondrá de relieve la importancia de proceder así.

[2]La lata tiene el código correcto y este es legible.

6.2. Modos de trabajo en MONICOD

6.2.1. Modos de trabajo

En MoniCOD hay cuatro modos distintos:

- Modo/Tiempo de Calibrado

  MONICOD calibra los parámetros que utilizará para el resto de modos. Auto-ajuste de configuración con supervisión de un usuario debidamente entrenado.

- Modo/Tiempo de Validación

  Estado habitual de MONICOD. MONICOD debe determinar si las latas de la cinta transportadora son correctas o no.

- Modo/Tiempo de Aprendizaje

  Aprendizaje de MONICOD con supervisión de un usuario debidamente entrenado. Es un paso que hay llevar a cabo tanto para validar como para reconocer.

- Modo/Tiempo de Reconocimiento

  El reconocimiento de MONICOD es el modo más lento. Busca 'reconocer' el texto. El reconocimiento se centra en determinar qué es lo que está escrito y no si eso escrito es o no válido.

Estos modos se activan de forma exclusiva (sólo un modo a la vez). Comparten algoritmos, y sus propósitos están relacionados.

6.2.2. Orden de ejecución entre modos

Existe un orden de ejecución natural de los modos (expuesto en la figura 6.2).

El **modo de calibrado** configura la calidad y condiciones de la adquisición de imágenes. Es un modo previo a cualquier otro modo. Una vez establecida dicha calidad y condiciones raramente deberemos volver a él mientras la plataforma física (soporte físico, iluminación, cámaras, lentes,...) no se vea modificada.

El **modo de aprendizaje** establece el grueso del conocimiento del sistema sobre las formas de carácter con las que tendrá que tratar. Un cambio de fuente, o un cambio de escala en la impresión de caracteres puede obligar a reejecutar este modo de trabajo. Es altamente recomendable lanzar este modo inmediatamente después de la entrada en el modo de calibrado (que habitualmente implicará un cambio en las condiciones de calibrado).

El **modo de validación** es el modo de operación habitual de MONICOD. Una perdida de eficacia en este modo puede significar tener que lanzar una vez más el modo de calibrado y/o el modo de aprendizaje.

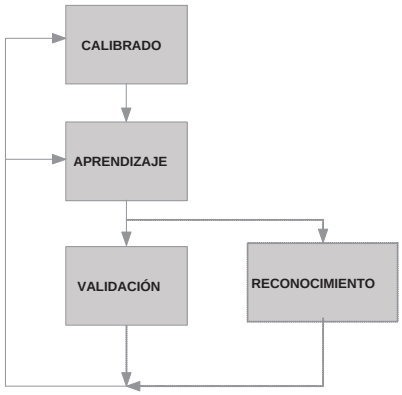


Figura 6.2: Modos de trabajo. El primer paso siempre es calibrar. El segundo paso aprender. El tercer paso dependerá de si se quiere validar o reconocer. Llegará un momento en que el sistema se degrade de tal manera que lo aprendido o incluso lo calibrado deje de ser válido. Es el momento de reiniciar el ciclo.

El **modo de reconocimiento NO** es una operación habitual en MONICOD. Aquí también una perdida de eficacia en este modo puede significar tener que lanzar una vez más el modo de calibrado y/o el modo de aprendizaje.

6.2.3. Modo de validación y modo de reconocimiento

Los modos de reconocimiento y validación comparten una gran parte de las actuaciones. La principal diferencia reside en que la validación cuenta con la figura del código esperado. El código esperado proporciona (de manera infalible) qué caracteres deben ser recuperados de la lata. Los caracteres recuperados corresponden o no a los esperados. El código esperado actúa como un mecanismo de supervisión infalible que nos permite evaluar si las latas están bien impresas o no. Y también, adaptar la validación a cambios progresivos en la forma de los caracteres. Todo esto será desarrollado con profusión en el apartado 8.1.

Por otro lado en el modo reconocimiento no se cuenta con la figura de código esperado. En otras palabras, el modo de reconocimiento no tiene prejuicios sobre qué caracteres cabe esperar recuperar de la lata. Fácilmente un reconocedor puede convertirse en un validador utilizando un comparador entre la salida reconocida y el código esperado 6.3.

Pero MONICOD no ha seguido ese curso de acción en aras de obtener velocidades mayores. Un reconocedor tiene que considerar más información con el propósito de enfrentar todos los caracteres que conoce a las formas recuperadas de la lata para identificarlas. En cambio un validador restringe el dilema a determinar si las formas recuperadas de la lata corresponden con los caracteres que se esperan.

Este documento se centra en la actuación primaria de MONICOD: La validación.

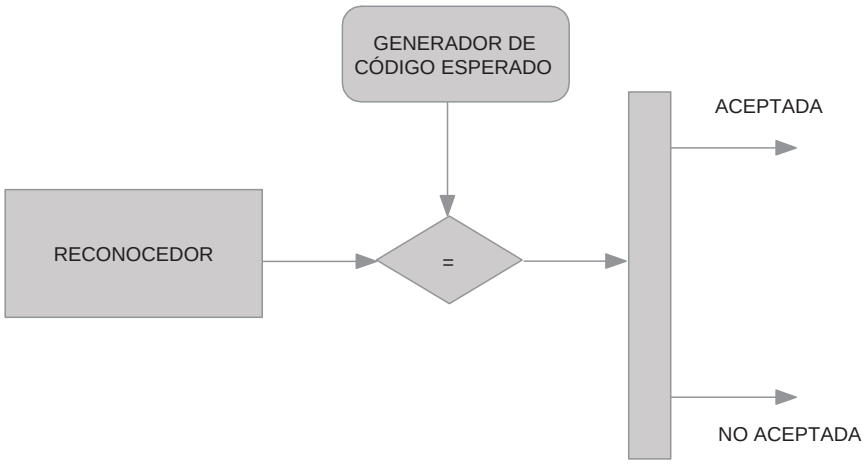


Figura 6.3: Reconocedor convertido en validador

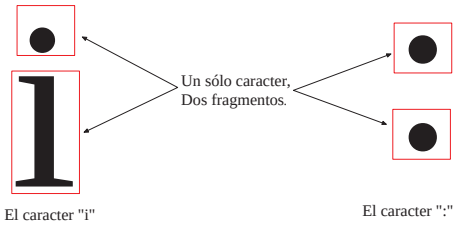


Figura 6.4: Fragmentación natural en caracteres fragmentarios

6.3. Conceptos previos

6.3.1. Caracteres

El carácter, especialmente en el lenguaje escrito alfabético, es la unidad básica.

En general, en el lenguaje natural escrito un carácter tiene escaso significado semántico por si mismo aunque puede afectar notablemente el significado general que pretende transmitirse. El conjuntor "y" o el disyuntor "o" en castellano son ejemplo de esto. **Este no es el caso de la codificación industrial, donde cada carácter tiene un fuerte significado característico.**

El alfabeto[3] común (del tipo con que opera MONICOD) tiende a caracteres con una forma sin fisuras. Aunque hay excepciones:

Un fragmento de carácter es una forma donde todos los puntos que la com-

[3] Alfabeto que da soporte a la lengua escrita inglesa.

ponen presentan conectividad entre sí. Gran parte de los caracteres constan de un único fragmento. La característica "Ñ" es un carácter compuesto por dos fragmentos. Todas las letras acentuadas también: á, é, í, ó, ú. Los puntos que componen el cuerpo principal del carácter no tienen conectividad con los puntos que componen la tilde. Ver figura 6.4.

En MONICOD llamaremos a esta particularidad gráfica *caracteres constituidos por más de un fragmento*. A efectos prácticos no hay en nuestro dominio caracteres constituidos por tres fragmentos o mas. Sin embargo ocurre que un carácter puede resultar fragmentado accidentalmente por una mala impresión (escenario muy habitual) o por ruido en la imagen pero no dejando de ser por ello legible. Esta es una de las razones por las que los caracteres constituidos por más de un fragmento representan un particular desafío que se examinará con detalle en7.3.1.5.

6.3.1.1. Niveles de carácter

- El carácter a nivel físico

  Los caracteres corresponden a formas en la imagen. Físicamente están hechos de tinta sobre la superficie metálica de la lata.

- El carácter a nivel visual

  En la imagen aparecen como concentraciones de píxeles con baja luminancia en la superficie de la base de la lata.

- El carácter a nivel lógico

  Corresponden a identidades lógicas que tienen significado concreto dentro del código. Al tratarse de un código basado en caracteres

6.3.2. Palabras

La palabra es un elemento constitutivo del lenguaje humano. Un vocabulario es una colección de palabras. Un diccionario es un compendio de definiciones de palabras. En lenguaje escrita alfabética una palabra es una determinada combinación, de longitud variable, de caracteres donde puede haber repetición. La identidad de la palabra viene dada por el orden posicional y por la presencia de tales o cuales caracteres. Pero además por el contexto en el que está sumergida.

Para un reconocedor de caracteres es importante separar palabras, obteniendo la ventaja adicional de poder dar uso a diccionarios de palabras

> En MONICOD no existe la "palabra". La palabra tiene existencia gráfica, pero su importancia esta concentrada en su papel lingüístico en el concierto del mensaje construido en lenguaje natural humano. La codificación con la que trata MONICOD en cambio no utiliza palabras para transmitir su mensaje. En cada superficie de lata hay un conjunto de caracteres que debe ser validadas y ninguna palabra.

Líneas En escritura occidental se lee (esto es; se reconocen) de izquierda a derecha "líneas" de texto, y de arriba a abajo las diferentes "lineas" que componen el texto completo.

Texto En MONICOD el texto completo es el conjunto de líneas, palabras, caracteres y espacios que forman la totalidad del mensaje. El mensaje es el código.

El espacio Con espacio hablamos de áreas de no-tinta que, sin embargo aportan información (crítica) del texto. Sin el espacio el texto puede ser ilegible o cargado de ambigüedades. Hay varios tipos de espacio:

- *Interlineado* (espacio entre líneas)
  - El espacio entre líneas es aquella región sin tinta que separa distintas líneas de texto. Dicho de otro modo, las líneas están separadas entre sí por el espacio de interlineado. Ver 6.6
- Espacio *interpalabra*.
  - El espacio *interpalabra* separa las palabras. El uso del espacio como separador es muy importante en lenguaje occidental escrita. Aunque el tamaño medio de estas y la frecuencia de espacios *interpalabra* varía de una lengua a otra. Como en MONICOD no existen palabras, no hay espacios *interpalabra*.
- Espacio *intercaracter*.
  - El espacio *intercaracter* separa a los caracteres y asegura la individualidad de estos. Ver 6.6
- Espacio *intracaracter*.
  - El espacio de *intracaracter* natural es aquel espacio que separa los fragmentos en caracteres fragmentados naturalmente como los descritos en 6.3.1. Probablemente la "i" (cuerpo de la i y punto de la i) ofrezca el ejemplo de espacio *intracaracter* más común.
  - Hablamos de espacio *intracaracter* "natural" en oposición al que se produce en una mala segmentación. Esto se tratará con mayor detalle en 7.3.1.5.
- Espacio interior
  - El espacio interior es parte constitutiva del carácter y es característico. Aparece en forma de huecos/agujeros cerrados. Ver 6.5 y 6.6.

Figura 6.5: Ejemplos de caracteres con huecos interiores.

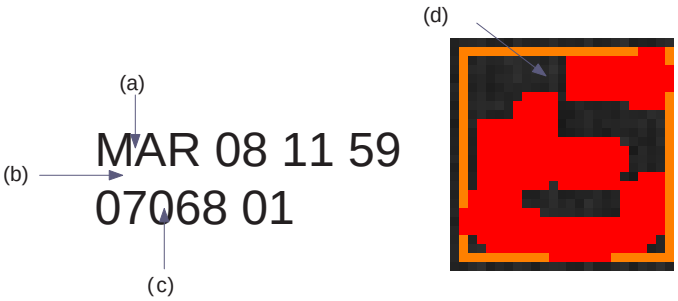


Figura 6.6: (a) Espacio intercaracter. (b) Interlineado. (c) Espacio Interior. (d) Espacio que fragmenta un carácter artificialmente..

6.3.3. Continuidad de un carácter

En MONICOD la continuidad de un carácter tiene que ver con la ausencia de espacios intercaracter (naturales o accidentales) en horizontal y en vertical.

En particular es especialmente importante contemplar la continuidad vertical en el procedimiento de búsqueda de espacios de interlineado

- Continuidad vertical

  Decimos que un carácter presenta *continuidad vertical* cuando se cumple que no es posible trazar una línea horizontal (dentro del rango de extremos superior e inferior del caracter) que no colisione con el carácter.

- Continuidad horizontal

  Decimos que un carácter presenta *continuidad horizontal* cuando se cumple que no es posible trazar una línea vertical (dentro del rango de extremos izquierdo y derecho del caracter) que no colisione con el carácter.

6.4. Los hechos

6.4.1. Imágenes y escenas

Empezamos desde una cámara con la que obtenemos una vista aérea de latas que se desplazan sobre una cinta transportadora. La cámara está fijada a la plataforma física según consideraciones descritas en los capítulos preceden-

tes y proporciona las imágenes, pero la interpretación de la escena procede de MONICOD.

- Imagen. Parrilla bidimensional de valores de iluminación. Ver figura 6.7.
- Escena. Una aproximación lógica a los elementos que aparecen retratados en las imágenes. Ver figura 6.7

Hechos

1. Capturamos imágenes con cero, una o más bases de lata. La base de lata es una superficie parte de la lata, mas o menos circular, NO pintada ni decorada, hecha de aluminio u hojalata donde se halla impreso el código de caducidad de la lata. Ver figura 6.8.
2. La iluminación utilizada hace que un gran porcentaje de la superficie que no corresponde a la base de la lata sea más oscura que la superficie que sí corresponde. Ver figura 6.9.
3. No todas las escenas contienen bases de lata. NO siempre pasan latas sobre la cinta transportadora, y por tanto, la cámara captura imágenes sin latas. Consecuentemente en esas capturas no aparece ninguna base de lata. Ver figura 6.10.
4. Un código de caducidad es un conjunto de caracteres alfabético-numéricos ordenados de una determinada manera impresos en las bases de lata. Ver figura 6.11.
5. El trazo de los caracteres es siempre de un color con intensidad muy baja[4]. Los códigos de caducidad se inscriben con tinta negra. Ver figura 6.12.
6. Las bases de lata contienen los códigos de caducidad[5]. Fuera de una base de lata NO hay código de caducidad. Ver figura 6.13.
7. Las bases de lata PUEDEN no aparecer completas en la imagen. Si la cinta transportada está en marcha toda lata sobre ella está en tránsito (se mueve). La cámara lleva a cabo capturas de imagen en intervalos regulares fijos así que una base de lata puede aparecer total o parcialmente en el campo de visión de la cámara. Ver figura 6.14.
8. Así mismo un código de caducidad PUEDE no aparecer completo en la imagen[6]. Ver figura 6.15.

[4] Los códigos de caducidad se inscriben con tinta negra en los casos estudiados.

[5] ... y sólo un código de caducidad a la vez porque solo debe haber un código de caducidad en cada lata. Pero esto no deriva del estudio de la escena, sino de la definición de código de caducidad y de las directrices del procedimiento de etiquetado de latas, que establecen una-lata-un-código.

[6] Determinar si el código está completo o no no compete sólo a la escena. Si el codigo de caducidad no está completo, entonces una base de lata no está completa, pero su viceversa no es cierta. Por otra parte, un código de caducidad incompleto sigue siendo "un conjunto de caracteres alfabético-numéricos ordenados de una determinada manera impresos en las bases de lata". Saber si no está completo compete a instancias que van más allá del procesamiento de imágenes bruto.

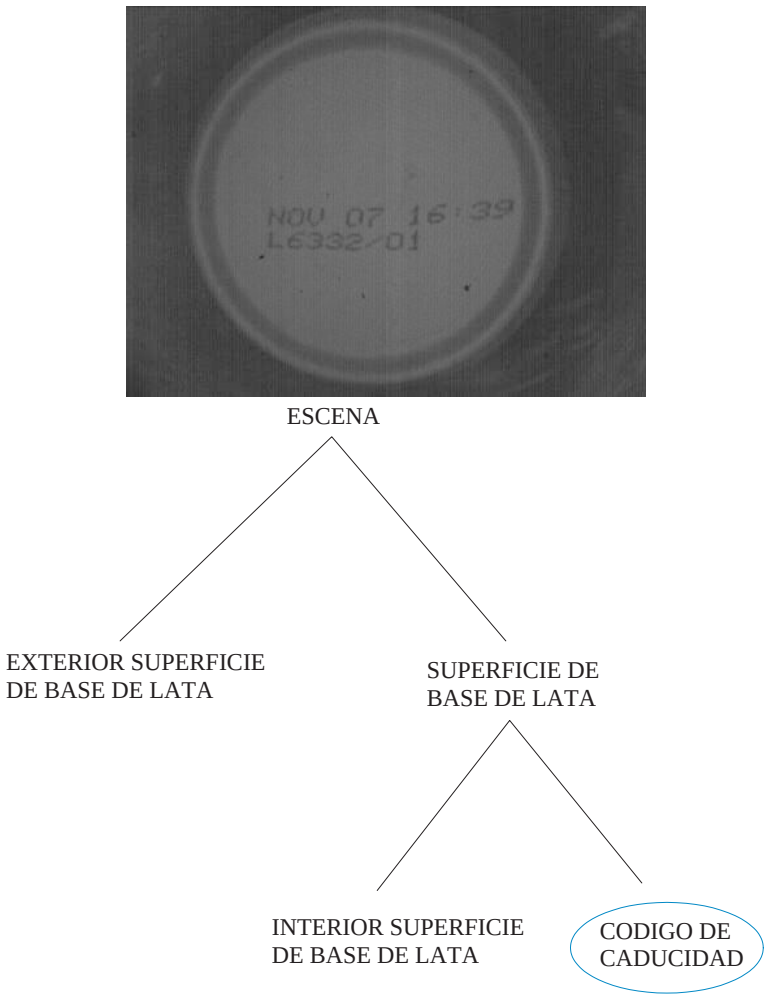


Figura 6.7: Parrilla de valores de iluminación y descomposición lógica de la escena. (☼)

Figura 6.8: Superficie de base de lata. (☼)

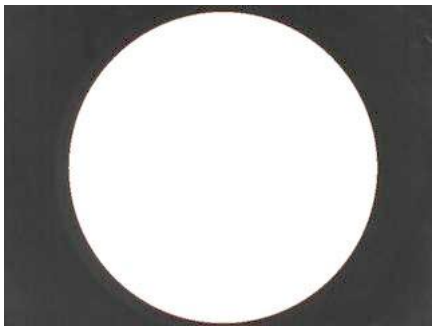


Figura 6.9: Exterior de una superficie de lata. (☼)

Figura 6.10: Un *frame* sin latas. (☼)

Figura 6.11: Código de caducidad extraído de un *frame*. La información para la validación está aquí.

Figura 6.12: Región de la lata que contiene el código. (☼)

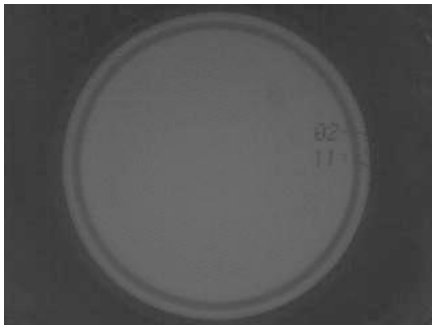


Figura 6.13: Todo código que no esté dentro de la superficie de la base de lata, simplemente *no* está. (☼)

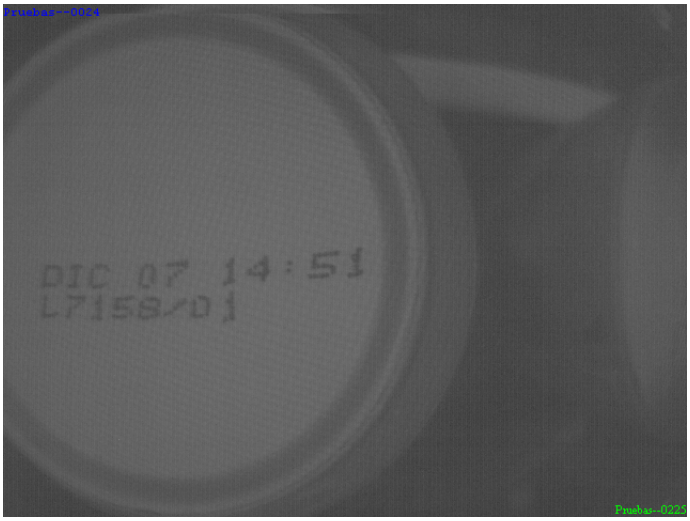


Figura 6.14: La superficie de la base de lata aparece cortada. (☼)

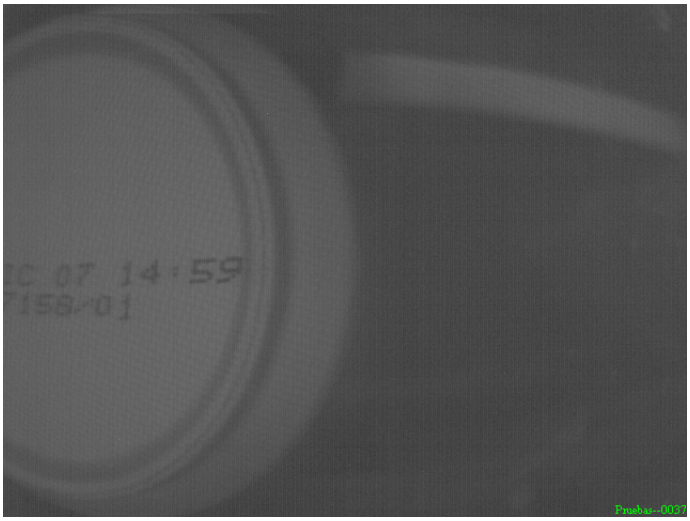


Figura 6.15: El código de la base de lata aparece cortado. (☼)

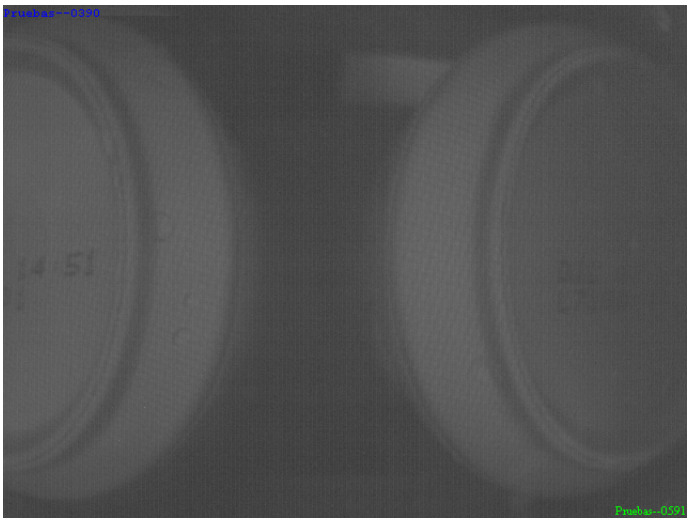


Figura 6.16: Dos latas en un *frame*. (☼)

9. Si las latas sobre la cinta transportadora están muy cerca, varias bases de lata pueden aparecer simultáneamente en la captura. Ver figura 6.16.

10. Las imágenes son rectangulares. Las bases de lata son aproximadamente circulares. El código de caducidad es "texto".

Con estos hechos es posible una descomposición de la escena en sus partes. Distinguimos en la escena tres objetos. Ver figura 6.17.

- Código de caducidad
- Base de lata.
- Fondo. Región exterior de la base de lata.

6.4.2. Código de caducidad: El objeto de interés

En el 'objeto de interés' está todo lo que necesitamos de la imagen. Si el objeto de interés está bien definido y su presencia NO cubre toda la imagen sino una parte de ella, carece de sentido desviar nuestros recursos en tratar áreas que sabemos no contienen nada del objeto de interés. Sin embargo:

1. En muchas aplicaciones es imposible establecer con seguridad donde se hallará el objeto de interés por tanto resulta inevitable explorar toda la imagen para localizarlo.

2. En otras ocasiones, el objeto de interés cubre un área de la imagen demasiado extensa para que consideremos descartar una parte significativa de la imagen como objeto de estudio.

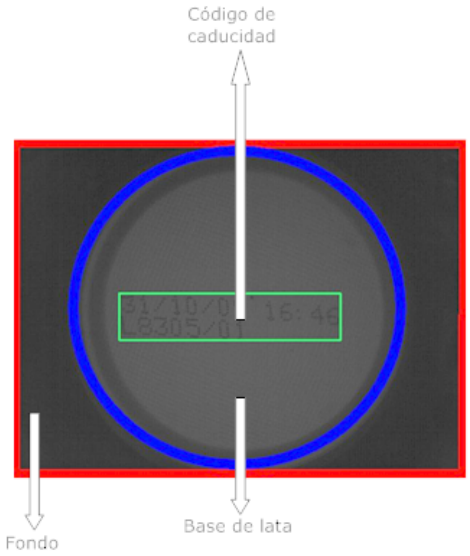


Figura 6.17: Elementos de la escena en el *frame*. (☼)

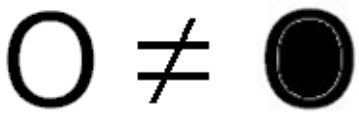


Figura 6.18: La "O" se ve muy distinta sin su característico espacio interior.

3. No existen objetos de interés propiamente dichos. En tales casos la respuesta a la pregunta "¿Qué ocurre en esta imagen?" no es respondida en términos simples de objetos interesantes sobre un fondo no interesante.

Para nuestros propósitos es posible aseverar que:

- *El objeto de interés por excelencia es el código de caducidad.*

La validación de códigos sólo necesita el código de la imagen para llevar a cabo la validación. Nuestra área de interés será entonces el área de la imagen que contiene el código de caducidad. En otras palabras: Sólo con esa parte de la imagen tendremos todo lo que necesitamos de ella. Ver figura 6.12.

Es importante denotar que para identificar un carácter importa tanto donde hay tinta, como donde no la hay. Ver 6.18

6.4.2.1. Tamaño/área

En nuestro escenario el tamaño, entendido como las dimensiones de un polígono que circunscribe el objeto de interés (ver figura 6.19), se mantendrá fijo:

1. La distancia entre la cámara y la lata permanece inalterable así como ópticas y resolución empleada permanecen fijas durante el tiempo de operación.

2. Los parámetros hardware y software del sistema de impresión (distancia entre cabeza impresora y lata, tamaño de la fuente,...) permanecen inalterables durante el tiempo de operación.

Dentro de un *frame* el porcentaje de imagen que contiene sólo y exclusivamente el objeto de interés se mantiene más o menos constante con las siguientes excepciones:

- Lata no visible. Ver figura 6.10.
- El código es sólo parcialmente visible. Ver a este respecto figuras 6.15 y 6.16.
- Código ausente. Ver a este respecto figura 3.23.
- Código defectuoso. Ver figura 6.13.

Número de caracteres En general el número de caracteres del código no varía durante la jornada. Una longitud fija simplifica la comparación y detección de discrepancias o la interpretación del código por parte del usuario.

Tamaño de fuente de caracteres El tamaño de carácter es un parámetro de configuración del sistema impresor tal como lo es la fuente de texto o el estilo, que también afectan al tamaño. Es especialmente importante la distancia de la cabeza impresora a la lata porque para una misma fuente de texto, estilo... se obtienen diferentes tamaños con diferentes distancias cabeza impresora-lata. En cualquier caso, una vez establecida una configuración no suele variar durante la jornada.

Píxeles ocupados por la tinta Fijado el número de caracteres y tamaño de fuente de caracteres él área ocupada por la tinta en la lata puede variar entre latas según los caracteres presentes en cada una de ellas.Ver figura 6.20. Sin embargo estas variaciones no tienen impacto en el tamaño del área de interés[7].

- *El tamaño del objeto de interés en la imágenes procesadas por MONICOD es relativamente fijo.*

[7] Entendido como las dimensiones de un polígono que circunscribe el objeto de interés.

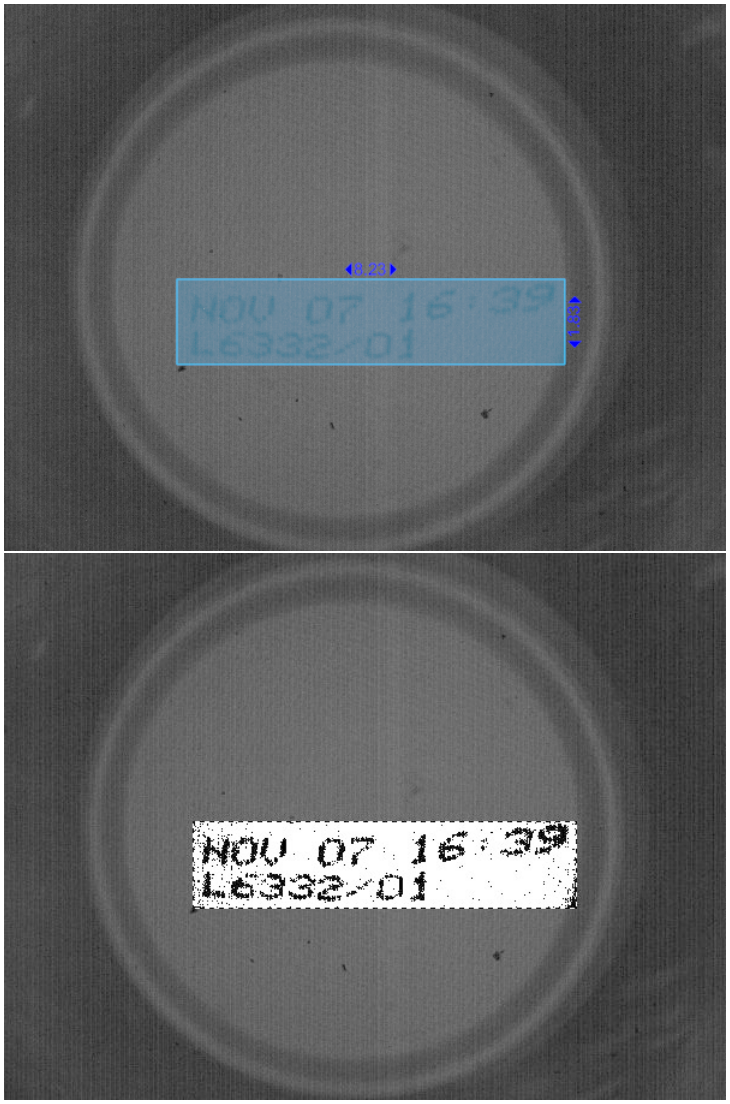


Figura 6.19: Arriba, rectángulo y sus dimensiones(en cm) que circunscribe el código. Abajo, se ha llevado a cabo un binarizado del interior del rectángulo evidenciando el área real que es cubierta por la tinta. (☼)

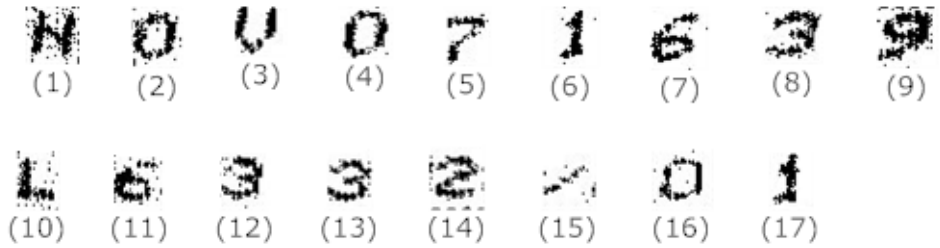


Figura 6.20: Caracteres distintos pueden ocupar más o menos espacio. Un "1" ocupa menos que un "9", y así requiere menos tinta y menos píxeles.

6.4.2.2. Posición

Desafortunadamente el código no ocupa posiciones fijas de *frame* a *frame*.

Con cada *frame* es necesario localizar de nuevo el código porque la lata se mueve en el campo de visión. Esto es; Debe buscarse el código en cada *frame*. Ver figura 6.21.

Existen desplazamientos horizontales (ver sección 3.7) y verticales (ver sección 4.7). Pero algunas ubicaciones de código son más probables por la propia configuración de la cámara frente a la cinta transportadora. Ver a este respecto la figura 6.26 que muestra, de forma aproximada, áreas de probabilidad de aparición del código. Los desplazamientos en la dirección de movimiento de la lata son mucho mas acusados que en la dirección perpendicular a este.

6.4.3. Base de lata: El contenedor del objeto de interés

- *El código de caducidad está inscrito única y exclusivamente en las bases de las latas.*

Por ende, si podemos leer un código de caducidad de una captura entonces, al menos una parte considerable de la base de la lata aparece en la imagen. Esa base contiene el código. El código NO puede estar fuera de una base de lata. Hay una relación física, y bastante obvia, de pertenencia entre base de lata y código. Ver 6.22.

En una base de lata distinguiremos dos partes:

- Superficie de lata que incluye la sección sobre la que se ha llevado a cabo la impresión del código de caducidad. Naturalmente este área es del máximo interés.
- Superficie de lata que NO incluye la sección sobre la que se ha llevado la impresión del código de caducidad.

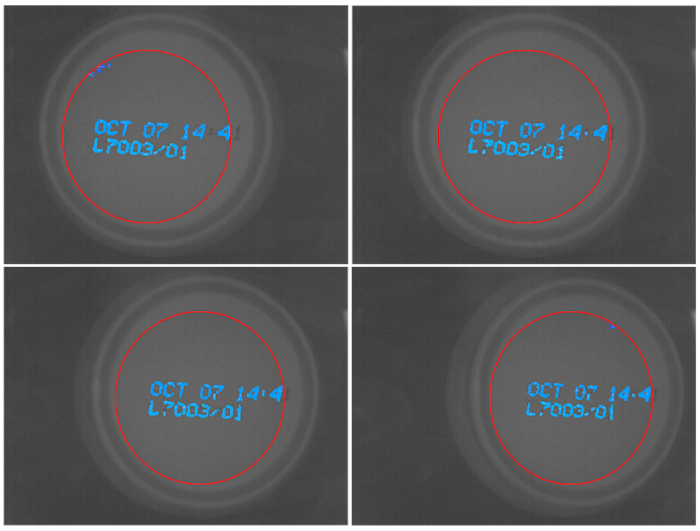


Figura 6.21: Misma lata en movimiento. Diferentes posiciones para el código. La lata se desplaza de izquierda a derecha frente a la cámara. Los *frames* están dispuestos en la ilustración de izquierda a derecha y de arriba a abajo. (☼)

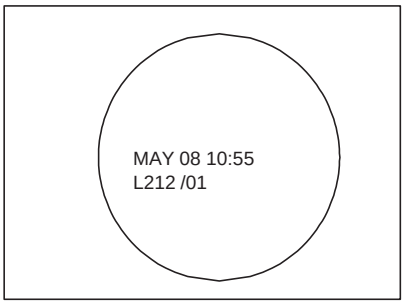


Figura 6.22: Representación de código y frontera entre región interior y exterior de la base de lata.

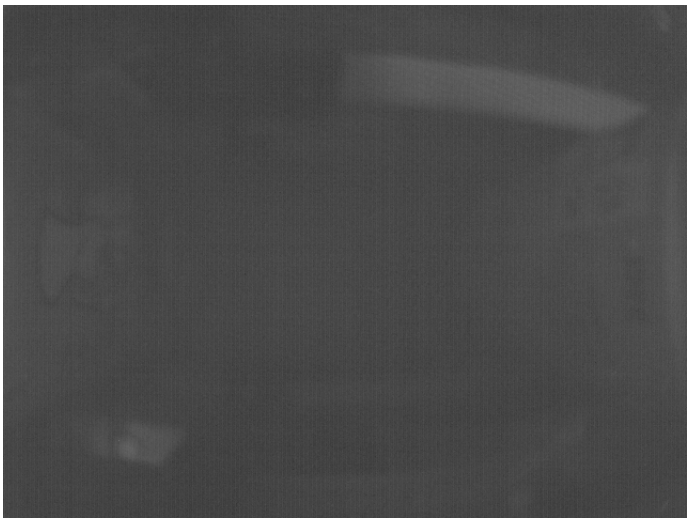


Figura 6.23: "Área oscura" . (☼)

6.4.4. Fondo: La parte "no interesante"

Región exterior a la base de la lata de interés. Este elemento PUEDE invadir toda la imagen. Por ejemplo, cuando NO hay una lata de interés delante de la cámara. Este elemento, a su vez, contiene dos subelementos:

- "Área oscura". Zona sin presencia de lata. Con iluminación convencional sería posible ver la cinta transportadora, el suelo y otros elementos ajenos al objeto de la validación y al universo de la escena. En la figura 6.23 que corresponde a la situación de un *frame* sin lata mostrada en la figura 6.10 sólo tenemos "área oscura".
- Superficies de base de latas que no son la lata de interés. Habitualmente una lata entrante (será procesada en instantes posteriores) o saliente (ha sido procesada en instantes anteriores). En 6.16 a la izquierda una lata entrante y a la derecha una lata saliente. Ver figura 6.16 que ejemplifica una situación en la que aún en presencia de latas y muy poca "área oscura" está desposeída de partes interesantes.

6.4.5. La composición de la escena

Componiendo la escena, como si se tratara de un mosaico podemos obtener información interesante. Las diferentes partes y relaciones de contención entre ellas son mostradas esquemáticamente en la figura 6.24. En la figura 6.25 la disección de un *frame* bajo este enfoque.

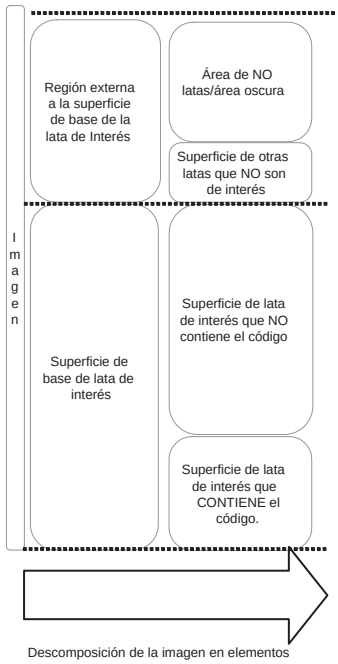


Figura 6.24: Concepto esquemático de diferentes elementos en la imagen y su relación con el objeto de interés.

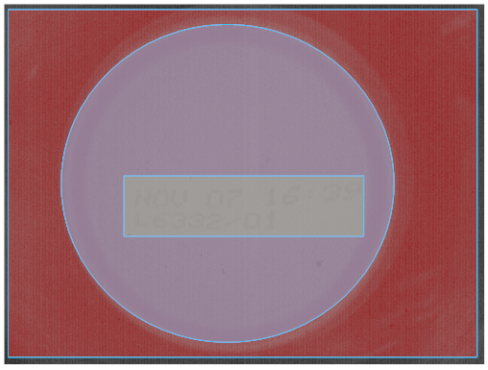


Figura 6.25: Regiones de importancia de la lata. En rojo región externa a la superficie de base de la lata. En azul región interna a la superficie de base de la lata. En verde superficie de la lata que contiene el código. (☼)

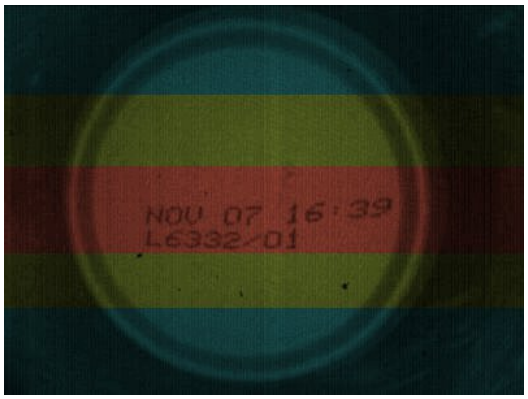


Figura 6.26: Regiones de aparición de código menos probable en colores más fríos, en contraste con regiones de aparición de código más probable con colores más cálidos.

Los elementos de la imagen anteriormente descritos no aparecen siempre en las mismas posiciones de la imagen. Ninguno de los elementos comentados está sujeto a ningún recinto riguroso de la imagen. Sin embargo, verdaderamente es posible garantizar regiones de la imagen que sólo serán frecuentadas por algunos elementos. Esto se hace físicamente en la fase de calibrado (ver sección 6.8) situando la cámara de una determinada manera y a una cierta distancia con respecto a la cinta transportadora. Ver figura 6.26.

De lo anterior es fácil deducir que el elemento "Código de Caducidad" es sólo un elemento más que compone la imagen. Además el elemento "Código de Caducidad" está entre los elementos que no aparecen siempre en todas las imágenes encaradas por MONICOD, de manera que:

- *El objeto de interés no está presente en todas las imágenes que MONICOD debe examinar.*
- *El objeto de interés, cuando aparece, lo hace en zonas (o regiones) de la imagen predecibles. Mientras tanto, en otras zonas NO aparece nunca.*

6.5. Áreas de interés: Definición, detección y localización

6.5.1. ¿Qué es el área de interés?

Área de interés es aquella región de la imagen que se procesa para obtener información de interés. Las áreas de interés actúan a modo de filtro, 'cegando' al sistema a lo que cae fuera de ellas. y evitando que sea considerado. Esto no

es contraproducente porque la información que aparece dentro del recinto del área es la necesaria y suficiente para cumplir con los objetivos.

Remarcamos que, en lo que respecta a MONICOD, las áreas de interés son recintos de la imagen y existen haya o no un objeto de interés presente. No están condicionadas por su presencia.

En MONICOD las áreas de interés son definidas en un momento previo a la validación. Durante la validación permanecen inmutables. Por tanto, en ningún caso se invierte precioso computo de operación en darles forma.

6.5.2. Motivaciones del área de interés

Las motivaciones para mantener un sistema basado en área de interés son:

1. Eficiencia

   Manteniendo áreas de interés reducimos drásticamente el volumen de datos de entrada con relativamente poco esfuerzo. MONICOD hace un uso tan intensivo de las áreas de interés que NUNCA barre por completo el *frame* bajo ninguna circunstancia. Esto es posible porque para saber si una imagen tiene o no elementos de interés NO es necesario examinar toda la imagen. En su lugar, basta con examinar las áreas de interés y comprobar allí si hay presencia de objeto de interés o no. Lo que haya en áreas de no interés no merece ninguna consideración y es desestimado.

2. Recursos de proceso liberados

   Ha quedado establecido que no hay motivos para desviar recursos de proceso abordando áreas que no contienen nada interesante para nosotros. Reduciendo el campo de entrada liberamos recursos que opcionalmente pueden emplearse en otros menesteres. No supone ningún sacrificio con respecto a la calidad del resultado porque cualquier esfuerzo de cómputo invertido en un área sin interés repercutirá en poco o ningún beneficio. Y dado que los recursos de computo son limitados, es interesante aplicarlos preferentemente en áreas fértiles en información valiosa antes que en áreas estériles[8].

3. Menos distracciones en un entorno más controlado

   Otra ventaja adicional que no debe menospreciarse es que, en un entorno tan controlado como MONICOD, áreas más pequeñas implican universos más pequeños sujetos a un menor número de estados posibles. Esto es; Los elementos presentes y los sucesos posibles en el área de entrada son abarcados más fácilmente. Si puede hacerse una buena clasificación entre ellos sesgamos efectos de distracción o confusión en el sistema que puede conllevar a falsos negativos o falsos positivos.

[8] El caso más evidente de esto lo tenemos con cualquier imagen donde no haya latas. Si no hay lata no hay código por tanto no es necesario seguir invirtiendo recursos en la imagen.

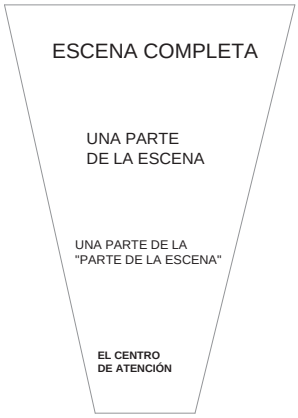


Figura 6.27: Descomposición sucesiva para identificar el área de interés.

4. Son prácticamente gratuitas

   Por último, en un dominio tan controlado como el que enfrenta MONICOD la construcción de las áreas de interés -esto es la selección de los puntos que forman parte de cada área- puede llevarse a cabo en un momento previo a la validación. Durante el tiempo de validación el sistema sólo consulta las áreas de interés predefinidas.

6.5.3. Selección del área de interés

Por supuesto sólo podremos beneficiarnos de las ventajas de las áreas de interés si la selección que hacemos es adecuada. Para la selección correcta el conocimiento del dominio es fundamental.

Para empezar resulta evidente que el área donde reside el objeto de interés es el área de interés ideal. En nuestro caso ya quedó establecido en la sección 6.4 que el objeto de interés corresponde al código de la lata. Pero llegar hasta ella, localizarla e imponer fronteras fijas en un único paso no es trivial. Una aproximación por descomposiciones sucesivas puede resultar más conveniente. Ver figura 6.27.

Según este procedimiento de análisis es necesario recurrir a diferentes etapas de descomposición cuando alcanzar la descomposición última directamente no es viable o poco práctico. La descomposición consiste en dividir el todo en sus partes. Cada parte es susceptible de ser descompuesta a su vez.

En MONICOD toda área de interés que contenga el fragmento de imagen donde está presente el código de caducidad hereda esa importancia. En la figura 6.28 resulta evidente que la base de la lata, o al menos una parte de ella son candidatos naturales a área de interés.

Por otro lado no siempre está presente el código de caducidad. Cuando eso ocurre la imagen carece de interés. No necesitamos procesarla más allá de lo necesario para decidir si contiene o no un objeto de interés.

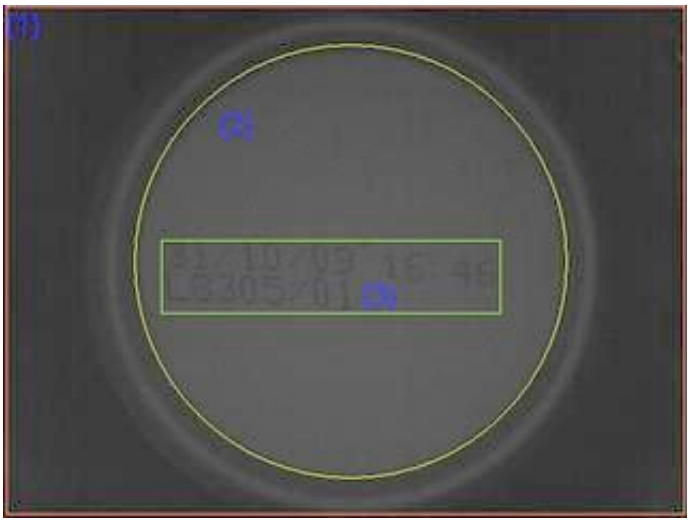


Figura 6.28: Áreas de interés. (☼)

Es importante subrayar una vez más que las áreas de interés demandan entornos controlados en donde se conoce perfectamente qué escenarios pueden ocurrir delante del dispositivo capturador de imágenes.

Cómo será descrito más adelante MONICOD mantiene sus propias áreas de interés, sin embargo es un concepto tan extendido que muchas cámaras lo incorporan por hardware. Ver figura 6.29. MONICOD ignora esas soluciones protegiendo su independencia de un hardware concreto.

6.5.3.1. El tamaño del área de interés

Entre mayor sea esa área de interés más fácil será localizarla e imponer sus fronteras. Pero inevitablemente la posibilidades de que contenga otros elementos además del objeto de interés son mayores. Entre mayor cantidad de objetos no interesantes a tener en cuenta en el área de interés mayor será el esfuerzo de discernir entre ellos y tomar una determinación. Este esfuerzo de refinamiento puede amortiguarse si nos esforzamos en encontrar área de interés más pequeñas pero igualmente certeras.

6.5.3.2. La posición del área de interés

En la sección 4.7.1 se examino los desplazamientos de la lata en el tramo sistema-impresor MONICOD de CEC[9] y CCC[10]. Esto queda expuesto en la

[9] Compañia Embotelladora de Canarias
[10] Compañia Cervecera de Canarias

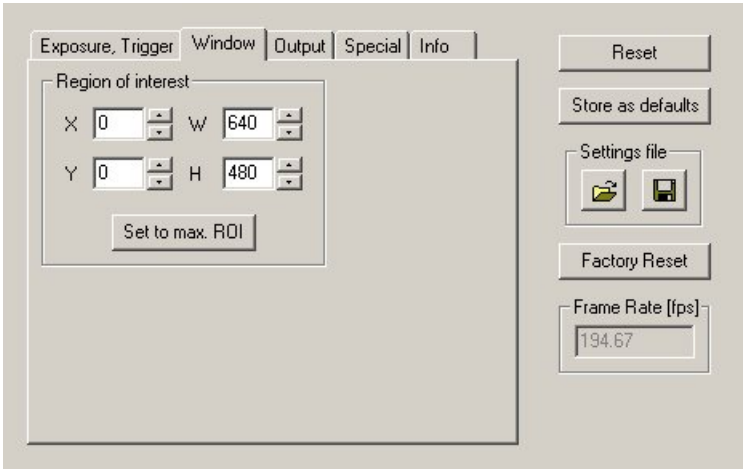


Figura 6.29: El uso de áreas de interés está contemplado por los drivers de algunas cámaras industriales. Arriba la configuración de un área de interés rectangular a través del driver de un fabricante.

figura 4.14 donde puede apreciarse como las latas siguen un trayecto mas o menos rectilíneo al atravesar el campo de visión de la cámara. Como el código de caducidad es solidario a la lata, se imprime el código en aproximadamente el mismo lugar de la lata (establecido el sistema de impresión), y se adquieren las imágenes desde la misma posición (establecida la plataforma física MONICOD), los códigos de caducidad (cuando aparecen) se ubican dentro de la imagen en posiciones predecibles.

6.5.3.3. La forma del área de interés

Ver figura 7.10.

6.5.3.4. Áreas de interés en MONICOD

En MONICOD existen dos áreas de interés.

1. Áreas de interés de contorno de superficie que determinan la presencia o ausencia de contornos de lata. Por cada *frame* todas las áreas de interés de contorno son procesadas.

2. Áreas de interés de superficie destinadas a procesar la mejor lata. Se procesa (eventualmente) una sola de estas áreas de interés por lata.

Situación	Respuesta del sistema	Denominación
El código esperado **SI** corresponde con el código obtenido.	Código Válido	Validación positiva
El código esperado **SI** corresponde con el código obtenido.	Código **NO** Válido	Falsa validación negativa
El código esperado **NO** corresponde con el código obtenido.	Código Válido	Falsa validación positiva
El código esperado **NO** corresponde con el código obtenido.	Código **NO** Válido	Validación negativa

Figura 6.30: Situaciones MONICOD

6.6. Salida del sistema

Estas son las posibles situaciones a que da lugar la salida MONICOD. MONICOD proporciona tres posibles respuestas para cada *frame*.

1. Ignora el *frame*

   En MONICOD es el estado de NO respuesta. Indica que los *frames* que recibe no contienen latas por alguna razón. Por ejemplo no hay latas en la cinta transportadora o la cámara no está bien emplazada.

2. El código impreso es juzgado como válido. Validación positiva.

   En MONICOD, el sistema incrementa el contador de latas totales y válidas de manera visible.

3. El código impreso es juzgado como NO válido. Validación negativa.

   El sistema incrementa el contador de latas totales y NO válidas de manera visible. Eventualmente puede enviar una alarma mediante sirena, retirar la lata mediante algún mecanismo de expulsión,... Todo ello dependerá de la política elegida ante fallos.

Naturalmente MONICOD podría errar en su veredicto. Podemos contemplar estas situaciones.

1. Ignora *frames* que NO debe ignorar.

2. Falsa validación positiva. El código es inválido pero se responde con una validación positiva.

3. Falsa validación negativa. El código es válido pero se responde con una validación negativa.

El cuadro 6.30 condensa las situaciones comentadas.

El grado de no deseabilidad de las falsas validaciones depende del criterio de verificación que se elija.

- Si se quiere ser muy sensible y dar alarma ante cualquier sospecha de error las falsas validaciones negativas no nos preocuparán demasiado.

- Si se quiere evitar alarmas y detenciones siempre que sea posible, minimizando demoras en la producción, entonces se querrá estar muy seguro de que el error sea real. En este escenario falsas validaciones positivas podrían ser tolerables.

6.7. La adquisición

Es importante dedicar algo de tiempo a conocer con algún detalle como opera el mecanismo de adquisición de imágenes a alta velocidad. Evitaremos entrar en los aspectos de la cuestión más técnicos y específicos al hardware.

Para nuestros efectos la adquisición trata de una transacción entre dos entidades. SAPERA y MONICOD.

- SAPERA (como quedó explicado en 5.3.3.1) es la API que trata directamente con los driver de las tarjetas de adquisición de *Coreco Imaging* (ahora *Teledyne Dalsa*) .

  Debemos subrayar, sin embargo, que el procedimiento de adquisición presentado, basado en llamadas a una función de callback, es común en la tecnología de adquisición y tratamiento por computador. SAPERA puede fácilmente ser substituida por otras APIs asociadas a tarjetas de adquisición de otros fabricantes.

- MONICOD es el objeto de estudio de este documento.

6.7.1. El procedimiento

Esta transacción sigue este esquema:

1. Inicialización de recurso

   En esta etapa MONICOD ordena a SAPERA que se adueñe del recurso Tarjeta y recurso Cámara para nuestro uso. Se inicializan estructuras internas. Se configura la adquisición de acuerdo al modo de trabajo. Se reserva un área de memoria principal inviolable donde son depositadas

```
CORSTATUS  CCONV  Funcion _Callback _SAPERAC
                              (        void  *contexto)
{

...
...
...

return   CORSTATUS _OK;
}
```

Figura 6.31: La función de Callback

las imágenes de adquisición. Esta zona queda dispuesta como un vector de imágenes de n elementos. Todos los recursos son utilizados de modo exclusivo por SAPERA. Ninguna otra aplicación tiene acceso a ellos mientras estén asignados.

2. Registro

   Aquí indicamos a que función queremos que invoque[11] cada vez que el sistema tenga una imagen lista. Esta es la función es denominada función "callback" (ver figura 6.31) . La función de "callback" es parte de MONICOD pero sólo es invocada por SAPERA.

3. Ciclo de adquisición

   Los principios del mecanismo de "callback" son elementales. MONICOD nunca solicita imágenes al driver, ni tiene control sobre la llegada de estos a él. Se limita a esperar pasivamente a que SAPERA le llame con el nuevo *frame*.

   a) MONICOD da la orden: Iniciar la adquisición.

   b) SAPERA gestiona el hardware de adquisición e inicia la adquisición.

   c) SAPERA espera un evento. Cuatro posibles eventos:

      1) SAPERA recibe el evento de recepción de nuevo frame. INTENTA copiar el *frame* en la siguiente zona libre de la memoria principal reservada. Pueden ocurrir dos cosas en ese intento.

         a' Existe una zona libre. El *frame* se copia en esa zona. SAPERA bloquea la zona marcándola como ocupada. SAPERA invoca a la función de callback. SAPERA pasa de nuevo a "Esperar un evento".

         b' Una zona libre no existe. **El frame se descarta** y SAPERA pasa de nuevo a "Esperar un evento".

11

a) La invocación consiste simplemente en llamar a una función "registrada" por MONICOD.

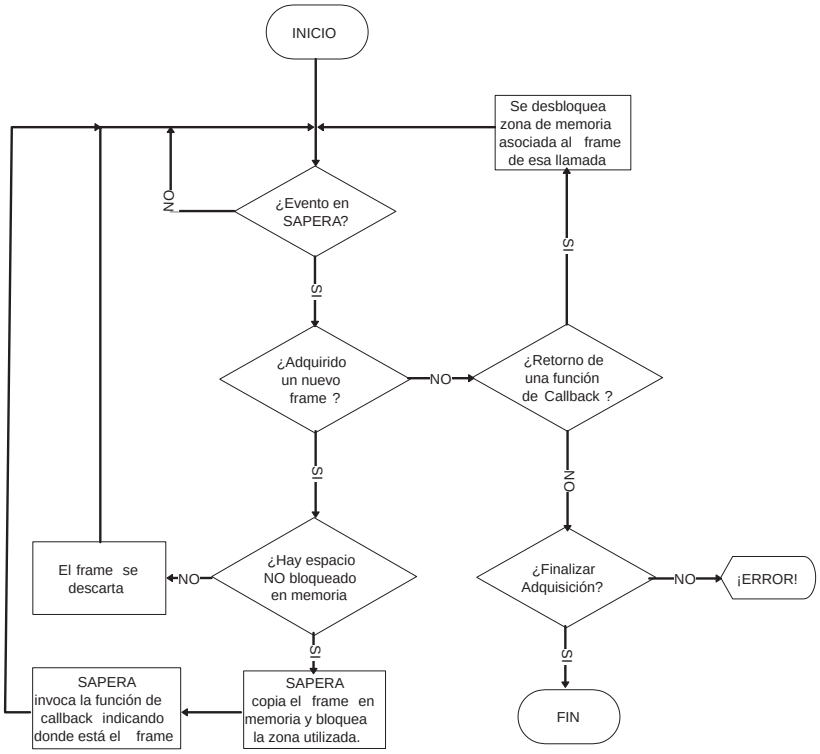


Figura 6.32: Ciclo de adquisición desde el punto de vista de SAPERA

2) SAPERA recibe la notificación de que una función de callback invocada ha retornado. Desbloquea la zona de memoria del *frame* asociado a esa llamada, y deja libre la zona para futuros *frames*.
3) SAPERA recibe la orden de finalizar la adquisición. SAPERA la ejecuta y se da fin al ciclo de adquisición.
4) Cualquier otro evento es inesperado y es interpretado como un error en la adquisición o en la transacción. SAPERA da fin al ciclo de adquisición.

Ver el esquema de flujo en figura 6.32.

4. Liberación de recurso.

6.7.2. Memoria Cíclica

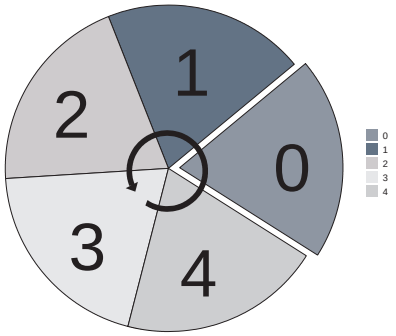


Figura 6.33: Recreación de un buffer de memoria cíclico capaz de albergar cinco *frames*.

En MONICOD/SAPERA un *buffer* de memoria cíclica de *frames* es una estructura capaz de contener un número N de *frames*. Ver figura 6.33. SAPERA se ocupa de almacenar los *frames* que adquiere en un *buffer* de memoria cíclica.

Ante la recepción de un nuevo *frame* siempre se intenta usar preferentemente el espacio libre (espacio aún no utilizado). Si se ha consumido el espacio libre se tratará de utilizar el espacio NO bloqueado que exista en el *buffer*. La zona de memoria NO bloqueada que contenga el *frame* más antiguo se sobrescribe, perdiéndose definitivamente.

En cualquier caso la zona que se acaba de escribir (o sobrescribir) es bloqueada inmediatamente por SAPERA. SAPERA desbloqueará zonas de memorias del *buffer* a petición de MONICOD. MONICOD solicitará su liberación cuando haya acabado con el *frame* contenido en dichas zonas.

Si no hay zonas NO bloqueadas no es posible el almacenamiento, y sólo cabría esperar a que se liberen. Para SAPERA esperar implica retener el actual *frame* y detener la adquisición. Sin embargo MONICOD/SAPERA NO puede esperar. Los *frames* están llegando continuamente desde la fuente de adquisición, y no es posible almacenarlos en ninguna otra parte. En lugar de eso el *frame* es "descartado". Descartar consiste en ignorar el *frame* (no almacenándolo) y permitir que SAPERA continúe la adquisición con normalidad. Los descartes son contabilizados así que, al menos, queda un rastro de su existencia.

Tengamos por ejemplo, un *buffer* de memoria de un sólo *frame*.

- Caso 1

  Llega el primer *frame*. El *buffer* está libre. El *frame* es copiado en el *buffer*. El *buffer* está bloqueado. Se invoca a la función de callback. Llega

el segundo *frame*. El *buffer* está ocupado. El *frame* es descartado.

- Caso 2

  Llega el primer *frame*. El *buffer* está libre. El *frame* es copiado en el *buffer*. El *buffer* está bloqueado. Se invoca a la función de callback. La función de callback retorna. El buffer es desbloqueado. Llega el segundo *frame*. El *buffer* está libre. El *frame* es copiado en el *buffer*. El *buffer* está bloqueado. Se invoca a la función de callback...

6.7.3. Consideraciones

El esquema de adquisición nos instruye con la siguiente información clave.

- SAPERA es la parte activa de la adquisición. MONICOD es pasivo respecto a la adquisición..
- MONICOD no tiene control sobre los tiempos con que van llegando *frames* del sistema de adquisición.
- SAPERA descarta frames sistemáticamente si MONICOD no es capaz de acogerlos. No obstante mantiene un registro del número de descartes hechos.
- Si quiere evitarse el descarte de frames MONICOD debe estar disponible para cada *frame*. Para ello **debe retornar de la función de callback antes de la recepción del siguiente *frame***.
- Es prioritario optimizar el flujo de las operaciones que se ejecuten en el cuerpo de la función de callback.
- Retornar la función 'mucho' antes de la recepción del siguiente *frame* no supone ventajas o desventajas al funcionamiento global. NO hay pérdida de *frames* . Pero es una buena noticia. Significa que podemos ajustar al sistema a una cadencia de operación mayor porque tenemos margen.

6.8. Preparativos iniciales: El calibrado

6.8.1. Las razones del calibrado

La plataforma física MONICOD es rígida. La estructura permanece fija, limitando modificaciones mecánicas en lo posible. Es trasladable a diferentes

localizaciones y permite cierto margen de maniobra, pero siempre como un todo, y no en sus diferentes partes. La causa de estos imperativos debe buscarse en la delicadeza del equilibrio óptimo que permite alcanzar las altas velocidades de validación exigidas.

El corazón de MONICOD es la colección de algoritmos de procesamiento de imagen que se ejecuta de forma permanente mientras está en activo el sistema. Dada una imagen (*frame*) los algoritmos deben tomar una decisión respecto a él. Por ejemplo:

1. No hay lata a procesar.
2. La lata existe pero el código está ausente.
3. La lata existe, el código está presente, pero es ilegible.
4. La lata existe, el código está presente, es legible, pero no es el esperado.
5. La lata existe, el código está presente, es legible, es el código esperado. Por tanto validación superada.

Ahora bien, estas decisiones hay que hacerlas en un tiempo límite. Por tanto, conviene reducir, tanto como sea posible, las tareas a llevar a cabo en la toma de decisiones. De hecho, tanto es así que es la primera medida a adoptar en el diseño de dicha colección de algoritmos.

¿Qué tareas pueden ser prescindibles? Existen varios conjuntos de tareas de los que es posible "evadirse" en modo de validación. Es crítico aclarar que esta evasión no debe menoscabar la calidad de la consecución del objetivo perseguido. En realidad evitar las tareas superfluas es la primera táctica comprometida con nuestro objetivo.

El conjunto de tareas "sorteable" que nos ocupa en este apartado no es, ni más ni menos, que el conformado por aquellas que pueden ser desplazadas a antes o después del periodo crítico de validación. Así pues aquí tenemos a los mecanismos destinados a garantizar una calidad mínima de imagen.

6.8.2. Protocolo de calibrado

El protocolo de calibrado establece la línea de acción a seguir cada vez que se inicia el sistema tras un periodo más o menos largo de inactividad, ha habido desplazamientos de la plataforma de soporte, se ha modificado consciente o inconscientemente alguna característica de la estructura cámara-óptica-iluminación, o se sospecha de un malfuncionamiento del sistema.

Aquí se detallan algunas situaciones que pueden motivar un recalibrado:

- Periodo de inactividad prolongado.
- Cambio de ubicación del sistema de validación.
- Micro-desplazamientos del soporte.

- Cambio o variaciones graves de iluminación.
- Modificaciones en la localización de la cámara.
- Modificaciones sobre el sistema de impresión(configuración, localización del cabezal impresor...)
- Alteraciones en la óptica
- ...

El protocolo de calibrado comprende los siguientes pasos. El orden persigue seguir una lógica de afinamiento de sistema.

1. Inspección directa

   Con la inspección directa tomamos conciencia de los aspectos más generales que permiten al sistema funcionar. Sólo superada esta, los test de calibrado subsiguientes son aplicables.

2. Test de calidad de imagen

   El test de calidad de imagen determina cuan buena es la captura que se está llevando a cabo desde el punto de vista de la máquina (en oposición al punto de vista humano). Los propios algoritmos destinados a la validación nos proporcionarán esa medida de calidad.

3. Test de preproceso

   El test de preproceso tiene dos metas. La primera es ayudarnos a conocer como de bien, o de mal, está ajustado el sistema. Esto forma parte de lo que se espera de un proceso de calibrado. La segunda meta es igualmente trascendente: Recolectar información útil para los algoritmos de trabajo intensivo en la verificación de código. Hay buenas razones para que ambas metas estén combinadas en este apartado. La principal debe resultar obvia. Sólo seremos capaces de sustraer esta información de preproceso en un sistema aceptablemente bien ajustado, de manera que si no es posible hacerlo algo falla en el ajuste, y no debe resultar difícil de encontrar. Al fin y al cabo hemos llegado a este punto después de haber superado dos test primarios que han segado las dificultades más acuciantes

4. Ajuste de parámetros.

6.8.3. Inspección directa

La utilidad *Inspección Directa* permite ajustar a grandes rasgos las características generales del sistema. Para ello el operario tiene acceso al display de sistema, desde el cual puede conocer con exactitud y en tiempo real el campo visual que está cubriendo la cámara. Ver figura 6.34.

Figura 6.34: Inspección directa del proceso

El display de sistema es un área del monitor que proyecta las imágenes que captura la cámara en tiempo real. Su uso está limitado a circunstancias excepcionales tales como el calibrado debido que es una herramienta que consume serios recursos en tiempo.

La inspección directa pretende ser un mecanismo ágil, que además resulta amigable de cara al operario/usuario. Traslada información muy valiosa que posibilita la aplicación de mejoras notables a la calidad del sistema. Entre sus beneficios tenemos:

- Es amigable porque es intuitivo, y no requiere otro entrenamiento que la experiencia de uso.
- Es rápido y fácil porque el sistema realimenta directamente al operario/usuario.
- Es valioso porque de forma inmediata puede conocerse el estado de una gran parte de los elementos que componen la plataforma.
- La validación implica que la inspección directa sea practicable. Si no es posible una inspección directa, el sistema simplemente no funcionará.

Las características que permite evaluar la inspección directa son:

- Ajuste de eje principal (izquierda, derecha, arriba, abajo) del sistema (eje de visión)
- Campo de visión despejado
- Campo de visión suficiente
- Campo de visión correcto
- Distancia de la cámara a la cadena de montaje.
- Distancia de la cámara a la fuente de iluminación.
- Iluminación suficiente
- Iluminación no sobresaturada.
- Primera aproximación a un eje principal de visión perpendicular al eje de desplazamiento de los centros de latas.
- Primera aproximación a un enfoque óptimo de la óptica.

Sin embargo la inspección directa tiene una importante limitación: Un ser humano NO puede hacer fácilmente medidas de calidad objetivas aproximadas en varios órdenes de precisión. Esta incapacidad probablemente sea debida que el cerebro humano no necesita una calidad excesiva. Nuestros mecanismos de tratamiento de ruido y extracción de la información son tan potentes que las menudencias de falta de calidad de imagen nos pasan desapercibidas. Así puede darse la paradoja de que prefiramos una imagen peor contrastada que otra mejor contrastada, y nos obstinemos en suponerla de mejor calidad.

6.8.4. Enfoque

6.8.4.1. Maximización de la magnitud del gradiente

Existe una relación entre el grado de contraste de la imagen y la calidad del enfoque: Una imagen con un enfoque deficiente presenta difuminados los bordes de los objetos respecto a la misma imagen bien enfocada. Con esto principio en mente podemos evaluar la calidad del enfoque con un detector de bordes como el operador de Sobel[115] visto en la sección 2.5.4. El método consiste en estimar la magnitud del gradiente y utilizar el criterio de que un punto estará tanto mejor enfocado cuanto mayor sea dicha magnitud. El problema ahora es determinar cuando la magnitud del gradiente puede considerarse o no grande o pequeña. Por fortuna en el dominio de MONICOD es posible designar imágenes tipo: Por ejemplo una imagen con una lata centrada. Es posible obtener un buen enfoque manualmente para esa imagen tipo y calcular entonces la magnitud de gradiente de referencia para los procedimientos automáticos.

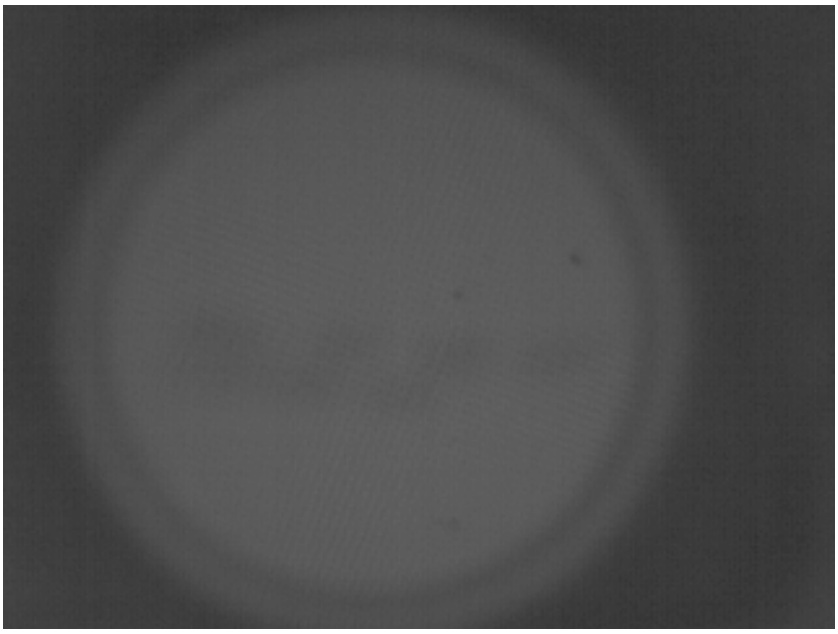


Figura 6.35: Un mal enfoque que impide ver el código. (☼)

En las figuras 6.35, 6.36 y 6.37 ejemplos de diferentes enfoques.

6.8.4.2. Varianza del nivel de gris

Por otra parte puede intuirse que valores altos de la varianza del nivel gris están asociadas con estructuras de la imagen enfocadas y viceversa[26]. La definición estándar de varianza es:

$$\sigma^2 = \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} (f(x,y) - m)^2 \tag{6.1}$$

donde m es la media de la distribución de los niveles de gris y $f(x,y)$ es la intensidad de los niveles de gris. La varianza se obtiene para una vecindad dada por la variación de x e y desde 1 hasta N. El criterio consiste en maximizar σ^2

6.8.5. Iluminación: Cálculo de umbral óptimo

Los problemas de iluminación en MONICOD son una cuestión central. Como se describe en el capítulo 3 para aliviarlos se utiliza una cámara oscura y una fuente de iluminación especializada. Sin embargo, NO es suficiente. En 10.2 se

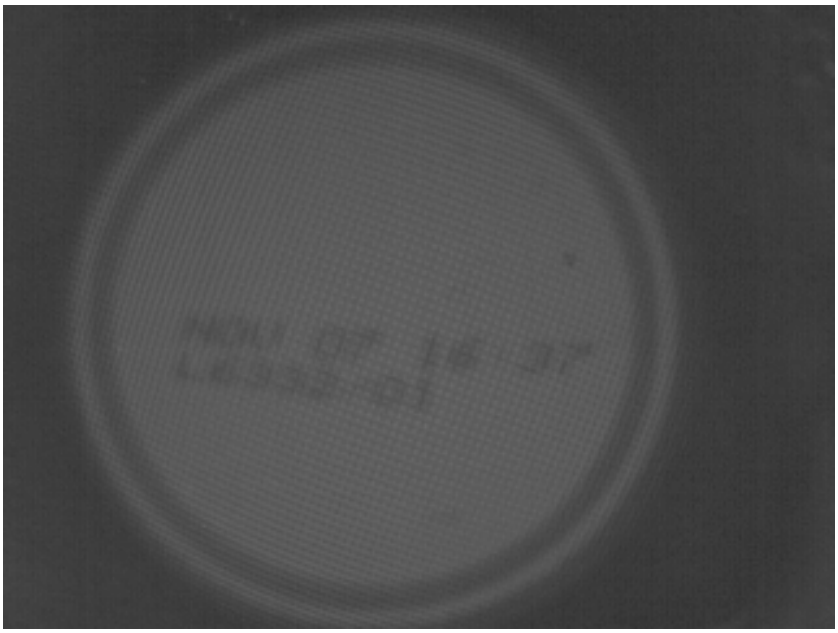


Figura 6.36: Un mal enfoque que permite ver la rejilla de la fuente de iluminación polarizada. Esto es desastroso porque hace que la magnitud del gradiente sea elevada(encuentra muchos bordes) cuando en realidad la imagen no está bien enfocada. (☼)

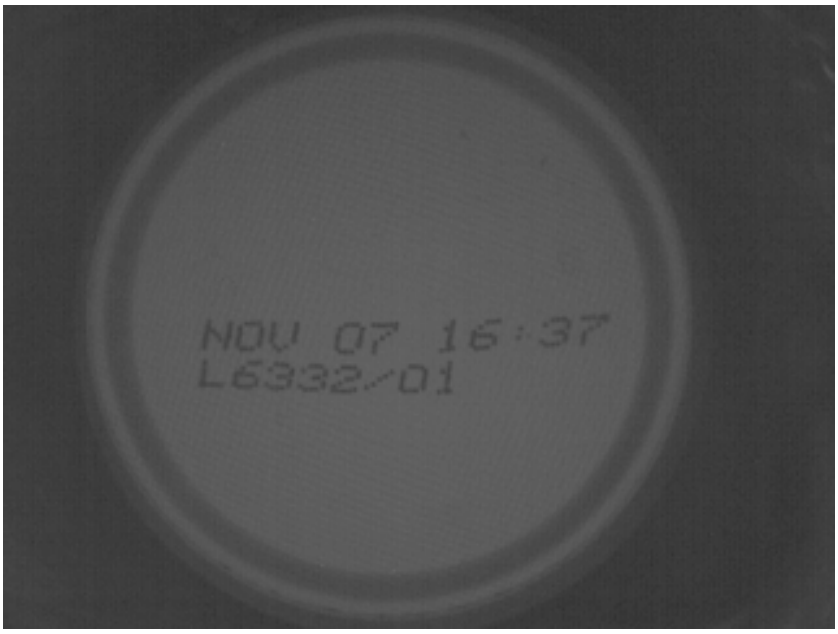


Figura 6.37: Una imagen bien enfocada. Podría servir como referencia. (☼)

examinan mejoras adicionales. Por ahora se ha simplificado la cuestión con un único umbral global aplicable sobre la superficie de la lata sobre el área de interés para procesamiento en tiempo real. Una solución imperfecta pero rápida

En el apartado 2.5.2 se examinaron múltiples métodos para el cálculo de umbralizado óptimo. MONICOD usa Otsu y Kittler_Illingworth pero se ha evitado su aplicación en tiempo de validación. En lugar de eso el umbral global es evaluado en tiempo de calibrado y, opcionalmente puede ser refinado con la inspección directa. Ver figura 6.38.

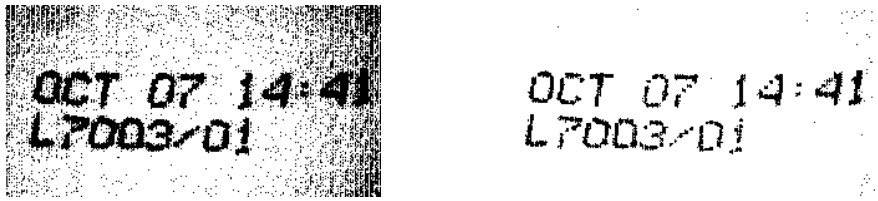


Figura 6.38: MONICOD y umbrales globales. Otsu a la derecha. Kittler_Illingworth a la izquierda.

Capítulo 7

MONICOD: Extracción de caracteres

"Donde las latas se echan a la carrera y nos lanzamos a cazarlas"

MONICOD extrae los caracteres por etapas en un esquema de dos procesos[1] concurrentes/paralelos. El primero trata los *frames*, el segundo trata las latas.

Un proceso de tratamiento de frames está dedicado a atender los frames que envía el hardware de adquisición. Por cada lata el sistema registra los *frames*, selecciona el mejor, marca el área del *frame* que contiene la superficie de la lata y lo transfiere al proceso concurrente de tratamiento de latas.

El proceso de tratamiento de latas preprocesa la superficie de la lata para mejorar el contraste y realzar los detalles, asocia la tinta contenida en fragmentos, se separan los fragmentos en líneas de texto, y se agrupan los fragmentos en lo que se espera serán caracteres.

7.1. La Validación: La detección de la lata

7.1.1. Código y lata

Quedó establecido en la sección 6.4 que el código de caducidad inscrito en la base de la lata es el objeto de interés por excelencia. Naturalmente si hay código de caducidad presente habrá una superficie de lata frente a la cámara[2](ver 6.4.3). Su recíproca no es cierta: Puede ocurrir que haya superficie de lata pero no código de caducidad. Eso sucederá, como se examinó previamente, en el caso de incurrir en la situación - *Código Ausente* -. Dicha situación es indeseable, DEBE ser detectada por el sistema y proporcionar una salida de Validación Negativa. Ver figura 7.1.

[1] Técnicamente son implementados como hilos de ejecución.

[2] Además, el código de caducidad es solidario a las superficies de lata que lo contienen, "moviéndose" con ellas.

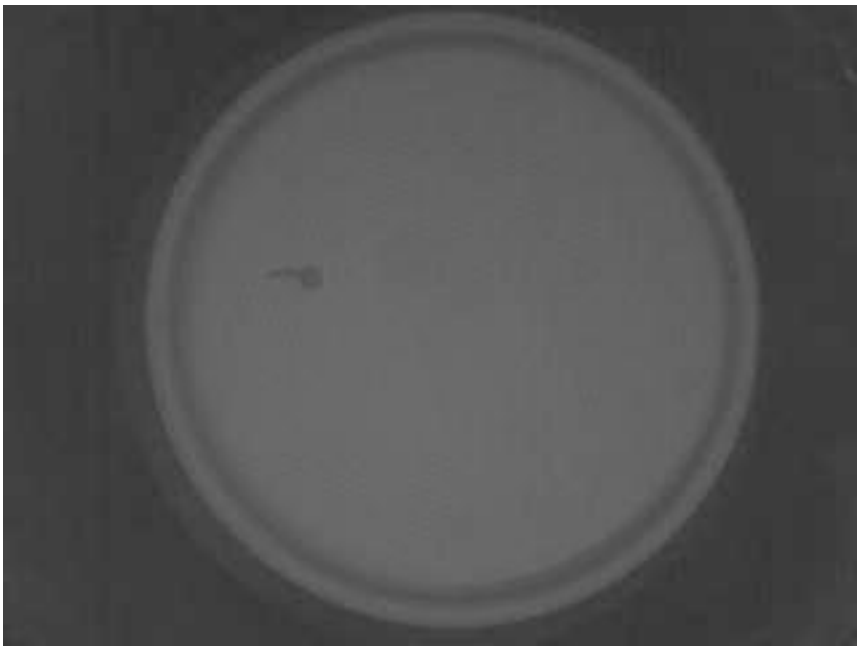


Figura 7.1: Ausencia de código. La lata existe y se dispone de una buena captura de ella pero no hay presencia del código de caducidad. (☼)

El procedimiento quedaría tal que así: MONICOD está obligado a fijarse si hay superficie de lata. De existir debe verificarse la presencia del código de caducidad. Esto es relativamente fácil porque toda superficie de lata tiene un código. Si no hay código, estamos frente al error de validación mencionado: *Código Ausente.* Por último MONICOD evaluaría si el código de caducidad es el esperado o no.

Por otra parte si NO hay superficie de lata no hay código de caducidad, y si no hay código de caducidad no es necesario lanzar los tests siguientes (por ejemplo el que verifica la presencia de código de caducidad). Parece razonable que el primer test a aplicar sobre un *frame* recién llegado sea: *¿Hay una lata tratable en el frame?*

Latas tratables y no tratables Las latas tratables corresponden a capturas de latas completas. Si hay una lata completa es factible lanzar el proceso de validación. Si la lata aparece sólo parcialmente delante de la cámara presumíblemente el código de caducidad puede haber sido capturado sólo a medias. En ese caso la validación no es aplicable (resultaría en una validación negativa no concluyente). Ver a este respecto las figuras 7.2 y 7.9.

Lo dicho anteriormente queda resumido en el diagrama 7.3.

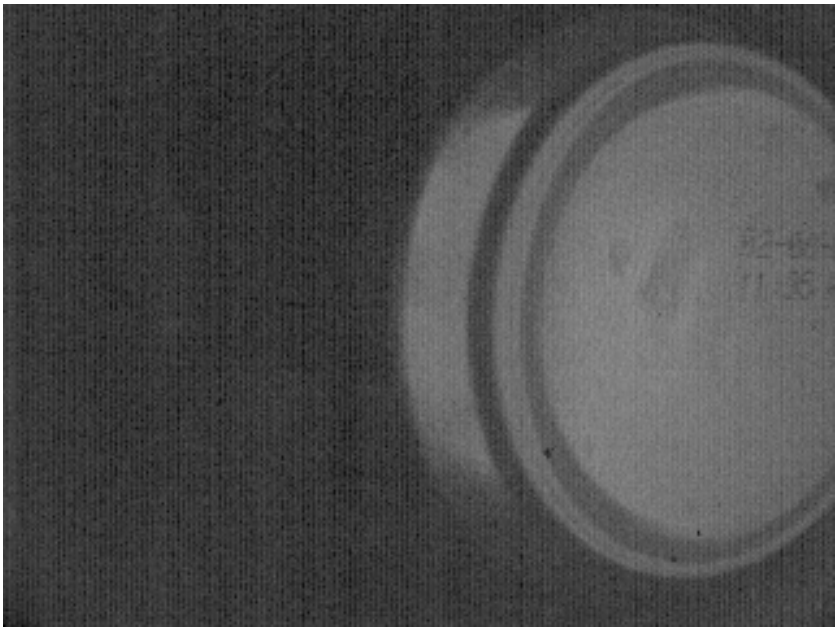


Figura 7.2: Lata no tratable (parte de la lata cae fuera de la imagen). (☼)

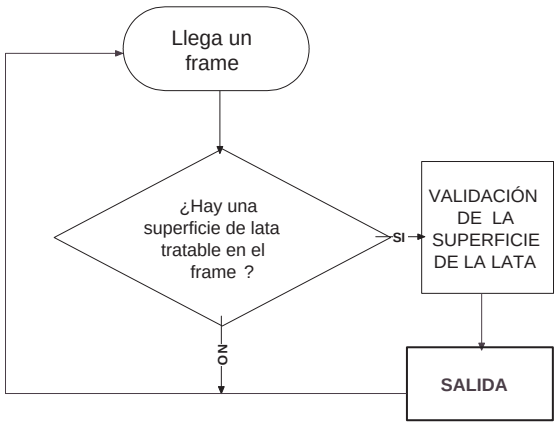


Figura 7.3: Esquema de detección

7.1.2. El estímulo "lata"

Sabemos que existe una cierta regularidad en el tránsito de las latas bajo la cámara. Por ejemplo, *frames* con una superficie de lata son precedidos y sucedidos por cierto número de *frames* sin superficie de lata.

Considerando esto, ¿sería posible conjeturar la presencia o no de lata basándonos en cuántos *frames* han pasado desde la última aparición de lata?

Se observa que el espacio entre latas en la cinta transportadora no es fijo. Así pues tampoco lo es el número de *frames* entre latas. Inevitablemente caeremos en dos posibles estados indeseables:

- Conjeturar que SI hay lata, cuando NO es así.
  - Lanzamos la Validación que irremediablemente falla.
  - La Validación debe distinguir ahora entre una situación de validación negativa y una situación de lata NO presente.
  - El computo de la validación es más costoso que la verificación sobre la presencia o no de lata (como veremos en los apartados siguientes)
- Conjeturar que NO hay lata, cuando SI la hay.
  - Perdemos información en forma de *frame* de superficie de lata apto para el procesamiento.
  - Corremos el riesgo inaceptable de perder una lata.

Se concluye de esto que conjeturar es inaceptable. En lugar de esto intentaremos responder eficientemente y sin conjeturas a la pregunta de si hay o no una lata tratable en el *frame*. MONICOD asumirá que esa pregunta se formula con la llegada de todo *frame* adquirido. La respuesta a la pregunta viene otorgada por el "estímulo" presencia de lata.

7.1.3. Detección de lata instantánea

Considerando lo establecido en el apartado 7.1.1, presentaremos la razón por la que el problema de la detección de lata es central para una adquisición de alta velocidad:

En el apartado 6.7 quedó explicado porque es clave optimizar el cuerpo de la llamada a la función de callback tanto como sea posible. Considerando la figura 7.3 planteamos cuatro esquemas de operación.

- Esquema sin concurrencia
- Esquema totalmente concurrente
- Esquema parcialmente concurrente
- Esquema parcialmente concurrente refinado

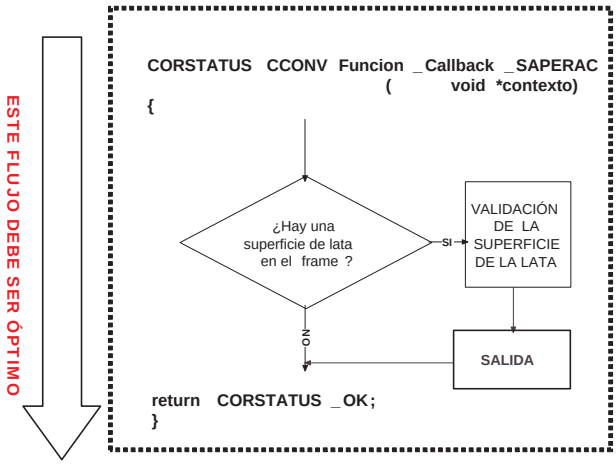


Figura 7.4: Flujo ineficiente. El flujo que debe optimizarse para no perder *frames* en la adquisición. Este esquema nos presenta un desafiante peor caso. Es necesario optimizar intensamente el flujo de la validación de la superficie de la lata para evitar la pérdida de *frames*.

Mejor Caso	Detección : ¿Es el frame tratable ?
Peor Caso	Detección : ¿Es el frame tratable ? + Validación de la superficie de lata

Cuadro 7.1: Mejor y peor caso para un esquema sin hilos

7.1.3.1. Esquema sin concurrencia

En el primer esquema trasplantamos el diagrama 7.3 al cuerpo de la función de callback. Ver figura 7.4. Proceder así obliga a optimizar un peor caso difícil si queremos evitar la pérdida de frames tal y como quedó explicado en 6.7. Ver cuadro 7.1

7.1.3.2. Esquema totalmente concurrente

En el segundo esquema hacemos uso radical de la concurrencia introducida en el apartado 4.2.1. El código del cuerpo de la función de callback se limita a lanzar un hilo de ejecución, y acto seguido se abandona la función de callback. Ver figura 7.5 y cuadro 7.2.

En el hilo de ejecución hallamos el esquema propuesto en la figura 7.3.

Lanzar un hilo concurrente es una operación de muy bajo costo comparativo. Aparentemente, a partir del cuadro cualitativo de costos 7.1, parece una buena solución. Sin embargo lanzar un segundo hilo antes de que el primero haya terminado (cosa que pasaría en la segunda llamada a la función de callback)

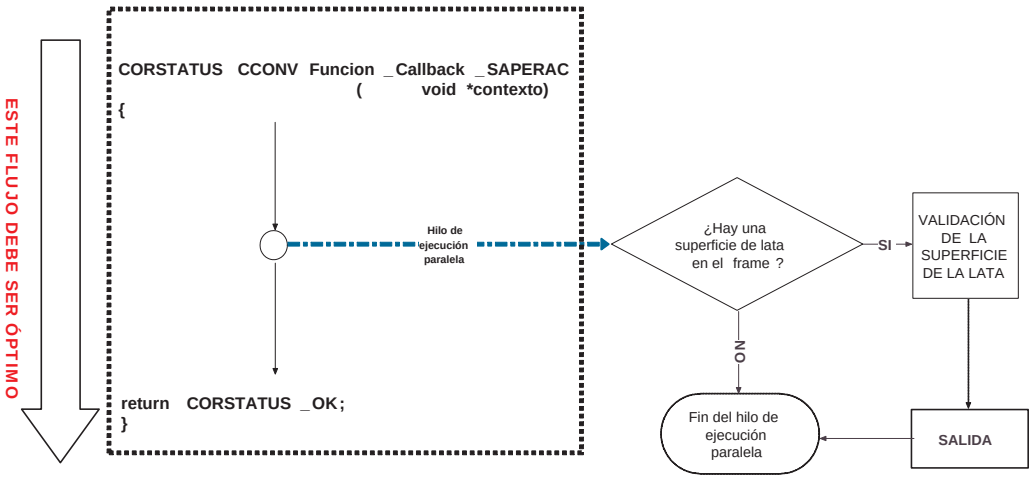


Figura 7.5: El flujo que debe optimizarse para no perder *frames* en la adquisición.

Mejor Caso	Lanzar hilo concurrente
Peor Caso	Lanzar hilo concurrente

Cuadro 7.2: Mejor y peor caso para un esquema totalmente concurrente.

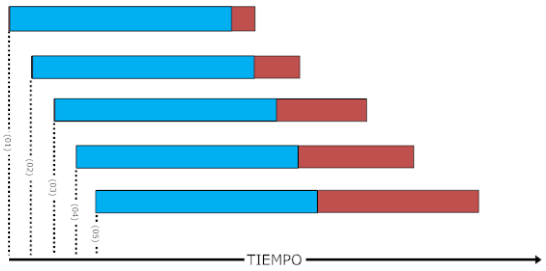


Figura 7.6: Lanzamiento de hilos en intervalos regulares. Azul para el tiempo de ejecución del hilo en un escenario sin concurrencia. En rojo los "overheads" derivados de la exigencias más altas de concurrencia.

Mejor Caso	Detección : ¿Es el frame tratable ?
Peor Caso	Detección : ¿Es el frame tratable ?

Cuadro 7.3: Mejor y peor caso para un esquema con hilo

supondrá que los ciclos del microprocesador[3] deberán repartirse con una tarea concurrente. El conjunto será globalmente más lento. Un tercer hilo agravará esta cuestión. Aunque el primer hilo termine y no ocupe ya ciclos del microprocesador el computador será progresivamente más lento. Ver a este respecto la figura 7.6.

Este proceso no se produciría si lográramos optimizar la ejecución del hilo de tal manera que pueda garantizarse que el hilo actual ya ha terminado[4] antes de lanzar el siguiente hilo.

7.1.3.3. Esquema parcialmente concurrente

En el tercer esquema hacemos un uso mixto. Evitamos lanzar tantos hilos como llamadas de callback. Sólo nos limitamos a lanzarlos cuando haya que comprobarse el código de caducidad y eso solo ocurre cuando hay una superficie de lata presente. Este esquema aligera el costo del cuerpo de la función de callback respecto al esquema sin concurrencia. Y alivia la sobrecarga por demasiadas llamadas concurrentes. Además la ejecución del hilo ya no involucra la comprobación de presencia de lata. Ver figura 7.7 y cuadro 7.3.

[3] El número de nucleos del computador es irrevelante aquí porque el número de hilos a lanzar es "virtualmente" infinito, y se lanzan regularmente.

[4] Si fuera posible garantizarlo podríamos aplicar el esquema sin concurrencia sin las complejidades añadidas de los hilos.

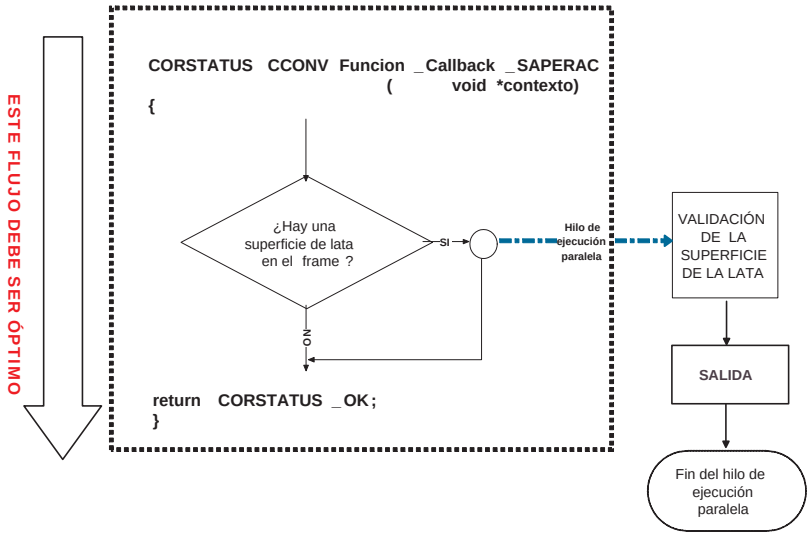


Figura 7.7: El flujo que debe optimizarse para no perder *frames* en la adquisición.

7.1.3.4. Esquema parcialmente concurrente refinado

El cuarto esquema es el que ha sido llevado a MONICOD. Se apoya en un hecho incontestable: Una lata validada NO tiene porque volver a ser validada una segunda vez[5]. En el esquema parcialmente concurrente bastaba la presencia de lata para lanzar un hilo concurrente. En este esquema refinado, en cambio, lanzaremos un hilo sólo si ya no se había lanzado un hilo para esa lata. Al disminuir el número de lanzamientos de hilos, la sobrecarga por exceso de llamadas concurrentes es amortiguada aún más. A cambio, es necesario complicar el procesamiento en el cuerpo de la función de callback, porque ahora no basta detectar la presencia de una superficie de lata. También es necesario saber si esa superficie ha sido procesada o no. Ver figura 7.8 y cuadro 7.4.

Una ventaja de esta aproximación es que ofrece una selección de la mejor captura de la validación. Un inconveniente es que la validación ocurre cuando la lata ya ha abandonado el campo de visión. Esto supone un retraso respecto a la adquisición real.

7.1.4. Histogramas elípticos

7.1.4.1. Eventos de el campo de entrada

Las áreas de interés de contorno (ver sección 6.5) nos permiten:

[5]Supuesto que la validación (negativa o positiva) ha sido ejecutada con éxito, y no necesita apoyarse en validaciones independientes.

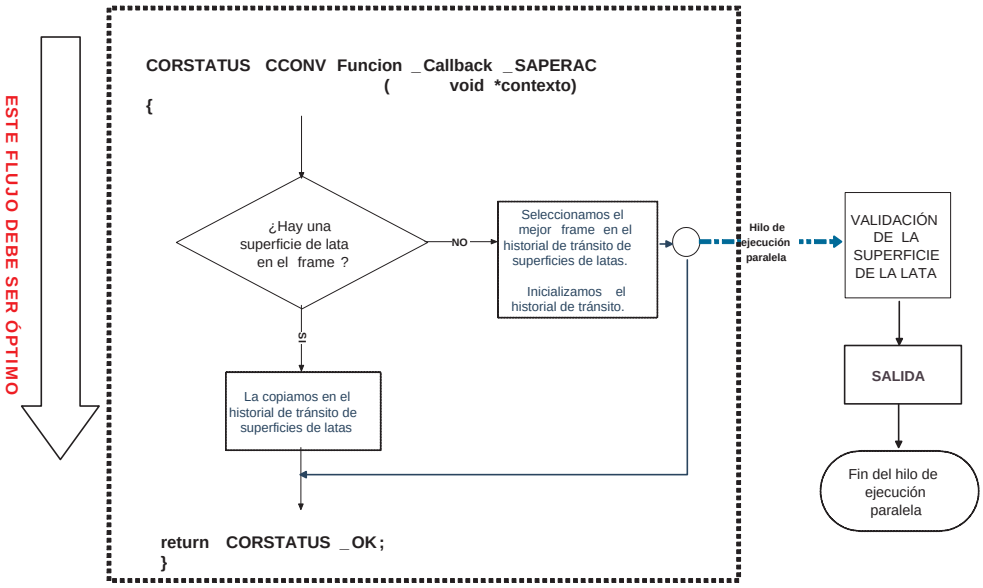


Figura 7.8: El flujo que debe optimizarse para no perder *frames* en la adquisición.

Mejor Caso	Detección : ¿Es el frame tratable ? + Almacenamiento en el acumulador de frames
Peor Caso	Detección : ¿Es el frame tratable ? + Selección del mejor frame en el acumulador de frames + Lanzar hilo concurrente + Inicializar acumulador

Cuadro 7.4: Mejor y peor caso para un esquema con hilo

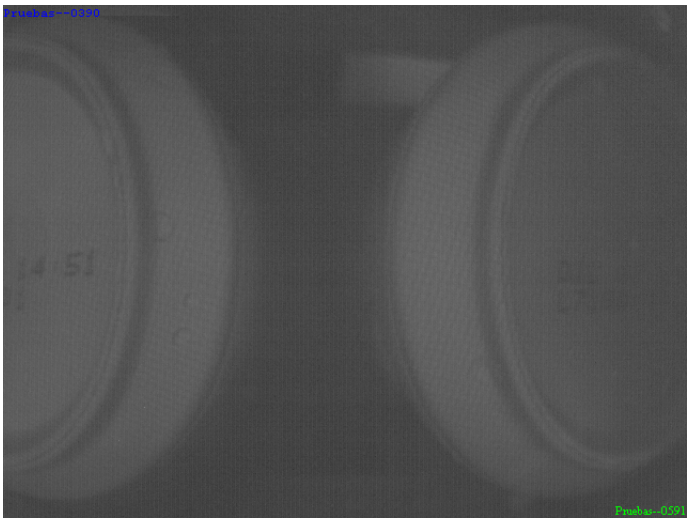


Figura 7.9: Dos latas en un *frame* pero ninguna elipse se activa. (☼)

1. Saber si existe superficie de lata o no en la imagen actual. De haber varias latas simultáneamente, se escogerá la lata "dominante"/mejor centrada en la imagen.

2. Saber la ubicación concreta de la lata dominante en el eje de avance de las latas.

Esta información es bastante primaria pero fácilmente podemos integrarla en un tipo de conocimiento de orden superior que nos permitirá monitorear el paso de latas por el campo de visión de la cámara.

1. Es posible saber cual es *frame* en el que se produce el evento *Entra una lata en el campo de entrada.*

2. Es posible saber cual es *frame* en el que se produce el evento *Una lata abandona el campo de entrada.* Ese evento sólo se producirá si previamente ha habido un suceso del tipo *Entra una lata en el campo de entrada.*

3. Ante la eventualidad de que se produjera el evento *Entra una lata en el campo de entrada* para una nueva lata antes de que se haya producido el evento *Una lata abandona el campo de entrada* se asumirá automáticamente un evento *Una lata abandona el campo de entrada* que resuelva la entrada de la lata previa. Ver figura 7.9. De cualquier modo esta eventualidad es minimizada por la disposición física entre cámara y latas que impide que una lata invada el campo de entrada sin antes haberlo abandonado la lata previa.

4. La ocurrencia de los sucesos mencionados en los apartados 2-3 nos permite contar cuantas latas se han deslizado por el campo de entrada del sistema.

Un par *Entra una lata en el campo de entrada*-*Una lata abandona el campo de entrada* supone un incremento en una unidad del contador de latas.

5. Es posible almacenar temporalmente[6] en un contenedor los *frames* entre los eventos *Entra una lata en el campo de entrada* y *Una lata abandona el campo de entrada*. Ver a este respecto el apartado 7.1.4.3. Con eso se abre la posibilidad de seleccionar entre los *frames* adquiridos para esa lata el mejor *frame* para validación.

7.1.4.2. Procedimiento de detección de lata

Detectar la presencia de lata se reducirá a ejecutar una consulta sobre las áreas de interés de contornos de lata. Ver figura 7.10. Una consulta sobre un área de interés consiste en hacer un recuento de valores que estén dentro de un rango de valores $[U_A, U_B]$ excedan un umbral para los puntos que componen cada una de las áreas de interés de contorno.

El porcentaje de activación del área de contorno vendrá dado por:

$$\frac{N_U}{N_T} \times 100 \tag{7.1}$$

Siendo N_U el número de valores dentro del rango de umbral $[U_A, U_B]$ y N_T el número total de valores del área de interés de contorno. Ver el algoritmo en 7.1.

Este porcentaje es computado para todas las áreas de interés de contorno existentes. Sólo nos interesarán aquellos porcentajes de activación del área de contorno superior al proporcionado por la expresión 7.1.

Se decide que hay una lata en la imagen cuando existe al menos un área de interés de contorno que exceda el porcentaje de activación descrito en 7.1.

El algoritmo es esbozado en 7.2.

7.1.4.3. El historial de tránsito

El historial de tránsito hace referencia a un contenedor de *frames* adquiridos que capturan el paso de una lata a través del campo de visión de la cámara. Ver figura 7.11.

Con *framerates* de adquisición tan altos tenderemos a tener más de un *frame* que contenga cada lata. En un momento dado podemos tener dos, tres, cuatro o incluso cinco *frames* de la misma lata dependiendo de la velocidad a la que esa lata se desplaza en la cinta transportadora.

El historial de tránsito es un área de memoria que almacena sólo y exclusivamente el último tránsito. Se sacrifica memoria para ofrecer la oportunidad de elegir entre esos *frames* el más adecuado para someterlo al proceso de validación.

[6] En memoria principal, no en disco duro.

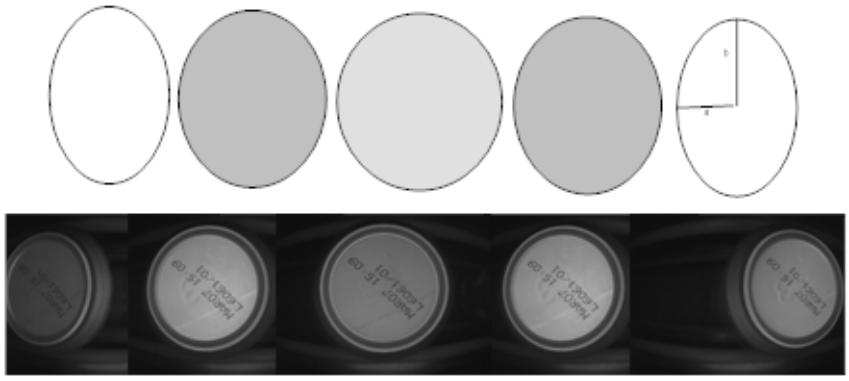


Figura 7.10: Elipses y superficies de lata.

Algoritmo 7.1 Evaluar rangos de intensidad en una vecindad

Precondición: *imagen*[][]: La imagen que estamos procesando.
Precondición: *contorno*[]:Coordenadas de un recinto a evaluar (área de interés)
Precondición: *nPuntosContorno*:Número de puntos del recinto a evaluar (área de interés)
Precondición: *luminanciaminima*,*luminanciamaxima*: Rango de luminancia en la que estamos interesados
Postcondición: Retorna el porcentaje de píxeles cuya luminancia se encuentra en el rango de luminancias sobre el que estamos interesados.

```
nluminanciasbajomaximo ← 0
nluminanciasbajominimo ← 0
para todo [x,y] tal que [x,y] esté en el Contorno[] hacer
    luminancia ← Leer-luminancia(imagen[x][y])
    si luminancia es-menor-que luminanciamaxima entonces
        nluminanciasbajomaximo ← nluminanciasbajomaximo + 1
        si luminancia es-menor-que luminanciaminima entonces
            nluminanciasbajominimo ← nluminanciasbajominimo + 1
        fin si
    fin si
fin para
porcentaje ← (nluminanciasbajomaximo − nluminanciasbajominimo) ∗
100/nPuntosContorno
retornar porcentaje
```

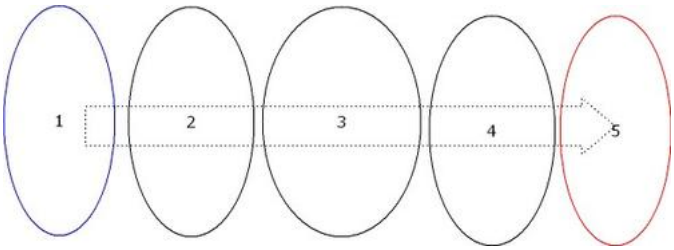


Figura 7.11: Secuencia de activación de elipses que implican el tránsito de lata sobre el campo de entrada de la cámara.

¿Cual es el *frame* más adecuado? En general el *frame* más adecuado es aquel que exhiba la superficie de la lata más cercana al centro del campo de entrada. En el centro hay mejor respuesta a la iluminación y además se evita la distorsión óptica de los extremos de la imagen.

7.1.4.4. El seguimiento (tracking) de la lata

Para llevar a cabo el seguimiento de la lata es necesario definir el sentido de avance de las latas en el campo de entrada del sistema. El sentido de avance es fijo durante la ejecución de MONICOD. Los contornos de lata se distribuyen sobre el eje de desplazamiento de las latas. Dado este sentido de desplazamiento de las latas podemos asignar a las áreas "contornos de lata" las denominaciones "anterior" y "posterior" dependiendo del orden de activación esperable para un escenario donde una lata recorre el campo de entrada siguiendo el sentido de avance esperable.

En las figuras 7.12, 7.13, 7.14, 7.15, 7.16, y 7.17 se recogen los estados posibles en el seguimiento.

Para cada *frame* se aplicará un chequeo de activación para cada contorno de lata. Tres escenarios son posibles:

1. No hay activación para ningún contorno. No hay latas en el campo de entrada.
2. Se activa un contorno. Esta es una clásica situación de presencia de lata en el campo de entrada.
3. Se activa más de un contorno. Es preciso decidir ante varias activaciones cual es la dominante. Sólo podremos quedarnos con una para el seguimiento.

Considerando esto, el procedimiento será tal que así :

No hay activación de contornos de lata. Inicialmente, en una ejecución normal, no existen latas en el campo de entrada. Las áreas de detección permanecen inactivas (figura 7.12). Ver figura 7.18, subimagen (a).

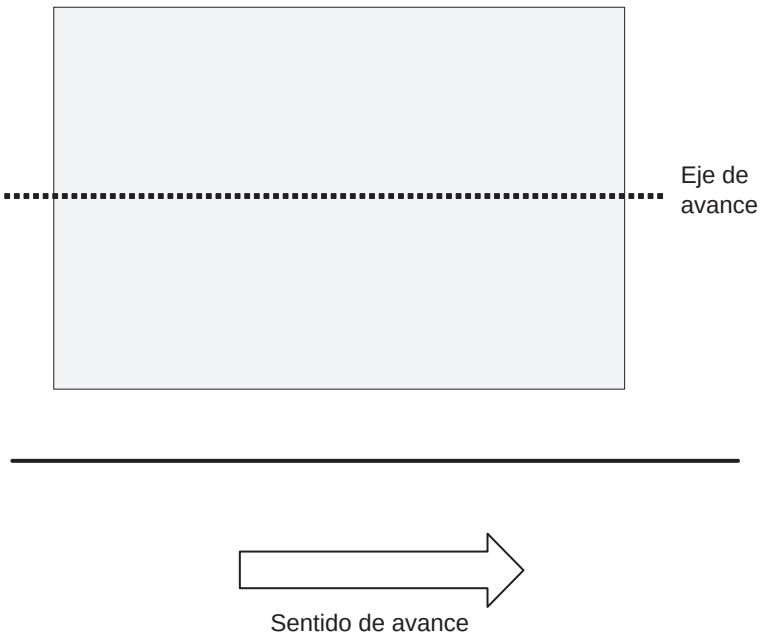


Figura 7.12: No hay presencia de superficie de lata. El historial de tránsito esta vacío.

Una contorno de lata es localizado. Ha ocurrido un evento de *Entra una lata en el campo de entrada del sistema* (ver figura 7.13). Ver figura 7.18, subimagen (b).

Una contorno de lata es localizado en una posición posterior a la activación anterior. La lata avanza por el campo de entrada (ver figura 7.15). En cada *frame* es relocalizada. Sus posiciones pueden no ser consecutivas pero son coherentes con el sentido de la marcha de la cinta transportadora.

De nuevo, no hay activación de contornos de lata. Ha ocurrido el evento *Una lata abandona el campo de entrada del sistema*. Incrementamos el contador de latas. y pasamos a un estado idéntico al primero (figura 7.12). Ver figura 7.18, subimagen (c).

Otra posibilidad es que haya habido una activación de un área de contorno de lata anterior al área de contorno de lata previamente activado (ver figura 7.17). Esta situación debe ser minimizada con la configuración del soporte físico. Se asume que han ocurrido simultáneamente los eventos *Una lata abandona el campo de entrada del sistema* y *Entra una lata en el campo de entrada del sistema*.

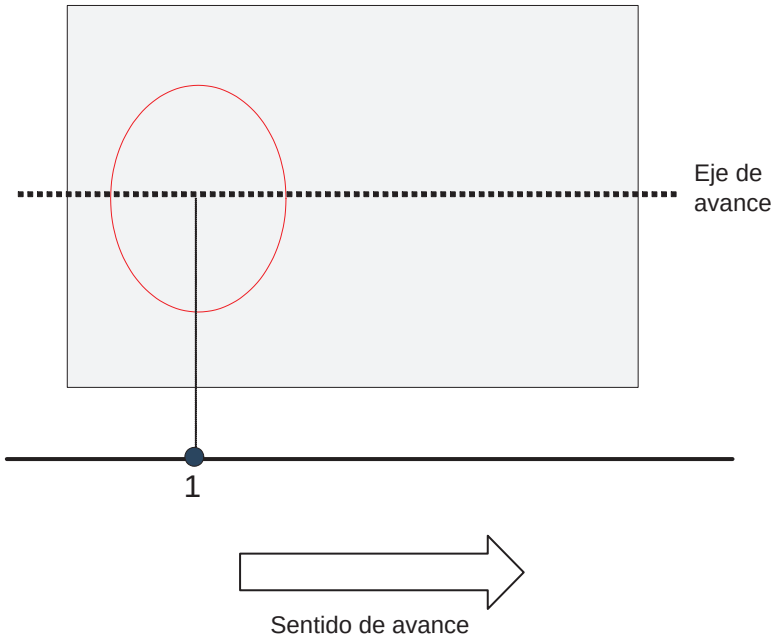


Figura 7.13: Evento *Entra una lata en el campo de entrada*. Se detecta presencia de superficie de lata. Se anota su posición en el eje de avance. La imagen se almacena en el historial de tránsito.

Algoritmo 7.2 Calcular Mejor Elipse

Precondición: *imagen*[]:La imagen adquirida. *elipses*[]:Una colección de areas de interés con forma eliptica

Postcondición: *elipseganadora*:Un índice a la elipse ganadora. -1 si NO hay ganadora.

para $i = 0$ **a** *numerototaldeelises*-1 **hacer**
  $niveldeactivaciondeelipse \leftarrow Calcular-Activacion-Elipse-Basada-En-Contorno(i)$
  si *niveldeactivaciondeelipse* es mayor que *umbraldeactivaciondeelipse* **entonces**
    si *niveldeactivaciondeelipse* es mayor que *maximoniveldeactivaciondeelipse* **entonces**
      $maximoniveldeactivaciondeelipse \leftarrow niveldeactivaciondeelipse$
      $mejorindicedeactivacion \leftarrow i$
    fin si
  fin si
fin para

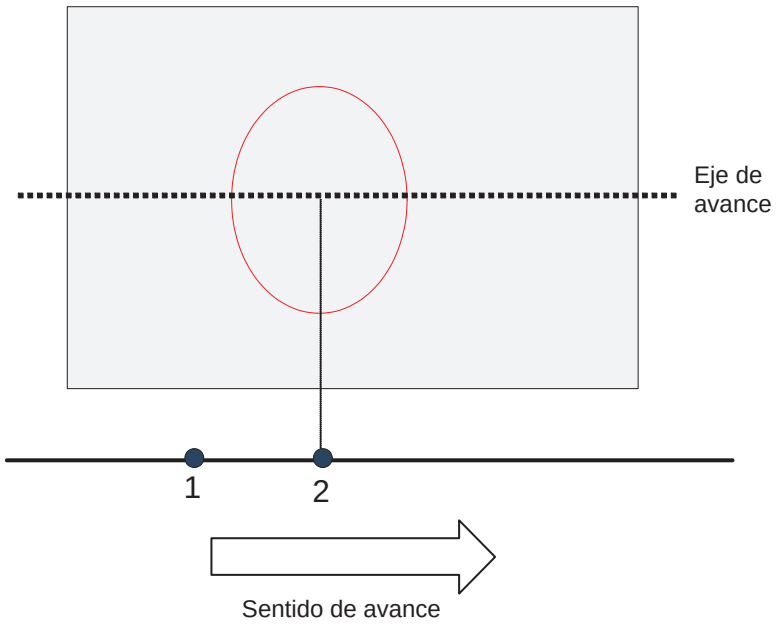


Figura 7.14: Se detecta presencia de superficie de lata. Se anota una nueva posición en el eje de avance.La imagen se almacena en el historial de tránsito.

7.1.4.5. El recuento de latas

El recuento de la latas indica el número de latas diferentes que han pasado delante de la cámara. Se espera que coincida con el número de latas efectivamente validadas por el sistema. De otro modo no todas las latas detectadas por el sistema habrán sido validadas lo que es indeseable.

El contador de latas es reiniciado cada vez que se inicia un procedimiento de validación. En el diagrama 7.19 se expone el flujo de control común que incrementa en una unidad el contador de latas. No es la única posibilidad. Como se ha descrito puede darse la situación de dos eventos *Entra una lata en el campo de entrada del sistema* sin que entre ellos ocurra un evento *Una lata abandona el campo de entrada del sistema.* En ese caso, como se ha dicho, se desliza un evento *Una lata abandona el campo de entrada del sistema* (que en realidad no hemos detectado) y el contador de latas se incrementa.

7.1.4.6. Optimización

Como se ha descrito, MONICOD utiliza el contorno de la elipse para detectarla, pero en realidad puede utilizarse cualquier otra forma. Por ejemplo, el área de interés de los ejes mayores o menores, o los "trópicos" de la elipse,... En fin, es posible experimentar con otras áreas de detección con el fin de detectar latas. Minimizar los puntos a consultar para verificar que la elipse esté activa,

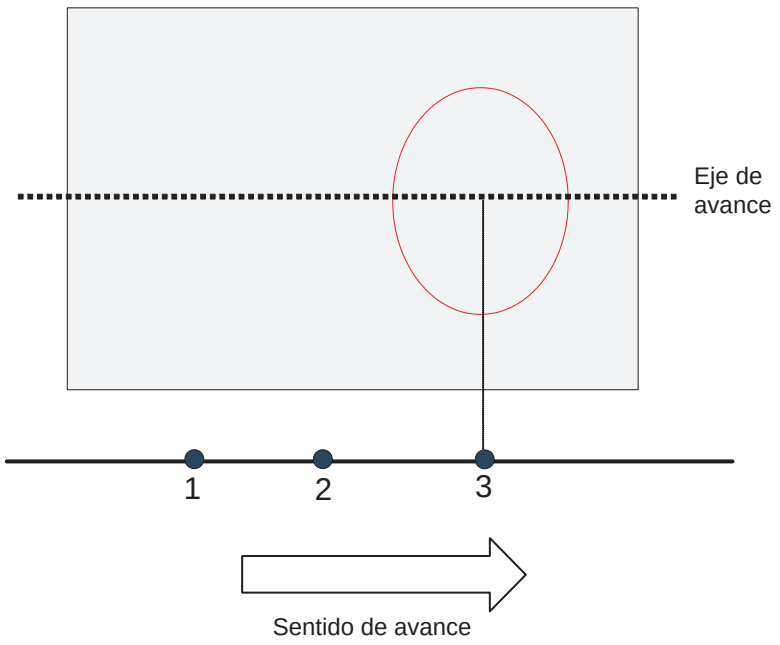


Figura 7.15: Se detecta presencia de superficie de lata. Se anota su posición en el eje de avance. La imagen se almacena en el historial de tránsito.

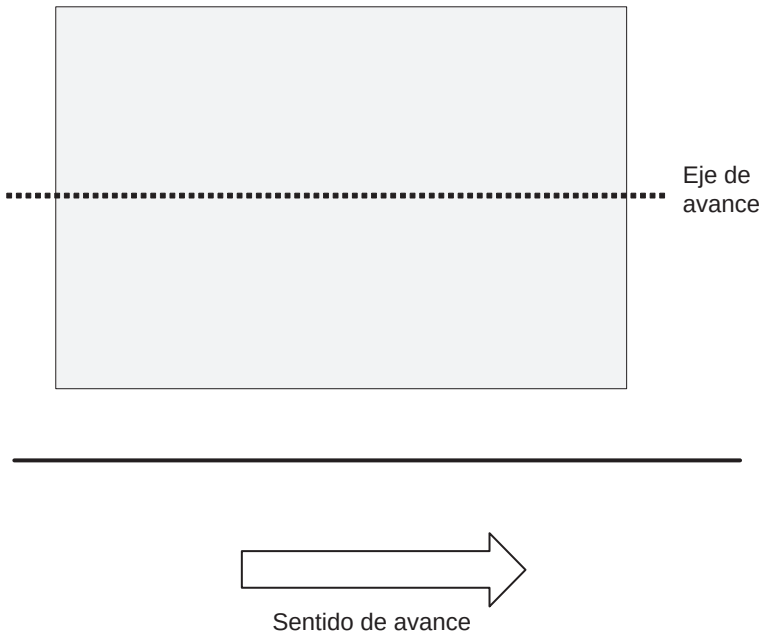


Figura 7.16: Evento *Una lata abandona el campo de entrada*. No hay presencia de superficie de lata. Necesariamente la superficie de lata ya ha pasado. Se elije la mejor superficie de las latas acumuladas y se lanza el hilo de validación con ella. Se reinicia el historial de tránsito. Se incrementa el contador de latas.

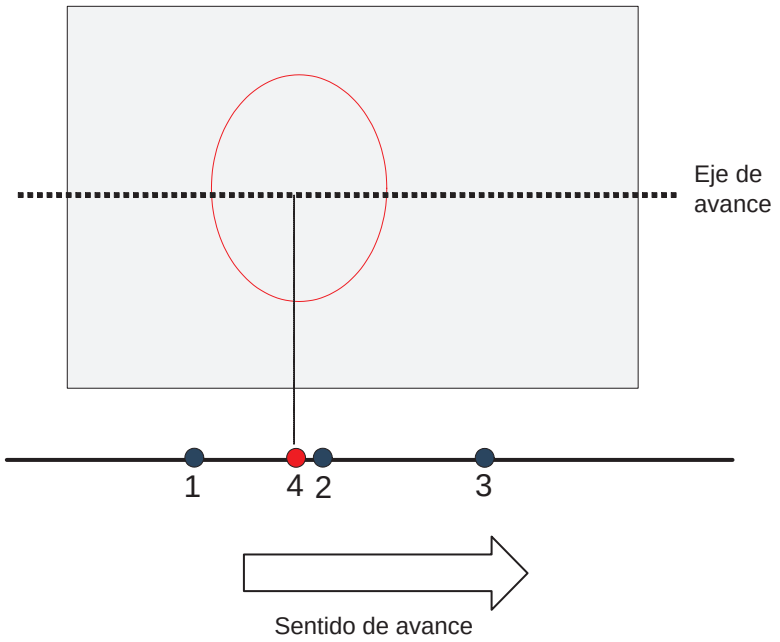


Figura 7.17: Evento *Una lata abandona el campo de entrada* también puede producirse aunque hayan latas en la imagen. Supóngase que se detecta presencia de superficie de lata. Se anota su posición en el eje de avance. Se detecta de nuevo superficie de lata. Se anota su posición en el eje de avance y ocurre que esta última posición es anterior, en el eje de avance, a la penúltima posición registrada. En modo de operación normal las superficies de lata no retroceden de manera que necesariamente es una nueva lata. Se elije la mejor superficie de las latas acumuladas y se lanza el hilo de validación con ella. Se reinicia el historial de tránsito. Se incrementa el contador de latas. Se almacena el *frame* actual.

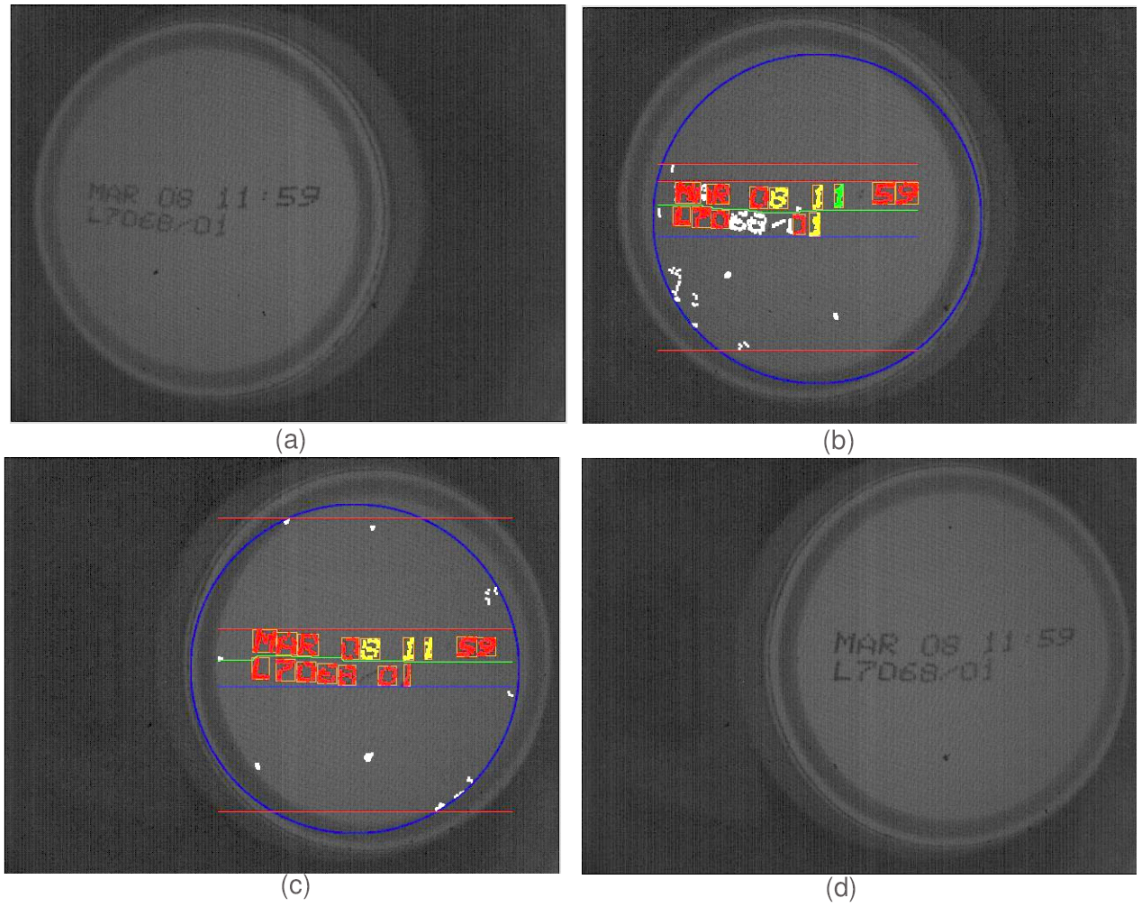


Figura 7.18: Movimiento de lata: (a) Una lata ha entrado en el campo de visión de la cámara pera no ha sido detectada aún. (b) Evento *Entra una lata en el campo de entrada del sistema.* (c) Evento *Una lata abandona el campo de entrada del sistema.* (d) Una lata abandona el campo de visión de la cámara después de haber sido procesada. (☼)

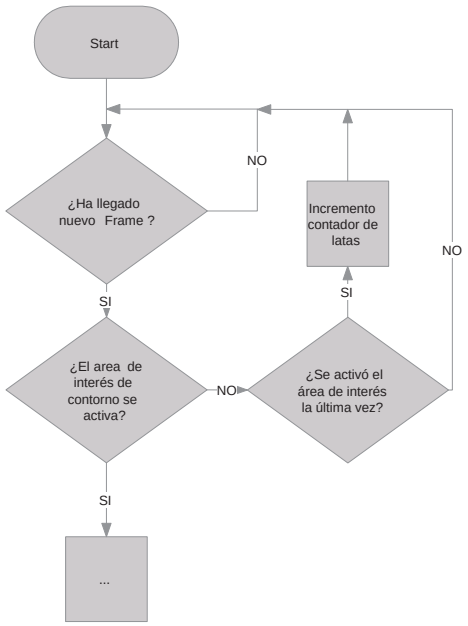


Figura 7.19: El procedimiento de recuento de latas cuando se produce un par *Entra una lata en el campo de entrada del sistema* y *Una lata abandona el campo de entrada del sistema.*

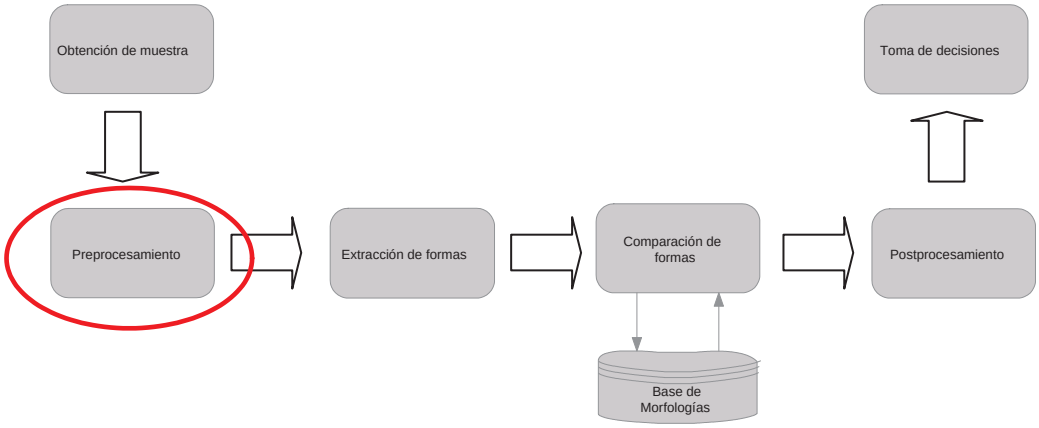


Figura 7.20: El preproceso.

y maximizar la calidad de la detección de la elipse dominante son los criterios a tener en cuenta.

7.2. La Validación: El preproceso

La imagen adquirida es denominada 'imagen original' y es discutida en 4. Sobre esta imagen original MONICOD se aplica un tratamiento global (el preprocesamiento, ver figura 7.20) que mejora sensiblemente sus características generales frente al procesamiento posterior.

7.2.1. Realce MONICOD: El mínimo local

En el apartado donde se describe la convolución (A.3.6) se habla sobre el costo prohibitivo de un procedimiento de convolución. El realce es un procedimiento de convolución pero es factible aplicarlo a un sistema de alta velocidad como MONICOD porque:

- Las operaciones involucradas se limitan a comparaciones entre enteros por lo que su costo es muy reducido.
- No se aplica a toda la imagen sino al área de interés de la base de lata.

Por otro lado, las ventajas cualitativas que comporta son considerables. La relación resultado obtenido/costo computacional ha decidido su integración en MONICOD.

Algoritmo 7.3 Mínimo de una vecindad

Precondición: *posicion*:Una posición valida de la imagen con la que construir un vecindario
Postcondición: Retorna el valor mínimo del vecindario
minimo ← `Leer-Color(`*posicion*`)`
vecinos`[]` ← `Calcular-Vecindario(`*posicion*`)`
numerodevecinos ← `Obtener-Numero-De-Vecinos(`*vecinos*`[])`
para $i = 0$ **a** *numerodevecinos*-1 **hacer**
    color ← `Leer-Color(`*vecinos*`[i])`
    si *minimo* es-mayor-que *color* **entonces**
        minimo ← *color*
    fin si
fin para
retornar *minimo*

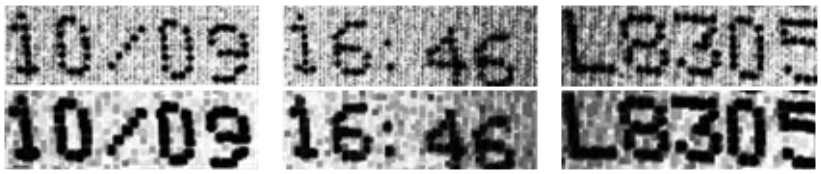


Figura 7.21: Sin realce arriba. Con realce abajo

7.2.1.1. Mínimo local

Dada una imagen A construimos una imagen B tal que cada posición *(i,j)* en B tiene el valor de gris correspondiente al mínimo valor de gris de la vecindad de *(i,j)* -incluyendo al propio *(i,j)*- en A.

$$b_{i,j} = min \begin{pmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{pmatrix}$$

Ver algoritmo 7.3.

7.2.1.2. Resultados

El mínimo local "engrosa" el código suprimiendo gran parte de los espacios *intracaracter* (ver definición en A) indeseables. Además acentúa el contraste con el fondo. Ver figura 7.21.

El principal inconveniente de la técnica es que también puede suprimir espacios *intercaracter* (ver definición en A), unificando caracteres distintos en uno.

Figura 7.22: *Frame* sin ecualizar (izquierda) y ecualizado (derecha).

Algoritmo 7.4 Histograma de una imagen

Precondición: $Imagen[][]$: Imagen sobre la que se construye el histograma
Precondición: $Alto, Ancho$: Dimensiones de la imagen
Postcondición: H[] es el vector histograma de la imagen

```
Inicializamos el vector H a 0
para i = 0 a Alto-1 hacer
    para j = 0 a Ancho-1 hacer
        color ← Imagen[i][j]
        H[color] ← H[color] + 1
    fin para
fin para
```

7.2.2. El ecualizado del histograma

Cómo se ha explicado anteriormente el tiempo de exposición empleado en la adquisición de imágenes (*frames*) es muy breve resultando en un contraste de la imagen deficiente. Esto es, el rango de niveles de grises presentes está demasiado comprimido. El ecualizado es una técnica simple y rápida que permite aprovechar mejor el espectro[7]. Las bases de este procedimiento se han descrito en 2.4.2.3. Ver algoritmos 7.4, 7.5 y 7.6.

7.2.2.1. Resultados

Se aprecia que el ecualizado de imagen contrarresta notablemente el problema de una imagen excesivamente oscura (ver figura 7.22). A cambio el costo computacional es sostenible. Adicionalmente se obtiene información secundaria que será aprovechada por otros procesos MONICOD. Para acelerar aún más el procedimiento empleamos tablas LUT (ver apartado A.3.7).

[7] Sin embargo está contraindicada para imágenes con buen contraste.

Algoritmo 7.5 Construir tabla de ocurrencias.

Precondición: *imagen*[]:La imagen. *areadeinteres*[]:Un contenedor que proporciona las coordenadas de los puntos que corresponden al área de interes.
Postcondición: Retorna una tabla con el número de ocurrencias de colores para el area de interes en la imagen.

```
Inicializar-a-cero(ocurrencias[])
para i = 0 a numerodeposicionesdelareadeinteres-1 hacer
    color ← Leer-Color(imagen[areadeinteres[i]])
    ocurrencias[color] ← ocurrencias[color] + 1
fin para
retornar ocurrencias[]
```

Algoritmo 7.6 Construir tabla de ecualizado.

Precondición: *ocurrencias*[]:La tabla de ocurrencias.
Postcondición: Retorna una tabla de asignacion de colores para ecualizar la imagen.

```
probabilidades[0] ←ocurrencias[0]/numerototaldeocurrencias
probabilidadesacumuladas[0] ← probabilidadesacumuladas[i]
tabladeecualizado[i] ← probabilidadesacumuladas[0]*numerototaldeocurrencias
para i = 1 a numerodecoloresdelaimagen-1 hacer
    probabilidades[i] ←ocurrencias[i]/numerototaldeocurrencias
    probabilidadesacumuladas[i] ← probabilidadesacumuladas[i] +
    probabilidadesacumuladas[i − 1]
    tabladeecualizado[i] ← probabilidadesacumuladas[i]*numerototaldeocurrencias
fin para
retornar tabladeecualizado[]
```

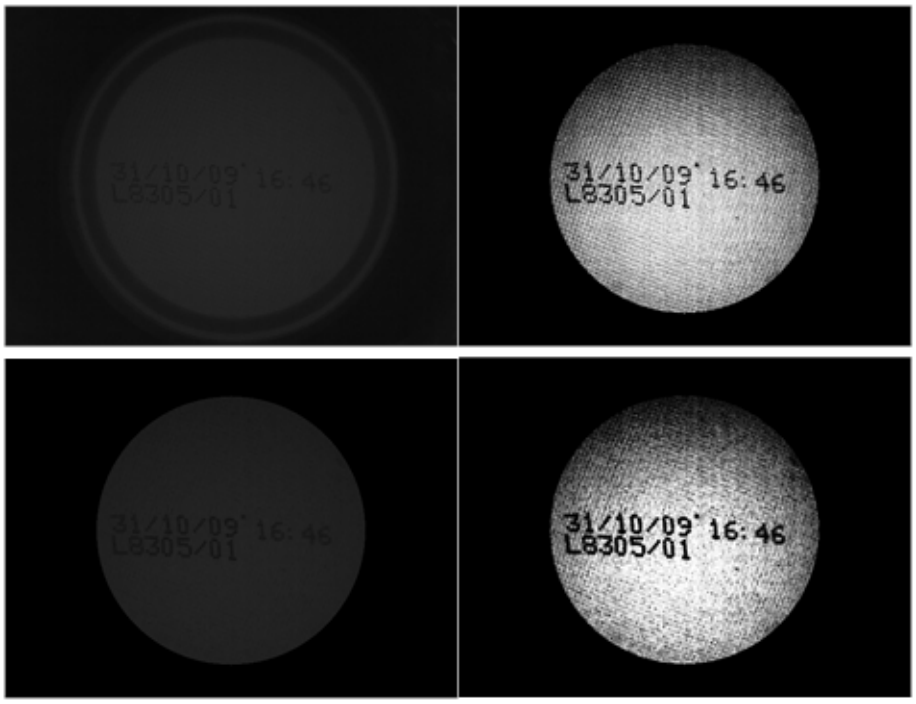


Figura 7.23: Arriba-izquierda imagen original. Arriba-derecha ecualizado de la imagen original.Abajo izquierda imagen original realzada. Abajo derecha imagen original realzada y posteriormente ecualizada.

7.2.3. Combinando las operaciones de realce y ecualizado

Las operaciones de realce y ecualizado son combinadas para obtener mejores resultados. Ver 7.23.

MONICOD lleva a cabo un realzado, y a continuación, un realzado. Finalmente opera sobre el resultado obtenido:

1. Imagen original.
2. La imagen original es realzada.
3. La imagen realzada es ecualizada.

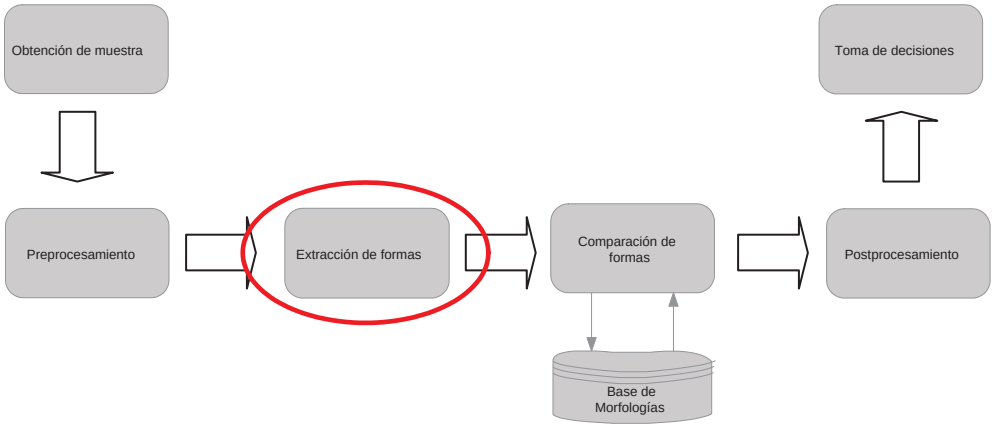


Figura 7.24: La extracción de las formas.

7.3. La Validación: Segmentando caracteres

7.3.1. Generalidades

7.3.1.1. ¿Por qué segmentar caracteres?

La etapa de segmentación de caracteres (ver figura 7.24) trata de rescatar los píxeles con tinta de la sección de imagen delimitada por el área de interés.

Tratamos con un código basado en caracteres (Ver sección 3.8) donde no podemos prescindir de la legibilidad de ninguno[8] Parece razonable que la validación de caracteres (como se detallará en el capítulo 8) se lleve a cabo a nivel de carácter. Para proceder así es necesario localizarlos y aislarlos.

7.3.1.2. Validar segmentando

Segmentar los caracteres (extraer las formas de carácter de la imagen), además de alimentar etapas posteriores, opera también como un test sobre la lata:

- Si el procedimiento no segmenta ningún carácter viable entonces el código está ausente o está severamente corrompido. El código es ilegible.
- Si el procedimiento no segmenta suficientes caracteres para formar un código tal como el que estamos validando entonces hay caracteres ausentes. Si uno o más de los caracteres ausentes está clasificado como 'imprescindible' el código es ilegible.

[8] En el reconocimiento de caracteres estándar es posible acudir a contextos y diccionarios para tratar de identificar la palabra aunque no se hayan podido reconocer todos los caracteres.

7.3.1.3. Construyendo caracteres: Píxeles, fragmentos, caracteres

En A.3.1 se describe el píxel como la entidad autónoma con contenido mínimo dentro de la imagen. La aportación de información de un píxel aislado es casi insignificante. Sin embargo píxeles asociados conforman fragmentos y caracteres.

En 6.3.1 se presenta la relación entre fragmento de carácter y carácter así como la definición de fragmento de carácter con la que operamos. Un carácter esta constituido por uno o más fragmentos. Esta fragmentación puede ser natural, propia del carácter (véase el caso de la "i" o los dos puntos ":") (ver figura 7.26), o accidental (ver figura 7.27), derivada de una impresión defectuosa o ruido en la adquisición. La situación de fragmentación artificial es común pero no menoscaba necesariamente la legibilidad de los caracteres para un lector humano[9].A pesar de todo los fragmentos están ahí y el sistema debe lidiar con ellos.

7.3.1.4. Segmentación en tres etapas

La segmentación en MONICOD se lleva a cabo en cuatro etapas:

1. La primera etapa separa la tinta del fondo mediante una sencilla operación de ecualizado (ver apartado 7.3.2).

2. La segunda etapa (ver apartado 7.3.3) asocia los píxeles etiquetados como tinta según su adyacencia.

   A estos asociaciones los llamaremos fragmentos. Esos fragmentos son de dos tipos: Fragmentos de carácter y fragmentos de ruido.

   Los fragmentos de ruido no interesan y deben ser descartados. Para descartarlos hay que reconocerlos como ruido.

   Puede ocurrir que los denominados fragmentos de carácter sean en realidad caracteres completos.

3. La tercera etapa (ver apartado 7.3.4) etiqueta los fragmentos de carácter de acuerdo a la "banda" a la que pertenecen.

   El concepto 'banda' tendrá una definición apropiada en 7.3.4. Baste decir ahora que fragmentos que corresponden a una misma línea de texto corresponden a la misma banda.

4. La cuarta etapa (ver apartado 7.3.5) agrupa los fragmentos de carácter en caracteres allí donde es necesario basándose en la información recopilada de las dos etapas previas.

   Un carácter puede estar fragmentado por razones accidentales y/o naturales. Ver apartado 7.3.1.5 sobre esto.

[9]Quizás porque la distancia interfragmento en esos casos es demasiado corta para ser apreciada por el ojo humano y, cuando no sucede así, la información de contexto es suficiente para "rellenar" los huecos e identificar el carácter en la mente del lector.

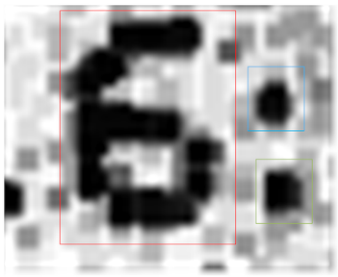


Figura 7.25: Dos caracteres. Uno de ellos es un carácter fragmentario: Los dos puntos.

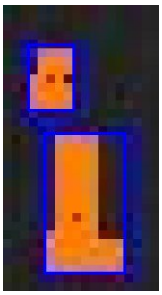


Figura 7.26: Una segmentación para una "i". ¿O es un "l")

7.3.1.5. Fragmentación natural y accidental

En el apartado 6.3.1 se indicaba que si bien los caracteres que deben ser procesados por MONICOD suelen ser formas compactas, existían excepciones a las que denominábamos *caracteres fragmentados*

El *fragmentado natural* consiste en caracteres compuestos por mas de un fragmento (ver figuras 7.25 ¿y 7.26?) . Están construidos con fragmentos no conectados en origen. Algunos de ellos forman parte del subconjunto de caracteres con el que trabaja MONICOD. Es el caso de los caracteres que presentan acento gráfico o tilde ("á", "é", "í", "ó "y "ú").

Inevitablemente estos caracteres producirán fragmentos en lugar de caracteres completos y se requiere agruparlos en un único carácter.

El *fragmentado de caracteres accidental* (ver figura 7.27) responde a deficiencias en el proceso de impresión, deficiencias en la adquisición (como una resolución demasiado detallada) y/o ruido en la imagen de entrada. En este caso los espacios entre los fragmentos son indeseables y requieren agruparse para formar las identidades completas carácter.

Tanto en un caso como en otro hablamos de espacios 'intracaracter'. Es decir, espacios entre los fragmentos que conforman un único carácter.

Hay dos tipos de espacios intracaracter.

- Espacios intracaracter que separan al carácter verticalmente. Ver figura

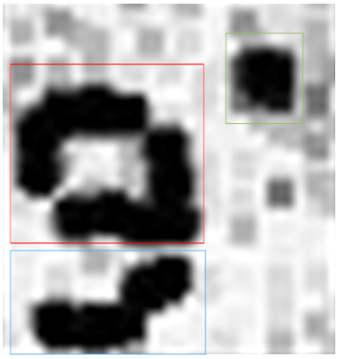


Figura 7.27: Fragmentación accidental en el "9". El fragmento a la izquierda está causado por el ruido y no pertenece a ningún carácter.

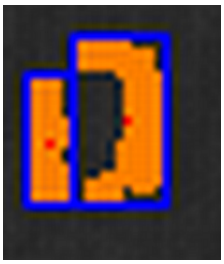


Figura 7.28: Ejemplo de espacios intracaracter verticales.

7.28.

- Espacios intracaracter que separan al carácter horizontalmente. Ver figura 7.29.

7.3.2. Binarizado

En MONICOD la operación de binarizado proporciona las semillas para el proceso de segmentación. El binarizado construye una lista (o tabla) simple de

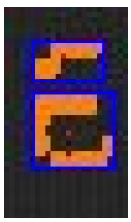


Figura 7.29: Ejemplo de espacios intracaracter horizontales.

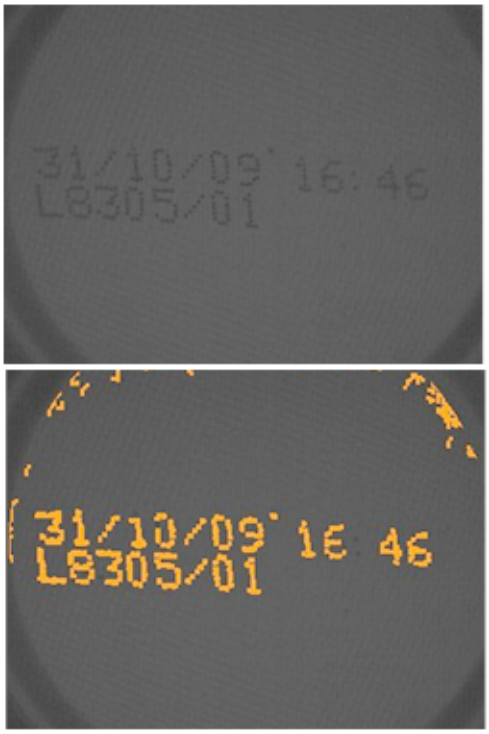


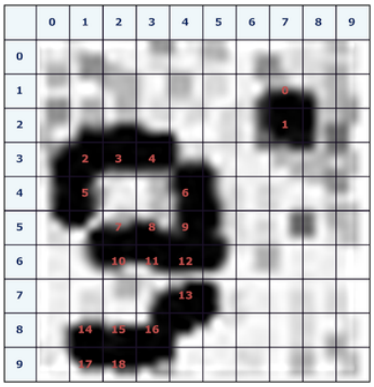
Figura 7.30: Binarizado de una imagen. (☼)

coordenadas de tinta recorriendo y etiquetando cada píxel del área de interés como tinta y fondo. Ver figura 7.30 donde se muestra el resultado del binarizado. MONICOD no trabaja nunca con la imagen binarizada explícita sino con la citada lista. Ver figura 7.31. El criterio utilizado es un umbral global computado en tiempo de calibrado porque las contramedidas hardware (la iluminación polarizada + la cámara oscura) lo permiten. La solución hardware garantiza estabilidad en la iluminación y cierta homogeneidad. No es una solución perfecta pero amortigua la necesidad de un computo de umbrales locales en tiempo de validación. Este beneficio redunda en la simplicidad y en un ahorro en cómputo que se ha juzgado satisfactorio.

7.3.3. Asociando tinta

En esta fase agrupamos los píxeles de tinta en fragmentos de carácter 'legales'.

7.3.3.1. Agregación de píxeles



Semillas	
X	Y
1	7
3	1
3	2
3	3
4	1
4	4
5	2
5	3
5	4
6	2
6	3
6	4
7	4
8	1
8	2
8	3
9	1
9	2

Figura 7.31: Ejemplo de Binarizado y tabla de coordenadas de semillas. Las semillas se obtienen directamente del binarizado.

El mecanismo de asociación elegido para MONICOD es la agregación de píxeles (introducido en el apartado 2.5.3.1). Las razones para escoger este método frente a otros se fundan en:

- Adecuación al problema

  El método de agregación de píxeles se adapta a un escenario donde los objetos a segmentar son muy pequeños en relación al entorno desde donde deben ser extraídos. Tal es el caso de los caracteres en las superficies de lata.

- Resultado explícito

  La agregación de píxeles proporciona explícitamente toda la lista de coordenadas que forman parte de cada fragmento. Resulta indiferente el orden de estas coordenadas dentro de la lista. Por otra parte los fragmentos mismos no están ordenados de ningún modo, ni de izquierda a derecha, ni por líneas.

- Eficiencia

  El método de agregación de píxeles es muy rápido porque:

  - Sólo involucra operaciones de aritmética entera[10].

[10] Esto no es necesariamente siempre cierto pero en MONICOD lo es. Por ejemplo en MONICOD la elección de agregar o no un píxel se reduce a una comparación. Pero pueden existir criterios más complejos que requieran otro tipo de operaciones y seguiría siendo el método de Agregación de Píxeles.

- El número de consultas es muy bajo.

Además durante el proceso de agregación puede aprovecharse para computar características de los objetos segmentados. Por ejemplo, el área, el centroide, altura, anchura... . Adicionalmente también crea información secundaria que puede ser empleada por etapas posteriores directamente. Ese es el caso del mapa de consultas que se utilizará en la búsqueda de líneas de código.

7.3.3.2. Semillas

En este punto ya tendremos, como un producto de la etapa descrita en 7.3.2, un conjunto de coordenadas que denominaremos 'semillas de inundación' o simplemente 'semillas'. Las semillas han sido localizadas durante el binarizado (ver 7.3.2). En ese punto las semillas son solo una colección de puntos inconexa, pero es de esperar (si el código está presente) que un subconjunto de esas semillas sean coordenadas de píxeles de los caracteres del código de caducidad en el interior del área de interés. El resto deben ser coordenadas de ruido.

7.3.3.3. Asociaciones

Las asociaciones se producen entre puntos que:

- Tiene una cualidad común. Para MONICOD, esa cualidad es compartir un determinado rango de luminancia.
- Son adyacentes directamente. Ver el significado de adyacencia en A.3.2.1.
- Son adyacentes indirectamente. Esto quiere decir que es posible encontrar una serie de puntos consecutivamente adyacentes que los conecten y compartan dicha cualidad común. Ver el significado de conectividad en A.3.2.3.

Esta asociación tiene un carácter extensivo:

a) A y B o están asociados o no lo están. No puede ocurrir que dos puntos estén y no estén asociados al mismo tiempo.

b) Si A está asociado a C, y B está asociado a C entonces A y B están asociados. No puede ocurrir que un punto esté asociado a un segundo y no lo esté a un tercero, cuando el segundo y el tercero están asociados entre sí.

En esta asociación ningún punto tiene especial preeminencia sobre otro.

7.3.3.4. Inundación

La inundación es el procedimiento por el que buscamos **asociaciones en el conjunto de semillas**. Su nombre hace alusión a la formación de los fragmentos, que crecen desde las semillas como si de una inundación se tratase.

El funcionamiento del esquema es tal y como sigue[11]:

1. Inicializamos la lista de fragmentos a *0* fragmentos contenidos.
2. Inicializamos a NO VISITADO un mapa de consultas. Un mapa de consultas es una matriz de las dimensiones de la imagen. Cada valor de la matriz indica si esa posición de la imagen ha sido visitada o no. Al principio ninguna posición de la imagen ha sido visitada.
3. Mientras queden semillas NO VISITADAS en la lista de semillas hacemos:
   a) Cogemos una semilla i NO VISITADA de la lista de semillas. No importa cual.
   b) La marcamos como VISITADA.
   c) Inauguramos un fragmento nuevo F_i y ponemos en su lista de puntos la semilla i que acabamos de obtener.
   d) $n = 0$
   e) Mientras n sea menor que el número de semillas en la lista contenida en el fragmento F_i hacemos
      1) Cogemos la n-esima semilla del fragmento F_i
      2) Calculamos la adyacencia directa de la semilla i. Ignoramos las posiciones ya visitadas. Marcamos las posiciones que no son semilla como VISITADAS. Introducimos las semillas NO VISITADAS[12] en el fragmento F_i y las marcamos también como VISITADAS.
      3) $n = n + 1$
      4) Regresamos al paso e)
   f) Verificamos si el fragmento puede ser un fragmento de carácter: Fragmentos demasiado grandes o pequeños, demasiado altos o anchos son declarados INÚTILES. Los fragmentos restantes son declarados VALIDOS.
   g) Regresamos a 3.

Este esquema queda esbozado en los algoritmos 7.8 y 7.7.

Un ejemplo de resultado en figura 7.32.

7.3.3.5. Características de un fragmento

Como el método de la inundación recorre cada punto de cada fragmento resulta muy fácil el cálculo de las características:

- Para cualquier fragmento F_i con n elementos o coordenadas (x, y) tendremos que:

[11] Por claridad se han omitido en la exposición optimizaciones llevadas a cabo asi como el cálculo paralelo de las estadísticas del fragmento presentadas más adelante.

[12] Podría no haber ninguna.

Algoritmo 7.7 Calcula el Vecindario NO visitado

Precondición: *imagen*[]:La imagen. *semilla*:Una posicion dentro de la imagen
Postcondición: *vecindario*[]:Una serie de posiciones inmediatamente adyacentes que NO habían sido visitadas en el momento de la llamada.

```
i ← 0
mientras Hayan posiciones adyacentes NO visitadas hacer
    posicionadyacente ← Coger-una-posicion-adyacente-NO-visitada()
    Marcar-como-visitada(posicionadyacente)
    vecindario[i] ← posicionadyacente
    i ← i + 1
fin mientras
```

Algoritmo 7.8 Algoritmo de Inundación

Precondición: *imagen*[]:La imagen. *areadeinteres*[]:Un contenedor que proporciona las coordenadas de los puntos que corresponden al área de interes.
Postcondición: *zonas*[][]:Una serie de zonas que corresponden aproximadamente a las áreas ocupadas por los caracteres

```
nzonas ← 0
semillas[] ← Rastrear-todas-las-semillas-viables-del-área-de-interes(areadeinteres[])
mientras Hayan semillas NO visitadas hacer
    semillainicial ← Coger-una-semilla-NO-visitada(semillas[])
    Marcar-como-visitada(semillainicial)
    i ← 0
    cola[i] ← semillainicial
    finaldecola ← 1
    mientras i sea menor que finaldecola hacer
        vecindario[] ← Calcular-Vecindario-NO-Visitado(cola[i])
        cola[] ← vecindario[]
        finaldecola ← finaldecola+Obtener-Numero-Vecinos(vecindario[])
        i ← i + 1
    fin mientras
    si Es cola[] una zona valida entonces
        zonas[nzonas] ← cola[]
        nzonas ← nzonas + 1
    fin si
fin mientras
```

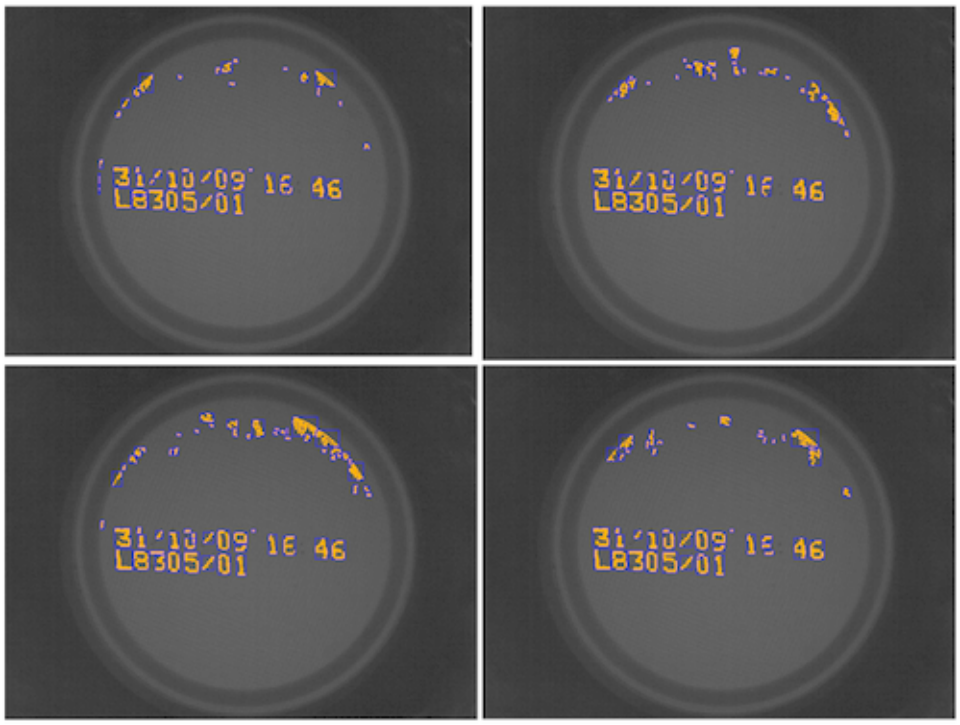


Figura 7.32: Segmentación en cuatro *frames* consecutivos. Obsérvese que cada vez la segmentación genera resultados diferentes para los mismos parámetros. Tales diferencias exhiben el efecto nocivo de la iluminación. (☼)

- n. Esto es, el recuento de los píxeles que forman el área. Es una medida de la cantidad de tinta en el fragmento, medida en número de píxeles.
- $(x_c, y_c) : x_c = \frac{\sum_{j=1}^{n} x_j}{n}, y_c = \frac{\sum_{j=1}^{n} y_j}{n}$ con $(x_j, y_j) \epsilon F_i$.Centroide del conjunto de puntos.
- x_{min} .Esto es, el valor más bajo de la coordenada x entre todas las coordenadas $(x, y) \epsilon F_i$ de los puntos contenidos en el conjunto.
- x_{max}. Esto es, el valor más alto de la coordenada x entre todas las coordenadas $(x, y) \epsilon F_i$ de los puntos contenidos en el conjunto.
- y_{min}. Esto es, el valor más bajo de la coordenada y entre todas las coordenadas $(x, y) \epsilon F_i$ de los puntos contenidos en el conjunto.
- y_{max}. Esto es, el valor más alto de la coordenada y entre todas las coordenadas $(x, y) \epsilon F_i$ de los puntos contenidos en el conjunto.
- Altura. Medida que se deriva de forma inmediata de la diferencia entre x_{max} y x_{min}.
- Anchura. Medida que se deriva de forma inmediata de la diferencia entre y_{max} y y_{min}.
- Puntos exteriores. Aquellas posiciones que son adyacentes de forma directa con una o más posiciones que no son semilla.
- Puntos interiores. Aquellas posiciones que sólo son adyacentes directamente con posiciones que son semilla.

7.3.4. Lineas y bandas

7.3.4.1. El objetivo

En la fase anterior nos hemos hecho con una colección de fragmentos de carácter. Este apartado versará sobre ordenar estos fragmentos de acuerdo con la línea de código a la que pertenece.

Hay dos razones por las que esta ordenación es importante.

- Para validar el código es crítico conocer el orden de lectura de los caracteres. Los caracteres tienen un significado de acuerdo con la posición en la que estén respecto al resto de caracteres del código. Los caracteres se leen de izquierda a derecha línea a línea. Los caracteres esperados están en posiciones esperadas así que necesitamos saber en qué línea está cada carácter.
- Tenemos fragmentos de carácter que debemos agrupar. Fragmentos que estén en líneas diferentes no pueden ser parte del mismo carácter. Si identificamos las líneas a las que pertenece cada fragmento reducimos el número de agrupamientos a considerar.

7.3.4.2. El principio

Las líneas del código en un código sin inclinación[13] son horizontales (en escritura occidental). Por otra parte las líneas de código están separadas entre sí por espacios de interlineado descritos en 6.3.2. Expresado de otro modo: El área entre dos espacios de interlineado consecutivos tiene al menos uno o más fragmentos de tinta localizados en la etapa anterior a los que identificaremos como líneas.

El principio utilizado se fundamenta en identificar el espacio de interlineado (ver 6.3.2). Para localizarlo buscaremos "caminos divisores" de izquierda a derecha (y/o de derecha a izquierda) que tengan costo mínimo. El costo se define como la longitud del camino considerando que:

- Un camino divisor sin obstáculos (tinta que se interpone en el trayecto) es un camino óptimo o de costo 0.
- Si hay tinta en el trayecto el camino divisor intentará rodearla. Llevar a cabo un rodeo aumenta la longitud del camino divisor. El costo de rodear el obstáculo es variable.
- Si no es posible rodear la tinta el costo del camino divisor se asume infinito, y se designará como un camino divisor muerto (no hay espacio de interlineado allí).

Los caminos divisores encontrados corresponden a espacios de interlineado.

Este principio se sustenta en dos ideas:

1. Las líneas de texto están separadas por espacios de interlineado que ofrecen costo 0 (o de muy bajo costo) para los caminos que tracemos de izquierda a derecha (o de derecha a izquierda).
2. Es poco probable que podamos trazar un camino de costo 0 (o de muy bajo costo) fuera de los espacios de interlineado (separando una línea de código en dos).

Estas ideas serán descritas con detalle en apartados siguientes pero debe subrayarse la presunción de que el código no se presenta inclinado respecto al eje de lectura. O, de estarlo, la inclinación es suficientemente suave para no invalidar las ideas mencionadas.

7.3.4.3. ¿Por qué hacerlo ahora?

La razón por la que la separación de líneas es anterior al agrupamiento de fragmentos de carácter (cuando la lógica parece decirnos que debe ser al contrario) está en la segunda razón expresada en el apartado 7.3.4.1. Dos fragmentos en líneas distintas no son asociables. Si sabemos en qué linea está cada fragmento sólo trataremos de asociar cada fragmento a fragmentos en su misma

[13] En inglés, Skew.

línea, repercutiendo en beneficios computacionales. Como veremos ES POSIBLE encontrar las líneas antes de definir los caracteres porque los fragmentos de carácter nos ofrecen información necesaria para identificar los espacios de interlineado.

Sin embargo existe un obstáculo severo: Los espacios *intracaracter* e *intercaracter*.

Considerando el principio aplicado descrito en 7.3.4.2 utilizamos el espacio de interlineado para separar lineas. Pero aparte de este, hay otros espacios en el texto: El espacio *intercaracter* y el espacio *intracaracter*. Se comenta en 7.3.1.5 que los espacios *intracaracter* son las hendiduras entre los fragmentos de código. Por otro lado los espacios *intercaracter* son los espacios entre caracteres. La última fase de la segmentación agrupa los fragmentos, y con ello, define que espacios son *intracaracter* e *intercaracter* y cuales no. Pero en este punto aún no tenemos esa información. En el procedimiento seguido por MONICOD un espacio *interlinea* es indistinguible de un espacio *intercaracter* o *intracaracter*. Por ello corremos el riesgo de establecer que dos fragmentos de un carácter queden en líneas separadas. O que caracteres queden asignados a líneas que en realidad no le corresponden.

¿Cómo puede confundirse un espacio *intercaracter*, *intracaracter* e *interlinea*?

Dependerá de cuan bien esté alineado el código con el eje de "lectura" establecido en la adquisición.

Este problema sólo existe si el código no está alineado con el eje de "lectura". Es decir: Hay rotación de código. Como se comenta en **??** al contar con la ventaja de un entorno controlado, no necesitamos lidiar con el fenómeno. En conclusión el obstáculo severo de los espacios *intracaracter* e *intercaracter* es susceptible de ser minimizado.

7.3.4.4. Bandas y fronteras de banda

El área de interés se divide en bandas. Una banda es una franja rectangular horizontal que secciona el área de interés.

Las bandas vienen definidas por sus límites superior e inferior que denominamos fronteras de banda. La obtención de esos límites es el procedimiento de división de bandas.

Cada línea de código debe corresponder a una, y sólo a una, banda. Sin embargo no toda banda contiene código: Puede contener ruido. Para enfrentar esa posibilidad dos pruebas son efectuadas:

- Comprobar que haya en el interior de la banda suficiente tinta para una línea de código.
- Comprobar que la altura de la banda (distancia vertical media entre las fronteras de banda) debe ser, al menos, igual o mayor a la altura mínima de un carácter.

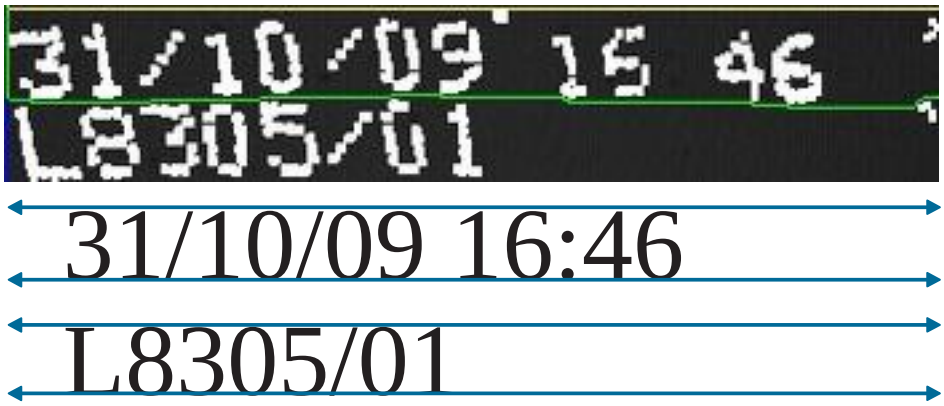


Figura 7.33: Búsqueda de espacios interlineales.

7.3.4.5. Dos Mecanismos de búsqueda de caminos

El procedimiento de separación de líneas y bandas comprende dos mecanismos.

El primer mecanismo -búsqueda de fronteras evidentes- es simple y altamente eficiente, y permite resolver la mayoría de las situaciones de interlineado a las que debe enfrentarse el sistema. Sin embargo es rígido y en ciertos escenarios, no cumple. Se detalla en la subsección 7.3.4.6.

Para esos casos se ha ideado un segundo mecanismo -búsqueda de fronteras no evidentes- menos eficiente pero más flexible que complementa al primero. Puede entenderse el primer mecanismo como un caso particular (el mejor caso) del segundo mecanismo. Se detalla en la subsección 7.3.4.7.-

Los dos mecanismos comparten el principio de buscar caminos de mínimo costo o incluso costo 0, con cierto grosor, que recorren el área del interés transversalmente. El costo viene dado por los píxeles con tinta que incluye el camino. En el caso de costo cero, el camino no incluye tinta.

7.3.4.6. Fronteras evidentes

Simplemente se buscan filas sin tinta. Una fila sin tinta corresponde a un camino rectilíneo de grosor de un píxel donde no hay presencia de tinta.

En general una línea de código no tiene, entre sus límites inferior y superior, filas sin tinta. Eso es así porque los códigos están construidos con caracteres del alfabeto occidental y estos exhiben continuidad vertical de arriba a abajo (vertical).

Son precisamente los casos de fragmentación natural (comentados en 7.3.1.5) los que presentan una, y solo una, discontinuidad vertical (ñ, á, é, í, ó, ú, i, j, :). No hay casos de fragmentación natural con más de una discontinuidad vertical en el conjunto de caracteres que nos interesan.

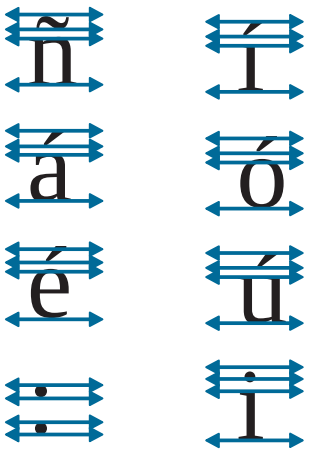


Figura 7.34: Las filas evidentes determinan bandas evidentes con caracteres fragmentarios.

Como puede verse en la figura 7.34 se generan franjas adicionales que siendo correctas dividen la línea de código.

Considerando la frecuencia de aparición de los caracteres en la generación de códigos, es improbable que en una línea todos los caracteres estén fragmentados de forma natural. La separación errónea de una línea en dos bandas sólo podría ocurrir si el código está muy fragmentado. Y cuando eso sucede dar un veredicto de ilegibilidad no es descabellado.

En 7.9 se muestra el algoritmo de búsqueda de filas evidentes.

En la implementación real se ha optimizado para aprovechar información secundaria del proceso de agregación de píxeles descrito en 7.3.3.4.

Este procedimiento tiene dos debilidades:

- El ruido puede frustrar una separación por banda evidentes. Ver a este respecto la figura 7.35.
- El código puede estar inclinado respecto a la horizontal. Es decir no está alineado al eje de coordenadas de la imagen Esto menoscaba severamente el procedimiento porque los caminos rectilíneos de costo 0 dejan de ser paralelos a la horizontal de la imagen. Ver figura 7.43.

El mecanismo de fronteras no evidentes que se describe en el siguiente apartado hace frente a ambas debilidades.

7.3.4.7. Fronteras no evidentes

El procedimiento de fronteras no evidentes es menos eficiente que el de fronteras evidentes, y se aplicará sólo sobre aquellas franjas que dispongan de suficiente

Algoritmo 7.9 Búsqueda de Filas Evidentes

Precondición:
Postcondición:

```
aperturadebanda ← verdadero
tintaacumulada ← 0
contador ← 0
para i = FilaMinima a FilaMaxima hacer
    si la fila i tiene tinta entonces
        si aperturadebanda es igual a verdadero entonces
            filasevidentes[contador] ← i − 1
            contador ← contador + 1
            aperturadebanda ← falso
        fin si
        tintaacumulada ← tintaacumulada + tinta[i]
    sino si tintaacumulada es mayor que 0 entonces
        filasevidentes[contador] ← i
        contador ← contador + 1
        tintaacumulada ← 0
        aperturadebanda ← verdadero
    fin si
fin para
```

Recorremos las filas de arriba a abajo. Inicialmente buscamos una fila con tinta. Cuando la encontramos registramos la fila anterior (que no tenía tinta) y pasamos a buscar una fila sin tinta. Cuando la encontramos la registramos. Repetimos el proceso. Las filas registradas delimitan las bandas. En una banda todas las filas tienen tinta, o bien ninguna de ellas tiene.

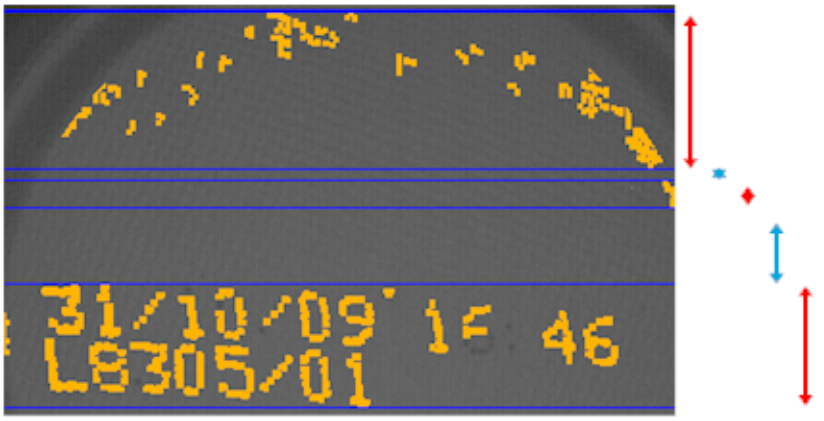


Figura 7.35: En este caso la búsqueda de líneas evidentes no puede separar las líneas de código a causa de caracteres no perfectamente alineados con el renglón del texto: El inicial '3' de la primera línea ha quedado a más altura que el '6' final; en condiciones ideales estarían a la misma altura porque los dos caracteres pertenecen a la misma línea. (☼)

tinta para ser código, y su altura sea mayor que numero de lineas de codigo × altura minima de caracter píxeles.

El procedimiento tiene carácter multiagente y recursivo. El principio es lanzar un número indeterminado de agentes independientes que compiten buscando caminos[14] divisores (fronteras de banda). De los caminos hallados nos quedamos con el mejor: El camino más corto, que también es el primero en completarse.

Denominaremos a estos agentes *divisores.* Los *divisores* son idénticos entre si, salvo que la mitad de ellos tienen una dirección preferente de izquierda a derecha (OESTE A ESTE) y la otra mitad de derecha a izquierda (ESTE A OESTE). Ver figura 7.36. También difieren en el punto de partida, distinto para cada uno: Los preferentes de izquierda a derecha empiezan en el lado izquierdo (OESTE) de la región de interés. Los preferentes de derecha a izquierda empiezan en el lado derecho (ESTE).

El *divisor* mantiene un cuaderno de bitácora (o de ruta) consistente en un historial de las coordenadas recorridas por él hasta ese momento. Esas coordenadas no podrán ser exploradas por *divisores* con su misma dirección preferente. Sin embargo están disponibles para los *divisores* de dirección preferente opuesta.

Cuando el *divisor* no puede puede progresar se produce la mitosis[15] del *divisor*: El *divisor* genera dos copias con un historial de ruta idéntico al suyo. Hecho esto, el *divisor* original "muere" .

[14]La razón del término 'camino' para denominar a una frontera de banda procede precisamente de la forma con la que los agentes -entidades móviles- se comportan: Buscan "caminos o rutas" para alcanzar su destino.

[15]Llamada así por su similitud con la división celular natural.

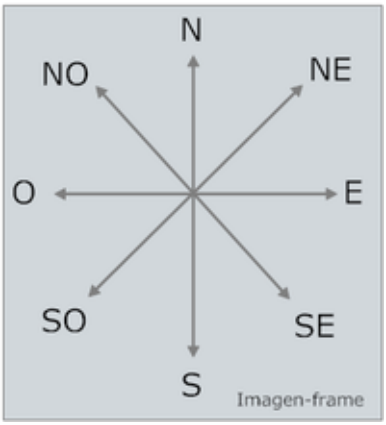


Figura 7.36: Rosa de los vientos. El sistema de referencia de un *divisor*

El *divisor* El *divisor* sólo se desplaza a posiciones legales. Una posición legal es aquella donde no hay tinta, no ha sido visitada por otro *divisor* con la misma dirección preferente, y no cruza la frontera evidente.

El procedimiento del *divisor* será:

1. Avanza en la dirección prioritaria[16] (este u oeste). Ver figura 7.37. Si no es una posición legal se va al paso 2. Una posición ilegal lo es por cumplir una de estas condiciones: Contiene tinta o ya ha sido visitado por un *divisor* con la misma dirección prioritaria. Ver figura 7.38.

2. Se lanza el protocolo oblicuo. Ver figura 7.39.

   a) Si no hay tinta al noreste (o noroeste) de donde está, se produce una copia del original que se lanzará desde ahí.

   b) Si no hay tinta al sureste (o suroeste) de donde está, se produce una copia del original que se lanzará desde ahí.

3. Si el paso 2 ha fracasado se lanza el protocolo perpendicular (ver figura 7.40).

   a) Si no hay tinta al norte de donde está, se produce una copia del original que de lanzará desde ahí.

   b) Si no hay tinta al sur de donde está, se produce una copia del original que de lanzará desde ahí.

4. Si el paso 3 ha fracasado el *divisor* se declara *muerto*. Ver figura 7.41

[16] De izquierda a derecha(dirección Este) o de derecha a izquierda(dirección Oeste)

Los Pasos Procedemos así:

1. Selección de franja sobre el que aplicaremos el procedimiento.
2. Selección de mejor fila de inicio.
3. Se lanzan dos *divisores* simultáneamente. Uno de prioridad izquierda a derecha. Y otro de prioridad derecha a izquierda. El punto de origen será la fila de inicio pero cada uno empezará en posiciones opuestas. Sus historiales estarán inicializados a ruta vacía.
4. Se avanza un paso en la exploración mientras queden *divisores* vivos:
   a) Los *divisores* existentes ejecutan el procedimiento de *divisor* una vez.
   b) Si un *divisor* alcanza el extremo opuesto del que partió la exploración se declara finalizada con éxito.
5. Si la exploración finaliza con éxito la ruta del *divisor* es frontera de banda y se abandona el procedimiento.
6. Si la exploración fracasa se selecciona la siguiente mejor fila de inicio y se vuelve a 2. Tras n fracasos la operación de interlineado ha fracasado y la validación NO es posible. En este punto se podría tomar la mejor fila de inicio (o alternativamente el mejor camino[17] encontrado hasta el momento) pero se han desechado estas opciones y se prefiere declarar la lata como no válida.
7. Se vuelve al paso 4.

Algunos ejemplos de la división por bandas pueden verse en las figuras 7.42 y 7.43. En el primer caso el código no presenta prácticamente inclinación. Con todo ha sido necesario aplicar la técnica de *divisores* de bandas a causa del ruido. En el segundo caso el código presenta una considerable inclinación. Aún así ha sido capaz de separarlo.

Algoritmos Con más detalle se presentan los algoritmos 7.10, 7.11 y el simétrico de este último, 7.12.

7.3.4.8. Debilidades en el método

Caracteres fragmentados legalmente Caracteres no atómicos como la "í", "j" o "ñ" pueden verse disgregados en diferentes bandas si el *divisor* logra "colarse" entre los fragmentos que lo componen.

[17] El camino que ha estado más próximo al otro extremo de cualquiera de las exploraciones iteradas.

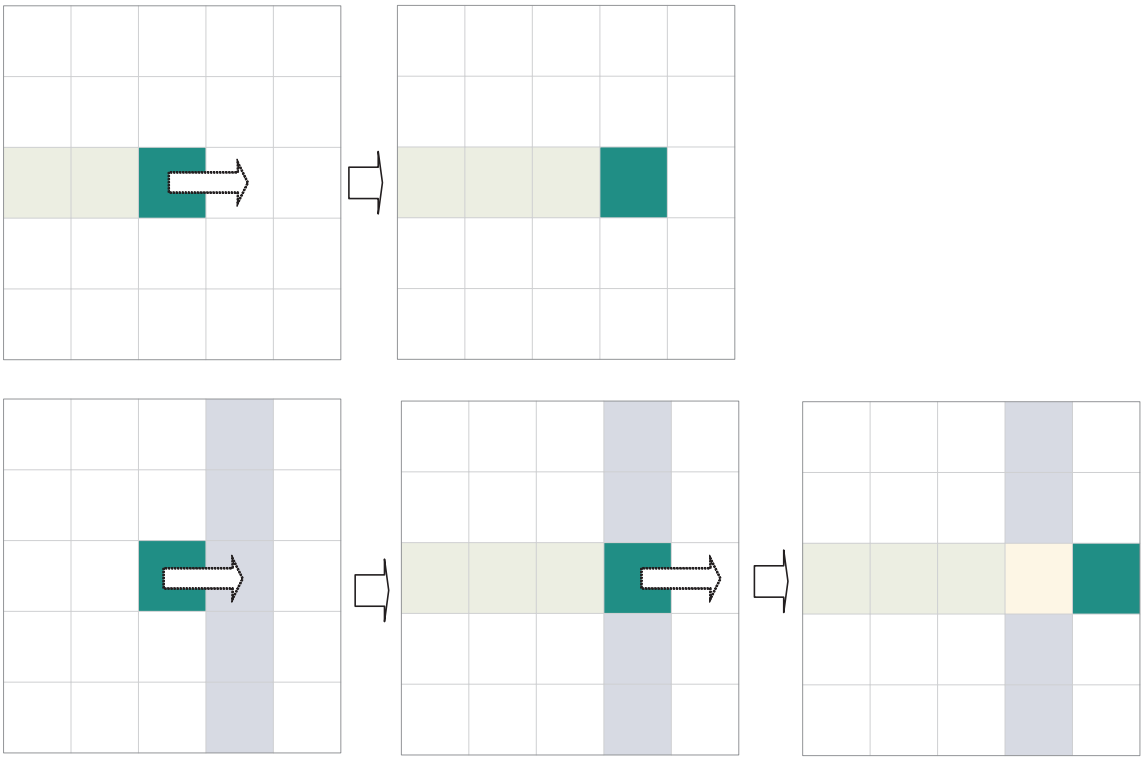


Figura 7.37: El *divisor* avanza. Arriba, un avance sin contingencias. Abajo avanza sobre una posición visitada por un *divisor* con dirección prioritaria opuesta.

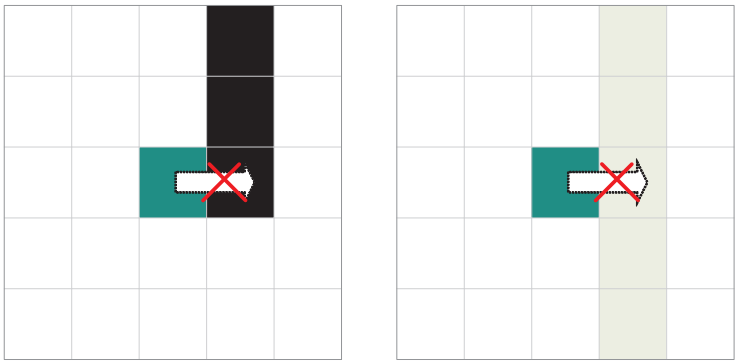


Figura 7.38: El *divisor* no puede avanzar. A la izquierda encuentra tinta. A la derecha encuentra una posición visitada por un *divisor* con la misma dirección prioritaria.

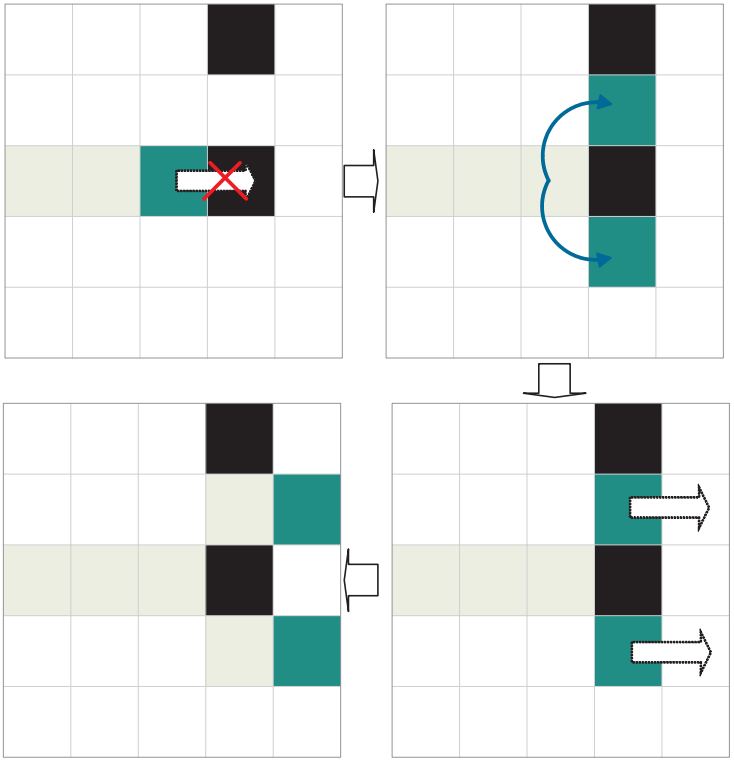


Figura 7.39: Mitosis/División oblicua. El *divisor* se reproduce. Arriba izquierda el *divisor* no puede avanzar. Arriba derecha el *divisor* se reproduce al noreste y al sureste dado que es posible. Abajo derecha los dos *divisores* tratan de avanzar hacia adelante y lo consiguen. Abajo izquierda, el nuevo estado. Los dos *divisores* han superado el obstáculo y progresan.

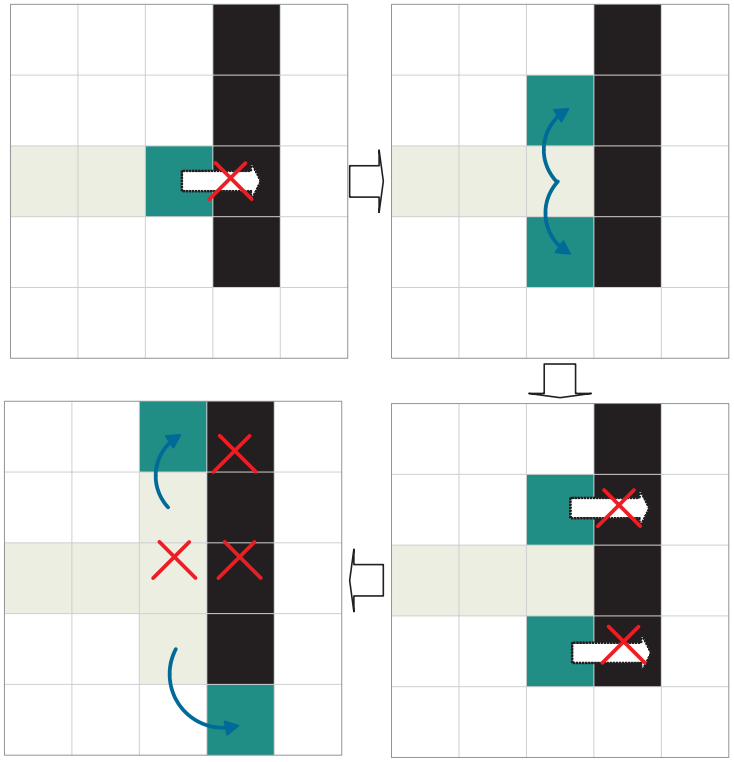


Figura 7.40: Mitosis/división perpendicular. El *divisor* se reproduce. Arriba izquierda el *divisor* no puede avanzar. Arriba derecha el *divisor* se reproduce al norte y al sur dado que puede hacerlo y el protocolo oblicuo ha fallado. Abajo derecha los dos *divisores* tratan de avanzar hacia adelante. Ambos fracasan. Abajo izquierda. El *divisor* superior fracasa al protocolo oblicuo y aplica el protocolo perpendicular. Trata de avanzar hacia el norte y lo consigue pero hacia abajo no puede hacerlo porque la posición ha sido ya visitada por un *divisor* con la misma dirección prioritaria. El *divisor* inferior aplica el protocolo oblicuo y, aunque falla en el noreste consigue avanzar hacia el sureste.

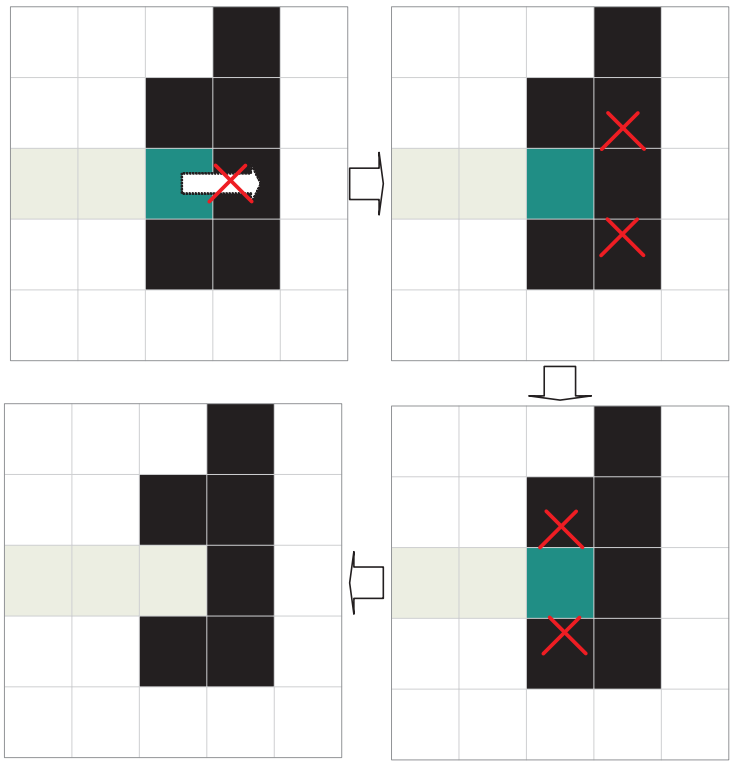


Figura 7.41: Un ejemplo de muerte de *divisor*. Arriba izquierda el *divisor* no puede avanzar. Arriba derecha el *divisor* aplica el protocolo oblicuo tratando de avanzar al noreste y/o al sureste. Fracasa. Abajo derecha, el divisor aplica el protocolo perpendicular tratando de avanzar al norte y/o al sur. Fracasa de nuevo. Abajo izquierda, sin alternativas el *divisor* se declara truncado y "muere".

Figura 7.42: Separación de líneas.

Algoritmo 7.10 Interlineado Por *divisores*.

Precondición: *mapa*[][]: Mapa de consultas producto de la fase de inundación
Precondición: $xinf, xsup$: Límites superior e inferior para la exploración de los divisores
Precondición: $yizq, yder$: Límites izquierda y derecha para la exploración de los divisores
Postcondición: *lineadivisoria*[]: Una línea divisoria

```
mientras Hay divisores vivos hacer
    para i = 0 to numerodivisoressimultaneos hacer
        si DivisorDeIzquierdaADerecha[i] esta vivo entonces
            Se marca como visitado por el divisor de IzquierdaDerecha la
            posición actual de DivisorDeIzquierdaADerecha[i]
            si DivisorDeIzquierdaADerecha[i] ha alcanzado el otro extremo
            yder entonces
                lineadivisoria ← Obtener-Ruta(DivisorDeIzquierdaADerecha[i])
                retornar
            sino si DivisorDeIzquierdaADerecha[i] no esta bloqueado en-
            tonces
                Avanza-Al-Este(DivisorDeIzquierdaADerecha[i])
            sino
                Ramificacion-hacia-el-Este(DivisorDeIzquierdaADerecha[i])
                Se declara muerto y se destruye el
                DivisorDeIzquierdaADerecha[i]
            fin si
        fin si
        si DivisorDeDerechaAIzquierda[i] esta vivo entonces
            Se marca como visitado por el divisor de DerechaIzquierda la
            posición actual de DivisorDeDerechaIzquierda[i]
            si DivisorDeDerechaAIzquierda[i] ha alcanzado el otro extremo
            yizq entonces
                lineadivisoria ← Obtener-Ruta(DivisorDeDerechaAIzquierda[i])
                retornar
            sino si DivisorDeDerechaAIzquierda[i] no esta bloqueado en-
            tonces
                Avanza-Al-Oeste(DivisorDeDerechaAIzquierda[i])
            sino
                Ramificacion-hacia-el-Oeste(DivisorDeDerechaAIzquierda[i])
                Se declara muerto y se destruye el
                DivisorDeDerechaAIzquierda[i]
            fin si
        fin si
    fin para
fin mientras
```

Algoritmo 7.11 Ramificación hacia el ESTE.

Precondición: $mapa[][]$: Mapa de consultas producto de la fase de inundación
Precondición: $xinf, xsup$: Límites superior e inferior para la exploración de los divisores
Postcondición:
Se destruye el DivisorDeIzquierdaADerecha[i]
si Hay salida oblicua **entonces**
  si Hay salida al Noreste **entonces**
    $clon \leftarrow$ `Clonar(DivisorDeIzquierdaADerecha[`i`])`
    $Avanza - Al - Noreste(clon)$
  fin si
  si Hay salida al Sureste **entonces**
    $clon \leftarrow$ `Clonar(DivisorDeIzquierdaADerecha[`i`])`
    $Avanza - Al - Sureste(clon)$
  fin si
  Se destruye el DivisorDeIzquierdaADerecha[i]
  retornar
fin si
si Hay salida perpendicular **entonces**
  si Hay salida al Norte **entonces**
    $clon \leftarrow$ `Clonar(DivisorDeIzquierdaADerecha[`i`])`
    $Avanza - Al - Norte(clon)$
  fin si
  si Hay salida al Sur **entonces**
    $clon \leftarrow$ `Clonar(DivisorDeIzquierdaADerecha[`i`])`
    $Avanza - Al - Sur(clon)$
  fin si
  Se destruye el DivisorDeIzquierdaADerecha[i]
  retornar
fin si

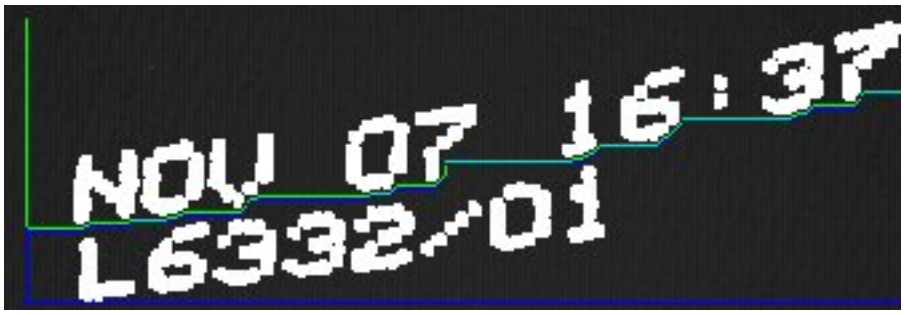


Figura 7.43: Código ligeramente inclinado afecta sobremanera la presencia de fronteras evidentes como separadores de banda. La separación entre las dos líneas de código NO puede ser una fila evidente. En su lugar la frontera entre dos líneas obtenida es obtenida mediante bots *divisores* que resuelven a pesar de la inclinación notoria del código.

Algoritmo 7.12 Ramificación hacia el OESTE.

Precondición: $mapa[][]$: Mapa de consultas producto de la fase de inundación
Precondición: $xinf, xsup$: Límites superior e inferior para la exploración de los divisores
Postcondición:

```
si Hay salida oblicua entonces
    si Hay salida al Noroeste entonces
        clon ← Clonar(DivisorDeDerechaAIzquierda[i])
        Avanza − Al − Noroeste(clon)
    fin si
    si Hay salida al Suroeste entonces
        clon ← Clonar(DivisorDeDerechaAIzquierda[i])
        Avanza − Al − Suroeste(clon)
    fin si
    Se destruye el DivisorDeDerechaAIzquierda[i]
    retornar
fin si
si Hay salida perpendicular entonces
    si Hay salida al Norte entonces
        clon ← Clonar(DivisorDeDerechaAIzquierda[i])
        Avanza − Al − Norte(clon)
    fin si
    si Hay salida al Sur entonces
        clon ← Clonar(DivisorDeDerechaAIzquierda[i])
        Avanza − Al − Sur(clon)
    fin si
    Se destruye el DivisorDeDerechaAIzquierda[i]
    retornar
fin si
```

Figura 7.44: Las micro-concavidades del "7" y el "1" extinguen los *divisores*.

Figura 7.45: Ausencia de espacios de interlineado impiden avanzar al *divisor*.

Concavidades La presencia de concavidades en el código resulta letal tal como puede apreciarse en la figura 7.41. El *divisor*, tal como ha sido diseñado, es incapaz de escapar. Hay caracteres, como la "C" o el "5", que son ineludiblemente cóncavos, pero además el código presenta micro-concavidades con las que es difícil lidiar. Ver figura 7.44. Sobre esto en la sección 2.4.1 se expuso algunas técnicas susceptibles de amortiguar este problema.

Ausencia de espacio interlineal en todo el recorrido El engrosamiento que produce el preprocesamiento es efectivo reduciendo el número de fragmentos de carácter y facilitando el agrupamiento posterior pero también afecta negativamente a espacios muy necesarios como el que separa a los caracteres o, en este, caso, el espacio de interlineado. Ver la figura 7.45.

Sin espacio de interlineado la división por bandas fallará irremediablemente.

7.3.4.9. Clasificación de bandas

- Bandas de interés
- Bandas de NO interés

Los criterios para separar entre bandas de interés y bandas de no interés tienen que ver con las propiedades que debe tener una banda para contener una línea de código:

1. Banda con tinta suficiente.
   - Si no hay tinta suficiente entonces no hay línea código (o la línea de código está incompleta y supondría un error automático)
2. Banda de anchura mínima
   - Si la anchura de la banda no es suficiente entonces no puede albergar caracteres, con lo que no es una línea de código (o la línea de código está muy degradada y supondría un error automático)

7.3.5. El agrupamiento de fragmentos en caracteres

7.3.5.1. Lo que tenemos hasta ahora: Fragmentos y líneas

Es necesario agrupar los fragmentos en caracteres porque la unidad mínima con significado en la validación es el carácter. Agrupar no será una tarea trivial.

Para empezar, y suponiendo una impresión que NO presente inclinación respecto al eje (u orientación) de lectura, habrá que distinguir entre dos tipos de fragmentación de carácter (ver figura 7.46) :

- **Fragmentación horizontal**

  La fragmentación horizontal descompone el carácter en fragmentos que se sitúan a ambos lados de una linea divisoria imaginaria horizontal (paralela al eje de lectura).

- **Fragmentación vertical**

  La fragmentación horizontal descompone el carácter en fragmentos que se sitúan a ambos lados de una linea divisoria imaginaria vertical (perpendicular al eje de lectura).

Se ha constatado que la fragmentación horizontal es mucho más común que la fragmentación vertical (ver figura 7.47) por lo que se han desestimado procedimientos específicos para afrontar esta última. Como luego veremos, este hecho será importante para el agrupamiento.

Otros tipos de fragmentación, como la **fragmentación oblicua** no se manifiestan.

Es importante denotar que el tipo de fragmentación sólo tiene sentido cuando se aplica sobre caracteres. No se aplica a la fragmentación de ruido, porque para el ruido no es aplicable una orientación de lectura.

Figura 7.46: Fragmentación vertical a la izquierda. Fragmentación horizontal a la derecha.

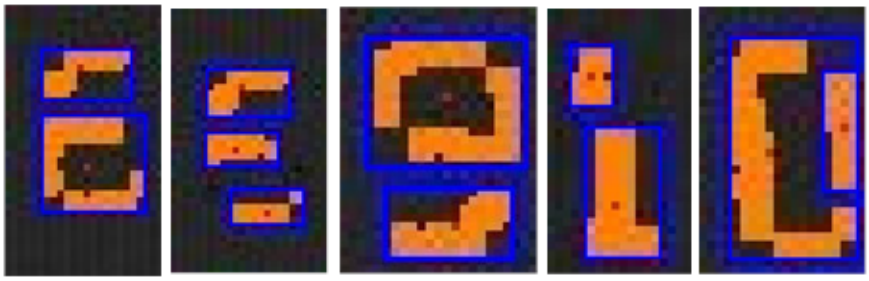


Figura 7.47: Caracteres fragmentados horizontalmente. La fragmentación horizontal es mucho más común que la fragmentación vertical .

Figura 7.48: Bandas y grupos. En (a) la linea verde indica una separación (incorrecta) de bandas. En (b) el resultado del agrupamiento:El '7' queda fragmentado.

7.3.5.2. Propósitos

Los efectos que se obtienen con el agrupamiento de fragmentos son:

- Pasamos de tratar con fragmentos a tratar con caracteres.
- Al discernir que un grupo de fragmentos conforman un carácter, resolvemos los espacios *intracaracter*.
- Cuando dos caracteres consecutivos[18] se constituyen se localiza el espacio *intercaracter* que media entre ellos.

7.3.5.3. Los principios

- Nunca agruparemos fragmentos de diferentes bandas porque NO existen caracteres que ocupen dos líneas. En otras palabras, **sólo son candidatos** para formar parte de un agrupamiento los fragmentos de la misma banda. Ver sobre esto figura 7.48

  Dado que la fase anterior ha etiquetado los fragmentos en las líneas a las que pertenecen, con el principio descrito se reduce considerablemente el número de combinaciones posibles para intentar el agrupamiento porque sólo fragmentos de la misma línea pueden ser agrupados entre sí.
- **Son fuertes candidatos** a formar parte de un agrupamiento aquellos que presentan solapamiento vertical con fragmentos que están ya dentro del agrupamiento. Ver el apartado 7.3.5.4.
- Un fragmento **es candidato** a formar parte de un agrupamiento sólo por estar más "próximo" de ese agrupamiento que de cualquier otro. La proximidad se establece con respecto a qué distancia se use (Euclidiana, Manhattan,..). Ver el apartado A.3.2.2.

[18] Entiendase por caracteres consecutivos aquellos que son leidos el uno después del otro inmediatamente.

7.3.5.4. Solapamiento de proyecciones sobre el eje *y*

El solapamiento de las proyecciones sobre el eje y es una característica de los fragmentos que componen un carácter. En general, cuando un carácter queda fragmentado, el espacio inter*caracter* más común es el de tipo horizontal. Suponiendo que el código esté perfectamente alineado con el eje de lectura habrá muchas oportunidades de reagrupar esos caracteres basándonos en el solapamiento de sus fragmentos sobre el eje y.

Dada dos formas A y B cualesquiera sobre el plano con dimensiones (anchura) sobre el eje y de $y_2^A - y_1^A$ y $y_2^B - y_1^B$ respectivamente siendo $y_1^A \leq y_2^A$ y $y_1^B \leq y_2^B$ tenemos que las proyecciones en el eje y de A y B se solapan entre sí si se cumple al menos una de estas dos condiciones:

$$y_2^A \leq y_2^B \leq y_1^A \tag{7.2}$$

$$y_2^A \leq y_1^B \leq y_1^A \tag{7.3}$$

En la figura 7.49 se indican las proyecciones de diferentes fragmentos sobre el eje y (eje horizontal de la imagen). En la figura 7.52 algunos ejemplos de fragmentos agrupables. Obsérvese que no se atiende si los fragmentos son parte de un carácter o son ruido.

La relación entre la dimensión total (anchura total) de la proyección de A y la fracción de la dimensión total de A que está efectivamente solapada con B la denominamos magnitud del solapamiento de B sobre A.

La magnitud del solapamiento podemos asociarla con la fortaleza del agrupamiento.

- Solapamiento (fuerte) total de B en A (Figura 7.50).
- Solapamiento(fuerte) central de B en A (Figura 7.50)
- Solapamiento(débil) por la izquierda de A en B (Figura 7.51)
- Solapamiento(débil) por la derecha de B en A (Figura 7.51)

Propiedades

- El solapamiento de proyecciones sobre el eje y tiene carácter recíproco.
- El solapamiento de proyecciones sobre el eje y puede no ser equivalente en magnitud. Ver 7.53.

En MONICOD, la propiedad del solapamiento vertical sólo se mide entre fragmentos que estén en la misma banda.

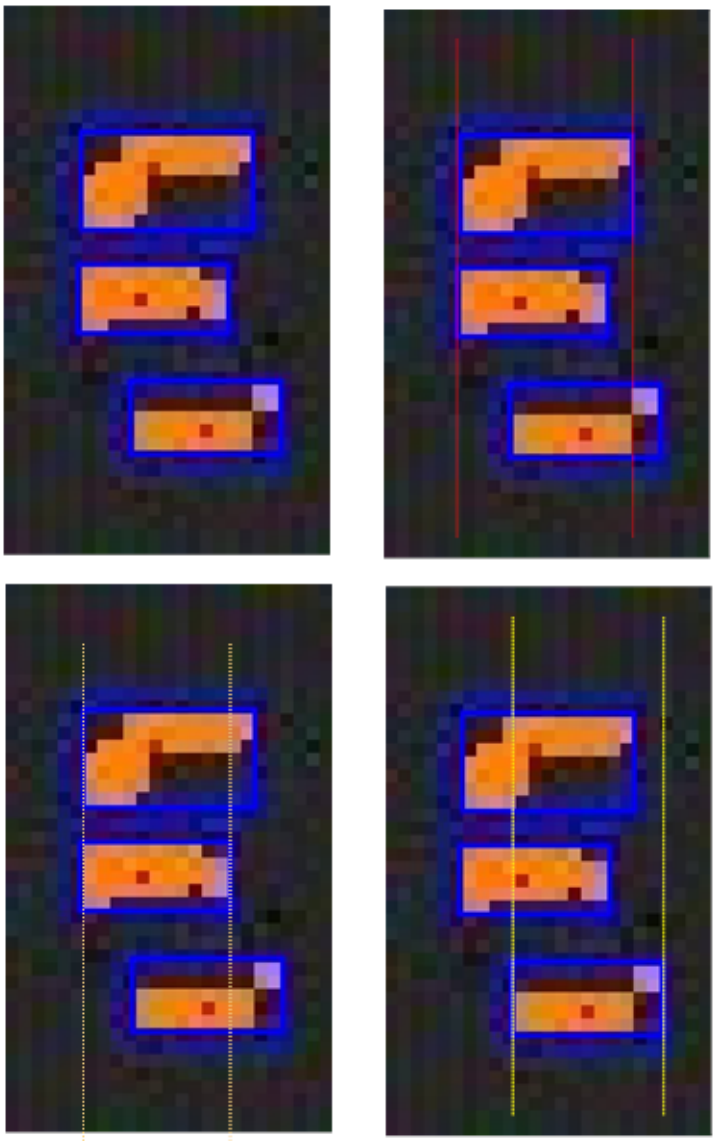


Figura 7.49: Líneas limítrofes: Arriba, izquierda carácter fragmentado. Arriba derecha, líneas limítrofes verticales del fragmento superior. Abajo izquierda, líneas limítrofes verticales del fragmento intermedio. Abajo derecha, líneas limítrofes verticales del fragmento inferior.

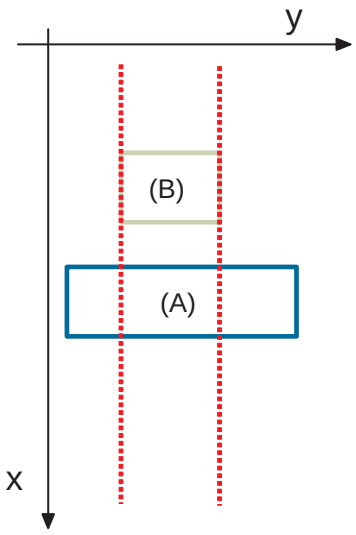


Figura 7.50: “B” está solapado completamente por “A”. “A” está solapado por el centro por “B”.

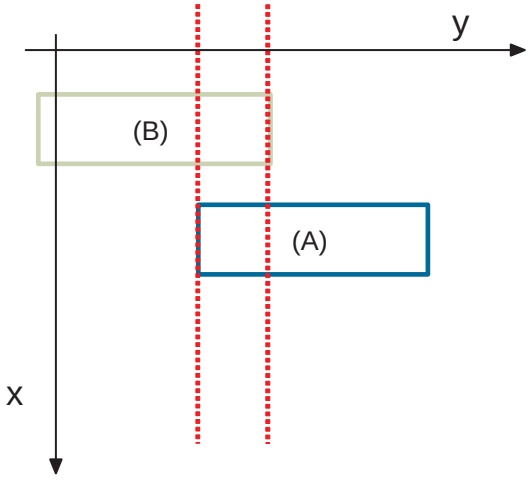


Figura 7.51: “B” solapa por la derecha con “A”. “A” solapa por la izquierda con “B”.

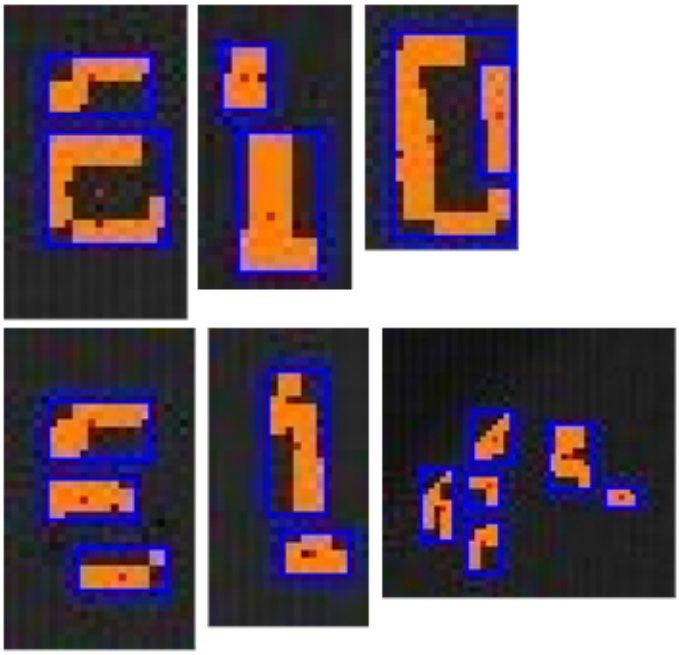


Figura 7.52: Fragmentos que son agrupables basándose en el solapamiento vertical.

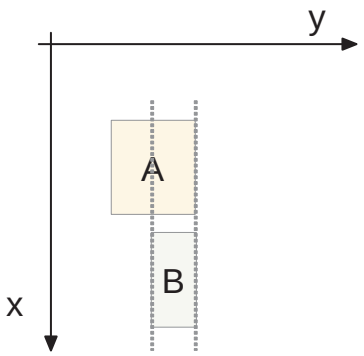


Figura 7.53: Solapamiento de magnitudes asimétricas. B solapa sobre A un 50 %. A solapa sobre B un 100 %. Cada fragmento tiene un porcentaje diferente de su dimensión total superpuesto.

7.3.5.5. Fusión

Para que se lleve a cabo una fusión debe cumplirse dos condiciones:

1. Esta debe designarse como "*legal*". Tres condiciones establecen la *legalidad*:
   a) Los fragmentos implicados forman parte de la misma banda.
   b) La fusión de los dos fragmentos (o grupos de fragmentos) NO supera el ancho máximo posible para un carácter.
   c) La fusión de los dos fragmentos (o grupos de fragmentos) NO supera la cantidad de tinta posible para un carácter.
2. Existe solapamiento vertical entre los fragmentos (o grupos de fragmentos)

La fusión puede errar de dos formas:

- **No** hay fusión cuando **si** tiene que haberla.
  - Eso sólo podría ocurrir si las condiciones de legalidad o de presencia de solapamiento vertical NO se cumplen. Una fusión correcta que no ha tenido lugar dejará huérfanos a fragmentos que no conforman por si mismos un carácter. Un indicativo de fragmento huérfano puede ser la tinta insuficiente. Esta situación indeseable es contrarrestada específicamente por el realce MONICOD descrito en 7.2.1
- **Si** hay fusión cuando **no** tiene que haberla.
  - Esa situación sólo se detectará con éxito en etapas posteriores. Un código que presente una inclinación muy fuerte es la causa más habitual para esto. La etapa de calibrado del sistema debe contrarrestar en alguna medida este problema.
  - Si se detecta esta situación se lanzará el procedimiento de fisión descrito en el apartado 7.3.5.6.

  Es necesario alcanzar un compromiso en la solución de estas dos situaciones indeseables pero opuestas. Las mismas medidas que neutralizan una situación pueden agravar a la otra.

7.3.5.6. Fisión

Una impresión que presente un trazo inusualmente grueso combinado con el procedimiento de realzado descrito en 7.2.1 puede suprimir los espacios *intercaracter* provocando que caracteres diferentes (o partes de este) terminen unidos en masas de tinta compactas. Estas masas derivarán durante el proceso de la asociación de tinta ya descrito en 'superfragmentos'

Por lo anterior tenemos dos circunstancias problemáticas:

1. Físicamente existe tinta que forma un puente entre los caracteres. Este problema es complicado de tratar porque responde a una anomalía física indeseable durante la impresión. Sin embargo bien podría el código ser legible a pesar del defecto. Declararlo código no válido puede conducir a un exceso de celo.

2. No existe tinta entre los caracteres, pero el engrosamiento (preproceso + umbral de inundación) hace que los caracteres se fusionen. Ese problema puede ser abordado conservando el campo de entrada posterior al proceso, donde la separación aún existía. Con esa información previa podemos reasignar los puntos del conjunto a dos conjuntos distintos, superando la anomalía (fisión).

El primer paso es detectar la presencia de estos 'superfragmentos'. Tenemos dos indicadores.

1. Anchura excesiva en un fragmento.

2. Cantidad de tinta excesiva en un fragmento.

Una vez detectada la situación aplicaremos el procedimiento de fisión. Este consiste simplemente en repetir los procedimientos previos (preproceso y asociación de tinta) sólo a las coordenadas de los superfragmentos, pero ignorando el realzado. Con ello suprimimos de raíz una de las dos causas para la aparición de superfragmentos.

1. Dada la imagen original y la tabla LUT de ecualizado (tabla de substitución de grises) que tenemos (no ha sido destruida) ecualizaremos las coordenadas del superfragmento.

2. Asociaremos tinta sólo en las coordenadas del superfragmento.

3. NO es necesario asignar banda alguna. Todos los fragmentos heredan la banda asignada del superfragmento.

4. El superfragmento original ha desaparecido y en su lugar debemos tener fragmentos comunes. Si aún así aparecen superfragmentos los caracteres estaban unidos originalmente, y estaríamos ante el caso de impresión defectuosa ya comentado.

7.3.5.7. El ciclo de agrupamiento

El ciclo del agrupamiento es un proceso iterativo que agrupa los fragmentos de una forma ordenada. Es el corazón del agrupamiento de fragmentos.

El ciclo cruza todos fragmentos no agrupados (y grupos de fragmentos ya agrupados) calculando las superposiciones de proyección sobre el eje y 'legales'. Los agrupamientos 'ilegales' (condiciones definidas en el apartado 7.3.5.5) se les asigna un solapamiento nulo.

Con la información obtenida se llevan a cabo tentativas de agrupamiento entre fragmentos no agrupados (y grupos de fragmentos ya agrupados) siguiendo un orden estricto basado en las magnitudes de solapamiento. Un fragmento que ya ha sido agrupado en el ciclo actual no puede ser de nuevo considerado para agrupar. Si ya no son posibles más agrupamientos, se recalcula la tabla de solapamientos y se repiten las tentativas de agrupamiento. Si en un ciclo todas las tentativas de agrupamiento fracasan se da por finalizado el proceso iterativo.

1. Se inicializa la tabla de solapamientos donde cada fila y cada columna corresponde a un fragmento dentro de la misma línea.
2. Se calcula la tabla de solapamientos. Ver figura 7.54 y algoritmo 7.13.
3. Se elige un par (i, j) que ofrezca el solapamiento máximo de la tabla (ver algoritmo 7.15) y donde ni i ni j hayan sido agrupados con éxito en el actual ciclo. Si no existe ninguno se pasa al paso 5.
4. Con el par (i, j) se hace una tentativa de agrupamiento.
   a) Si tiene éxito el par (i, j) es agrupado con éxito. Ahora forma un grupo de fragmentos y no serán considerados individualmente. Ni i ni j son elegibles para nuevos agrupamientos en este ciclo. Se retorna al paso 3.
   b) Si fracasa, se retorna al paso 3.
5. Si al menos una tentativa de agrupamiento ha tenido éxito se retorna al paso 2. Si no es el caso, se da por finalizado el ciclo de agrupamiento.

7.3.5.8. Un ejemplo

Si examinamos la muestra de agrupamientos exitosos de la figura 7.55 apreciamos que casi todos los casos consisten en dos agrupamientos salvo algunas excepciones de tres agrupamientos. También podemos observar que casi siempre hay solapamiento vertical, que a la postre, es clave para el mecanismo de agrupamiento. Algunos caracteres son más proclives que otros a fragmentarse, y sus puntos de fragmentación casi siempre son los mismos.

Un caso particularmente problemático es la barra oblicua de derecha a izquierda ("/"). Al fragmentarse no presenta solapamiento vertical y su delgadez propicia una fragmentación de más de dos fragmentos.

7.3.6. El carácter adquirido

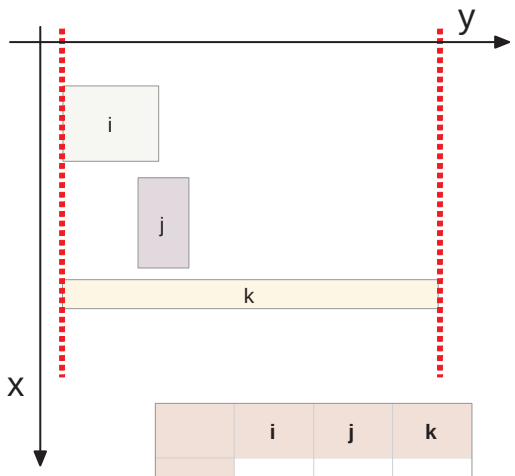
El carácter adquirido incluye:

- Una lista de fragmentos. El fragmento fue descrito en 7.3.3.5. La lista de fragmentos es el producto del procedimiento descrito en 7.3.5.

Algoritmo 7.13 Calculo de solapamiento

Precondición: i, j son los índices de los fragmentos o grupos de fragmentos (que estan en la misma línea) sobre los que queremos calcular el agrupamiento.
Postcondición: El porcentaje del solapamiento en el rango (0-100)

```
si NO hay solapamiento por la izquierda Y NO hay solapamiento por la derecha entonces
    retornar 0
fin si
si hay solapamiento por la izquierda Y hay solapamiento por la derecha entonces
    si la fusion de los fragmentos es legal entonces
        retornar 100
    sino
        retornar 0
    fin si
fin si
si hay solapamiento por la izquierda entonces
    si la fusion de los fragmentos es legal entonces
        retornar puntossolapados * 100/puntostotalesRegionI
    sino
        retornar 0
    fin si
sino
    si la fusion de los fragmentos es legal entonces
        retornar puntossolapados * 100/puntostotalesRegionI
    sino
        retornar 0
    fin si
fin si
retornar 0
```



	i	j	k
i	-1	25	100
j	50	-1	100
k	100	100	-1

Figura 7.54: Ejemplo de tabla de solapamientos para tres fragmentos en una misma banda donde cualquiera de sus agrupamientos es legal.

Algoritmo 7.14 Tabla de solapamientos

Precondición: Lista de fragmentos y grupos de fragmentos que conformarán las filas y columnas de la tabla de solapamiento.

Postcondición: $solapamientos[][]$ contiene todas las magnitudes de solapamiento vertical de los cruces entre fragmentos y grupos de fragmentos.

para todo i tal que i sea un fragmento libre **hacer**
  para todo j tal que j sea un fragmento libre **hacer**
    si i es distinto a j **entonces**
      si i está en la misma banda que j **entonces**
        $solapamientos[i][j] \leftarrow CalcularSolapamiento(i, j)$
      sino
        $solapamientos[i][j] \leftarrow 0$
      fin si
    fin si
  fin para
fin para

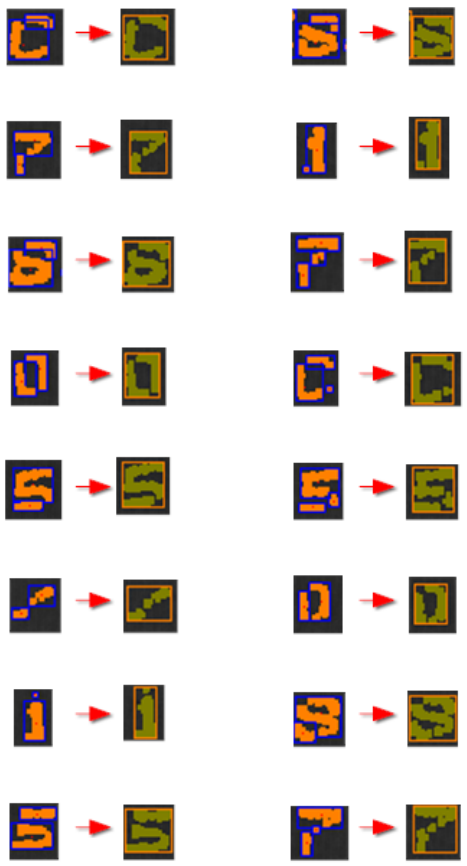


Figura 7.55: Ejemplos de agrupamiento extraídos del experimento MONICOD. El sistema de agrupamiento es capaz de enfrentar con éxito los casos más habituales. Especialmente si el código no presenta pendiente.

Algoritmo 7.15 Búsqueda de un máximo en la tabla de solapamientos

Precondición: $solapamientos[][]$ es la tabla de solapamientos
Postcondición: $maximo, zonacandidataafusion_A, zonacandidataafusion_B$ contienen el par o cruce de magnitud de solapamiento máximo en la tabla de solapamientos.

```
maximo ← 0
zonacandidataafusion_A ← −1
zonacandidataafusion_B ← −1
para todo i tal que i sea un fragmento libre hacer
    para todo j tal que j sea un fragmento libre hacer
        si maximo es menor que solapamientos[i][j] entonces
            maximo ← solapamientos[i][j]
            zonacandidataafusion_A ← i
            zonacandidataafusion_B ← j
        fin si
    fin para
fin para
```

- Área del carácter. En realidad el área de carácter es sólo el área que ocupa la tinta de ese carácter en la imagen medida en píxeles. No se consideran espacios interiores (Ver 6.3.2), y otras áreas de NO tinta que ayudan a caracterizar el carácter en el cómputo del área. El cómputo del área consiste en la suma de las áreas de los fragmentos que constituyen la lista de fragmentos.
- Centroide del carácter.
- La banda a la que pertenece el carácter. Por las restricciones descritas en el apartado 7.3.5.3, un carácter sólo puede formar parte de una única banda.
- Coordenada X (fila) mínima. Esto es, el valor más bajo de la coordenada X entre los fragmentos que constituyen la lista de fragmentos.
- Coordenada X (fila) máxima. Esto es, el valor más alto de la coordenada X entre los fragmentos que constituyen la lista de fragmentos.
- Coordenada Y (columna) mínima. Esto es, el valor más bajo de la coordenada Y entre los fragmentos que constituyen la lista de fragmentos.
- Coordenada Y (columna) máxima. Esto es, el valor más alto de la coordenada Y entre los fragmentos que constituyen la lista de fragmentos.

Las siguientes propiedades son computables directamente desde lo anterior.

- Altura. Medida que se deriva de forma inmediata de la diferencia entre la coordenada X (fila máxima) y la coordenada X (fila mínima).

- Anchura. Medida que se deriva de forma inmediata de la diferencia entre la coordenada Y (columna máxima) y la coordenada Y (columna mínima).

7.3.7. Ordenación

La interpretación de cada carácter dentro del código depende de su posición dentro del código (ver 3.8.2). Por convenio MONICOD ordena los caracteres adquiridos para una lectura de izquierda a derecha, de arriba a abajo. Esto se hace en dos fases.

1. Los caracteres se agrupan por sus bandas. Sus bandas son ordenadas de arriba a abajo.

2. Los caracteres de cada banda se ordenan por su coordenada Y mínima (mencionada en la subsección 7.3.6).

Para el problema de ordenación existe una importante bibliografía. Ver por ejemplo el estupendo [116]. En MONICOD se utiliza la ordenación proporcionada por la librería STL[19]. El algoritmo de ordenación depende de la implementación concreta de la librería. En nuestra implementación se utiliza el algoritmo introspectivo o algoritmo *introsort* que combina el *quicksort* para los casos promedio y mejor y el *heapsort* para el peor caso.

Se considera un buen comportamiento para un algoritmo de ordenación una complejidad de orden $O(nlog(n))$ siendo un mal comportamiento $O(n^2)$. El comportamiento ideal es $O(n)$ (porque es irreductible la necesidad de recorrer la lista de valores a ordenar) pero es difícil conseguirlo en el caso promedio. Los algoritmos de ordenación basados en comparación, esto es, que emplean una operación de comparación en su evaluación, necesitan al menos una complejidad de $O(nlog(n))$.

La complejidad computacional en la ordenación es un factor crítico cuando el número de elementos a ordenar es considerable (cientos, miles, millones). En muchas aplicaciones reales importantes esto es un escenario común, de ahí el gran número de algoritmos de ordenación existentes en el área.

En 7.5 se comparan algunos algoritmos de ordenación según la complejidad computacional. No se considera el consumo de otros recursos como la memoria en la actividad de ordenación, que pueden ser relevantes en otras aplicaciones.

Dentro del dominio específico de MONICOD, esta instancia concreta del problema de ordenación opera sobre una breve colección de claves que son enteros pequeños. En este escenario es factible emplear algoritmos de ordenación no basada en comparación de complejidad lineal, próxima a $O(n)$. Este es el caso del algoritmo CountingSort. Sin embargo no ha resultado necesario mejorar esta etapa frente a otras que si lo demandaba.

[19] Standard Template Library

Algoritmo	Mejor caso	Caso promedio	Peor caso
IntroSort	$n\log(n)$	$n\log(n)$	$n\log(n)$
QuickSort	$n\log(n)$	$n\log(n)$	n^2
HeapSort	$n\log(n)$	$n\log(n)$	$n\log(n)$
MergeSort	$nlog(n)$	$nlog(n)$	$nlog(n)$
TimSort	n	$nlog(n)$	$nlog(n)$
CountingSort	$n+r$	$n+r$	$n+r$

Cuadro 7.5: Una breve comparativa en términos de eficiencia de algoritmos de ordenación. n: longitud del vector de ordenación. r es la longitud del rango de valores en el el vector de ordenación.

7.3.8. Código de la fuente

El código real es la estructura soporte de los caracteres adquiridos. Ver figura 7.56. Aunque a lo largo de este capítulo hemos hablado de extracción de caracteres, en realidad no sabemos aún si las formas extraídas son efectivamente caracteres de código, o siquiera caracteres en algún lenguaje. Son un producto del resultado de la segmentación, donde cabe esperar que un subconjunto de lo obtenido sera ruido de impresión o adquisición.

Estas formas esperan a ser validadas discerniendo sin son o no las formas esperadas en las posiciones correctas. Por eso esta es la información de entrada para la etapa descrita en el capítulo 8.

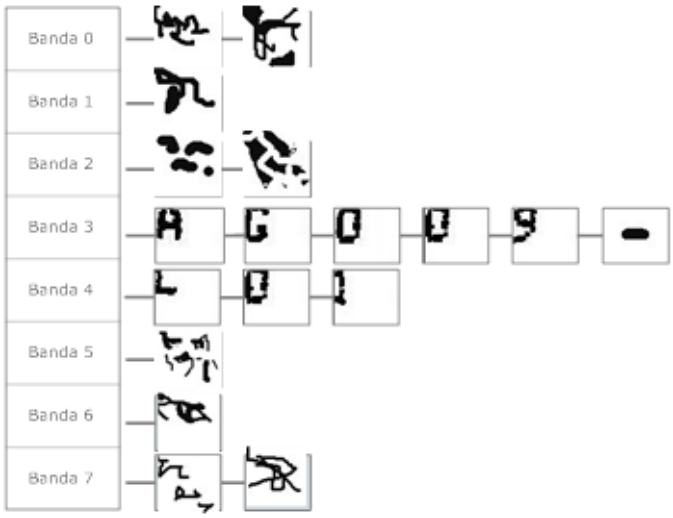


Figura 7.56: Código extraído de la fuente.

Capítulo 8

MONICOD: Validación y aprendizaje

"Ahora toca abrir los ojos y fijarse en detalles nimios"

MONICOD es un sistema de validación de caracteres. Sin embargo no es hasta este momento que se lleva una auténtica "validación de caracteres". Es bastante posible que una lata incorrecta se descarte en alguna de las etapas anteriores. Por ejemplo ante ausencia de código sin que ningún carácter se haya validado. Pero si se lleva a este punto hay que responder a la cuestión de si son estos o no los caracteres esperados.

Hay dos situaciones para encontrar un "carácter no esperado": O bien el carácter no es legible (por ejemplo el carácter está borroso, o es ruido) o bien, es un carácter legible, pero no es correcto. Esto es; el código de caducidad esperado tiene un carácter diferente para esa posición. Como veremos, MONICOD no distingue entre ambas situaciones.

La primera parte de capítulo versará sobre esta "validación de caracteres". Por otra parte la validación de caracteres está estrechamente relacionada con el aprendizaje. El aprendizaje es el procedimiento asociado a la inyección de conocimiento (esto es de formas de reconocibles) en MONICOD. Su exposición comprende la segunda parte del capítulo.

8.1. Validación de caracteres

8.1.1. La NO clasificación: Validar NO es reconocer

MONICOD verifica códigos de caducidad en latas. La verificación consiste en comparar el código (segmentado) hallado en la imagen con el código esperado generado internamente por MONICOD (que se supone siempre correcto).

En el caso de un reconocimiento el procedimiento sería:

1. Reconocer el texto

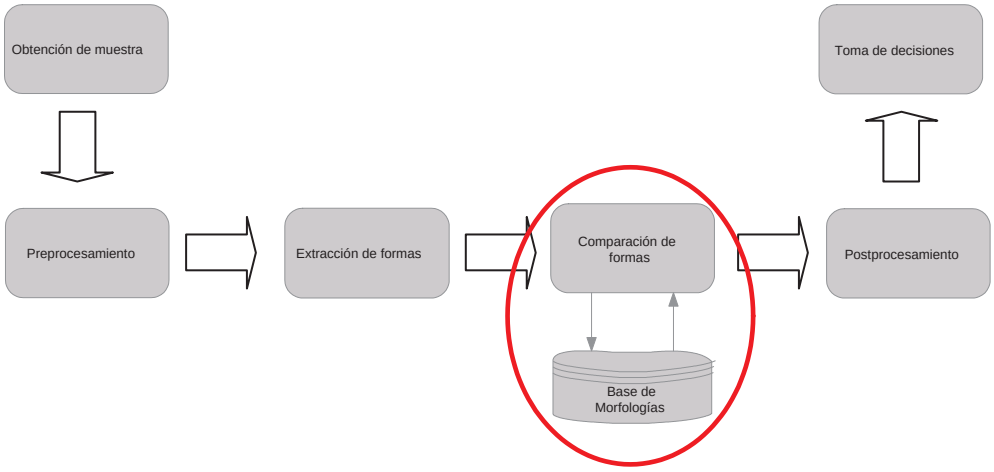


Figura 8.1: Validación por comparación

2. Comparar el texto reconocido con el texto esperado.
3. Dar un veredicto a partir de la comparación.

MONICOD no lleva a cabo un reconocimiento cuando valida el carácter en el sentido de que no tratamos de determinar a qué clase corresponde sino si corresponde o no a una clase esperada en particular. Esto es así en aras de la eficiencia. En la validación el procedimiento sería:

1. **Comparar** las formas extraídas con las formas esperadas. Ver figura 8.1.
2. Dar un veredicto a partir de la comparación.

8.1.2. Lo que tenemos hasta ahora: Caracteres adquiridos

Las etapas anteriores nos proporcionan un conjunto de candidatos a caracteres a los que denominamos de forma genérica como "formas". Estos caracteres están ordenados por bandas (de arriba a abajo) y de izquierda a derecha dentro de una banda tal como vemos en el apartado 7.3.8. Una forma puede ser o no un carácter válido pero todo carácter válido es una forma.

8.1.3. Código esperado

El código esperado es actualizado cada vez que se efectúa la validación de un código. La actualización del código esperado es comentada con cierto detalle en 3.8.2.

La semántica del código carece de interés para la validación salvo por dos consideraciones:

- Necesitamos saber cómo generar el código esperado.
- El significado del carácter afecta a su importancia y la importancia es el factor que determina cuando su validación es irrenunciable.

8.1.4. La base de familias morfológicas de caracteres(BFM)

La base de familias morfológicas(BFM) contiene los caracteres aprendidos por MONICOD y gestiona recuperaciones, adiciones y supresiones. La BFM puede ser reiniciada, esto es eliminar todos los elementos, manualmente[1]. La base de familias morfológicas se mantiene, se consulta y se modifica en memoria. De esta forma no accedemos al disco para rescatar o actualizar datos[2].

La BFM opera con dos conceptos clave:

- Morfologías
- Familias Morfológicas

La base de familias morfológicas posibilita el aprendizaje y la validación de caracteres. Y si procede, también el reconocimiento, aunque MONICOD no es un sistema de reconocimiento[3]. Ver figura 8.8.

8.1.4.1. Morfologías

MONICOD basa su validación en la forma de los caracteres. Para ello:

- Hace una extracción de características básica de la forma.
- Compara la propia forma discretizada con formas discretizadas de carácter que se almacenan y recuperan integras en la BFM de caracteres. MONICOD se refiere a estas formas contenidas en la BFM de caracteres como *'morfologías'*.

La morfología es la unidad mínima de la base de familias morfológicas de caracteres. Su estructura es simple. Se trata de una representación discreta de de ceros y unos sin compresión de dimensión $m \times n$. Por convención 'cero'(0) representa NO-TINTA y 'uno'(1) TINTA. Las posiciones de ceros y unos (TINTA y NO-TINTA) mantienen la misma relación que en la forma que representan. Así la morfología es una representación explícita de formas a tamaño real binarizadas. Puede verse como un mapa de bits o celdas. Ver figura 8.2.

[1] Naturalmente resetear la base de familias morfológicas mientras se desarrolla un proceso de validación generará falsas validaciones negativas.

[2] Acceder a disco es siempre altamente contraproducente para la velocidad.

[3] Se emplea la BFM en un sistema de reconocimiento en el experimento descrito en la sección 9.7 del capítulo 9.

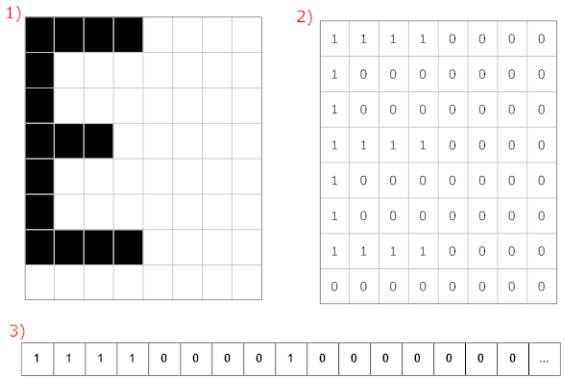


Figura 8.2: La representación de morfología en tres partes. En 1) tenemos la forma binarizada. En 2) una matriz de ceros y unos que corresponde con 1). En 3), la matriz 2) dispuesta como un vector que es como trabaja MONICOD internamente. Esta última representación es denominada plantilla.

¿Por qué utilizar morfologías?

1. Si las formas son legibles y reconocibles las morfologías asociadas retendrán esas cualidades.

2. Las morfologías se extraen directamente de la segmentación sin que haya ningún análisis que involucre nuevas operaciones como la creación de un vector de características.

3. La impresión industrial estandariza. Es de esperar que los caracteres impresos sean parecidos entre latas y se puedan asociar a una misma morfología.

8.1.4.2. Distancia entre dos morfologías

La acción más importante que se puede hacer sobre una morfología es compararla con otra morfología y evaluar cuanta distancia o similitud existe entre ambas. En la tabla 8.1 se muestran algunos ejemplos de distancias.

MONICOD define la distancia entre morfologías a partir del simple recuento de coincidencias y no coincidencias en tinta y fondo entre las celdas de dos morfologías A y B. Las propiedades de distancia descritas en A.3.2.2 también son válidas aquí. Para que dos morfologías sean comparables deben tener el mismo alto y ancho (m y n).

Se utilizan cuatro contadores independientes. Comparando A y B:

- Coincidencias de Tinta (CT): Cuenta coincidencias de tinta entre A y B

- Coincidencias de No Tinta (CNT): Cuenta coincidencias de fondo entre A y B

Distancia	Expresión de distancia al comparar dos expresiones binarias
Distancia de Hamming	$\|A \oplus B\|$
Producto escalar	$\|A \cdot B\|$
Distancia de Jaccard	$1 - J(A,B)$ donde $J(A,B) = \frac{\|A \cdot B\|}{\|A+B\|}$
Índice de Sørensen-Dice	$1 - D(A,B)$ donde $D = \frac{2\|A \cdot B\|}{\|A\|^2+\|B\|^2}$
Índice de Tversky	$T(A,B) = \frac{\|A \cdot B\|}{\|A \cdot B\|+\alpha\|A-B\|+\beta\|B-A\|}$ con $\alpha, \beta \geq 0$ y $\alpha + \beta = 1$
MONICOD	$\frac{1}{2}\left[\frac{\|E \cdot R\|}{\|E\|} + \frac{\overline{\|E+R\|}}{\overline{\|E\|}}\right]$

Cuadro 8.1: Distancias entre plantillas binarias, expresadas en términos de operadores lógicos, donde $|X|$ es el número de unos en el vector X. En el caso de MONICOD $|E|$ es la plantilla esperada y $|R|$ es la plantilla real obtenida en la extracción de características.

- Tinta Ausente (TA): Cuenta no-coincidencias debido a tinta ausente en B que está presente en A.
- Tinta Inesperada (TI): Cuenta no-coincidencias debido a tinta presente en B que no está presente en A.

El recuento es descrito en 8.2. Ver nota al pie[4].

Si intercambiamos los operandos A y B a B y A se intercambian las magnitudes de los contadores TA y TI mientras que permanecen igual CNT y CT.

Siempre se cumple que:

$$CNT + TA + TI + CT = m \times n \tag{8.1}$$

donde m y n son el alto y el ancho de las morfologías que se comparan.

Por otro lado, en nuestra comparación A y B:

$$CT + TA = \text{Cantidad de tinta en A} \tag{8.2}$$

$$CNT + TI = \text{Cantidad de no tinta o fondo en A} \tag{8.3}$$

Los valores de los contadores permite una sencilla evaluación de similitud. En la implementación actual de MONICOD:

[4]Los términos "ausente" e "inesperada" se debe a que el cálculo de distancias ocurre entre morfologías adquiridas por el sistema (A) y morfologías almacenadas (B) en la BFM. Se espera que la morfología almacenada es "correcta" o esperada y la adquirida debe ser validada con respecto a esta. Así se puede hablar de tinta ausente y tinta inesperada.

Plantilla A	Plantilla B	Consecuencia
NO HAY TINTA	NO HAY TINTA	CNT= CNT+1
NO HAY TINTA	HAY TINTA	TA= TA+1
HAY TINTA	NO HAY TINTA	TI = TI+1
HAY TINTA	HAY TINTA	CT= CT+1

Cuadro 8.2: Tabla de evaluación de coincidencias y no coincidencias entre dos plantillas o morfologías A y B. La primera plantilla

$$\phi = f(\text{CNT}, \text{TA}, \text{TI}, \text{CT})) = \frac{1}{2} \times \left[\frac{CT}{CT + TA} + \frac{CNT}{CNT + TI} \right] =$$
$$= \frac{1}{2} \times \left[\frac{CT}{\text{Cantidad de tinta en A}} + \frac{CNT}{\text{Cantidad de no tinta o fondo en A}} \right] \quad (8.4)$$

Así la distancia entre dos muestras viene dada por:

$$d(A, B) = 1 - \phi \quad (8.5)$$

Metodología El calculo de comparación comprende dos sub-fases. La primera se limita a llevar a cabo un recuento de similitudes y diferencias, celda a celda, entre plantilla almacenada y plantilla adquirida. No se considera la posición dentro de la plantilla. Ver cuadro 8.2. La segunda computa una función, parametrizada por los valores de recuento de la primera fase, que nos proporciona una medida de similitud.

El algoritmo de comparación de plantillas

1. Inicializar a 0 los contadores COINCIDENCIA_DE_NO_TINTA(CNT), TINTA_AUSENTE(TA), TINTA_INESPERADA(TI) y COINCIDENCIA_TINTA(CT).
2. Para cada celda de la plantilla A|B

Algoritmo 8.1 Algoritmo de comparación. La implementación llevada a cabo utiliza comparación binaria.

Precondición: *morfologia* extraida de la familia de morfologías para ese carácter contenida en la pase de conocimiento
Precondición: *plantilla* extraida de la segmentación de la fuente. La fuente es la imagen adquirida.
Postcondición: Devuelve una medida de la similitud entre plantilla y morfología.

```
TA ← 0
TI ← 0
CT ← 0
CNT ← 0
para i = 0 a tamanodeplantilla hacer
    si morfologia[i] es igual a 1 entonces
        si plantilla[i] es igual a 1 entonces
            CT ← CT + 1
        sino
            TA ← TA + 1
        fin si
    sino
        si plantilla[i] es igual a 0 entonces
            CNT ← CNT + 1
        sino
            TI ← TI + 1
        fin si
    fin si
fin para
```

a) Si no hay tinta en plantilla A(0) y no hay tinta plantilla B(0) se incrementa en uno COINCIDENCIA_DE_NO_TINTA(CNT)

b) Si no hay tinta en plantilla A(0) pero si hay tinta en plantilla B(1) se incrementa en uno TINTA_AUSENTE(TA).

c) Si hay tinta en plantilla A(1) pero no hay tinta en plantilla B(0) se incrementa en uno TINTA_INESPERADA(TI).

d) Si hay tinta en plantilla A(1) y hay tinta en plantilla B(1) se incrementa en uno COINCIDENCIA_TINTA(CT).

3. Se computa la función de similitud. Ver expresión 8.4.

Ver el algoritmo en 8.1.

8.1.4.3. Consideraciones sobre la distancia de plantillas descrita.

Sus ventajas resultan adecuadas para MONICOD.

1. Es intuitiva.

   Es una comparación sistemática que evalúa el número de posiciones de tinta y no tinta coincidentes entre dos morfologías.

2. Es simple.

   La única operación involucrada es la comparación booleana y se puede acelerar aún más con la ayuda de operaciones bit a bit (AND)

3. Es extremadamente rápida.

   La comparación de plantillas implica cálculo intensivo[5], pero fácilmente paralelizable. El máximo número de operaciones involucradas es fijo. En un caso de validación (como éste) sin reconocimiento no necesitamos llegar al final de una comparación. La comparación permite obtener un resultado parcial a medida que progresa. Superado un umbral de no coincidencias podemos detener la comparación, y estipular que ha fallado, siempre y cuando no se requiera conocer la magnitud de la diferencia[6].

4. Es apta para este dominio.

   La propia dinámica repetitiva de la impresión de latas sugiere el mecanismo de comparación de plantillas: La fuente del texto, la escala, la rotación del texto... son propiedades fijas que no deben variar salvo por la presencia de anomalías,

Sin embargo, no es el método predilecto para reconocimiento de carácter porque:

1. Es muy sensible al ruido.

2. No atiende a la importancia relativa de donde ocurren coincidencias y discrepancias, sólo a su magnitud(el número de veces que ocurre). En cambio un carácter se caracteriza por donde exactamente se localiza tinta y fondo antes que por la cantidad presente.

   Ver sobre esto el ejemplo de la figura 8.3 y cuadro 8.3. La distancia obtenida de A frente a las 8 plantillas B es la misma: $d(A,B) = 1 - \phi = 1 - \frac{1}{2} \times \left[\frac{CT}{CT+TA} + \frac{CNT}{CNT+TI}\right] = 1 - \frac{1}{2} \times \left[\frac{0}{0+1} + \frac{7}{7+1}\right] = 1 - \frac{7}{8} = \frac{1}{8} \approx 0{,}125$

3. Escasa capacidad de generalización.

   a) No soporta las invarianzas[7] comunes. Así es vulnerable al escalado, rotación y traslación, entre otras. Ver figura 8.4.

   b) No ofrece soporte para la plasticidad natural de los caracteres (un caso particular de transformación). Este inconveniente puede apreciarse como caso particular del inconveniente anterior[8].

[5] Aunque existen técnicas programáticas que aceleran enormemente la comparación.
[6] Esta optimización no ha sido implementada porque se ha juzgado suficientemente rápida.
[7] Un invariante es algo que no cambia bajo un conjunto de transformaciones. La propiedad

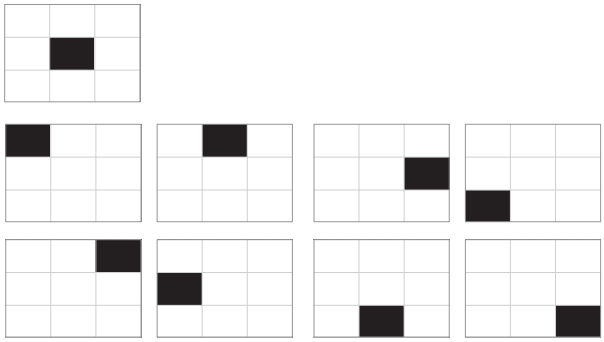


Figura 8.3: A contra B. Plantillas 3×3. Para MONICOD la primera plantilla(A) esta a la misma distancia de las otras ocho(B).

Contador	Recuento
CNT	7
TA	1
TI	1
CT	0

Cuadro 8.3: Recuento de A contra B de la figura 8.3: El valor de los contadores es igual para las 8 plantillas.

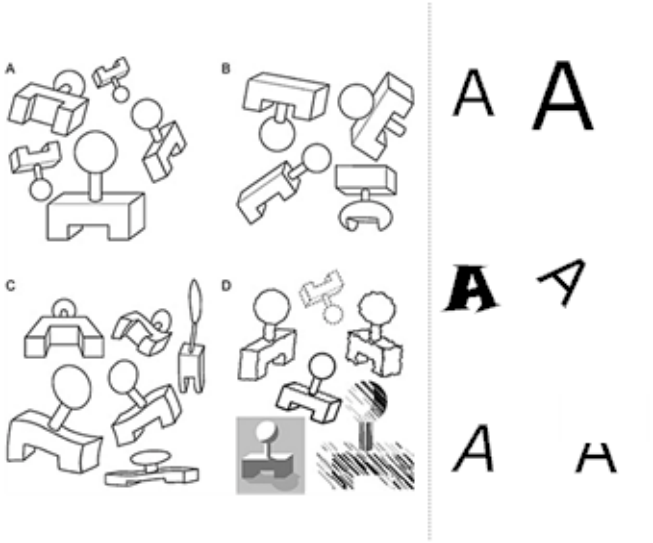


Figura 8.4: Invarianzas en un objeto en 3D y en un objeto en 2D.

8.1.4.4. Familias morfológicas

¿Por qué? En una situación ideal bastaría una única forma para identificar un carácter porque estaría asociada a un único carácter de modo inequívoco[9]. Sería suficiente comparar la forma adquirida con una forma generada artificialmente decidida por el generador de código automático y dirimir si hay coincidencia o no. Esta aproximación parece tener sentido si nos atenemos a que se está imprimiendo siempre con la misma máquina sobre superficies del mismo tipo. Los caracteres que componen los códigos de impresión deberían ser facsímiles unos de otros. Sin embargo en la práctica se comprueba que esto no ocurre(ver por ejemplo la figura 8.6).

Así pues debemos considerar tres etapas de distorsión. Ver figura 8.5

- Impresión

  Las formas impresas de un mismo carácter varían de lata a lata, incluso en la misma lata. Las razones de esta falta de consistencia son expuestas en 4.3 y 4.4. Las contramedidas específicas para este problema quedan fuera del ámbito de MONICOD.

- Adquisición

  En condiciones de impresión idéntica las formas adquiridas de un mismo carácter varían de lata a lata, incluso en la misma lata[10] por razones expuestas en los capítulos 3 y 4. MONICOD, con la inclusión de una cámara oscura y un protocolo de calibrado trata de aliviar en lo posible algunos de estos problemas de la adquisición. Sin embargo estas contramedidas se han revelado insuficientes para evitar que la forma de un carácter adquirido difiera de su forma estándar.

- Segmentación

  En condiciones de impresión idéntica y de adquisición idéntica de los caracteres - y del fondo- , las formas segmentadas de un mismo carácter NO PUEDEN variar de lata a lata, o dentro de la misma lata. Esto debe ser así porque MONICOD es un algoritmo determinista. Por tanto esta etapa no debe introducir ninguna distorsión.

La familia morfológica La familia morfológica es un agrupamiento lógico de morfologías que responde al hecho de cada carácter está asociado a un conjunto

---

de ser invariante es la invarianza.

[8] De hecho, en un entorno de caracteres manuscritos o con presencia de estilos este método estaría fuertemente contraindicado. También en capturas de texto creadas a través de un medio analógico (como un escáner) es común que diferentes formas sean sinónimas del mismo carácter.

[9] Es evidente que no ocurre así en el texto manuscrito, que está lejos de ser "ideal".

[10] Principalmente por problemas derivados de la iluminación, localización del código dentro de lata y de esta dentro de la imagen.

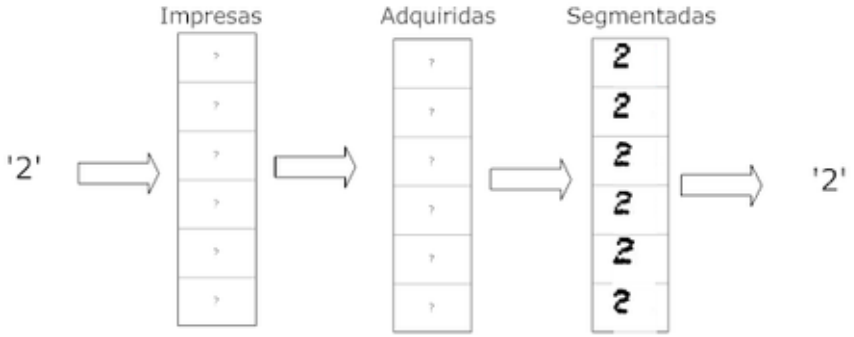


Figura 8.5: La necesidad de familias morfológicas. En la figura un "2" resultará en versiones impresas o adquiridas diferentes que derivan en diferentes formas segmentadas. Todas esas formas comprenden la familia morfológica del "2"..

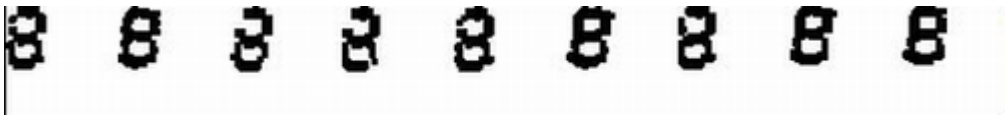


Figura 8.6: Todos estos ochos son legibles y por tanto aceptables. Sin embargo son diferentes entre sí.

o colección de formas igualmente válidas. Véase a este respecto la figura 8.6. En 8.7 tenemos un agrupamiento lógico de formas como respuesta a esta realidad.

Propiedades

- No repetición: No existen nunca dos morfologías idénticas dentro de una familia morfológica.
- Parentesco: Toda morfología guarda más similitud con cualquiera de las morfologías de su propia familia que con cualquier otra morfología perteneciente a otra familia.

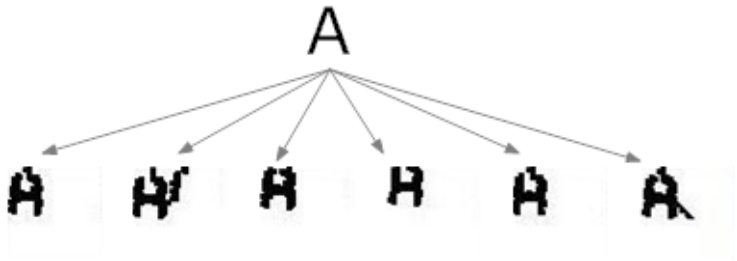


Figura 8.7: Familia morfológica para la 'A'

8.1.4.5. Operaciones sobre la BFM

La BFM (Ver figura 8.8) almacena las morfologías clasificadas en familias morfológicas. Varias operaciones sobre la BFM son posibles tanto en Tiempo de Validación como en Tiempo de Aprendizaje.

- Adición de una nueva morfología
- Supresión de una morfología
- Consulta sucesiva de las morfologías de una determinada familia morfológica.

8.1.5. La metodología de verificación

Hay dos etapas en la validación.

La primera etapa se reduce a elegir qué carácter esperado compararemos con qué carácter adquirido (ver 7.3.8). Será examinado en detalle en 8.1.5.1.

La segunda etapa consiste en comparar el carácter adquirido y el carácter esperado. A esto llamamos verificación. Será examinado en detalle en 8.1.5.2.

8.1.5.1. Seleccionando carácter adquirido y carácter esperado

La política de selección exige que los caracteres buscados se encuentren en la lata. No se examina el siguiente carácter esperado hasta que el actual haya sido encontrado. Por otra parte, si el carácter adquirido no corresponde con el esperado, se asumirá que el carácter adquirido es ruido y se avanzará al siguiente carácter adquirido (pero se mantiene fijo el carácter esperado).

Orden de comparación La comparación se lleva a cabo carácter a carácter. Dicha comparación mantiene el orden de lectura occidental de los caracteres en el código a validar. De izquierda a derecha. De arriba a abajo.

En propiedad resulta indiferente el orden de comparación de morfologías esperadas y obtenidas siempre que sepamos la posición de lectura (dentro del código) sobre la que estamos leyendo (en los códigos NO existen palabras, ni hay contexto) pero obrar de esta forma comporta ventajas:

- Al leer de izquierda a derecha tenemos información útil para refinar los agrupamientos de la fase previa: Siempre dejamos atrás caracteres completos de manera que, si el agrupamiento actual no corresponde a un carácter la única alternativa es contemplar fusiones con el agrupamiento de la derecha.
- En los códigos de caducidad parece que la importancia de los caracteres decrece de izquierda a derecha y de arriba a abajo. Es decir, dentro de una línea, es más probable que un carácter tenga una importancia igual o mayor que el carácter inmediato a su derecha. Igual ocurre entre líneas.

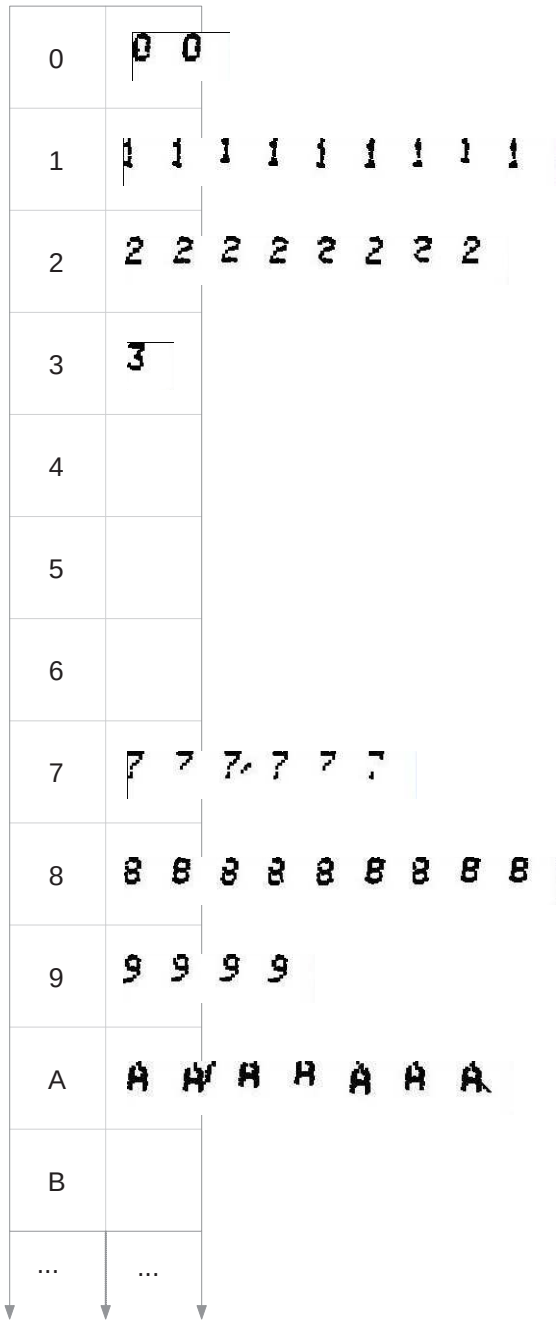


Figura 8.8: Base de familias morfológicas de caracteres.

> Es más probable que una linea tenga una importancia mayor o igual que la inmediatamente inferior. Si el código va a ser validado negativamente es mejor saberlo cuanto antes y dando prioridad a los caracteres importantes se avanza en ese propósito[11].

Metodología La metodología de validación y aprendizaje tiene como núcleo un ciclo de comparaciones.

El primer paso es seleccionar la primera línea del código esperado aún no procesado siguiendo el orden de arriba a abajo. De dicha línea seleccionamos el primer carácter esperado aún no verificado del código esperado siguiendo izquierda a derecha.

El segundo paso es buscar el carácter adquirido al que debe corresponder en el código de la fuente rescatado por la segmentación. Para ello debemos elegir una banda que pueda contener la línea donde está ese carácter esperado. Se hará de arriba a abajo empezando desde la siguiente banda posterior a la última banda verificada con éxito.

Seleccionada la banda, empezamos con el primer carácter adquirido no verificado de izquierda a derecha. Ver figura 8.9.

Con el carácter adquirido y con el carácter esperado aplicamos la verificación que se explica en 8.1.5.2.

Si la verificación es negativa, fusionamos este carácter con el siguiente y reintentamos la verificación.Ver figura 8.10. La lógica que hay detrás de fusionar con el siguiente es que, en este punto, es la mejor alternativa a ignorarlo.

Si la verificación vuelve a ser negativa, ignoramos el carácter y pasamos al siguiente carácter dentro de esa banda repitiendo el ciclo de verificación. Esto se hará siempre que la banda cuente con suficientes caracteres para contener la línea de código esperado. En caso contrario buscaremos una nueva banda tal como se indicó en el segundo paso. Ver figura 8.11.

Si la verificación es un éxito nos desplazamos al siguiente carácter esperado y al siguiente carácter real y repetimos el ciclo de verificación. Ver figura 8.12.

Si, a pesar de verificaciones exitosas los fallos de verificación obligan a designar la banda como fallida, se regresa al primer carácter esperado de la línea y buscaremos una nueva banda más abajo que cumpla las especificaciones. Si no se localiza se retorna una validación negativa de la lata.

El algoritmo de verificación

1. Seleccionamos una línea (índice de línea esperada) no examinada siguiendo el orden de arriba a abajo.

2. El índice de banda se posiciona en una banda apta para esa línea de la fuente siguiendo dos reglas:

   - La banda seleccionada tiene igual o mayor número de formas que la línea de código esperado apuntada por el índice de línea esperada.

[11] Por supuesto el código puede ser personalizado a voluntad por el operario en el sistema impresor y de hecho hay excepciones a esta regla. Ver en 4.24 las imágenes (c) y (g).

- La banda seleccionada no está por encima de una banda que ya ha sido reconocido adecuadamente (o descartada previamente para ese carácter).

3. Seleccionamos el primer carácter del código esperado(índice de carácter esperado) de la banda apuntada por el índice de línea esperada no examinado siguiendo el orden de izquierda a derecha.
4. El índice de caracteres adquiridos apunta al primer carácter adquirido de la banda apuntada por el índice de banda.
5. Si no hay ningún carácter apuntado por el índice de carácter esperado la línea esperada ha sido validada y regresamos a 1.
6. Si no hay ninguna forma apuntada entonces faltan formas para esta línea esperada. Es necesario seleccionar otra banda, y regresamos a 2.
7. Verificamos el carácter esperado con el carácter adquirido.

   a) La verificación es positiva. Movemos el índice de formas a la siguiente forma a verificar de la fuente permaneciendo en la misma banda. Seleccionamos el siguiente carácter esperado moviendo el índice de carácter esperado hasta la siguiente posición y regresamos al paso 5.

   b) La comparación es negativa.

      1) Se fusiona -temporalmente- la forma apuntada por el índice de formas con la siguiente forma y comparamos la forma así fusionada con la lista de morfologías resultado del paso 5.

         a′ Si es positiva entonces hubo un exceso de celo en la fisión. Consideramos los dos caracteres de la fuente ser en realidad un sólo carácter. Movemos el índice de formas a la siguiente forma no examinada ni fusionada. Seleccionamos el siguiente carácter esperado moviendo el índice de carácter esperado hasta la siguiente posición y regresamos al paso 5.

         b′ Si es negativa, asumimos que el carácter fuente es ruido y movemos al índice de formas a la siguiente forma dentro de la banda actual regresamos a 6. En caso contrario regresamos al paso 2.

8.1.5.2. Comparando carácter adquirido y carácter esperado

Esta etapa comprende tres fases bien diferenciadas.

La primera fase consiste en recuperar de la BFM la familia morfológica que corresponde al carácter esperado. La familia debe existir, y deben haber morfologías dentro de esa familia. En caso contrario estaremos ante un fallo crítico en la validación y debería ser comunicado al operario de inmediato.

La segunda fase recupera una a una las morfologías de la familia morfológica y las somete a:

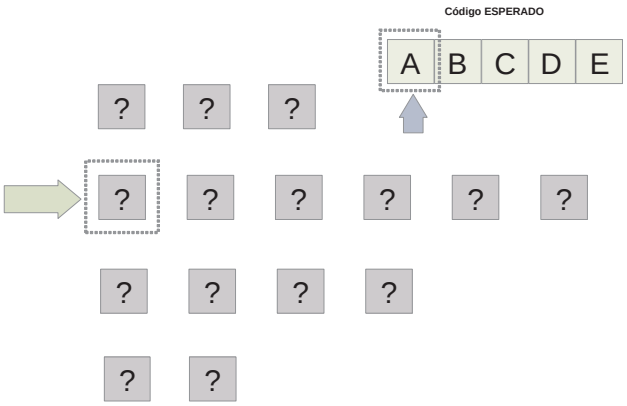


Figura 8.9: Seleccionada una banda.

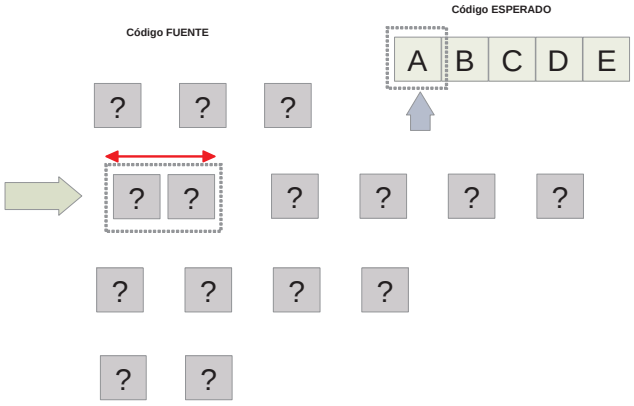


Figura 8.10: Fusión temporal de caracteres consecutivos.

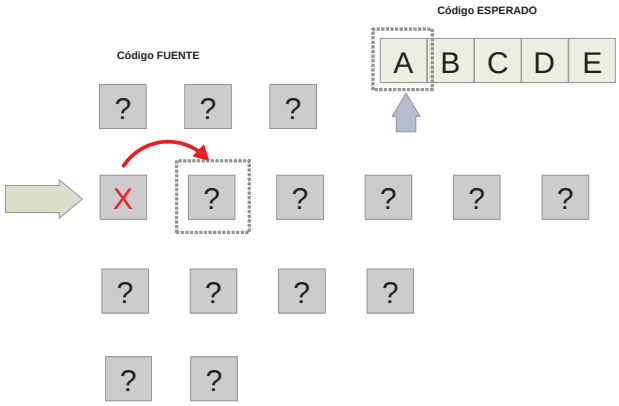


Figura 8.11: Conjeturamos que la banda es correcta pero el carácter actual es ruido. Pasamos al siguiente carácter.

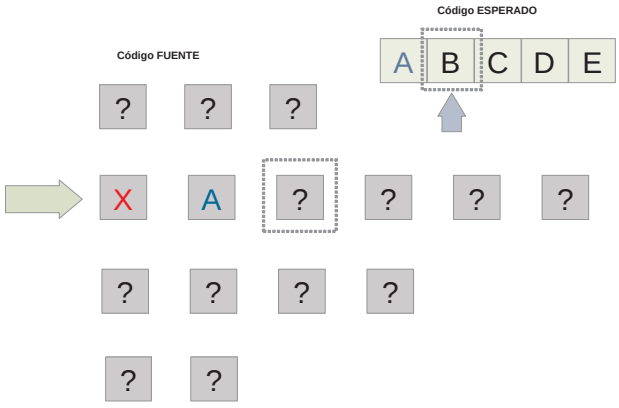


Figura 8.12: Verificación con éxito.

- El test de área cubierta de tinta con el carácter adquirido. Si no lo supera se estima que su similitud es cero(0).

  Se compara la diferencia de tinta entre la morfología y el carácter adquirido. Una diferencia por encima de un umbral de diferencia máxima de tinta admisible hace innecesario llevar a cabo una comparación de plantillas (se puede asumir similitud cero(0)).

- Si supera el test de área cubierta de tinta, se aplica la comparación de plantillas utilizando la distancia entre morfologías descrita en el apartado 8.1.4.2 para evaluar la similitud entre la morfología y el carácter adquirido.

La tercera fase escoge el valor máximo de similitud obtenido en la fase anterior al enfrentar todas las morfologías de la familia morfológica con el carácter adquirido. Si supera cierto umbral la verificación ha sido positiva. En caso contrario es negativa. Este resultado alimenta la resolución de validación descrita en 8.1.6.

8.1.6. Resolución de validación

En ese punto se determina si la lata es válida (validación positiva) o no lo es(validación negativa).

Para decidir se trabajará con:

1. Caracteres esperados verificados positivamente (en posición y línea) en la imagen de lata adquirida.
2. Qué caracteres son importantes. La configuración de MONICOD hecha por el usuario suministra esta información. Ver sobre esto el apartado 3.8.3.

La política que sigue MONICOD es sencilla:

- Si todos los caracteres importantes están presentes en la lata en la línea/orden esperado la lata es válida. En caso contrario NO lo es.

MONICOD se limita a actualizar una estadística de latas validadas totales, positivas y negativas y lanzar un mensaje de aviso en el caso de una validación negativa.

El tratamiento de este mensaje de aviso va más allá de MONICOD convirtiéndose en responsabilidad del operario. Si la validación es positiva es de esperar que no se acometa ninguna medida. En cambio, si la validación es negativa el operario podría:

- Ignorar la validación negativa.
- Retirar la lata afectada de forma manual o mecánica.
- Detener la línea para resolver un problema crónico. En este escenario puede ser necesario un recalibrado del sistema.

8.2. Aprendizaje

Con el término "Aprendizaje" englobamos a los procesos que añaden o suprimen morfologías en la BFM. En MONICOD el aprendizaje ocurre en dos momentos: En tiempo de aprendizaje y en tiempo de validación:

El aprendizaje en -tiempo de aprendizaje- tiene carácter intensivo, es decir, permite un fuerte trasiego de adiciones y supresiones. Es adecuado para construir desde cero la BFM aunque también puede limitarse a hacer actualizaciones en una BFM ya existente.

El aprendizaje en tiempo de validación es de carácter adaptativo. Persigue adaptar progresivamente a MONICOD a los cambios que pueda ocurrir durante la jornada entre los ciclos de aprendizaje intensivos (tiempo de aprendizaje).

8.2.1. Aprendizaje en -tiempo de aprendizaje-

La etapa de aprendizaje ocurre fuera del ciclo de validación tal como sucede con la etapa de calibrado. Al no estar el sistema en tiempo de producción no hay restricciones temporales y pueden aplicarse procedimientos que serian de un costo prohibitivo en tiempo de validación. La operación de calibrado debe ser siempre anterior al aprendizaje. La supervisión humana está indicada.

Se constituye en dos subsistemas: Automático y manual.

8.2.1.1. Subsistema automático

Subsistema automático de procesamiento de plantillas sin intervención del usuario. Esencialmente el principio de funcionamiento es similar al proceso de validación ya descrito con una captura de lata que se garantiza correctamente impresa. La diferencia es que ahora el código real obtenido (Ver subsección 7.3.8 y figura 7.56) puede descomponerse y agregarse de manera directa a la BFM como actualización de ejemplos legítimos de morfologías. Para ello se seguirá un esquema basado en eventos. Ver figura 8.13.

Metodología Partamos de una BFM vacía.

Primero se recrean las condiciones de una validación sin incidencias. Esto es; el sistema impresor imprime correctamente en las latas, MONICOD ha sido calibrado con éxito y es capaz de generar códigos esperados. Dado que la BFM está vacía MONICOD no puede validar porque no tiene morfologías para evaluar la entrada.

Entramos en modo de aprendizaje automático.

Se aplica el procedimiento de validación descrito hasta alcanzar el denominado código real. Para cada uno de los caracteres esperados se extrae la forma correspondiente del código real obtenido y se consulta la BFM para la familia morfológica asociada.

A partir de aquí, el comportamiento dependerá de la respuesta de la BFM.

Si no existen morfologías en la BFM para ese carácter esperado se lleva a cabo la adición directa en la base de la forma extraída del código real. Es el evento *inaugurar familia morfológica*.

Si existen morfologías en la BFM para ese carácter esperado se comparan todas las morfologías almacenadas con la forma extraída del código real para determinar cual de ellas se parece más. No son posibles los empates (por la manera en la que se pueblan las familias morfológica). A partir de aquí existen varios escenarios posibles.

Si la similitud está por encima del umbral de voto, la morfología seleccionada de la BFM recibe un voto. Es el evento *votar(asimilar) morfología*. Ver 8.14.

Si la similitud está por debajo del umbral de voto pero por encima del umbral de admisión se añade la morfología a la familia morfológica. Es el evento *ingresar morfología*. Ver 8.14.

Si la similitud está por debajo del umbral de admisión ha habido una anomalía y se produce el rechazo del candidato a morfología. Es el evento *rechazar morfología*. Ver 8.14.

Naturalmente hay un número máximo de morfologías que puede ser soportada.

El proceso se repite hasta que se tenga un número suficiente de morfologías para cada familia morfológica o el usuario supervisor detenga el proceso. Otras condiciones de parada son posibles: BFM estable (no hay ingresos), suficientes votos acumulados en cada familia morfológica, completado un cupo de morfologías para cada familia morfológica...

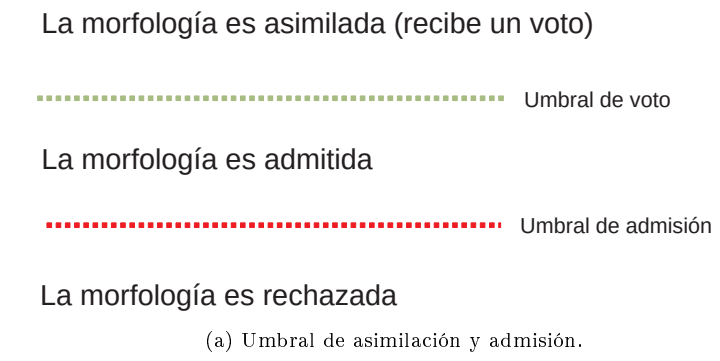
Concluido el ciclo de aprendizaje automático se lleva a cabo una purga automática de morfologías basada en el número de votos conseguido. Se procede a la supresión de morfologías por debajo de un umbral de permanencia. Es el evento *purgar morfología*. Ver 8.14.

El algoritmo del subsistema automático

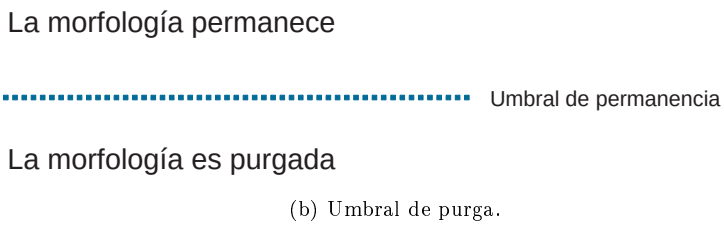
1. Inicio ciclo de aprendizaje. La BFM podría o no estar vacía.
2. Llega una nueva morfología para la familia morfológica X.
   - *a*) La familia morfológica X está vacía(no tiene morfologías). Se inaugura la familia morfológica X con la nueva morfología que pasa a ser morfología almacenada. Ir a 3.
   - *b*) La familia morfológica tiene morfologías. Se busca(en el conjunto de morfologías almacenadas que pertenecen a esa familia morfológica) la morfología almacenada con mayor similitud obtenida.
     1) La similitud obtenida está por encima del umbral de voto. Se vota por la morfología almacenada. Ir a 3.
     2) La similitud obtenida esta por encima del umbral de admisión(pero por debajo del umbral de voto). Se añade la nueva morfología a la familia morfológica X, y pasa a ser morfología almacenada. Ir a 3.

Evento	Descripción
inaugurar familia morfológica	No existen morfologías de esta familia. Adición directa de la nueva morfología.
votar(asimilar) morfología	La similitud con la morfología más parecida en la familia está por encima del umbral de voto. Se vota por la morfología almacenada.
ingresar morfología	La similitud con la morfología más parecida en la familia está por encima del umbral de admisión. Se añade la nueva morfología.
rechazar morfología	La nueva morfología es rechazada.
purgar morfología	Morfologías almacenadas con número de votos por debajo de un umbral de permanencia son eliminadas.

Figura 8.13: Tabla de eventos



(a) Umbral de asimilación y admisión.



(b) Umbral de purga.

Figura 8.14: Umbrales empleados en tiempo de aprendizaje.

3) En cualquier otro caso se rechaza la nueva morfología. Ir a 3.

3. Fin de ciclo de aprendizaje Volver a 1.

4. Se recorren todas las morfologías almacenadas eliminando aquellas cuyo número de votos esté por debajo del umbral de permanencia.

Limitaciones El subsistema no garantiza de ningún modo que el aprendizaje sea correcto o suficiente. Un ejemplo evidente de esto son las condiciones del evento *inaugurar familia morfológica*. Cualquier morfología, correcta o no, puede inaugurar cualquier familia morfológica. No hay restricciones. Sin embargo esta primera morfología condiciona los futuros ingresos en esa familia. Por tanto es plausible que si la primera morfología resulta errónea se produzcan eventos *rechazar morfología* erróneos para candidatos legítimos que invaliden el aprendizaje. La supervisión humana. es un requerimiento bajo la actual formulación.

8.2.1.2. Subsistema manual

Subsistema manual de procesamiento de plantillas que requiere la interacción del usuario. Se busca refinar los resultados crudos obtenidos por el subsistema automático ya descrito.

Metodología El supervisor visualiza las morfologías almacenadas y, siguiendo su propio criterio cualitativo, las elimina o sanciona.

El algoritmo del subsistema manual

1. Para cada morfología...

2. El supervisor visualiza la morfología almacenada en la BFM catalogada por familia morfológica.

3. El usuario decide si suprimir o no la morfología.

4. Si quedan morfologías pendientes de revisión volver a 1.

Limitaciones Sólo se contempla la operación de supresión. No se pueden mover morfologías entre familias morfológicas, ni editar las morfologías almacenadas.

8.2.2. Aprendizaje en 'tiempo de validación'

Esta etapa de aprendizaje ocurre dentro del ciclo de validación. Durante la validación nuevas morfologías son adquiridas y morfologías almacenadas son eliminadas. Se busca con ello absorber pequeños cambios durante el tiempo de operación. No se inauguran nuevas familias morfológicas en este tipo de aprendizaje. Debe existir una base de morfologías de largo plazo almacenadas ya en la BFM obtenidas por aprendizaje en 'tiempo de aprendizaje'.

Definiciones

Morfologías de corto plazo Morfologías de corta duración que son añadidas y eliminadas en tiempo de validación. Conservan una similitud suficiente con el conjunto de morfologías de largo plazo.

Morfologías de largo plazo Morfologías de larga duración de legibilidad mínima. Todo carácter fuente debe tener una similitud mínima con estas morfologías para ser legible. Si tiene suficiente similitud, y es suficientemente diferente a las morfologías de corto plazo actualmente presentes puede ser 'aprendido' en tiempo de validación, ingresando como morfología de corto plazo. Si el número de morfologías para esa familia morfológica ya está cubierto substituirá a la morfología de corto plazo menos utilizada hasta el momento.

8.2.2.1. Subsistema automático

Se lleva a cabo el proceso de validación.

Si el código es validado positivamente y la morfología corresponde en un 100 % a una morfología de corto plazo almacenada se vota por ella.

Si el código es validado positivamente y está completo el cupo de morfologías se elimina la morfología de corto plazo con menos votos y se añade la nueva morfología. Otra opción es substituir a la morfología menos utilizada en las verificaciones recientes.

Si el código es validado positivamente y no está completo el cupo de morfologías de corto plazo las morfologías del código son añadidas a la BFM como morfologías de corto plazo.

8.2.2.2. Consideraciones

En el momento de redacción del presente documento el trabajo sobre el sistema de aprendizaje en tiempo de validación está en un estado preliminar. Introduce nuevas complejidades que están a la espera de ser sopesadas.

Capítulo 9

Resultados

"Donde se miden las fuerzas, y se juzgan los logros"

Este capítulo se compone en dos secciones.

Durante la primera sección (ver 9.2) se operará sobre MONICOD con una muestra real de 10000 *frames* exhibiendo y comentando los resultados relativos a calidad y tiempo. Estimamos la muestra como razonablemente representativa porque, a igual resolución de entrada (640x480), igual distancia cámara-lata y mismo tamaño de la fuente de texto, los resultados son fácilmente escalables para cualquier número de *frames*.

En la segunda sección examinaremos cualitativamente las partes de MONICOD. Además se comparará el preproceso y la clasificación con otros métodos alternativos, prestando especial atención al tiempo invertido.

.

9.1. Condiciones previas

- TODOS los experimentos de este capítulo se han hecho sobre la misma máquina.
  - Características: Intel(R) Xeon(TM) CPU 3.80GHz , 3.79 GHz 2.00 GB de RAM (Dos núcleos)
- Entorno general
  - El entorno (sistema operativo, software cargado) ES EL MISMO para todos lo experimentos.
  - Windows XP SP3 32 bits
- Compilador
  - El compilador utilizado y opciones empleadas para generar el código objeto que realiza los experimentos ES EL MISMO.

  - Microsoft Visual Studio 2010 32 bits

- Medición de tiempos

  - La medición de tiempo se ha llevado a cabo a través de la implementación presentada en [117] que utiliza las APIS de alta eficiencia ofrecidas en Windows. Existe una imprecisión inherente a la medición de tiempo en un computador[1] pero es de esperar que múltiples repeticiones de la misma lectura la suavicen.
  - Todos los tiempos en gráficas y tablas están medidos en segundos.

9.2. El experimento MONICOD

9.2.1. La muestra

La muestra con la que operaremos ha sido extraída directamente de planta y almacenada en disco para poder repetir el mismo experimento cuantas veces sea necesario. Se ha verificado manualmente que las latas contenidas en la muestra están CORRECTAMENTE etiquetadas, sin exhibir ninguna tara grave, salvo una excepción.

El "vídeo" almacenado (9885 *frames*) corresponde en realidad a cinco adquisiciones independientes de 10 segundos cada una a 200 *frames* por segundo (2000 *frames* por extracción). El motivo de proceder así y no disponer de una adquisición continua es descrito en 4.8. Las adquisiciones han tenido lugar en el espacio de 20 minutos aproximadamente.

En el almacenamiento se ha usado un formato de imagen sin compresión para evitar la introducción de artefactos, degradación de color o pérdida de *frames* a causa del computo asociado a la comprensión (900kb por frame). El espacio en disco que ocupa la muestra son 8,48 GB.

Se ha dividido la muestra en dos conjuntos:

- El conjunto de aprendizaje (compuesto de 1005 *frames*). Esto es, aproximadamente 5 segundos de adquisición no continua. Ese conjunto se ha

[1] Sobre esto último es interesante denotar que, al menos en el momento de redacción de este documento, medir tiempo con precisión por debajo de una cierta magnitud en un computador no es, en modo alguno, trivial. Todo lo contrario. Puede ser directamente imposible. Hay razones hardware y software para esto:

Entre las razones hardware la frecuencia procesador no es constante: Depende del hardware y de las políticas *power-save*. La presencia de varios núcleos, el uso de multithreading, etcétera complican la cuestión. Es posible alcanzar precisión por debajo del milisegundo pero no precisiones del orden de nanosegundos.

Respecto al software, el ecosistema de procesos en un sistema tal como Windows XP SP 3 es complejo. Incluso en condiciones de aparente reposo, un proceso no relacionado (por ejemplo una tarea interna de S.O) podría inadvertidamente consumir ciclos en el procesador distorsionando la lectura de tiempo. Desafortunadamente, por razones relativas a la compatibilidad del driver de la tarjeta de adquisición, se descartó considerar otros sistemas operativos (real-time operating system) que quizás podrían ofrecer una mejor personalización sobre esto. También el uso de hilos oscurece la obtención de lecturas.

construido al fundir extractos de cada una de las cinco adquisiciones (201 *frames* por cada adquisición). Los *frames* del conjunto mantienen el orden cronológico de adquisición. Manualmente se ha verificado que contiene 53 latas. A 200 frames por segundo se trata de una adquisición de 5,025 segundos de duración

- El conjunto de validación (compuesto por 8884 *frames*). Esto es, aproximadamente, 45 segundos de adquisición no continua. Los frames del conjunto mantienen el orden cronológico de adquisición. Manualmente se ha verificado que contiene 465 latas. A 200 *frames* por segundo se trata de una adquisición de 44,42 segundos de duración

9.2.2. Parámetros MONICOD

Ver en el anexo B un listado exhaustivo de todos los parámetros MONICOD con sus respectivos valores y descripciones para el experimento.

9.2.3. Aprendizaje

Una pasada del conjunto de entrenamiento es suficiente para llevar a cabo una sesión de aprendizaje.

9.2.4. Calidad

Los resultados del sistema se indican en la tabla 9.1 y se muestran graficados en la figura 9.1.

- El recuento de latas llevado a cabo por MONICOD es correcto. Este resultado positivo en particular no resulta representativo al tratarse de una simulación: Cargar una imagen de disco es una operación muy lenta y por tanto, no hay riesgo de perdida de *frames* como ocurriría en una adquisición desde cámara[2].

- Dado que las latas del vídeo están correctamente etiquetadas (verificado manualmente) salvo una, las validaciones negativas obtenidas deben corresponder a fallos del sistema (falsas validaciones negativas: Se indica que el código de caducidad es erróneo cuando de hecho es correcto) salvo una, que sería una validación negativa correcta.

- Todos los caracteres serán tomados como IMPORTANTES salvo los dos puntos (":") y la barra inclinada("/") que serán ignoradas.

- Se ha desactivado el aprendizaje en tiempo de validación dada su condición incipiente y la dificultad que entraña interpretar correctamente su impacto.

[2]Es posible mejorar la simulación cargando previamente en memoria los *frames* , pero el sistema se degradaría invalidando la posibilidad de obtener medidas realistas de tiempo, y por otra parte sólo podríamos cargar una fracción del vídeo.

Validaciones Totales	Validaciones Positivas	Validaciones Negativas
465	438	27

Cuadro 9.1: Resultados del proceso MONICOD

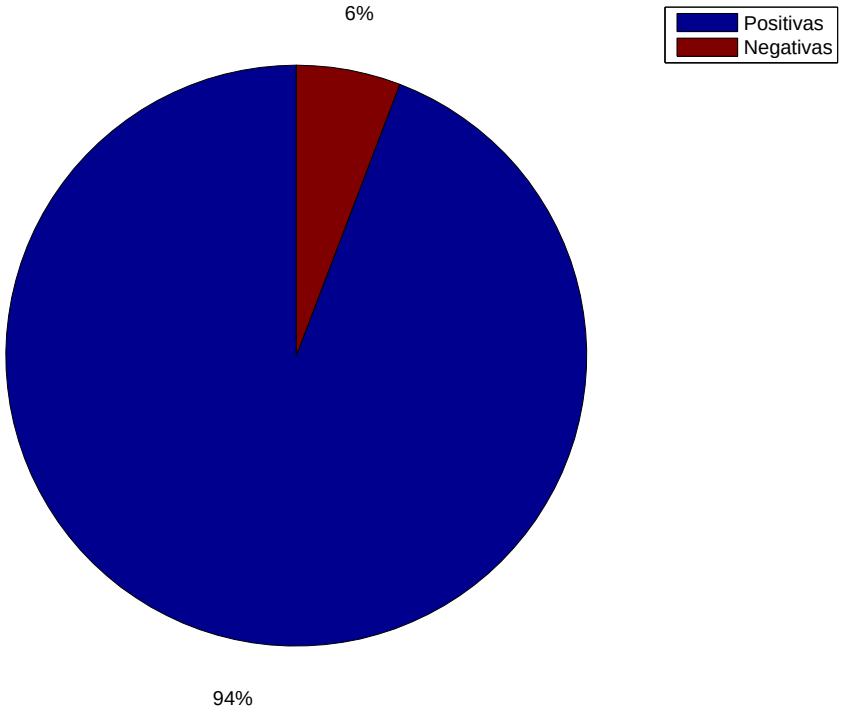


Figura 9.1: Representación gráfica

Causa del fallo	Fallos	% Total	Descripción
Fallo histogramas elípticos	13	50 %	El código completo no está dentro del área de interés.
Fallo división de bandas	11	42.3 %	El código no ha sido separado correctamente en bandas/líneas de texto.
Fallo agrupamiento	1	3.8 %	Un agrupamiento ha sido mal ejecutado.
Fallo validación de carácter	1	3.8 %	Un carácter no ha sido validado.

Cuadro 9.2: Causas de validaciones negativas fallidas

Como se ha establecido:

- 1 lata fue correctamente declarada no valida (de manera que no cuenta como fallo sino como acierto). El resto son fallos en validación. En la tabla 9.2 se recoge la causa de esos errores. En todos estos errores el código es legible.

Fallos en los histogramas elípticos Todos los fallos en los histogramas elípticos consistieron en la pérdida total o parcial del carácter más a la derecha, próximo al borde de lata. Ver figura 9.2.

La diagnosis revela un problema de umbral: El código extraviado es ensombrecido por la sombra proyectada por el borde de la lata. Esto es agravado por una iluminación no homogénea. Para reducir fallos en esté área pueden intentarse otras formas de áreas de detección (buscando el centro de la elipse en lugar de sus límites, o bien introducir una configuración multiumbral).

Fallos en la división de bandas Distinguiremos aquí entre dos escenarios. Código no rotado, y código rotado.

Código no rotado Cuando el código está rotado más allá de 30 grados la eficacia de la división por bandas se ve considerablemente degradada. Ver figura 9.3. Este problema puede ser combatido con eficacia con una configuración física donde *se minimice la distancia entre cámara y cabeza impresora*. Para esta prueba la distancia entre ambas era superior a 0.5m.

Código rotado La división por bandas propuesta no es infalible, ni siquiera cuando el código sólo está ligeramente rotado. Ver figura 9.4. Existen dos dificultades; un código demasiado grueso suprime espacio de interlineado y

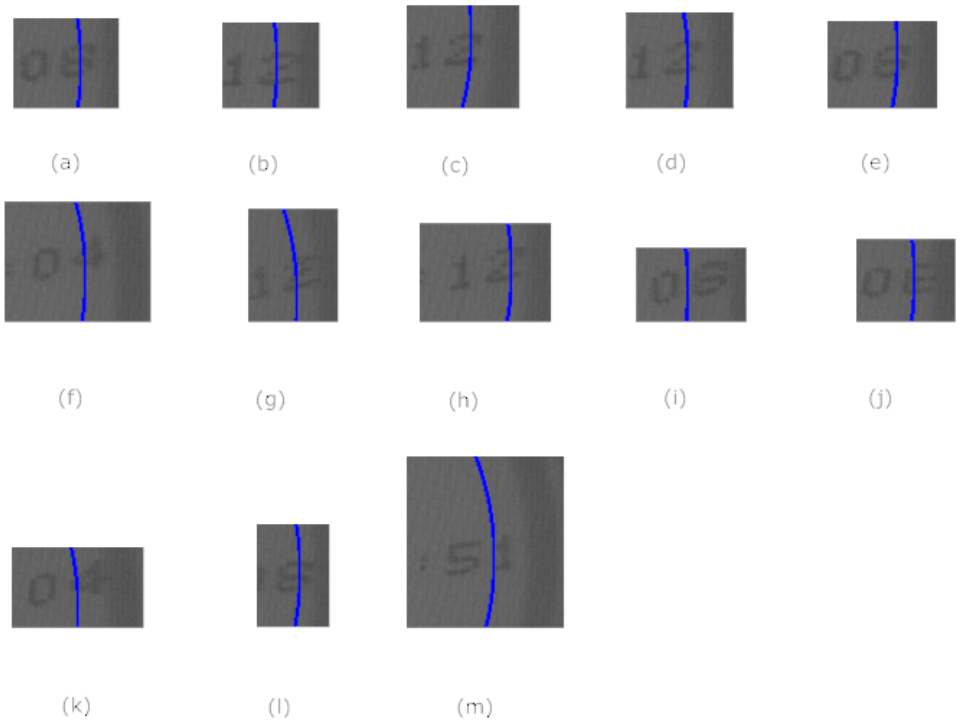


Figura 9.2: 13 fallos en los histogramas elípticos. El borde azul es el límite de la región de interés. Todo lo que queda fuera de ella es ignorado. Circunstancialmente en esta codificación, el carácter extraviado corresponde siempre al último dígito del minutero de la hora de impresión que suele ser un carácter irrelevante para la caducidad. (☼)

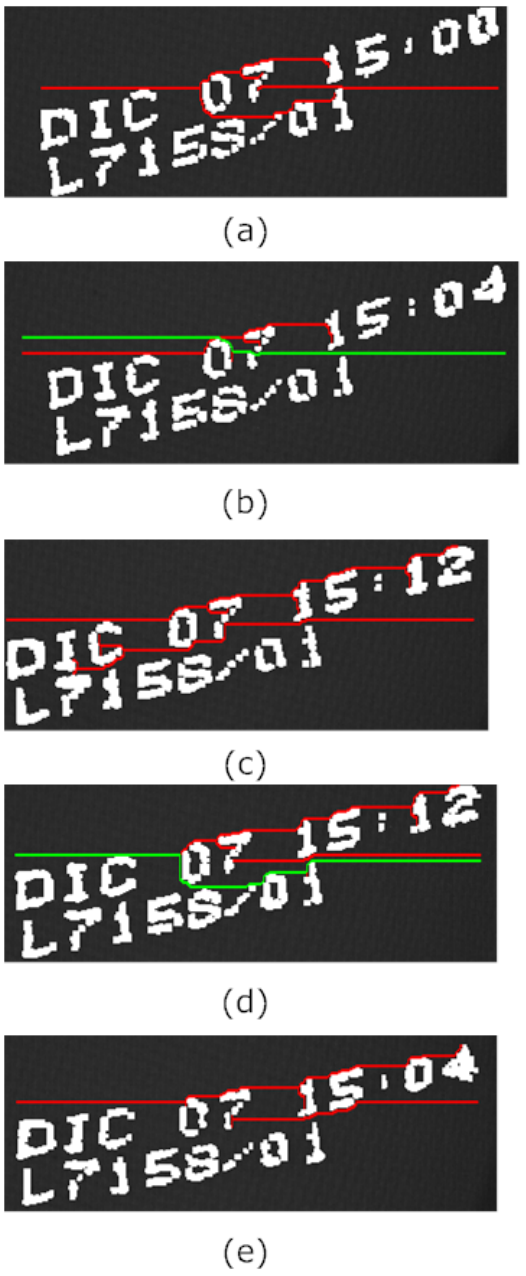


Figura 9.3: Fallos en división por bandas con códigos rotados 30 grados o mas correspondientes al primer intento de división. En (c) está cerca de conseguirlo pero el engrosamiento del código se lo impide. La mala elección de la fila de inicio por la izquierda y las concavidades son las principales causas.

Max	Min	Media	Mediana	Moda	Desv. Estándar
0,0137	0,0088	0,01	0,01	0,0088	$0{,}40337 \times 10^{-4}$

Cuadro 9.3: Estadística de tiempo por lata. La dispersión es muy baja como cabría esperarse de latas que guardan importantes similitudes (legibles, mismo tamaño de fuente, mismo número de caracteres,...).

un código con demasiadas concavidades resulta problemático de explorar por los agentes. Las soluciones propuestas para ambos problemas se oponen entre sí: Para combatir las concavidades el engrosamiento es efectivo, pero a cambio se pierde espacio interlineal (también espacio intercaracter, se pierde definición en los caracteres, ...). Reducir el engrosamiento aumenta la fragmentación de código, y resulta más fácil que los caracteres queden atravesados por una banda.

Fallo en el agrupamiento El agrupamiento ha probado de forma consistente un gran comportamiento en calidad y, como luego veremos, en eficiencia. Además aunque impone como fuerte restricción que el código NO esté rotado/inclinado[3], el algoritmo parece menos sensible que el *divisor* de bandas. agrupando con éxito códigos rotados sobre 30º. En la figura 9.5 se presenta el único caso de fallo de agrupamiento en el experimento MONICOD.

Fallo en validación de carácter La validación de caracteres basada en plantillas depende de un único umbral que determina el grado de similitud. El carácter esperado debe parecerse -lo suficiente- a al menos una de las morfologías en la familia morfológica esperada. Esto no siempre va a ocurrir. Ver figura 9.6. Con el sistema de validación actual NO ES POSIBLE garantizar una colección directa de morfologías absolutamente representativa.

El fallo que no era tal En un caso MONICOD indicó correctamente que una lata no correspondía al código de caducidad (es decir, era un fallo legítimo). En el pie de la figura 9.7 se explica que ha hecho MONICOD en este caso.

9.2.5. Tiempo

En 9.8 puede apreciarse de forma gráfica el tiempo invertido en cada lata. El límite teórico por *frame* es rebasado pero quedamos bastante por debajo del límite de tiempo por lata. Ver en el cuadro 9.3 la estadística correspondiente.

Si desglosamos estos tiempos por lata en las diferentes fases obtenemos la figura 9.9. Su estadística asociada queda presentada en 9.4. Se constata que la operación del realce y, ya bastante por debajo, la operación de ecualizado son las operaciones más costosas en MONICOD en el cómputo de lata.

[3]Tiene sentido hablar de código "rotado" porque, efectivamente, es consecuencia de una lata que ha girado grados respecto a su posición en el momento de la impresión.

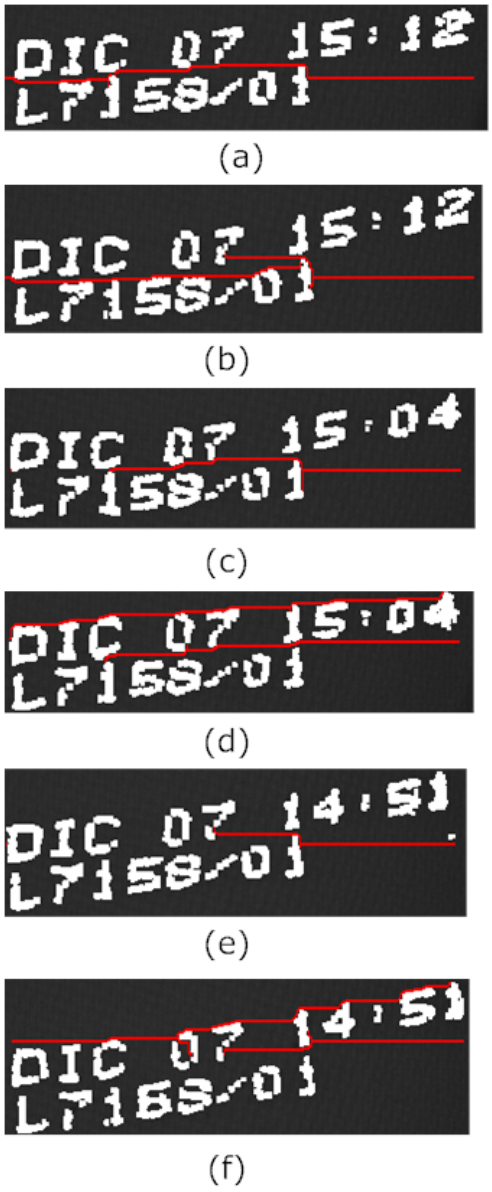


Figura 9.4: Fallos en división por bandas en códigos no rotados (menos de 30 grados) correspondientes al primer intento de división. (a), (c) y (d) son incapaces de encontrar un camino válido para separar las líneas: El engrosamiento del código se lo impide. Los agentes en (b), (e) y (f) quedan atrapados en concavidades o generan rutas invalidas.

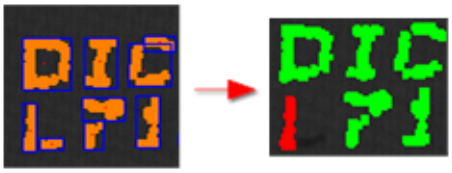


Figura 9.5: Fallo en el agrupamiento. La "L", dividida en dos fragmentos que no presentan solapamiento vertical y están distantes, no ha podido ser unificada por el ciclo de agrupamiento.

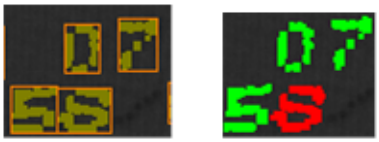


Figura 9.6: Fallo en la validación. El "8" sencillamente acumula demasiadas taras dispersas para que su validación sea efectiva. Si en tiempo de aprendizaje no apareció una plantilla similar entonces no es un "8".

Los picos pronunciados en el agrupamiento derivan de cuanto es necesario agrupar. Con los actuales parámetros de engrosamiento (asignador de tinta) y código con escasa rotación, en muchas ocasiones no se necesita agrupamiento alguno.

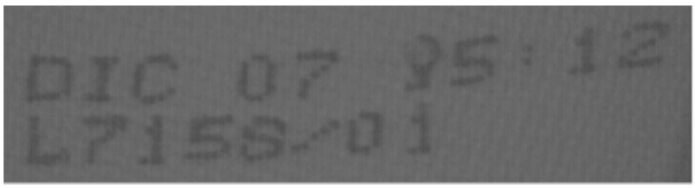
En la figura 9.10 puede apreciarse el tiempo total invertido por el sistema en el procesamiento de latas (4.6669 segundos[4]). Considerando que el tiempo de adquisición asciende a 44,42, el porcentaje de ese tiempo invertido en el procesamiento efectivo de latas es un 10,5 % del tiempo de adquisición. Se examina con más detalle la proporcionalidad de las diferencias entre fases en 9.11.

[4]Tómese siempre con precaución los valores del orden de submilésimas.

Fase	Max	Min	Media	Mediana	Moda	Desv. Estándar
Realce	0,0044	0,0032	0,0036	0,0036	0,0032	$1,5885 \times 10^{-4}$
Ecualizado	0,0023	0,0020	0,0020	0,0020	0,0020	$3,6394 \times 10^{-5}$
Inundación	0,0016	0,0012	0,0013	0,0013	0,0013	$4,4609 \times 10^{-5}$
Interlineado	0,0042	0,0015	0,0017	0,0017	0,0018	$1,4375 \times 10^{-4}$
Agrupamiento	0,0031	$2,7445 \times 10^{-6}$	$4,9381 \times 10^{-5}$	$2,5110 \times 10^{-5}$	$2,0471 \times 10^{-5}$	$2,5619 \times 10^{-4}$
Validación	0,0022	$3,4446 \times 10^{-4}$	0,0013	0,0013	$3,4446 \times 10^{-4}$	$1,6596 \times 10^{-4}$

Cuadro 9.4: Estadística por lata desglosado por fases.

(a)



(b)

Figura 9.7: La validación negativa es legal. (a) El sexto carácter de la primera línea no es reconocible. Se espera un "1" y el carácter encontrado no es (en realidad es irreconocible). Se compara con el siguiente carácter que tampoco es (en su lugar es un "5"). Se compara con el siguiente carácter que si que es un "1". Por eso aparece en verde. Se busca ahora el "5" que no se encuentra. El código es incorrecto. Con todo se buscan los caracteres que quedan. El "1" que no se encuentra y el "2" que se encuentra pero "descolocado", por eso aparece en amarillo. En (b) se comprueba que la lata está efectivamente impresa: El sexto carácter de la primera línea está emborronado. Probablemente porque en el momento de la impresión en ese punto había una gota de agua. (☼)

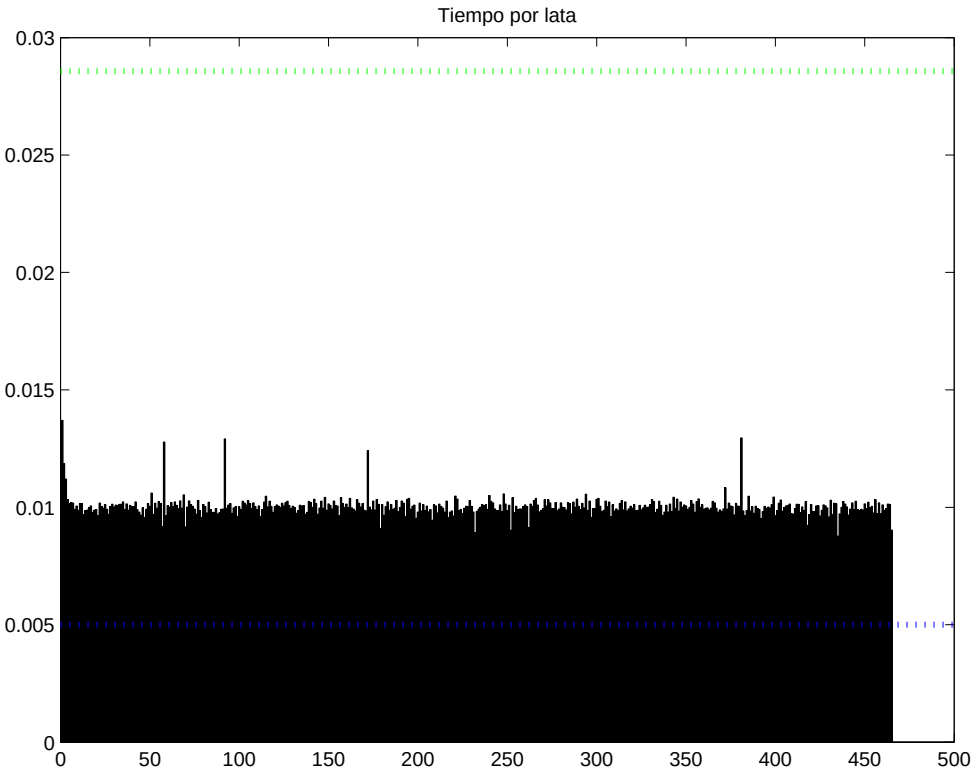


Figura 9.8: Gráfica de barras de tiempo invertido (eje de ordenadas) en cada lata (eje de abscisas). El límite de tiempo teórico que puede invertirse por lata (1/35 segundos) es indicado con una línea verde. El límite de tiempo teórico que puede invertirse por *frame* (1/200) es indicado con una línea azul.

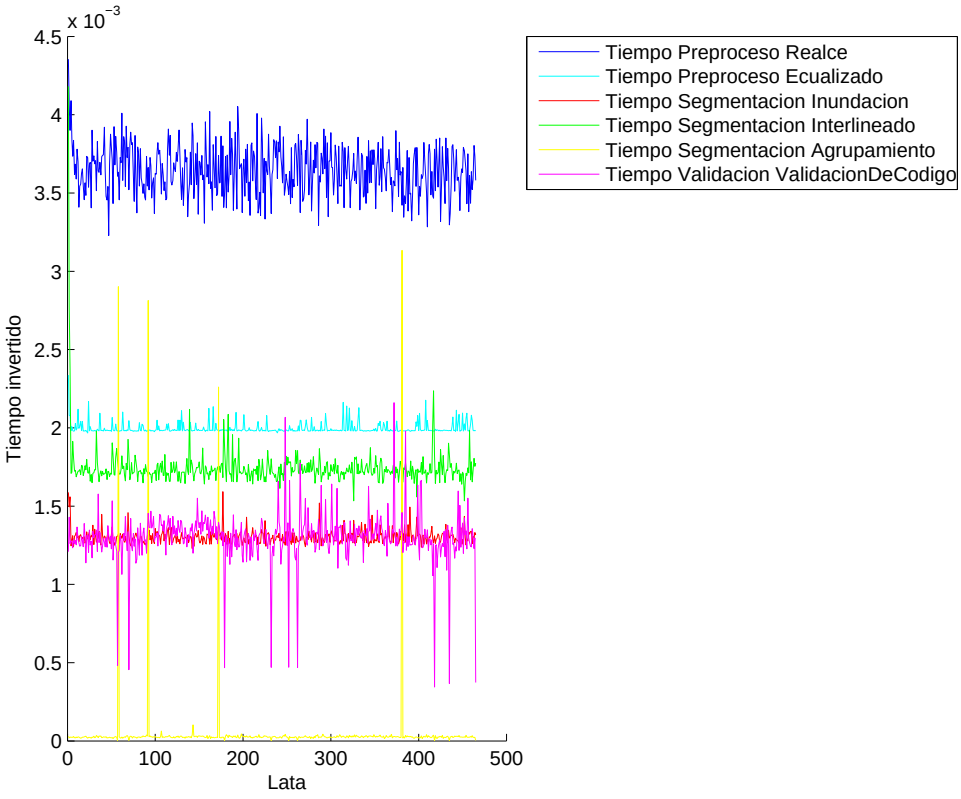


Figura 9.9: Tiempo invertido en cada lata desglosado por fases.

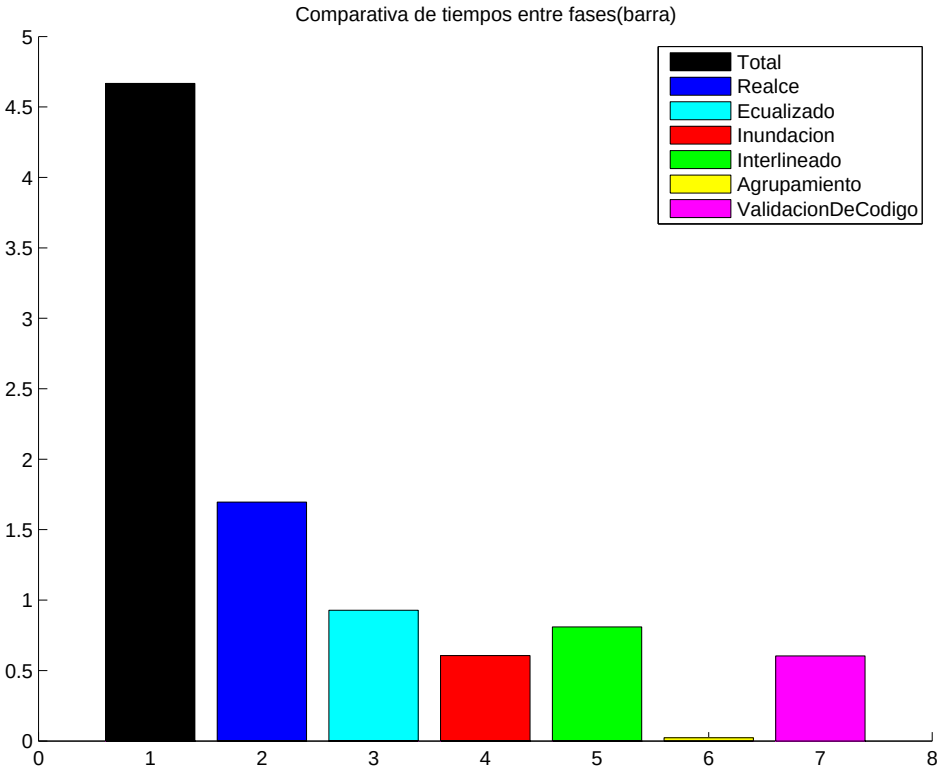


Figura 9.10: Tiempo total invertido por el sistema en el procesamiento de latas de la muestra desglosado en fases.

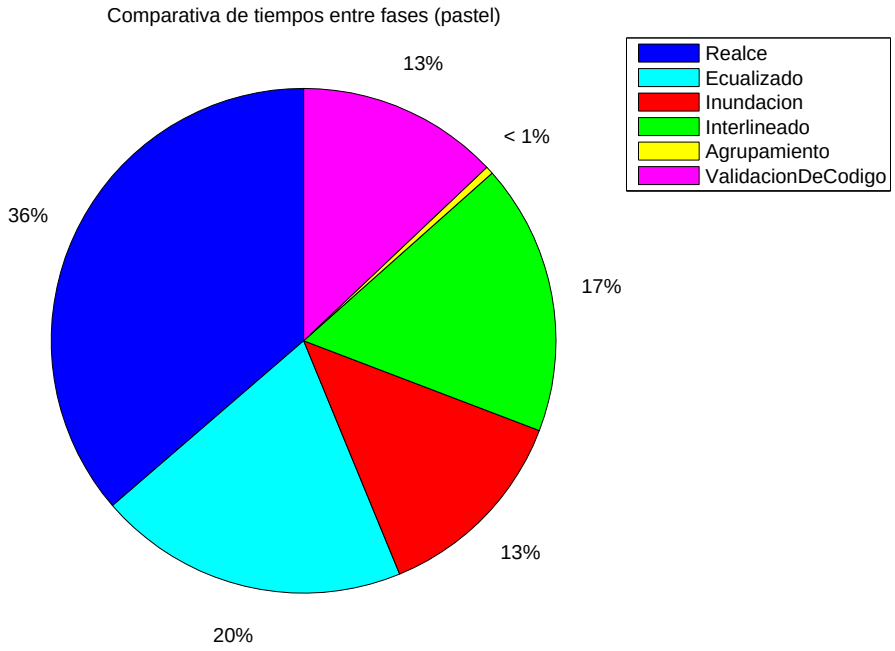


Figura 9.11: Comparativa por fases en el tiempo total invertido por el sistema en el procesamiento de latas de la muestra

9.3. Preprocesamiento

Para un estudio comparativo del costo computacional durante el preprocesamiento se ha llevado a cabo un estudio comparativo de algoritmos clásicos en el área enfrentándolo a una de las restricciones más importantes de tiempo en MONICOD: La adquisición de 200 *frames* por segundo, lo que supone dedicar $\frac{1}{200}$ segundos al procesamiento de un *frame*. Es lo que denominamos límite tope y es indicado en las gráficas con una linea vertical verde y discontinua. Nótese que la velocidad de adquisición NO es la velocidad a la que debemos procesar cada lata, que es 35 latas por segundo ($\frac{1}{35}$ segundos por lata). Por tanto la restricción es mucho mucho más severa de lo que se necesita.

Algunas consideraciones:

- Los algoritmos de esta sección se aplican sobre un área de interés elíptica (en realidad circular) y no sobre toda la imagen, tal como hace MONICOD, para que la comparación sea válida.
- Todos los algoritmos de esta sección han sido codificados por el autor de este documento, empleando toda la habilitad de la que ha sido capaz para optimizarlos.

9.3.1. Algoritmos de preprocesamiento examinados

- Tarea Nula. La tarea nula es una referencia del costo necesario que hay que invertir en la convolución del área de interés con una matriz de 3×3.
- Realce MONICOD. Ver figuras 9.12 y 9.13.
- Ecualizado. Ver figuras 9.14 y 9.15.
- Umbrales Globales. Ver figuras 9.16 y 9.18.
  - Otsu
  - Kittler-Illingworth
- Umbrales Locales. Ver figuras 9.19, 9.20, 9.21 y 9.22.
  - Bernsen
  - Mean
  - Median
  - NiBlack
  - Sauvola
- Detectores de bordes. Ver figuras 9.23, 9.24, 9.25, 9.26 y 9.27.
  - Roberts por filas y columnas
  - Prewitt por filas y columnas
  - Sobel por filas y columnas
  - Frei y Chen por filas y columnas

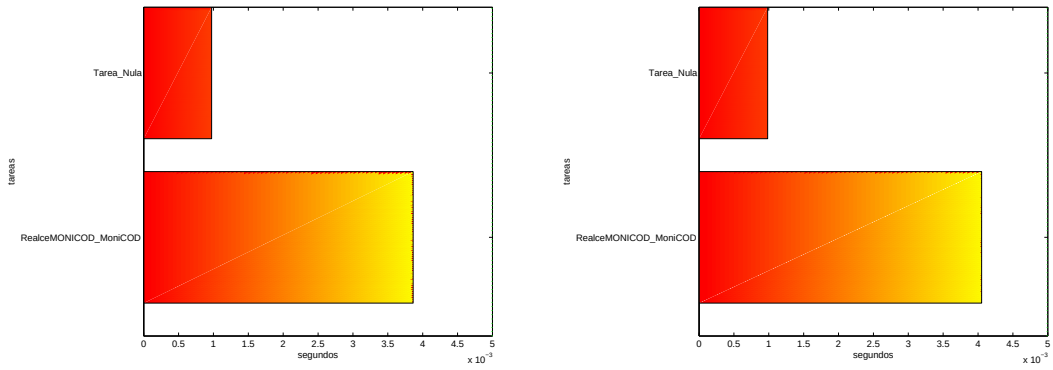


Figura 9.12: Realce MONICOD sobre hojalata(izquierda) y aluminio(derecha).

Realce MONICOD La operación de realce está diseñada para reducir la fragmentación de los caracteres. Consiste en substituir los valores de la ventana local por el mínimo local de la ventana. Como puede verse en la gráfica 9.12 resulta difícil aplicarla *frame* a *frame*. Los resultados quedan constatados en la figura 9.13. La versión mostrada aquí es la que emplea MONICOD y es, a la postre, la parte que más consumo de tiempo demanda.

Ecualizado El ecualizado es uno de los pocos algoritmos no originales de MONICOD. Consigue un buen resultado aumentado el contraste de la superficie de lata y su simplicidad resulta seductora aunque use aritmética flotante. Se hace necesario recalcular la tabla de ecualizado debido a variaciones de histograma entre superficies de lata.

Umbrales Globales El bajo consumo de los umbrales globales ha supuesto una auténtica sorpresa. Como puede verse en 9.16, para nuestro ejemplo, el tiempo de computo es francamente bajo e invita a incorporarlo en el trasiego lata a lata de MONICOD. Desde luego recalcular el umbral de forma automá-

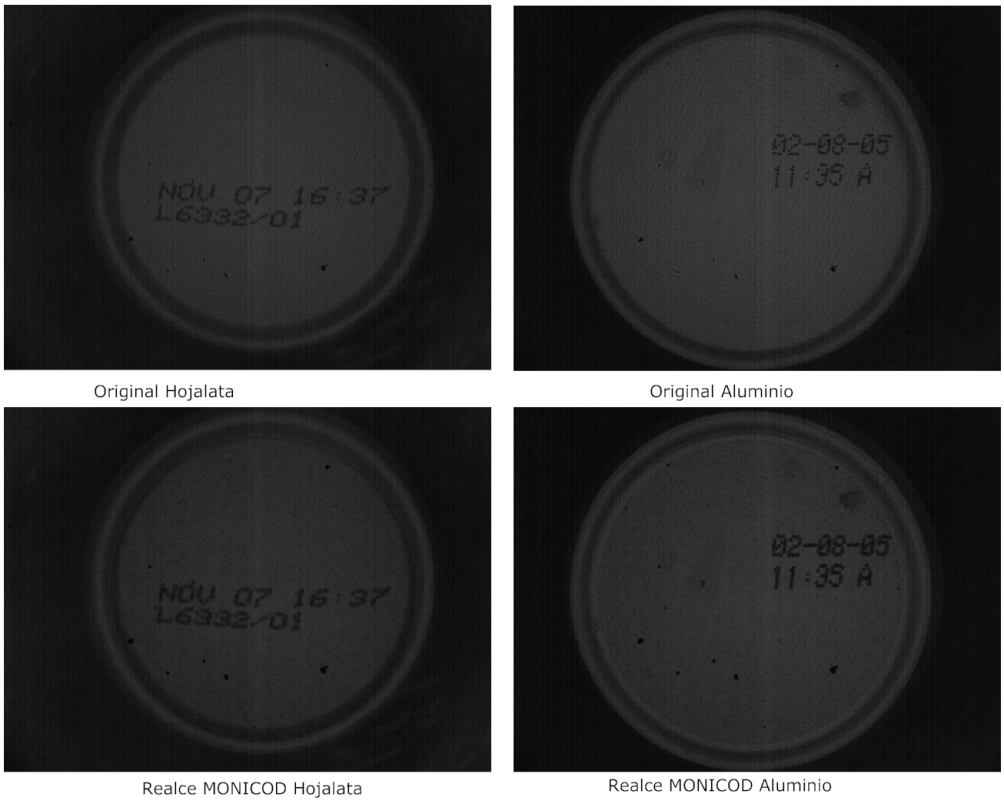


Figura 9.13: Realce MONICOD

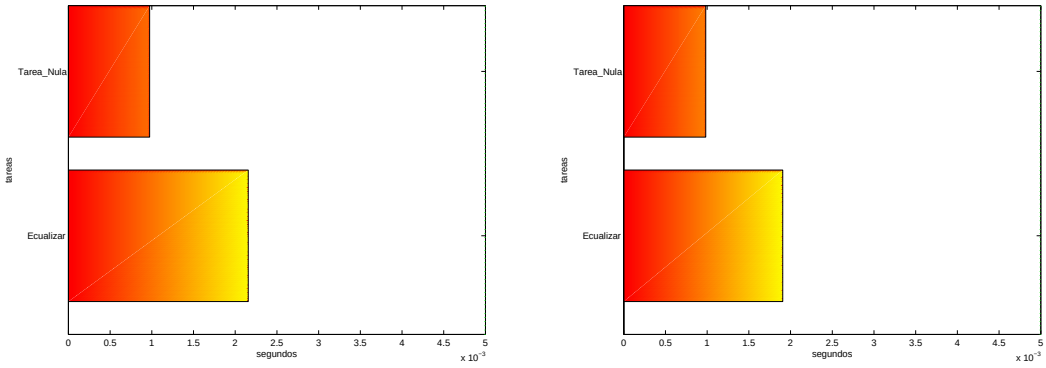


Figura 9.14: Ecualizado de Hojalata (izquierda) y aluminio (derecha).

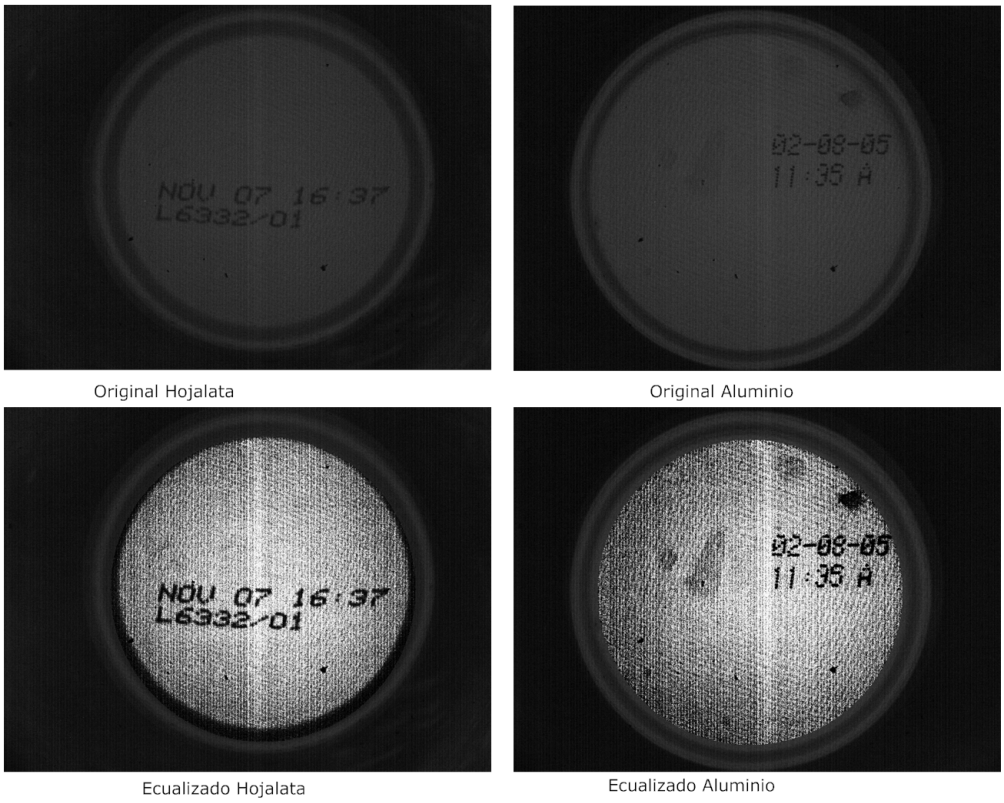


Figura 9.15: Ecualizado

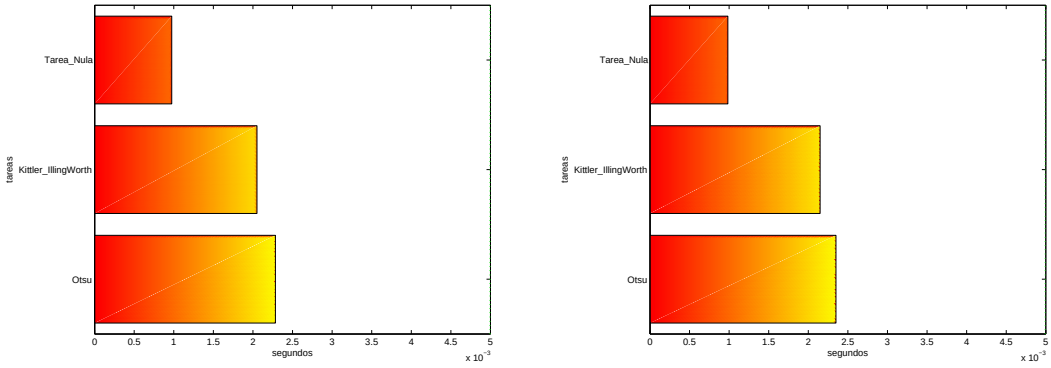


Figura 9.16: Umbrales globales sobre hojalata (izquierda) y aluminio (derecha). Se incluye el costo del calculo de histograma, histograma normalizado, la obtención del umbral y la operación de binarizado cuyo resultado se muestra en la figura 9.18.

tica en cada superficie ofrece muy buenas perspectivas. En la figura 9.18 una muestra de los excelentes resultados que se obtienen. Sin embargo el carácter iterativo-convergente de estos algoritmos hace que su consumo en tiempo resulte imprevisible y pueda variar de una lata a otra. En otras palabras el número de operaciones depende de cuan pronto encuentre el valor de umbral óptimo. Ver por ejemplo la figura 9.17. Esto no ocurre con el resto de los algoritmos presentados en esta sección cuyo consumo es bastante estable. Con todo no parece un problema insalvable y habría que examinar con qué frecuencia ocurre. Ver las extensiones en 10.2 al respecto.

Umbrales Locales Tal como puede verse en la figura 9.19 el cálculo de umbrales locales resulta demasiado lento para un reconocimiento lata a lata salvo quizás Bernsen. En las figuras 9.20, 9.21, y 9.22 pueden observarse los resultados del umbralizado.

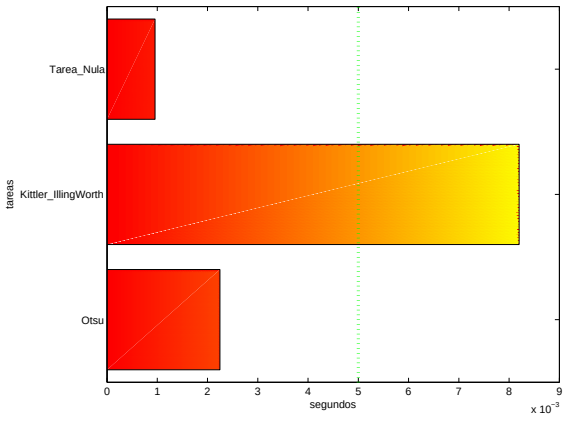


Figura 9.17: Kittler lento. Para las mismas condiciones y el mismo algoritmo se cuadriplica el consumo.

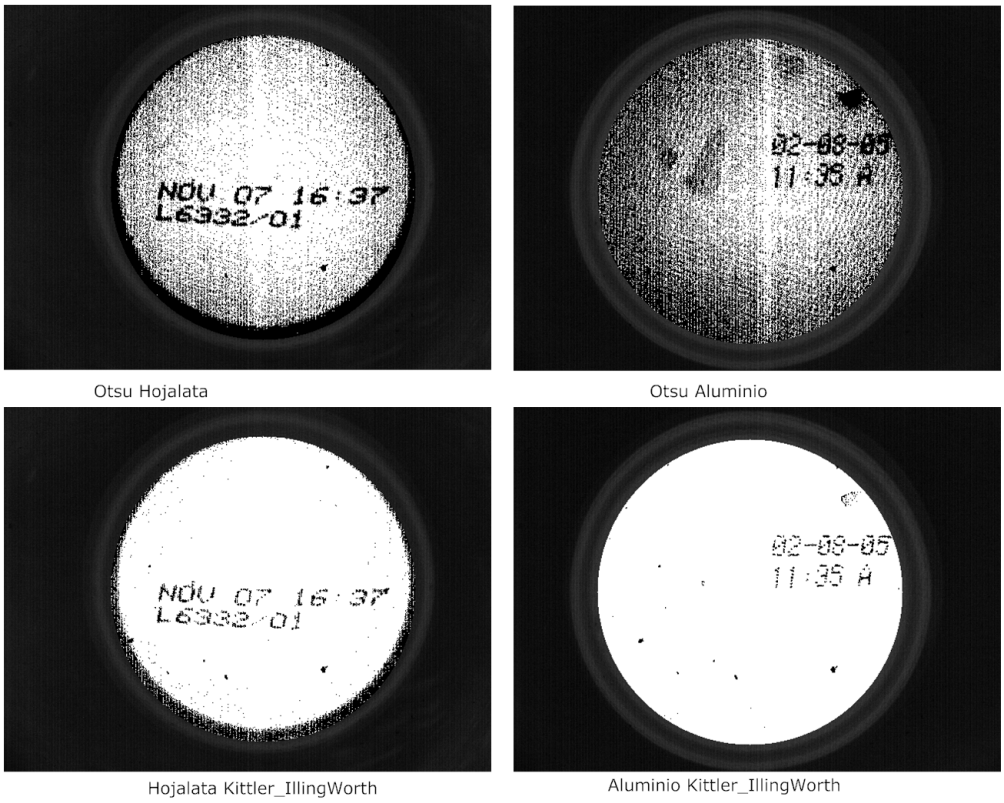


Figura 9.18: Umbralizado Global: Otsu y Kittler_IllinngWorth.

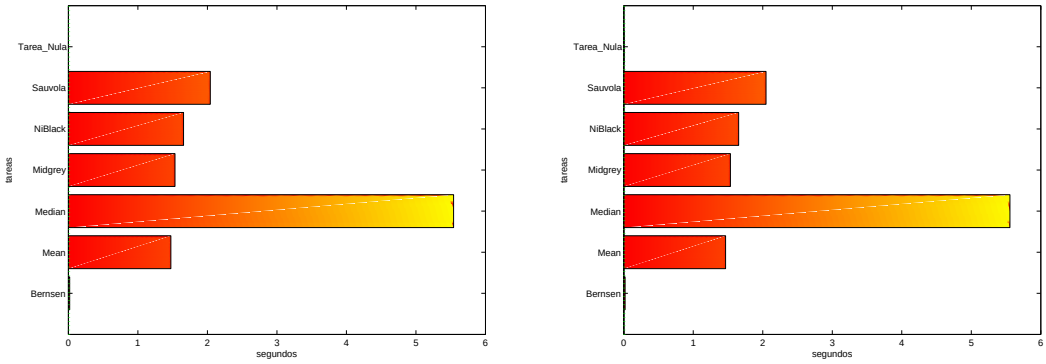


Figura 9.19: Umbrales locales sobre hojalata (izquierda) y aluminio (derecha).

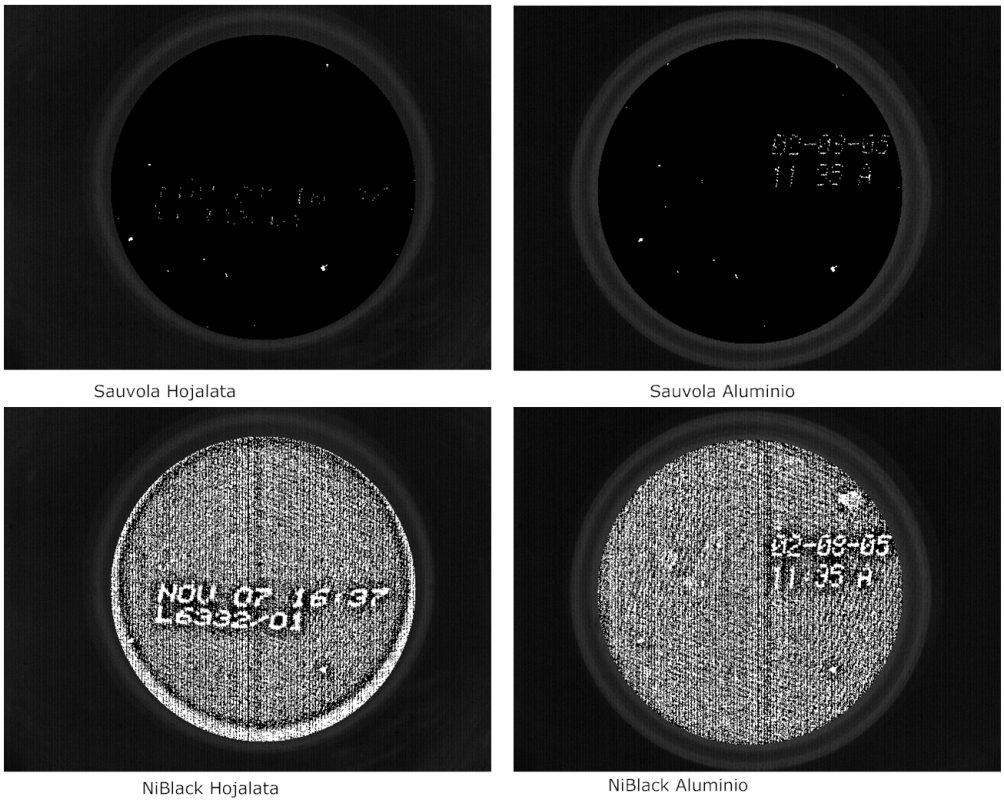


Figura 9.20: Umbralizado local: Sauvola, NiBlack

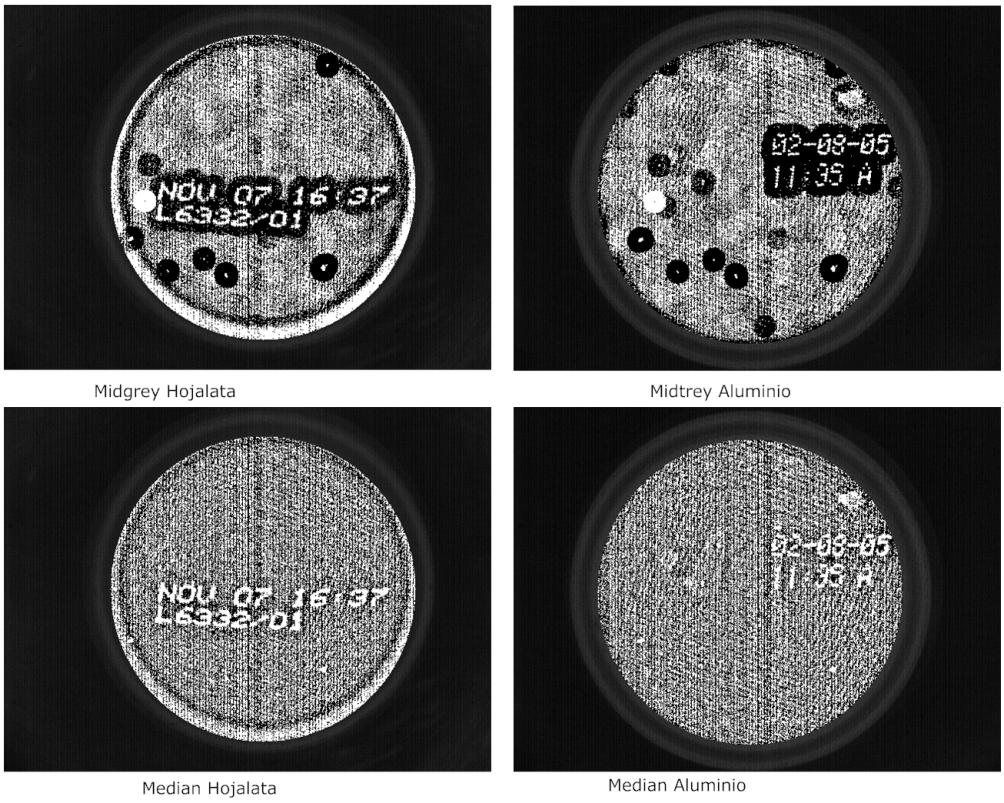


Figura 9.21: Umbralizado local: Midgrey, Median

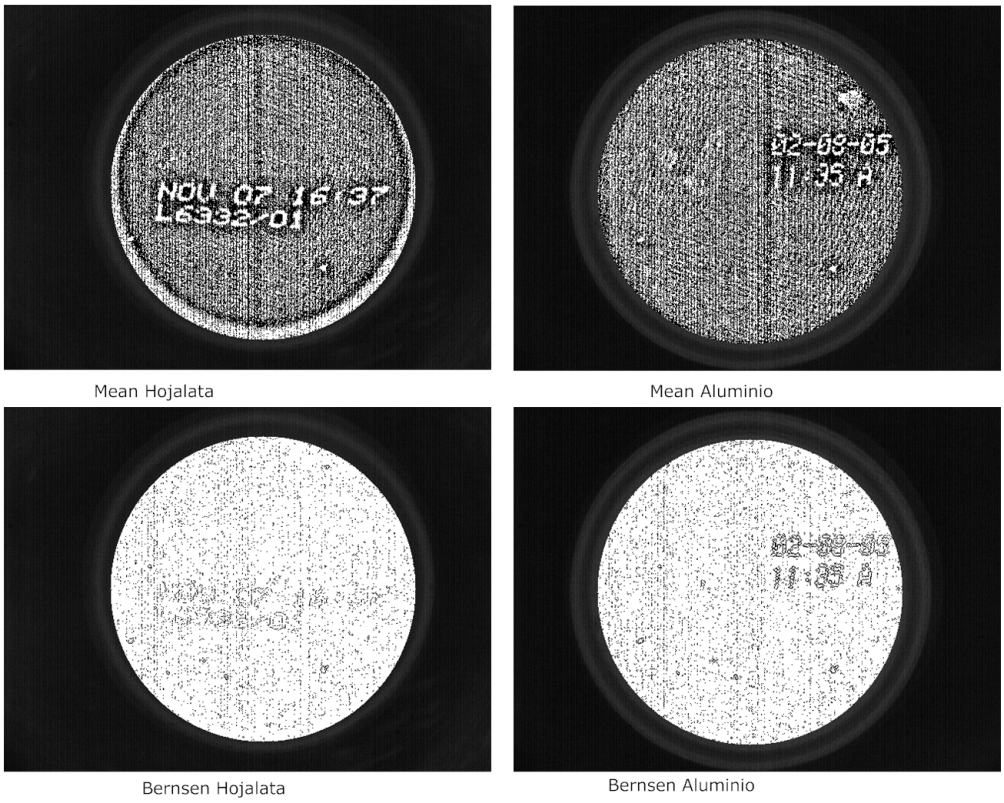


Figura 9.22: Umbralizado local: Mean, Bersen

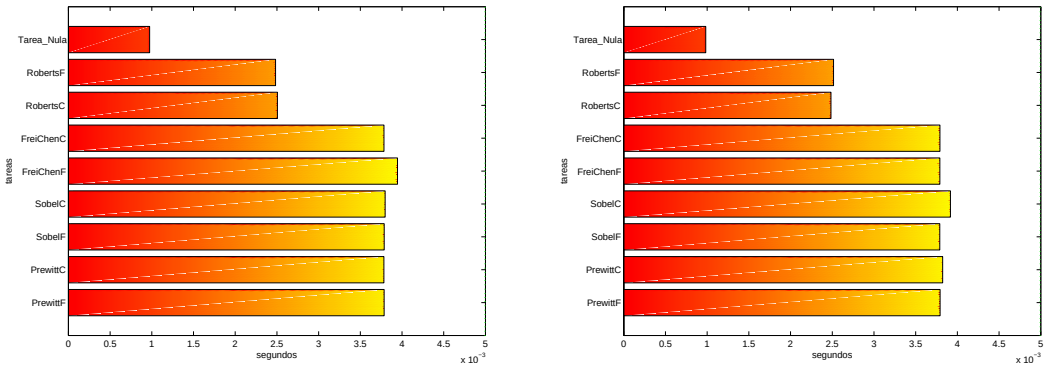


Figura 9.23: Detectores de bordes sobre hojalata (izquierda) y aluminio (derecha).

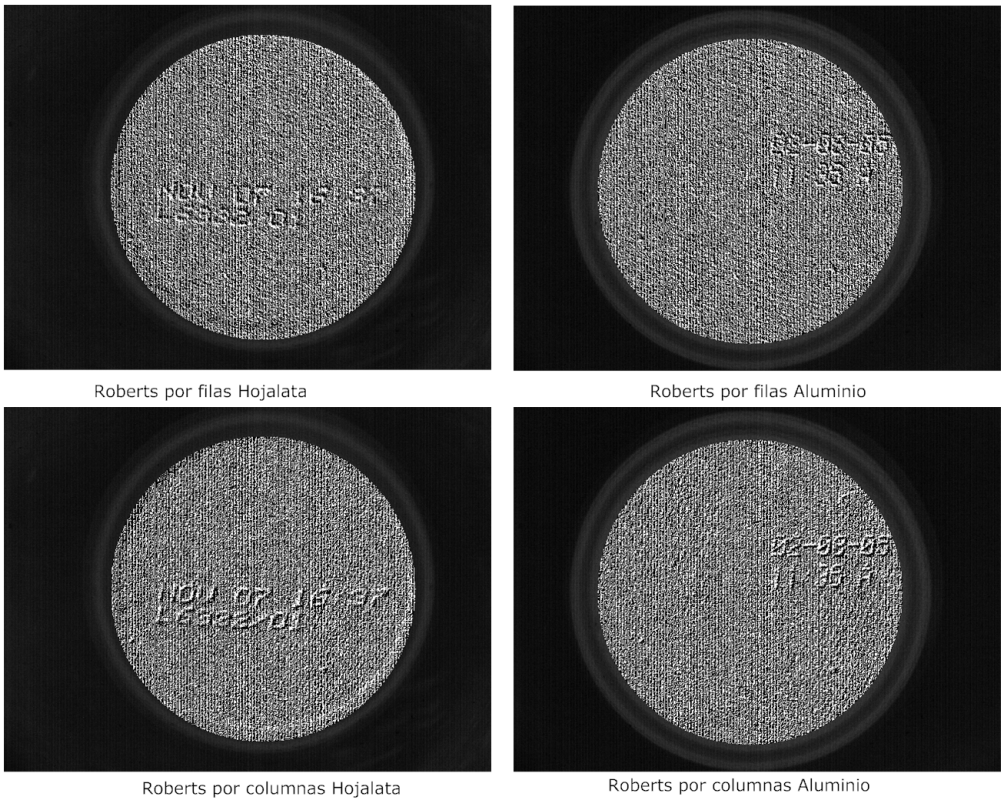


Figura 9.24: Detectores de bordes: Roberts

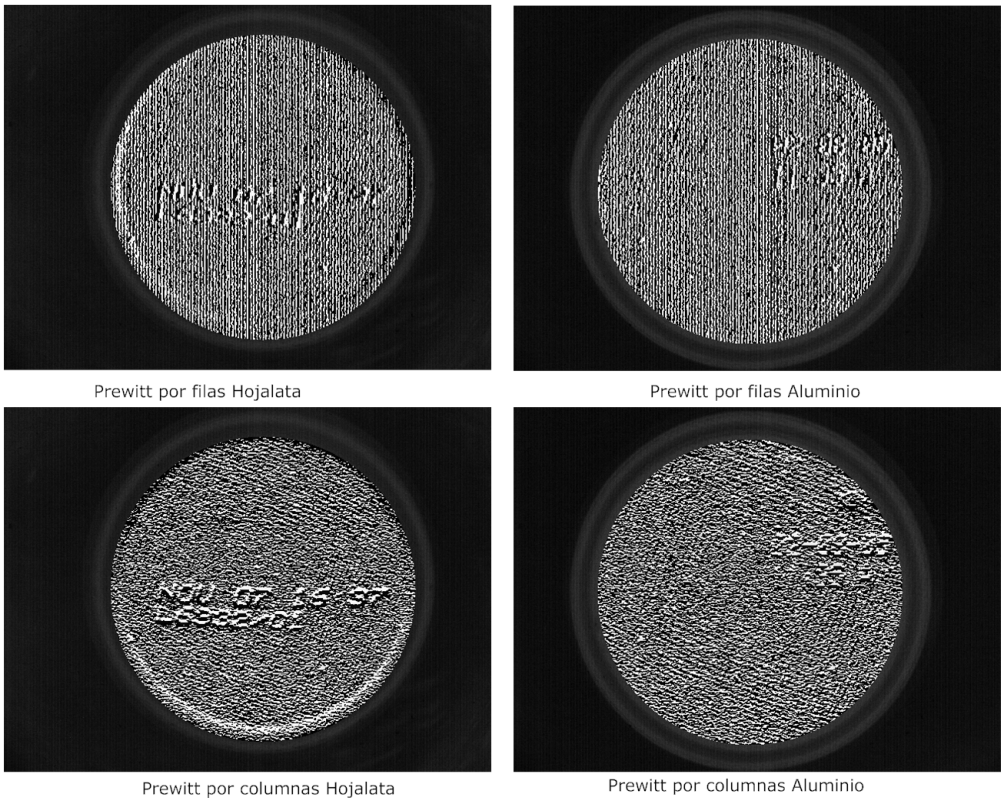


Figura 9.25: Detectores de bordes: Prewitt

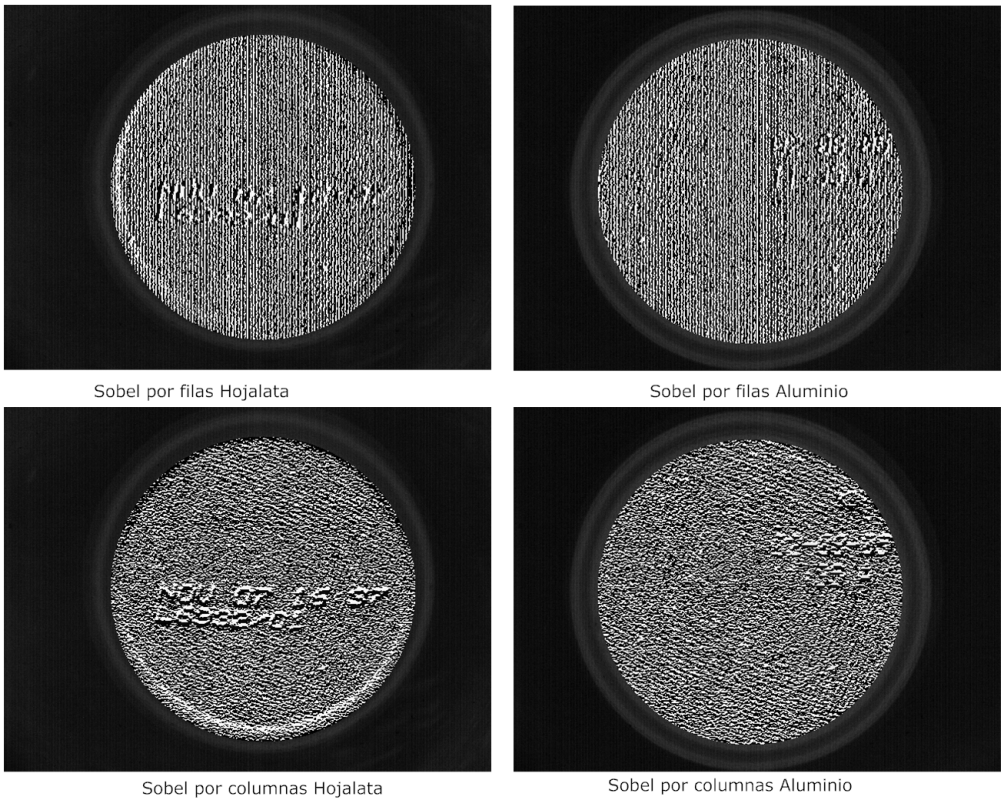


Figura 9.26: Detectores de bordes: Sobel

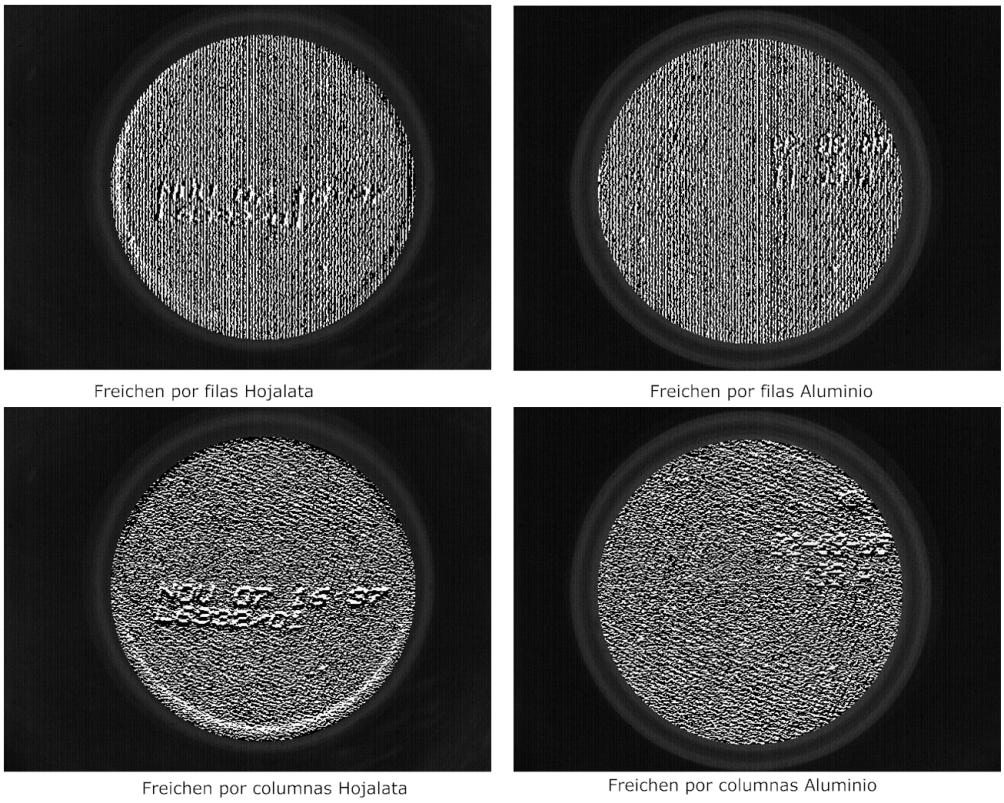


Figura 9.27: Detectores de bordes: Frei-Chen

9.4. Asociación de tinta

El asociador de tinta funciona mejor entre menor sea la inclinación del código. Los fragmentos que tienden a elongarse de forma oblicua también tienden a descomponerse en mayor número de fragmentos. Obsérvese en la figura 9.28 obtenida con la parametrización del cuadro B.3 como las líneas oblicuas suelen estar más fragmentadas. Eso puede ser originado por una resolución asimétrica ($640x480$) pero la causa principal estaría en el sistema de impresión. Ignoramos si es una limitación técnica o de configuración.

Figura 9.28: Diferentes inclinaciones bajo el asignador de tinta. (☼)

9.5. División de bandas

He aquí una colección de ejemplos de códigos a diferentes inclinaciones con parámetros en el cuadro B.4: Figuras 9.29, 9.30, 9.31, 9.32, 9.33, 9.34 y 9.35.

Como se ha dicho el método es efectivo siempre que no se le exija más de lo que pueda dar. Esto es, inclinaciones "severas" dañan el rendimiento general del método. Por otra parte la aparición de concavidades y la desaparición de espacios de interlineado son otras variables a tener en cuenta.

Creemos que este resultado puede mejorarse si añadimos a las direcciones

preferentes Oeste(O) y Este(E) actuales las direcciones Noroeste(NO), Suroeste(SO), Noreste(NE), Sureste (SE). Esto implicaría agregar algún mecanismo que permita dilucidar si el ángulo que forma con la vertical es positivo o negativo.

9.6. Agrupamiento de fragmentos

El agrupamiento muestra un excelente comportamiento tanto en tiempo (menos del 1 % del total) como en calidad del resultado obtenido. Es interesante subrayar que fácilmente el agrupamiento es correcto aunque la división por bandas no lo sea. Eso es así porque lo esencial para el agrupamiento es que fragmentos de un mismo carácter no estén asociados a bandas distintas, no que la bandas en sí sean correctas.

Con los parámetros de el cuadro B.5 se agrupa el código como se muestra en la figura 9.36.

9.7. Validación

9.7.1. Clasificadores contendientes

Como se revela en la figura 9.11 el validador de caracteres TM ocupa la cuarta posición en el consumo de tiempo (13 % del total).

Para un estudio comparativo del costo computacional de la clasificación se ha llevado a cabo una serie intensiva de test enfrentando cuatro clasificadores con el implementado para MONICOD.

- MONICOD.
  - Descrito en este mismo documento.
  - Configuración: Ver cuadro B.6 para TM (1), cuadro B.7 para TM (2) y cuadro B.8 para TM (3). **La diferencia entre las tres configuraciones estriba en el tamaño máximo (respectivamente 8, 4 y 1 morfologías en una familia morfológica) de la BFM**. Una revisión exhaustiva de todos los parámetros y su significado puede encontrarse en B.
  - Deshabilitada la capacidad de adaptar la base de morfologías en tiempo de clasificación.
  - En MONICOD el comparador de plantillas se utiliza como **validador**. Para este experimento se utilizará como **reconocedor**.
- KNN.
  - Descrito en 2.7.1.1.
  - Configuración: Ver cuadro B.9.

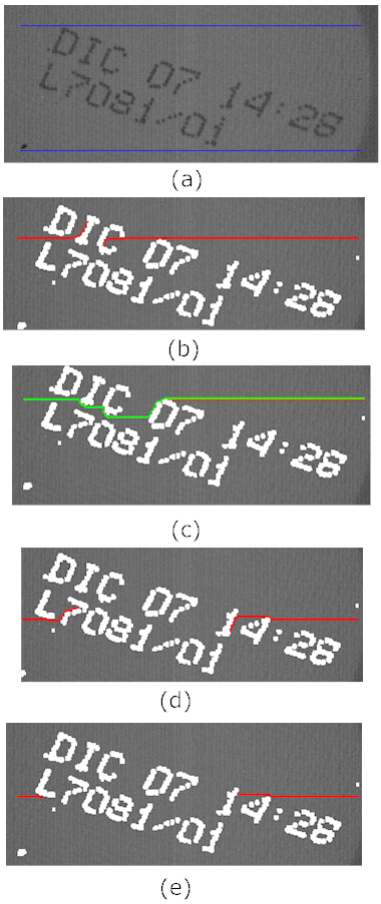


Figura 9.29: Una inclinación entre -30º/45º. Repetidos intentos de división no tienen éxito salvo en (c) pero la banda obtenida no es correcta. Por eso se reintenta en (d) y en (e). En (a) el método de filas evidentes. (☼)

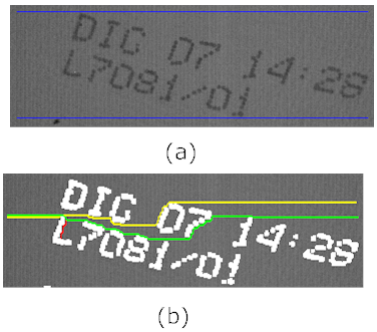


Figura 9.30: Una inclinación de -30 grados aproximadamente. De derecha a izquierda se consigue una división (en verde). En amarillo la división de izquierda a derecha que estaba en curso pero que fue cancelada en favor de la división más rápida. Ambas son incorrectas.(☼)

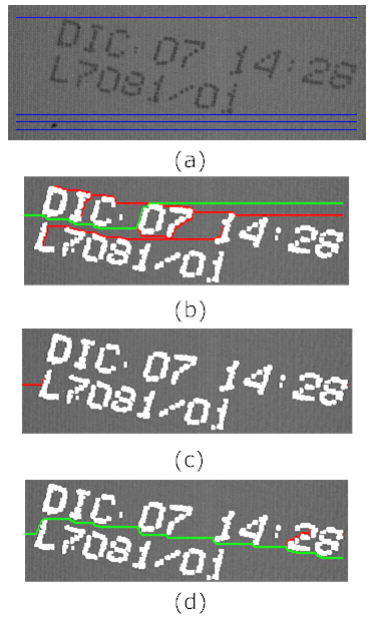


Figura 9.31: Una inclinación en torno a -10 grados. La división tiene éxito al tercer intento en (d). (☼)

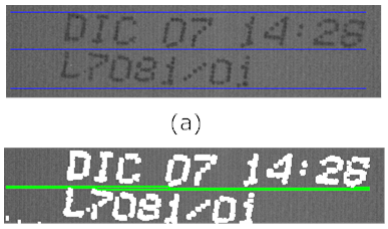


Figura 9.32: Una inclinación de 0 grados aproximadamente. Arriba con un umbral de tinta de 14 (una fila evidente). Abajo con un umbral de tinta de 16 (se tuvo que utilizar el *divisor* de bandas). Esto es una prueba del impacto del engrosamiento sobre la división de bandas. (☼)

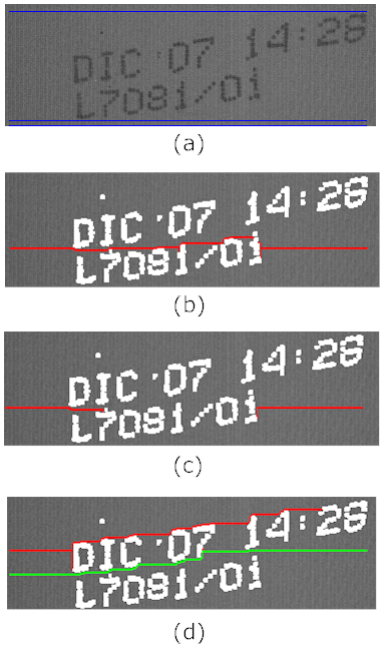


Figura 9.33: Una inclinación de 10/20 grados aproximadamente. (☼)

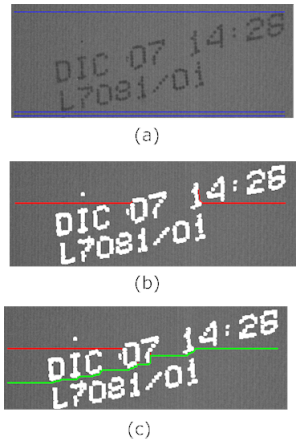


Figura 9.34: Una inclinación en torno a 20/30 grados. La división tiene éxito al segundo intento. (☼)

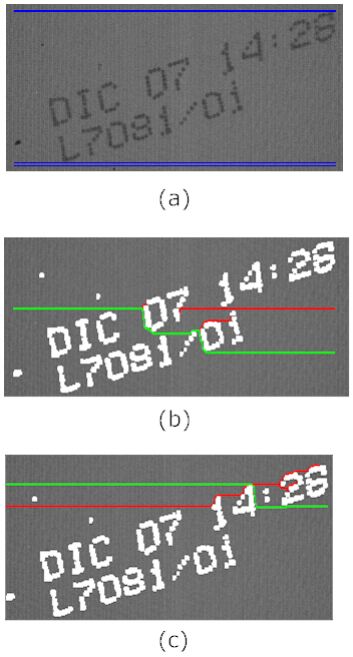


Figura 9.35: Una inclinación de mas de 30 grados. La división tiene "éxito" en los dos intentos aunque que en ambos el resultado es incorrecto. (☼)

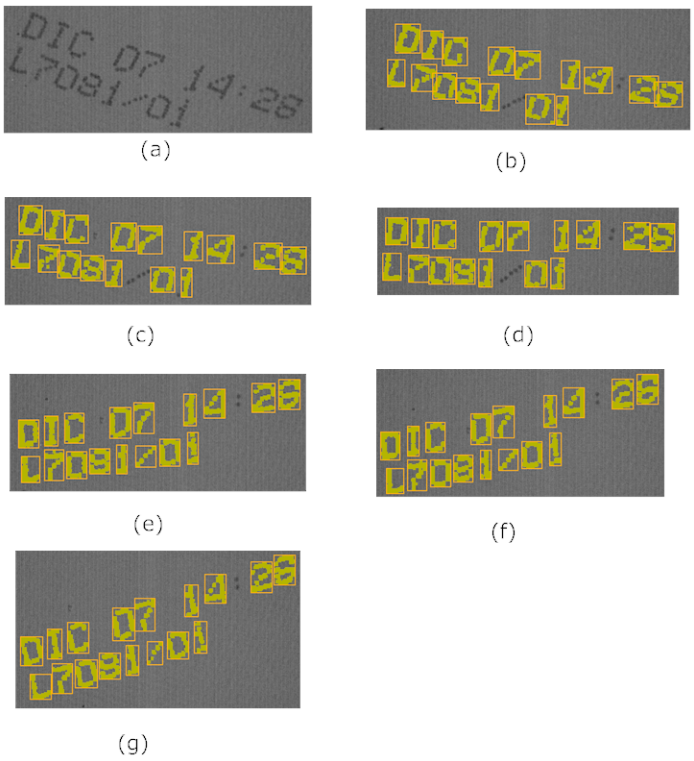


Figura 9.36: El mismo código sometido a diferentes inclinaciones. Salvo el caso (a), el resto de casos no presenta dificultades. (a) más allá de -30 grados. (b) poco menos de 20 grados. (c) en torno a 10 grados. (d) 0 grados aproximadamente (e) sobre 10 grados. (f) poco más allá de 15 grados. (g) sobre 30 grados. (☼)

- Implementado en [118].

- MLP con retropropagación.
  - Descrito en 2.7.2.4.
  - Configuración: Ver cuadro B.10.
  - Implementado en [119].

- RBF.
  - Descrito en 2.7.2.5.
  - Configuración: Ver cuadro B.11.
  - Implementado en [119].

;

- SVM.
  - Descrito en 2.7.3.
  - Configuración: Ver cuadro B.12.
  - Implementado en [120].

9.7.2. Extractor de características

- MONICOD
  - La propia plantilla extraída por MONICOD: Una matriz bidimensional $n \times m$ de enteros $(0, 1)$.
  - Configuración: Plantillas de entrada 35 x 35.

- KNN, MLP, RBF, SVM
  - Se ha utilizado los momentos de Hu (vector de características de 7 componentes) descrito en 2.6.2.1 aplicado sobre las plantillas expuestas en 9.37.
  - Configuración: Plantillas de entrada 35 x 35.

9.7.3. Condiciones

Las condiciones del experimento son las mismas que las precedentes.

- MONICOD
  - La implementación utilizada es exactamente la misma que la incorporada en MONICOD. Lo mismo se aplica al código de extracción de características.

9.7.4. Criterios

- Calidad
  - Recuento de los aciertos de clasificación.
  - Recuento de los fallos de clasificación.
- Tiempos
  - Tiempo de preentrenamiento. El tiempo invertido en la adquisición de la imagen (lectura de disco) y extracción de características para la clasificación.
  - Tiempo de preclasificación. El tiempo invertido en la adquisición de la imagen (lectura de disco) y extracción de características para la clasificación.
  - Tiempo de entrenamiento. Tiempo invertido en el entrenamiento.
  - Tiempo de clasificación. Tiempo invertido en la clasificación.

9.7.5. Procedimiento

- Se adquiere con MONICOD un conjunto base de entrenamiento de 8000 muestras (plantillas). 1000 muestras de cada clase.
- Se adquiere con MONICOD un conjunto de clasificación de 8000 muestras (plantillas). 1000 muestras de cada clase.
- Repetimos el experimento para los porcentajes 0.1, 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100.
  - Se construyen **diez** (10) subconjuntos de entrenamiento seleccionando de forma aleatoria el porcentaje dado desde el conjunto base de entrenamiento. Siempre se garantiza que haya el mismo número de muestras representante de cada clase.
  - Para cada subconjunto (en este caso uno):
    - Para cada clasificador:
      - Se extraen las características de cada plantilla del subconjunto adecuándolas a la entrada del clasificador.
      - Se entrena el clasificador con el subconjunto.
      - Se extraen las características de cada plantilla del conjunto de clasificación adecuándolas a la entrada del clasificador.
      - Se clasifica el conjunto de clasificación completo con el clasificador.
      - Se registran las estadísticas de la iteración (tiempo consumido en cada fase, aciertos, fallos).
  - Se calcula media con las estadísticas de los subconjuntos.

9.7.6. Plantillas utilizadas

En la figura 9.37 se muestran las plantillas utilizadas en los experimentos. Son caracteres de lata de hojalata extraídas por MONICOD.

Figura 9.37: Ocho plantillas. Corresponden a los caracteres '0', '1', '5', 'A', 'R', '9', 'M' y '8'.

9.7.7. Resultados

En las figuras de este apartado el azul indica **el número de aciertos**, *el verde oscuro señala* **el tiempo invertido por el extractor de características**, *el amarillo* **el tiempo de clasificación** *y en verde claro* **el tiempo de entrenamiento**.

Los resultados se ordenan según el tamaño creciente de los conjuntos de entrenamiento.

En las figuras 9.38 y 9.39 se entrena con el 0.1 % del conjunto de entrenamiento (8 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). Se aprecia que los tiempos de entrenamiento de MLP y RBF son muy superiores a los tiempos de SVM, KNN y TM. Esta brecha no sólo se mantendrá sino que se incrementará a medida que los conjuntos de entrenamiento sean mayores. En calidad de clasificación TM ofrece la mejor calidad de resultado, también TM prueba ser el clasificador más rápido en clasificación. También lo es en entrenamiento.

En las figuras 9.40 y 9.41 se entrena con el 1 % del conjunto de entrenamiento (80 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). La brecha entre los tiempos de entrenamiento de MLP y RBF se incrementa considerablemente de SVM, KNN y TM. En calidad de clasificación TM mantiene su resultado promedio y KNN y SVM lo igualan. Por otra parte la

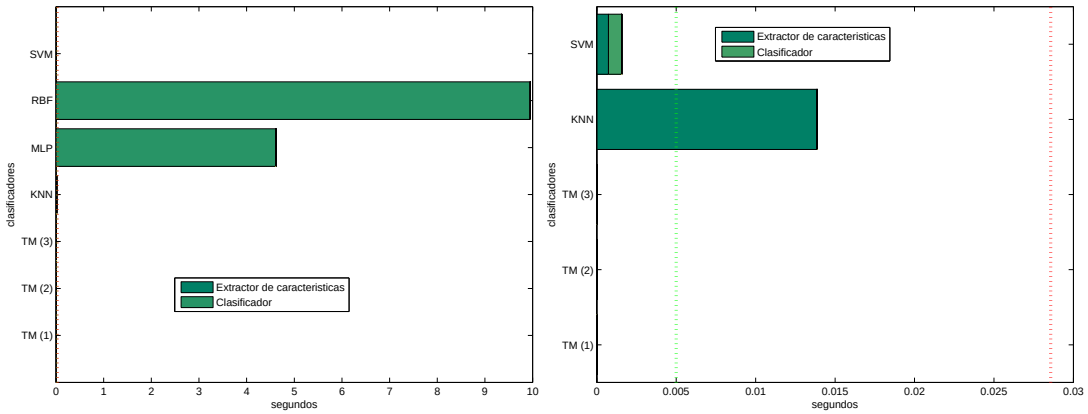


Figure 9.38: Tiempos 0.1 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

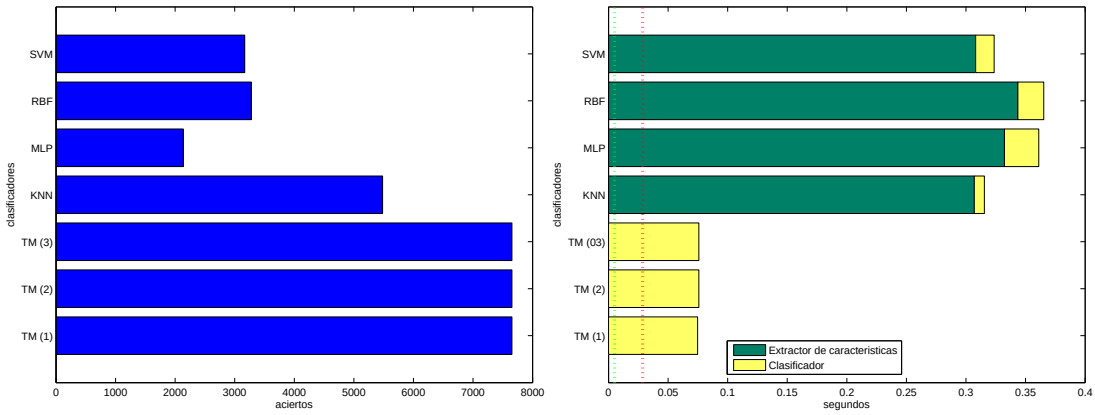


Figure 9.39: 0.1 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

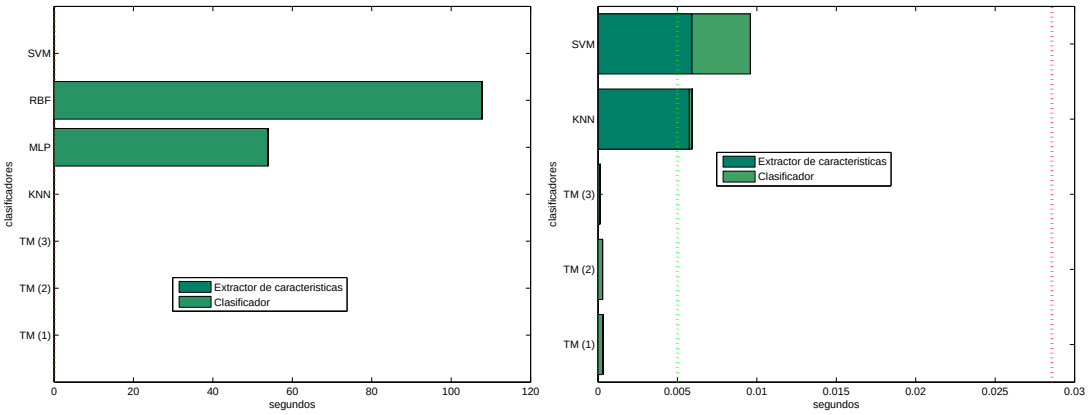


Figure 9.40: Tiempos 1 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

diferencia en tiempo entre TM y los otros clasificadores en tiempo de clasificación disminuye notablemente. Esto se explica porque hay muchas mas plantillas almacenadas en la BFM.

En las figuras 9.42 y 9.43 se entrena con el 5 % del conjunto de entrenamiento (400 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). La tendencia en las diferencias de los tiempos de entrenamiento se mantiene. SVM, KNN y TM presentan calidades en clasificación comparable. TM (en su configuración de capacidad BFM mayor) es ahora el clasificador más lento en este punto.

En las figuras 9.44 y 9.45 se entrena con el 10 % del conjunto de entrenamiento (800 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). La brecha en los tiempos de entrenamiento entre las redes neuronales y los demás clasificadores se incrementa. TM mantiene su calidad en clasificación y vuelve a ser el más rápido. En este punto el tamaño de la BFM se estabiliza, y así su consumo de tiempo.

En las figuras 9.46 y 9.47 se entrena con el 20 % del conjunto de entrenamiento (1600 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). Igual tendencia en los tiempos de entrenamiento. La calidad de la clasificación TM no se resiente pero tampoco aumenta. Como para los demás clasificadores la representatividad del conjunto de entrenamiento es esencial, pero TM es más sensible, y parece incapaz de mejorar su resultado.

En las figuras 9.48 y 9.49 se entrena con el 30 % del conjunto de entrenamiento (2400 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). KNN consigue una mejor calidad de clasificación que TM. Sin embargo computacionalmente KNN es consideramente más lento que los demás clasificadores.

En las figuras 9.50 y 9.51 se entrena con el 40 % del conjunto de entrenamiento (3200 muestras). Se clasifica con el 100 % del conjunto de clasificación

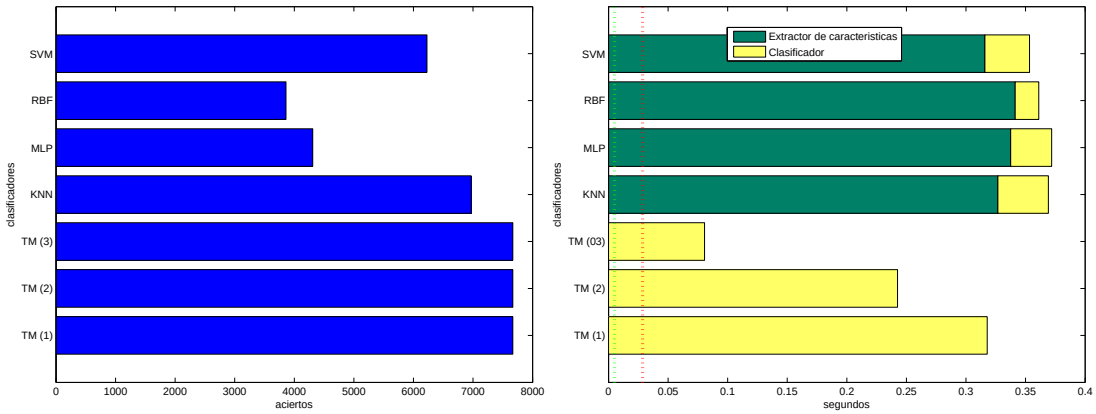


Figure 9.41: 1 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

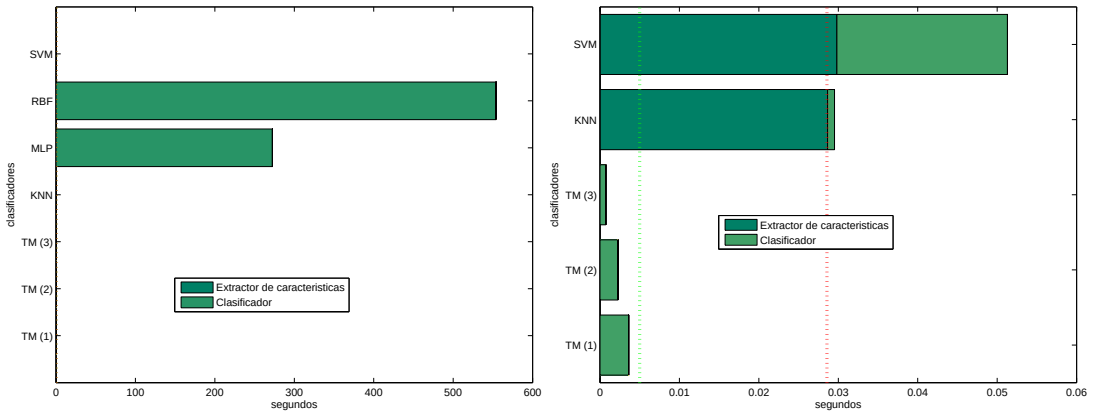


Figure 9.42: 5 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

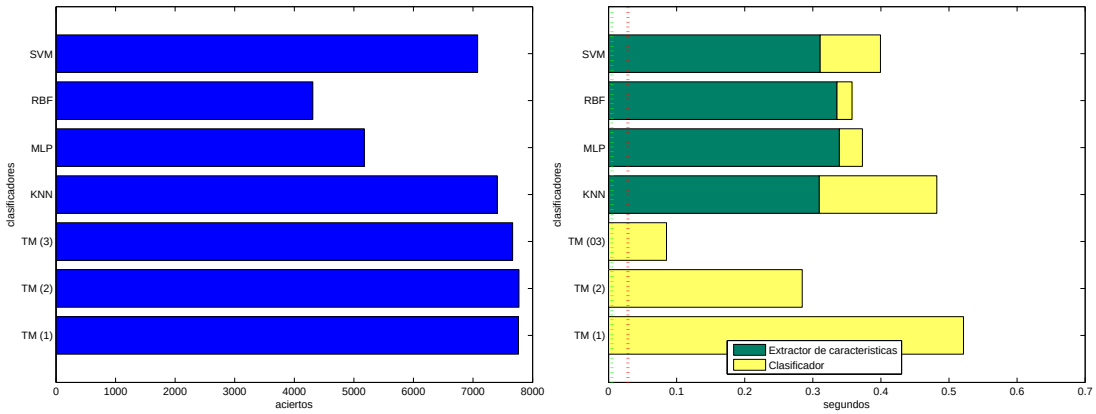


Figure 9.43: 5 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

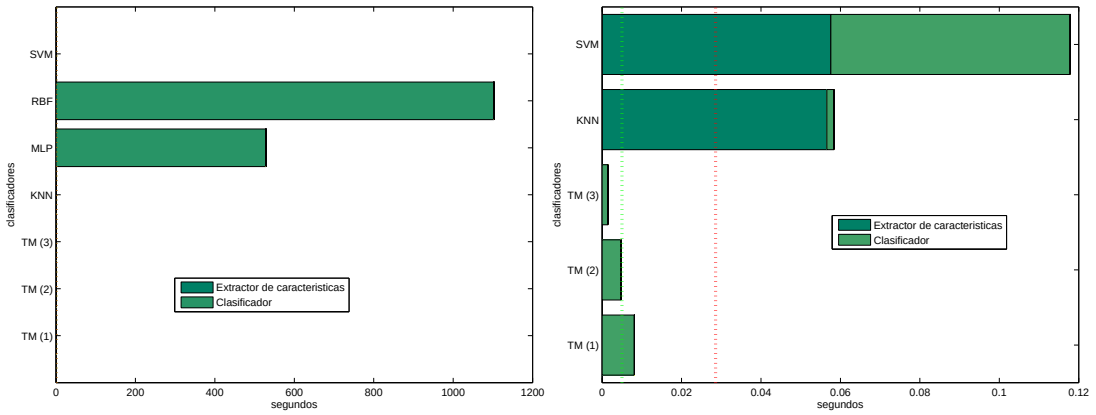


Figure 9.44: 10 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

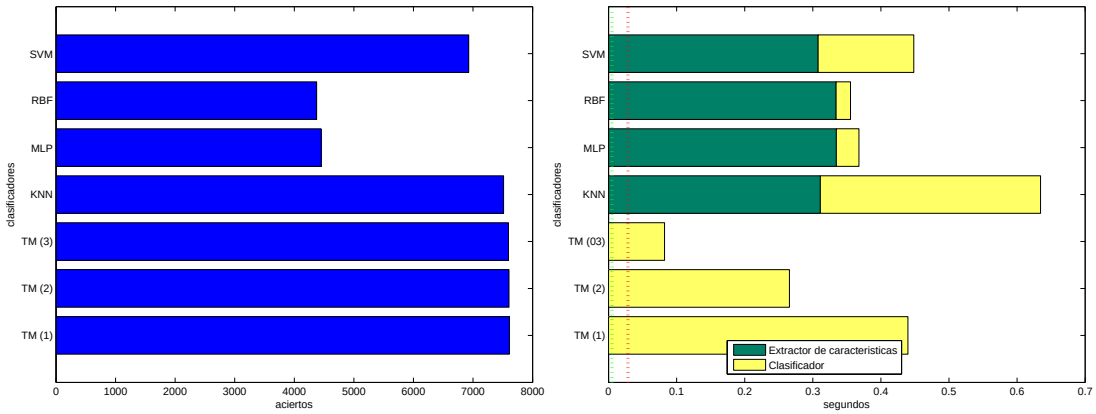


Figure 9.45: 10 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

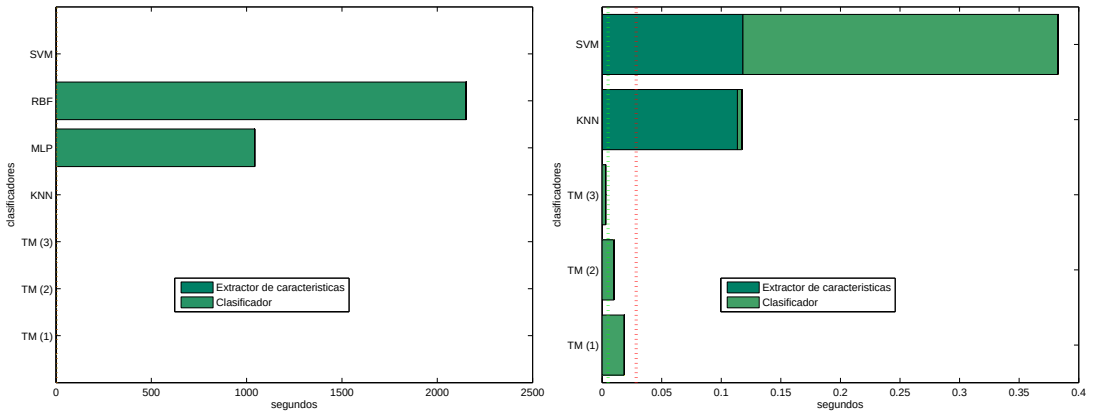


Figure 9.46: 20 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

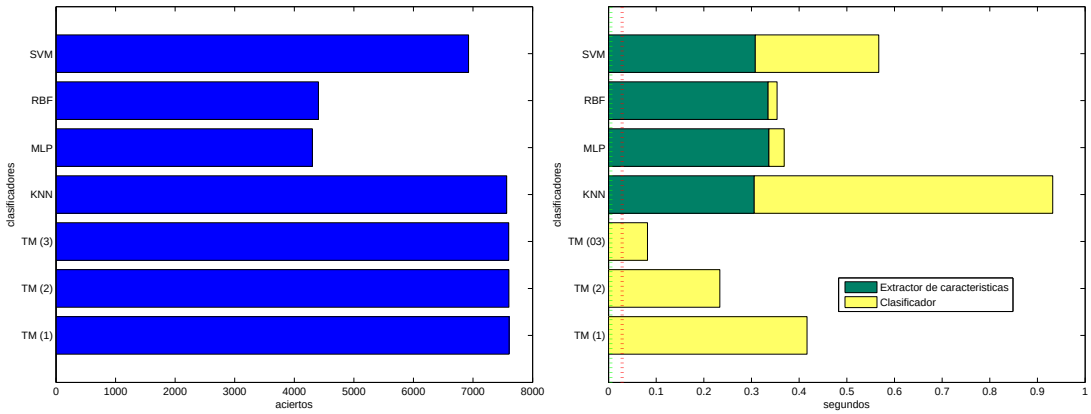


Figure 9.47: 20 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

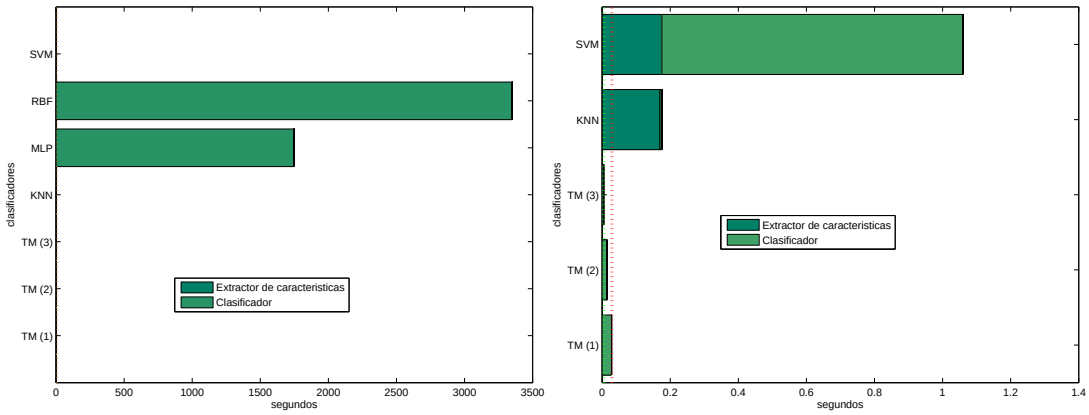


Figure 9.48: 30 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

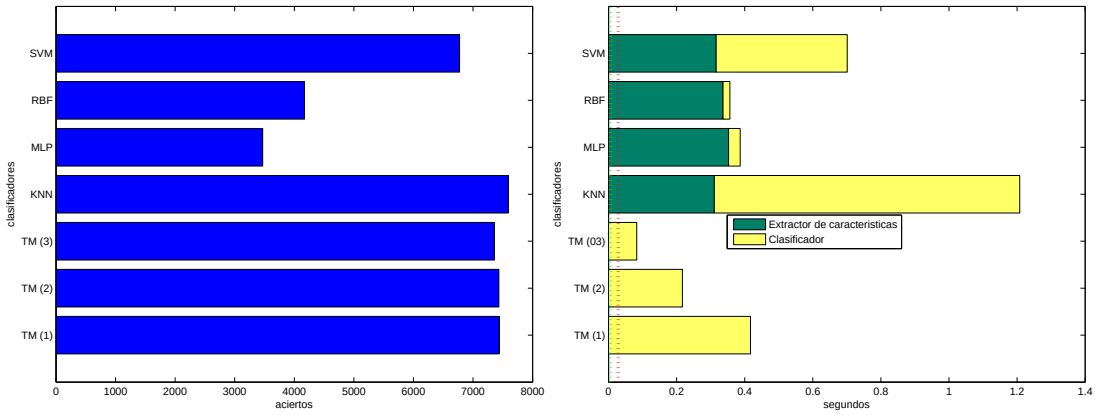


Figure 9.49: 30 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

(8000 muestras). Creemos que los efectos del sobreentrenamiento empiezan a notarse en este punto. TM no es capaz de mejorar sus números. El peso de la mayoría es excesivo. Los otros clasificadores mantienen estables, probablemente una señal reconocible de que su configuración es mejorable.

En las figuras 9.52 y 9.53 se entrena con el 50 % del conjunto de entrenamiento (4000 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). TM parece ofrecer de forma consistente un comportamiento al menos tan bueno como KNN.

En las figuras 9.54 y 9.55 se entrena con el 60 % del conjunto de entrenamiento(4800 muestras). Se clasifica con el 100 % del conjunto de clasificación(8000 muestras). En ocasiones TM logra aventajar a KNN como puede verse. Este resultado es posible porque TM ignora intencionadamente los pequeños detalles y es poco restrictivo al asociar una forma con una familia morfológica.

En las figuras 9.56 y 9.57 se entrena con el 70 % del conjunto de entrenamiento (5600 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). El costo en tiempo de KNN se incrementa al contar con nuevos vecinos para comparar. Los conjuntos de entrenamientos ámplios no benefician a KNN (sobreentrenamiento). Probablemente una mejor definición de distancia para el problema incremente la calidad de KNN.

En las figuras 9.58 y 9.59 se entrena con el 80 % del conjunto de entrenamiento (6400 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). No se detectan cambios en la tendencia descrita.

En las figuras 9.60 y 9.61 se entrena con el 90 % del conjunto de entrenamiento (7200 muestras). Se clasifica con el 100 % del conjunto de clasificación (8000 muestras). Los aciertos en clasificación mejoran de forma apreciable pero a costa de un entrenamiento intensivo.

En las figuras 9.62 y 9.63 se entrena con el 100 % del conjunto de entrenamiento (8000 muestras). Se clasifica con el 100 % del conjunto de clasificación

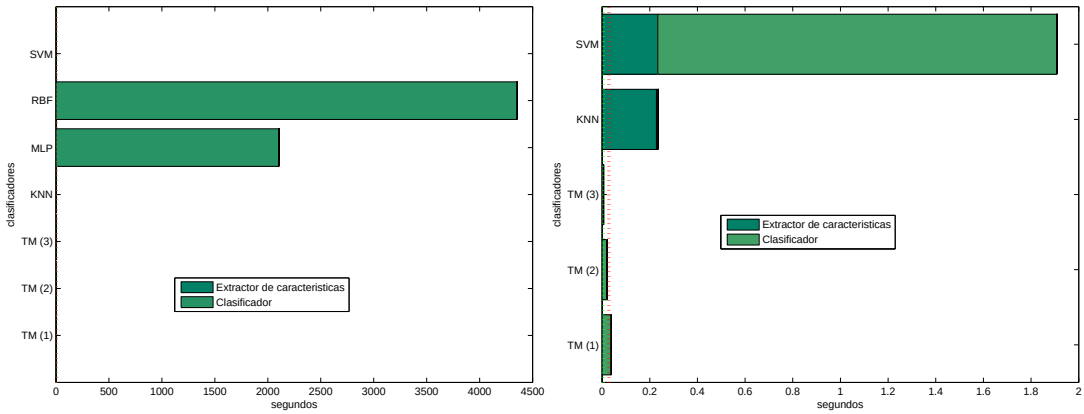


Figure 9.50: 40 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

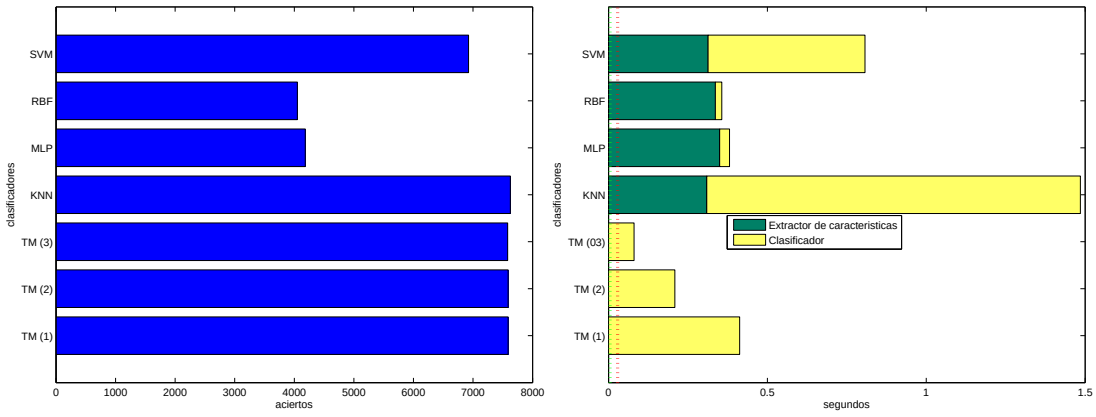


Figure 9.51: 40 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

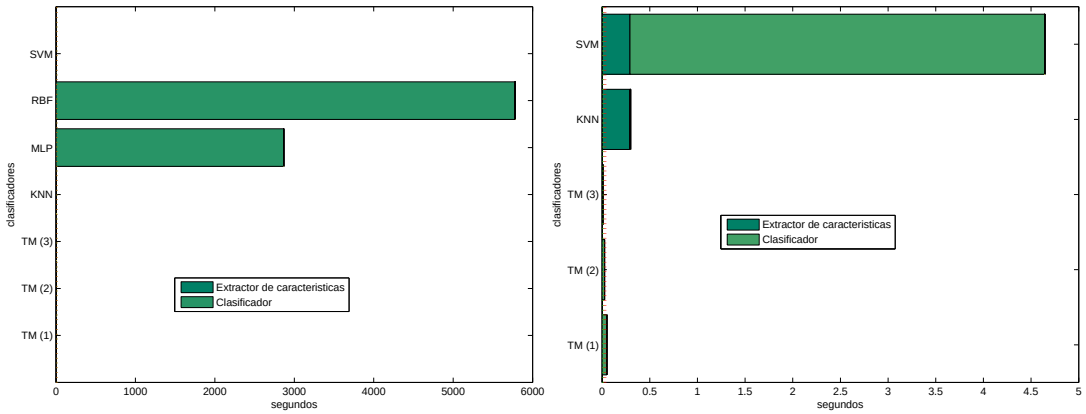


Figure 9.52: 50 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

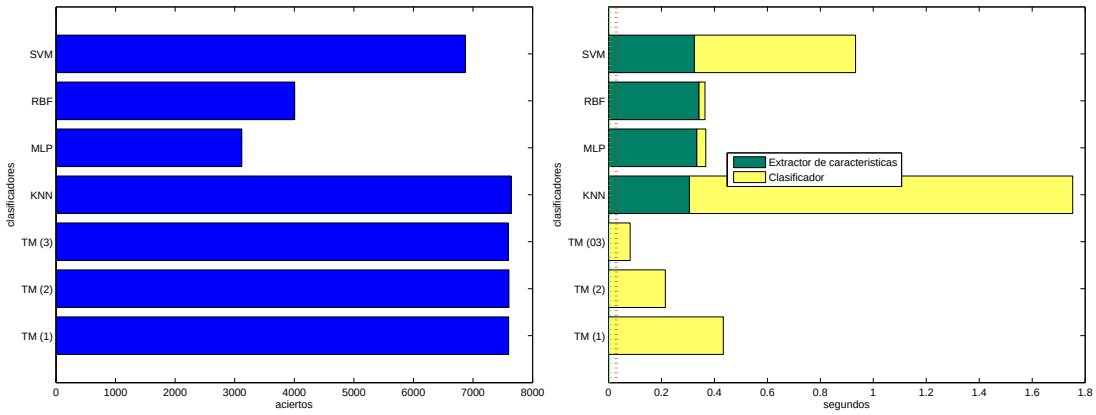


Figure 9.53: 50 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

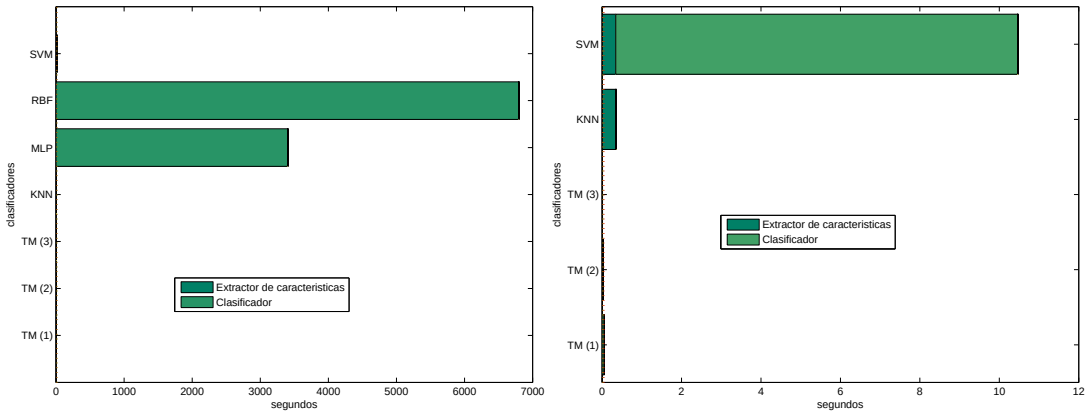


Figure 9.54: 60 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

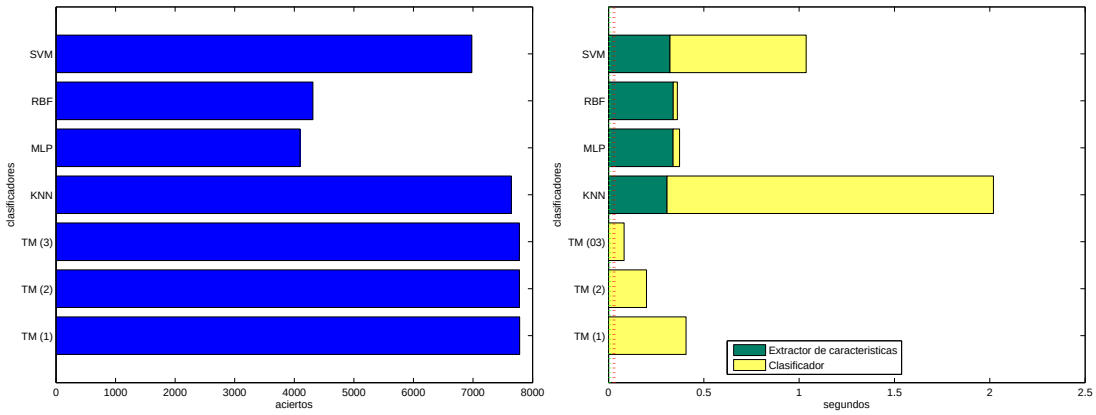


Figure 9.55: 60 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

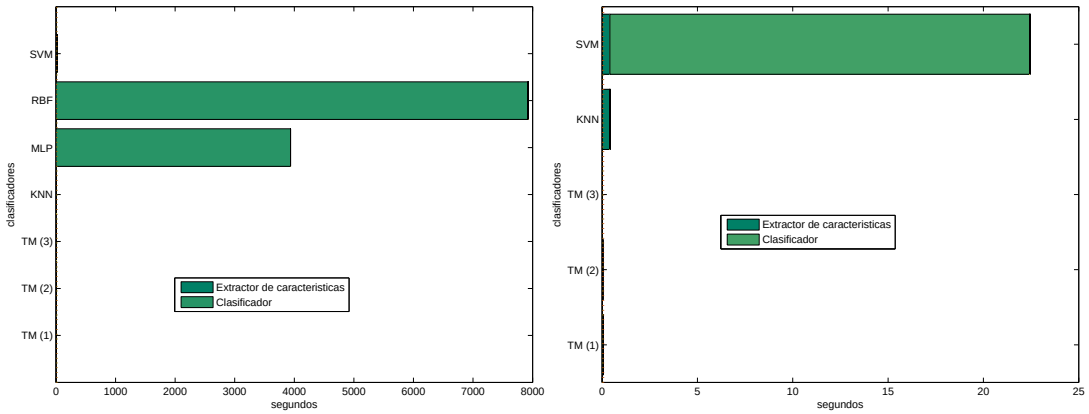


Figure 9.56: 70 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

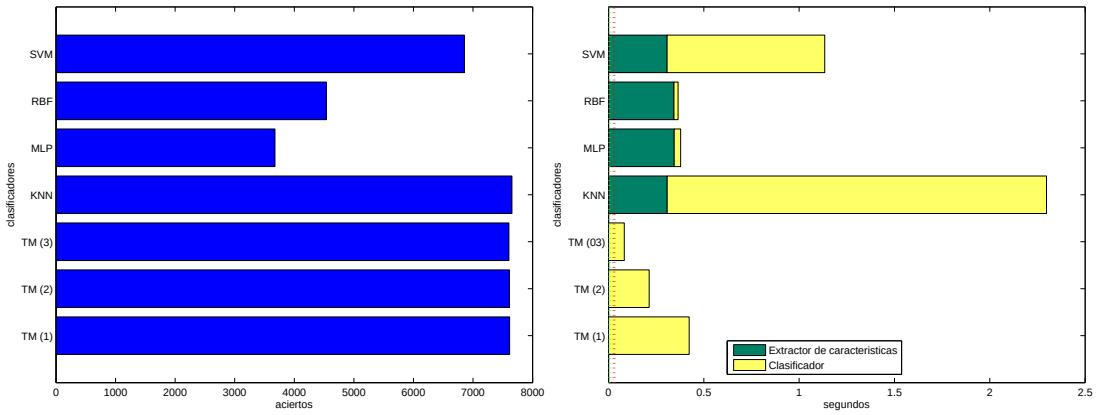


Figure 9.57: 70 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

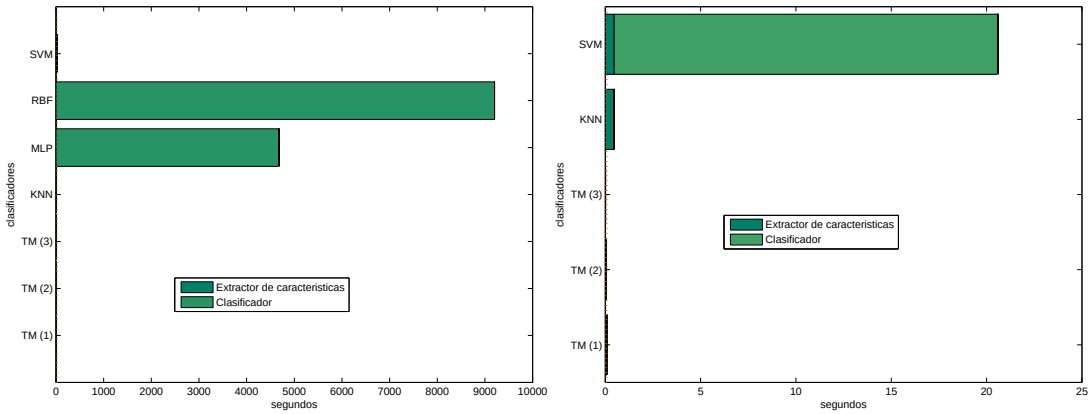


Figure 9.58: 80 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

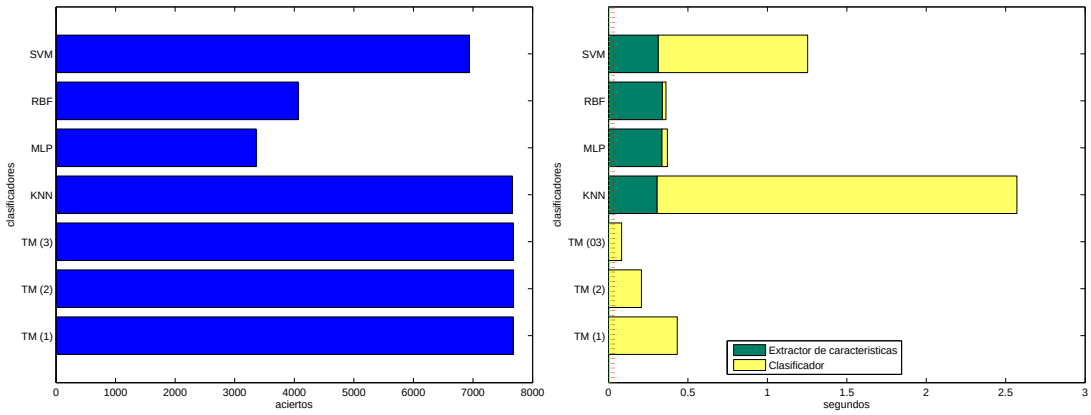


Figure 9.59: 80 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

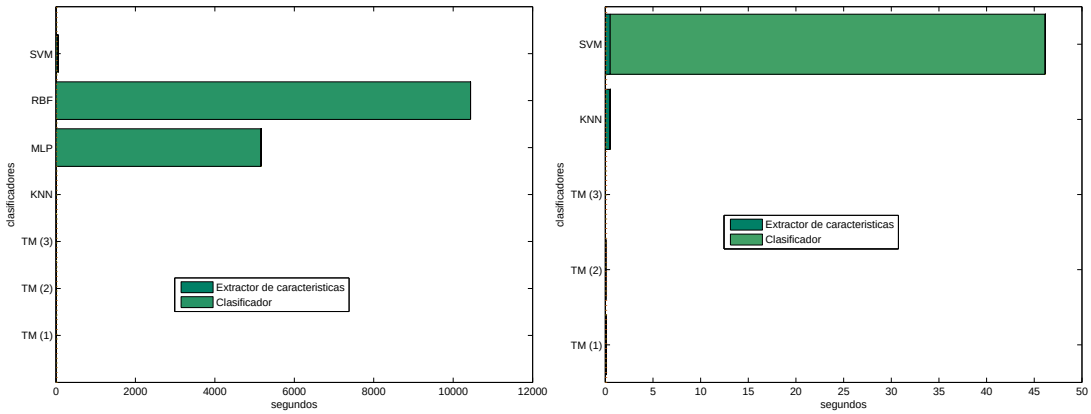


Figure 9.60: 90 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

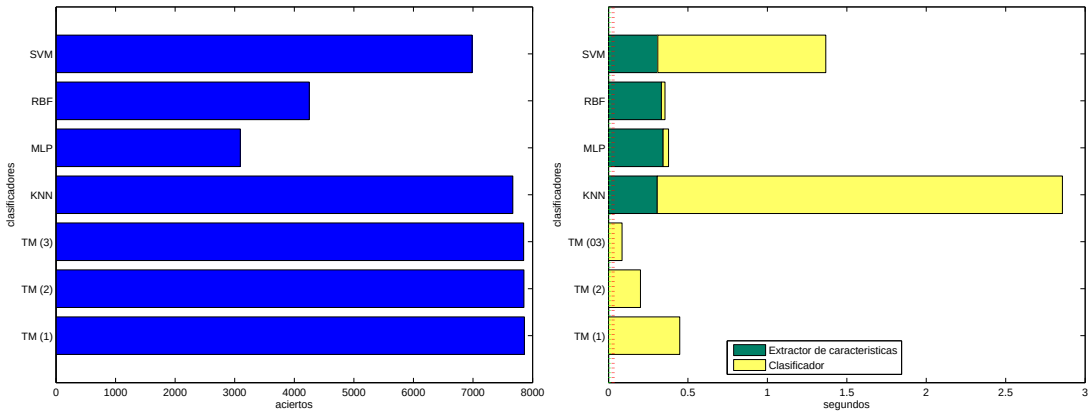


Figure 9.61: 90 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

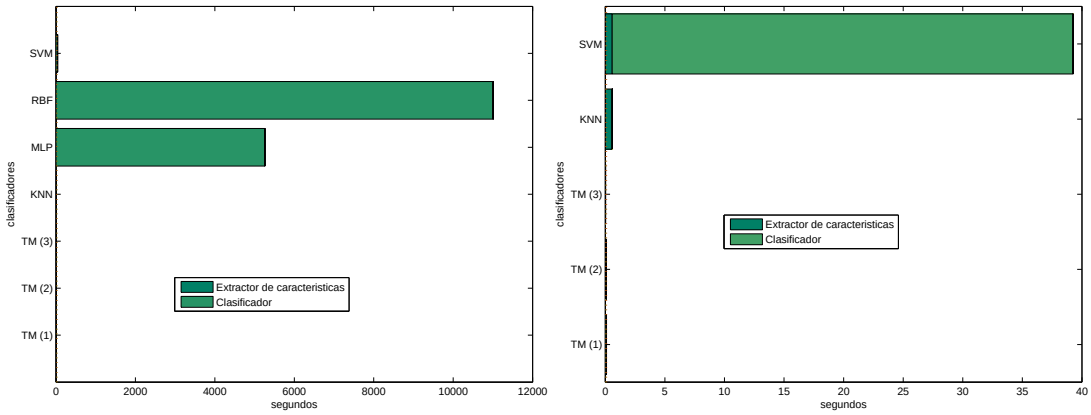


Figure 9.62: 100 % de 8000. Tiempos de entrenamiento (TODOS) izquierda, tiempos de entrenamiento (solo SVM, KNN y TM) derecha.

(8000 muestras). Los tiempos de entrenamiento MLP y RBF son insostenibles para un ciclo diario. TM es el más rápido en entrenamiento. En clasificación MLP Y RBF son los clasificadores más rápidos (en tiempo neto), pero para los parámetros utilizados no ofrec,e el mejor resultado. KNN ha resultado ser el clasificador más lento lo que no es sorprendente.

9.7.8. Consideraciones y algunas conclusiones

8000 caracteres es el texto necesario para 400 latas a 20 caracteres por lata. A 35 latas por segundo tenemos 11.4 segundos de procesamiento.

- En general TM ofrece una buena calidad en el reconocimiento así como un buen ratio velocidad/calidad de reconocimiento.
- El aprendizaje de TM tiene el menor costo, seguido de KNN. TM se limita a incluir nuevas plantillas en la base de conocimiento donde corresponde (el número de comparaciones necesarios es bastante bajo). Otros métodos como MLP y RBF requieren un proceso iterativo de reajuste de pesos [En nuestro ejemplo se han utilizado 50000 iteraciones para entrenarlos].
- La calidad de acierto de los contendientes de TM (salvo KNN) es decepcionante, pero es probable que mejore con la generación de vectores de características superiores. Estos vectores de características superiores se construirían presumíblemente con métodos más sofisticados que consumirían más tiempo.
- Con conjuntos de entrenamiento pequeños (hasta el 10 %) TM tiende a ser el clasificador neto más lento en comparación con el tiempo neto[5] de

[5] Es decir el tiempo invertido puramente en la clasificación sin considerar un proceso previo de extracción de características de la forma.

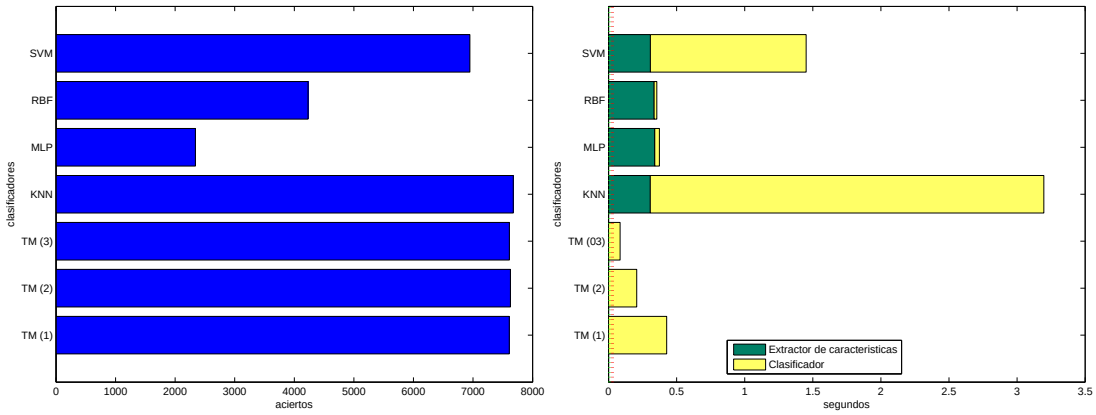


Figure 9.63: 100 % de 8000. Calidad de la clasificación izquierda, velocidad neta y bruta en la clasificación derecha.

clasificación de los otros clasificadores, pero la necesidad de un proceso de extracción de características (que no necesita TM) le da una ventaja bruta[6].

- Operar con pocas muestras de entrenamiento no es realista: La calidad de resultado resulta demasiado dependiente de las muestras concretas utilizadas para aprender y se pierde generalización.
- TM no utiliza extractor de características. En su lugar trabaja directamente con la plantilla. Los otros clasificadores lo utilizan (en nuestro experimento se ha usado los populares momentos de Hu) para reducir el tamaño del vector de entrada de ahí que se haya contabilizado en el computo de tiempo total.
- Dado que TM no emplea extractor de características su capacidad de generalizar es limitada con respecto a los otros clasificadores que si operan sobre características. Debido a esta limitación lo hace más dependiente de la representatividad del conjunto de entrenamiento que otros clasificadores. Sin embargo esto no impide que en los experimentos exhiba un comportamiento excelente.
- MONICOD no fue concebido como un sistema de reconocimiento, sino de validación, por tanto su costo es dependiente del número máximo de morfologías que puede haber en una familia morfológica y no, como en este caso, del número total de morfologías en la BFM.
- En el experimento el tamaño de la BFM no ha resultado decisivo. La razón subyace en la escasa varianza que presenta el conjunto de muestras

[6] Es decir el tiempo invertido en la clasificación neta mas un proceso previo de extracción de características de la forma.

utilizados y en la consistencia del extractor de características descrito en el capítulo 7.

- Una estrategia no utilizada aquí que es posible en TM es la posibilidad de abandonar la búsqueda si la familia morfológica encontrada hasta este momento ya ofrece morfologías suficientemente parecidas (por encima del umbral de aceptación).
- La estrategia complementaria a esta, que es posible en TM, es la posibilidad de abandonar la exploración de la familia morfológica actual si las morfologías encontradas hasta el momento son demasiado distintas.
- Debería ser posible mejorar la calidad de TM forzando una mayor variedad en la BFM. El tiempo de computo no empeoraría porque el número de morfologías se mantendría fijo.

Capítulo 10

Conclusiones y Extensiones

"*Donde todo termina y todo empieza*"

Como el propio nombre del capítulo indica se sigue el ya clásico formato de dos partes.

En primer lugar una exposición de las conclusiones agrupadas en fases, y a continuación una selección de extensiones y mejoras posibles.

10.1. Conclusión

Se ha presentado a MONICOD, un sistema de validación de códigos de latas en tiempo real capaz de tratar 200 frames por segundo, validando 35 latas por segundo. No disponemos de una línea de producción que trabaje con esa cadencia pero en la sección 9.2 se lleva a cabo una simulación a 560 caracteres por segundo con 16 caracteres por cada lata. Por otra parte es imperfecto. Nuestros resultados indican un 6 % de error. Esto es, en un 1.000.000 de latas (una jornada de 8 horas a 35 latas por segundo), validaría de forma errónea 60000 latas. Sin embargo, al explorar las razones para este porcentaje de error estimamos que puede ser reducido sin demasiado esfuerzo.

La importancia de la detección de lata

1. MONICOD hace un uso intensivo de las áreas de interés porque disminuyen el volumen de datos a procesar. En este trabajo se proponen tipos de área de interés elíptica (sólo-contorno e interior-de-la-elipse) adaptadas a la forma de los objetos a tratar: Superficies de lata. Sin embargo nada en MONICOD restringe el uso de otras formas que mejoren velocidad y calidad tal como se propone en la sección 10.2 . Sobre esto cabe mencionar que en el capítulo 9 se utilizan circunferencias/círculos (eje mayor igual a eje menor).

2. MONICOD NO procesa todos los *frames* del mismo modo. Todos los *frames* pasan por un procesamiento (a) que deriva en (b) para un pequeño porcentaje de los adquiridos y en (c) para todos los demás:

   a) Común: Mientras la lata transita a través del campo de visión se recolectan *frames*. Cuando la lata ha pasado se evalúa cual de los *frames* recolectados es el más adecuado para ser sometido a una validación de código. La medida para la selección NO debe invertir mas de $\frac{1}{200}$ segundos de ahí que en este momento se use un simple recuento de píxeles. Está descrito en el capítulo 7.

   b) El *frame* designado como más adecuado representará a esa lata y sobre este se aplicará una validación de código. Descrito también en el capítulo 7.

   c) El resto de los *frames* de esa lata será descartado.

3. Es ESENCIAL que la selección descrita ocurra en paralelo a la validación de código de la lata anterior. Para ello la arquitectura de la máquina donde corre MONICOD debe soportarlo. Esta condición no es demasiado restrictiva habida cuenta de que casi toda máquina actual dispone de capacidades *multithreading* con soporte para dos hilos o mas.

4. Los tramos de recolección que ocurren en los intermedios entre latas dentro de la cadena de montaje suponen auténticas ventanas que MONICOD aprovecha. Operando a 35 latas constantes tendremos una ventana de $\frac{1}{35}$ segundos en el peor escenario que debe aprovecharse para validar la lata actual de forma completa. En ese espacio de tiempo de $\frac{1}{35}$segundos , a 200 fps se recolectan 5.714 *frames*. El resultado confirma este logro.

5. Este método prescinde de una célula fotoeléctrica o cualquier otro elemento de trigger externo a la propia adquisición.

Preproceso: Realce y ecualizado

1. Los resultados en el capítulo 7 prueban que está etapa es la que consume más tiempo.

2. MONICOD opera sobre un área de interés (una sección de la superficie de la lata).

3. Se han escogido dos métodos muy sencillos para atajar los dos problemas principales de forma específica: La falta de contraste producto de un tiempo de exposición reducido por el alto *framerate* de la adquisición y la fragmentación del código consecuencia de la velocidad y tecnología de la impresión.

4. El orden, primero realce y luego ecualizado, es el que experimentalmente muestra mejores resultados.

5. El realce ataja la fragmentación de tinta engrosando código (y ruido) sin embargo tiene el efecto negativo de debilitar los espacios intercaracter e interlinea. Computa el mínimo de la ventana local de cada uno de los píxeles de la región de interés, por tanto, existe solapamiento entre las lecturas siendo esta la primer causa de su lentitud relativa. Para cada ventana local precisa una comparación igual al área de la ventana. En cada comparación podría haber o no una asignación.

6. El ecualizado se lleva a cabo en dos pasadas. La primera pasada calcula la tabla LUT. La segunda reasigna los colores según la tabla. El cálculo de la tabla LUT involucra gran número de operaciones reales pero a cambio produce un buen contraste.

7. Tanto para realce o ecualizado cualquier medida que posibilite la utilización de áreas de interés más pequeñas y más ajustadas al código de caducidad supondría beneficios directos en velocidad. Por ejemplo, el preprocesamiento de los "casquetes" de las elipses/circunferencias es prescindible porque sabemos que ahí no va a haber código. Esta técnica se emplea en la sección 9.2.

Asociador de tinta en fragmentos

1. La asociación de tinta convierte los píxeles en fragmentos de carácter (y de ruido). Se ha empleado el método de agregación de píxeles por inundación por su alta velocidad. Sólo involucra operaciones de comparación y asignación.

2. El binarizado es la primera etapa de la fase de extracción de caracteres. Trata de separar los píxeles tintados del fondo. Para mayor velocidad se hace uso de un umbral global precomputado. Sin embargo una iluminación no homogénea hace que el umbralizado no proporcione una separación óptima, y se infiltre ruido. El resultado del binarizado son las semillas de inundación.

3. Se escoge una semilla arbitraria no visitada (al principio no hay visitadas), y se explora a sus vecinos agregando a aquellos que son también semillas no visitadas. Todo píxel agregado se marca como visitado. Se repite el ciclo está vez sobre los agregados en la etapa anterior. Cuando no es posible agregar nuevos píxeles el fragmento se declara cerrado. Todas estas acciones se reducen a comparaciones, sumas y asignaciones.

4. Si un fragmento es demasiado grande se declara como ruido. Fragmentos demasiado pequeños son descartados. Si al final no hay suficientes fragmentos válidos el sistema declara el código como no valido. Con este sencillo mecanismo se detecta la causa de error más frecuente: La ausencia de código. Adicionalmente se detecta códigos mutilados (una línea no ha sido impresa, parte del código ha quedado fuera de la lata,...). Esto nos permite descartar la lata antes de intentar un procesamiento más profundo.

5. Mientras el proceso de asociación de tinta tiene lugar se obtiene fácilmente valiosa información como los límites geométricos del fragmento, cantidad de tinta de el fragmento, o la cantidad de tinta que hay en cada fila del área de interés. Esto evita nuevas pasadas sobre los fragmentos.

División por bandas

1. A partir de la información recolectada en el paso anterior (cantidad de tinta en una fila) es posible separar de forma inmediata el código en líneas de texto. Esto es denominado "búsqueda de bandas evidentes".

2. Cuando lo anterior no es posible (por ejemplo el código presenta una ligera inclinación) se propone un sistema de búsqueda de líneas de código multiagente recursivo computacionalmente económico.

3. En este sistema, y a partir de las franjas obtenidas en la "búsqueda de bandas evidentes" múltiples agentes hacen una exploración simultánea. El trazado del primer agente que lo consiga se considera como posible división para esa banda. Los agentes siguen unas reglas sencillas:

   a) Un agente evita avanzar directamente sobre tinta: Debe rodearla.

   b) Si las alternativas para el rodeo ofrecen perspectivas idénticas de éxito el agente se divide (mitosis) y se progresa por todas las alternativas.

   c) Los caminos de los agentes que avanzan con la misma dirección prioritaria no se cruzan.

   d) Tampoco se cruzan los límites dados por la búsqueda de bandas evidentes.

   e) Si un agente carece de alternativas se extingue.

4. Esta aproximación sólo involucra comparaciones, sumas y asignaciones.

5. La forma del agente empleada es un cuadrado de $3x3$. Pero es posible tratar con agentes $1x1$, $2x2$, etc. E incluso agentes no cuadrados como $3x1$. $3x3$ reduce las desviaciones de trayectoria (el agente puede avanzar parcialmente sobre tinta sin desviar su trayectoria).

6. Esta estrategia sólo es efectiva cuando el código está ligeramente rotado (menos de 30^{o}).

7. Esta estrategia con su actual configuración no es infalible: Los agentes son sensibles a las concavidades del código y al engrosamiento del código (porque reduce el espacio de interlineado).

8. Si al finalizar el proceso no hay suficientes bandas para tener código se declara lata no válida.

Agrupamiento de fragmentos

1. La estrategia propuesta de agrupamiento de fragmentos ha probado ser considerablemente efectiva.

2. La existencia de un procedimiento de agrupamiento evita que tengamos que hacer un engrosamiento demasiado intensivo.

3. Sólo se paga por lo que se agrupa. Si no hay agrupamientos el consumo de tiempo del método es muy bajo.

4. Si el código presenta buena definición, no está rotado mas de 30^{o}, está bien umbralizado, ha sido engrosado y la separación de bandas es correcta la relación fragmento-grupo será próxima a 1:1, con lo que el agrupamiento intervendrá en pocas ocasiones.

5. Las reglas de agrupamiento se caracterizan por su simplicidad:

   a) Dos fragmentos tienen que pertenecer a la misma banda para ser agrupables.

   b) Dos fragmentos que presentan solapamiento vertical son candidatos a ser agrupados. La fuerza de su candidatura al agrupamiento depende del porcentaje de solapamiento.

   c) Para que dos fragmentos sean agrupables la distancia entre sus píxeles más próximos debe ser menor que el umbral de distancia máxima.

   d) La suma de tinta de los fragmentos no debe superar cierto umbral de tinta máxima por carácter.

   e) Los límites geométricos del agrupamiento ($xmin, ymin, xmax, ymax$) no pueden sobrepasar los límites geométricos tolerados para un carácter.

6. El método de agrupamiento no declara latas no válidas. El método de división de bandas y la asociación de tinta si son capaces de hacerlo.

Validador de caracteres

1. El validador de caracteres (TM) no representa un cuello de botella en el sistema MONICOD tal como se demuestra en la sección 9.2.

2. El validador de caracteres nos permite prescindir de un extractor de características. Eso se revela como un aspecto clave en el experimento 9.7.

3. TM es muy limitado en cuanto a generalización. En particular el validador de caracteres es muy sensible a la varianza.

4. Sin embargo en un escenario industrial como este la necesidad de generalización deberia ser débil y la varianza puede ser restringida severamente:

a) No existe varianza por traslación. Los métodos previos mantienen al validador de caracteres ignorante de la posición que el carácter ocupa en la lata.

b) No existe varianza en la escala. Durante la jornada se espera que el texto impreso tenga especificaciones constantes respecto a escala.

c) La varianza respecto a la rotación es un efecto de la inclinación de código. Sin embargo este efecto puede ser atenuado con una sencilla aproximación física acercando cámara y sistema impresor. Además puede contemplarse un catálogo de rotaciones de código en el aprendizaje.

5. Es posible restringir el consumo de tiempo estableciendo un número máximo de morfologías más restrictivo. Esto queda probado en experimento 9.7. Con esto se aumenta el número de falsas validaciones negativas para algunas familias morfológicas pero otras permanecerán inalteradas.

6. Con caracteres MONICOD y en igualdad de condiciones TM (comparación de plantillas) es más rápido en clasificación que KNN y SVM. Por otro lado las redes neuronales (RBF y MLP) pueden ser igual o más rápidas que TM en clasificación pero la calidad de esta es inferior.

7. Bajo las condiciones de reconocimiento de caracteres TM es el más rápido en tiempo en entrenamiento entre los comparados. Suficientemente rápido para ofrecer la posibilidad del aprendizaje en caliente. Esto es; puede adquirirse nuevo conocimiento mientras se clasifica. Esto no parece posible con MPL, RBF y SVM; Y difícil de conseguir con KNN.

8. TM puede hacer uso del historial de clasificaciones pasadas para actualizar la BFM. Esto tiene sentido hacerlo porque las plantillas son impresas en el mismo orden en el que son presentadas al sistema y no de forma aleatoria. Es de esperar que el conocimiento reciente resulte útil para tratar efectos como la degradación progresiva del código.

9. La inclinación de código puede ser asimilada por el validador de caracteres en tiempo de clasificación.

10. Puede aplicarse estrategias/heurísticas más elaboradas en el esquema de comparación de plantillas. Por otra parte el índice de correlación -que proporciona una medida de la distancia entre dos entre dos caracteres- es flexible e intuitivo y abre un abanico de posibilidades (descrito parcialmente en el cuadro 8.1).

11. Al contrario que otros clasificadores (como las redes neuronales), este método ofrece una medida de cuanto se desvía un carácter de la norma: En un entorno industrializado las etiquetas y los caracteres deben ajustarse a un estándar por diseño.

12. Las plantillas, que son vectores de ceros y unos, pueden ser asimiladas como cadenas de bits, y así aplicar operadores optimizados para su manipulación como el XOR y el AND. Ver el apéndice C para una comparación entera (vectores de enteros) y binaria (vectores de bits).

13. El esquema puede ser internacionalizado con gran facilidad aceptando otros alfabetos o glifos.

MONICOD y la calidad de validación MONICOD es paranoico. Esto es; cuando falla es proclive a indicar validaciones negativas falsas (6 %). A cambio minimiza la posibilidad de emitir validaciones positivas falsas. Esta filosofía está en consonancia con la legislación, pero, supone un perjuicio para la empresa, que debe atender falsas alarmas o rechazar latas perfectamente válidas.

A efectos prácticos, es muy fácil circundar en gran medida el problema de validaciones negativas falsas: Se ha constatado que fallos de impresión aislados son raros. En otras palabras, cuando hay un problema con la impresión probablemente se manifieste en todas las latas a partir de la primera incidencia. Aprovechando esto puede darse la alarma sólo cuando se detecta una secuencia de validaciones negativas consecutivas (por ejemplo una secuencia de dos o tres latas), o cuando un periodo de tiempo (por ejemplo dos o tres segundos) particularmente nefasto (por ejemplo más de un 10 % de latas validadas negativamente).

MONICOD y la iluminación La superficie de lectura sobre la que opera MONICOD es hostil: Se desplaza a gran velocidad y es metálica generando brillos metálicos agresivos. La captura de alta velocidad requiere una alta frecuencia de obturación en la cámara; como consecuencia se reduce el tiempo de exposición; esto fuerza la necesidad de aplicar una iluminación intensa que excite al sensor y genere suficiente contraste; una iluminación intensa exacerba el brillo. MONICOD utiliza una iluminación especial (cuyo rendimiento puede apreciarse en la figura 4.13) que amortigua en gran medida el brillo metálico pero a costa de no producir un buen contraste.

MONICOD y los aspectos técnicos La parte lógica de MONICOD es una colección de algoritmos de computo transparente y auto-contenida. Es compatible con máquinas de propósito general. No requiere hardware de computo especializado. Cualquier máquina doméstica de perfil bajo puede ejecutar los algoritmos descritos. Algunos detalles:

1. La actual encarnación de MONICOD, empleada por ejemplo para generar los resultados presentados en este documento, no guarda dependencia de software propietario (salvo el SDK de adquisición) lo que lo convierte en un software de bajo costo. La interfaz, la generación del display, el trasiego de ficheros, etcétera es sólo dependiente de licencias liberales (siendo la más restrictiva LGPL). En la actual encarnación todo el código del que de-

pende MONICOD es abierto de manera que puede recompilarse en nuevas máquinas, aprovechándose así los avances en procesadores y compiladores.

2. En la actual encarnación de MONICOD la única dependencia existente de licencias propietarias es el software de adquisición. Dado que la parte lógica de MONICOD se abstrae de la tecnología con la que los *frames* son adquiridos, la dependencia no es tal.

   a) No dependencia del sistema operativo. La actual encarnación de MONICOD, que ya se ejecuta en entorno Microsoft Windows, puede ser ejecutada en un entorno Linux/Unix, y previsiblemente, en un entorno Apple MacOSX, dado que todas las dependencias presentes son multiplataforma. De esta versatilidad se ha beneficiado la actual encarnación MONICOD. Sin embargo el software de adquisición sólo opera en plataforma Microsoft Windows.

   b) Los elementos físicos de MONICOD son de bajo costo (relativamente) y fácilmente reemplazables.

10.2. Extensiones y trabajo futuro

MONICOD tiene MUCHO espacio para la extensión. Es necesario sacar partido al excedente de tiempo conseguido y profundizar en la mejora de la tasa de acierto frente a falsas validaciones positivas. En esta sección separamos refinamientos y extensiones respecto a su propósito: Relativa a la calidad y relativa a la velocidad.

10.2.1. Calidad

Dado que disponemos de margen de tiempo considerable es posible introducir algoritmos más sofisticados sin cambiar la arquitectura MONICOD. Por otro lado existen partes en MONICOD con margen de mejora.

10.2.1.1. Algunas mejoras a corto plazo

Detección de lata

1. ¿Es posible un criterio de activación alternativo que no malogre la adquisición?

2. Deben buscarse otros patrones de búsqueda y detección alternativos al contorno de la elipse/círculo. Por ejemplo patrones rectangulares[1], los casquetes de la elipse/círculo, o el centro de la elipse/círculo. El patrón debe ser mínimo y certero.

Asociador de tinta en fragmentos Aplicar algoritmos de suavizado binario a los fragmentos asociados puede resultar particularmente beneficioso para las etapas posteriores: Recuperación de espacio de interlineado, resolver las dificultades de fragmentos de distintos caracteres unidos o demasiado cercanos, suavizado de las morfologías para la clasificación las ajustarían más a un prototipo, etcétera... El problema es el costo asociado.

Validación de caracteres Parece muy conveniente aplicar sobre las morfologías filtros de suavizado como los descritos en la subsección 2.4.1 antes de su inclusión en la BFM. Con ello se mejoraría la correlación dentro de una familia morfológica y el costo en cómputo quedaría acotado al aplicarse el suavizado sólo sobre las morfologías.

Por otra parte debe concluirse el desarrollo de un sistema de aprendizaje en tiempo validación esbozado en la sección 8.2.2 del capítulo8.

10.2.1.2. Tres líneas de desarrollo futuro a medio/largo plazo.

Umbralizado por zonas Es IMPERATIVO pasar a alguna clase de modelo automático que neutralice en la medida de lo posible el problema de la iluminación no homogénea en la superficie de la lata. Un éxito aquí, aunque fuese parcial, impactaría positivamente a la totalidad del sistema de validación.

Reconsiderar la detección de espacio interlineal

- Un agrupamiento que prescindiera del etiquetado de pertenencia de fragmento a banda haría prescindible en gran medida la necesidad de una detección de bandas.
- Una mejor preservación de los espacios interlineales haría posible lo primero.
- A partir de lo anterior las bandas podrían obtenerse por un procedimiento de clustering/regresión lineal de los centroides de los grupos si la operación se revela lo suficientemente eficiente.
- Por otra parte ¿sería concebible prescindir completamente de la separación de bandas y simplemente etiquetar/ordenar los grupos de arriba a abajo, de izquierda a derecha?. Para esto seguiría siendo necesario restringir la rotación de las latas.

Aplicación de otras tecnologías para validación de caracteres Hasta ahora se había descartado la introducción de mecanismos más sofisticados en validación para garantizar la máxima celeridad (ignorando tarea de extracción de características) y evitar un exceso de generalización que podría resultar contraindicado en una tarea de validación (que no es reconocimiento). Sin embargo el sistema de comparación de plantillas es, como cabría esperar demasiado rígido: Cualquier desviación es duramente castigada.

Por otra parte las máquinas de vectores soporte ofrecen un buen rendimiento tanto en clasificación como en aprendizaje con una capacidad superior en identificación. ¿Sería posible implantar ambas virtudes en MONICOD?

10.2.2. Velocidad

El desarrollo de los procesadores multinucleo y el computo derivado a una o varias GPU (Unidad de proceso de tarjeta gráfica) impulsan la paralelización. Estos avances son directamente aprovechables porque partes de MONICOD, que ahora se ejecutan secuencialmente, pueden afrontarse en paralelo. Además es de esperar que MONICOD sea tan rápido como el hardware disponible se lo permita. Previsiblemente el progreso tecnológico corre a favor de MONICOD. Estimamos que, de ser necesario, hay un margen enorme para la mejora en tiempo sólo ateniéndose a consideraciones técnicas y sin modificaciones relevantes a los algoritmos descritos. Esto es importante porque las extensiones relativas a la calidad descritas en el epígrafe anterior pueden exigirnos aprovechar dicho margen. SIN EMBARGO, a la vista de los resultados, aumentar la velocidad tiene escaso sentido porque la ingeniería industrial no ofrece una cadencia de latas que pueda aprovechar esa ventaja.

En cualquier caso, el primer paso para una optimización es establecer un orden de prioridades. Y la regla es: Las tareas que consumen más tiempo deben ser las primeras a considerar. Por consistencia, listaremos las posibles optimizaciones por fases y no por prioridades.

Detección de lata

1. Pueden aprovecharse las capacidades de las tarjetas de adquisición para seleccionar áreas de interés a costa de establecer una dependencia con el hardware.

Preproceso: Realce y ecualizado

1. Es factible realzar en paralelo cada parte de una partición de la imagen de entrada, depositando el resultado en una imagen de salida. No existen interferencia entre hilos porque la partición garantiza que los conjuntos de lectura y escritura de los preprocesos son disjuntos. Creemos que la mejora sería significativa: El número de particiones fraccionaría más o menos el tiempo invertido en el realce. Por ejemplo: una partición de dos dividiría el consumo actual del realce a un poco mas de la mitad. El realce es la tarea más costosa en MONICOD.

2. Tecnologías como CUDA facilitan el cálculo matricial a través de la GPU. ¿Sería posible aprovecharlas abriendo la puerta a preprocesos más sofisticados sin pérdida de velocidad?

Asociador de tinta en fragmentos

1. Es concebible crear los fragmentos en paralelo. El único conflicto a resolver es el intento de agregar el mismo píxel a dos fragmentos distintos por parte de dos flujos de ejecución distintos. Este conflicto sólo se produce, cuando los dos fragmentos en construcción son en realidad uno. La resolución podría consistir en una transferencia del trabajo de un fluyo a otro y liberarlo.

División por bandas

1. Los *divisores* de banda operan como agentes independientes de facto. Sólo habría que garantizar la atomicidad en el acceso de escritura al mapa de consultas compartido para tener una exploración segura multiagente que ocurra en paralelo. Esto debería acelerar la división de bandas y abriría el camino a búsquedas más sofisticadas.

Validador de caracteres

1. Las líneas o bandas de código pueden ser validadas por separado, ya que en la práctica son plenamente independientes. Incluso es posible enfrentar en paralelo a cada agrupamiento/carácter con la familia morfológica esperada para esa posición. Es decir; se pueden explorar las familias morfológicas simultáneamente. Por último también debería ser posible comparar varias plantillas simultáneamente.

2. Al contrario que los otros clasificadores, TM favorece de forma intuitiva la divisibilidad del computo, dado que se pueden explorar las familias morfológicas simultáneamente.

3. Ahora bien, como prueban los resultados, en MONICOD la validación completa supone un 13 % del tiempo total. Mejorar este área no debería ser prioritario.

.

Bibliografía

[1] D. G. González, "Sistema de visión artificial para control de calidad industrial en una planta de envasado." ULPGC, 2006.

[2] *Guinness World Records (The Guinness Book of Records)*, Jim Pattison Group, 1955 at present.

[3] The LyX Team, "LyX 2.0.1 - The Document Processor [Computer software and manual]." Internet: http://www.lyx.org, 2011. Retrieved November 03, 2011, from http://www.lyx.org.

[4] JabRef Development Team, *JabRef*. JabRef Development Team, 2010.

[5] Dimitri van Heesch, "Doxygen 1.7.5 - Generate documentation from source code [Computer software and manual]." Internet: http://www.stack.nl/ dimitri/doxygen/, 2011. Retrieved November 03, 2011, from http://www.stack.nl/ dimitri/doxygen/.

[6] Smartdraw developers, "Smartdraw - Communicate visually]." Internet: http://www.smartdraw.com/. Retrieved June 19, 2012, from http://www.smartdraw.com.

[7] I. The MathWorks, "Matlab r2012a," 2012.

[8] B. E. M. Stanley B. Lippman, Josée LaJoie, *C++ Primer*, Addison-Wesley Professional, 5º ed., 2012.

[9] B. Stroustrup, *The C++ Programming Language: Special Edition*, 2000.

[10] F. Thompson, "Fordism, post-fordism and the flexible system of production." www.willamette.edu, December 2008.

[11] S. Morison, *First Principles of Typography*, Macmillan, 1936.

[12] P. W. Handel, "Statistical machine," June 1933.

[13] F. L. H. E. Djamel Gaceb, Véronique Eglin, "Improvement of postal mail sorting system," p. 14, 2008.

[14] H. Fujisawa, "Forty years of research in character and document recognition - an industrial perspective," *Pattern Recognition* **41**, pp. 2435–2446, 2008.

[15] INC, "Instituto nacional de consumo (inc)."

[16] P. P. M. K. S. H. A. M. K. R. Nagarajan, Sazali Yaacob, "A real time marking inspection scheme for semiconductor industries," *Int j Adv Manuf Technol (2007)* , pp. 926–932, 2006.

[17] D. G. S. Richard O. Duda, Peter E. Hart, *Pattern Classification*, A Wiley-Interscience Publication, second edition ed., 2001.

[18] colaboradores de Wikipedia, "Wikipedia," 2012.

[19] C.-l. L. C. Y. S. Mohamed Cheriet, Nawwaf Kharma, *Character Recognition Systems: A Guide for Students and Practicioners*, John Wiley & Sons, Inc., 2007. ISBN: 978-0-471-41750-1.

[20] D. V. V. Saradhadevi, "A survey on digital image enhancement techniques," *International Journal of Computer Science and Information Security* **8(8)**, pp. 173–178, 2010.

[21] C. F. Zhu H. and Z. Y., "Image contrast enhancement by constrained local histogram equalization," *Computer Vision Image Understanding* **73(2)**, pp. 281–290, 1999.

[22] R. Maini and H. Aggarwal, "A comprehensive review of image enhancement techniques," *JOURNAL OF COMPUTING* **2(3)**, pp. 8–12, 2010.

[23] J. Russ, *The Image Processing Handbook, 6nd ed.*, CRC Press., 6 ed., 2011.

[24] J. González Jiménez, *Visión por computador*, Editorial Paraninfo, 2000. ISBN: 84-283-2630-4.

[25] S. C. Galatsanos, N.P. and Katsaggelos, *Digital Image Enhancement*, pp. 388–402. 2003.

[26] J. M. d. l. C. Gonzalo Pajares, *Visión por computador:Imágenes digitales y aplicaciones*, 2001. ISBN: 84-7897-472-5.

[27] J. K. Manpreet Kaur, Jasdeep Kaur, "Survey of contrast enhancement techniques based on histogram equalization," *International Journal of Advanced Computer Science and Applications* **2(7)**, pp. 137–141, 2011.

[28] D. S. Changming Sun, "Skew and slant correction for document images using gradient direction," *4th International Conf. on Document Analysis and Recognition,* , pp. 142–146, 1997.

[29] A. K. Jian-xiong Dong, Dominique Ponson and C. Y. Suen, "Cursive word skew/slant corrections based on radon transform," *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)* , pp. 478–483, 2005.

[30] L. G. S. Robert M. Haralick, "Image segmentation techniques," *Computer Vision, Graphics, and Image Processing* **29**, pp. 100–132, 1985.

[31] T. Pavlidis, *Structural pattern recognition*, 1977.

[32] L. Spirkovska, "A summary of image segmentation techniques," *Nasa Technical Memorandum* **June 1993**, 1993.

[33] Y.-J. Zhang, *.^An Overview of Image and Video Segmentation in the Last 40 Years". Advances in Image and Video Segmentation*, IGI Global, 2006.

[34] J. M. K.S. Fu, "A survey on image segmentation," *Patter* **13**, pp. 3–16, 1981.

[35] P. S. K. Pal, N. R., "A review on image segmentation techniques," *Pattern Recognition* **26**, pp. 1277–1294, 1993.

[36] M. F. E. F. Marcello, J., "Evaluation of thresholding techniques applied to oceanographic remote sensing imagery," *SPIE, 5573* , pp. 96–103, 2004.

[37] Y. J. Zhang, "Evaluation and comparison of different segmentation algorithms.," *Pattern Recognition Letters* **18(10)**, pp. 963–974, 1997.

[38] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Electronic Imaging* , pp. 146–165, 2004.

[39] A. Rosenfeld and P. D. la Torre, "Histogram concavity analysis as an aid in threshold selection," *IEEE Trans. Syst. Man Cybern.* **13**, pp. 231–235, 1983.

[40] M. I. Sezan, "A peak detection algorithm and its application to histogram-based image data reduction," *Computer Vision, Graphics, and Image Processing* **49(1)**, pp. 36–51, 1990.

[41] Y. J.-H. S. I. Ramesh, N., "Thresholding based on histogram approximation," *Vision, Image and Signal Processing, IEE Proceedings* - **142(5)**, pp. 271–279, 1995.

[42] W. Tsai, "Moment-preserving thresholding: a new approach," *Computer Vision, Graphics, and Image Processing* **19**, pp. 377–393, 1985.

[43] L. Hertz and R. W. Schafer, "Multilevel thresholding using edge matching," *Comput. Vis. Graph. Image Process.* **44**, pp. 279–295, 1988.

[44] C. A. Murthy and S. K. Pal, "Fuzzy thresholding: A mathematical framework, bound functions and weighted moving average technique," *Pattern Recogn. Lett.* **11**, pp. 197–206, 1990.

[45] L. K. Huang and M. J. J. Wang, "Image thresholding by minimizing the measures of fuzziness," *Pattern Recogn* **28**, pp. 41–51, 1995.

[46] S. N. S. L. G. K. Ramar, S. Arunigam and D. Manimegalai, "Quantitative fuzzy measures for threshold selection," *Pattern Recogn. Lett.* **21**, pp. 1–7, 2000.

[47] C. S. Ridler TW, "Picture thresholding using an iterative selection method," *IEEE Trans. System, Man and Cybernetics* **SMC-8**, pp. 630–632, 1978.

[48] W. M.-J. J. Huang L-K, "Image thresholding by minimizing the measures of fuzziness," *Pattern Recognition* **28(1)**, pp. 41–51, 1995.

[49] M. M. Prewitt JMS, "The analysis of cell images," *Annals of the New York Academy of Sciences* **128**, pp. 1035–1053, 1966.

[50] L. C. Li CH., "Minimum cross entropy thresholding," *Pattern Recognition* **26(4)**, pp. 617–625, 1993.

[51] T. P. Li CH., "An iterative algorithm for minimum cross entropy thresholding," *Pattern Recognition Letters* **18(8)**, pp. 771–776, 1998.

[52] W. A. Kapur JN, Sahoo PK, "A new method for gray-level picture thresholding using the entropy of the histogram," *Graphical Models and Image Processing* **29(3)**, pp. 273–285, 1985.

[53] C. Glasbey, "An analysis of histogram-based thresholding algorithms," *CVGIP: Graphical Models and Image Processing* **55**, pp. 532–537, 1993.

[54] J. Kittler and J. Illingworth, "Minimum error thresholding," *Pattern Recognition* **19**, pp. 41–47, 1986.

[55] N. Otsu, "A threshold selection method from gray-scale histogram.," *IEEE Transactions on System, Man, and Cybernetics* **9**, pp. 62–66, 1979.

[56] W. Doyle, "Operation useful for similarity-invariant pattern recognition," *Journal of the Association for Computing Machinery* **9**, pp. 259–267, 1962.

[57] A. Shanhbag, "Utilization of information measure as a means of image thresholding," *Graphical Models and Image Processing* **56(5)**, pp. 414–419, 1994.

[58] R. W. E. Zack, G. W. and S. A. Latt, "Automatic measurement of sister chromatid exchange frequency," *Journal of Histochemistry and Cytochemistry* **25(7)**, pp. 741–753, 1977.

[59] C. S. Yen JC, Chang FJ, "A new criterion for automatic multilevel thresholding," *IEEE Trans. on Image Processing* **4(3)**, pp. 370–378, 1995.

[60] M. P. R. S. Abramoff, M.D., "Image processing with imagej," *Biophotonics International* **11(7)**, pp. 36–42, 2004.

[61] J. Bernsen, "Dynamic thresholding of grey-level images," *Proc. of the 8th Int. Conf. on Pattern Recognition* , 1986.

[62] W. Niblack, *An introduction to Digital Image Processing*, Prentice-Hall, 1986.

[63] J. Sauvola and M. Pietaksinen, "Adaptive document image binarization," *Pattern Recognition* **33(2)**, pp. 225–236, 2000.

[64] O. Trier and A.K.Jain, "Goal-directed evaluation of binarization methods," *IEEE Transactions on System, Man, and Cybernetics* **25(5)**, pp. 738–754, 1995.

[65] O. Trier and A.K.Jain, "Evaluation of binarization methods for document images," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**, pp. 312–314, 1995.

[66] S. S. P. K. Sahoo and A. K. C. Wong, "A survey of thresholding techniques," *Computer Vision, Graphics, and Image Processing* **41**, pp. 233–260, 1988.

[67] R. E. W. Rafael C. González, *Tratamiento Digital de Imágenes*, Addison Wesley/Diaz de Santos, 1996.

[68] L. Roberts, "Machine perception of three-dimensional solids," in *Optical and Electro-Optical Information Processing*, J. T. et al(eds.), ed., pp. 159–197, The MIT Press, Cambridge,MA, 1965.

[69] R. Kirsch, "Computer determination of the constituent structure of biomedical images," *Computer and Biomedical Research* **4(3)**, pp. 315–328, 1971.

[70] L. C. Beucher S., "Use of watersheds in contour detection," *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation,* , 1979.

[71] E. Iturrate, "De píxeles a objetos:hacia un procedimiento de clasificación más inteligente.," *Geomática 2007* , 2007.

[72] A. M. Jos B. T. M. Roerdink, "The watershed transform: Definitions, algorithms and parallelization strategies," *Fundamenta Informaticae* **41**, pp. 187–228, 2001.

[73] H. Niemann, *Pattern analysis and understanding*, second edition ed., 1990.

[74] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Trans. Information Theory* **IT-8**, pp. 179–187, 1962.

[75] K.-M. L. Wai-Hong Wong, Wan-Chi Siu, "Generation of moment invariants and their uses for character recognition," *Pattern Recognition Letters* **16(2)**, pp. 115–123, 1995.

[76] K. Singh, "A comparison of 2d moment based description techniques for classification of bamboo plant," in *International Conference on Computational Intelligence and Communication Networks (CICN)*, *International Conference on Computational Intelligence and Communication Networks (CICN)*, , pp. 15–20, 2011.

[77] J. Flusser, "Moment invariants in image analysis," *World Academy of Science, Engineering and Technology* **11**, pp. 1307–6884, 2006.

[78] J. Flusser, "On the independence of rotation moment invariants," *Pattern Recognition* **33**, pp. 1405–1410, 2000.

[79] Y. S. Abu-Mostafa, "Recognitive aspects of moment invariants," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6(6)**, pp. 698–706, 1984.

[80] Y. S. Abu-Mostafa, "Image normalization by complex moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-7(1)**, pp. 46–55, 1985.

[81] M. R. Teague, "Image analysis via the general theory of moments," *Optical Society of America, Journal* **70**, pp. 920–930, 1980.

[82] R. T. C. Cho-Huak, Teh, "On image analysis by the methods of moments," *IEEE Trasactions On Pattern Analysis and Machine Intelligence* **10(4)**, pp. 496–512, 1988.

[83] C. T. G. H. G.Y.Yang, H.Z. Shu and L. Luo, "Efficient legendre moment computation for grey level images," *Pattern Recognition* **39**, pp. 74–80, 2006.

[84] H. Y. Khotanzad, A., "Invariant image recognition by zernike moments," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **12(5)**, pp. 489 – 497, 1990.

[85] C.-W. Chong, "A comparative analysis of algorithms for fast computation of zernike moments," *Pattern Recognition* **36(3)**, pp. 731–742, 2003.

[86] R. Mukundan, "Image analysis by tchebichef moments," *IEEE Trans. on Image Proc* **10**, pp. 1357–1364, 2001.

[87] C. B. J.-H. P. L. L. Shu H.Z., Zhang H., "Fast computation of tchebichef moments for binary and gray-scale images," *IEEE Transactions on Image Processing* **19(12)**, pp. 3171–3180, 2010.

[88] M. B. Spiliotis IM, "Real-time computation of two-dimensional moments on binary images using image block representation," *IEEE Trans Image Process* **7(11)**, pp. 1609–1615, 1998.

[89] D. K. G.A. Papakostas, E.G. Karakasis, "Efficient and accurate computation of geometric moments on gray-scale images," *Pattern Recognition* **41(6)**, pp. 1895–1904, 2008.

[90] I. A. L. Kotoulas, "Image moments based on hahn polynomials," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29**, pp. 2057–2062, 2007.

[91] R. . S.-H. O. Pew-Thian Yap, Paramesran, "Image analysis using hahn moments," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **29**, pp. 2057 – 2062, 2007.

[92] L. Kotoulas and I. Andreadis, "Image analysis using moments," *5th Int. Conf. on Technology and Automation* , pp. 360–364, 2005.

[93] M. Abdul-Hameed, "High order multi-dimensional moment generating algorithm and the efficient computation of zernike moments," *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1997. ICASSP-97.* **4**, pp. 3061 – 3064, 1997.

[94] M. Hatamian, "A real-time two-dimensional moment generating algorithm and its single chip implementation," *IEEE Transactions on Acoustics, Speech and Signal Processing* **34(3)**, pp. 546–553, 1986.

[95] A. L. I. A. Dalhoum, "A comparative survey on the fast computation of geometric moments," *European Journal of Scientific Research* **24(1)**, pp. 104–111, 2008.

[96] A. M. C. Di Ruberto, "A comparison of 2-d moment-based description techniques," in *Proceedings of the 13th international conference on Image Analysis and Processing, ICIAP'05 Proceedings of the 13th international conference on Image Analysis and Processing* (8), pp. 212–219, Springer-Verlag, (Berlin, Heidelberg), 2005.

[97] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Comm. ACM* **15**, pp. 11–15, 1972.

[98] J. A. Storer and M. Cohn, eds., *Algorithms for fast vector quantization*, IEEE Press, 1993.

[99] N. S. N. R. S. A. Y. W. Sunil Arya, David M. Mount, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)* **45(6)**, p. 891(923), 1998.

[100] ANN developer, "ANN: A Library for Approximate Nearest Neighbor Searching." http://www.cs.umd.edu/ mount/ANN/, 2010. Retrieved May 05, 2012, from http://www.cs.umd.edu/ mount/ANN/.

[101] S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, Inc., 2009. ISBN: 0-13-129376-1 978-0-13-129376-2.

[102] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics* **22(3)**, pp. 400–407, 1951.

[103] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery* **2**, pp. 121–167, 1998.

[104] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, 1º ed., 2000.

[105] C. M. Institute, "History of the can." http://www.cancentral.com/, 2009.

[106] R. Barrow, "Canned beers notch up first half century." OFF LICENCE NEWS Magazine, December 1985.

[107] B. P. E. GmbH, "Cans for fans." Official Website, 2009.

[108] F. S. F. A. Q. A. José Tomás Palma Méndez, Mª del Carmén Garrido Carrera, *Programación Concurrente*, Paraninfo, S.A, 2003.

[109] wxWidgets developers, "wxWidgets - Cross-Platform GUI Library [Computer software and manual]." Internet: http://www.wxwidgets.org/, 2011. Retrieved November 05, 2011, from http://www.wxwidgets.org/.

[110] ImageMagick developers, "ImageMagick - Convert, Edit And Compose Images[Computer software and manual]." Internet: http://www.imagemagick.org, 2011. Retrieved November 05, 2011, from http://www.imagemagick.org/.

[111] Boost developers, "boost - C++ libraries [Computer software and manual]." Internet: http://www.boost.org/, 2011. Retrieved November 05, 2011, from http://www.lyx.org.

[112] SDL developers, "SDL - Simple Directmedia Layer [Computer software and manual]." Internet: http://www.libsdl.org/, 2011. Retrieved November 05, 2011, from http://www.libsdl.org/.

[113] Codeblocks developers, "Codeblocks - The open source, cross platform, free C++ IDE [Computer software and manual]." Internet: http://www.codeblocks.org/, 2011. Retrieved November 15, 2011, from http://www.codeblocks.org/.

[114] Microsoft Corporation, "Visual Studio 2010 Products [Computer software and manual]." Internet: http://www.microsoft.com/visualstudio/en-us/products/2010-editions, 2011. Retrieved November 15, 2011, from http://www.microsoft.com/visualstudio/en-us/products/2010-editions.

[115] E. P. Krotkov, "Focusing," *International Journal of Computer Vision* , pp. 223–237, 1987.

[116] T. H. Cormen, *Introduction to Algorithms*, McGraw-Hill, 3rd edition ed., 2009.

[117] J. S. Thornton, "Precision timing and the principe of least action." http://www.codeproject.com/Articles/12177/Precision-timing-and-the-Principle-of-Least-Action, Nov 2005.

[118] OpenCV developers, "OpenCV - Open Source Computer Vision [Computer software and manual]." Internet: http://opencv.willowgarage.com/wiki/, 2011. Retrieved November 15, 2011, from http://opencv.willowgarage.com/wiki/.

[119] Swagat Kumar, "NNLIB - Neural Network class library [Computer software and manual]." Internet: http://sourceforge.net/projects/nnlib/, 2011. Retrieved November 15, 2011, from http://sourceforge.net/projects/nnlib/.

[120] Chih-Chung Chang and Chih-Jen Lin, "LIBSVM - A Library for Support Vector Machines [Computer software and manual]." Internet: http://www.csie.ntu.edu.tw/ cjlin/libsvm/, 2011. Retrieved November 15, 2011, from http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

Apéndice A

Definiciones

A.1. Color y espacio de color

El fenómeno del color Para la escritura basada en pigmentación la materia prima del campo de entrada es el "color". En términos humanos el color es la etiqueta que nuestro cerebro pone a las diferentes longitudes de onda de la luz que llega a nuestro ojo.

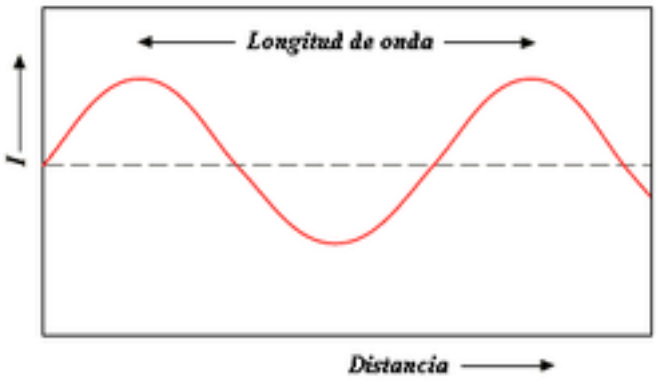


Figura A.1: La luz tiene características de fenómeno ondulatorio.

Dada la dualidad onda-corpúsculo de la luz, esta tiene una longitud de onda. Ver figuraA.1. En la evolución del sistema visual humano[1] este aspecto ha pesado lo suficiente como para dotarse de mecanismos específicos de detección de color (los conos). Y con ellos el procesamiento del color. Ver A.2

[1] Así como también para nuestros primos primates y para otras muchas especies animales

Color	Longitud de onda
violeta	~ 380-450 nm
azul	~ 450-495 nm
verde	~ 495-570 nm
amarillo	~ 570–590 nm
naranja	~ 590–620 nm
rojo	~ 620–750 nm

Figura A.2: La longitud de onda es "caracterizada" con un color en el cerebro.

Color y escritura El lenguaje escrito común utiliza el color de una forma muy limitada (si bien decisiva) en la representación gráfica de caracteres. A saber; Carácter y fondo son los dos objetos que componen la representación gráfica de la escritura. Y la primera y principal diferencia es que utilizan colores distintos. Existen sólo dos colores relevantes. El color de los caracteres y el color del fondo.

Respecto a los dos colores en particular no hay reglas. Lo óptimo es que la combinación facilite su lectura para el lector.

El color no proporciona el mensaje. Lo hace la forma de los caracteres y la disposición que tengan entre sí. El uso de un efectista abanico de color suele perseguir captar la atención del lector indicando alguna información suplementaria sobre el mensaje(por ejemplo su nivel de importancia).

Dicho esto, es comprensible que los Reconocedores Ópticos de Caracteres no diriman con demasiadas consideraciones respecto al color, salvo, el modo de separar y delimitar claramente los dos existentes. Esta separación no es inmediata o trivial. Ni siquiera en una superficie de papel perfectamente blanca escrita con tinta de un negro genuino existen únicamente dos colores para un sistema de adquisición mínimamente competente. Es necesario agrupar los colores captados en dos conjuntos que etiquetaremos como color de fondo y color de carácter. A este procedimiento se le denomina binarizado y su importancia es fundamental. Fracasar en el binarizado lleva a un fracaso casi seguro en el reconocimiento porque, en definitiva, no podremos "ver" los caracteres confundidos con el fondo.

Desafortunadamente, para el trabajo presentado, y como veremos bajo el epígrafe 4.5, el binarizado es particularmente complicado.

Espacio de Color El espacio de color es una forma de representar el color. Algunos espacios de color son más adecuados que otros al afrontar cierto tipo de problemas. MONICOD no trabaja con color sino con intensidades de luz

reflejada. El color negro puro absorbe toda la luz incidente, y el blanco puro la refleja en su totalidad.

Transformar una imagen del espacio de color RGB(tres valores numéricos de intensidad de rojo, verde y azul) a un espacio de intensidades es posible usando una aproximación tal como A.1. Se trata de una aproximación donde los pesos de las componentes RGB dependen de aspectos fisiológicos que pueden variar entre individuos, o a lo largo de la vida, tales como las proporciones de conos y bastones. Hay pérdida de información por lo que no es posible la operación inversa.

$$(i,j) = 0{,}3 \cdot f_r(i,j) + 0{,}59 \cdot f_g(i,j) + 0{,}11 \cdot f_b(i,j) \tag{A.1}$$

La plataforma de MONICOD sólo adquiere en el espacio de grises por lo que no se tratará la problemática del color. Los métodos comentados en este capítulo, así como en el resto del documento, son descritos en términos de imágenes en grises.

A.2. Vídeo: Imagen y *frame*

Para nuestros propósitos una imagen es una función bidimensional de la intensidad del nivel de gris, y contiene N píxeles cuyos niveles de gris se encuentran 1 y L. Si denotamos f_i como el número de ocurrencias del nivel de gris i en una imagen dada, tendremos que la probabilidad de ocurrencia del nivel gris i es:

$$p_i = \frac{f_i}{N} \tag{A.2}$$

La imagen estática es la entrada por excelencia del reconocedor óptico de caracteres.

Por esta razón la teoría descrita en este capítulo trata sobre imágenes, no vídeos. Ahora bien, el trabajo que se presenta en este documento tiene como campo natural de entrada al vídeo. De ahí que presentemos este tópico.

- Vídeo es una sucesión de imágenes (ver A.3.1) cuyo orden obedece a ubicaciones (o momentos) en el tiempo. Las imágenes que constituyen vídeos se las denomina *frames* o fotogramas. Estos *frames* o fotogramas son imágenes aptas (con algunas reservas que se tratan en 4.1) para el tratamiento por reconocimiento de caracteres. Podemos entender el vídeo como un contenedor de imágenes. Ver A.3.

- Un *frame* particular puede ser localizado dentro de un vídeo a partir de una única coordenada. Dicha coordenada NO puede ser compartida por otro *frame* dentro del mismo vídeo. Dicha coordenada tienen un significado temporal. Un *frame* (asociado a una coordenada temporal) es anterior o posterior en el tiempo respecto a los otros (que tienen necesariamente coordenadas temporales diferentes).

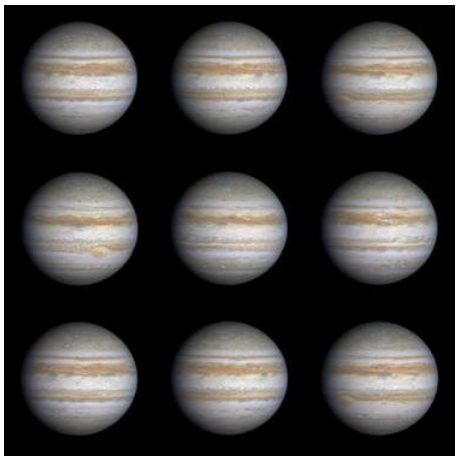


Figura A.3: Nueve *frames* extraídos de un vídeo que exhibe la rotación sobre su eje de Júpiter. Puede observarse en el *frame* más a la izquierda de la hilera central la característica Gran Mancha Roja del hemisferio sur.

Matemáticamente podemos expresarlo como una función de imágenes unidimensional $V(t)$. Véase el apartado A.3.1 donde queda explicado la representación de imágenes.

A.3. Captura y reproducción de vídeo[2]

Registrar un vídeo es la acción de ejecutar capturas sucesivas de imágenes que constituirán un vídeo y almacenarlas. Como se comentó (ver A.2) el orden de la captura es crítico y queda consignado en la etiqueta que se le otorga a la captura. El tiempo entre capturas determina la velocidad de captura de nuestro vídeo. Para todos nuestros efectos la velocidad de adquisición o captura de un vídeo es constante en su registro.

Reproducir un vídeo es la acción de mostrar los *frames* o fotogramas (imágenes) del vídeo según el orden establecido durante la captura. La velocidad de reproducción establece cuanto tiempo debe mostrarse cada *frame* antes de pasar al siguiente. Para nuestros efectos la velocidad de reproducción de un vídeo es constante. Es perfectamente plausible que dicha velocidad sea diferente a la de la velocidad de captura. Por tanto podemos hablar de dos velocidades.

1. Velocidad de captura (o adquisición)
2. Velocidad de reproducción

[2]Presumiblemente todas estas nociones no deben resultar desconocidas al lector que tenga alguna experiencia en la reproducción de vídeo.

Cuando la velocidad de reproducción es igual a la velocidad de captura hablamos de reproducción en tiempo real.

> En MONICOD se llevará a cabo un procedimiento de adquisición de imágenes pero NO se creará un vídeo (que implicaría almacenamiento en disco para posteriores reproducciones). Las razones para eso serán discutidas en la sección 4.8.

framerate Anglicismo que usaremos a lo largo de este documento con cierta frecuencia. Es el ratio entre número de *frames* (adquiridos, reproducidos, ...) y unidades de tiempo. Habitualmente se expone como número de *frames* por unidad de tiempo. La unidad de tiempo común es el segundo. El *framerate* es una medida de velocidad para captura o reproducción. En este documento usaremos el término referido a la adquisición, salvo que se indique otra cosa.

A.3.1. Campo de entrada e imagen/fotograma/*frame*

La escena capturada por la cámara es el campo de entrada. Su amplitud depende de la óptica empleada. El campo de entrada se ve proyectado sobre un plano de dos dimensiones en una representación visual/óptica discreta consistente en información de color. Dicha proyección es la imagen(tal como la entenderemos en MONICOD).

Representar una imagen implica representar dos cosas bien diferenciadas pero igual de relevantes: Color y posición del color.

La cantidad de colores posibles es infinita porque hay infinitas longitudes de una onda entre dos cualesquiera longitudes de onda. La longitud de onda es una magnitud continua. Sin embargo, por computabilidad, el espacio de posibles longitudes de onda es discretizado. Idéntico proceso sufre la posición del color. Nos serviremos de magnitudes discretas para posicionar aproximadamente.

Color y posición pueden ser representadas numéricamente mediante una dupla a la que, en este documento, denominaremos píxel.

$$(coordenada, color)$$

Aunque un número puede ser suficiente para representar la posición dentro de un plano, representarlo con dos números - uno para cada dimensión- resulta muy conveniente, simplificando las relaciones de localidad (localización) entre píxeles que trataremos más adelante. Así que:

$$(coordenadax, coordenaday, color)$$

A su vez, la representación numérica (e interpretación) del "color" es diversa y dependerá de la elección del espacio de color que hagamos.

A partir de esto resulta natural ver porque la representación habitual para imagen es la de una función bidimensional $I(x, y)$

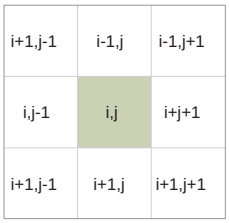


Figura A.4: La localización de un píxel en la imagen.

En MONICOD la cantidad de colores es 256. 256 niveles de grises expresados con un número en el rango [0..,256] . La cantidad de posiciones con las que se opera es 640×480.

A.3.2. Píxeles y matrices

Para nuestros efectos, píxel es la unidad mínima de información de la imagen. El píxel tiene la propiedad de tener un color asignado y le corresponde una posición.

$$(coordenadax, coordenaday, nivelgris)$$

Se disponen en una matriz. Esto es, equidistantes entre sí cubriendo el área de un rectángulo. La función bidimensional $I(x, y)$ comentada en A.3.1 se almacena comúnmente de forma explícita. Es decir, todos los valores que tiene la función están almacenados.

A.3.2.1. Localización y vecindad

La localización de un píxel en la imagen, es decir, su posición respecto al resto , es caracterizada por la dupla (i, j).Dos píxeles no pueden compartir la misma posición.Ver figura A.4

Dos píxeles son vecinos si son adyacentes. La vecindad de un píxel es determinada por la colindancia entre dos localizaciones de píxeles. La colindancia no es suficiente porque depende de la definición particular de adyacencia que se emplee. Ver figura A.5. Para una vecindad $N_4(p)$ los píxeles en sus diagonales no se consideran vecinos. Más allá de los límites de la imagen no hay píxeles así que píxeles adyacentes a los límites de esta tienen vecindades más pequeñas.

A.3.2.2. Distancia

Decimos que D es una función distancia entre píxeles de una imagen(conjunto de elementos que pertenecen a X) si cumple que $\forall p_1, p_2, p_3 \in X$ siendo p_1, p_2 y p_3 píxeles con coordenadas (x_1, y_1), (x_2, y_2) y (x_3, y_3):

1. No negatividad: $D(p_1, p_2) \geq 0$

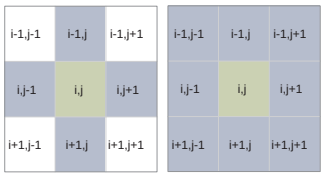


Figura A.5: Vecindad $N_4(p)$ y Vecindad $N_8(p)$

2. $D(p_1, p_1) = 0$
3. $D(p_1, p_2) = 0$ si $p_1 = p_2$
4. Simetría: $D(p_1, p_2) = D(p_2, p_1)$
5. $D(p_1, p_3) \geq D(p_1, p_2) + D(p_2, p_3)$

Hay múltiples posibles distancias. Las más habituales son:

- La distancia euclídea
  - $D_e(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
  - La función de distancia más conocida
  - Muy costosa computacionalmente para los requerimientos de MONICOD
- La distancia rectangular o Manhattan
  - $D_R(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$
- La distancia Tchebichev
  - $D_T(p_1, p_2) = \text{máx}\left(|x_1 - x_2|, |y_1 - y_2|\right)$

A.3.2.3. Conectividad

Para nuestros efectos, dos píxeles están conectados si comparten una propiedad (relacionada con el nivel de gris) y es posible trazar un camino de uno a otro de longitud 0(píxeles vecinos) o pasando sólo por píxeles que también comparten dicha propiedad.

El concepto de conectividad es especialmente importante en la fase de segmentación. las áreas segmentadas se comprenden de píxeles conectados.

A.3.3. Resolución, profundidad y definición de imagen

La resolución determina cuánto detalle tenemos en una imagen sobre la escena capturada. Para ello se divide la proyección bidimensional de la escena en una malla rectangular compuesta por celdas iguales. Asignando cada celda de la malla a un píxel independiente, la correspondencia del píxel con el tamaño del área física que cubre en la escena, y la cantidad de píxeles que tenemos de la escena, nos proporciona la resolución. La profundidad de la imagen es una medida de la precisión con que cada píxel almacena lo que ocurre, por ejemplo el color, en ese área física que cubre.

Dada una superficie que se captura en una imagen, mayor resolución supone un conocimiento más perfecto de la escena retratada. Esto es, mejor definición de imagen.

En MoniCOD la resolución será 640×480.

A.3.4. Histogramas

El popular histograma de una imagen es una representación gráfica[3] del número de ocurrencias con la que los niveles de color-nivel de gris aparecen en dicha imagen.

Por ejemplo en una imagen en grises (del tipo que manipula MONICOD) el eje de abscisas indica los distintos niveles (discretos) de grises y el eje de ordenadas el número de ocurrencias. De manera que proporciona información estadística sobre la distribución de los diferentes niveles de grises. Ver A.6.

Para construir el histograma basta barrer la imagen contabilizando el número de píxeles que poseen cada nivel de gris. Un simple recuento del número de ocurrencias de aparición en la imagen de cada valor de color. Es una operación extremadamente eficiente que envuelve un número conocido de antemano: $Alto \times Ancho$

Es importante dejar patente que los histogramas son representaciones construidas con la información del color que nos provee la imagen, pero ignorando la información de la posición donde esa ocurrencia de color se produce. No es posible, sólo con la información del histograma, reconstruir la imagen. El histograma corresponde al procesamiento de imagen en el dominio espacial.

El histograma de niveles de intensidad o grises juega un papel importante en MONICOD. Podemos encontrar el pseudocódigo que lo calcula en el apartado 7.2.2.

Puede entenderse el histograma como una función discreta que representa el número de píxeles en la imagen en función de los niveles de intensidad[26].La probabilidad $P(g)$ de ocurrencia de un determinado nivel de gris g se define como:

[3] Aunque puede prescindirse de 'representarlo gráficamente' para poder sacar partido de su información.

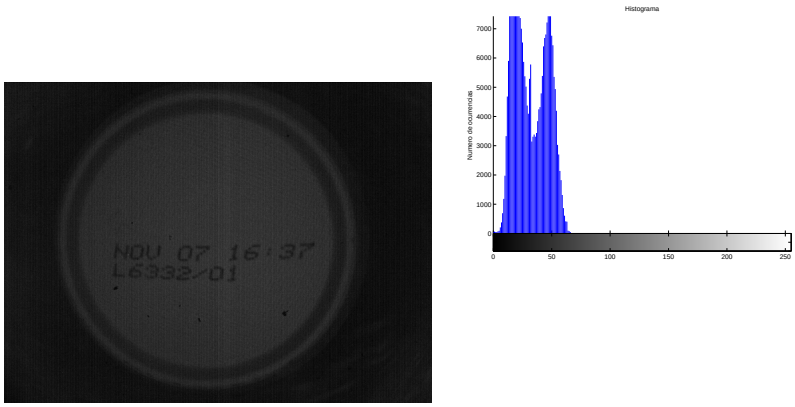


Figura A.6: Una lata y su histograma

$$P(g) = \frac{N(g)}{M} \tag{A.3}$$

donde $N(g)$ es el número de píxeles con el nivel de intensidad g y M el número total de píxeles en la imagen.

Como es habitual se cumple para todo g que $P(g) \leqslant 1$ y $\sum_{g=0}^{L-1} P(g) = 1$ donde L es el número total de posibles de niveles de gris. Por convención los niveles de gris se disponen como $[0, 1, 2, 3..., L-1]$ en orden creciente de intensidad.

A.3.4.1. Propiedades estadísticas del histograma

Las siguientes propiedades estadísticas del histograma son interesantes para conocer la imagen representada complementando a la información cualitativa de la representación gráfica (ver figura A.7):

Media Valor medio de los niveles de gris. Es una medida del brillo general de la imagen.

$$\overline{g} = \sum_{g=0}^{L-1} gP(g) \tag{A.4}$$

A mayor media, mayor brillo.

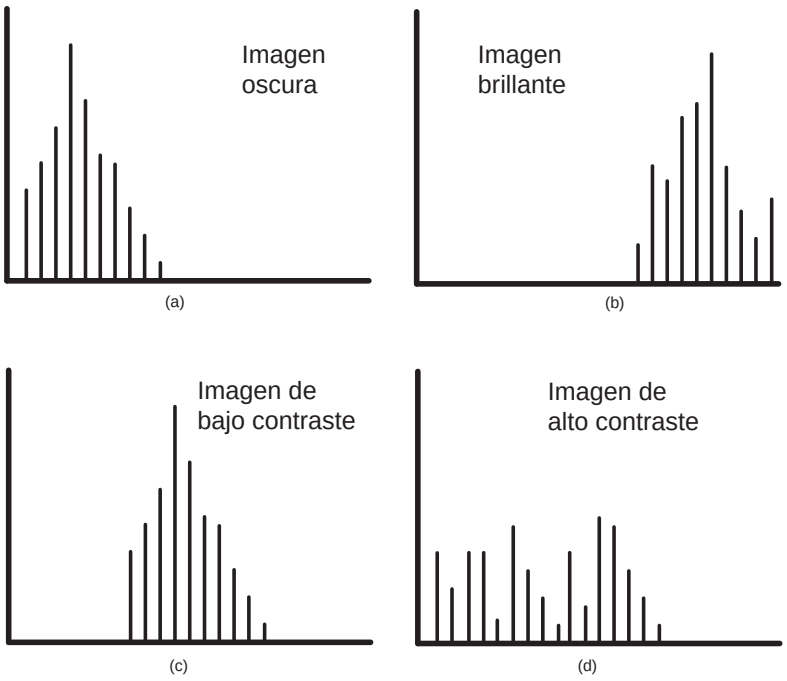


Figura A.7: Información cualitativa del histograma

Varianza Mide la dispersión de los alrededores de la media.

$$\sigma^2 = \sum_{g=0}^{L-1} (g - \overline{g})^2 P(g) \tag{A.5}$$

Una varianza alta indica contraste alto.

Asimetría Es una medida de la simetría de la distribución de los niveles de gris en torno a la media.

$$a = \sum_{g=0}^{L-1} (g - \overline{g})^3 P(g) \tag{A.6}$$

Un valor absoluto en a indica una asimetría alta.

Energía Energía de la distribución de grises.

$$E = \sum_{g=0}^{L-1} (P(g))^2 \tag{A.7}$$

El valor máximo de la energía es 1 y ocurre cuando en la imagen sólo aparece un nivel de gris. A medida que en la imagen hay presentes mayor variedad de niveles de gris la energía disminuye.

Entropía Entropía de la distribución de grises.

$$e = -\sum_{g=0}^{L-1} P(g) \log_2 [P(g)] \tag{A.8}$$

La entropía es mayor a medida que la variedad de niveles de grises presentes en la imagen es mayor. La entropía tiende a comportarse de forma inversa a la energía.

A.3.5. Campo de entrada

El campo de entrada en MONICOD es la escena capturada. Las dimensiones del campo de entrada se mantienen constantes durante la validación, siendo ajustadas en tiempo de calibrado.

A.3.6. La convolución

La convolución es una operación que asigna un nivel de color -nivel de gris para MONICOD- de los píxeles de la imagen considerando los niveles de color (gris) de los píxeles de su entorno de vecindad. Es fundamental para el procesado de la imagen, y en particular, para la eliminación de ruido y detección de bordes.

Ver figura A.8La convolución corresponde al procesamiento de imagen en el espacio de frecuencia.

El producto de convolución entre dos señales bidimensionales discretas se define como:

$$g(i,j) = f \otimes h = \sum_m \sum_n f(i-m, j-n)h(m,n) \tag{A.9}$$

donde los sumatorios se extienden sobre el área de solapamiento de dos señales.

En esta formulación una de las señales, por ejemplo f, es la imagen a procesar mientras que la función h es denominada máscara o filtro[4] de convolución. Las máscaras pueden tener tamaños diferentes. Desde la clásica 3×3 hasta tamaños significativamente mayores (por ejemplo 63×63)

A pesar de su simplicidad se trata de una operación que involucra cálculo intensivo a medida que el tamaño de la máscara es mayor. De ahí que exista abundante hardware "especializado" en convolucionar imágenes.

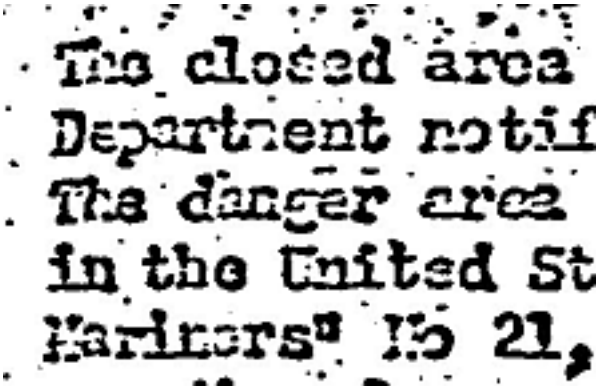


Figura A.8: Una segmentación deficiente originada por un excesivo ruido en la fuente.

A.3.7. Tablas LUT

Una tabla LUT es una estructura de datos(frecuentemente un vector o un vector asociativo) usada para reemplazar un computo en tiempo real por un simple indexado o acceso a memoria. El ahorro en computo puede ser significativo dado que un acceso a memoria suele ser más rápido que ejecutar una operación costosa o una operación de entrada y salida. En MONICOD se utiliza, por ejemplo, para construir una tabla de substitución de grises no ecualizados a grises ecualizados.

[4] De ahí que a menudo se denomine a esta operación *filtrado de imagen*.

Apéndice B

Parámetros MONICOD

En el cuadro B.1 podemos ver un repaso exhaustivo de todos los parámetros de MONICOD, y los valores que mejor comportamiento parecen mostrar hasta el momento de esta publicación.

XElipseCentral

Posición x de la elipse central (que es un circulo).

YElipseCentral

Posición y de la elipse central (que es un circulo).

RadioElipseCentral

El radio de círculo central.

ExcentricidadMaximaExcentricidad

Excentricidad máxima en los extremos. Parece ser que el mejor resultado se obtiene trabajando sólo con elipses. Tampoco hay presencia de distorsión óptica.

NumeroFilasCentrosDeElipses

Numero de filas de elipses en torno a la fila central.

IntervaloFilasCentrosDeElipses

Distancia entre filas de elipses.

IntervaloColumnasCentrosDeElipses

Distancia entre centros de elipses dentro de la fila.

distanciaMinimaEntreElipsesParaElRecuento

Distancia mínima entre activaciones para que haya un recuento.

Parámetro	**Valor**
XElipseCentral	243
YElipseCentral	320
RadioElipseCentral	200.0000
ExcentricidadMaxima	0.0000
NumeroFilasCentrosDeElipses	1
IntervaloFilasCentrosDeElipses	1
IntervaloColumnasCentrosDeElipses	1
distanciaMinimaEntreElipsesParaElRecuento	100
iColumnaMasALaIzquierdaBarridoElipses	50
iColumnaMasALaDerechaBarridoElipses	590
ColorIniElipse	35
ColorFinElipse	255
minPorcentajeContornoClaro	60.000
UmbralNormal	20
maxZonasInundadas	128
maxPuntosZonaInundada	1800
minInundacionesPorBanda	4
maxInundacionesPorBanda	16
nBandasMax	128
nDivisoresSimultaneos	4
UmbralDeCambioDeRumboDeCucaracha	2
PorcentajeDeCaracteresPorBandaNecesariosParaLaConsideracion	50.0
PorcentajeDeCaracteresPorBandaNecesariosParaLaAceptacion	50.0
maxZonasInundadasPorGrupo	16
distanciaMaximaEnYEntreDosCaracteresFusionables	5
UmbralAgrupamiento	0.0
UmbralFision	15
minPuntosPorCaracter	100
maxPuntosPorCaracter	800
AlturaMaximaCaracter	40
AnchuraMaximaCaracter	40
UmbralDeCorrelacionCompleta	0.8999
UmbralDeCorrelacionParcial	0.8000
UmbralDeNuevaAdmision_Aprendizaje	0.8999
UmbralDeNuevoIngreso_Aprendizaje	0.8000
minVotosTotalesPorFamilia_Aprendizaje	1
PorcentajeDeVotosDeAdmision_Aprendizaje	0.0
PorcentajeDePlantilla	30.0
maxMorfologiasDisponibles	1000
maxMorfologiasPorFamilia	10
PorcentajeDeExitoGlobalEstricto	80.0
PorcentajeDeExitoImportanteEstricto	80.0
PorcentajeDeExitoGlobalRelajado	80.0
PorcentajeDeExitoImportanteRelajado	80.0

Cuadro B.1: Parámetros de Hojalata de Histogramas Elípticos

iColumnaMasALaIzquierdaBarridoElipses

Columnas más bajas que esta no tienen elipses. (Se evitan elipses en los extremos)

iColumnaMasALaDerechaBarridoElipses

Columnas más altas que esta no tienen elipses(Se evitan elipses en los extremos)

ColorIniElipse

Valor mínimo en el nivel de color para que cuente como píxel activo en el recinto de contorno de la elipse.

ColorFinElipse

Valor máximo en el nivel de color para que cuente como píxel activo en el recinto de contorno de la elipse.

minPorcentajeContornoClaro

Porcentaje de píxeles activos en el recinto de contorno de la elipse para considerarse activo.

UmbralNormal

Umbral de color que termina cuando un pixel es "semilla". Esto es, algo que podría ser tinta.

maxZonasInundadas

Número máximo de zonas de inundación(fragmentos diferentes. Es necesario para la gestión de recursos.

maxPuntosZonaInundada

Número de píxeles máximo que puede albergar una zona inundada/fragmento..

minInundacionesPorBanda

Número mínimo de fragmentos para que una banda sea considerada.

maxInundacionesPorBanda

Número máximo de fragmentos para que una banda sea considerada.

nBandasMax

Número máximo de bandas.

nDivisoresSimultaneos

Número de divisores que pueden existir simultáneamente.

UmbralDeCambioDeRumboDeDivisor

Umbral a partir del cual el divisor se divide en direcciones oblícuas o perpendiculares.

PorcentajeDeCaracteresPorBandaNecesariosParaLaConsideracion

Porcentaje de caracteres a partir del cual se considera la banda.

PorcentajeDeCaracteresPorBandaNecesariosParaLaAceptacion

Porcentaje de caracteres a partir del cual se acepta la banda.

maxZonasInundadasPorGrupo

Máximo número de fragmentos que pueden ser empaquetados en un grupo.

distanciaMaximaEnYEntreDosCaracteresFusionables

Distancia máxima entre dos fragmentos agrupables.

UmbralAgrupamiento

Grado de solapamiento vertical a partir del cual se considera el agrupamiento.

UmbralFision

Umbral que se aplicará en la nueva asociación de tinta sobre los puntos del fragmento que se fisiona.

minPuntosPorCaracter

Mínimo puntos que debe tener un caracter.

maxPuntosPorCaracter

Máximo de puntos que puede tener un caracter.

AlturaMaximaCaracter

Altura máxima de un caracter legal.

AnchuraMaximaCaracter

Anchura máxima de un caracter legal.

UmbralDeCorrelacionCompleta

Umbral de correlación que debe superarse para declararse correspondencia completa.

UmbralDeCorrelacionParcial

Umbral de correlación que debe superarse para declararse correspondencia parcial.

UmbralDeNuevaAdmision

El umbral a partir del cual una plantilla puede ser asimilada como un voto a una plantilla existente.

AprendizajeUmbralDeNuevoIngreso

El umbral a partir del cual una plantilla puede ser ingresada en la familia morfológica.

AprendizajeminVotosTotalesPorFamilia_

Mínimo de votos totales para que una familia morfológica se considere "aprendida".

AprendizajePorcentajeDeVotosDeAdmision

Lo mismo que **UmbralDeNuevaAdmision** para Aprendizaje

AprendizajePorcentajeDePlantilla

Porcentaje de votos dentro de la familia morfológica por encima del cual la plantilla no será purgada.

maxMorfologiasDisponibles

Número máximo de morfologías disponibles para ser utilizadas.

maxMorfologiasPorFamilia

Número máximo de morfologías que puede haber en una familia.

PorcentajeDeExitoGlobalEstricto

Porcentaje de caracteres total que deben haber sido reconocidos para dar el código por validado.

PorcentajeDeExitoImportanteEstricto

Porcentaje de caracteres importantes que deben haber sido reconocidos para dar el código por validado. Es de esperar que este parámetro siempre sea el 100%

PorcentajeDeExitoGlobalRelajado

Porcentaje de caracteres total que deben haber sido reconocidos para dar el código por parcialmente validado.

PorcentajeDeExitoImportanteRelajado

Porcentaje de caracteres importantes que deben haber sido reconocidos para dar el código por parcialmente validado.

Parámetro	Valor
XElipseCentral	243
YElipseCentral	320
RadioElipseCentral	190.0000
ExcentricidadMaxima	0.0000
NumeroFilasCentrosDeElipses	1
IntervaloFilasCentrosDeElipses	1
IntervaloColumnasCentrosDeElipses	1
distanciaMinimaEntreElipsesParaElRecuento	32
iColumnaMasALaIzquierdaBarridoElipses	610
ColorIniElipse	25
ColorFinElipse	255
minPorcentajeContornoClaro	70.000

Cuadro B.2: Parámetros de Hojalata de Histogramas Elípticos

Parámetro	Valor
maxZonasInundadas	128
maxPuntosZonaInundada	1800
maxZonasInundadasPorGrupo	16
minPuntosPorCaracter	100
maxPuntosPorCaracter	800
UmbralNormal	14
AlturaMaxima	35
AnchuraMaxima	35

Cuadro B.3: Parámetros de Asignador de Tinta (Hojalata)

Parámetro	Valor
minInundacionesPorBanda	2
maxInundacionesPorBanda	14
nBandaMax	256
nDivisoresSimultaneos	8
UmbralDeCambioDeRumboDeCucaracha	2

Cuadro B.4: Parámetros de bandas(hojalata)

Parámetro	Valor
UmbralAgrupamiento	0.0000
UmbralFision	15

Cuadro B.5: Parámetros de agrupamiento (hojalata)

Parámetro	Valor
AltoMaxMorfologia	30
AnchoMaxMorfologia	30
maxMorfologiasPorFamilia	8
maxMorfologiasDisponibles	512
numeroMaximoFamiliasMorfologicas	256
minimosVotosEnFamiliaMorfologica	1
UmbralNuevoVoto	0.99
UmbralNuevoIngreso	0.9
UmbralPrevioParaAprobarAnalisis	0.1
minPorcentajeDeVotosTotalesParaSupervivenciaDeUnaMorfologiaDentroDeUnaFamilia	5.0

Cuadro B.6: Parámetros de TM (1) (MONICOD)

Parámetro	Valor
AltoMaxMorfologia	30
AnchoMaxMorfologia	30
maxMorfologiasPorFamilia	4
maxMorfologiasDisponibles	512
numeroMaximoFamiliasMorfologicas	256
minimosVotosEnFamiliaMorfologica	1
UmbralNuevoVoto	0.99
UmbralNuevoIngreso	0.9
UmbralPrevioParaAprobarAnalisis	0.9
minPorcentajeDeVotosTotalesParaSupervivenciaDeUnaMorfologiaDentroDeUnaFamilia	5.0

Cuadro B.7: Parámetros de TM (2) (MONICOD)

Parámetro	Valor
AltoMaxMorfologia	30
AnchoMaxMorfologia	30
maxMorfologiasPorFamilia	1
maxMorfologiasDisponibles	512
numeroMaximoFamiliasMorfologicas	256
minimosVotosEnFamiliaMorfologica	1
UmbralNuevoVoto	0.99
UmbralNuevoIngreso	0.9
UmbralPrevioParaAprobarAnalisis	0.1
minPorcentajeDeVotosTotalesParaSupervivenciaDeUnaMorfologiaDentroDeUnaFamilia	5.0

Cuadro B.8: Parámetros de TM (3) (MONICOD)

Parámetro	**Valor**
NumeroDeEntradas	7
K	1

Cuadro B.9: Parámetros de KNN

Parámetro	**Valor**
TipoDeRed	0
Bipolaridad	0
AlgoritmoDeAprendizaje	0
NumeroDeCapas	3
NumeroDeNeuronasPorCapa	*[7, 5, 38]*
Bias	*1.0*
Etha	*0.5*
FuncionDeActivacionOculta	*2.0*
AlphaOculta	*1.0*
BethaOculta	*0.0*
FuncionDeActivacionSalida	*2.0*
AlphaSalida	*1.0*
BethaSalida	*0.0*
IteracionesDeEntrenamiento	*10000*
EntradaMin	*0.1000000015*
EntradaMax	*0.8999000192*

Cuadro B.10: Parámetros de MLP

Parámetro	Valor
AlgoritmoDeAprendizaje	0
NumeroDeCapas	3
NumeroDeNeuronasPorCapa	*[7, 5, 38]*
FuncionDeActivacionOculta	4.0
AlphaOculta	1.0
FuncionDeSalidaLineal	true
FuncionDeActivacionSalida	4.0
AlphaSalida	1.0
EntradaMin	*0.1000000015*
EntradaMax	*0.8999000192*
pInit	*1000.0*
Sigma	*0.0*
Eta	*0.009800000116*
Eta_C	*0.009800000116*
Eta_W	*0.009800000116*
Eta_S	*0.001000000047*
IteracionesDeEntrenamiento	*10000*
seaplicaclusteringksom	true

Cuadro B.11: Parámetros de RBF

Parámetro	Valor
svm_type	1
kernel_type	2
degree	5
nr_weight	0
weight_label	[]
weight	[]
gamma	5.0
coef0	0.5
cache_size	200.0
eps	0.001
C	1.0
nu	0.1
p	0.0
shrinking	1
probability	0
numerodeentradas	7
nrfold	2

Cuadro B.12: Parámetros de SVM

Apéndice C

Comparación bit a bit frente a comparación entero a entero

C.1. Condiciones

Condiciones análogas a los experimentos previos:

- La misma máquina
- El mismo sistema operativo
- El mismo compilador

C.1.1. Métodos contendientes

- Comparación entera de plantillas (sin optimizar)
- Comparación entera de plantillas optimizada
- Comparación bit a bit de plantillas empleando el bitset estático que nos provee la STL[1]
- Comparación bit a bit de plantillas empleando el bitset dinámico que nos provee la librería Boost[111].

El resultado de comparación por los cuatro métodos. Son intercambiables.

C.1.2. Procedimiento

- Se compara un par fijo de plantillas de 35×35
- La comparación se lleva a cabo 512000 veces (que corresponde a una tarea de reconocimiento de 8000 plantillas con una BFM de 8 plantillas en 8 familias morfológicas)

[1] Standard Template Library

C.1.3. Resultados

En la figura C.1 se aprecia una comparativa en las mismas condiciones citadas en experimentos anteriores.

C.1.4. Conclusiones

La comparación bit a bit se prueba **al menos dos veces más rápida** que la comparación entera en el escenario más adverso.

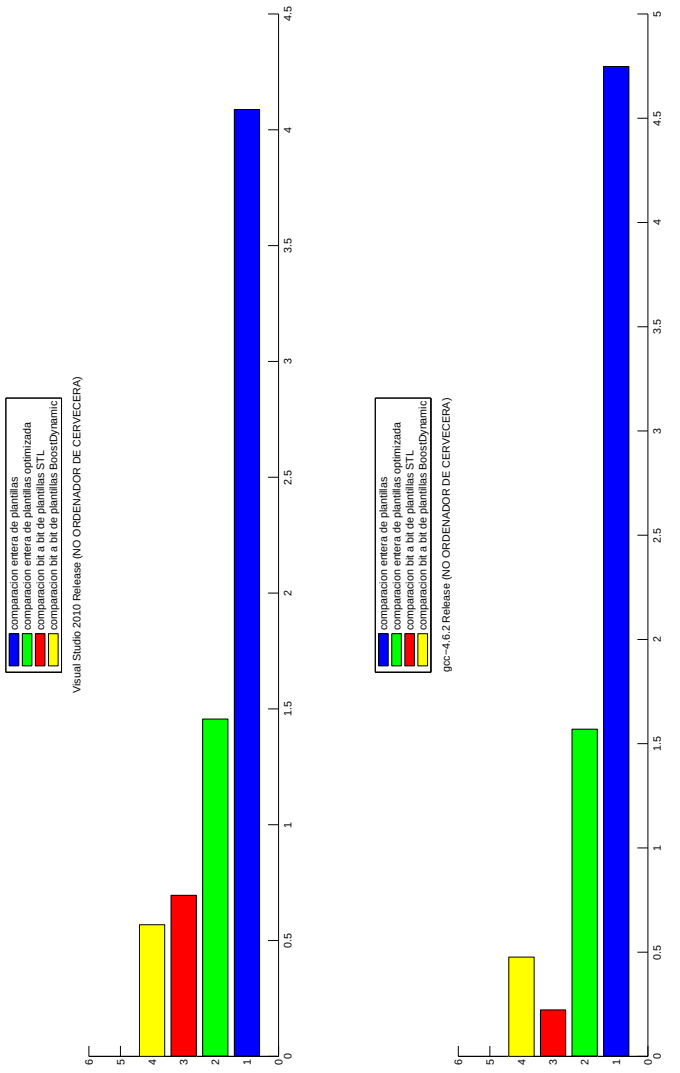


Figura C.1: Costos para diferentes métodos de comparación de plantillas.

Índice de figuras

Índice de cuadros

Índice de algoritmos