Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño y desarrollo de un sistema de seguimiento de objetivos móviles con Raspberry Pi

Autor:

Javier Santana Godoy

Tutor:

Pedro Medina Rodríguez

Julio de 2018

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Javier Santana Godoy, autor del Trabajo de Fin de Título "Diseño y desarrollo de un sistema de seguimiento de objetivos móviles con Raspberry Pi", correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) _____

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[x] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 4 de Julio de 2018.

El estudiante

SANTANA GODOY JAVIER - 45782796P

Firmado digitalmente porSANTANA GODOY JAVIER - 45782796P
DN: cn=SANTANA GODOY JAVIER - 45782796P gn=JAVIER c=ES
Motivo:
Ubicaci· n:
Fecha:2018-07-04 23:04+02:00

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[x] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Firmado por MEDINA RODRIGUEZ PEDRO - 42793260N el día 04/07/2018 con un certificado emitido por AC FNMT Usuarios

Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A mis padres, ya que sin el apoyo y la educación que me han dado no hubiera podido llegar hasta aquí. A mis abuelos y mi hermana, ya que han sido los que me han sacado una sonrisa en los momentos más duros de estos años.

También me gustaría agradecer a mi tutor, Pedro Medina, ya que sin sus consejos y ayuda no hubiera terminado nunca este proyecto.

Por último, a mis amigos, esas personas que han sido compañeras de estudios en Arquitectura hasta altas horas de la madrugada, ya que, con compañía, todo esfuerzo es más fácil de llevar.

Resumen

Este proyecto surge de la voluntad de realizar un sistema automático de detección y seguimiento de objetivos móviles, con la finalidad de que pueda ser utilizado en actividades lúdico-deportivas de estrategia basadas en la simulación militar, como pueden ser el airsoft o el paintball.

El objetivo en este tipo de actividades es el conseguir un impacto en uno de los componentes del equipo contrario. El sistema desarrollado será capaz de seguir al objetivo de forma automática, funcionado de manera similar al sistema de apuntado de una torreta no tripulada.

Abstract

This project arises from the desire of creating an automatic system for detecting and monitoring targets in movement, which is designed to be used in sports or activities, such as airsoft and paintball.

The objective in this type of activities is to achieve impacts on the components of the opponent team. The developed system should be able to follow the target automatically, working in a similar way to a no tripulated sentry gun.

Índice General

Índice de figuras

Índice de tablas

1. Introducción

1.1 – Estado del arte

Hoy en día es cada vez más frecuente ver proyectos caseros que incorporan ordenadores de una sola placa (SBC, por sus siglas en inglés). Esto es debido a que se trata de una tecnología cada vez más accesible, y es una puerta abierta para iniciar a los más jóvenes o a personas con pocos recursos en la informática. Este auge del conocido como "movimiento maker" ha provocado que surjan comunidades de usuarios que comparten el interés por desarrollar proyectos personales, como Hackster.io [1], enfocada exclusivamente a proyectos realizados con componentes informáticos o electrónicos, o Instructables [2], en cuya web tienen cabida todo tipo de actividades, desde recetas de cocina hasta servidores caseros.

Esta comunidad ha potenciado el concepto "D.I.Y", o "Do It Yourself", en español "Hágalo usted mismo", concepto definido por la edición inglesa del diccionario de Cambridge como "la actividad de decorar o reparar tu casa, o de hacer cosas para tu casa tu mismo, antes que pagarle a alguien para que lo haga por ti" [3].

Esta corriente ha generado las comunidades mencionadas anteriormente, en las que usuarios que tienen una idea que no tiene una solución comercial buscan soluciones propuestas por la comunidad, o exponen las suyas propias en busca de consejo. También son frecuentes los casos en los que, aun existiendo productos funcionales a la venta, los usuarios prefieren hacer ellos mismos el proyecto, ya que, en ocasiones, prima la satisfacción personal por encima del resto de factores.

En el mundo del Airsoft [4] (actividad lúdica similar al paintball), es habitual el uso de dispositivos caseros, llegando a niveles sorprendentes, como podemos ver en la siguiente imagen, un "explosivo" utilizado para añadir interés a las partidas por objetivos.

Figura 1.1 – Réplica de sistema explosivo, elaborada con Arduino

Como usuario habitual de estas comunidades y practicante del Airsoft, tenía claro que el proyecto debía ser algo que pudiera usarse en esta actividad. A su vez, la visión por computador me resulta un campo de estudio muy interesante, así que pensé en un proyecto que incluyera ambos conceptos.

De esta forma, se escogió el trabajo a desarrollar en este TFG, un sistema que permitiera detectar enemigos a través de una webcam, y seguirlos mediante el uso de motores paso a paso,

para conseguir así un sistema base de autoapuntado que pudiera ser utilizado luego para controlar una réplica de Airsoft.

Al explorar la oferta comercial existente, se comprobó que es reducida y de coste muy elevado, por lo que realizar una versión "D.I.Y." era algo interesante, ya que nos daba dos objetivos: reducir costes y conseguir integrar todos los componentes necesarios.

En cambio, en la comunidad maker sí que existe un cierto número de sistemas similares, aunque en su mayoría utilizan juguetes de la marca Nerf [5], empresa que fabrica unos lanzadores de dardos de gomaespuma.

El hecho de que ya existieran proyectos similares nos indica que es posible realizar el proyecto propuesto con los materiales escogidos, ya que hay usuarios que lo han conseguido anteriormente.

Por estos motivos, los factores con mayor peso a la hora de escoger el proyecto fueron la certeza de que era posible unida a la escasez de oferta comercial.

1.2 – Motivación

Como practicante habitual de una de las actividades lúdicas antes mencionadas (Airsoft), he pensado en numerosas ocasiones que este tipo de sistemas le añadirían un interés renovado al juego, ya que te obligan a pensar en mayor medida tus desplazamientos, al ser imposible averiguar de antemano si el otro equipo ha colocado un dispositivo de este tipo.

A su vez, también influyó en la elección de este trabajo la convicción personal de que sería más interesante realizar un trabajo que combinara software y hardware, y no fuera solo la parte más lógica y teórica de la informática, que es a lo que habitualmente nos dedicamos en Ingeniería del Software.

Otro aspecto a tener en cuenta fue la satisfacción personal, ya que durante la carrera nunca hemos tenido que diseñar un proyecto completo, sin ningún tipo de indicación o requisitos ya definidos por un profesor. Ser el usuario y el desarrollador nos hace tener en cuenta todos los aspectos, desde el coste de los materiales hasta la estructura del proyecto o la resistencia a los fallos que puedan darse durante la ejecución.

También fue relevante el hecho de que el proyecto pudiera realizarse en Python. Como se analiza posteriormente en este mismo documento, Python es uno de los lenguajes con mayor crecimiento en los últimos años. Un buen ejemplo de ello es el hecho de que algunos proyectos muy interesantes, como Google Assistant [6], o CodeCombat [7] (web para enseñar programación a los más jóvenes) están escritos en este lenguaje.

Por todas estas razones, se consideró que este proyecto es una buena oportunidad para poner en práctica todo lo aprendido a lo largo de estos años realizando el Grado en Ingeniería Informática.

1.3 – Objetivos

Para llegar a ser funcional y dar los resultados para los que ha sido diseñado, este sistema deberá cumplir los siguientes requisitos:

1. **Ser de bajo coste,** lo que ha llevado a la elección de la Raspberry Pi como componente informático.
2. **Ser capaz de funcionar sin intervención humana**, una vez encendido y calibrado.
3. **Ser de fácil transporte y montaje,** no requiriendo herramientas para su uso una vez construido.
4. **No caer en errores de ejecución que puedan llevar a su desactivación involuntaria,** para así asegurar su completa funcionalidad.

1.4 – Estructura del documento

En la primera parte de este documento, se realizará una introducción del trabajo realizado, así como de las motivaciones para ello.

Después de analizar las competencias y aportaciones cubiertas por el mismo, se procederá a explicar cómo se ha realizado el trabajo, reflejando aspectos como la metodología utilizada, los recursos utilizados y el porqué de la elección de estos, así como relatando los pormenores del proceso final de desarrollo y montaje.

Para terminar, se reflejarán los resultados obtenidos, los posibles trabajos futuros y las conclusiones obtenidas de dichos resultados, así como las fuentes de la información utilizada y otra información de interés, como el manual de usuario o los detalles del diseño de las piezas.

2. Competencias y aportaciones

2.1 - Competencias

La elaboración de este proyecto ha llevado a cubrir una serie de competencias profesionales definidas en el proyecto docente de la asignatura. Por definición, la realización del proyecto final del grado cumple todas las competencias generales, aunque en este caso destacan las siguientes:

- **CII01 –** Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.
- **IS01 -** Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

La propia elaboración de este Trabajo de Fin de Grado cumple la última competencia necesaria, es decir, la **TFG01**:

- **TFG01:** Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

2.2 – Aportaciones

A nivel personal, la realización de este proyecto me ha servido para completar mi formación, ya que es la síntesis de todo lo aprendido a lo largo de la realización del Grado en Ingeniería Informática.

Generalmente, a lo largo de la carrera, cuando hacemos desarrollos complejos, contamos con uno o más compañeros, por lo que la carga de trabajo queda dividida al menos entre dos personas, y es probable que aquello en lo que uno de los componentes no es muy hábil pueda ser solventado por el resto del equipo. Esta es la primera vez que no ha sido así, ha sido un trabajo completamente autónomo y personal.

Fue algo que se notó especialmente la hora de tomar decisiones, ya que era complejo pedirle consejo a un compañero. Cada proyecto se enfoca a un aspecto distinto, y en muchos casos sólo nosotros sabemos de lo que estamos hablando, ya que ha sido nuestro trabajo durante meses. Gracias a esto, me he dado cuenta de que estamos capacitados para llevar a cabo proyectos extensos de forma completamente autónoma, siempre y cuando sea necesario.

La experiencia ha sido gratificante, porque es la prueba de que la formación recibida a lo largo de estos años nos sirve para resolver un problema real, propuesto por nosotros, que no está guiado ni tiene indicaciones sobre cuál puede ser el camino correcto. Ha sido un proceso difícil, ya que estoy acostumbrado a desarrollos software, donde un cambio en los requisitos sólo se traduce en tiempo o en tener que buscar un sistema compatible. En este tipo de proyectos, un cambio en los requisitos lleva a tener que comprar hardware adicional.

Si hablamos de las aportaciones al entorno del entorno, los posibles usuarios son la comunidad del Airsoft. A los jugadores de dicha actividad, este proyecto les proporciona una manera económica de poder construir dispositivos con apuntado automático para sus partidas, ya que el código y el manual de usuario para utilizarlo estarán disponibles de forma pública una vez realizada la defensa de este proyecto.

A su vez, ha sido necesario desarrollar dos librerías para manejar desde Python los drivers utilizados (L298N al principio y A4988 como elección final), por lo que, indirectamente, podrá servir de ayuda a cualquier proyecto que utilice Python y motores paso a paso, ya que podrán aprovechar las librerías.

Otro colectivo que se beneficia con esta aportación es la comunidad “maker”, ya mencionada anteriormente, que encuentra de gran interés la mayoría de los proyectos realizados autónomamente, y podrá aprovechar parte del código utilizado o de los conceptos empleados.

3. Metodología y plan de trabajo

3.1 – Metodología

Para elaborar este proyecto, comenzamos realizando una planificación en la que luego basamos el desarrollo, para así tener claros en todo momento los objetivos a conseguir, y evitar que se implementaran aspectos del software que no fueran necesarios.

A la hora de organizar el desarrollo del software, debido a los conocimientos adquiridos en la carrera, se redujo la elección a tres posibles modelos: desarrollo en cascada, desarrollo iterativo-incremental y SCRUM.

3.1.1 – Desarrollo en cascada

Según Wikipedia, el desarrollo en cascada [8] es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al finalizar cada etapa, se realiza una revisión, en la que se determina si el proyecto está listo para avanzar a la siguiente fase.

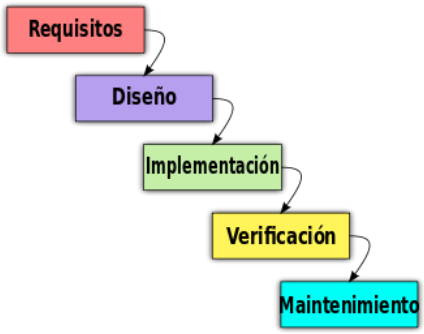


Figura 3.1 – Desarrollo en cascada

Como modelo de desarrollo tiene aspectos positivos, ya que es fácil de implementar y comprender, ya que es conocido y se utiliza con frecuencia, pero en la vida real rara vez un proyecto sigue una secuencia lineal, lo cual puede llevar a este método al fracaso.

Al no tener experiencia previa en el desarrollo de este tipo de proyectos, era imposible seguir una estructura cerrada, ya que se no se conocía en profundidad el funcionamiento de parte de los componentes utilizados, por lo que sería necesario aprender sobre la marcha. Por esta rigidez que posee el modelo, fue descartado como método de organización para el proyecto.

3.1.2 – Scrum

En la Ingeniería del Software, conocemos como Scrum [9] a los marcos de desarrollo ágiles caracterizados por:

- Adoptar una estrategia de desarrollo incremental
- Basar la calidad del resultado en el conocimiento tácito de las personas en equipos auto organizados.
- Solapamiento de las diferentes fases del desarrollo.

Hoy en día, es habitual ver el uso de Scrum en empresas de desarrollo de software, así como exigir su conocimiento como requisito a la hora de contratar nuevos empleados.

Tiene ventajas evidentes frente al modelo en cascada, como su flexibilidad ante el cambio de requisitos, o las revisiones continuas del trabajo realizado, así como los incrementos continuos de funcionalidad.

A pesar de estas ventajas, fue descartado para la realización de este trabajo porque hace uso de una serie de figuras, como el **Product Owner (**la persona encargada de que el equipo Scrum trabaje de forma adecuada desde la perspectiva de negocio) o el **ScrumMaster** (facilitador que se encarga de eliminar los obstáculos que se le presentan al equipo para terminar el sprint) que en este caso recaerían sobre la misma persona, perdiendo gran parte de su sentido.

También serían difíciles de implementar otros conceptos como el **Daily Meeting** (una pequeña reunión diaria para organizar el trabajo del día), ya que el equipo está formado por una sola persona.

Por estos motivos, y a pesar de considerarse una metodología ideal para el desarrollo de software, en esta ocasión tuvo que ser descartada.

3.1.3 – Modelo iterativo incremental

Este modelo surgió en respuesta a las debilidades del modelo tradicional en cascada. El desarrollo iterativo e incremental [10] consiste en un conjunto de tareas agrupadas en pequeñas etapas repetitivas (iteraciones).

En cada iteración se repite un determinado proceso de trabajo que brinda un resultado que se vuelve más completo con cada iteración, de forma que se va mejorando el producto con cada iteración que se realiza. Con esta metodología, lo que se busca es que en cada iteración se logre evolucionar el producto basándose en los logros de las etapas anteriores.

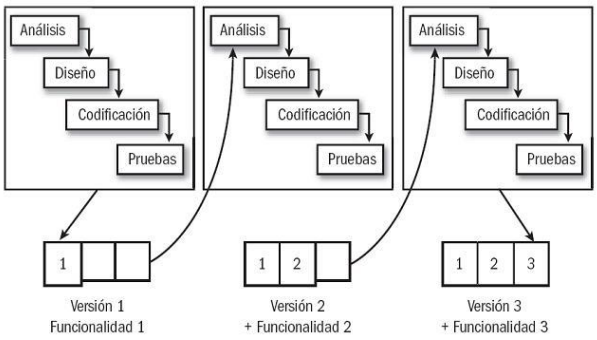


Figura 3.2 – Modelo iterativo incremental

Se permite el añadido de requisitos en cada iteración, lo que nos proporciona la flexibilidad que no nos ofrece el modelo en cascada, y a su vez evita las figuras y reuniones difíciles de implementar cuando el equipo está formado por un solo desarrollador, por lo que fue elegido como base de la organización del proyecto.

Para ello, se definieron una serie de puntos clave en el proyecto (es decir, de objetivos para cada iteración), con una complejidad que iba aumento gradualmente, de tal forma que cada uno de estos puntos clave le daba un valor añadido al desarrollo.

Ante la ausencia de un cliente que pudiera proporcionarnos retroalimentación sobre los avances, se probó cada fase individualmente, comparando los resultados obtenidos con los esperados, y evitando pasar a la fase siguiente sin haber verificado la anterior.

Por la experiencia en desarrollos anteriores, así como por la intención futura de poner el código generado disponible de forma pública en un repositorio, se decidió seguir en la medida de lo posible la filosofía Clean Code[11], por lo que, al finalizar cada etapa, se añadió una fase de refactorización, en la que se adaptó, modularizó y mejoró el código para hacer más fácil su lectura y comprensión por parte de otros programadores-

3.2 – Planificación del trabajo

Las primeras etapas del proyecto se emplearon en el estudio de las alternativas disponibles en cuanto a librerías y lenguajes de programación, así como en la investigación de posibles componentes hardware para el sistema, y su posterior elección.

Para elegir el lenguaje a utilizar en el proyecto, se comenzó investigando proyectos de visión por computador, y se comprobó que existen dos lenguajes predominantes: C++ y Python.

Ya que no se poseían conocimientos de ninguno de los dos lenguajes, iba a ser necesario el aprendizaje en ambos casos, así que la experiencia previa no fue un aspecto relevante en la elección.

El siguiente aspecto que se tuvo en cuenta fue la utilidad futura del lenguaje que se iba a aprender. Según la encuesta anual que realiza anualmente StackOverflow[12] (una famosa comunidad utilizada habitualmente por programadores para resolución de dudas) en el presente año 2018 Python fue uno de los lenguajes más populares entre los desarrolladores profesionales, ocupando el puesto 7 de 25 en la lista de los lenguajes más utilizados.

Es decir, del total de 73,248 programadores encuestados, el 37.9% de los programadores encuestados utilizan habitualmente Python, frente al 24.6% que afirman utilizar C++.

Ya en el año 2017, en un artículo escrito por David Robinson [13], autor de varias entradas en el blog de StackOverflow, se define Python como “el lenguaje de mayor crecimiento”.

En cambio, si buscamos información en otros índices que reflejen la influencia del software, en el Índice Tiobe [14], C++ aparece por encima de Python. Según su propia web, “el índice TIOBE es un indicador de la popularidad de los lenguajes de programación. El índice se actualiza una vez al mes. Algunos de los motores de búsqueda más populares son utilizados para elaborar este ranking”.

En lo personal, la experiencia me ha enseñado que el aspecto que tiene mayor relevancia a la hora de aprender un lenguaje nuevo es la comunidad que tiene detrás, ya que, entre más gente utilice el lenguaje, más dudas surgirán, y más probable es que le haya surgido la misma duda a otro usuario. Por estos motivos, fue valorada en mayor medida los resultados de la web utilizada habitualmente para resolver dudas (StackOverflow).

Ahora que conocemos las posibilidades en lo que a lenguaje se refiere, pasamos a buscar métodos que permitieran controlar motores paso a paso desde la Raspberry Pi.

Resultó que no había gran cantidad de librerías para usar los controladores más comunes de Arduino desde Python o C++, pero sí que existían dichas librerias para Arduino. El tener unas librerías “base” desde las que partir para desarrollar las propias hizo que en este sentido fuera indiferente el lenguaje escogido para el proyecto.

Por lo que, por interés personal y proyección de futuro, Python fue el lenguaje escogido.

Debido a que solo se tenía un conocimiento superficial de este lenguaje, fue necesario aprenderlo prácticamente desde cero. El proceso comenzó por la lectura de documentación y consejos, para finalmente realizar un curso gratuito ofrecido por la web CodeAcademy [15], que sirvió para coger soltura en el desarrollo.

Una vez finalizado dicho curso, se definieron los objetivos que se debían lograr con el proyecto, y se comenzó su desarrollo. Al mismo tiempo, se procedió de forma gradual a la compra del hardware y el resto de los componentes necesarios, hasta que fueron adquiridos todos los elementos que finalmente se han utilizado.

De los motivos de la elección de los diferentes componentes del hardware hablaremos en el apartado 7, correspondiente a "Diseño e Implementación".

3.3 – Temporización

Teniendo en cuenta los principios utilizados por la metodología del modelo iterativo, se procedió a la elaboración de la siguiente planificación inicial, también necesaria para la documentación realizada al presentar la solicitud del proyecto:

Fases	Duración estimada (horas)	Tareas
Estudio previo / Análisis	30	1.1: Investigación previa
		1.2: Selección de herramientas a utilizar
		1.3: Instalación y adquisición de herramientas
Diseño / Desarrollo / Implementación	180	2.1: Sistema de seguimiento de movimiento
		2.2 – Librería para el uso de los motores
		2.3 – Obtención de las coordenadas de los pasos
Evaluación / Validación / Prueba	60	3.1 – Seguir objetivos en el eje vertical
		3.2: Seguir objetivos en el eje horizontal
		3.3: Seguir a un objetivo en ambos ejes
Documentación / Presentación	30	4.1: Realización de la memoria del TFT
		4.2: Preparación de la presentación

Tabla 3.1 – Planificación inicial del proyecto

Utilizando el calendario anterior como base de la planificación definitiva, se definieron las fases necesarias para lograr la realización del proyecto:

Fases comunes:

COMUNES	
1	Investigación y selección de componentes
2	Compra de los componentes necesarios

Tabla 3.2 – Planificación de las fases comunes del proyecto

Fases relacionadas con el software:

SOFTWARE	
1	Instalación de OpenCV y prueba de cámara
2	Detección básica de movimiento y dibujo de marcadores
3	Detección mejorada de objetivos (Reducir falsos positivos)
4	Marcar solo el objetivo de mayor área
5	Control del motor de la base
6	Control del motor del soporte
7	Relacionar movimiento de los motores con objetivo real
8	Control de ambos motores
9	Mejoras

Tabla 3.3 – Planificación del desarrollo software del proyecto

Como se mencionó en el apartado anterior, al final de cada iteración (es decir, una vez contábamos con una versión funcional del código que nos ofrece una mejora respecto a la versión anterior) se realizó una refactorización y modularización del código, buscando así cumplir con los principios del Clean Code.

Fases relacionadas con el hardware y el montaje:

HARDWARE / MONTAJE	
1	Corte de tablero de DM para la base móvil
2	Impresión 3D de enganches para los motores
3	Sujeción del motor al tablero de DM
4	Compra de la estructura de montaje y colocación de componentes sobre la misma
5	Diseño y corte de madera para el soporte
6	Montaje del motor del soporte
7	Añadido láser
8	Pruebas del sistema completo
9	Mejoras

Tabla 3.4 – Planificación del montaje del proyecto

4. Análisis

4.1 – Aplicaciones similares

Como se mencionó en el apartado 1, a nivel comercial, la oferta es muy reducida. No encontramos sistemas enfocados al seguimiento de objetivos, pero sí encontramos torretas que pueden utilizarlo. Con cierta facilidad podemos ver las torretas manuales, es decir, aquellas que requieren una persona a los mandos para funcionar, pero hay muy pocas torretas de apuntado automático disponibles.

Lo más similar que podemos encontrar a un producto comercial que incorpore un sistema de seguimiento de objetivos móviles son los kits que vende el sitio web RealSentryGun [16], cuyos precios van desde los 49 dólares por su software hasta los 649 dólares por el kit completo.

A juzgar por el contenido multimedia de la web mencionada anteriormente, el rendimiento de estas soluciones es bastante bueno, aunque el coste es mucho mayor que el necesario para realizarlo por nuestra cuenta, y además necesitan un ordenador convencional, requisito no necesario en este proyecto al usar una Raspberry Pi.

En cambio, a nivel "maker", si hay una comunidad mucho mayor de proyectos realizados de este tipo, como una torreta con sistema de apuntado autónomo [17], que utiliza de base una Nerf, asi como otro sistema que puede ser controlado por voz desde Alexa ([18], el asistente inteligente de Amazon), o algunos sistemas que incorporan reconocimiento facial [19], permitiendo así diferenciar entre objetivos y "aliados" .

Gracias a la comunidad "maker" y a los proyectos mostrados en algunas de los foros utilizados por dicha comunidad surgió la idea de realizar este proyecto, sirviendo las opiniones y la experiencia previa de los usuarios de dichos foros de gran ayuda a la hora de escoger componentes o diseñar piezas para el proyecto.

4.2 – Requisitos del usuario

A continuación, se desglosarán los requisitos de usuario, dividiéndose en requisitos funcionales y no funcionales:

1. **Funcionales:**
   - El sistema debe ser capaz de reconocer los objetivos en movimiento.
   - El sistema debe ser capaz de señalar el objetivo de mayor importancia, es decir, el que mayor área ocupe.
   - El sistema debe ser capaz de funcionar sin intervención externa una vez encendido y calibrado.
   - El sistema debe evitar en la medida de lo posible caer en situaciones que provoquen un final no deseado de la ejecución del programa.
2. **No funcionales:**
   - El sistema debe ser económico de construir.
   - El sistema debe ser de fácil transporte.
   - El sistema debe poder ser puesto en marcha por alguien que carezca de conocimientos técnicos en lo que a la Informática se refiere.

4.3 – Interfaz de usuario

Como el sistema desarrollado es un sistema de apuntado autónomo, no requiere de interacción alguna con el usuario, excepto a la hora de calibrar el sistema. Para ello, en el repositorio en el que se encuentra el código del proyecto, en la carpeta "calibrate" podemos encontrar cuatro ficheros que nos ayudan a realizar dicha calibración, "`calibrate_pan.py`" , "`calibrate_tilt.py`" , "`full_calibration.py`" y "`usb_camera_target.py`".

Los dos primeros archivos nos permiten calibrar los motores de la base y el soporte, cada uno por separado.. El tercero ("`full_calibration.py`") nos permite realizar la calibración de ambos motores a la vez, siempre desde el terminal.

El archivo "`usb_camera_target`" nos muestra el streaming procedente de la cámara, con un punto marcado en el centro de la imagen. De esta forma, sabemos el centro exacto de la imagen y nos resulta mucho más fácil calibrar el sistema.

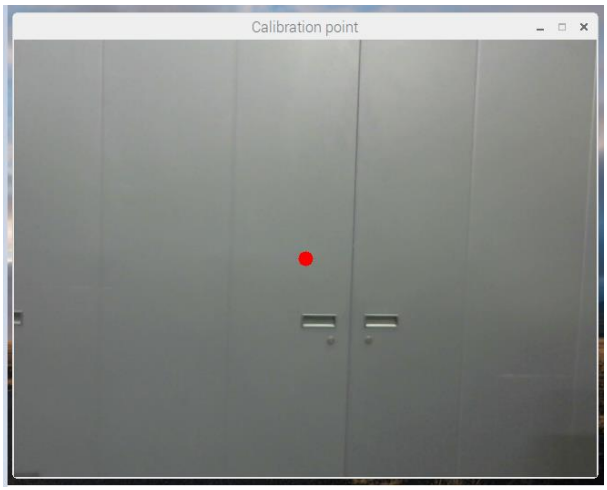


Figura 4.1 – Vista del archivo "usb_camera_target.py"

Se planteó el diseño de una interfaz gráfica para este calibrado, pero no se consideró necesario, ya que uno de los requisitos del sistema es poder funcionar sin una pantalla externa, y la interfaz la haría obligatoria. Con los archivos definidos anteriormente, sólo será necesaria la pantalla si se desea utilizar el "`usb_camera_target.py`" para que nos marque el centro de la imagen. Si prescindimos de esto, todo el proceso podrá realizarse desde un terminal.

Durante el desarrollo (y especialmente en las fases iniciales) sí que fue necesario mostrar por pantalla la ejecución del programa, para comprobar las imágenes captadas por la cámara y su posterior procesamiento. Para ello, utilizando las funciones propias de OpenCV, durante la ejecución del programa se despliegan 3 ventanas, en las que pueden observarse el streaming de vídeo de la cámara, el umbralizado de dicha imagen y el frame delta.

Para facilitar la comprensión de este apartado, es conveniente definir el umbralizado y el frame delta:

Umbralizado / Valor umbral [20]**:**

Los métodos de valor umbral son un grupo de algoritmos cuya finalidad es segmentar gráficos rasterizados, es decir, separar los objetos de una imagen que nos interesen del resto. En este programa se utiliza para distinguir el objeto en movimiento del fondo de la imagen.

Frame Delta [21]**:**

En la codificación interframe de vídeo, el frame delta es el frame que nos muestra la diferencia que provoca un cambio incremental del frame anterior, es decir, es el reflejo de los cambios que se han producido en la imagen. Llevado a nuestro caso particular, el frame delta será el reflejo de la aparición de un nuevo objetivo en movimiento sobre el fondo inicial .

Una vez detectamos movimiento, procedemos a dibujar un rectángulo en el que enmarcamos el objetivo detectado (el sistema recorre todos los objetivos posibles y enmarca solo el mayor). También dibujamos su centro, cuyas coordenadas nos servirán para mover los motores en sus ejes correspondientes.

A continuación, se muestran capturas de las tres ventanas descritas anteriormente:

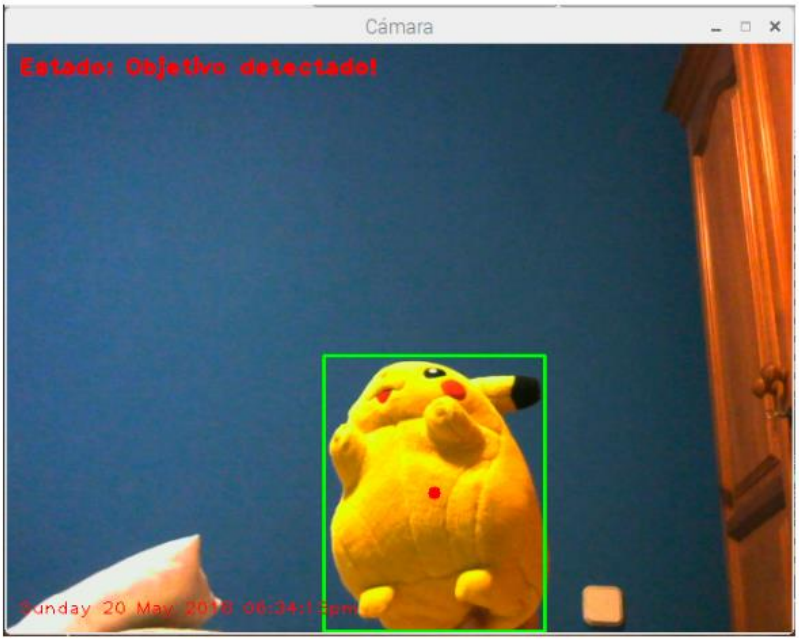


Figura 4.2 – Vista de la cámara, con objetivo detectado

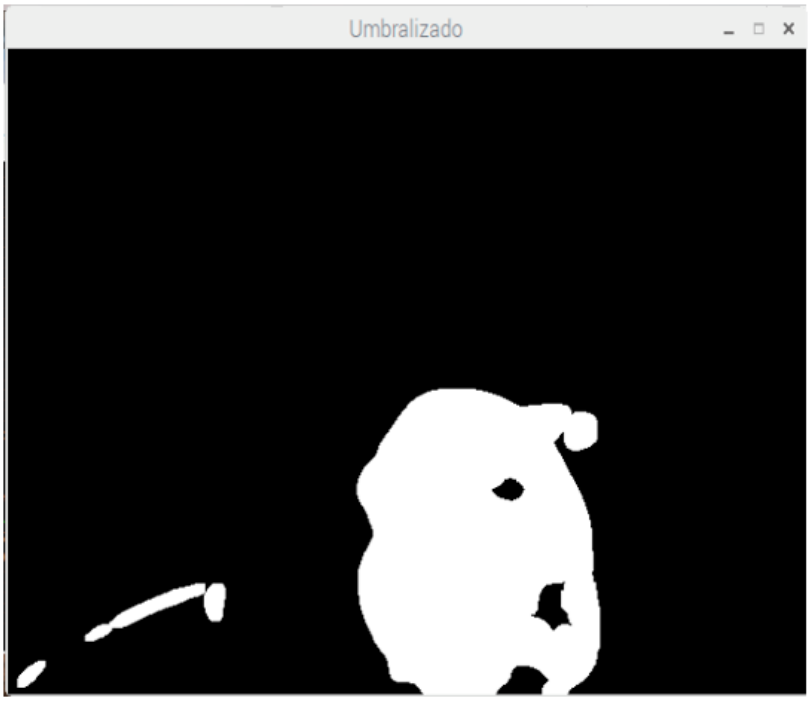


Figura 4.3 – Vista del umbralizado de la imagen

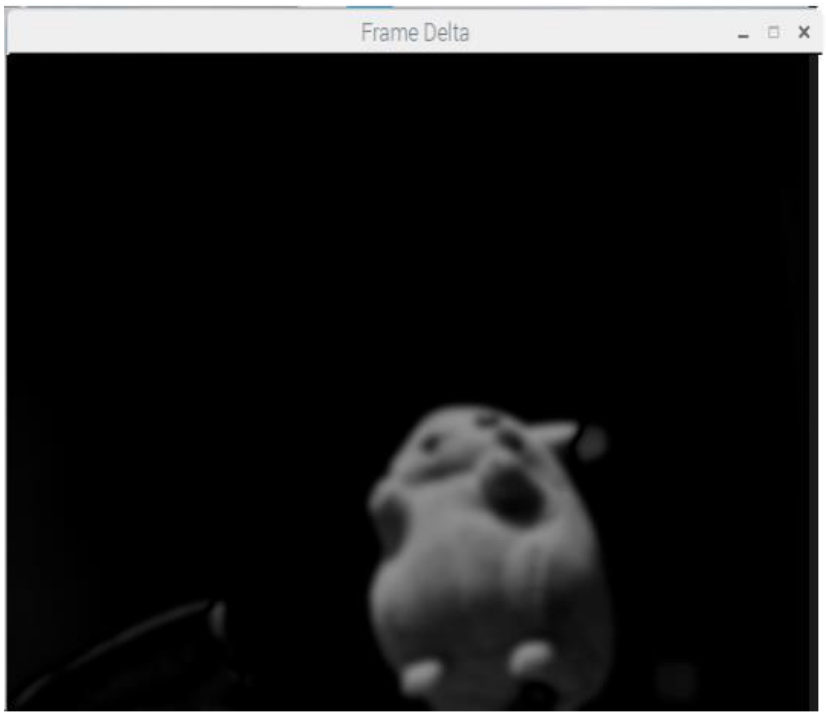


Figura 4.3 – Vista del frame Delta

5. Recursos a utilizar en el TFG

En este apartado se enumerarán todos los componentes necesarios para la realización de este proyecto, incluyendo elementos de software, de hardware, y otro tipo de componentes, incluyendo el análisis previo realizado y la motivación de las elecciones finales.

5.1 – Elección de componentes

Debido al grado que he realizado y a la especialidad escogida, la educación recibida se ha centrado en sistemas de tipo software, utilizando sólo hardware en dos asignaturas de la carrera "Periféricos e Interfaces" y "Fundamentos Físicos de la Informática". Por ello, para diseñar el sistema fue necesario realizar una investigación previa.

5.1.1 – Componente informático

Es la base de todo el proyecto. Era necesario que fuera un sistema económico, de pequeño tamaño, estable y con soporte para las librerías de tratamiento de imagen, así como lo suficientemente potente para utilizar OpenCV de forma fluida.

Por estos requisitos, estaba claro que la placa escogida debía ser una de tipo SBC (Single Board Computer, computador de placa única), ya que son de coste reducido, consumen poca energía (lo que les hace ideal para utilizarse con baterías) y son pequeñas y resistentes.

Es cierto que el uso de un ordenador portátil nos daría una mayor potencia de procesamiento, pero no podemos olvidar que el objetivo final de la torreta es usarse en una partida de Airsoft en la que la autonomía de un equipo estándar es inferior a la de una placa de este tipo, y sería bastante probable que alguno de los componentes del mismo acabara dañado, ya que la reacción natural de un jugador es responder a los impactos, lo que haría que el equipo estuviera en un peligro constante, especialmente su pantalla.

En el mercado existen diferentes marcas de SBC, siendo la más conocida la Raspberry Pi 3B. De un precio similar podemos encontrar otras alternativas, como la Orange Pi Plus 2E o, en un rango mayor de precios, la Odroid XU4, en apariencia todas válidas para el proyecto.

En este caso, aspectos como la calidad de la salida de vídeo o la velocidad de su puerto ethernet no son especialmente relevantes, por lo que se realizó una comparativa según procesador, memoria RAM y configuración de los GPIO:

Placa	Procesador	Memoria RAM	GPIO
Raspberry Pi 3B	ARM Cortex-A53, QuadCore de 64 bits	1 GB LPDDR2	40 pines
Orange Pi Plus 2E	Allwiner, ARM Cortex A7 QuadCore de 64 bits	2 GB LPDDR3	40 pines
Odroid XU4	Samsung Exynos5422 Cortex A15 + Cortex.A7 (OctaCore)	2 GB LPDDR3	30 pines + 12 pines

Tabla 5.1 – Comparativa entre las diferentes SBC candidatas para el proyecto

La Raspberry Pi 3B es la más conocida de las tres con diferencia, especialmente para aquella gente que no suele comprar por internet. Una muestra de ello es que la Raspberry puede comprarse en numerosas grandes superficies dedicadas a la electrónica, y la Orange Pi y la Odroid han de ser adquiridas por internet.

Por este motivo, se puede afirmar que la Raspberry Pi es el estándar en el mercado, lo que provoca que la mayoría de las placas de expansión que se conectan a los GPIO están hechas pensando en ella, lo cual es un aspecto clave a la hora de decidir.

Por ejemplo, si observamos las características de la Odroid XU4, es cierto que es la que mayor potencia tiene de las tres que participan en la comparativa, pero tiene una distribución diferente en los pines, lo que hará que de fábrica no funcione con las placas anteriormente mencionadas, Este hecho, unido a su elevado precio, hizo que la Odroid fuera descartada, ya que su mayor rendimiento no compensa las carencias que tiene.

Según el fabricante de la Orange Pi Plus 2E, este modelo es compatible con todas las placas diseñadas para la Raspberry Pi, pero, en la fecha en la que se redactó este documento, no fue posible encontrar algún sitio web en el que un usuario real contara su experiencia utilizando una placa controladora de Raspberry Pi.

Este aspecto, unido a la imposibilidad de adquirirla en una tienda física (y al hecho de que ya contaba con una Raspberry Pi 3B de mi propiedad), hizo que se descartara la Orange Pi en favor de la Raspberry.

Figura 5.1 – Raspberry Pi 4 ModelB

Al margen de que ya se poseía una unidad, también fue un aspecto decisivo la experiencia personal previa, ya que he sido usuario de Raspberry desde que salió a la venta su segundo modelo, por lo que estoy familiarizado con Raspbian, el sistema operativo utilizado habitualmente en ella.

Al ya contar con la placa, el diseño del sistema se basó en todo momento en su uso. En las etapas de investigación previas al diseño, se buscaron proyectos similares que hubieran sido realizados utilizándola, y se comprobó la existencia de dichos proyectos, lo cual nos demostraba de antemano que era posible montar el sistema diseñado de forma exitosa.

5.1.2 – Componentes software

En el apartado anterior definimos que nuestro componente informático iba a ser una Raspberry Pi, por lo que debíamos encontrar un software que fuera capaz de ejecutarse en ella, por lo que se realizó una búsqueda de alternativas existentes de visión por computador que fueran compatibles con ella, destacando las siguientes: OpenCV, Pillow y SimpleCV.

Librería Pillow [22]:

Con esta librería podemos añadir procesamiento de imágenes a nuestro intérprete de Python. Esta librería open source nos ofrece funciones muy básicas, ya que está pensada para trabajar con imágenes estáticas y no con streaming de vídeo, con lo cual no nos sirve para el objetivo de este proyecto, por lo que fue descartada.

SimpleCV [23]:

Si hablamos de SimpleCV, nos encontramos ante una versión simplificada de OpenCV. Tal y como se define en su página web, SimpleCV es "la visión por computador hecha fácil". Es cierto que parece la opción ideal para un principiante, pero se consideró más útil (teniendo en cuenta que era necesario aprender cualquiera de las opciones desde cero), aprender directamente la versión completa (OpenCV).

OpenCV [24]:

OpenCV es una librería de visión por computador y procesamiento de imagen. Liberada bajo una licencia BSD, es gratuita para uso académico y comercial. El código de OpenCV está escrito en C/C++, y permite su uso (en la Raspberry Pi) desde C++ y Python.

OpenCV supera ampliamente las posibilidades de Pillow en cuanto a vídeo, y, evidentemente, es más potente que SimpleCV, ya que este último es una versión simplificada de OpenCV, por lo que OpenCV fue la librería escogida finalmente.

Teniendo clara la librería de visión por computador a utilizar, debíamos elegir en qué lenguaje íbamos a utilizarla: C++ o Python. Como no se conocían ninguno de los dos lenguajes, se basó su elección en la comunidad existente en foros y las posibles aplicaciones futuras en proyectos personales y profesionales.

A nivel personal, campos de la informática como el Big Data y la Inteligencia Artificial me parecen de gran interés, y es habitual que el software utilizado en estos campos use Python. También otros proyectos interesantes, como la versión de Google Assistant para la Raspberry Pi, utilizan este lenguaje.

Por este motivo, se descartó C++ y se escogió Python como lenguaje a utilizar.

5.1.3 – Componentes hardware

Igual que en el apartado anterior, necesitamos que el hardware sea compatible con la Raspberry Pi 3, por lo que fue un requisito necesario en todos los componentes utilizados.

Sabemos que el sistema a desarrollar va a moverse en dos ejes (vertical y horizontal), por lo que necesitaremos dos motores. En este tipo de proyectos de electrónica se utilizan habitualmente tres tipos de motores: de corriente continua, servo motores y motores paso a paso.

Motor DC o de corriente continua [25]:

Un motor de corriente continua se compone principalmente de dos partes: el estátor, que da soporte mecánico al aparato y contiene los polos de la máquina, y el rotor, generalmente de forma cilíndrica, y alimentado con corriente directa a través de delgas, que están en contacto alternante con escobillas fijas. Estos motores tienen un precio muy asequible, pero no nos permiten realizar movimientos a posiciones concretas, salvo que se utilicen sensores de posición angular, lo cual complicaría su uso en este proyecto

Figura 5.2 – Motor DC

Servomotor [26]:

Un servomotor es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, así como de mantenerse estable en dicha posición. Es más complejo conseguir movimientos precisos con este tipo de motores si lo comparamos con los paso a paso, por lo que fueron descartados

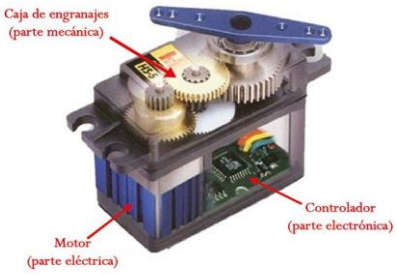


Figura 5.3 - Servomotor

Motor paso a paso [27]:

Un motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados dependiendo de sus entradas de control. Son utilizados habitualmente en las impresoras 3D. Estos motores son precisos, tienen una buena relación entre el tamaño y el torque que son capaces de realizar, por lo que son ideales para este proyecto, así que eran el tipo de motor ideal para nuestro proyecto.

Inicialmente se escogió el modelo 17HS2048 para el sistema, y la placa Adafruit Motor HAT como controladora, ya que la librería proporcionada nos facilitaba en gran medida el desarrollo del proyecto.

No obstante, después de haberlo comprado, soldado y probado, se demostró que dichos motores no eran capaces de hacer girar la estructura utilizada como base, por lo que fueron descartados.

El siguiente modelo a la venta que encontramos fue el 17HS4401, un motor de tipo Nema17, paso a paso, y con 200 pasos por vuelta (como el anterior), aunque en esta ocasión el 17HS4401 es capaz de cargar con 4.28 kg/cm, superando al 17HS2048, que mueve 1.22 kg/cm.

Figura 5.4 – Motor paso a paso Nema 17HS2048

Controlador:

Ahora que ya hemos escogido los motores, debemos buscar una controladora que nos permita manejarlos desde la Raspberry, ya que no es posible hacerlo directamente desde los GPIO. Necesitamos poder controlar dos motores paso a paso a la vez (uno por cada eje), poder controlarlos desde Python y poder conectarles una fuente de alimentación externa, ya que nuestra Raspberry no es capaz de proporcionarle la corriente que necesita.

Se encuentran con facilidad placas controladoras para un solo motor, por lo que la solución inicial propuesta fue el Motor HAT de Adafruit [28].

Esta placa, fabricada una de las empresas más conocidas en el mundo de la Raspberry Pi (Adafruit), nos permite controlar hasta cuatro motores de corriente continua o dos motores paso a paso a la vez, admitiendo una alimentación externa y con una librería de Python disponible, por lo que parecía ideal para nuestro proyecto. Sin embargo, no era capaz de mover los motores escogidos, por lo que tuvo que ser descartada.

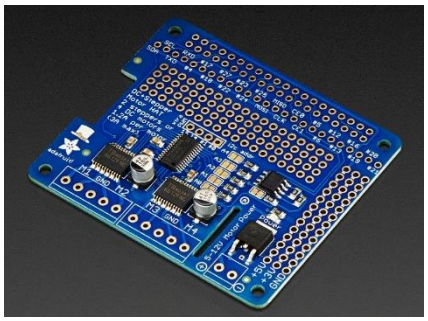


Figura 5.5 - Adafruit Motor Hat (Antes de soldar)

Una vez descartado el Motor HAT, debíamos escoger un modelo de driver que nos permitiera controlar un motor paso a paso, preferiblemente que pudiera ser adquirido en una tienda física en Gran Canaria, y capaz de aguantar los 1.7A que consume nuestro motor. El modelo que encontramos que cumplía todas estas características fue el L298N V2.

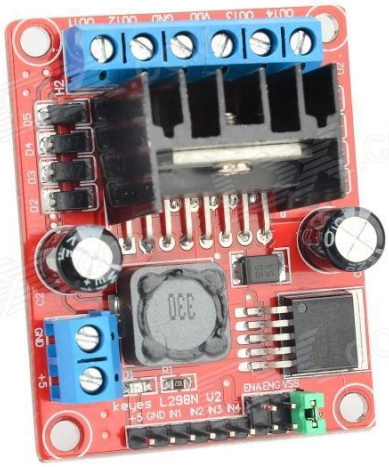


Figura 5.6 –Driver Keyes L298N V2 [FOTO]

Este tipo de controladores son un estándar en el mundo de Arduino, aunque son compatibles con la Raspberry, ya que se controlan desde los GPIO. No fue posible encontrar una librería en Python que nos permitiera utilizar estos motores correctamente, por lo que fue necesario desarrollar una propia (`L298N/stepper.py` en el repositorio).

Una particularidad de los motores paso a paso es que la excitación de las bobinas cambia según el paso que deban dar, es decir, el paso 1 y el paso 2 utilizarán señales completamente diferentes, hasta completar un ciclo de cuatro pasos, que se repiten indefinidamente. Para averiguar esta secuencia fue muy útil la librería “Stepper.cpp”, disponible como programa ejemplo en el IDE de Arduino, que nos sirvió para verificar la correcta conexión de los pines del motor, así como para conocer la secuencia correcta.

Una vez desarrollada la librería se procedió a probar el sistema completo. Los L298N no son controladores pensados específicamente para motores paso a paso, por lo que no permiten uno de los aspectos más interesantes de dichos motores: el microstepping.

Cuando hablamos de un motor paso a paso, se conoce como microstepping a la capacidad del driver de dividir un paso completo del motor en varias partes. Esto se traduce en una mayor precisión a la hora de mover los motores.

En principio no era necesario el uso de esta característica en el sistema desarrollado, pero al probar el L298N comprobamos que mover el motor un solo paso cada vez era algo demasiado brusco, ya que producía vibración e inercia, lo que lleva a pérdidas de precisión en el apuntado del láser.

Hay ocasiones en las que sólo necesitamos un paso, como cuando un objetivo se mueve una distancia corta, o a la hora de calibrar el sistema, por lo que mover siempre más de un paso no era una opción viable. Por ello, se decidió cambiar el L298N por un driver con soporte para microstepping, para así reducir la inercia en movimientos pequeños.

Después de explorar las alternativas existentes, se decidió que el modelo ideal estaba entre dos opciones: el A4988 y el DRV8825. Ambos modelos son muy similares, con conexiones prácticamente idénticas, ya que solo varían en la configuración de los pines que definen el tipo de micropaso y en la temperatura que son capaces de soportar, siendo el DRV el que más corriente soporta.

En esta ocasión volvió a ser un problema el hecho de vivir en Canarias, ya que nos vemos limitados de nuevo por el stock de las tiendas locales. El DRV8825 fue imposible de encontrar, por lo que, a pesar de ser este la mejor opción, se descartó en favor del A4988, que sí estaba disponible en tiendas de electrónica.

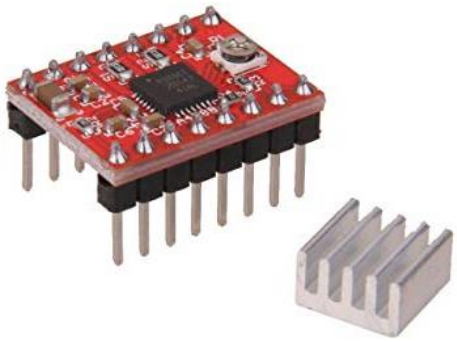


Figura 5.7 – Driver A4988

Cámara:

Necesitábamos una cámara compatible de forma nativa con Linux y OpenCV, que fuera económica y que tuviera una calidad de imagen. La primera elección fue la Playstation Eye, cámara diseñada para la videoconsola Playstation 3, utilizada en algunos juegos que incorporan detección de movimiento y color. Fue descartada después de realizar la prueba de conexión, ya que su resolución era bastante escasa y dificultaba la detección de movimiento. En el mercado actual encontramos una buena opción, la webcam Logitech C270, que cumplía todos los requisitos necesarios, y cuenta con una resolución de 720p, por lo que fue escogida.

Figura 5.8 - Webcam Logitech C270.

5.1.4 – Otros componentes

Elemento de apuntado:

La opción más ligera y sencilla fue el utilizar un puntero láser, similar a los usados en astronomía, aunque de mucha menos potencia. Se procedió a retirarle el depósito de las pilas, colocándolas en un porta pilas, para así poder encenderlo y apagarlo con un interruptor, en vez de tener que mantener pulsado manualmente el botón.

Un puntero láser es un elemento muy ligero, y es fácil de ver el lugar a donde está apuntando, por lo que es ideal para el proyecto.

Soporte:

Era necesario escoger una estructura sobre la que montar todos los componentes necesarios. Debía de ser un elemento portátil, resistente, ligero y con una cierta altura respecto al suelo. La primera intención fue la adquisición de un trípode, que fue descartada debido a su precio y al hecho de que requería quitar la rótula de la cámara de la parte superior y poner una tabla o algún tipo de plataforma para montar el sistema, por lo que la estructura final que nos quedaba es algo muy similar a un taburete, pero con mucho más trabajo y coste.

Por estos motivos, se buscó en el catálogo de Ikea un taburete que cumpliera los requisitos necesarios, siendo finalmente escogido el taburete Marius, fabricado en plástico y metal, lo que lo hace resistente y muy económico.

Figura 5.9 – Taburete Marius

Fuente de alimentación (Raspberry Pi)

Para alimentar a la Raspberry Pi se utilizó su fuente de alimentación oficial, que funciona a 5 voltios, y proporciona una intensidad de 2.5 amperios.

Figura 5.10 – Fuente de alimentación oficial para Raspberry Pi

Fuente de alimentación (Motores)

Para proporcionarle la energía necesaria a los motores se utilizaron dos fuentes de alimentación universales, ambas regulables entre 4.5 y 12V, y capaces de proporcionar 2.5A de salida para nuestros motores.

Figura 5.11 – Fuente de alimentación universal

Carpintería:

Estaba claro que necesitaríamos una serie de piezas para montar la parte superior de la estructura. Por coste y peso, se escogió la madera como material.

La estructura de este tipo de sistemas habitualmente se compone de un pie, un motor y un soporte para el mecanismo de disparo.

En nuestro caso, el pie es el taburete Marius antes mencionado, aunque necesitábamos aún la base que debe mover el motor y el soporte del mecanismo de disparo. Debía de ser una madera ligera, aunque resistente, por lo que se eligió el DM.

El DM es un aglomerado de fibras de madera, por lo que es un material ligero y fino, aunque es lo suficientemente grueso como para poder usar tornillos en la tabla sin que la atraviesen.

Explorando las alternativas existentes, nos encontramos con un proyecto de torreta diseñada por Hacker Shack [31], cuyo diseño de estructura sirvió como base para la realizada en este proyecto, eso sí, después de una serie de modificaciones. También son de su autoría las piezas 3D originales que fueron adaptadas para cubrir nuestras necesidades.

La base está formada por dos piezas, dos discos de 25 centímetros de diámetro, uno de 21 milímetros de grosor y otro de 6 milímetros. Ambos discos tienen una rebaja en el centro para facilitar que se fije el rodamiento en su interior.

En el soporte, tenemos una pieza con forma trapezoidal de 22 centímetros de base y 30 centímetros de altura. En esta pieza se realizó un agujero a 4 centímetros de la parte superior para fijar el láser (Los esquemas de las piezas se encuentran en el Anexo 1).

Todas estas piezas fueron encargadas a una carpintería especializada, ya que no se contaba con los materiales necesarios para hacerlas de forma casera.

Rodamiento:

Para facilitar el giro del sistema se introdujo un rodamiento axial ranurado de bolas entre los discos de madera que conforman la base. Las partes fijas fueron pegadas a la parte superior e inferior de la base, para así permitir el giro libre de la parte central (el anillo de bolas).

Figura 5.12 - Rodamiento axial ranurado de bolas

Cableado y tornillería:

Para realizar las conexiones necesarias se utilizó el cableado que traían de fábrica los motores paso a paso, así como un cable unipolar estándar para conectar los cabezales de las alimentaciones.

En cuanto a tornillería, fue necesario adquirir tornillos, tuercas y arandelas varias para montar la estructura y fijarla al asiento de la silla, así como para alargar el eje del motor para que fuera capaz de atravesar ambas capas de la base.

Piezas impresas en 3D

Estas piezas son necesarias para lograr la rotación de la estructura de la torreta en la base, así como la rotación en el eje Y del mecanismo de disparo.

La pieza original fue descargada de la web SketchFab[21], tratada y editada con el software de impresión y diseño 3D Cura Ultimaker, para ser finalmente impresa con la Ultimaker 2 Go.

Inicialmente se imprimieron dos unidades iguales, pero se comprobó que la pieza colocada entre las piezas de la base producía cierto rozamiento, así que se añadió el rodamiento entre dichas piezas, por lo que la pieza 3D inicial fue descartada.

La otra pieza 3D se utilizó en el soporte para fijar el láser al eje del motor, y así permitir el movimiento en el eje Y.

Al alargar el eje con una barra metálica, perdimos la parte plana del eje, con lo cual fue necesario modificar el diseño e imprimir una pieza más larga, con un tornillo lateral para aumentar la sujeción de la estructura, y así reducir las vibraciones del conjunto.

Esta pieza fue inicialmente impresa en 3D, pero al ser plástico se desgastaba con facilidad. Tuve la oportunidad de fabricarla en aluminio mediante un torno CNC, así que fue aprovechada, aumentando así la resistencia del sistema.

Piezas utilizadas

Figura 5.13 (A) – Pieza para adaptar un motor Nema17 a una superficie plana

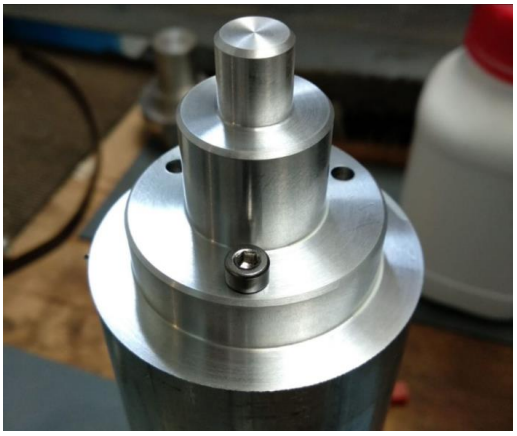


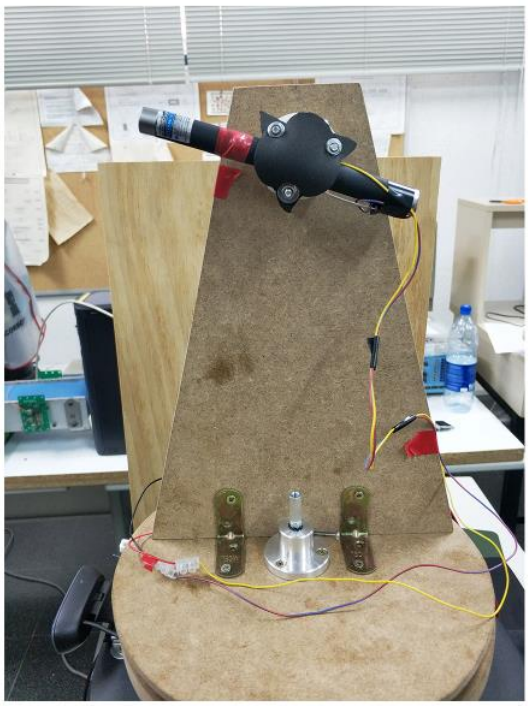
Figura 5.13 (B) – Modificación en aluminio de la figura 5.13 (A)

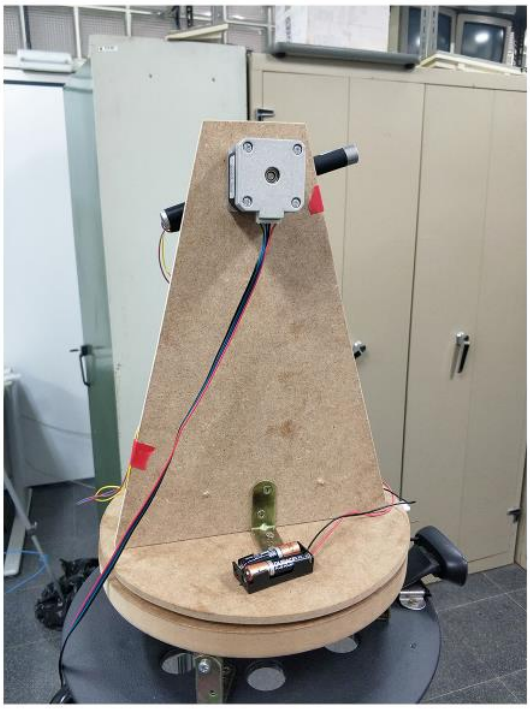
Impresora 3D utilizada:

Figura 5.13 (C) – Impresora Ultimaker 2 Go

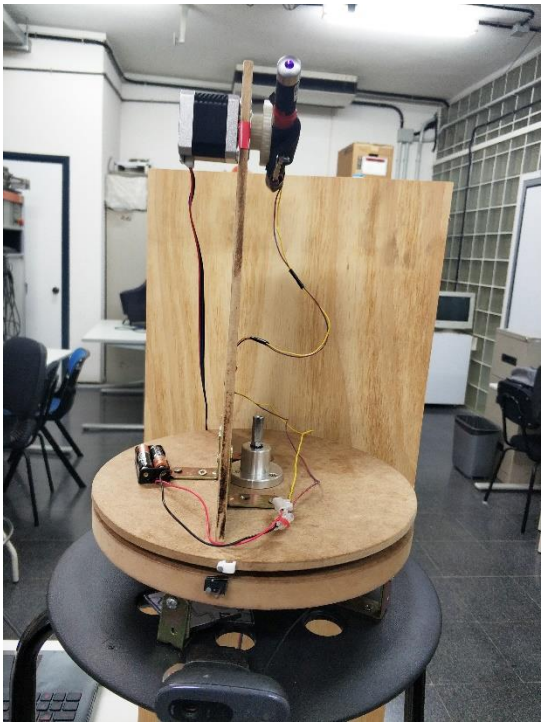
5.1.5 – Aspecto final

Una vez montado, el aspecto final del proyecto es el siguiente:





Vistas laterales del sistema



Vista frontal

5.2 – Software utilizado

A continuación, se desglosarán los diferentes programas, utilidades y librerías utilizadas durante la elaboración del TFG:

- **JetBrains Pycharm 2017.3.4:**
  Pycharm es un IDE para Python, desarrollado por la empresa JetBrains. Ha sido utilizado con la licencia de estudiante proporcionada como miembro de la ULPGC.

- **Synergy:**
  Synergy es un software Open Source que permite compartir un teclado y un ratón entre varios ordenadores, sin la necesidad de ningún hardware adicional. Se empleó para compartir teclado y ratón entre el ordenador y la Raspberry Pi, ya que el teclado utilizado no era reconocido de forma nativa.

- **Raspbian:**
  Raspbian es una distribución del sistema operativo GNU/Linux (y por lo tanto libre), basado en Debian Jessie (Debian 8.0) para la placa computadora (SBC) Raspberry Pi, orientado a la enseñanza de informática. El lanzamiento inicial fue en junio de 2012.

- **SFTP Net Drive 2017:**
  Utilidad software que permite mapear discos remotos como unidades locales de Windows vía SFTP, lo que permite utilizar Pycharm para desarrollar desde un ordenador convencional sobre los ficheros alojados en la Raspberry Pi

- **Python 2.7.13:**
  Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

- **OpenCV 3.1.0:**

  OpenCV es una biblioteca libre de visión artificial desarrollada originalmente por Intel. Publicada bajo licencia BSD, está permitido que sea usada libremente para propósitos comerciales y de investigación. En este proyecto se utiliza desde Python.

- **Imutils:**
  Librería para Python que nos facilita el procesado básico de imágenes, asi como el mostrar imágenes de Matplotlib utilizando OpenCV.

- **Numpy:**
  Extensión de Python que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel, diseñadas para operar con vectores o matrices. Es open source.
- **Fritzing:**

  Fritzing es un programa libre de automatización de diseño electrónico que busca ayudar a diseñadores y artistas para que puedan pasar de prototipos a productos finales. Se utilizo para realizar los esquemas de las conexiones de los componentes.

6. Diseño e implementación

En este apartado se desglosarán las diferentes fases de diseño y desarrollo que se han llevado a cabo durante la realización de este proyecto.

6.1 – Diseño

A nivel de organización, el código está desarrollado en funciones implementadas, zona de inicialización de variables y bucle infinito, que se encarga de mostrar la cámara y buscar objetivos hasta que se le indique lo contrario. Esto está desarrollado en el siguiente apartado (Implementación).

En cuanto a la estructura del software desarrollado ha intentado seguir siempre la filosofía Clean Code. Es una práctica común en un desarrollador escribir un código, no volver a utilizarlo en meses, necesitar un módulo de dicho código y, al volver atrás, ser incapaz de comprender el código escrito por el mismo.

Debido a esto, para ahorrarnos tiempo y quebraderos de cabeza a nosotros mismos (y a otros desarrolladores que puedan utilizar nuestro código), se consideró una buena idea seguir las directrices del código limpio, ya que el seguir sus estándares y buenas prácticas puede ayudarnos en gran medida en el futuro, y ser la diferencia entre tener que implementar la misma función dos veces o poder reutilizar funciones de nuestro código.

En este caso cobra una mayor importancia, ya que se pretende dejar el repositorio abierto para cualquiera a quien pueda interesarle el proyecto, por lo que un código fácil de comprender nos facilita el trabajo a todos.

Principios del Clean Code [32]:

El Clean Code nos da una serie de indicaciones y consejos para que consigamos desarrollar “código limpio”, siendo algunos de los aspectos más relevantes los siguientes:

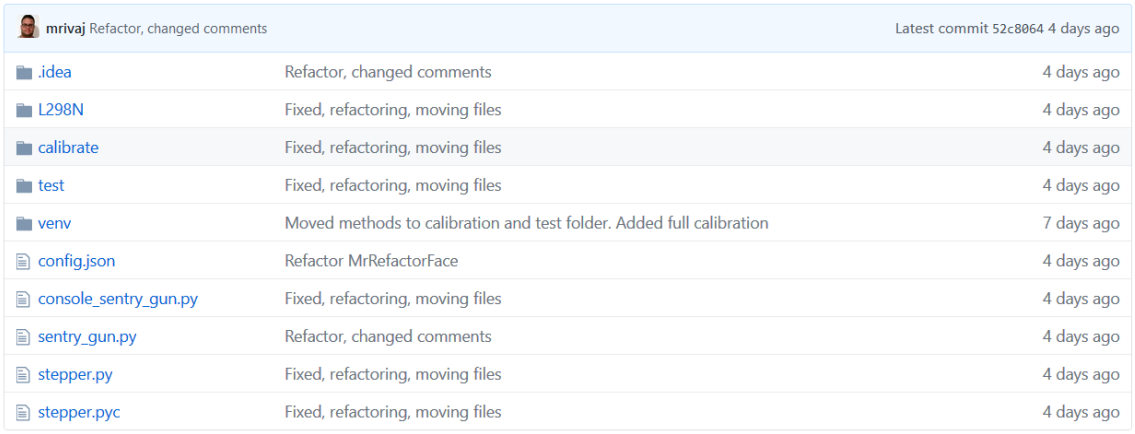
1. **El principio de la menor sorpresa:**
   - Las funciones o clases deben hacer lo que (razonablemente) se espera de ellas. Es decir, los nombres de variables, clases y funciones deben ser acordes a su comportamiento.
2. **Principio de responsabilidad única**
   - Cada clase debe tener una y sólo una razón para cambiar. Es decir, cada clase tendrá solo una responsabilidad.
3. **Principio Open-Closed:**
   - Una clase debe estar abierta a extensiones, pero cerrada a modificaciones. Esto quiere decir que, ante nuevos requisitos, la clase debe ser extendida y no modificada.
4. **Principio DRY (Don’t Repeat Yourself):**
   - El código duplicado es un gran problema, ya que si falla deberemos cambiarlo en múltiples sitios, y olvidar uno nos puede llevar a errores. Si vamos a usar varias veces una porción del código, es preferible extraerlo a una función.

En cuanto a la estructura del código, se fueron desarrollando las diferentes funciones a medida que avanzaba el montaje y se iba necesitando, para así cumplir los requisitos definidos en el apartado 4. Al principio se buscaba la funcionalidad, así que era una estructura poco legible, con bloques muy grandes de código (el tipo de desarrollo conocido como "código monolítico"), pero esta estructura es muy difícil de mantener, por lo que se modularizó y refactorizó el código, buscando cumplir así los principios del código limpio..

Si hablamos de la organización del código, el aspecto final del repositorio fue el siguiente:

Directorio raíz:

A este nivel encontramos la carpeta de tests, la carpeta de calibración de los motores, el archivo de configuración del programa (`config.json`), la versión de consola del programa principal y el propio programa, que se ejecutará para poner en marcha el sistema (`sentry_gun.py`).

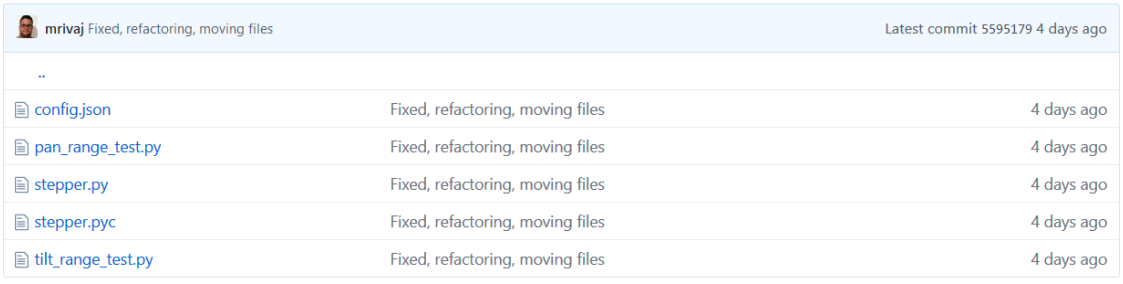


mrivaj Refactor, changed comments		Latest commit 52c8064 4 days ago
.idea	Refactor, changed comments	4 days ago
L298N	Fixed, refactoring, moving files	4 days ago
calibrate	Fixed, refactoring, moving files	4 days ago
test	Fixed, refactoring, moving files	4 days ago
venv	Moved methods to calibration and test folder. Added full calibration	7 days ago
config.json	Refactor MrRefactorFace	4 days ago
console_sentry_gun.py	Fixed, refactoring, moving files	4 days ago
sentry_gun.py	Refactor, changed comments	4 days ago
stepper.py	Fixed, refactoring, moving files	4 days ago
stepper.pyc	Fixed, refactoring, moving files	4 days ago

Figura 6.1 – Estructura de la raíz del repositorio en Github

Carpeta tests:

Aquí encontramos la carpeta con los test elaborados para comprobar el correcto funcionamiento de los motores.

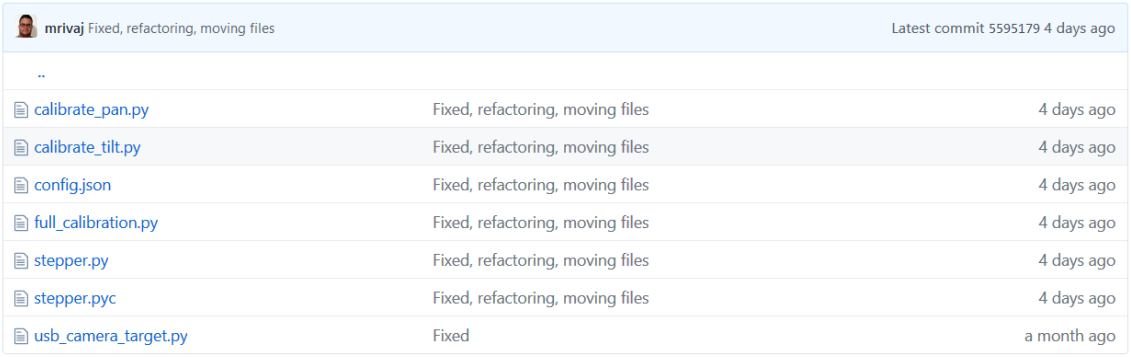


mrivaj Fixed, refactoring, moving files		Latest commit 5595179 4 days ago
..		
config.json	Fixed, refactoring, moving files	4 days ago
pan_range_test.py	Fixed, refactoring, moving files	4 days ago
stepper.py	Fixed, refactoring, moving files	4 days ago
stepper.pyc	Fixed, refactoring, moving files	4 days ago
tilt_range_test.py	Fixed, refactoring, moving files	4 days ago

Figura 6.2 – Carpeta "tests" del repositorio

Carpeta calibrate:

En esta carpeta se encuentros los ficheros necesarios para calibrar el sistema, así como el archivo que muestra la cámara con el centro marcado para facilitar su calibración .

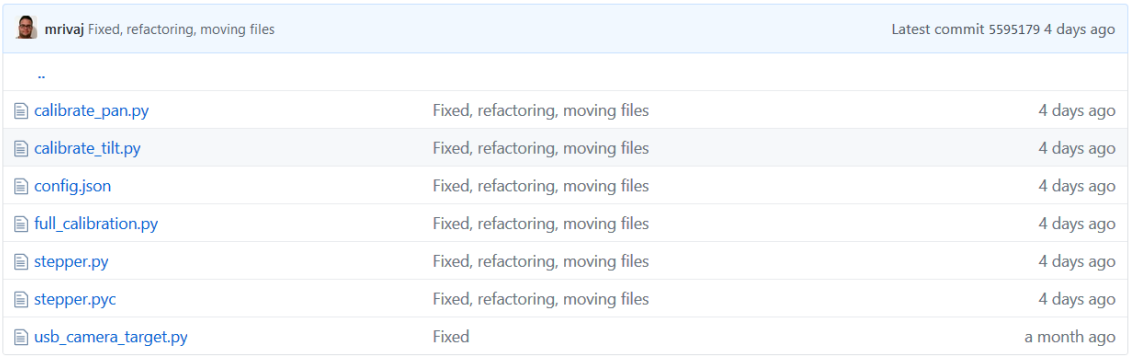


mrivaj Fixed, refactoring, moving files		Latest commit 5595179 4 days ago
..		
calibrate_pan.py	Fixed, refactoring, moving files	4 days ago
calibrate_tilt.py	Fixed, refactoring, moving files	4 days ago
config.json	Fixed, refactoring, moving files	4 days ago
full_calibration.py	Fixed, refactoring, moving files	4 days ago
stepper.py	Fixed, refactoring, moving files	4 days ago
stepper.pyc	Fixed, refactoring, moving files	4 days ago
usb_camera_target.py	Fixed	a month ago

Figura 6.3 – Carpeta "calibrate"

Carpeta L298N:

En esta carpeta se encuentran los ficheros necesarios para ejecutar el programa utilizando el driver L298N. Es funcional, pero como se comentó anteriormente, provoca demasiadas vibraciones al carecer de microstepping, por lo que no se ha utilizado finalmente. No obstante, podría ser de utilidad para cualquier persona que quisiera utilizar dicho driver, así que se ha decidido incorporarlo.



mrivaj Fixed, refactoring, moving files		Latest commit 5595179 4 days ago
..		
calibrate_pan.py	Fixed, refactoring, moving files	4 days ago
calibrate_tilt.py	Fixed, refactoring, moving files	4 days ago
config.json	Fixed, refactoring, moving files	4 days ago
full_calibration.py	Fixed, refactoring, moving files	4 days ago
stepper.py	Fixed, refactoring, moving files	4 days ago
stepper.pyc	Fixed, refactoring, moving files	4 days ago
usb_camera_target.py	Fixed	a month ago

Figura 6.4 – Carpeta "L298N"

6.2 – Implementación

En este apartado se definirá la metodología utilizada, y se explicarán las diferentes partes del código desarrollado.

6.2.1 - Metodología

Como se definió en el apartado 4 de este documento, para desarrollar la parte software de este proyecto se utilizó una metodología basada en el modelo iterativo incremental, consistente en las siguientes fases:

- **Análisis:**
  Fase de estudio previa. En ella se define qué es lo que se quiere conseguir y se busca información sobre los posibles métodos para conseguirlo.
- **Diseño:**
  Se define la estructura del programa a desarrollar.
- **Codificación:**
  Se escribe el código necesario para lograr el objetivo de esta fase.
- **Pruebas:**
  Se comprueba que se cumple el objetivo de la iteración actual.

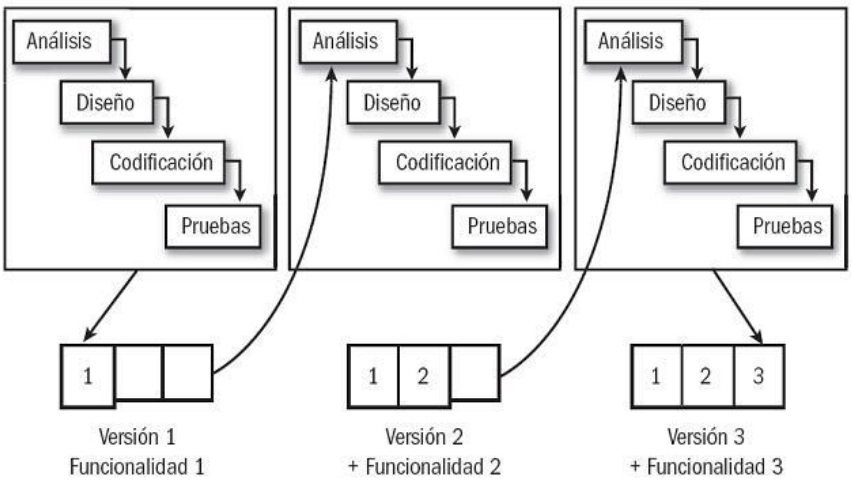


Figura 6.5 – Fases del modelo iterativo.

En este desarrollo, se añadió una fase de refactorización al final de cada iteración, para así mejorar la legibilidad del código, e intentar cumplir los valores que defiende la filosofía Clean Code, para así lograr que el código escrito sea de mejor calidad.

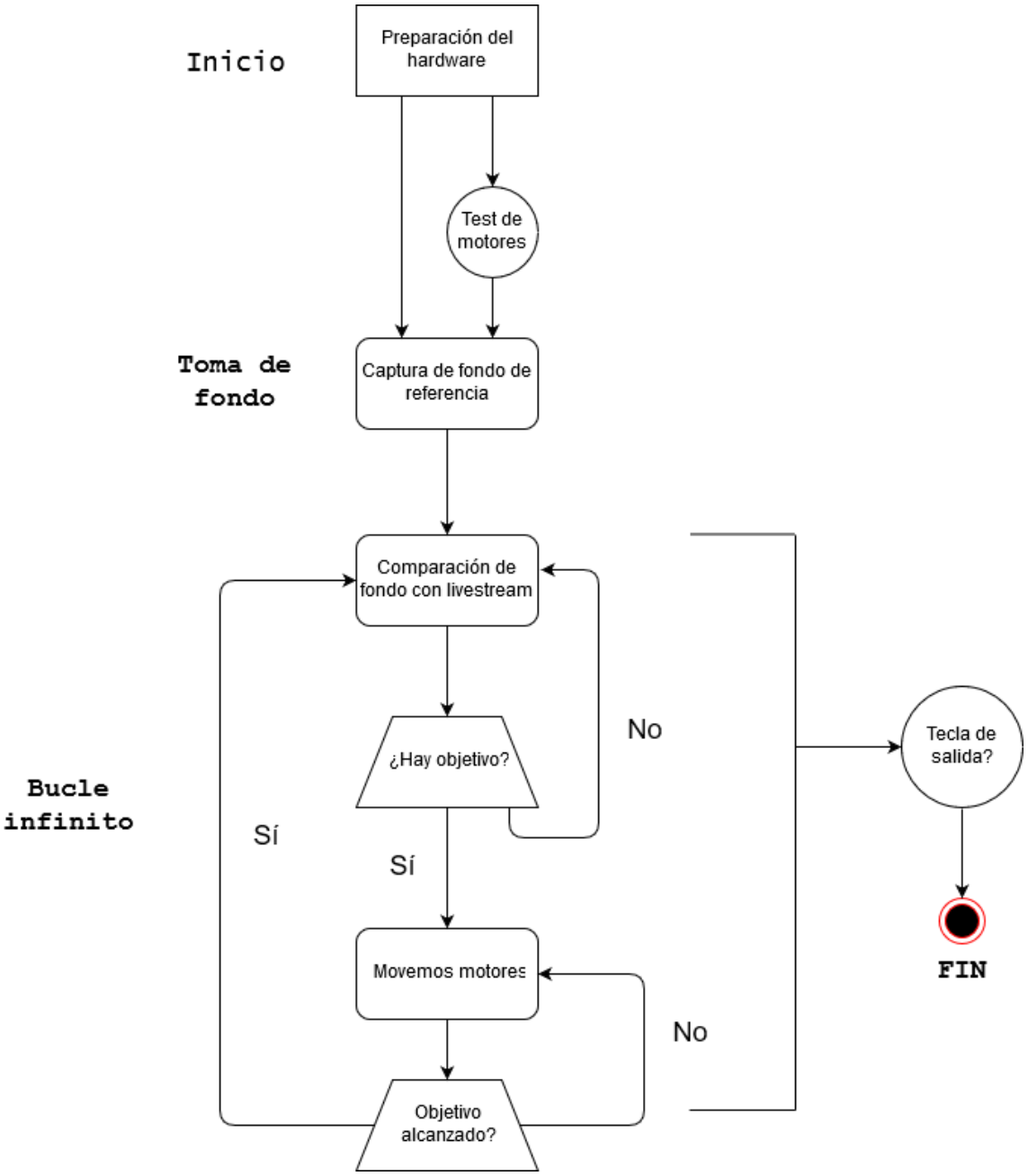
Una explicación más detallada de las iteraciones que se siguieron a lo largo del desarrollo de este software puede encontrarse en el apartado 4 de este documento, en el que se desglosa el plan de trabajo definido, o en el apartado 7, en el que se analizan los resultados obtenidos.

6.2.2 – Código desarrollado

En este apartado se explicarán y analizarán los módulos más relevantes del código implementado. Todas las capturas proceden del código desarrollado, y que está disponible adjunto y en el repositorio del proyecto.

6.2.2.1 – Diagrama de flujo del algoritmo

El diagrama de flujo del algoritmo implementado es el siguiente:



6.2.2.2 - Funciones:

1. **Load_config():**

Esta función se encarga de cargar la configuración del fichero "config.json" en variables que puedan ser utilizadas durante la ejecución del programa.

```
def load_config():
    config = json.load(open('config.json'))
    print "[INFO] Cargamos la configuración del usuario"
    global minimum_target_area, exit_key,frame_color,center_color,\
        message_target_detected, message_target_not_detected, \
        base_testing_steps, top_testing_steps, test_motors,\
        print_movement_values

    # General config
    minimum_target_area= config['GENERAL']['MINIMUM_TARGET_AREA']
    exit_key = config['GENERAL']['EXIT_KEY']
    frame_color = string_to_rgb(config['GENERAL']['TARGET_FRAME_COLOR'])
    center_color = string_to_rgb(config['GENERAL']['TARGET_CENTER_COLOR'])

    # Messages
    message_target_detected = config['MESSAGES']['TARGET_DETECTED']
    message_target_not_detected = config['MESSAGES']['TARGET_NOT_DETECTED']
    # Motor config
    base_testing_steps = config['MOTOR']['BASE_TESTING_STEPS']
    top_testing_steps = config['MOTOR']['TOP_TESTING_STEPS']

    # Debug
    test_motors = config['DEBUG']['TEST_MOTORS']
    print_movement_values = config['DEBUG']['PRINT_MOVEMENT_VALUES']
```

2. **String_to_rgb()**

El fichero "`config.json`" solo nos permite cargar string. Este método auxiliar nos ayuda a pasar una string con los valores RGB a una tupla BGR, para así definir los colores de los elementos que dibujamos con OpenCV (Centro, marcos de objetivos, fecha…).

```
def string_to_rgb(rgb_string):  # OpenCV uses BGR
    b,g,r = rgb_string.split(",")
    return (int(b),int(g),int(r))
```

3. Find_best_target():

OpenCV es capaz de detectar múltiples objetivos en movimiento. Con este método, iteramos entre todos los objetivos detectados para quedarnos con el que sea más fácil de apuntar, en este caso, el más grande de los detectados.

```
def find_best_target():
    image, borders, h = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    big_area = 5000
    best_contour = None
    for b in borders:
        area = cv2.contourArea(b)
        if area > big_area:
            big_area = area
            best_contour = b
    return best_contour
```

4. Draw_target_frame(), Draw_target_center():

Este método obtiene las dimensiones necesarias para dibujar el rectángulo que enmarca a los objetivos, y llama a la función auxiliar “draw_target_center()”, que se encarga de dibujar el centro del cuadrado.

```
def draw_target_frame(contour):
    # Calculamos y dibujamos el marco y su centro
    (x, y, w, h) = cv2.boundingRect(contour)
    cv2.rectangle(frame, (x, y), (x + w, y + h), frame_color, 2)
    draw_target_center(x, y, w, h)


def draw_target_center(x,y,w,h):
    # PARÄMETROS PARA DIBUJAR EL CÍRCULO
    # cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]])

    square_center_x = x + w / 2
    square_center_y = y + h / 2
    cv2.circle(frame, (square_center_x, square_center_y), 5, center_color, -1)
    calculate_moves(square_center_x,square_center_y)
```

5. Print_info_on_video():

Este método se encarga de mostrar la información necesaria sobre el streaming de video .

```
# Mostramos por pantalla el estado y la fecha
def print_info_on_video():
    cv2.putText(frame, "Estado: {}".format(text), (10, 25),
        message_font, 1.25, center_color, 2)
    cv2.putText(frame, datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"),
        (10, frame.shape[0] - 15), message_font, 1, center_color, 1)
```

6. **Motor_test():**

Este método (que se ejecuta o no según que opción se indique en el fichero de configuración) realiza pequeños movimientos con los motores de la base y el soporte, para así descartar problemas de conexión o de alimentación.

```
def motor_test():
    print "  [TEST] Probando el motor de la base"
    pan_motor.move_forward(base_testing_steps)
    time.sleep(0.5)
    pan_motor.move_backwards(base_testing_steps)
    time.sleep(0.5)
    print "[DONE] \n [TEST] Probando el motor del soporte"
    tilt_motor.move_forward(top_testing_steps)
    time.sleep(0.5)
    tilt_motor.move_backwards(top_testing_steps)
    time.sleep(0.5)
    print "[DONE]"
```

7. **Calculate_moves() y get_position():**

Este método calcula cuantos pasos debemos mover los motores para que apunten al objetivo, así como la dirección en la que debemos movernos. Hace uso de un método auxiliar, get_position(), que contiene dos arrays con las posiciones posibles de cada motor. Dicho método nos devuelve la posición en pasos equivalente según la posición que ocupa el objetivo en píxeles. Como se explica en esta memoria, los valores utilizados proceden de una calibración manual realizada en las primeras pruebas del programa.

Como podemos observar, el movimiento de los motores ha de realizarse mediante hilos, ya que si no se paralizaría el resto de la ejecución del programa, detección de nuevos objetivos incluida.

```
def calculate_moves(center_x, center_y):
    # Apertura cámara: 60 grados. Equivale a 38 pasos del motor
    target_x_position, target_y_position = get_position(center_x, center_y)
    steps_to_target_in_x = target_x_position - pan_motor.get_position()
    steps_to_target_in_y = target_y_position + tilt_motor.get_position()

    launch_threads(steps_to_target_in_x,steps_to_target_in_y)


def get_position(x_position,y_position):
    steps_x = []
    steps_y = []

    if steps_x is not []:
        for i in range(-13,14):
            steps_x.append(i)

    if steps_y is not []:
        for i in range(-7,8):
            steps_y.append(i)

    # 23.7 -> Sabemos que 640/x = 27, y 320/x = 13.5, así que x debe ser 23.
    # 32 -> 480/x = 15, y 240/x = 7.5, así que x debe ser 32
    return steps_x[int(x_position/23.7)],steps_y[ int(y_position/32)]
```

8. Move_motor():

Auxiliar del método anterior, se encarga de realizar el movimiento (e informar al usuario) una vez ha sido calculado.

```
def move_motor(motor, steps, direction):
    if print_movement_values == "True":
        print "MOTOR: " + motor.get_name() + ". LOCATION  : ["\
              + str(motor.get_position())  + "]. STEPS: " + str(steps)

    # Movemos motores
    if direction == FORWARD:
        if motor.get_name() == "BASE":
            motor.move_forward(steps)
            print "    --->"
        else:
            motor.move_forward(steps)
            print " UP "
    else:
        if motor.get_name() == "BASE":
            motor.move_backwards(steps)
            print "    <---"
        else:
            motor.move_backwards(steps)
            print " DOWN "
```

9. Launch_threads():

- Este método lanza los hilos correspondientes al movimiento de los motores según los objetivos.

```
def launch_threads(steps_to_target_in_x,steps_to_target_in_y):
    global pan_thread,tilt_thread
    if steps_to_target_in_x < 0:
        pan_thread = threading.Thread(target=move_motor(pan_motor, abs(steps_to_target_in_x), BACKWARD))
    else:
        pan_thread = threading.Thread(target=move_motor(pan_motor, abs(steps_to_target_in_x), FORWARD))

    if steps_to_target_in_y < 0:
        tilt_thread = threading.Thread(target=move_motor(tilt_motor, abs(steps_to_target_in_y), FORWARD))
    else:
        tilt_thread = threading.Thread(target=move_motor(tilt_motor, abs(steps_to_target_in_y), BACKWARD))

    pan_thread.start()
    tilt_thread.start()

    pan_thread.join()
    tilt_thread.join()
```

10. **Back_to_center():**

Este método hace que la torreta apunte a un objetivo imaginario que está justo en el centro de la ventana, por lo que vuelve a la posición inicial, lo que nos evita tener que recalibrar el sistema en cada ejecución.

```
def back_to_center():
    calculate_moves(320,240)
    time.sleep(0.5)
    print " [INFO] Colocados motores en posición inicial"
```

11. **Vacuum_cleaner():**

Método de limpieza del programa, se encarga de liberar la cámara, cerrar todas las ventanas que se han abierto y liberar recursos (GPIO y cámara).

```
# Liberamos cámara, GPIO y cerramos ventanas
def vacuum_cleaner():
    camera.release()
    time.sleep(1)
    cv2.destroyAllWindows()
    print "[DONE] Roomba pasada. Fin del programa"
```

6.2.2.2 – Desarrollo del código

A continuación, se analizará el desarrollo del resto del código, separado en bloques para facilitar su lectura:

Inicialización

En este bloque, cargamos la configuración de nuestro fichero JSON, esperamos a que la cámara esté lista e inicializamos las variables utilizadas y los objetos necesarios para manejar los motores.

Es especialmente relevante el hecho de esperar por la cámara, ya que OpenCV no nos proporciona como tal una función que nos permita bloquear la ejecución hasta que la cámara esté preparada, por lo que hubo que implementar un bucle del que el programa no puede salir hasta que sea capaz de capturar la cámara. Antes de tener en cuenta este aspecto, se producían frecuentes errores de ejecución, ya que la disponibilidad de la cámara dependía del momento de la última ejecución, y si de esta se había liberado correctamente o no, por los que el mismo código unas veces funcionaba perfectamente y otras no arrancaba.

Es oportuno señalar que todo el código del programa esta dentro de un bloque try-catch. Esto se ha definido así para asegurarnos de que independientemente de como se finalice el programa (Ya sea con la tecla de salida o con CONTROL + C), siempre se liberarán la cámara y los GPIO. Esto ayuda a evitar errores relacionados con la disponibilidad del hardware, como el anteriormente señalado.

```
try:
    load_config()    # Cargamos "config.json"

    # Capturamos webcam
    print "[START] Preparando cámara...."
    camera_recording = False

    while camera_recording is not True:
        camera = cv2.VideoCapture(0)
        time.sleep(1)

        # Esperamos a que la cámara esté preparada
        camera_recording, _ = camera.read()
    print "[DONE] Cámara lista!"

    print "[INFO] Inicializamos los motores..."
    pan_motor = Stepper("Base", 16,19,26)
    pan_motor.set_speed(5)
    print(pan_motor.print_info())

    tilt_motor = Stepper("Top", 16,6,13)
    tilt_motor.set_speed(5)
    print(tilt_motor.print_info())


    if test_motors == "True":
        motor_test()
```

Bloque 1 – Toma de imagen de referencia:

En este bloque comenzamos a leer de la cámara, y creamos las ventanas del streaming de video, el umbralizado y el desenfocado en escala de grises (Aunque no se muestran hasta más adelante).

Como podemos ver, en el caso de que no haya primer frame (es decir, es la primera vuelta al bucle), lo inicializamos, tomando la imagen de referencia y esperando un tiempo prudencial para que sea fiable. Recordemos que con nuestro método de substracción de fondo actual nuestra imagen de referencia se construye gradualmente, no es una sola imagen.

```
# Loop sobre la camara
while True:

    # Leemos el primer frame e imprimimos el texto
    (video_signal, frame) = camera.read()
    text = message_target_not_detected

    # Pasamos el frame a escala de grises, y lo desenfocamos (Facilitamos umbralizamos)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)

    # Si no hay primer frame, lo inicializamos
    if firstFrame is None:
        if actualFrame is None:
            print "[INFO] Empezando captura de vídeo... "
            actualFrame = gray
            continue
        else:
            #  Calculamos el frame Delta (Diferencia absoluta entre
            # primer frame y # el frame actual)
            abs_difference = cv2.absdiff(actualFrame, gray)
            actualFrame = gray
            thresh = cv2.threshold(abs_difference,5, 255, cv2.THRESH_BINARY)[1]
            thresh = cv2.dilate(thresh, None, iterations=2)

            if count > 30:
                print "[INFO] Esperando movimiento..."
                if not cv2.countNonZero(thresh) > 0:
                    firstFrame = gray
                else:
                    continue
            else:
                count += 1
                continue
```

Bloque 2 – Frame Delta y umbralizado

En este bloque, cogemos la imagen obtenida en el apartado anterior y calculamos las diferencias con el frame actual (obteniendo el frame Delta), que luego umbralizamos para así realizar la detección de movimiento de forma más sencilla.

```
# frameDelta = Diferencia absoluta entre la imagen actual y el frame de referencia
frameDelta = cv2.absdiff(firstFrame, gray)
thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

# Dilatamos la imagen umbralizado, para asi buscar sus contornos
thresh = cv2.dilate(thresh, None, iterations=2)
```

Bloque 3 – Búsqueda de objetivos y movimiento

En este bloque actualizamos el objetivo actual. Sólo puede existir un "`best_contour`" si se ha detectado un objetivo. Parece un bloque pequeño, pero si observamos las funciones explicadas anteriormente, es donde se realiza prácticamente todo el trabajo del programa, ya que una vez que se detecta un objetivo, se calcula su centro, y, una vez tenemos el centro, calculamos los movimientos necesarios de los motores, y les ordenamos que realicen dicho movimiento.

```
# Buscamos el contorno del mayor objetivo
best_contour = find_best_target()

# Bucle sobre los contornos
if best_contour is not None:
    draw_target_frame(best_contour)
    text = message_target_detected
```

Bloque 4 - Ventanas

En este bloque imprimimos la fecha en la ventana de la webcam y abrimos el resto de las ventanas generadas (Umbralizado y Frame Delta).

```
# Mostramos las ventanas y les añadimos el texto
print_info_on_video()
cv2.imshow("Cámara", frame)
cv2.imshow("Umbralizado", thresh)
cv2.imshow("Frame Delta", frameDelta)
```

Bloque 5 – Limpieza

Bloque final, en el que controlamos si el usuario pulsa la tecla de salida, y liberamos los recursos y cerramos las ventanas utilizadas. También devolvemos el motor a su posición original.

```
        # Comprobamos si el usuario quiere salir
        key = cv2.waitKey(1) & 0xFF
        if key == ord(exit_key):
            print " [INFO] Apagando el sistema..."
            break

finally:
    # Liberamos recursos, cerramos ventanas y colocamos el motor
    back_to_center()
    vacuum_cleaner()
```

6.2.2.3 – Otros ficheros

Además del fichero principal del programa, son necesarios en este proyecto tres ficheros más, disponibles en el archivo adjunto, y explicados a continuación.

Fichero “config.json”:

Fichero de configuración del programa, de tipo JSON, en el que se pueden indicar las diferentes opciones disponibles para la ejecución de este, como la velocidad de los motores, la ejecución de las pruebas de movimiento o la elección de los colores para los marcos y centros de los objetivos.

Ficheros ”full_calibration”, “calibrate_pan” y “calibrate_tilt”

Estos archivos nos permiten hacer la calibración inicial d los motores. Para el buen funcionamiento del programa es necesario que el haz de luz del láser esté alineado con el centro de la imagen, tanto en vertical como en horizontal.

7. Resultados

En esta sección se analizarán los diferentes resultados obtenido a lo largo de las diferentes iteraciones que forman parte del proyecto, definidas según los objetivos que aparecen en la tabla 4.3.

Iteración 1 – Instalación de OpenCV y prueba de cámara

En esta iteración, se procedió a la instalación de OpenCV. Es una tarea que a priori parece sencilla, pero en este caso fue costosa, ya que la compilación de OpenCV es inestable, y al intentar utilizar los cuatro núcleos de la Raspberry Pi para acelerar el proceso se producían errores, lo que provocaba un fallo cuando se alcanzaban porcentajes en torno al 30% del progreso.

Para que la instalación finalizara sin errores, fue necesario ejecutarla utilizando sólo uno de los núcleos de la placa, tardando el proceso aproximadamente cinco horas en completarse.

La verificación se realizó comprobando que era posible el acceso a la cámara desde el software, utilizando OpenCV y un programa sencillo que mostrara la imagen de la cámara.

Iteración 2 – Detección básica de movimiento y dibujo de marcadores

En esta iteración, desarrollamos una versión inicial del código que permite una detección básica del movimiento, utilizando el método de la substracción de fondo con imagen de referencia [33]. Este método consiste en tomar una imagen de referencia del fondo en la que no haya ningún objeto en movimiento.

Una vez tenemos la imagen de referencia, podremos detectar el movimiento restando el fotograma actual con dicha imagen. En este caso, escogemos imagen de referencia el primer frame que capturamos de la cámara.

Cuando detectamos movimiento, dibujamos un marco para identificarlo utilizando las funciones propias de OpenCV.

La verificación se realizó comprobando que el sistema era capaz de detectar movimiento, y de señalarlo en el vídeo que la cámara nos proporciona. El sistema era capaz de hacer estas detecciones, pero tenía un número elevado de falsos positivos, sobre todo debido a cambios de luz y reflejos.

Al tomar una imagen de referencia, cualquier variación algo brusca de luz hace que la imagen actual sea diferente a la inicial, por lo que una nube puede llevar a la detección de falsos positivos.

Otro problema que fue encontrado de forma casual fueron los reflejos de la madera. En la habitación en la que se realizaron las pruebas se encuentra un armario de madera de color claro, que produce reflejos con facilidad. Al igual que los cambios de luz, estos reflejos acaban llevando a la detección de falsos positivos.

Debido a los problemas encontrados y a los falsos positivos que acarrean, se decide mejorar el código y cambiar a un método de detección de movimiento más robusto y resistente a los cambios de luz, llevando esta decisión a la siguiente iteración.

Iteración 3 – Detección mejorada de objetivos

En esta iteración, se cambió el método de substracción de fondo a la modalidad de la substracción con los fotogramas anteriores [34]. En este caso, el fondo o segundo plano se obtiene de los programas anteriores.

Funciona de manera similar al anterior, pero en este caso se consigue que la imagen de referencia sea capaz de adaptarse a ciertos cambios. El proceso es el siguiente:

1. Tomamos una imagen de referencia, dejando pasar un tiempo aplicando un retardo.
2. Al haber dejado pasar el tiempo de retardo, hemos obtenido una imagen de referencia que es resultado de la acumulación de imágenes anteriores.
3. Una vez tenemos la imagen de referencia "mejorada", repetimos el proceso anterior, y empezamos a comparar con los nuevos fotogramas que obtenemos.

Este método es mucho más robusto frente a los cambios de iluminación, aunque tiene un pequeño fallo: no es capaz de detectar siluetas. Esto ocurre debido a que, si el objetivo se queda inmóvil, pasa a ser parte de la imagen de referencia.

Para este proyecto, no ha sido considerado este aspecto como un factor relevante, ya que, debido a la configuración del sistema, para que un objetivo no sea detectado por este motivo tendría primero que entrar en el área de la cámara (en cuyo caso será detectado) y permanecer inmóvil en ella. Del mismo modo, cuando quiera salir de la zona que abarca la cámara, tendrá que moverse de nuevo, por lo que volverá a ser detectado.

Por estos motivos, aunque es cierto que puede evitarse la detección como posible objetivo de la torreta permaneciendo inmóvil, no se ha considerado como fallo, ya que podemos suponer que el jugador del equipo contrario no conoce el funcionamiento del sistema. También tenemos en cuenta que la reacción natural de una persona sería ponerse fuera del alcance de la torreta, con lo cual es un caso improbable el hecho de que un jugador tenga la sangre fría suficiente de permanecer inmóvil sabiendo que está siendo apuntado. A su vez, las reglas del airsoft establecen que basta un impacto para eliminar a un jugador, con lo cual no debería ser posible llegar a pararse delante del dispositivo sin ser impactado. (Todo esto suponiendo que hemos incorporado un mecanismo de disparo a nuestro sistema de apuntado).

En esta iteración, se realizó la verificación comprobando que el sistema era capaz de detectar y señalar el movimiento que ocurre en su área de acción.

Debemos recordar que en este momento marcamos todos los objetos en movimiento detectados. A veces, por la diferencia de color (como puede ocurrir con el anverso y el reverso de una mano), un mismo objetivo es detectado como si fuera varios objetos diferentes. Por este motivo, así como por el hecho de que, con un solo mecanismo de disparo es físicamente imposible impactar a varios objetivos a la vez, se llegó a la conclusión de que necesitamos un método para quedarnos con un solo objetivo a impactar, lo que nos lleva a la iteración siguiente

Iteración 4 – Dibujar sólo el objetivo de mayor área

En este momento, nuestro sistema es capaz de detectar y marcar los posibles objetivos en movimiento. Como señalamos en la iteración anterior, necesitamos un método que nos reduzca el ruido, dejándonos con un solo objetivo.

Como sabemos que un mismo objeto puede detectarse como si fueran objetivos diferentes dependiendo de los colores y las formas que este tenga, se decidió que el objetivo a impactar será aquel que mayor área tenga, ya que, entre más grande sea nuestro objetivo, más fácil será conseguir un impacto certero.

Pensando en el futuro, y en la manipulación del mecanismo de disparo, en esta iteración se procede a dibujar (con las funciones que nos proporciona OpenCV) el centro del objetivo detectado, que será el punto que guíe después el movimiento de los motores.

Además de la reducción de posibles objetivos a sólo aquel de mayor área, se añaden también una serie de opciones de configuración de manera sencilla, admitiendo parámetros como el tamaño mínimo del objetivo a detectar. Dichos parámetros se establecían a la hora de ejecutar el programa en la propia línea de comandos.

En esta iteración, realizamos la verificación comprobando visualmente que el sistema es capaz de detectar movimiento, recorrer los contornos y señalar sólo el objetivo de mayor área, marcando su centro mediante un punto. También comprobamos que el programa lee los parámetros que le pasamos por comandos, y es capaz de aplicarlos en su ejecución. Una vez conseguido este objetivo, pasamos a las dos siguientes iteraciones, consistentes en el control de los motores de la base y el soporte del mecanismo.

Iteración 5 – Control del motor de la base

Para controlar los motores, como se refleja en la lista de componentes, en un primer momento se pensó utilizar el Motor HAT, por lo que fue necesario adquirir la placa.

Dicha placa no trae los terminales puestos de fábrica, por lo que el primer paso para poder utilizarla fue soldarle dichos terminales. Una vez soldados se procedió a controlar los motores. En las especificaciones de la placa se indicaba que era capaz de alimentar los motores a través de la propia Raspberry, aunque no era recomendable. Después de realizar una serie de pruebas, fue comprobado que no era capaz de mover los motores utilizados en esta ocasión, por lo que era necesario adquirir una fuente de alimentación externa. Se adquirió una fuente de alimentación de voltaje variable y capaz de proporcionar 2.1 amperios de salida, energía suficiente para alimentar los dos motores.

El siguiente paso fue la instalación de los drivers de Adafruit, así como el estudio de la librería proporcionada con el Motor HAT, para así aprender a controlar los motores.

Para comprobar el buen estado del motor de la base, se utilizó el fichero "`StepperMotorTest.py`", proporcionado por Adafruit como parte de su librería de control. Al ejecutar este archivo se realizan serie de pruebas básicas (pequeños movimientos con el motor), a fin de comprobar que funciona todo correctamente.

En este momento, verificamos que somos capaces de controlar motores paso a paso utilizando el fichero proporcionado por Adafruit. Tomando como base el archivo de prueba de Adafruit, se añadió una sección de test al código del programa, en el que se realizan pequeños movimientos con el motor, para evitar errores relacionados con malas conexiones o cables sueltos.

Sin embargo, al montarlos en la estructura, comprobamos que no son capaces de manejar el peso de la base. Por ello ((y como se comenta anteriormente), se cambió esta pieza de hardware por dos L298N.

Para comprobar su correcto funcionamiento, se probaron conectándolos a un Arduino Uno, ya que su IDE nos proporciona librerías y programas de prueba para motores paso a paso.

Una vez comprobado que todo funcionaba, se intentó buscar una librería de Python compatible con dichos controladores. Había algunas ya existentes, aunque en su mayoría estaban orientadas a utilizar motores DC (También compatibles con los L298N).

De las librerías probadas, no se encontró ninguna que funcionara de la manera esperada, por lo que se procedió a diseñar una librería propia.

Ya que se diseñaba pensando en el programa, se aprovechó el proceso para añadirle una serie de métodos que no nos proporcionaba la librería de Adafruit, lo que fue bastante útil a la hora de desarrollar el programa.

Por ejemplo, con el Motor HAT, no teníamos un método que nos dijera cuánto se había movido el motor desde su primer paso, pero nada nos impidió incorporarlo en la librería desarrollada.

En mi vida académica nunca se había dado el caso de que me viera obligado a desarrollar una librería, por lo que fue una experiencia muy interesante.

No obstante, como se comentó anteriormente, estos drivers producían demasiadas vibraciones, especialmente en movimientos pequeños, ya que es algo brusco el movimiento paso a paso. Por ello, fueron sustituidos por los drivers A4988. Para estos drivers si que existía alguna librería, pero como ya se tenía el algoritmo del programa diseñado en torno a la librería propia, se prefirió adaptar la librería de los L298N a los A4988 en vez de buscar una externa, ya que esta tenía los métodos exactos que se necesitaba.

Fue una tarea fácil, ya que usar los A4988 nos abstrae de la gestión de pasos. El A4988 gestiona internamente la combinación de corriente en las bobinas necesaria para cada paso, utilizando sólo dos pines de control: uno de dirección y otro de paso.

Una vez conseguido el control de los motores desde Python, procedemos a desarrollar los métodos necesarios para indicarles que apunten a los objetivos móviles.

Gracias al método que nos permite saber cuántos pasos se han realizado, podemos saber en qué posición se encuentra actualmente el motor, pero no es posible relacionarlo directamente con la imagen de la cámara, ya que en la anterior ejecución el mecanismo podría haberse quedado apuntando en cualquier dirección. Por ello, es necesario realizar una calibración previa, alineando el sistema con el punto “central” de ambos ejes. En el repositorio encontramos la carpeta “`calibrate`”, en la que hay archivos que nos permiten realizar dicha calibración, así como un programa que nos muestra la imagen de la cámara con el centro marcado. De esta forma, podremos alinear el láser con el centro de la imagen.

Ahora ya tenemos un sistema que sabe dónde está y cuántos pasos ha dado, pero no conoce sus límites. Es decir, detecta movimiento y puede moverse en su dirección, pero, como no tenemos ningún tipo de relación entre la imagen de la cámara y el movimiento del motor, no podemos saber si ha sobrepasado el ángulo que es capaz de ver la cámara, por lo que no podemos realizar disparos con certeza.

Después de buscar la hoja de datos de la cámara, comprobamos que su ángulo de apertura es de 60 grados. Sabiendo que la imagen es de 640px de ancho, y que cada paso de nuestros motores equivale a un giro de 1.8 grados, podemos establecer una relación entre la imagen capturada y el número de pasos que puede dar el motor dentro de ese rango, de la siguiente manera:

1. El campo de visión de la lente de la cámara es de 60 grados.
2. Esto significa que, si alineamos el mecanismo de disparo con la lente de la cámara, podremos movernos 30 grados a la derecha y 30 grados a la izquierda, sin salirnos de su rango de visión.
3. Cada paso del motor hace que se mueva 1.8 grados.

Con estos datos, podemos obtener el número de pasos que puede dar el motor sin salirse del ángulo de la cámara de la siguiente manera:

$$Número\ de\ pasos = \frac{Ángulo\ de\ apertura\ de\ la\ cámara}{Grados\ que\ equivalen\ a\ un\ paso}$$

Si realizamos la operación, hallamos que el ángulo de apertura de la cámara equivale a unos 33 pasos en total. Es decir, que, si partimos del centro, podremos dar unos 16 pasos para cada lado.

También sabemos que la imagen es de 640x480 píxeles, y que podemos recorrerla con 33 pasos, por lo que realizamos el siguiente cálculo:

$$Píxeles\ por\ paso = \frac{Ancho\ de\ la\ imagen}{Pasos\ a\ los\ que\ equivale}$$

En este caso, un paso equivale a unos 19 píxeles de la imagen.

En las iteraciones anteriores hemos hallado el centro de los objetivos, definido por unas coordenadas x e y. Si queremos saber en qué posición se encuentra el objetivo respecto al eje X, solo tendremos que dividir la ubicación de su centro entre el número de píxeles por paso:

$$Ubicación\ del\ objetivo\ (eje\ X) = \frac{Coordenada\ X\ de\ su\ centro}{Píxeles\ por\ paso}$$

De esta forma, obtenemos un número entre 0 y 38 que nos da su ubicación como si de una recta numérica imaginaria se tratase, en la que el 19 sería el centro, punto de partida del motor.

En teoría, si suponemos que el primer objetivo tiene su centro en la coordenada (570, 315), el cálculo sería el siguiente:

$$Ubicación\ del\ objetivo\ (eje\ X) = \frac{570}{19} = 30$$

$$Ubicación\ del\ objetivo\ (eje\ Y) = \frac{315}{15} = 21$$

Como es el primer objetivo, el motor parte alineado con la cámara, es decir, en la posición 19, por lo que, para apuntar al objetivo, tendríamos que movernos 11 pasos a la derecha con el motor de la base y 5 pasos hacia arriba con el motor del soporte (480px, el centro estará en los 16 pasos) .

Esto es la primera impresión que nos da el sistema, pero en estos cálculos estamos obviando un aspecto clave: la cámara no está en la misma posición que el motor, por lo que el cálculo no es correcto. Al meter desplazamientos de por medio el cálculo se nos volvía demasiado complejo, por lo que se prefirió una calibración manual, utilizando los archivos "`full_calibration`" y "`usb_camera_calibration`". Así, es posible obtener las posibles coordenadas de un objetivo según los pasos que pueda dar el láser en el área que cubre la cámara

Una vez calibrado el sistema, pasamos a su configuración. En una iteración anterior, indicamos que realizábamos la configuración por línea de comandos. Después de utilizar este método en diferentes ejecuciones, comprobamos que es muy tediosa la tarea de definir en línea de comandos las variables a utilizar, ya que lleva a errores con una relativa facilidad.

Por ello, después de estudiar las diferentes alternativas que nos ofrece Python, se decide definir un archivo de configuración, "`config.json`", en el que se indican todas las opciones necesarias para la ejecución del programa, como los colores utilizados para marcar los objetivos, su tamaño mínimo o el número de pasos que darán los motores en la fase de test. Estos parámetros son cargados a nuestro programa utilizando la librería JSON disponible para Python.

20 lines (20 sloc) | 438 Bytes

```
{
  "GENERAL": {
    "MINIMUM_TARGET_AREA": 500,
    "EXIT_KEY": "q",
    "TARGET_FRAME_COLOR": "0,255,0",
    "TARGET_CENTER_COLOR": "0,0,255"
  },
  "MESSAGES": {
    "TARGET_DETECTED" : "Objetivo detectado",
    "TARGET_NOT_DETECTED":"No hay objetivos"
  },
  "MOTOR": {
    "BASE_TESTING_STEPS": 13,
    "TOP_TESTING_STEPS": 7
  },
  "DEBUG": {
    "TEST_MOTORS" : "False",
    "PRINT_MOVEMENT_VALUES": "True"
  }
}
```

Figura 7.1 - Estado actual del fichero "config.json"

Finalizada esta iteración, pasamos a la siguiente, en la que procederemos a montar el soporte del láser, y a controlar su motor para que apunte al objetivo.

Iteración 6 – Control del motor del soporte

Comenzamos esta iteración montando el soporte superior, consistente en una pieza de madera atornillada a la base. En esta pieza se fijará el láser, utilizado una sección de una alfombrilla de ratón y tres tornillos con sus respectivas tuercas. De esta forma hacemos que el puntero gire en el eje Y. Se encargó otra madera, con un orificio de 12 milímetros, por si en algún momento se quiere cambiar el láser por algún otro elemento, como podría ser un mecanismo de disparo o un láser más potente.

Para mover los dos motores a la vez fue necesario incorporar la programación multihilo a nuestro software, permitiendo así calcular el movimiento y mover ambos motores simultáneamente, reduciendo así el desfase entre lo que ocurre en el mundo real y los movimientos del sistema.

Iteración 7 – Pruebas

En esta iteración, se procedió a realizar una serie de pruebas con el sistema, consistentes en el paso de diferentes objetivos por su rango de alcance, comprobando si era capaz de detectarlos o no. La tasa de detección es bastante buena, aunque falla en algunas ocasiones, como en el caso de que aparezca alguien con un color de piel o de ropa parecido al color del fondo, aunque en este caso nos encontramos con limitaciones del hardware y la librería utilizadas. Recordemos que la parte lógica del proyecto la lleva un SBC, que cumple con su función, pero nunca será tan potente como un computador convencional.

En esta fase fue en la que se descubrió que se producían rozamientos con la estructura en el giro, o pequeños tambaleos en la base, lo que hace que se pierdan pasos. Por ello, se decidió cambiar la pieza 3D por el cojinete mencionado anteriormente, y alargar el eje del motor, lo que mejoró en gran medida el giro del sistema.

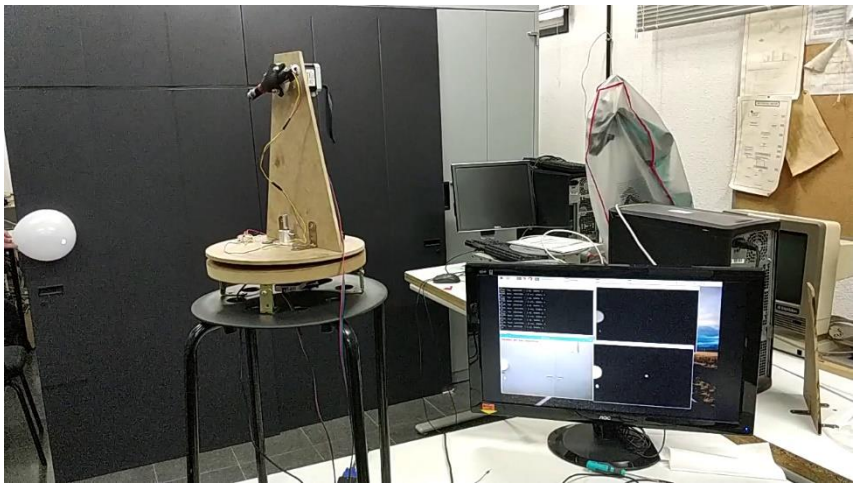


Figura 7.2 – Pruebas del sistema

8. Conclusiones y trabajo futuro

8.1 – Conclusiones

Este trabajo de fin de grado me ha servido para poner en práctica todos los conocimientos adquiridos a lo largo del grado, así que puedo afirmar que ha tenido un impacto positivo en mi educación. El utilizar componentes de hardware y electrónica me ha hecho ver que, lógicamente, es mucho más difícil realizar un sistema completo que el código para un sistema ya cerrado.

El hecho de tener que elegir todos los componentes me ha supuesto una toma de decisiones a la que no estoy acostumbrado, como el elegir entre potencia o portabilidad, o el tener que asegurar la compatibilidad de todos los componentes del hardware entre sí. Una mala elección de los componentes podría haber provocado fallos en el funcionamiento del sistema, que hubieran retrasado irremediablemente la finalización de este trabajo.

Un aspecto positivo ha sido la utilización de componentes electrónicos a los que no estaba acostumbrado, ya que ha hecho que me tuviera que formar en campos en los que mi conocimiento era superficial, para así entender su funcionamiento, y elegir así cual era el más adecuado para mi propósito.

Este proyecto también me ha hecho valorar más el software Open Source, ya que hubiera sido imposible desarrollar en el tiempo que tenemos para el Trabajo de Fin de Grado una librería completa de reconocimiento de imagen que funcione de la misma manera que OpenCV, que ha sido desarrollada sin ánimo de lucro por su comunidad de usuarios.

En definitiva, el TFG me parece un método ideal de evaluación, ya que te hace poner a prueba tus conocimientos en un proyecto escogido por ti mismo, lo que hace que lo elabores porque te interesa, no porque la universidad te lo exija. Es mucho más fácil trabajar cuando el proyecto que estás elaborando te parece interesante, ya que te anima a ver hasta dónde puedes llegar.

8.2 – Trabajo futuro

Aunque el trabajo realizado es funcional, es cierto que podría mejorarse, o incluso añadirle nuevas funcionalidades. Algunas ideas para ello son las siguientes:

1. **Mejorar la estructura,** ya que puede ser el principal problema del proyecto. Sería ideal una estructura que fuera un poco más estable frente a las vibraciones del motor, y que evitara aún más los posibles roces y resistencias al giro correcto de la estructura. También se podría profesionalizar el diseño, quizás con un soporte impreso en 3D que esconda toda la electrónica y le dé un aspecto más real al sistema.
2. **Añadirle un sistema de desactivación,** quizás algún interruptor o algún tipo de sensor de presión que, cuando sea accionado, desactive el sistema temporalmente, para que sea posible que un jugador lo suficientemente hábil pueda deshabilitar el sistema durante un tiempo reducido (Pensando en su aplicación dentro de una partida de Airsoft).
3. **Incorporarle sonido:** Podría llegar a ser interesante el añadir unos altavoces que emitieran avisos de las diferentes fases del proceso, o los clásicos mensajes como "Activado", "Apagando el sistema" o similares.

En el caso en el que se contara con un mayor número de desarrolladores, o más tiempo, una modificación muy interesante (aunque muchísimo más compleja) sería el convertirlo en un sistema móvil, que sea capaz de cambiar de ubicación de forma autónoma, o que pueda ser controlado remotamente.

Es una idea que es posible realizar, aunque complicaría enormemente la detección del movimiento, ya que la toma de la imagen de referencia en movimiento haría demasiado complejo saber qué es parte del fondo y que no, de ahí que se crea necesario la incorporación de más desarrolladores al proyecto.

Otra modificación posible sería añadirle un sistema de detección de colores, que solo realice el disparo si comprueba que la persona es del equipo contrario. Fue descartada para este proyecto porque la vestimenta habitual en este tipo de actividades es variada, ya que se suele utilizar camuflaje, pero no el mismo tipo para todos los equipos.

Es cierto que podría incorporarse un brazalete o similar de colores vivos, pero según la postura del jugador podría no verse, con lo que el sistema acabaría impactando a jugadores del mismo equipo, o no impactando a los jugadores del equipo contrario, a la vez que añade complejidad al cálculo.

Bibliografía

Páginas web

[1] Hackster.io. *Learn to design, create, and program electronics*. Recuperado el 11 de Mayo de 2018 de https://www.hackster.io

[2] Instructables. *How to make anything*. Recuperado el 11 de Mayo de 2018 de https://www.instructables.com/

[3] Diccionario Cambridge Inglés. *Significado de DIY en el Diccionario Cambridge inglés.* Recuperado el 11 de Mayo de 2018 de https://dictionary.cambridge.org/es/diccionario/ingles/diy

[4] Wikipedia en español. *Airsoft – Wikipedia, la enciclopedia libre.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Airsoft

[5] Hasbro. *Nerf Official Website | Nerf Blasters by Hasbro*. Recuperado el 11 de Mayo de 2018 de https://nerf.hasbro.com/es-es

[6] Google Developers. *Introduction to the Google Assistant Library | Google Assistant SDK for devices*. Recuperado el 11 de Mayo de 2018 de https://developers.google.com/assistant/sdk/guides/library/python/

[7] CodeCombat. *Learn to Code by Playing a Game.* Recuperado el 11 de Mayo de 2018 de https://codecombat.com/

[8] Wikipedia en español. *Desarrollo en cascada.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Desarrollo_en_cascada

[9] Wikipedia en español. *Scrum (desarrollo de software).* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[10] Wikipedia en español. *Desarrollo iterativo y creciente.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[11] Samuel Casanova. (Septiembre 2016) *Resumen Clean Code.* Recuperado el 11 de Mayo de 2018 de https://samuelcasanova.com/2016/09/resumen-clean-code/

[12] Stack Overflow. (2018). *Stack Overflow Developer Survey 2018*. Recuperado el 11 de Mayo de 2018 de https://insights.stackoverflow.com/survey/2018

[13] Robinson, D., Miller, A., Hanlon, J., & Ferrigno, R. (Diciembre 2017). *The Incredible Growth of Python*. Recuperado el 11 de Mayo de 2018 de https://stackoverflow.blog/2017/09/06/incredible-growth-python/

[14] Tiobe Index*. Documentation | Tiobe Index.* Recuperado el 11 de Mayo de 2018 de https://www.tiobe.com/documentation/

[15] CodeAcademy. (2013). *Learn Python*. Recuperado el 11 de Mayo de 2018 de https://www.codecademy.com/learn/learn-python

[16] Realsentrygun.com. (Junio 2017). *Real-Life Sentry Guns for Sale - Home*. Recuperado el 2 de Mayo de 2018 de https://realsentrygun.com .

[17] Engmann, N. (Febrero 2018). *Nerf Alexa Home Defense Turrent*. Recuperado el 2 de Mayo de 2018 de https://www.hackster.io/quodcertamine/nerf-alexa-home-defense-turrent-a50dd1

[18] Amazon. *Amazon Alexa***.** Recuperado el 2 de Mayo de 2018 de https://developer.amazon.com/alexa

[19] Noolas, D. (Enero 2018). *Nio Nerf Raspberry Pi, Oh My!*. Recuperado el 2 de Mayo de 2018 de https://www.hackster.io/tyler-lugger/nio-nerf-raspberry-pi-oh-my-1fa71c

[20] Wikipedia. *Thresholding (image processing*). Recuperado el 11 de Mayo de 2018 de https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[22] PC Mag. *Encyclopedia.* Recuperado el 11 de Mayo de 2018 de https://www.pcmag.com/encyclopedia/term/41105/delta-frame

[23] Sketchfab*. Your 3D content on web, mobile, AR, and VR*. Recuperado el 11 de Mayo de 2018 de https://sketchfab.com/

[24] Pillow**.** *Pillow – Pillow (PIL fork)***.** Recuperado el 11 de Mayo de 2018 de https://pillow.readthedocs.io/en/5.1.x/

[25] SimpleCV. *SimpleCV.* Recuperado el 11 de Mayo de 2018 de http://simplecv.org/

[26] OpenCV. *OpenCV library.* Recuperado el 11 de Mayo de 2018 de https://opencv.org/

[27] Wikipedia en español. *Motor de corriente continua.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Motor_de_corriente_continua/

[28] Wikipedia en español. *Servomotor.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Servomotor/

[29] Wikipedia en español. *Motor paso a paso.* Recuperado el 11 de Mayo de 2018 de https://es.wikipedia.org/wiki/Motor_paso_a_paso

[30] Wikipedia en español. *Airsoft.* Recuperado el 21 de Mayo de 2018 de https://es.wikipedia.org/wiki/Airsoft

[31] HackerShack. *Raspberry Pi Motion Tracking Gun Turret* . Recuperado el 19 de Mayo de 2018 de https://www.hackster.io/hackershack/raspberry-pi-motion-tracking-gun-turret-77fb0b

[32] Miguel Arlandy Rodríguez. *Clean Code – Reglas y principios.* Recuperado el 11 de Mayo de 2018 de https://www.adictosaltrabajo.com/tutoriales/clean-code-reglas-principios/

[33] y [34] Luis del Valle Hernández. (Febrero 2017) *Detección de movimiento con OpenCV y Python.* Recuperado el 2 de Mayo de 2018 de https://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/

Referencias de las figuras

[1.1] *Réplica de sistema explosivo, elaborada con Arduino*, Recuperada el 16 de Mayo de http://www.instructables.com/id/Arduino-defuseable-bomb-perfect-for-airsoft-games-/

[3.1] *Desarrollo en cascada.* Recuperado el 12 de Mayo de 2018 de https://es.wikipedia.org/wiki/Desarrollo_en_cascada

[3.2] *Modelo iterativo incremental*. Recuperado el 12 de Mayo de 2018 de http://isescom.blogspot.com.es/2013/08/desarrollo-en-cascada-vs-desarrollo.html

[5.1] *Raspberry Pi Model 3B.* Recuperado el 2 de Mayo de 2018 de https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[5.2] *Motor de corriente continua*. Recuperado el 2 de Mayo de 2018 de http://www.geekbotelectronics.com/motores-de-dc/

[5.3] *Servomotor.* Recuperado el 2 de Mayo de 2018 de http://panamahitek.com/que-es-y-como-funciona-un-servomotor/

[5.4] *Motor Nema 17HS2048.* Recuperado el 2 de Mayo de 2018 de https://www.amazon.es/Motor-Pasos-Fases-Cables-Impresora/dp/B06ZY9G8KG/ref=sr_1_1?ie=UTF8&qid=1525860578&sr=8-1&keywords=nema+17

[5.5] *Adafruit Motor Hat.* Recuperado el 2 de Mayo de 2018 de https://www.adafruit.com/product/2348

[5.6] *Keyes L298N V2. Recuperado el 2 de Mayo de 2018 de http://www.dx.com/es/p/keyes-l298n-v2-dc-stepper-gear-motor-drive-module-red-5v-334707*

[5.7] *A4988 StepStick Stepper Motor Driver with Heatsink. Recuperado el 3 de Junio de 2018 de https://www.amazon.in/StepStick-Stepper-Driver-Heatsink-Printer/dp/B01GECZDEU*

[5.8] *Webcam Logitech C270.* Recuperado el 2 de Mayo de 2018 de https://www.logitech.com/es-es/product/hd-webcam-c270

[5.9] *Taburete Marius.* Recuperado el 2 de Mayo de 2018 de https://www.ikea.com/es/es/catalog/products/10135659/

[5.10] Fuente de alimentación oficial para Raspberry Pi. Recuperado el 2 de Mayo de 2018 de https://www.pccomponentes.com/fuente-de-alimentacion-para-raspberry-pi-51v-25a-blanca

[5.11] Fuente de alimentación universal, regulable entre 4.5 y 12 voltios, de 2.5 amperios de salida. Autoría propia

[5.12 (A)] *Pieza para adaptar un motor Nema17 a una superficie plana.* Recuperado el 2 de Mayo de 2018 de https://sketchfab.com/models/39a08ad6a0084d7ebbfdc5c3fdde5a5c

[5.12 (B)] *Soporte del mecanismo de disparo en el eje Y.* Figura base (fue adaptada) recuperada el 2 de Mayo de 2018 de https://sketchfab.com/models/f3d0cbacafa5476dad12faa5d05735e7

[5.12 (C)] *Impresora Ultimaker 2 Go.* Recuperado el 2 de Mayo de 2018 de https://ultimaker.com/en/products/ultimaker-2-go

[6.1] *Estructura de la raíz del repositorio en Github*. Autoría propia

[6.2] *Carpeta "tests" del repositorio.* Autoría propia

[6.3] *Carpeta "calibrate" del repositorio*. Autoría propia

[6.4] *Carpeta "L298N" del repositorio*. Autoría propia

[6.5] *Fases del modelo iterativo.* Recuperado el 2 de Mayo de 2018 de http://isescom.blogspot.com.es/2013/08/desarrollo-en-cascada-vs-desarrollo.html

[7.1] *Estado actual del fichero "config.json".* Captura de autoría propia, recuperada el 3 de Junio de 2018 del repositorio del trabajo

Anexos

Anexo 1 - Diseño de piezas

1.1 – Madera

Para todas las piezas elaboradas en madera se utilizó un tablero de fibras de densidad media (DM), con un espesor de 6 MM-

Base:

Para la base, se utilizaron dos discos de 25 centímetros de diámetro.

1. **Disco inferior**:
   En este disco se realizó una perforación de 15 milímetros de diámetro, para permitir el paso del eje del motor. También se realizó una rebaja de 35 milímetros en el centro, colocándose el cojinete en el interior de la rebaja. Este disco es de 21 milímetros de grosor.

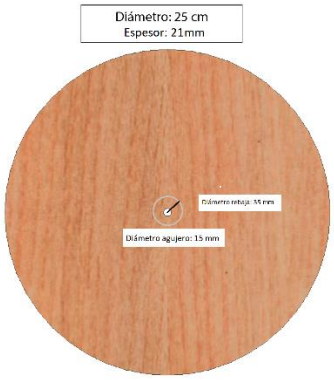


Figura 1. Disco inferior de la base

2. **Disco superior:**
   Se trata de una pieza idéntica a la anterior, pero en DM de 6 milímetros de grosor.

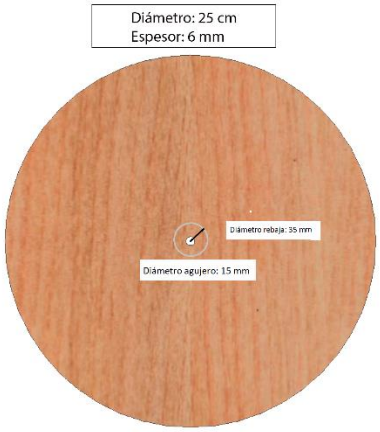


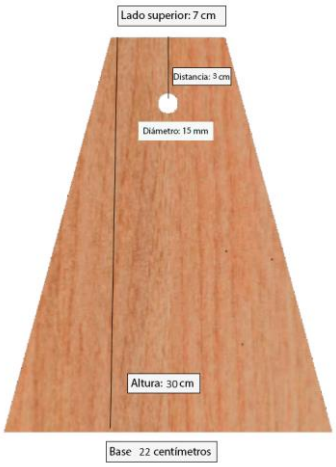
Figura 2. Disco superior de la base

Soporte:

En el soporte del mecanismo de disparo se utilizó una pieza de forma trapezoidal..

1. **Soporte láser:**

   En esta pieza se practicó un agujero de 15 milímetros para que pudiera pasar el eje del motor, de forma idéntica al disco inferior de la base. En este caso se utilizó una de las piezas impresas en 3D para sujetar el motor a la madera.



1.2 – Impresión 3D

Fueron necesarios dos modelos 3D, correspondientes a la pieza que se encarga de sujetar el láser al motor de la parte superior, y a la pieza que sujeta la parte superior de la base con la prolongación del eje.

Unión eje-láser

Se trata de una pieza consistente en dos discos, uno encima de otro:

- **Disco inferior:** Tiene un diámetro de 4.5 centímetros, y tres orificios para permitir que sea sujeto con tornillos.
- **Disco superior:** Diámetro de 2.4 centímetros.

Ambos discos tienen un orificio en su centro del tamaño de los ejes del motor, para permitir así el paso de dicho eje.

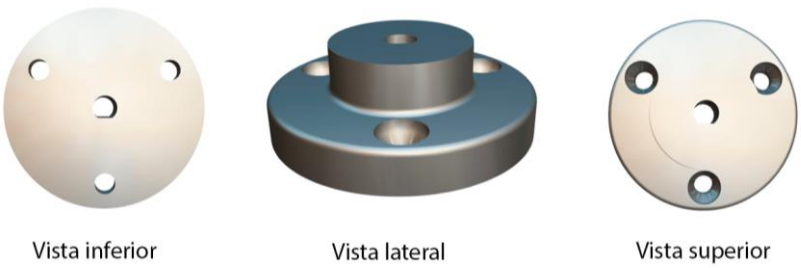


Figura 5 – Vistas de la pieza que se encarga de la unión motor - láser

Unión base – eje:

Es una modificación de la pieza anterior, alargando el disco superior y añadiendo el hueco para el tornillo que la ajustará al eje. Fue fabricada en aluminio.

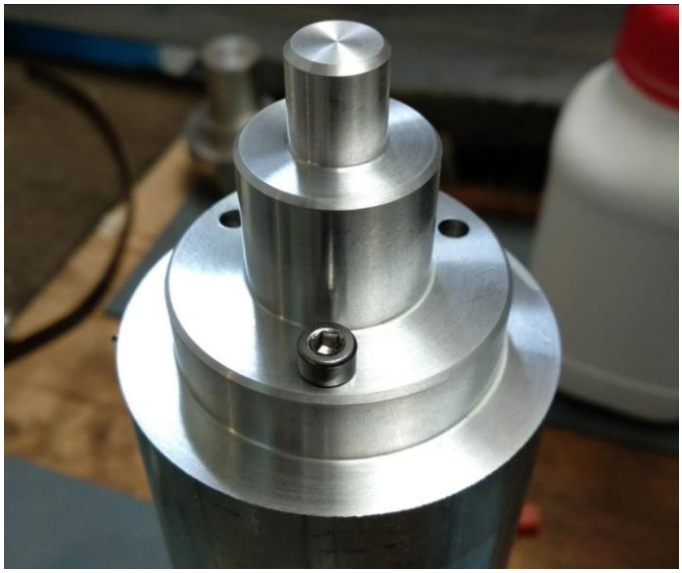


Figura 7 –Pieza de aluminio de la base

Anexo 2 – Conexiones y esquemas

2.1 – Adafruit Motor HAT

Aunque esta placa fue descartada, se llegó a conectar para probar si era válida en nuestro sistema. Se recopiló la siguiente información:

Según el fabricante, el esquema de la placa es el siguiente:

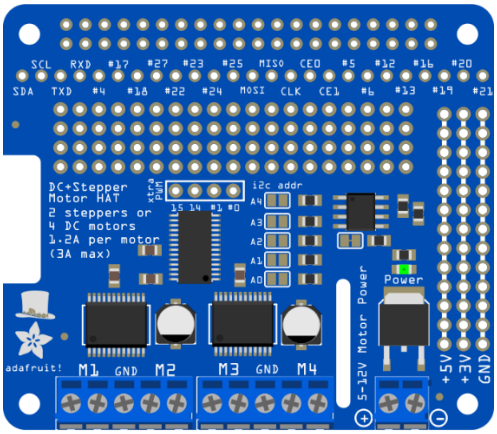


Figura 6 – Esquema del Adafruit Motor HAT

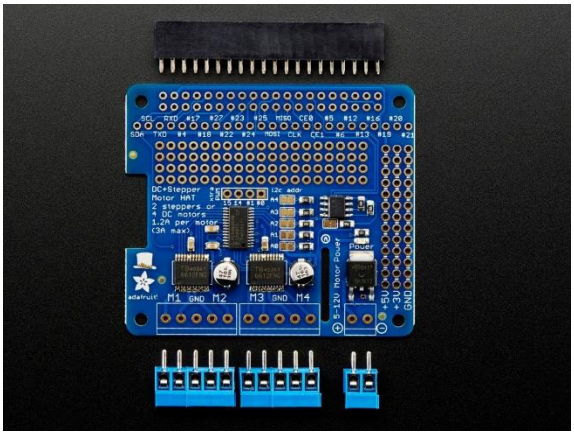


Figura 7 – Foto promocional del producto

Como podemos observar, en la parte superior de la placa tenemos el header de conexión con la Raspberry Pi, y en la parte inferior los conectores para los motores. Cada motor paso a paso utiliza dos conectores de motores DC (uno por cada bobina), por lo que conectaremos el motor de la base a los pines M1 y M2, y el motor del soporte a los pines M3 y M4.

Como vemos en la figura 8, el Motor HAT se vende con los conectores sin soldar, por lo que el primer paso necesario fue soldarlos. Después de hacerlo, la placa quedó de la siguiente manera:

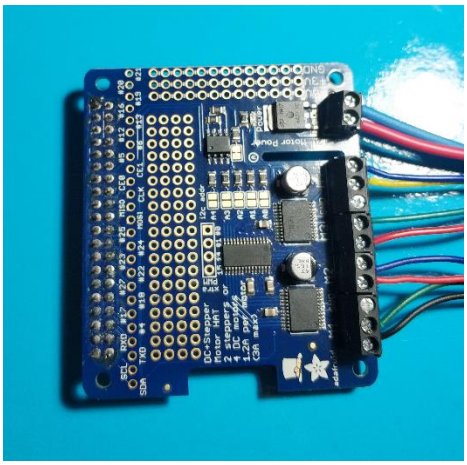


Figura 8 – Motor HAT con conectores soldados y conexiones de los motores

Con el Motor Hat, las conexiones fueron las siguientes:

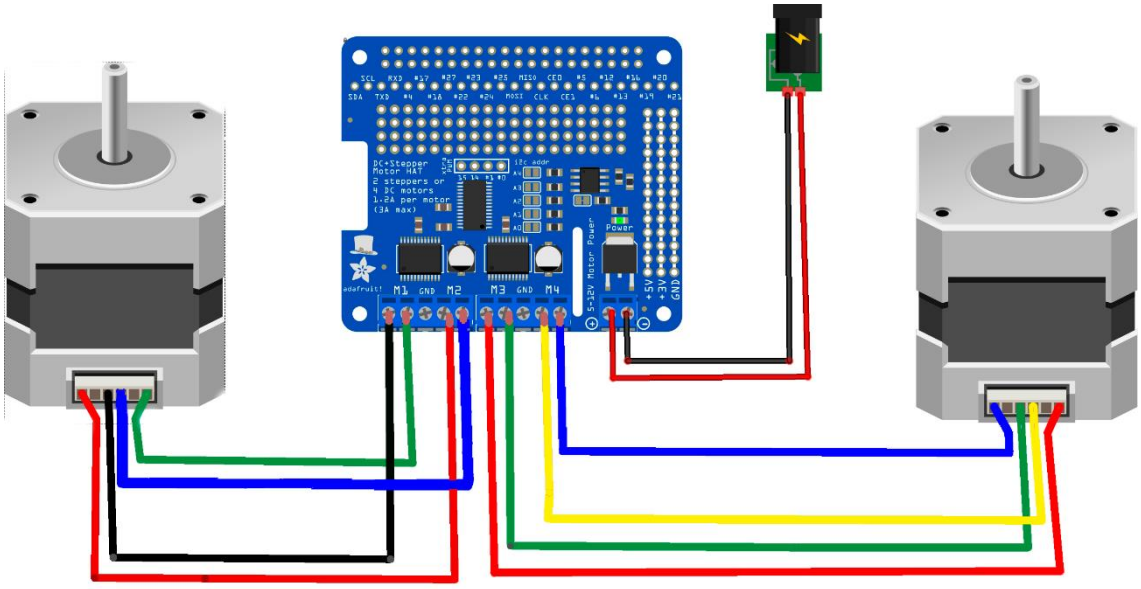


Figura 13 – Esquema de conexiones de los motores

2.2 – L928N

Otros de los controladores probados. Admite hasta 2 amperios de intensidad, y nos permite controlar hasta 2 motores de tipo CC o un motor paso a paso. Es muy utilizado en proyectos con Arduino debido a su coste y facilidad de uso.

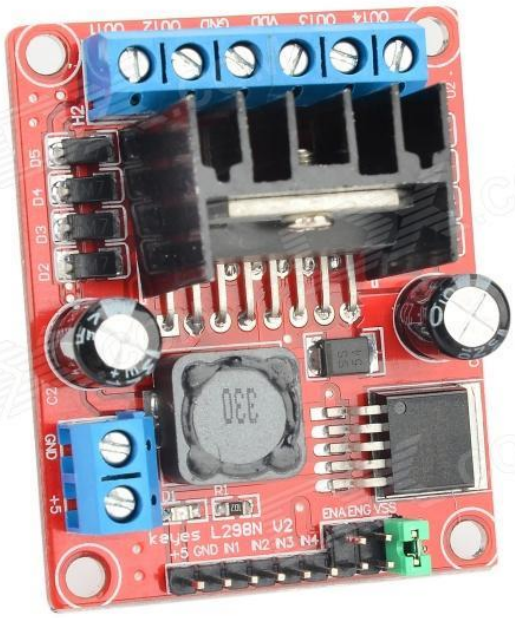


Figura 14 – Driver L298N - Keyes V2

Este motor cuenta con 4 entradas de control, dos señales de habilitación para los motores, y una conexión para habilitar o no la alimentación de la circuitería a partir de los motores, así como una entrada para una fuente externa, cuatro pines de salida para los motores y una entrada de 5 voltios, por si se desea alimentar el circuito de forma separada.

2.3 - A4988

El controlador finalmente utilizado. Tiene un chopper de tensión , asi que admite tensiones de entre 8 y 35V, y es capaz de entregar aproximadamente 1A por fase sin un o venitlación extra. Con suficiente refrigeración puede entregar hasta 2A por bobina. Nuestros motores consumen 1.7A, así que este driver es compatible.

Si hablamos de precisión, el A4988 admite hasta 1/16 pasos, es decir, es capaz de dividir un paso de 1.8º en 16 partes, lo que nos ayuda a reducir muchísimo la vibración ocasionada al girar.

Figura 15 – Driver A4988

Este es el esquema de las conexiones utilizadas:

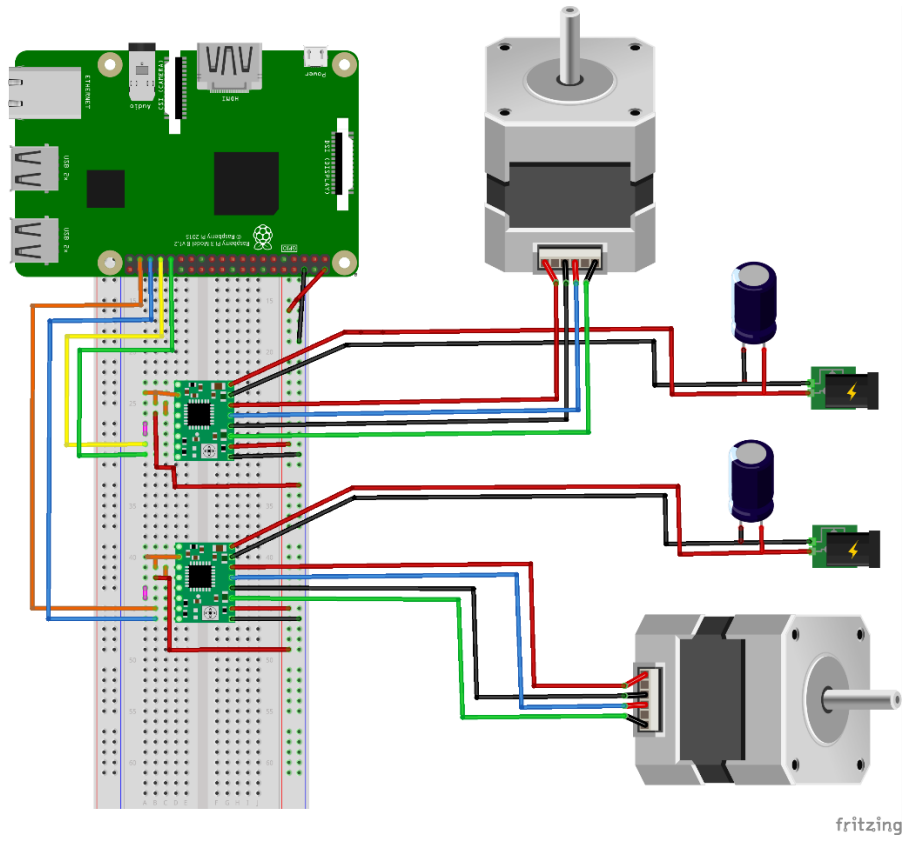
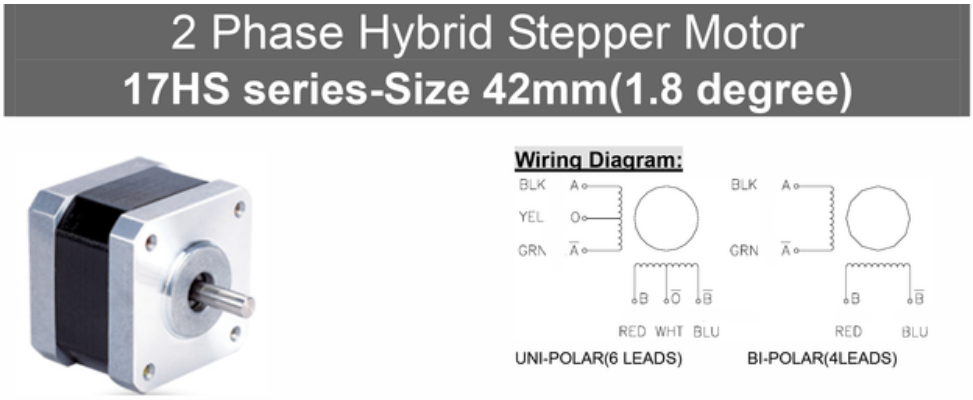


Figura 16 – Conexiones de los drivers A4988

2.3 – Motores

En este proyecto fueron utilizados dos motores idénticos, ambos paso a paso y de tipo Nema17 (es decir, con un frontal de 1.7 por 1.7 pulgadas)..

En vista de que el motor escogido inicialmente (17HS2048) era insuficiente, se pasó al modelo 17HS4401, bipolar, que consume hasta 1.7 amperios y es capaz de cargar con 4.28 kg/cm.



Electrical Specifications:

Series Model	Step Angle (deg)	Motor Length (mm)	Rated Current (A)	Phase Resistance (ohm)	Phase Inductance (mH)	Holding Torque (N.cm Min)	Detent Torque (N.cm Max)	Rotor Inertia (g.cm²)	Lead Wire (No.)	Motor Weight (g)
17HS4401	1.8	40	1.5	1.5	2.8	40	2.2	54	4	280

Figura 17 – Tabla de especificaciones del motor 17HS4401

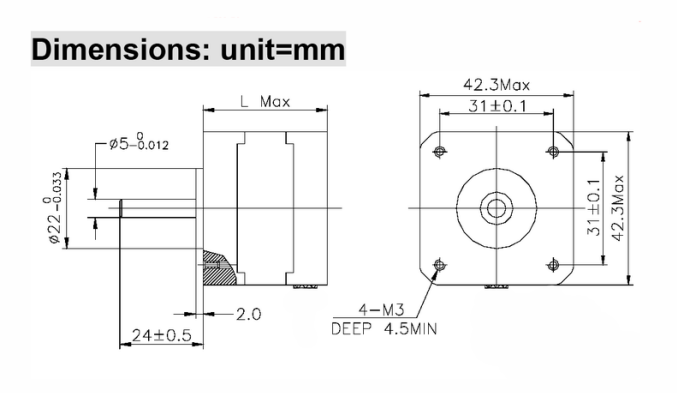


Figura 18 – Esquema con medidas del motor 17HS4401

Anexo 3 - Características técnicas

En este apartado desglosaremos las características técnicas de los componentes electrónicos utilizados:

3.1- Raspberry Pi 3

La Raspberry Pi 3 es la tercera generación de las placas Raspberry Pi. Se trata de un computador de placa única, del tamaño de una tarjeta de crédito. Se trata del primer modelo de la serie que incluye wifi y bluetooth integrados.

Figura 19 – Dibujo con los componentes de la Raspberry Pi 3

Sus especificaciones son las siguientes:

Specifications	
Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A
Connectors:	
Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Figura 20 – Especificaciones de la Raspberry Pi 3

3.2 – Adafruit Motor HAT

Este add-on para Raspberry Pi está fabricado por la empresa Adafruit, y tiene las siguientes características:

- **4 H-Bridges:** El chipset TB6612 proprociona 1.2ª por bridge con protección de apagado térmico y diodos internos de protección. Mueve mover motores de 4.5 VDC a 13.5 VDC.
- **Permite utilizar hasta 4 motores CC bidireccionales,** con elección individual de velocidad de 8 bit, lo que se traduce en aproximadamente el 0.5% de resolución.
- **Permite usar dos motores paso a paso** (unipolares o bipolares) con pasos en los que se emplea una sola bobina, las dos, ambas intercaladas o micropasos.
- **Viene con grandes terminales de conexión,** para facilitar la conexión de los cables.
- **Terminal de dos pines para alimentación** con polaridad protegida para conectar una alimentación externa de entre 5 y 12 VDC.

3.3 – Driver L298N

Es uno de los drivers más comunes en aplicaciones de robótica y Arduino. Tiene las siguientes características:

- Corriente pico de operación: 4A.
- Corriente constante de operación: 2A.
- Bajo voltaje de saturación en los transistores de salida.
- Corte de operación por sobrecalentamiento.
- Voltaje de alimentación de hasta 46V.
- Buena inmunidad ante el ruido.
- Ideal para controlar motores en robótica.

3.4 –A4988

Es un driver de motor paso a paso utilizado de forma habitual en impresoras 3D y máquinas CNC, debido a su precio y facilidad de uso. Sus características técnicas son las siguientes:

General specifications

Minimum operating voltage:	8 V
Maximum operating voltage:	35 V
Continuous current per phase:	1 A[2]
Maximum current per phase:	2 A[3]
Minimum logic voltage:	3 V
Maximum logic voltage:	5.5 V
Microstep resolutions:	full, 1/2, 1/4, 1/8, and 1/16
Reverse voltage protection?:	N
Bulk packaged?:	N
Header pins soldered?:	N[4]

Dimensions

Size:	0.6″ × 0.8″
Weight:	1.3 g[1]

Notes:

1 Without included optional headers.
2 Without a heat sink or forced air flow.
3 With sufficient additional cooling.
4 Male header pins are included for the board's 16 holes, but they are not installed.

3.5 – Logitech C270

Webcam fabricada por la empresa Logitech. De su web se extrajeron las siguientes especificaciones:

Camera Specifications:	
Available Image(s)	[Left Side Image]
Connection Type	Corded USB
USB Type	High Speed USB 2.0
USB VID_PID	VID_046D&PID_081A
Microphone	Built-in, Noise Supression
Lens and Sensor Type	Plastic
Focus Type	Fixed
Field of View (FOV)	60°
Focal Length	4.0 mm
Optical Resolution (True)	1280 x 960 1.2MP
Image Capture (4:3 SD)	320x240, 640x480 1.2 MP, 3.0 MP
Image Capture (16:9 W)	360p, 480p, 720p
Video Capture (4:3 SD)	320x240, 640x480, 800x600
Video Capture (16:9 W)	360p, 480p, 720p,
Frame Rate (max)	30fps @ 640x480
Video Effects (VFX)	N/A
Right Light	Right Light 2
Buttons	Other NA
Indicator Lights (LED)	Activity/Power
Privacy Shade	No
Clip Size (max)	0 to infinity
Cable Length	5 Feet or 1.5 Meters

Figura 21 – Especificaciones de la cámara Logitech C270

3.6 – Fuente de alimentación para Raspberry Pi

Es la fuente de alimentación oficial para la Raspberry Pi3, recomendada por el fabricante. Estas son sus especificaciones:

- **Salida:** 5.1 voltios, 2.5 amperios.
- **Cabezales:** Diferentes, para que pueda ser utilizada en cualquier país.
- **Cable:** 1.5 metros de largo.
- **Tipo de cable:** Micro-usb.

3.7 – Fuente de alimentación variable

Dos fuentes de idénticas prestaciones, con varios cabezales y capaces de proporcionar entre 4.5 y 12 voltios, y 2.5 amperios.

Anexo 4 – Requisitos de instalación

Para poder ejecutar el código de este proyecto, será necesario que se cumplan los siguientes requisitos:

4.1 - Sistema

Este proyecto está diseñado para funcionar en una Raspberry Pi 3B con sistema operativo Raspbian Stretch.

Raspbian está basado en Debian, por lo que debería ser compatible con cualquier distribución Linux con la que comparta base, o en cualquier otro modelo de Raspberry Pi, aunque sólo ha sido testado en ese sistema operativo y con ese modelo de Raspberry.

4.2- Lenguaje

El programa debe ser ejecutado con Python 2.7.

4.3 - Librerías

El programa, además de las librerías propias de Python, hace uso de algunas librerías externas:

- **OpenCV 3.1.0 –** Librería para visión por computador.
- **Imutils –** Librería que facilita la realización de operaciones sencillas con imágenes.
- **Numpy –** Librería que añade un mayor soporte para vectores y matrices, facilitando así las operaciones matemáticas.

4.4 - Instalación

Una vez tenemos las librerías necesarias y sus respectivas dependencias, para descargarnos el código fuente y poder ejecutar el proyecto, bastará con descargar el repositorio y descomprimirlo en una carpeta de nuestro sistema. La forma más sencilla es utilizando Git:

```
git clone https://github.com/mrivaj/sentry-gun.git
```

Es necesario decir que el proceso de instalación de OpenCV es bastante complejo. Si utilizamos los cuatro núcleos de la Raspberry Pi es probable que nos falle a mitad de la instalación, y si utilizamos un solo núcleo es un proceso que puede tardar unas cinco horas.

Hay usuarios que se han encontrado con el mismo problema, lo que los ha llevado a elaborar scripts que faciliten la instalación, que luego han compartido públicamente para hacer más sencilla la instalación al resto de los usuarios. Un ejemplo de ello sería esta publicación de Pedro Henrique Fonseca en la web Hackster.io.

Anexo 5 – Manual de usuario

En este apartado se reproducirá el manual de usuario del proyecto, disponible en el repositorio donde se aloja el proyecto (en formato `Markdown`).

Raspberry Pi Sentry Gun

Manual de usuario

1. Descarga

Para descargar este proyecto bastará con clonar el repositorio

```
$ git clone https://github.com/mrivaj/sentry-gun
$ cd sentry gun
```

2. Comprobaciones

En el repositorio que hemos descargado anteriormente podemos ver una carpeta "tests". En ella, están disponibles algunos pequeños programas para comprobar que los motores funcionan correctamente:

- `pan_range_test.py` , `tilt_range_test.py` : Realiza una serie de movimientos con los motores para comprobar que todo funciona correctamente

3. Calibración

Para que el programa funcione correctamente, es necesario calibrar primero ambos motores. para ello, en la carpeta `calibrate` tenemos los siguientes archivos:

- `calibrate_pan.py` , `calibrate_tilt.py` , `full_calibration.py` : Nos permiten calibrar la posición inicial del sistema utilizando las flechas del teclado
- `usb_camera_target:` Abre una ventana en la que podremos ver la webcam con el centro señalizado mediante un punto, para así facilitar la calibración

4. Ejecución

Una vez calibrado, debemos ejecutar el programa principal

```
$ python sentry_gun.py
```

Durante la ejecución se nos mostrarán mensajes informativos por consola. Una vez esté la cámara lista, se nos abrirán tres ventanas:

- **Cámara**: Nos muestra el streaming de vídeo de la cámara
- **Umbralizado**: Aquí podemos ver la imagen de la cámara despues de haber pasado por un algoritmo de valor umbral, lo que la transforma en una imagen blanco/negro
- **Frame Delta**: Esta ventana nos muestra el "frame acumulativo", es decir, las diferencias que aparecen entre el frame de referencia y la imagen actual

Una vez se detecta un objetivo, se procede a marcarlo en la imagen, así como a señalar su centro. Con los datos de la posición (x,y) de dicho centro, se calcula el número de pasos equivalente, y se mueve el motor para que apunte al objetivo.

5. Salida del programa

Lo ideal es cerrar el programa con la tecla de salida (Por defecto, la tecla `q`). No obstante, como usamos un bloque `try-catch` , el programa siempre se asegurará de liberar correctamente los recursos utilizados