UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Implementación de un sistema de detección y seguimiento de robots móviles basado en procesamiento de imágenes

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Pérez Martínez, David

Tutor/a: González Sorribes, Antonio

CURSO ACADÉMICO: 2022/2023

RESUMEN

El objetivo del presente proyecto consiste en el diseño de una solución para el seguimiento de un objetivo mediante un robot móvil en un entorno controlado utilizando técnicas de visión por computadora. El sistema se compondrá principalmente de un robot en condición de seguidor, y un objetivo que puede moverse en el escenario. Además, una cámara colocada en posición cenital transmitirá imágenes del escenario sobre las que se extraerá en tiempo real la localización de ambos elementos utilizando técnicas de procesamiento de imágenes, comunicando la posición al robot para llevar a cabo el seguimiento de forma precisa.

AGRADECIMIENTOS

Una vez acabada la realización de este trabajo, me gustaría aprovechar para dar crédito a la gente que ha estado ahí apoyándome y dándome ánimo cuando no veía la forma de sacar todo esto adelante.

En primer lugar, me gustaría darle las gracias a mi familia, a todos ellos, a mi padre por ayudarme en todo lo que ha podido para que este trabajo saliera lo mejor posible; a mi madre, por tener la paciencia necesaria conmigo y no dejarme que me viniera abajo; y a mi hermano, que me ha dado la experiencia necesaria a la hora de enfrentarme a un trabajo como este.

Por supuesto, también tengo que agradecerles a mis amigos de la carrera, gente que ha estado ahí en los buenos y en los malos momentos, que me han sacado una sonrisa siempre y con los que he compartido experiencias y recuerdos que no voy a olvidar nunca.

Gracias a la Universidad Politécnica y a sus profesores, por formarme, transmitirme conocimientos y ayudarme a conocer a los que ahora son mis amigos y compañeros.

Y, por último, estoy yo, que he puesto mi esfuerzo y mis ganas en todo momento para conseguir los mejores resultados posibles, afrontando los problemas, y siempre intentando ver el lado bueno de las cosas. Ha sido un camino largo, con risas y agobios, pero yo decidí no rendirme y seguir adelante. Gracias David.

David Pérez

Valencia,2023

ÍNDICE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO DE FIN DE GRADO

DOCUMENTO N.º 1. MEMORIA Y ANEXOS

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Objeto

El presente proyecto consiste en el diseño de una solución para el seguimiento de un objetivo utilizando un robot móvil en un entorno controlado aplicando técnicas de visión por computador. El proyecto se compone principalmente de un robot móvil de tipo diferencial que actuará como seguidor y un objetivo móvil que tendrá la función de guía.

El dispositivo de visión se encargará de transmitir una imagen en tiempo real y, mediante la utilización de técnicas de procesamiento de imágenes, se obtendrán posteriormente la localización y orientación del robot, además de la posición del punto objetivo y los parámetros necesarios para llevar a cabo el seguimiento de forma precisa.

Finalmente, teniendo en cuenta el objetivo del proyecto, se deberán tratar puntos tales como la selección de los componentes electrónicos a utilizar, la programación necesaria en los diferentes ámbitos de la aplicación, el montaje de la maqueta y la realización de pruebas para la comprobación del funcionamiento.

2 Antecedentes

2.1 Breve introducción a la robótica

La robótica es una disciplina que comprende ramas como la mecánica, electrónica, eléctrica y las ciencias de la computación. El conjunto y unificación de todos estos ámbitos permiten el diseño y la construcción de sistemas móviles autónomos con numerosas aplicaciones en la actualidad, a los que se les conocen como robots.

La historia de la robótica se puede remontar desde tiempos anteriores al sigo I a.C, donde ya se podían visualizar las primeras máquinas sin muchos medios para construirlas. Años después, personalidades como Leonardo Da Vinci con su diseño de un robot humanoide (1945) o el francés Jacques de Vaucanson con su pato mecánico (1739) realizarían avances importantes en el ámbito de la robótica.

Pero fue en el siglo XX donde la robótica comenzó un proceso de crecimiento notorio y donde se sentaron las bases de la llamada robótica moderna (*Fig. 1*). Durante la Segunda Guerra Mundial, se desarrollaron los primeros robots industriales, capaces de realizar tareas peligrosas o monótonas en las fábricas. Pero uno de los acontecimientos más importantes desemboca en la invención de Unimate, el primer robot industrial controlado por computadora y desarrollado por George Devol y Joseph Engelberger en 1960.

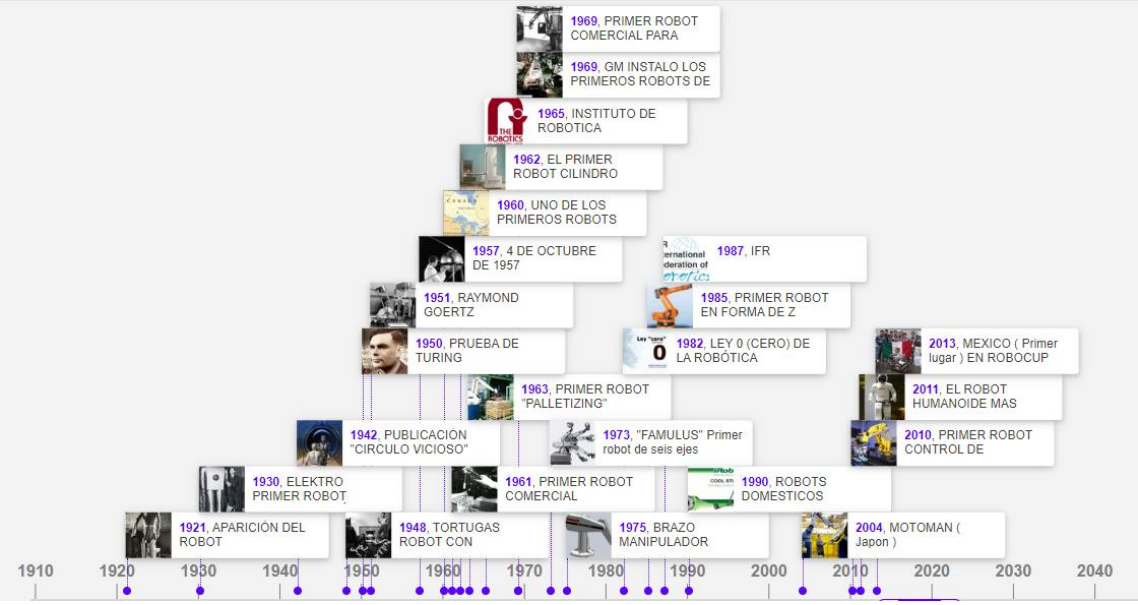


Figura 1. Evolución de la robótica durante el siglo XX.

Con el avance de las últimas décadas del siglo XX, la tecnología en el entorno de la robótica realizó un proceso de mejora, hasta que alcanzó su punto máximo en los años 80 iniciando la conocida Era Robótica, con un aumento en su fabricación y ventas de un 80% y con numerosos países sumándose a esta nueva etapa.

Finalmente, se implementó lo que se conoce hoy en día como la Robótica Inteligente, fusionando los conceptos de la robótica con la sensorización, la percepción del entorno y la anticipación y adaptación a situaciones imprevistas, introduciendo robots en un gran número de aplicaciones en la actualidad.

2.2 Robótica móvil

La robótica móvil se puede definir como el conjunto de sistemas robóticos con la capacidad de desplazarse por el entorno que les rodea de manera completamente autónoma con el objetivo de cumplir tareas de cierta complejidad.

A diferencia de la robótica tradicional, centrada principalmente en un patrón predeterminado, la robótica móvil implementa en gran medida una variedad de sensores (cámaras, ultrasonidos, infrarrojos) que, en combinación con su programación orientada a tareas, permiten llevar a cabo el planteamiento y ejecución de movimientos de forma independiente.

Además, cabe destacar el notable avance de este sector en las últimas décadas, principalmente originado por el progreso de la miniaturización de componentes electrónicos, la computación embebida y la inteligencia artificial. De esta forma, se han podido crear robots con un tamaño más reducido, más ágiles y con algoritmos de toma de decisiones mucho más desarrollados para las aplicaciones actuales.

Por otra parte, existen diferentes tipos de robots móviles, entre los cuales podemos destacar principalmente los robots terrestres, acuáticos y aéreos (*Fig. 2*).

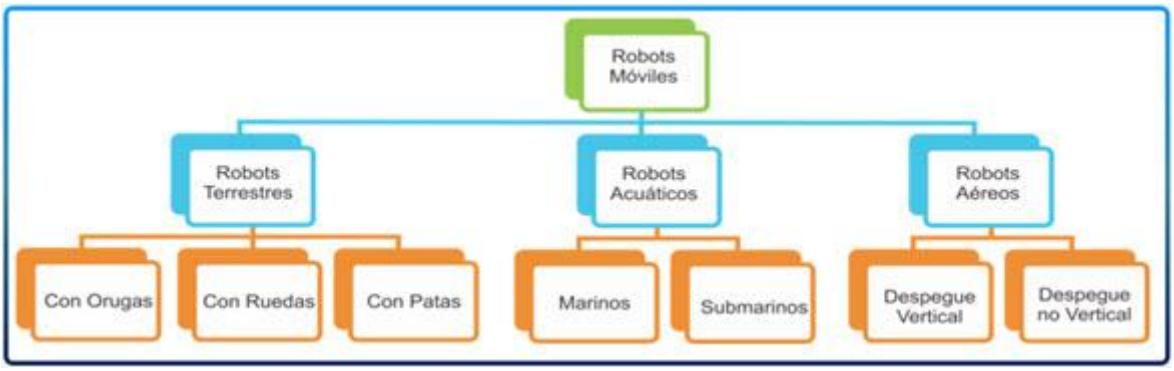


Figura 2. Clasificación de los robots móviles.

2.2.1 Robots móviles terrestres

Como su propio nombre indica, estos robots se desplazan por la superficie terrestre y pueden ser de diferentes formas y tamaños. Utilizados en la actualidad para innumerables aplicaciones, son capaces de llevar a cabo funciones militares y de exploración, captar información de su entorno, realizar operaciones de rescate o incluso aplicaciones en área médica como asistentes.

Dentro de esta clase de robot, destacan principalmente los **robots con ruedas**. Su componente de locomoción le permite desplazarse por distintos terrenos, alcanzando diferentes velocidades, pero con grandes desventajas frente a superficies accidentadas. Dentro de este grupo, se pueden encontrar diferentes clasificaciones teniendo en cuenta tanto el tipo de ruedas utilizadas como la configuración y posición de estas.

Respecto a los tipos de ruedas (*Fig. 3*), se pueden clasificar en:

- **Ruedas convencionales**: Son las ruedas más comunes y las que se observan mayoritariamente en las diferentes aplicaciones. Se caracterizan principalmente por su estructura circular y un sistema de rodadura que permite el movimiento de tracción. Normalmente este tipo de ruedas necesitan estar conectadas a un elemento mecánico como son los motores. Dentro de este grupo, se pueden destacar la rueda fija (orientación única y sin deslizamiento), la rueda orientable (orientación de la rueda variable) o la rueda orientable descentrada o de castor (orientación variable con el eje de rotación desplazado).
- **Ruedas no convencionales:** En este grupo se encuentran ruedas con características diferentes y que, por lo tanto, deben ser clasificadas en otra sección. Se pueden destacar ruedas como la rueda sueca u omnidireccional (capaz de desplazarse en el plano XY sin realizar maniobras) o la rueda loca (aseguran movimiento en todas direcciones).

Figura 3. Tipos de ruedas.

Por otra parte, es de gran importancia la configuración de las ruedas, es decir, el posicionamiento que estas tienen en el robot (*Fig. 4*). Algunas de las configuraciones más conocidas son:

- **Configuración Diferencial:** La tracción viene dada por un conjunto de dos ruedas que comparten eje de rotación. Normalmente dispone de una rueda loca extra para estabilizar el movimiento del robot.
- **Configuración Triciclo:** Está compuesto principalmente por dos ruedas fijas traseras y una rueda orientable normalmente delantera. El movimiento del robot consiste principalmente en la actuación sobre la orientación de la rueda delantera y su velocidad lineal de avance. Este tipo de configuración permite mayor adherencia y una mejor sensorización.
- **Configuración Ackerman:** Esta disposición de las ruedas en utilizada comúnmente en los coches convencionales. Se compone de 4 ruedas, dos de ellas libres (ruedas fijas) y las otras dos orientables, dando la dirección al sistema móvil. La estabilidad de esta configuración viene dada principalmente por el Principio de Ackerman, que enuncia que los ejes de las ruedas orientables deben concurrir en un mismo punto.

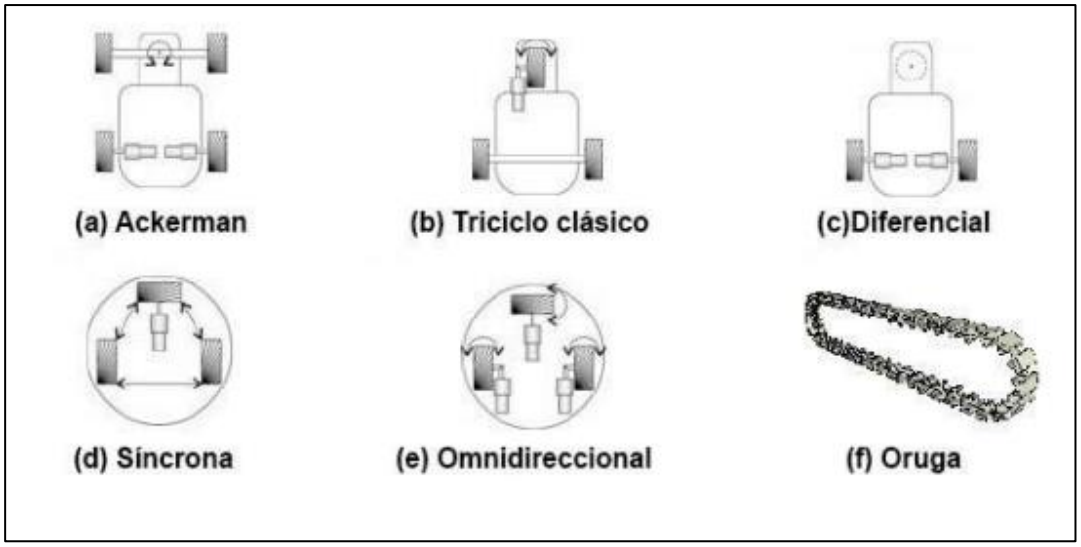


Figura 4. Configuración de los robots móviles terrestres.

Otra de los tipos de robots más comunes son los que disponen de una **configuración oruga**. En estos robots se utilizan dos cadenas laterales con ruedas en el interior (orugas). Esta configuración permite al robot desplazarse por superficies más inconsistentes, dotándolos de una mayor capacidad de tracción (aumento en los puntos de contacto con el terreno) en comparación a la configuración diferencial. Sin embargo, posee desventajas como la velocidad o el peligro del deslizamiento.

Finalmente, hay que destacar los **robots móviles terrestres con extremidades** o patas, basados principalmente en comportamientos humanos o animales. Su versatilidad les permite realizar movimientos que ni las ruedas ni las orugas serían capaces de llevar a cabo, sin embargo, su control de estabilidad y su diseño es de alta complejidad. Su clasificación depende de factores como el número y la disposición de las extremidades, y que, por lo tanto, determinaran la aplicación del robot.

2.2.2 Robots móviles acuáticos.

Los robots móviles acuáticos se caracterizan por su adaptación y versatilidad en aplicaciones relacionados con entornos acuáticos, ya sean océanos, ríos, lagos u otras masas grandes de agua.

Los robots acuáticos se pueden clasificar principalmente por su autonomía, el tipo de misión a realizar o la propulsión del sistema.

- **Autonomía:** Esta es una característica esencial de los robots, ya que establece la capacidad del robot de tomar decisiones por sí mismo y, por lo tanto, no necesita la acción humana para realizar su función. Se distinguen principalmente los vehículos submarinos autónomos (AUV), que, equipados con sensores y sistemas de navegación, no necesitan acción humana para funcionar; y los robots operados de forma remota (ROV), que son controlados por alguien externo.
- **Tipo de Misión:** El objetivo del robot es de gran importancia a la hora de realizar su diseño, ya que dependiendo de la función u aplicación en la que va a participar, se decidirán los sensores y actuadores a instalar. Las tareas más usuales para este tipo de robots son misiones de inspección, donde el robot realiza las acciones durante la navegación; y las misiones de manipulación (*Fig. 5*), donde aparece la necesidad de disponer de mecanismos de manipulación como brazos robotizados.
- **Sistema de propulsión:** Define los movimientos y las acciones que va a poder realizar el robot. Existen números mecanismos para la propulsión, entre ellos se pueden destacar los Impulsores por Hélice, los Planeadores Acuáticos y los Bioinspirados.

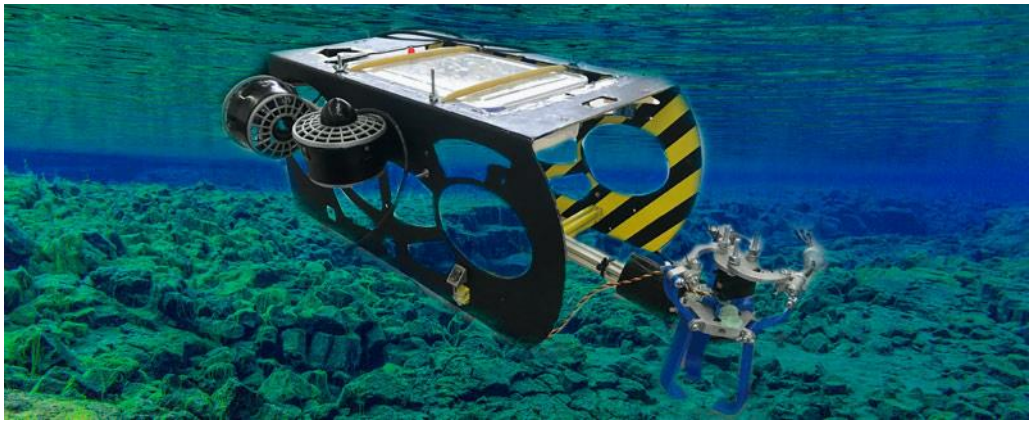


Figura 5. Robot acuático con brazo manipulador

Algunas de las aplicaciones principales de esta categoría de robots móviles serían la observación del lecho marino y la toma de muestras de agua para poder realizar análisis posteriores. Además, con la disponibilidad de herramientas de manipulación, pueden llevar a cabo tareas de mantenimiento en las profundidades o facilitar de manera notable la instalación de dispositivos subacuáticos, permitiendo que aplicaciones con difícil acceso humano se puedan realizar sin elevada complejidad.

2.2.3 Robots móviles aéreos.

Los robots móviles aéreos son dispositivos capaces de desplazarse por el aire, alcanzando elevadas alturas difíciles de alcanzar para los seres humanos, para llevar a cabo numerosas aplicaciones (*Fig. 6*).

La diferenciación más importante a la hora de definir la robótica aérea es la autonomía, donde se diferencia ente los vehículos aéreos no tripulados (UAV), que no necesitan intervención humana; y los RPA ("Remotely Piloted Aircraft"), donde el humano controla el vehículo aéreo.

Las clasificaciones más importantes en los robots aéreos son:

- **Aerodinámica:** Representa el método de sustentación del vehículo aéreo. Algunos ejemplos serían el ala fija (sustentación por diferencia de presión), batimiento de alas (visto en diferentes tipos de fauna) y el ala rotatoria (el giro de unas palas permite la sustentación, muy usado en los drones).

- **Despegue/Aterrizaje:** Se diferencian principalmente por el ángulo, vertical (VTOL) u horizontal (STOL o HTOL). Los UAV se caracterizan generalmente por un aterrizaje y despegue vertical.

- **Tipo:** Las aeronaves actuales se engloban en cuatro grandes clases, de tipo avión (ala fija), helicóptero (ala rotatoria), multirrotor (ala rotatoria) y tipo dirigible (impulsado con hélice y sustentado por aire caliente).

- **Peso:** Una de las propiedades más importantes para un robot aéreo, ya que es la fuerza principal que contrarresta a la sustentación de la aeronave. Se distinguen tres rangos de peso, que son 0-25 kg, 25-150 kg y mayor de 150 kg, y de esta propiedad dependerá los permisos necesarios para poder pilotarlas.

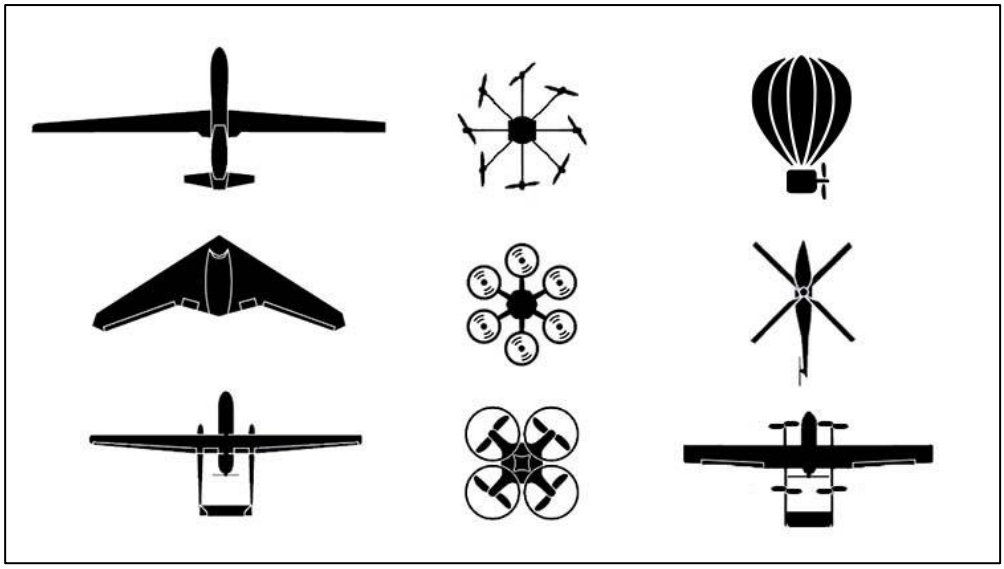


Figura 6. Diferentes ejemplos de vehículos aéreos.

Finalmente, hay que destacar algunas de las aplicaciones principales de este tipo de robots como podrían ser tareas de inspección y vigilancia en lugares de difícil acceso, fotografía y videografía aérea de alta calidad, o aplicaciones de otro tipo como entregas ligeras o trabajos de cartografía.

2.3 Introducción a la visión artificial

La visión artificial y el procesamiento de imágenes es una rama científica que tiene como principal objetivo la adquisición, el tratamiento y la comprensión de imágenes por los dispositivos informáticos. Está compuesta principalmente de algoritmos que tratan de representar la estructura y características del mundo tridimensional utilizando numerosas imágenes bidimensionales.

Esta importante disciplina de estudio se remonta a la década de 1960, donde se planteó por primera vez la posibilidad de que una máquina pudiera ver y comprender el entorno como un ser humano, utilizando una cámara con conexión al computador. Fue en 1959, donde un grupo de neurofisiólogos realizaron un sencillo experimento que consistía en mostrarle imágenes a un gato para analizar su comprensión de estas y su capacidad de procesarlas. El resultado daba a conocer que la primera respuesta del animal se centraba siempre en las líneas y bordes más acentuados, y que, por lo tanto, el procesamiento de imágenes debía enfocarse inicialmente en ese concepto.

A partir de este punto, la visión artificial comenzó un proceso de evolución donde se pueden destacar hitos como la aparición de la primera tecnología capaz de escanear artificialmente imágenes o la consecución por parte de las máquinas de transformar imágenes de tres dimensiones en imágenes planas o de dos dimensiones (1963). Cabe destacar también la aparición en 1960 de la inteligencia artificial y su implementación en el campo del procesamiento de imágenes para aplicaciones de visión artificial.

Ya en el siglo XXI (*Fig. 7*), los estudios en el campo de visión se orientaron principalmente en el reconocimiento de objetos (2000) y en la creación de las primeras aplicaciones capaces de detectar los rasgos faciales en tiempo real (2001). También aparecieron y se popularizaron las redes neuronales de convolución (2010), que destacaron por su increíble rendimiento en aplicaciones como la detección y clasificación de objetos.

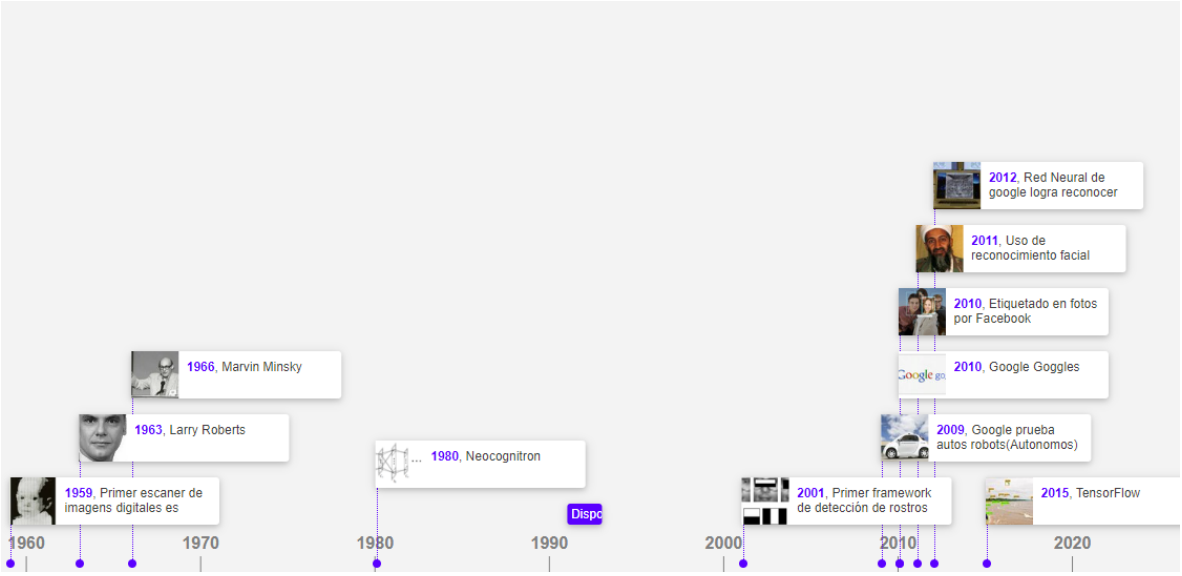


Figura 7. Línea temporal de la evolución de la visión artificial.

Los componentes de un sistema de visión artificial actual son la iluminación, elemento de gran importancia a la hora de la detección de objetos; la cámara, que debe ser capaz de transmitir luz al sensor y captar la imagen de manera enfocada; y el sistema de procesamiento, que contiene los algoritmos y funciones que permitirán realizar los análisis necesarios.

Respecto de las numerosas aplicaciones actuales de la visión artificial, se pueden destacar:

- **Robótica Industrial:** El seguimiento, localización y detección de objetos y piezas es una parte importante en aplicaciones dirigidas al sector industrial, permitiendo el reconocimiento de características físicas de estos y su clasificación y manipulación mediante robótica colaborativa (*Fig. 8*).
- **Industria automotriz:** La implementación de la visión artificial en los vehículos autónomos ha ganado mucha importancia en los últimos años, permitiendo la detección de peatones y señales de tráfico, o la conducción autónoma por visión de carreteras.
- **Seguridad:** La utilización de cámaras y sistemas de visión artificial ha permitido mejorar el seguimiento de personas y la identificación de comportamientos fuera de lo normal.
- **Medicina:** Un sector de gran importancia donde los avances pueden mejorar de manera notable la vida de la población. El procesamiento de imágenes se ha visto introducido mediante aplicaciones de detección de anomalías en imágenes relacionadas con pruebas o medicamentos, o la identificación de enfermedades tempranas.

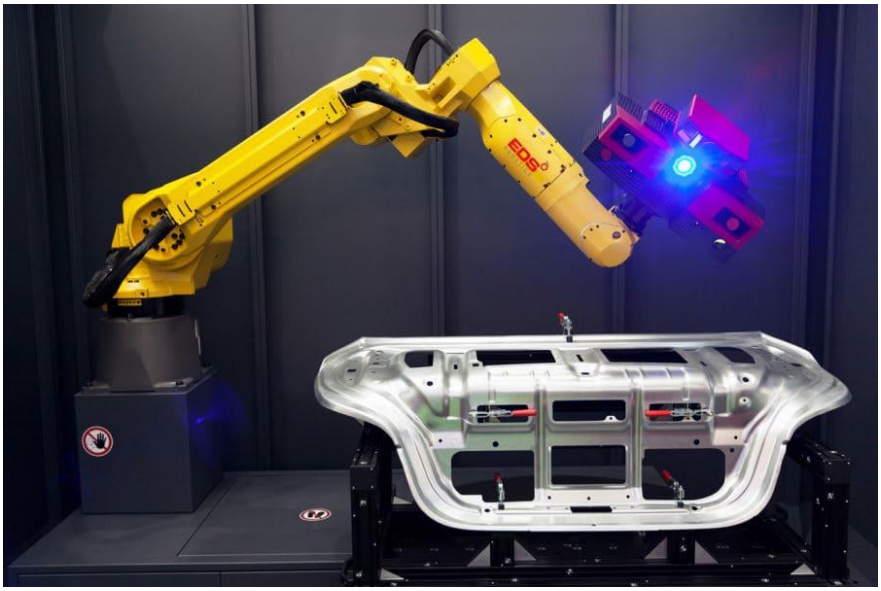


Figura 8: Visión artificial en conjunto a la robótica

En conclusión, la visión artificial ha experimentado una notable evolución a lo largo del tiempo, obteniendo numerosos beneficios en un gran número de sectores y aplicaciones, y con un amplio rango de mejora para brindar soluciones innovadoras y mejorar la sociedad actual.

3 Estudio de necesidades

En la aplicación que se va a desarrollar, se pretende unir conceptos de tres disciplinas estudiadas durante el grado: visión, robótica y control. De esta forma, para cada campo se deben tener en cuenta una serie de necesidades que tienen que ser estudiadas antes de comenzar la construcción del proyecto.

Primeramente, se van a concretar las necesidades en el campo de la **visión** y la detección. Esta es un parte de gran importancia para asegurar el correcto seguimiento del robot hacia el objetivo. Por ello, se deberán considerar:

Necesidades en el campo de la visión
• Diseñar y aplicar los algoritmos de procesamiento de imagen adecuados para la detección de los objetos de interés.
• Disposición del sistema de visión más adecuado y capacitado para transmitir imagen que pueda ser procesada en tiempo real.
• Estimar de manera precisa la posición y orientación del robot móvil, además de la posición del objetivo a seguir.
• Transmisión de los datos obtenidos al sistema de control del robot para poder llevar a cabo la aplicación.

Tabla 1.Necesidades funcionales en la parte de visión del proyecto.

Respecto del campo de la parte relacionada con la **robótica**, el enfoque principal reside en la construcción del robot que se va a utilizar en la aplicación. Por lo tanto, algunas necesidades son:

Necesidades en el campo de la robótica
• Realizar la selección de componentes más adecuada teniendo en cuenta criterios económicos, de tamaño, etc.
• Debe ser capaz de aplicar el sistema de control diseñado y llevarlo a cabo de forma precisa.
• Recepción de los datos provenientes del procesamiento de imagen de forma inalámbrica.

Tabla 2. Necesidades funcionales en la parte de robótica del proyecto.

La última parte del proyecto consiste en el diseño de la estructura de **control** adecuada, que permitirá que el robot siga la posición del objetivo a partir de los datos extraídos del apartado de visión. Se pueden destacar las necesidades siguientes:

Necesidades en el campo del control
• Diseñar la ley de control que permita el correcto seguimiento del robot ajustando las velocidades y la dirección.
• Implementar el controlador de la manera más robusta posible e intentando minimizar el error al máximo.
• Llevar a cabo el control teniendo en cuenta los actuadores de los que dispone el robot móvil (motores)

Tabla 3. Necesidades funcionales en la parte de control del proyecto.

Finalmente, cabe destacar la importancia de las **necesidades del entorno** en el que se van a llevar a cabo las pruebas de la aplicación. Principalmente se necesitará un espacio lo suficientemente grande como para que la aplicación pueda realizarse, además de una posición adecuada para la cámara, que deberá estar en posición cenital durante la ejecución de las pruebas.

4 Composición hardware de la aplicación

En este apartado se explicarán las diferentes partes en las que se puede dividir el hardware para la realización de la aplicación. Como se ha comentado anteriormente, el proyecto gira en torno a un sistema de visión y un robot móvil, el cuál debe ser capaz de seguir un objetivo que puede cambiar de posición.

Por lo tanto, la selección de los componentes es una parte de gran importancia para cumplir las especificaciones de forma adecuada. Se explicarán las decisiones que se hayan tomado en cada componente y algunas de las alternativas que se habían planteado antes de llegar a la solución final.

4.1 Organigrama

El objetivo principal de la realización de un organigrama consiste en facilitar la visualización de las diferentes secciones en las que se ha dividido el hardware de la aplicación. De esta forma, la segmentación que se ha realizado consta de tres partes destacadas: la electrónica, la estructura mecánica y la composición del escenario.

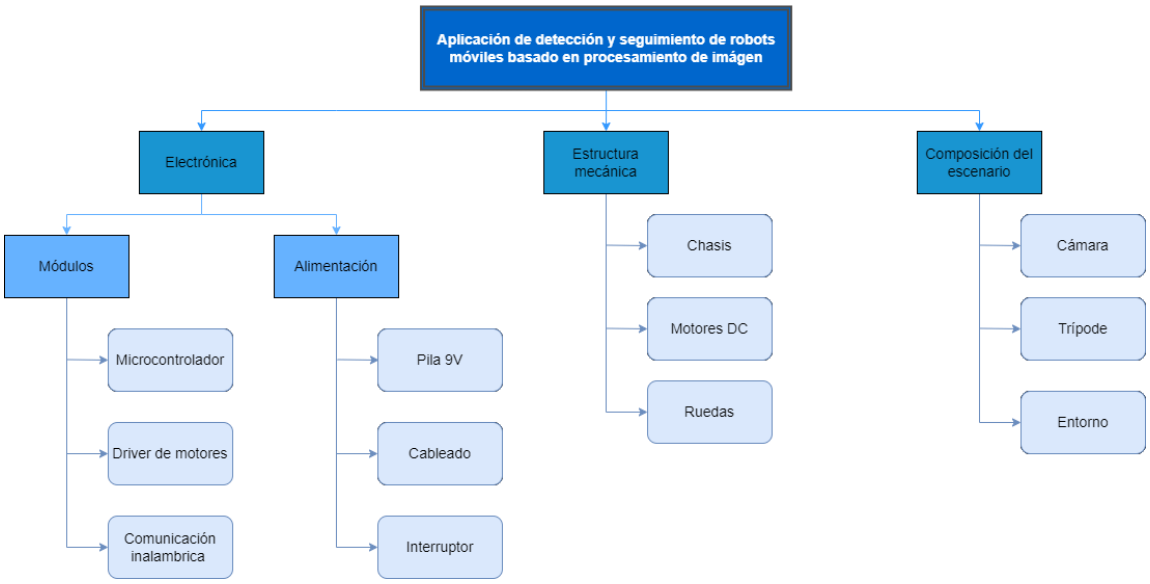


Figura 9. Organigrama del proyecto

Elaboración propia

Esta fragmentación, a su vez, incluirá los diferentes elementos o subsecciones que componen la aplicación y que serán explicadas en los posteriores apartados teniendo en cuenta la jerarquía vista en el esquema (*Fig. 9*).

4.2 Electrónica

En esta primera subsección se pretende englobar la electrónica utilizada principalmente en la parte de la robótica y su funcionamiento. Este apartado esta principalmente centrado en la alimentación de la que dispondrá el robot, además del sistema de control de este, que se explicará teniendo en cuenta los diferentes módulos de los que dispone.

4.2.1 Módulos

4.2.1.1 Microcontrolador: Arduino Nano

La primera decisión acerca de la composición del apartado de la robótica fue la elección del microcontrolador que se iba a utilizar para llevar a cabo el control del robot móvil. Para facilitar la implementación y el desarrollo del montaje, se decidió utilizar un kit de microcontrolador comercial. De esta forma, se escogió una opción de la familia Arduino, más concretamente la placa Arduino Nano (*Fig. 10*), con la que ya se había trabajado en otras aplicaciones y de la que ya se disponía con anterioridad a la realización del proyecto.

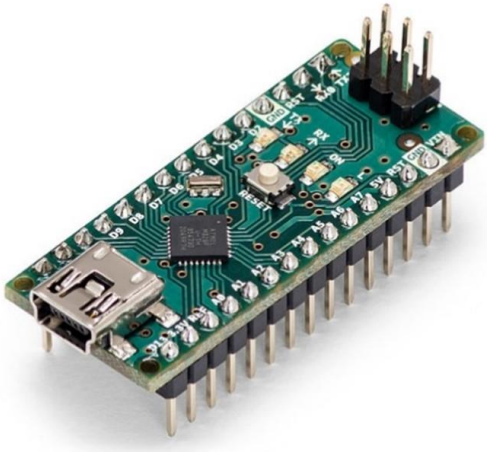


Figura 10. Arduino nano

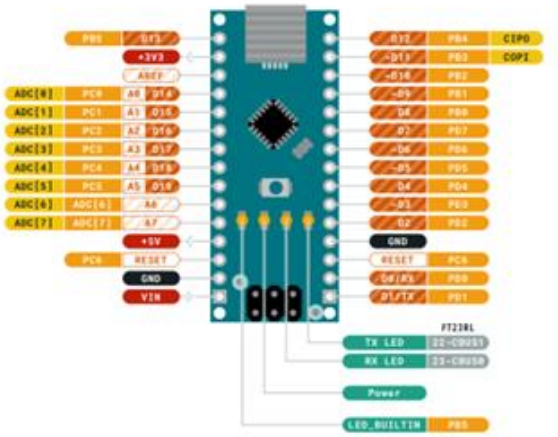


Figura 11. PinOut del Arduino Nano

Durante la búsqueda de posibles soluciones, se buscaba una opción que pudiera ser muy económica y fácil de usar y programar, para evitar perder tiempo en el desarrollo de la aplicación. Además, el tamaño también fue un criterio importante a la hora de seleccionar el microcontrolador que se iba a utilizar, ya que se buscaba una solución pequeña pero funcional para la implementación.

Respecto a sus características, se pueden destacar las siguientes:

- **Microcontrolador:** Utiliza un chip de Atmega328 (8 bits), que incorpora una velocidad de reloj de 16MHz.
- **Memoria:** Dispone de una memoria flash de 32 KB, que se utiliza para almacenar el programa, con 2 KB utilizados en el sistema de arranque. Por otra parte, dispone también de 1 KB de memoria EEPROM y 2 KB de SRAM.
- **Pines:** Como se ha podido ver en la *Fig. 11*, este módulo dispone de un gran número de pines, entre los cuales se diferencian 14 pines digitales, que pueden ser utilizados de entrada o salida, teniendo 6 de estos la posibilidad de ser utilizados como salidas PWM (muy útil para el control de los motores). También tiene 6 entradas analógicas.
- **Comunicación:** Dispone de una conexión mediante puerto USB para poder realizar la transmisión del programa o la impresión de datos por puerto serial.
- **Tamaño:** Sus dimensiones son reducidas en comparación con otras placas de la familia Arduino, con unas medidas de 4.5 cm x 1.8 cm.

Cabe destacar que junto al Arduino Nano se ha utilizado una shield (*Fig. 12*), que permite incrementar el número de pines de alimentación y masa para poder suministrar energía a todos los elementos del robot.

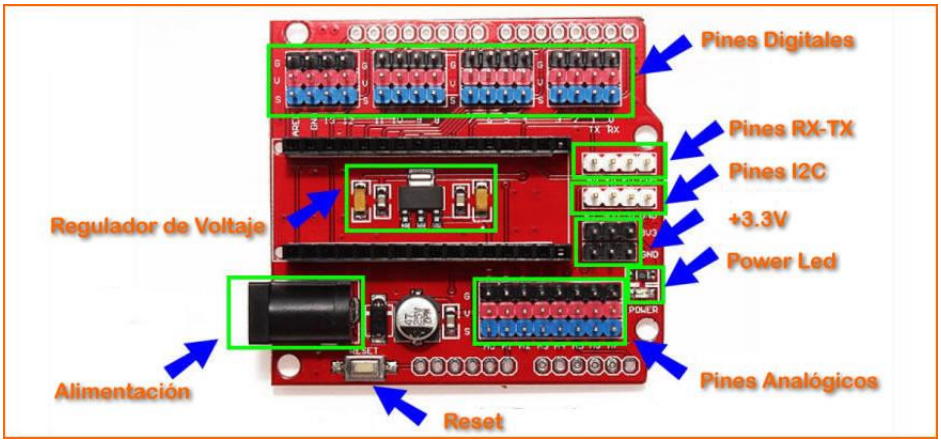


Figura 12. Shield de Arduino Nano con la ampliación de pines.

Una de las principales desventajas que se ha podido observar de esta solución sería la inexistencia de comunicación Bluetooth o Wifi propiamente incorporada en la placa. Sin embargo, se ha decidido implementar un módulo externo de comunicación Bluetooth (explicado más adelante en un apartado posterior), ya que, de esta forma, se evitaría comprar un nuevo microcontrolador.

4.2.1.2 Driver de motores DC: L298N

Para controlar los actuadores que se van a utilizar en el proyecto, que en este caso son motores de corriente continua, será necesario la utilización de un "driver" o controlador. El componente escogido corresponde al módulo L298N (*Fig. 13*).

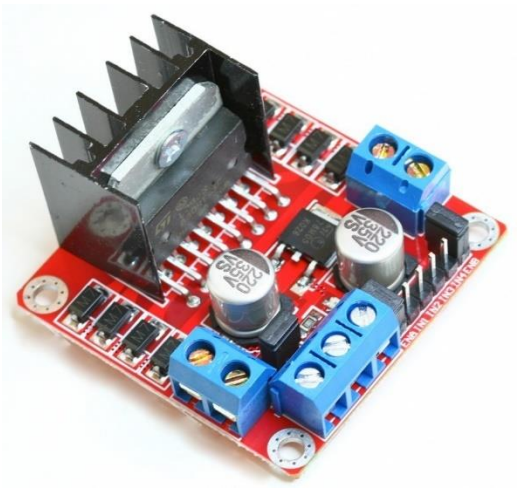


Figura 13. Controlador de motores DC L298N

La configuración principal de este chip se basa en un puente en H, un tipo de circuito muy utilizado en aplicaciones de robótica. Esta electrónica permite a un motor de continua invertir el giro, teniendo la posibilidad de avanzar o retroceder a voluntad del usuario. El nombre proviene de la representación del circuito (*Fig. 14*), que consta principalmente de cuatro interruptores que permiten que el motor gire en un sentido o en otro.

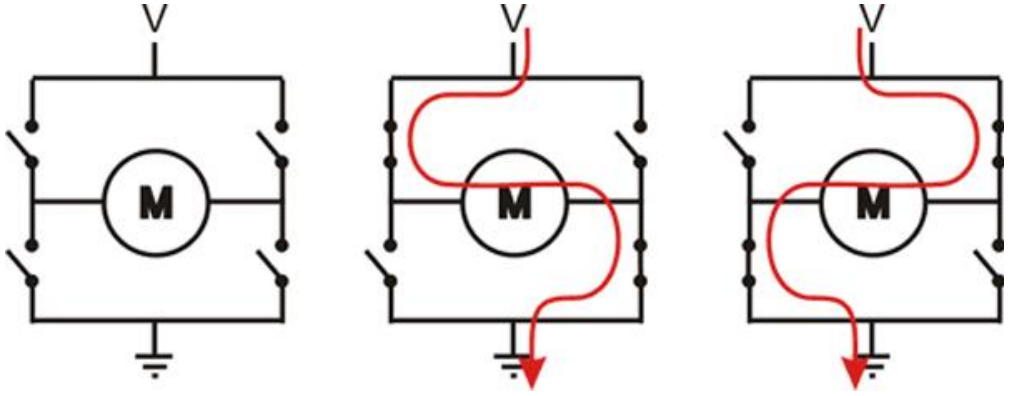


Figura 14. Puente en H y sus configuraciones.

Como se puede observar, se logra el movimiento del motor en ambos sentidos: si se accionan los interruptores 1 y 3, manteniendo los demás abiertos, el motor comenzará a girar en un sentido; y se invierte esta configuración, el giro del motor también lo hará.

Respecto a sus características principales, se pueden destacar:

- Control mediane señales TTL (se obtienen a partir de microcontroladores como Arduino). La velocidad de giro se regula mediante **modulación de ancho de pulso** (PWM).
- Ofrece un **voltaje de potencia** en un rango de 5–35 V, con varias entradas de alimentación y jumpers reguladores.
- **Tamaño:** Sus dimensiones se ajustan al espacio disponible en el robot, con unas medidas de 43 mm x 43 mm x 27 mm.

El modo de alimentación que se va a utilizar es el siguiente: se activará el jumper de selección de 5 V, por lo tanto, la entrada del controlador deberá ser de entre 6 V a 12 V, y contará con una salida de 5 V, que se utilizará para alimentar el microcontrolador y así el sistema completo estaría en funcionamiento.

4.2.1.3 Comunicación Bluetooth: SPP-C

La comunicación inalámbrica es necesaria a la hora de conectar el sistema de procesamiento de imagen y el sistema de control, ya que a partir de los diferentes parámetros tales como las posiciones del robot y del objetivo, o la orientación del robot, se deberá realizar el control pertinente. De esta forma, la opción más adecuada para la transmisión de datos fue un módulo SPP-C de comunicación Bluetooth (*Fig. 15*).

Figura 15. Módulo de comunicación Bluetooth SPP-C

Este dispositivo es compatible con la interfaz UART (comunicación serial) y admite el protocolo serie de Bluetooth SPP, con transmisiones de hasta docenas de kilobytes por segundo. Algunas características para destacar son:

- El tipo de bluetooth que soporta es de Bluetooth V2.1+EDR, adecuado para la aplicación.
- La alimentación debería ser de 3.3V, pero soporta hasta 5.6V, por lo que se puede alimentar con la salida del Arduino.
- Posee una banda de frecuencia de operación de 2.4 GHz - 2.48 GHz.

Se escogió esta opción frente a otros módulos como el HC-06 o HC-05, por ventajas como su bajo coste y consumo, además de una superior sensibilidad de envío y recepción y, por lo tanto, consiguiendo mayor eficiencia de trabajo.

4.2.2 Alimentación

En este apartado se tratará principalmente el abastecimiento eléctrico del robot, además de los elementos que permiten la conexión de todos los componentes del dispositivo móvil.

4.2.2.1 Pila 9 V

Para la alimentación general de todo el circuito, se necesitaba un voltaje lo suficientemente elevado para que todos los elementos del robot pudieran funcionar de manera correcta. Por lo tanto, la opción escogida fue una pila de 9 V (*Fig. 16*).

Figura 16. Pila de 9 V utilizada

El proceso de alimentación del circuito consistirá en alimentar el “driver” del motor L298N mediante la pila escogida (colocando el jumper de activación como se ha explicado en el apartado del controlador). Una vez el “driver” este alimentado, el propio módulo tiene una salida de 5 V, que se utilizará para la alimentación directa del microcontrolador Arduino Nano. De esta forma, con una única pila, se consigue la alimentación completa del robot.

4.2.2.2 Cableado

La utilización de cables es esencial para asegurar la conexión entre los diferentes elementos del robot, tanto en la parte mecánica como en la parte de electrónica. Para ello, se han utilizado cables Dupont de diferentes tipos y longitudes (*Fig. 17*).

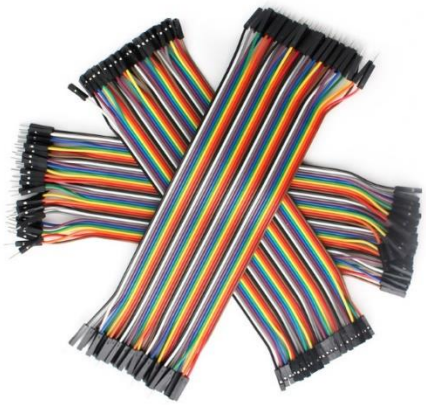


Figura 17. Cables Dupont de todos los tipos

Estos cables se caracterizan principalmente por tener muchas variantes, tanto en su conectividad como en su longitud, además de ser cables muy flexibles y con numerosas aplicaciones. En el proyecto se han utilizado los tres tipos existentes: macho-macho, hembra-hembra y macho-hembra, dependiendo del tipo de conectores que se necesitaran.

Por otra parte, cabe destacar que para la conexión de la pila de 9 V se ha utilizado un conector especial (*Fig. 18*) que se ajusta con los bornes de la pila, facilitando la conexión con el circuito.

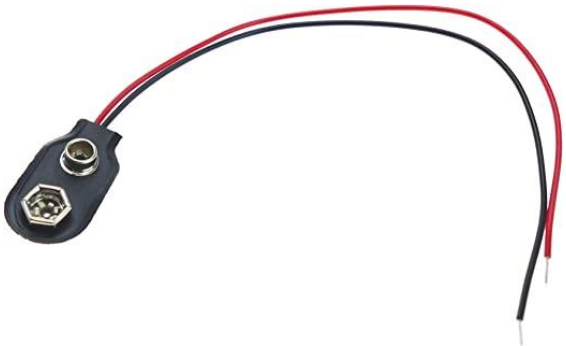


Figura 18. Cable de conexión para pila de 9 V

Finalmente, se ha utilizado un cable USB a Mini USB (*Fig. 19*) que ha permitido la conexión entre el ordenador y la placa Arduino. Mediante este cable se ha podido realizar la carga de los programas al microcontrolador, además de ser utilizado para realizar pruebas durante la conexión serial.

Figura 19. Cable de conexión entre el Arduino y el ordenador

4.2.2.3 Interruptor

Para evitar tener que estar desconectando la fuente de alimentación del robot cada vez que se tenga que mantener apagado, se ha utilizado un interruptor de dos posiciones (*Fig. 20*). Este elemento permite cortar la corriente que suministra la pila de entrada al controlador cerrando el circuito; o abriéndolo para alimentar el robot y llevar a cabo la aplicación.

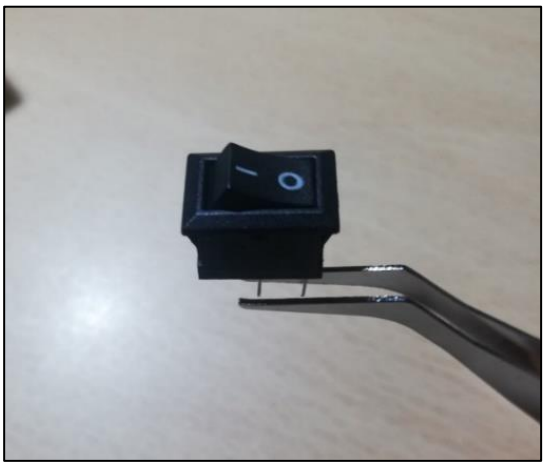


Figura 20. Interruptor utilizado en el robot

Elaboración propia

4.3 Estructura mecánica

En este apartado se explicarán principalmente los elementos mecánicos de los que estará compuesto el proyecto, centrándose en el movimiento y la forma del robot móvil que perseguirá al objetivo.

4.3.1 Chasis

El chasis del robot es el encargado de proporcionar un soporte físico y la base para todos los elementos que lo componen. Permite que el robot sea una estructura unida, dotándolo de resistencia, rigidez y estabilidad, ya que debe ser lo suficientemente robusto como para soportar los movimientos y fuerzas que el robot puede llegar a recibir durante su funcionamiento.

Para la elección del chasis se siguieron criterios principalmente relacionados con la sencillez a la hora de la fabricación y montaje del robot. Por lo tanto, se decidió utilizar una base prediseñada, que disponía de numerosas perforaciones, facilitando en gran medida la instalación de los elementos del robot. Además, la propia base incluía diferentes piezas que permitían la instalación tanto de los motores como de las ruedas, disminuyendo el coste económico del proyecto (*Fig. 21*).

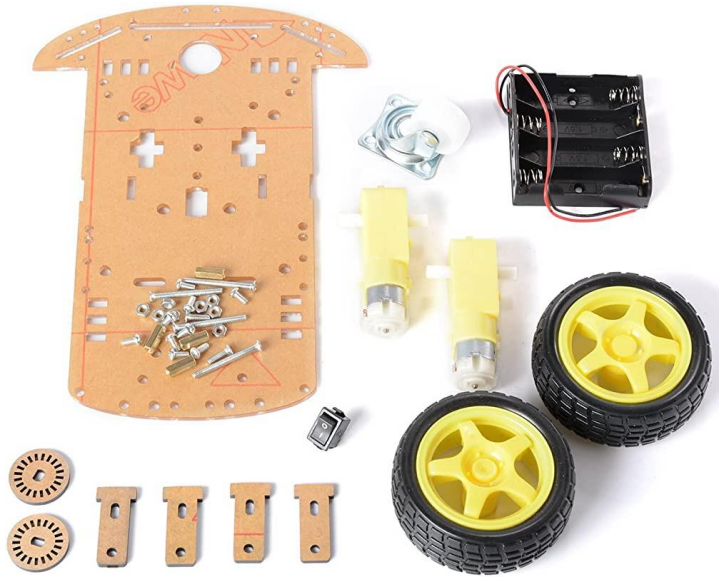


Figura 21. Base perforada del robot, incluyendo diferentes elementos mecánicos

Por otra parte, para completar el chasis, se ha construido una caja utilizando bastones y placas de madera, con el objetivo de proteger los componentes del interior del robot. Finalmente, se utilizaría una estructura igual que la base a modo de techo del robot, para acabar de sellarlo completamente.

4.3.2 Motores DC

Los motores corresponden a los actuadores principales del robot y son los que permiten que este se desplace por el entorno. Para este proyecto se han escogido unos motores de corriente continua con un voltaje de funcionamiento de 3 V a 6 V, con una reductora de 1:48, un peso de 50 g cada uno y una velocidad máxima con carga de 160 RPM (*Fig. 22*).

Figura 22. Motores utilizados para el movimiento del robot

Este tipo de motores son ampliamente utilizados en proyectos de robótica de bajo coste, gracias a su facilidad de integración con la plataforma de Arduino, su disponibilidad en las diferentes tiendas de electrónica, su económico precio y su eficiencia energética, adecuada en todo tipo de aplicaciones.

4.3.3 Ruedas

Para que el robot pueda desplazarse sin complicaciones en las diferentes direcciones posibles durante la ejecución de la aplicación, se han utilizado dos tipos de ruedas diferentes:

- **Ruedas fijas convencionales:** Como ya se explicó en el apartado de los robots móviles terrestres, estas ruedas (*Fig. 23*) son las más usuales y se montaran junto a los motores de continua. Se utilizarán dos ruedas (una por cada motor) las cuales están hechas de plástico, con un diámetro de 65 mm y un exterior de goma, que ayuda a la fijación con la superficie.

Figura 23. Rueda fija utilizadas en el robot Arduino

- **Rueda loca:** Este tipo de rueda (*Fig. 24*) corresponde al grupo de ruedas no convencionales, ya que tiene libertad para moverse en todas direcciones. La función principal de esta rueda es dotar al robot de estabilidad a la hora de hacer los movimientos durante la aplicación.

Figura 24. Rueda loca utilizada para el montaje del robot

4.4 Materiales y componentes del escenario

Una parte importante de la aplicación es la composición del escenario, es decir, todos los elementos que van a hacer falta en el entorno para conseguir que la aplicación funcione de manera correcta.

4.4.1 Cámara

Una de las partes más importante del proyecto es la infraestructura de visión que se va a utilizar, el cuál debe ser capaz de transmitir la imagen al sistema de procesamiento de imagen para llevar a cabo la detección. En este apartado se van a explicar algunas alternativas que se pensaron durante la elección de la cámara.

Una de las primeras opciones que se considero fue la de implementar una **cámara ESP32-Cam** (*Fig. 25*), una placa de desarrollo muy utilizada en proyectos de IoT (“Internet of Things”) por sus considerables prestaciones y aplicaciones. Posee conectividad wifi y bluetooth, además de una pequeña cámara modelo OV2640, con un sensor de 2 MP y una resolución de 1600 x 1200 píxeles. Aunque se consiguió programar de forma que se pudiera extraer la imagen para procesarla, esta opción finalmente quedó descartada por problemas de estabilidad para la obtención de la imagen.

Figura 25. Alternativa 1 de sistema de visión: Cámara ESP32-Cam

Otra opción que se comenzó a estudiar fue la de adquirir una cámara IP de vigilancia, más concretamente el **modelo Wansview Q5** (*Fig. 26*). Esta cámara se utilizaba principalmente en aplicaciones relacionadas con la seguridad, dando la posibilidad de conectarse a ella de forma remota mediante un software propio de la cámara y pudiendo realizar procesamiento de la imagen que se obtenía. Esta alternativa fue rechazada principalmente por el gasto económico que suponía su compra y las prestaciones que ofrecía este dispositivo, que excedían a lo necesario para el proyecto.

Figura 26. Alternativa 2 de sistema de visión: Cámara IP Wansview Q5

Finalmente, la opción escogida fue la utilización de un **teléfono móvil** con la cámara que este posee, ya que es un dispositivo del que todo el mundo actualmente dispone, además de que no tuvo coste adicional ya que no había que realizar ninguna compra extra, y su implementación sería relativamente sencilla en el proyecto con el uso de aplicaciones externas. El móvil utilizado en cuestión es un Huawei P20 lite (Manual Huawei P20 llite, s.f) (*Fig. 27*), que posee una cámara con las características siguientes:

- **Cámara trasera:** Posee 2 sensores, el principal es de 16 MP con una apertura focal de 2.2; mientras que el secundario es un sensor de 2 MP con una apertura focal de 2.4.
- **Cámara frontal:** Se corresponde con un sensor de 16 MP con una apertura focal de 2.0.

La grabación de video de este dispositivo permite grabar videos en Full HD (1080p) a 30 fotogramas por segundo, sin embargo, la resolución que se va a utilizar durante la aplicación va a ser mucho menos, ya que la elevada carga de procesamiento de imagen junto a una resolución muy grande introduciría mucho retraso en la transmisión de la imagen.

Figura 27. Sistema de visión utilizado: Huawei P20 lite

4.4.2 Trípode

Como se ha comentado anteriormente, la posición de la cámara debe ser cenital y elevada, ya que sino la aplicación no tendrá suficiente rango de visión para poder implementarse de manera correcta. De esta forma, se ha utilizado un trípode para dispositivos móviles que permite regular la posición y orientación del móvil (*Fig. 28*). Además, este soporte posee unos enganches que posibilitan el agarrarse a prácticamente cualquier superficie.

Figura 28. Trípode para dispositivos móviles utilizado

4.4.3 Entorno

El lugar donde se van a realizar las pruebas es muy importante, ya que de ser bastante espaciado para el robot tenga rango de movimiento y la aplicación tenga un sentido de implementación. Además, si no es adecuado, se pueden tener detecciones falsas o fallos durante la ejecución, por ello, se debe tratar de acondicionar lo mejor posible.

De esta forma, el trípode se colocará a una cierta altura y estará sujeto a una estructura de madera, dotándolo de mayor distancia con el mueble al que se ha fijado.

Para adaptar la superficie donde se va a realizar, se ha va a hacer uso de un mantel de papel completamente blanco, evitando así problemas de reflejos con la iluminación utilizada durante la aplicación y, a su vez ayudando, a mejorar la detección.

5 Descripción detallada de la solución adoptada

En esta sección se va a explicar de forma detalla los pasos seguidos para el desarrollo de la aplicación al completo, teniendo en cuenta tanto problemas como alternativas planteadas durante la ejecución y realización del proyecto.

Como ya se ha descrito anteriormente, la propuesta de funcionamiento se basa principalmente en la detección de un objetivo (posición) y un robot (posición y orientación) mediante técnicas de procesamiento de imágenes, para posteriormente diseñar una ley de control para que el robot lleve a cabo un seguimiento preciso de este objetivo. El funcionamiento queda detallado en el siguiente diagrama de bloques (*Fig. 29*):

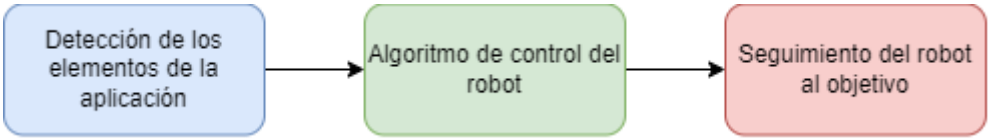


Figura 29. Diagrama de bloques del proyecto

Elaboración propia

Por lo tanto, este apartado se ha dividido en las tres ramas tratadas en el proyecto: robótica (donde se tratará el montaje del robot), visión y procesamiento de imagen y control.

5.1 Montaje del robot

En este apartado se tratarán todos los aspectos relacionados con el montaje del robot que se va a utilizar para llevar a cabo la aplicación de seguimiento del objetivo. Como ya se comentó en el apartado del hardware, la elección del chasis ha consistido en una base perforada y prediseñada para la implementación de los motores, todo junto a un pack de piezas para realizar el ensamblaje completo.

De esta forma, se comentarán puntos como el montaje del chasis y los motores, la implementación de ambos tipos de ruedas y el montaje de la electrónica para finalmente cerrar el robot.

5.1.1 Montaje de los motores

El primer paso que se ha llevado a cabo es el montaje de los motores en el chasis. Para realizar este procedimiento, se deben tener en cuenta las piezas que se van a necesitar para el ensamblaje de ambos motores y su colocación en la base perforada.

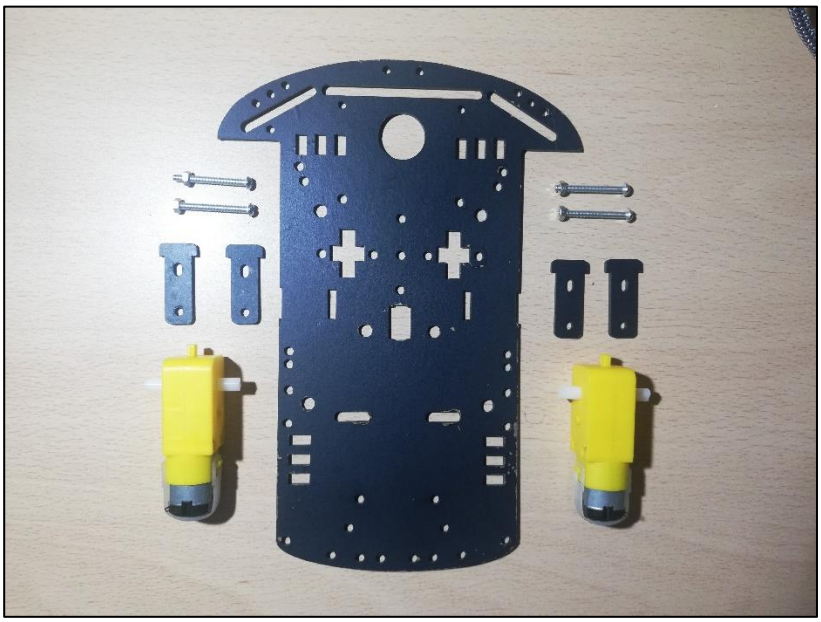


Figura 30. Piezas para el montaje de los motores

Elaboración propia

Como se observa en la imagen (*Fig. 30*), los componentes de este primer montaje son:

- 2 motores DC.
- 4 tornillos de L = 3 cm, junto a sus correspondientes tuercas.
- 4 piezas de madera diseñadas para el ensamblaje de los motores en el chasis.
- Base perforada.

Antes de comenzar cualquier montaje respecto del motor, se realizaría un proceso de soldadura para que los motores pudieran ser alimentados y controlados por el “driver”. Por lo tanto, se soldarán dos cables a cada motor en las pestañas metálicas que estos poseen, obteniéndose el siguiente resultado (*Fig. 31*):

Figura 31. Motor DC con el cableado soldado

Elaboración propia

Con los motores cableados, el montaje de estos en el chasis consiste en colocar las piezas de madera en los agujeros correspondientes de la base, indicados en la *Fig. 32*:

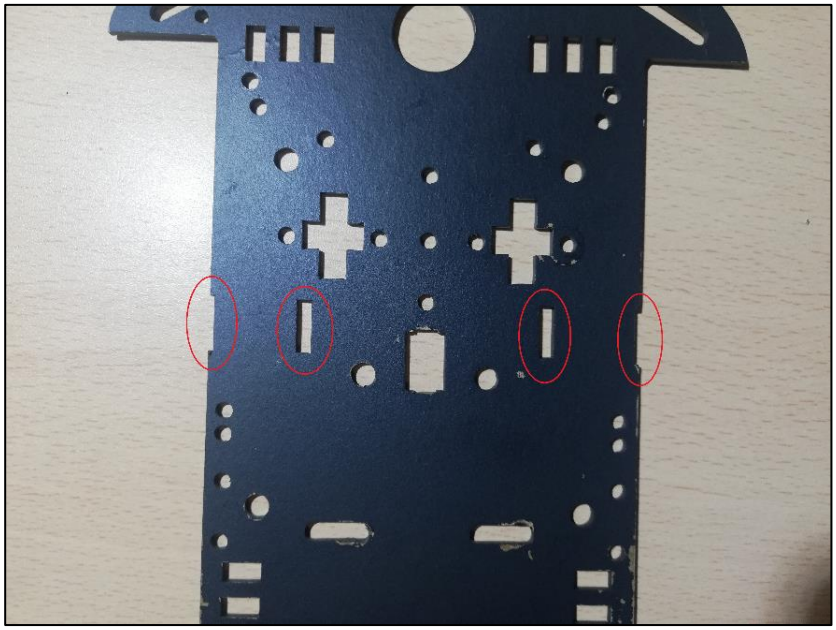


Figura 32. Lugar de colocación de las sujeciones de madera

Elaboración propia

En las secciones señaladas se colocarán las piezas de madera, que tienen dos agujeros al mismo nivel que los de los motores. Por lo tanto, el siguiente paso sería atornillar los motores a la pieza de madera mediante las tuercas, obteniendo un resultado similar al de la *Fig. 33*:

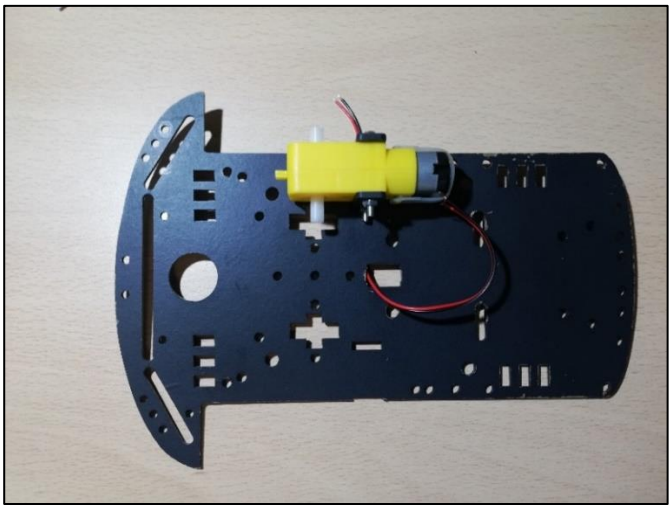


Figura 33. Montaje del motor izquierdo completo

Elaboración propia

Finalmente, se realizará el mismo procedimiento para el otro motor utilizando de nuevo las sujeciones de madera y los tornillos, finalizando así la primera parte del montaje:

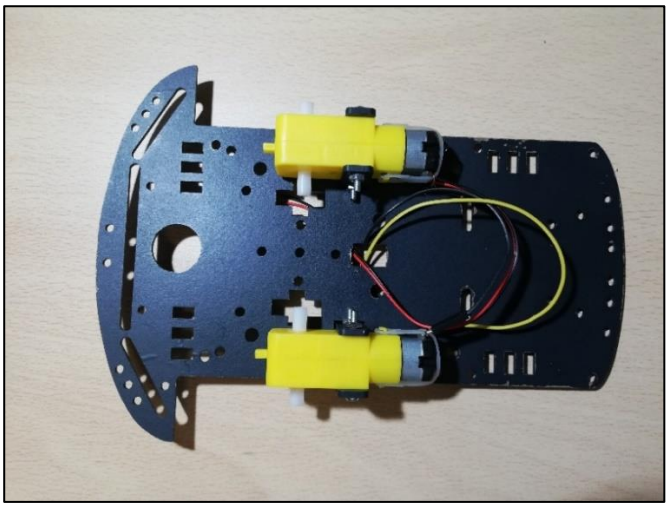


Figura 34. Montaje completo de ambos motores

Elaboración propia

5.1.2 Montaje de las ruedas

Con la parte delantera de los motores completada, el siguiente procedimiento es la instalación de la rueda loca o castor, y las ruedas convencionales.

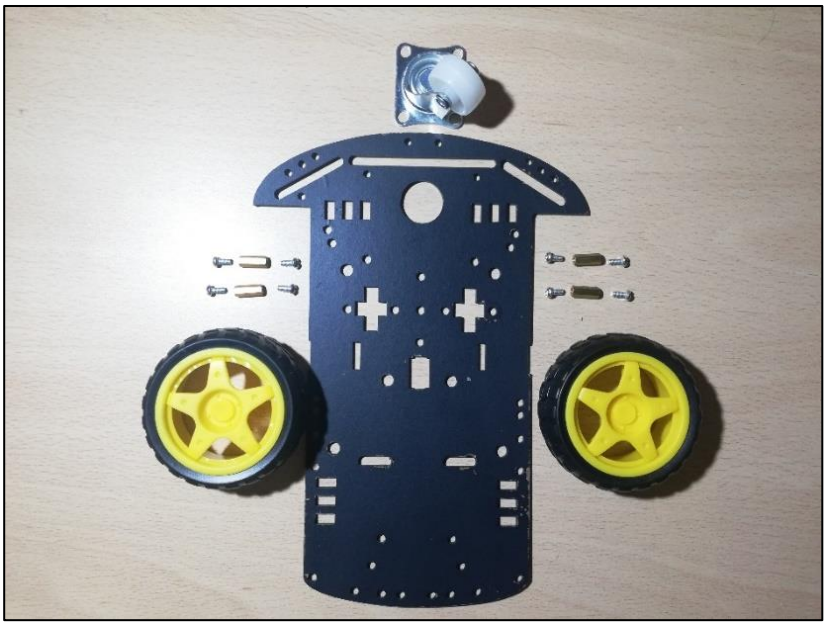


Figura 35. Piezas para el montaje de las ruedas

Elaboración propia

De nuevo, se pueden ver los diferentes elementos en la *Fig. 35*, donde se incluyen:

- 2 ruedas convencionales.
- 8 tornillos de L = 0.6 cm.
- 4 separadores de metal de longitud L = 1.1 cm.
- 1 rueda loca.
- Base perforada.

Con todos los elementos disponibles, el primer montaje se corresponderá con la rueda de tipo castor, que se deberá colocar en la zona inferior y en las perforaciones señaladas en la *Fig. 36*:

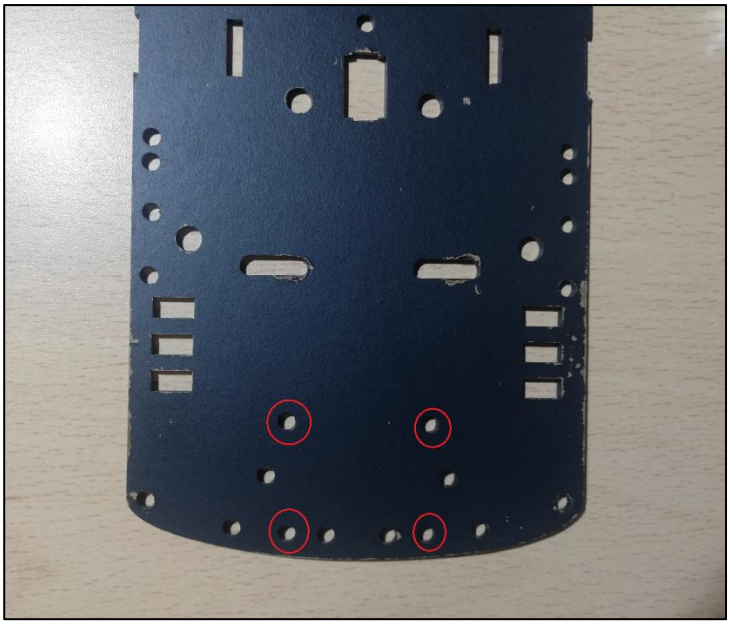


Figura 36. Lugar de colocación de la rueda loca

Elaboración propia

El primer paso sería atornillar la rueda loca a los separadores que vienen con el pack de montaje para posteriormente poder juntar la rueda loca con la propia base, utilizando los tornillos restantes. El resultado final debe ser el siguiente (*Fig. 37*):

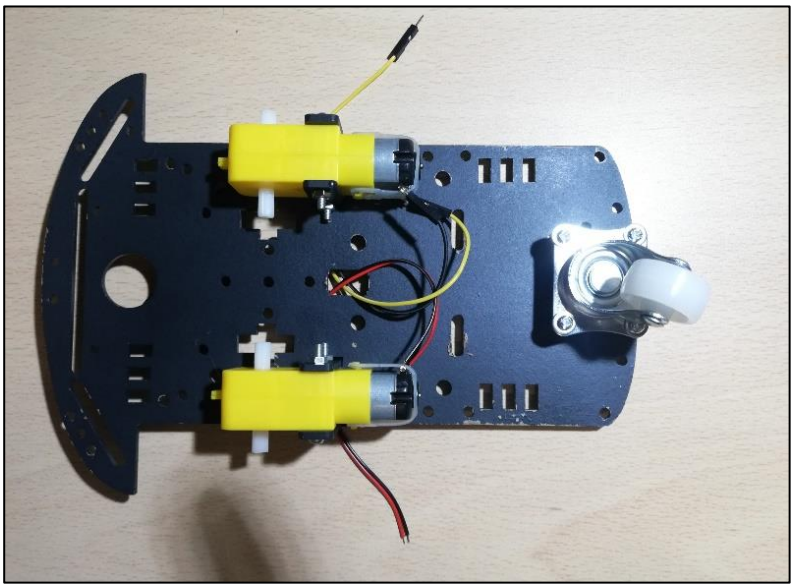


Figura 37. Chasis con el montaje de la rueda loca

Elaboración propia

Finalmente, se pueden instalar las ruedas convencionales en los motores DC, para obtener el chasis completamente montado (*Fig. 38*).

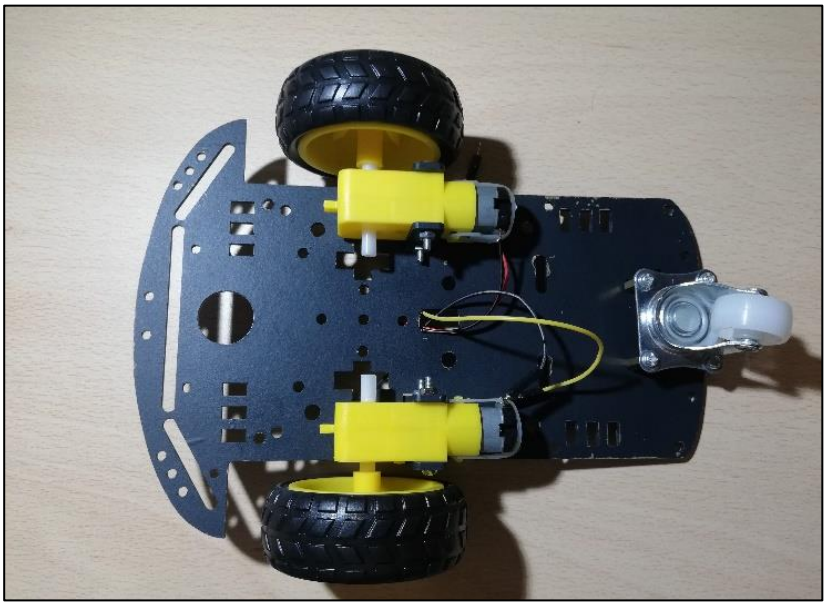


Figura 38. Montaje completo de la estructura mecánica

Elaboración propia

5.1.3 Montaje de la electrónica

En este apartado se va a tratar la disposición de la electrónica en el interior del robot, obviando la parte de conexionado que se podrá visualizar en el apartado de planos.

Para fijar la electrónica en el interior, se han utilizado perforaciones que ya tiene la propia base con una disposición en concreto de forma que los diferentes elementos de los que está compuesto el robot no se entorpezcan entre sí.

De esta forma, el montaje principal de la electrónica sería:

- **Shield de Arduino:** El microcontrolador de Arduino se colocará de forma elevada en la zona central del robot. Para llevar a cabo este montaje, se utilizarán tres separadores, tres tornillos metálicos y tres tornillos de plástico (para evitar cortocircuitos), atornillándolo directamente a la base del robot.

- **Driver L298N:** Para la colocación del driver se ha utilizado una estrategia similar a la explicada anteriormente en el Arduino, pero esta vez se ha decidido atornillarlo en la parte superior, es decir, en el techo del robot y boca abajo, utilizando nuevamente tornillos y separadores. De esta forma se consigue ahorrar espacio en la zona inferior para la pila de alimentación o el módulo bluetooth.
- **Módulo Bluetooth:** Estará en el interior del robot y se colocará pegado con cinta de doble cara debido a la falta de perforaciones en el módulo para atornillarlo.
- **Pila 9V:** La fuente principal de alimentación no irá sujeta, ya que teniendo en cuenta que es una pila y en algún momento se gastará, se colocará suelta o con una pequeña brida para evitar movimientos bruscos.
- **Interruptor:** Se colocará en el techo (que es igual que la base), ya que tiene una perforación donde se puede encajar a la perfección.

De esta forma, la colocación de la electrónica tendrá la forma que se observa en las *Figs. 39 y 40*:

Fig. 39. Colocación de la electrónica (visión lateral)

Elaboración propia

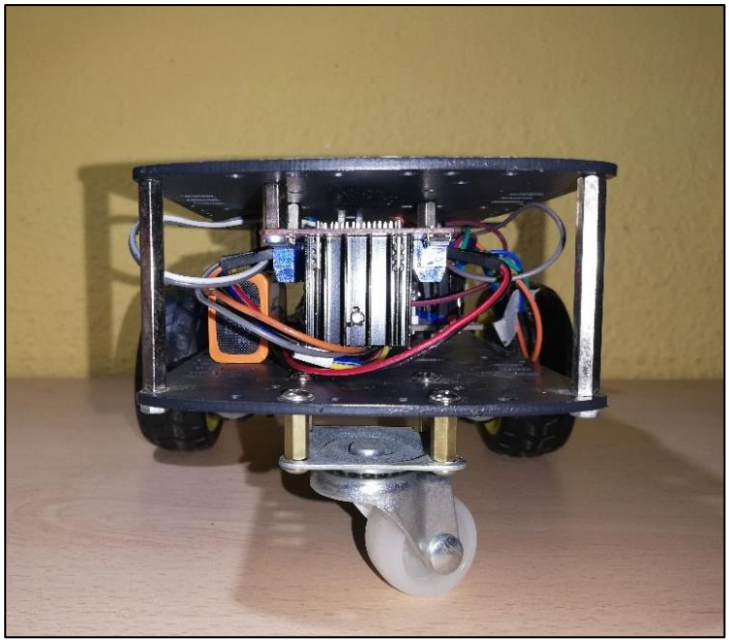


Fig. 40. Colocación de la electrónica (visión trasera)

Elaboración propia

5.1.4 Sellado del robot

Con todo el montaje realizado, solo faltaría sellar el robot, dotándolo de robustez y estabilidad para que pueda ser plenamente utilizado en la aplicación.

La estrategia de montaje utilizada ha consistido en la implementación de un recubrimiento de madera en forma de caja para tapar los laterales del robot, y la utilización de un techo con las mismas perforaciones y forma que la base, consiguiendo el sellado completo del robot.

Primeramente, se tratará la construcción de la caja que recubrirá el robot por los laterales, por la parte frontal y por la parte trasera. La construcción de esta carcasa es relativamente sencilla, ya que únicamente se han utilizado piezas de madera, tanto para las paredes como para los bastones de unión entre cada una de ellas. Por lo tanto, el procedimiento a seguir ha sido unir las paredes de madera de forma que formen un ángulo recto, y fijarlas con los bastones y el pegamento de contacto, obteniendo el resultado visto en la *Fig. 41*:

Figura 41. Carcasa del robot

Elaboración propia

Finalmente, se colocará la parte superior, que se debe tener en cuenta que tiene atornillado el controlador de los motores de continua. Para fijar el techo del robot, se han utilizado unos separadores de placa de 45 mm con una altura similar a la de la pared de la caja, de esta forma, se utilizarán las perforaciones tanto de la base como de la parte superior para atornillar completamente el robot y sellarlo (*Fig. 42*).

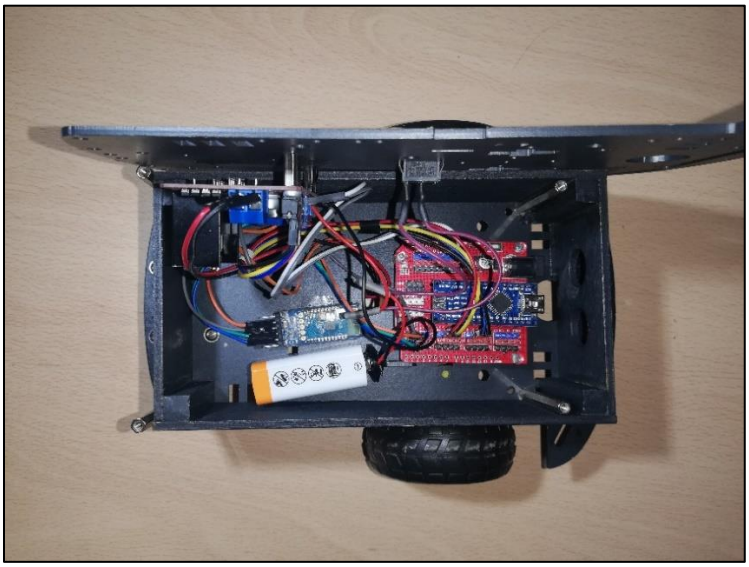


Figura 42. Robot montado visto desde arriba

Elaboración propia

Con todos los pasos realizados, el montaje del robot quedaría finalizado, y el resultado sería el siguiente (*Fig. 43*):

Figura 43. Montaje final del robot móvil

Elaboración propia

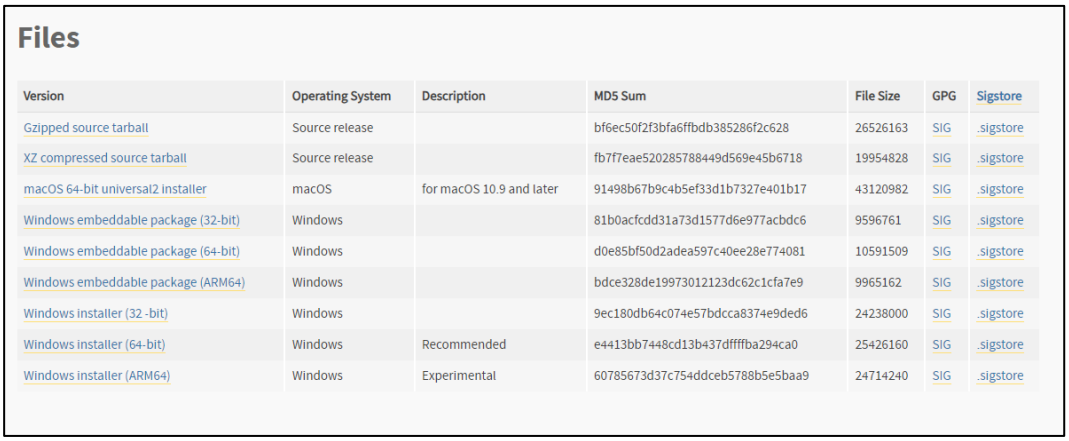
5.2 Visión y procesamiento de imagen

En este punto se tratarán todos los temas relacionados con la captación de la imagen (aplicaciones utilizadas y su instalación), además de las diferentes técnicas de procesamiento de imágenes utilizadas para conseguir las detecciones correspondientes y sus implementaciones en código.

5.2.1 Instalación de Python y Virtual Studio Code

Para realizar el procesamiento de imágenes se va a utilizar **Python**, un lenguaje de programación muy utilizado actualmente para diferentes aplicaciones debido a su facilidad de aprendizaje y a su versatilidad. Consta de una sintaxis muy similar al inglés y es un lenguaje tipeado dinámicamente, ya que no se necesitan declarar los tipos de variables.

Para la instalación de Python en el dispositivo, únicamente hay que dirigirse a la página oficial y descargar el archivo ".exe" que nos indique, teniendo en cuenta el tipo de dispositivo que se está utilizando (Windows, Linux o Mac) y el sistema operativo (32 o 64 bits) (*Fig. 44*).



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		bf6ec50f2f3bfa6ffbdb385286f2c628	26526163	SIG	.sigstore
XZ compressed source tarball	Source release		fb7f7eae520285788449d569e45b6718	19954828	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	91498b67b9c4b5ef33d1b7327e401b17	43120982	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		81b0acfcdd31a73d1577d6e977acbdc6	9596761	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		d0e85bf50d2adea597c40ee28e774081	10591509	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		bdce328de19973012123dc62c1cfa7e9	9965162	SIG	.sigstore
Windows installer (32-bit)	Windows		9ec180db64c074e57bdcca8374e9ded6	24238000	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	e4413bb7448cd13b437dffffba294ca0	25426160	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	60785673d37c754ddceb5788b5e5baa9	24714240	SIG	.sigstore

Figura 44. Pantalla principal para la instalación de Python

Una vez instalado, únicamente ejecutando el archivo descargado y siguiendo los pasos que propone el ejecutable, Python estaría disponible para ser utilizado en el dispositivo (*Fig. 45*). Es importante recalcar que también se debe instalar "pip" (que se incluye en la instalación de Python), ya que se usará posteriormente para la instalación de las librerías necesarias durante el proyecto.

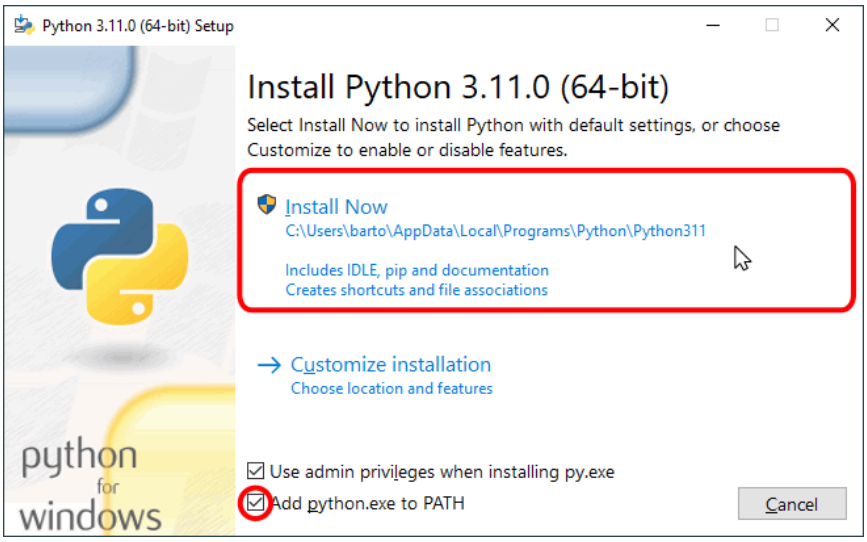


Figura 45. Pantalla principal del instalador de Python

Una vez se haya instalado correctamente el lenguaje que se va a utilizar para la realización de la parte de visión del proyecto, se va a descargar un entorno o IDLE para realizar la programación del código de forma cómoda y con diferentes ayudas durante la escritura de este. El entorno de programación escogido va a ser Virtual Studio Code (*Fig. 46*), un editor de código muy utilizado debido a su soporte para números lenguajes de programación (C, C++, Python, JavaScript…).

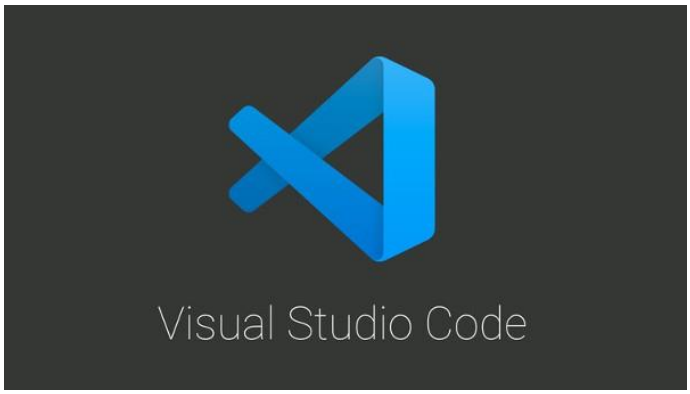


Figura 46. Editor de código utilizado para Python

La instalación es muy similar a la anteriormente explicada, se deberá entrar en la página oficial de la aplicación (*Fig. 47*) y descargar el archivo ".exe", que trae consigo el instalador. Se deberá tener en cuenta el tipo de sistema operativo que se utilizará para la aplicación.

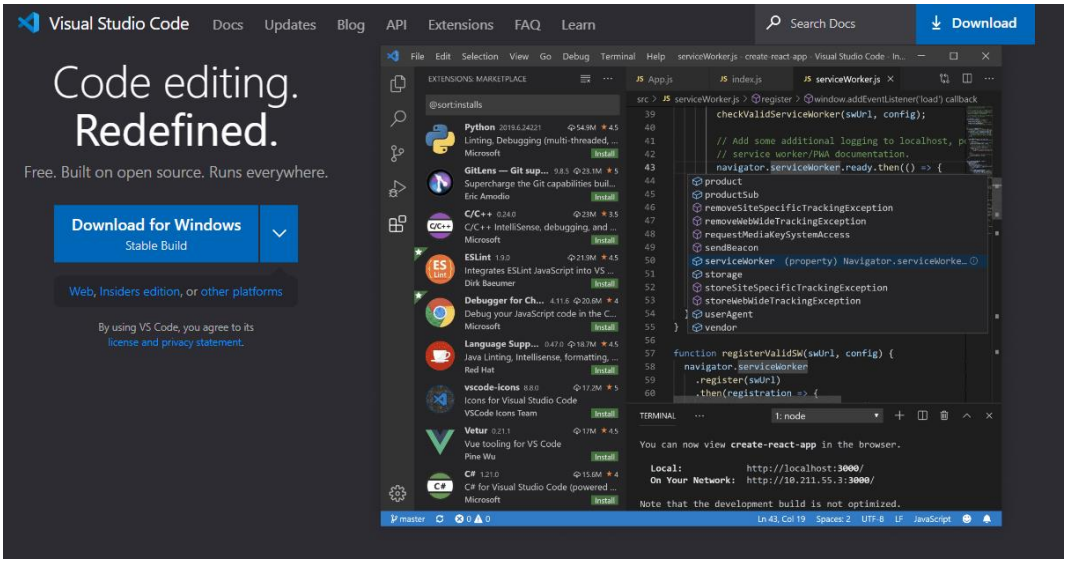


Figura 47. Pantalla principal para la instalación de Virtual Studio Code

Con ambas instalaciones completadas, únicamente será necesario descargar la extensión de Python en el entorno de programación, permitiendo de esta forma programar y realizar pruebas de código en el editor (*Fig. 48*).

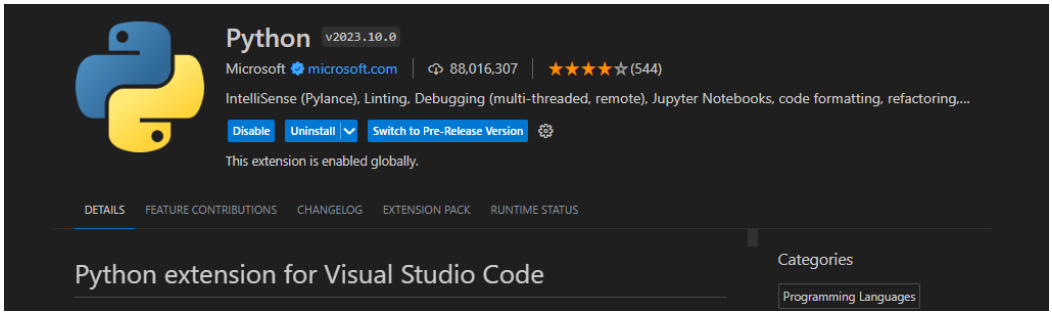


Figura 48. Instalación de Python en VSC

Elaboración propia

5.2.2 Instalación de librerías

Las librerías o bibliotecas son módulos y funciones ya definidas que permiten acortar y facilitar la escritura de código. Mediante el uso de librerías, Python se ha convertido en uno de los lenguajes más utilizados, gracias a la versatilidad y la eficiencia que estas herramientas le dan a la programación en este lenguaje.

Durante la ejecución de este proyecto, se ha hecho uso de numerosas librerías, cada una con sus funciones personalizadas, que facilitan la programación de la aplicación. De esta manera, las librerías utilizadas en el proyecto son:

- **OpenCV:** Esta librería es muy común en aplicaciones relacionadas con visión artificial, ya que cuenta con numerosas funciones para la implementación de métodos de análisis de imágenes. El uso de esta librería ha sido principalmente la captura de imagen, la calibración de la cámara, la detección de los elementos relevantes de la aplicación y la implementación de técnicas de procesamiento como la apertura, el cierre o el dibujado de formas y contornos.

- **Numpy:** Es una librería que facilita la computación numérica y la implementación de matrices N-dimensionales. La función principal que se le ha dado a esta librería en el código es la de creación de máscaras (para la realización de operaciones matemáticas en imágenes) o vectores para la implementación de los diferentes filtros de color utilizados en el proyecto.

- **Pickle:** El uso de esta librería se basa en la serialización, es decir, la conversión de datos en secuencias de bytes para que puedan ser comprendidos por la computadora y, por lo tanto, almacenarse en la memoria. La utilización que se le ha dado es la de almacenar los resultados de la calibración de la cámara en un archivo de extensión ".pkl" para poderse utilizar en cualquier código que incluya el mismo dispositivo de visión.

- **Math:** Librería muy utilizada no solo en Python sino también en otros lenguajes de programación y que contiene numerosas funciones matemáticas (definidas en C) para la realización de operaciones. Se ha utilizado principalmente para cálculos en los que intervenían funciones trigonométricas como el seno, el coseno o la tangente.

- **Serial:** Su función principal reside en establecer una comunicación serial. A partir de las funciones de esta biblioteca se ha programado el intercambio de datos con el robot, utilizando la tecnología Bluetooth.

- **Json:** Esta librería ha permitido la codificación de datos para que puedan ser enviados a Arduino por comunicación serial. Es muy útil para intercambiar datos estructurados entre diferentes sistemas. Cabe destacar que para que funcione correctamente y se puedan recibir los datos, se deberá instalar también una biblioteca en Arduino desde el menú de administración de librerías.

- **Glob:** Este módulo permite realizar una búsqueda en el dispositivo a partir de una ruta de archivos. Esta librería ha sido utilizada únicamente para implementar las imágenes del patrón de calibración (tablero de ajedrez) en el código utilizado para realizar la calibración de la cámara.

Con todas las librerías explicadas, hay que recalcar que no todas necesitan un proceso de instalación, ya que Python dispone de muchos de estos conjuntos de funciones implementadas directamente en el lenguaje, teniendo únicamente que realizar un *import* al comienzo del código. Las librerías que no deben instalarse de forma externa son Pickle, Math, Json, Glob; mientras que OpenCV, Numpy y Serial si deben instalarse con el método que se explicará a continuación.

Para realizar la instalación de las librerías externas se debe contar con el sistema de gestión de paquetes “pip”, que debe de haberse incluido durante la instalación de Python. De esta manera, para llevar a cabo las instalaciones, únicamente debe abrirse el símbolo del sistema o la consola de comandos y escribir las siguientes sentencias:

- ‘pip install opencv-python’
- ‘pip install numpy’
- ‘pip install pyserial’

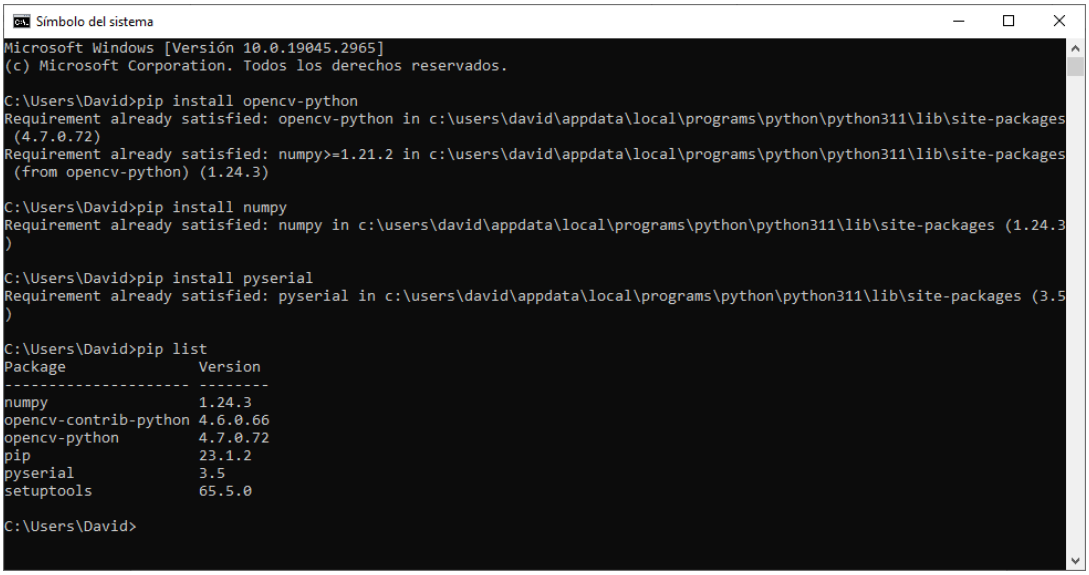


Figura 49. Símbolo del sistema con la instalación de librerías externas

Elaboración propia

Como se observa en la *Fig. 49*, tras la instalación de las diferentes librerías, se ha podido observar los diferentes paquetes que hay instalados utilizando ‘pip list’, además de la versión instalada de cada librería.

Finalmente, recalcar el proceso de instalación de la librería Json para la plataforma de Arduino, donde únicamente se debe realizar una búsqueda en el administrador de bibliotecas (en el apartado de herramientas) de la librería en cuestión e instalar la versión más actualizada de esta. El autor de la librería que se ha utilizado es Benoit Blanchon y su versión más actualizada es la 6.21.2 (*Fig. 50*).

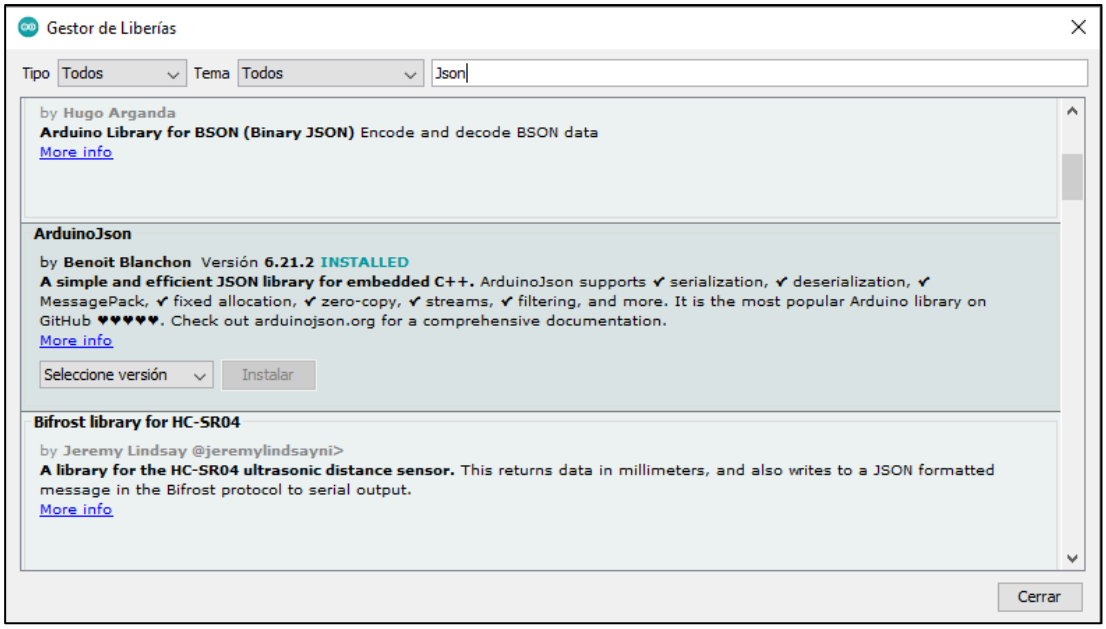


Figura 50. Instalación de la librería Json en Arduino

Elaboración propia

5.2.3 Configuración de la cámara IP

El primer paso para obtener una imagen a procesar es la elección de la aplicación que permitirá transmitir la imagen a Python. De esta forma, se plantearon dos alternativas principales, ambas aplicaciones fáciles de obtener y gratuitas: DroidCam y IP WebCam.

- **DroidCam:** Esta aplicación necesita la instalación extra de un programa en el ordenador, pero permite que la imagen se obtenga como una cámara extra al PC, por lo que no requiere de la lectura de un URL.
- **IP WebCam:** Esta aplicación únicamente debe instalarse en el dispositivo móvil y la transmisión de imagen se realiza mediante servidor http, teniendo que realizar una lectura de URL.

Finalmente, la alternativa escogida fue IP WebCam, ya que esta segunda opción permite la elección de la resolución de la cámara para la aplicación, no como la primera alternativa, que está totalmente limitada a una resolución de 640 x 480 píxeles y además no permite la realización de fotografías. También es importante destacar que se evita la instalación de un programa extra en el ordenador, facilitando el planteamiento del proyecto.

La instalación de IP WebCam es realmente sencilla, únicamente se debe entrar en la tienda de aplicaciones Play Store (en Android) y pulsar el botón de instalación (*Fig. 51*).

Figura 51. Menú de instalación de IP WebCam

Elaboración propia

Con la aplicación ya instalada en el dispositivo móvil, únicamente deberemos iniciarla como cualquier “app”. Ya dentro, aparecen diversas opciones, donde la mayoría de estas carecen de importancia para la ejecución del proyecto. Únicamente habría que destacar la opción de “Preferencias de Video” (*Fig. 52*), en la que se puede cambiar la resolución de video que se va a trasmitir al sistema de procesamiento de imagen.

Figura 52. Menú de "Preferencias de video" de IP WebCam

Elaboración propia

Para el proyecto, se ha escogido una **resolución de 960 x 720 píxeles**, ya que, aunque la imagen puede tener retraso debido a factores como la conexión o el continuo procesamiento en tiempo real con resoluciones elevadas, esta configuración permitía un amplio rango de visión para que el proyecto pueda ser ejecutado de manera correcta. Finalmente, la visualización en el dispositivo móvil se observa en la *Fig. 53*:

Figura 53. Pantalla principal en la aplicación IP WebCam

Elaboración propia

5.2.4 Calibración de la cámara

En cualquier proyecto en el que intervenga un sistema de visión, es necesario realizar un proceso de calibración, permitiendo conocer los parámetros de la cámara y usarla de manera efectiva en la aplicación. La calibración en este proyecto tiene como principal objetivo evitar la distorsión que la propia cámara pueda introducir en la imagen y, por lo tanto, eliminar imprecisiones en las detecciones correspondientes (*Fig. 54*).

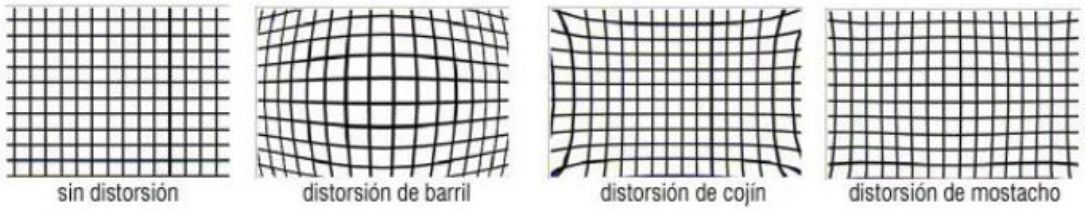


Figura 54. Tipos de distorsión en una imagen

Primeramente, se debe entender en que consiste el proceso de calibración de una cámara. El planteamiento principal es determinar una relación entre un punto en 3D y su correspondiente en el plano 2D de una imagen, es decir, su posición en píxeles. Para llevar esto a cabo, se deben tener en cuenta los parámetros que posee la cámara:

- **Parámetros extrínsecos:** Están relacionados con la posición y orientación de la cámara respecto al sistema de coordenadas global.
- **Parámetros intrínsecos:** Hacen referencia a características de la cámara y la lente, como podrían ser la distancia focal y los coeficientes de distorsión.

De esta forma, la ecuación esencial que permite establecer la relación de los puntos tridimensionales (X_w, Y_w, Z_w) en el plano bidimensional de la imagen (u', v') es:

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = P \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

La matriz P, a su vez, está constituida principalmente por la matriz de parámetros intrínsecos (K) y la matriz extrínseca, que contiene la información de la rotación y translación de la cámara respecto al sistema global.

$$P = K * [R|Tt]$$

Además, la matriz K de parámetros intrínsecos está formada por las distancias focales f_x, f_y; y las coordenadas del centro óptico de la imagen c_x, c_y. Normalmente tiene otro valor relacionado con la inclinación y el sesgo de los ejes, pero OpenCV no trabaja con este parámetro.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Una vez explicados los conceptos de la calibración, se comienzan a realizar los pasos para llevarla a cabo de manera correcta. Primeramente, es importante conocer que método se ha utilizado para el proyecto, que ha sido la utilización de un **patrón de calibración** (*Fig. 55*). Este procedimiento consiste en capturar numerosas fotos desde distintos puntos de vista de un patrón conocido (en este caso un tablero de ajedrez), permitiendo calcular de esta forma los parámetros de calibración necesarios.

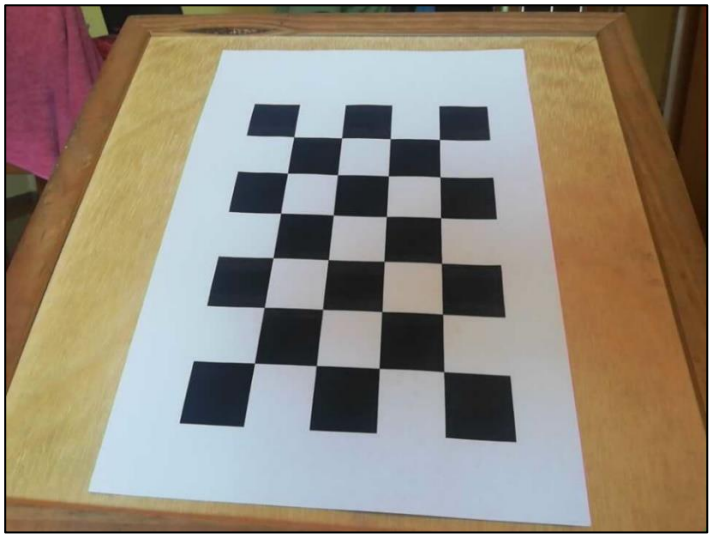


Figura 55. Patrón utilizado para llevar a cabo la calibración

Elaboración propia

El primer paso para llevar a cabo la calibración es definir las dimensiones del tablero que se va a utilizar y del tamaño de las imágenes que se van a tomar (que deberá ser la resolución de la imagen durante la ejecución el proyecto). También se deberán definir las matrices para las coordenadas del mundo real, y los vectores en los que se almacenarán tanto los puntos en 3D como los puntos de las imágenes en 2D (*Fig. 56*).

```
################ CÓDIGO PARA CALIBRACIÓN DE CÁMARA ##########################################
# Dimensiones del tablero y resolución de la cámara
Dimension_Tablero = (6,4)
Dimension_Frame = (960,720)


# Criterio de terminación
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)


# Definición de las coordenadas del mundo, como (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((Dimension_Tablero[0] * Dimension_Tablero[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:Dimension_Tablero[0],0:Dimension_Tablero[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 30
objp = objp * size_of_chessboard_squares_mm


# Vectores para almacenar los puntos objeto y los puntos imagen de todas las imagenes
P_objeto = [] # Vector para almacenar los puntos en 3D para cada imagen del tablero
P_imagen = [] # Vector para almacenar los puntos en 2D para cada imagen del tablero
```

Figura 56. Definición de vectores y matrices para calibrar la cámara

Elaboración propia

El siguiente paso sería implementar en el código las imágenes capturadas del patrón, que para llevar a cabo esto se utilizó *glob*, una de las librerías anteriormente explicadas (*Fig. 57*). Es importante recalcar que se deben tomar fotos del tablero en diferentes posiciones y orientaciones para mejorar el proceso de calibración.

```
# Extracción de las imagenes del directorio
imagenes = glob.glob('./imagenes/*.png')
```

Figura 57. Código para implementación de imágenes

Elaboración propia

Una cuestión para tener en cuenta fue la captura de fotos con la resolución que se iba a utilizar en la aplicación, ya que el dispositivo móvil no permitía realizar fotos a una resolución tan baja utilizando la cámara normal. Por lo tanto, se escribió un código que permitía realizar fotos en la resolución escogida para la aplicación mediante el uso de la aplicación IP WebCam durante la retransmisión en tiempo real (*Fig. 58*).

```
import cv2

url = "http://192.168.43.1:8080/video"
cap = cv2.VideoCapture(url)

leido, frame = cap.read()

if leido == True:
    cv2.imwrite("foto XX.png", frame)
    print("Foto tomada correctamente")
else:
    print("Error al acceder a la cámara")

"""
    Finalmente liberamos o soltamos la cámara
"""
cap.release()
```

Figura 58. Código para hacer fotos con la resolución de la aplicación

Elaboración propia

Con todas las imágenes del patrón cargadas en el programa, únicamente se deberán leer una a una, pasarlas a escala de grises y encontrar las esquinas del tablero (*Fig. 59*). Si estas esquinas se han encontrado de manera correcta, se realizará el guardado en los vectores, tanto de los puntos objeto 3D como de los puntos imágenes (que serán refinados antes de ser guardados para mejorar la precisión de los píxeles). Finalmente se dibujarán los resultados obtenidos en las diferentes imágenes tomadas (*Fig. 60*).

```python
for image in imagenes:

    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Busca las esquinas del tablero
    # Si se encuentra el número de esquinas necesarias, ret = true
    ret, corners = cv.findChessboardCorners(gray, Dimension_Tablero, None)

    """
    Si se encuentran las esquinas en cada imagen,
    se refinan las coordenadas en pixeles para
    conseguir mayor precisión y las mostramos
    en las imagenes del tablero
    """

    # Si se han encontrado esquinas, se añaden los puntos objeto y los puntos imagen
    if ret == True:

        P_objeto.append(objp)

        # Se refinan las coordenadas de los píxeles para los puntos en 2D encontrados
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        P_imagen.append(corners2)

        # Se dibujan las esquinas y se muestran por pantalla
        cv.drawChessboardCorners(img, Dimension_Tablero, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)
```

Figura 59. Código de extracción y dibujado de las esquinas del patrón

Elaboración propia

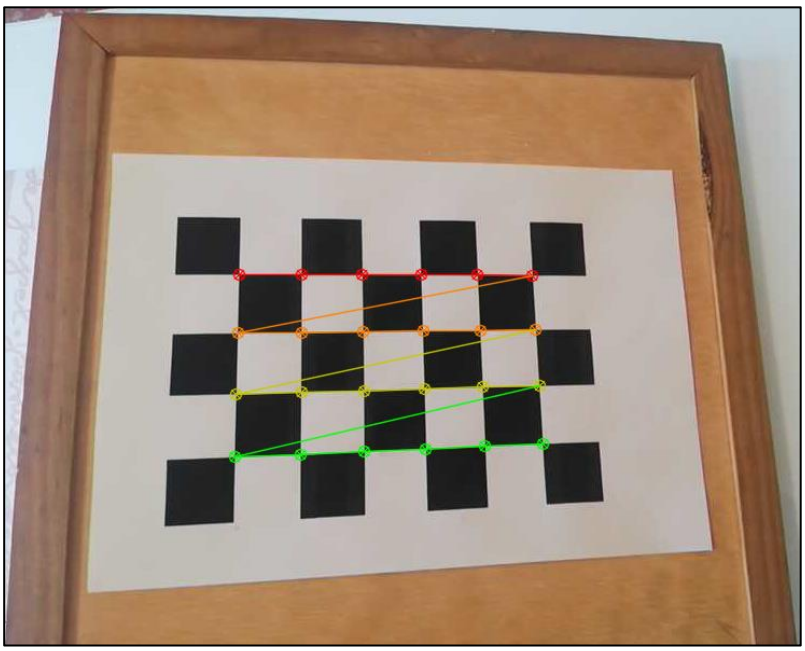


Figura 60. Ejemplo resultado calibración

Elaboración propia

Con los vectores completados, tanto de los puntos en 3D como en 2D, únicamente quedaría el proceso de relación entre ambos conjuntos, extrayendo de esta manera los parámetros necesarios para realizar la calibración. Para llevar a cabo este proceso utilizaremos la función *calibrateCamera()*.

```
ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(P_objeto, P_imagen, Dimension_Frame, None, None)
```

Figura 61. Función de extracción de parámetros de calibración

Elaboración propia

De la función (*Fig. 61*) se obtiene la matriz intrínseca (cameraMatrix), los parámetros de distorsión (dist) y la matriz extrínseca (rotación y translación). Además, se ha obtenido una estimación del error de calibración para observar la exactitud del proceso.

Finalmente se llama a la función *getOptimalNewCameraMatrix()* para obtener la nueva matriz a escala, y se aplica la función *undistort()* con todos los resultados obtenidos para eliminar la posible distorsión que pueda aparecer en la imagen (*Fig.* 62).

```
newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,h), 1, (w,h))
frame = cv.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)
```

Figura 62. Código de eliminación de la distorsión

Elaboración propia

Como la calibración de la cámara se ha realizado en un código aparte al principal, se ha hecho uso de la biblioteca *pickle*, que permite guardar tanto la matriz intrínseca resultante como los parámetros de distorsión para poder utilizarlos en otros archivos de código (*Figs. 63 y 64*).

```
# Guardar las variables (parámetros de la cámara) en un archivo para poder usarla en otro código
with open('parametros_camara.pkl', 'wb') as archivo:
    pickle.dump((cameraMatrix,dist), archivo)
```

Figura 63. Uso de la librería pickle para el guardado de parámetros

Elaboración propia

```
# Cargar las variables (parámetros de distorsión) desde el archivo
with open('parametros_camara.pkl', 'rb') as archivo:
    cameraMatrix,dist = pickle.load(archivo)
```

Figura 64. Uso de la librería pickle para cargar los parámetros de la cámara

Elaboración propia

5.2.5 Segmentación y filtrado de color

Para llevar a cabo la detección de los elementos del proyecto se han utilizado técnicas de segmentación y filtrado de color, que ha permitido diferenciar los componentes en la ejecución de la aplicación de seguimiento.

Antes de comenzar la explicación del proceso seguido para realizar los procedimientos de segmentación, se van a introducir los diferentes espacios de color, que corresponden a las distintas formas de representar la gama de colores universal. Los modelos mayoritariamente conocidos son:

- **Espacio RGB:** Muy utilizado en aplicaciones como adquisición y generación de colores en TV, impresoras… Este modelo está basado principalmente en la combinación aditiva de tres colores primarios: rojo (R), verde (G) y azul (A).
- **Espacio CMYK:** Este modelo constituye el inverso al RGB, ya que está basado en un modelo sustractivo de mezcla con tres colores primarios: cian (C), magenta (M) y amarillo (Y), aunque también añade el negro (K).
- **Espacio HSV:** Las siglas de este modelo corresponden a los componentes de matiz (H), saturación (S) y valor (V). El matiz es el tono de color, la saturación hace referencia a la intensidad y el valor está relacionado con el brillo.

Para el proyecto, el espacio que se va a utilizar para construir las máscaras de filtrado será el HSV, ya que el tono del color que se va a utilizar viene dado directamente con el matiz (H), por lo es muy útil en tareas de procesamiento de imágenes para detectar objetos por color (*Fig. 65*).

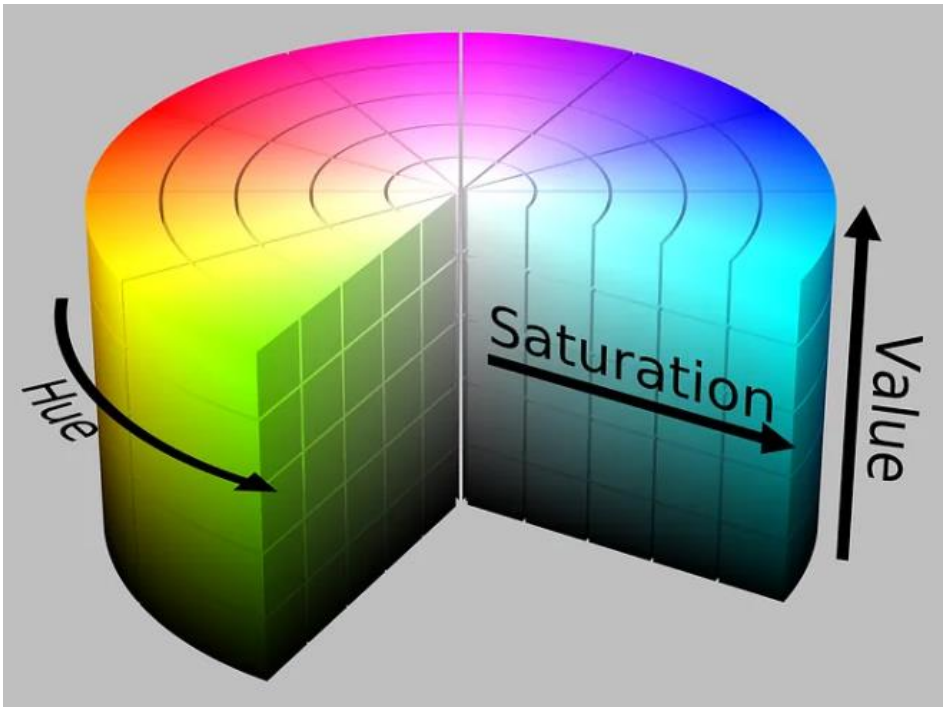


Figura 65. Representación del espacio de color HSV

Con el espacio de color escogido, el primer paso consiste en la creación de las máscaras de color, es decir, el rango de valores en los que queremos que detecte los diferentes colores. Por lo tanto, para construir los intervalos, se deben tener en cuenta los tres parámetros de HSV, cuya variación se puede ver en la *Fig. 66*:

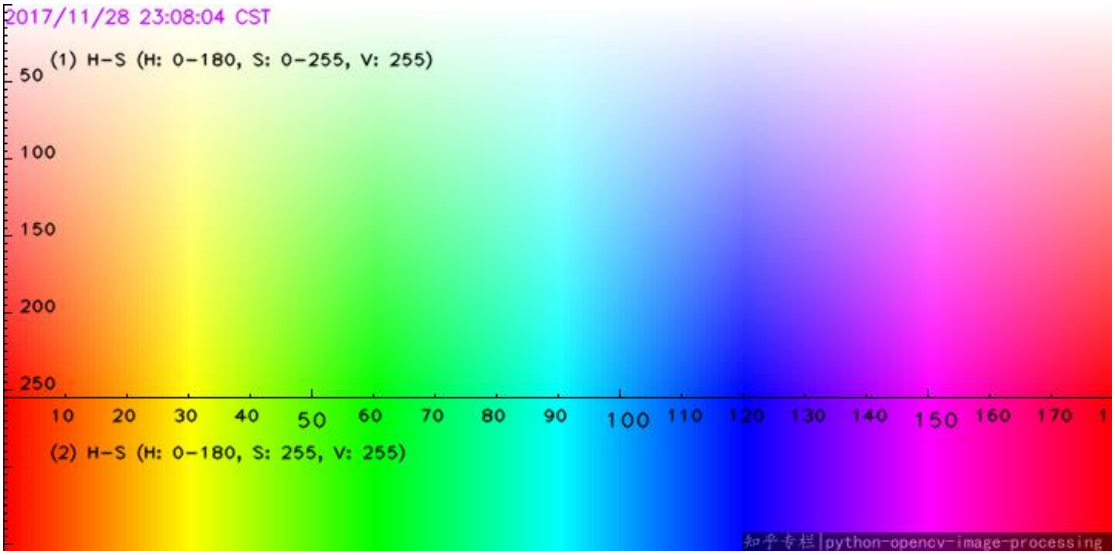


Figura 66. Variación de color en el espacio HSV

Para la aplicación, se tenían que buscar colores que fueran fácilmente reconocibles para la cámara pero que no se pudieran confundir con colores del entorno de ejecución. De esta forma, las máscaras se han construido a partir de la biblioteca *numpy* y los valores escogidos han sido los siguientes:

```
#Procesado de imagen mediante filtrado de color (robot guía)
azulBajo = np.array([100,100,70],np.uint8)
azulAlto = np.array([125,255,255],np.uint8)

#Procesado de imagen mediante filtrado de color (robot seguidor (posición))
rojoBajo = np.array([170,50,40],np.uint8)
rojoAlto = np.array([185,255,255],np.uint8)

#Procesado de imagen mediante filtrado de color (robot seguidor (orientación))
verdeBajo = np.array([45,25,50],np.uint8)
verdeAlto = np.array([75,255,255],np.uint8)
```

Figura 67. Máscaras para filtrado de color

Elaboración propia

Como se observa en el código (*Fig. 67*), el filtrado se realiza en tres colores diferentes: azul para el objetivo a seguir; el rojo para la posición del robot seguidor; y el verde, que se utilizará para indicar la orientación.

Para implementar el filtrado a partir de los vectores creados, se deberán usar dos funciones, ambas pertenecientes a la biblioteca *OpenCV*, que se utilizarán principalmente para el procesado de la imagen:

- *cvtColor():* Esta función permitirá realizar el cambio de espacio de color visto en la imagen. El espacio de color de captura de imagen de Python es el BGR, por lo que se deberá realizar el cambio a HSV para implementar las máscaras anteriormente declaradas.

- *inRange():* Con la imagen en HSV, esta función permite realizar el filtrado de color, estableciendo un rango inferior y superior. Todo lo que quede fuera de ese baremo de valores, se eliminará de la máscara y, por lo tanto, únicamente quedarán los objetos a detectar.

De esta manera, las funciones se utilizarían en los tres colores siguiendo las indicaciones descritas anteriormente. A continuación, se puede observar las implementaciones realizadas para los colores (*Figs. 68, 69 y 70*):

```
#Procesado para el robot (guía) (azul)
frameHSV_1 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
mask_azul = cv.inRange(frameHSV_1,azulBajo, azulAlto) #Filtrado de color
```

Figura 68. Implementación del filtro para el color azul

Elaboración propia

```
#Procesado para el robot seguidor (posición) (rojo)
frameHSV_2 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
mask_rojo = cv.inRange(frameHSV_2,rojoBajo, rojoAlto) #Filtrado de color
```

Figura 69. Implementación del filtro para el color rojo

Elaboración propia

```
#Procesado para el robot seguidor (orientación) (verde)
frameHSV_3 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
mask_verde = cv.inRange(frameHSV_3,verdeBajo, verdeAlto) #Filtrado de color
```

Figura 70. Implementación del filtro para el color verde

Elaboración propia

De esta manera, se conseguirá realizar el filtrado de los tres colores, permitiendo eliminar el fondo y aislar el objeto que se desea detectar. Para visualizar esto con claridad, se va a mostrar un ejemplo en el que se podrá ver el resultado. Primeramente, se capta una imagen en la cámara:

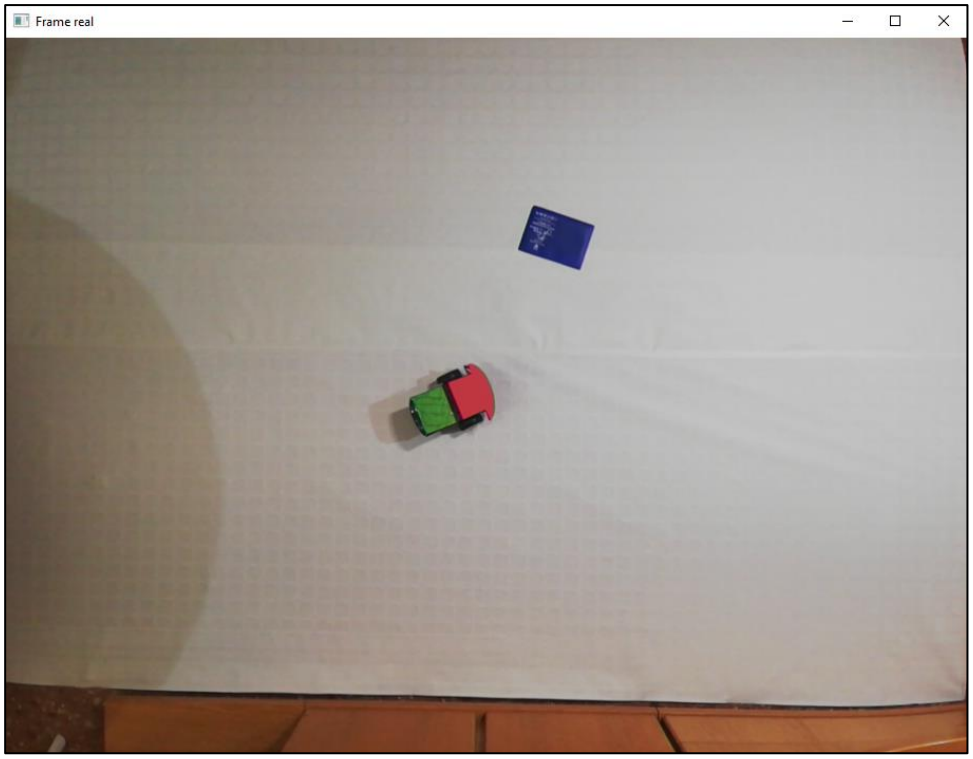


Figura 71. Imagen captada por la cámara (ejemplo)

Elaboración propia

De esta forma, se llevará a cabo el filtrado de los tres colores siguiendo la imagen que se observa en la *Fig. 71*, donde se puede visualizar el robot con las dos pegatinas ya colocadas (roja y verde) y el objetivo (azul).

Para poder distinguirlos de manera correcta, se mostrarán las máscaras de cada color en diferentes ventanas (*Figs.72, 73 y 74*), visualizando así los objetos detectados y la eliminación completa del entorno. Las siguientes imágenes muestran cómo se elimina el fondo:

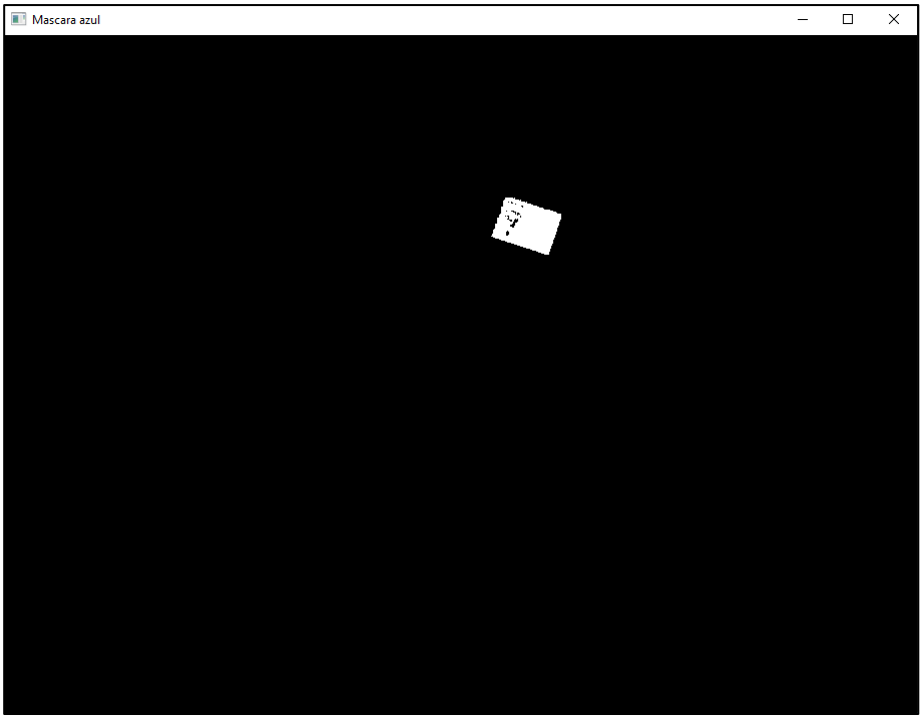


Figura 72. Visualización de la máscara para el filtrado del color azul

Elaboración propia

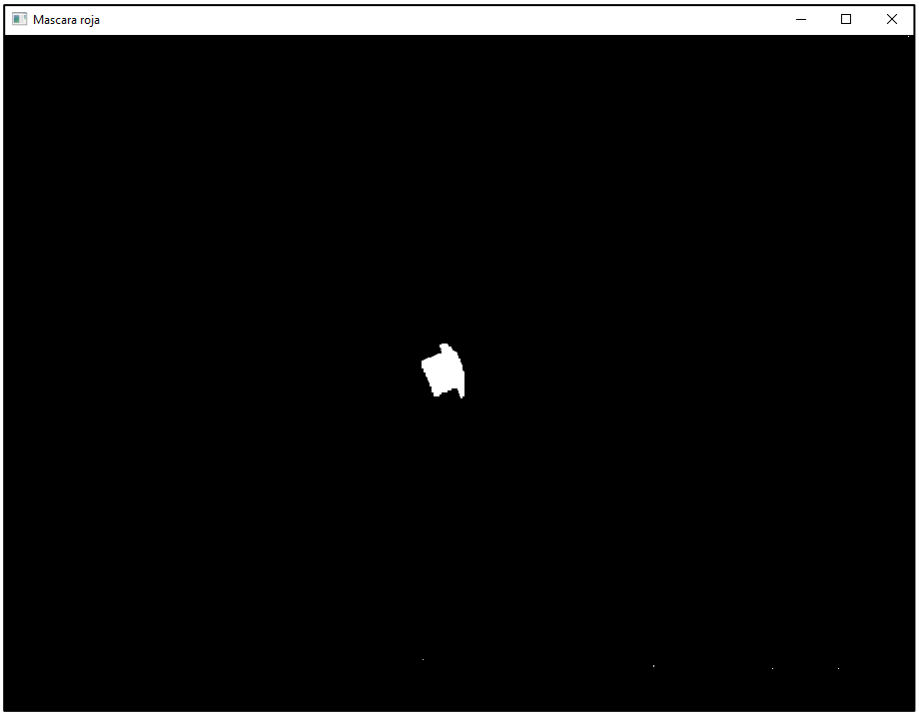


Figura 73. Visualización de la máscara para el filtrado del color rojo

Elaboración propia

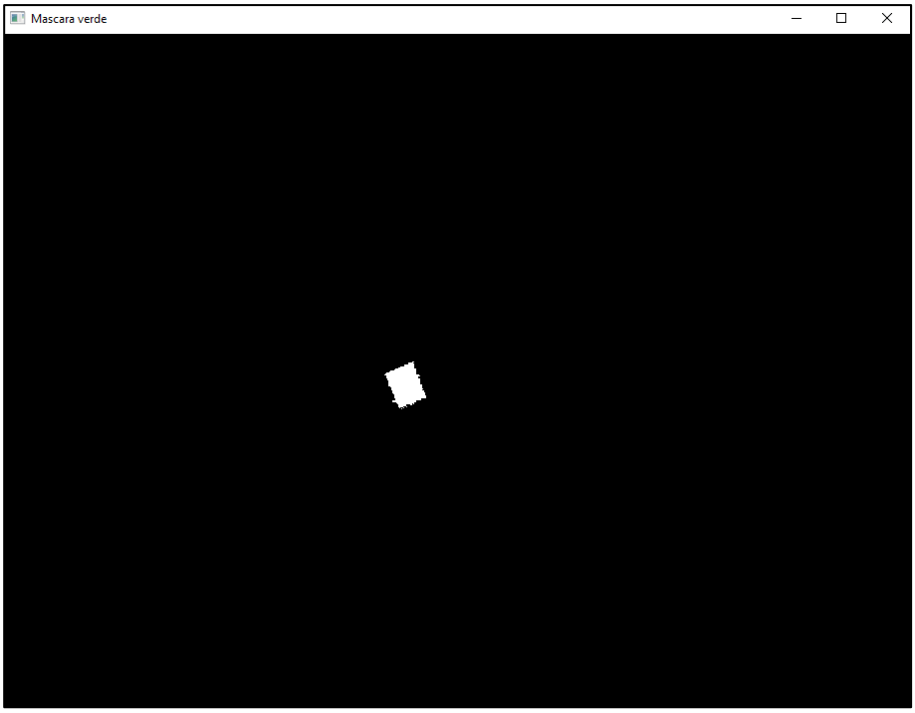


Figura 74. Visualización de la máscara para el filtrado del color verde

Elaboración propia

5.2.6 Morfología matemática: Eliminación de ruido

En el ámbito que compone el procesamiento de imágenes, la morfología trata las transformaciones que permiten analizar y manipular la forma y estructura de los objetos detectados en una imagen. Basado en la teoría de conjuntos, el objeto es alterado mediante un elemento estructural (EE) o kernel con el objetivo de clarificar su identificación y convertirlo en un elemento más fácil de detectar.

Dentro de la morfología matemática existen diversas operaciones, aunque en este proyecto se han utilizado las más básicas:

- **Erosión:** Esta operación consiste en reducir el tamaño de los elementos detectados. Su funcionamiento se basa en el desplazamiento de un elemento estructural por la imagen, y si todos los pixeles de este coinciden con la detección, el píxel central se mantiene; en cambio, si uno de los pixeles del kernel no coincide con los del elemento que se ha detectado, el pixel central se elimina (se convierte en fondo).

- **Dilatación:** En este caso, funciona al contrario que la erosión. Si la matriz estructural coincide con algún píxel de la detección, el pixel central del kernel se convertirá en parte de la detección. Este método se utiliza para aumentar el tamaño del objeto detectado.
- **Apertura:** Transformación que consiste en combinar una erosión seguida de una dilatación. De esta forma se consigue eliminar falsas detecciones que puede haber en el entorno (erosión), sin quitar pixeles al objeto detectado, ya que la dilatación final los recuperará.
- **Cierre:** Se invierte el orden de la apertura, y por lo tanto la dilatación se realiza primero y después se aplica una erosión. En este caso se consiguen cerrar posibles huecos que pueda tener la detección del objeto en su interior, y después se eliminan los pixeles que se le han añadido en el contorno del elemento durante la dilatación inicial.

En el proyecto que se está realizando, estas operaciones se van a utilizar para la eliminación de ruido, tanto exterior a la detección como interior a esta. Para ello se utilizarán el cierre y la apertura en las máscaras extraídas para los tres colores.

Primeramente, se deberá definir un elemento estructural mediante el cual se aplicarán las diferentes operaciones de morfología matemática. La definición vendrá dada según el tamaño de la matriz que se quiere aplicar a la imagen. En este caso, se ha creado un elemento estructural (*Fig. 75*), utilizando la librería *numpy*, que se corresponde con una matriz con todos los valores a 1:

```
#Kernel para realizar un filtrado de ruido
kernel = np.ones((10,10),np.uint8)
```

Figura 75. Elemento estructural para la morfología matemática

Elaboración propia

Para visualizar un ejemplo de las operaciones que se van a realizar, se va a ejemplificar y a visualizar su aplicación con la máscara azul vista en el final del apartado anterior.

La primera operación que se va a realizar en la imagen corresponde con un cierre (*Fig. 76*). Esto permitirá eliminar ruido que pueda existir en el interior de la detección, es decir, pequeños píxeles en el objeto que no se detecten de manera correcta pasarán a detectarse como objeto y no como fondo. El resultado de esta primera operación con la detección de uno de los colores sería el siguiente:

Figura 76. Resultado operación de cierre en la máscara azul

Elaboración propia

El siguiente paso tendrá como objetivo principal eliminar el ruido externo a la detección, por lo tanto, se realizará una operación de apertura, ya que con la erosión inicial que hace en la imagen se eliminarán posibles falsas detecciones que se puedan llevar a cabo. En este caso, como no aparece ruido externo de color azul no tendrá prácticamente efecto. El resultado de esta operación se observa en la *Fig. 77.*

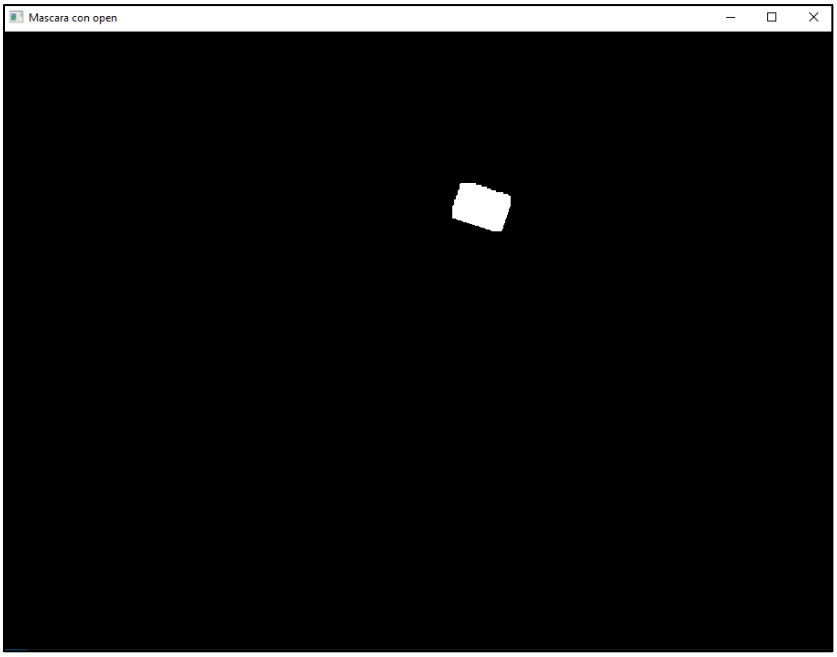


Figura 77. Resultado operación de apertura en la máscara azul

Elaboración propia

El conjunto de este procedimiento se ha llevado a cabo para la detección de los tres colores presentes en la aplicación, aunque únicamente se han mostrado ejemplos para uno de ellos. Por tanto, la declaración de todas las funciones utilizadas será la siguiente:

```
mask_close_azul = cv.morphologyEx(mask_azul, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
mask_def_azul = cv.morphologyEx(mask_close_azul, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
contornos_guia,_ = cv.findContours(mask_def_azul,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)
```

Figura 78. Morfología matemática aplicada en la máscara azul

Elaboración propia

```
mask_close_rojo = cv.morphologyEx(mask_rojo, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
mask_def_rojo = cv.morphologyEx(mask_close_rojo, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
contornos_seguidor_pos,_ = cv.findContours(mask_def_rojo,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)
```

Figura 79.Morfología matemática aplicada en la máscara rojo

Elaboración propia

```
mask_close_verde = cv.morphologyEx(mask_verde, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
mask_def_verde = cv.morphologyEx(mask_close_verde, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
contornos_seguidor_orient,_ = cv.findContours(mask_def_verde,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)
```

Figura 80. Morfología matemática aplicada en la máscara verde

Elaboración propia

Como se puede observar en la *Figs. 78, 79 y 80*; únicamente se utiliza la función *morphologyEx()*, pero dependiendo de la operación que se va a realizar, se cambia el segundo parámetro de entrada. Finalmente se guardan los contornos que cumplen todas las condiciones de filtrado en las variables que aparecen con el nombre de *contornos_*.

5.2.7 Estudio de la detección: estimación de la posición

En este apartado se realizará el estudio de las detecciones anteriormente realizadas mediante los diferentes filtrados, pero únicamente centrándose en la estimación de la posición.

Para analizar el procesamiento de imagen llevado a cabo para cada color, se deben crear tres bucles distintos, que realicen un barrido de los contornos detectados en cada color estudiado. En cada uno de estos bucles se calculará el área de cada contorno y se realizará un filtrado con el objetivo de eliminar contornos demasiado pequeños que se hayan podido detectar. De esta operación únicamente deben quedar los contornos de los elementos que interesan en la aplicación.

Posteriormente a este proceso, se realiza el cálculo del centroide del contorno encontrado. Este procedimiento se hace a partir de la función creada específicamente para ello y que tiene como parámetro de entrada el contorno al que se le quiere realizar la estimación. El cálculo del centroide está basado en la utilización de la función *moments()*, de la librería de OpenCV, que permite recopilar un diccionario de los momentos calculados en la imagen. Los momentos son una medida particular que indica una dispersión de puntos y que se definen de la siguiente forma:

$$M_{ij} = \sum_{x} \sum_{y} x^i y^j I(x, y)$$

En la función anterior “x” e “y” son las coordenadas de un píxel, y la función I(x,y) indica su intensidad, que en una máscara binaria únicamente puede valer 0 o 1. De esta forma, a partir de este cálculo se pueden definir tres momentos:

$$M_{00} = \sum_x \sum_y x^0 y^0 I(x,y) = \sum_x \sum_y I(x,y)$$

$$M_{10} = \sum_x \sum_y x^1 y^0 I(x,y) = \sum_x \sum_y xI(x,y)$$

$$M_{01} = \sum_x \sum_y x^0 y^1 I(x,y) = \sum_x \sum_y yI(x,y)$$

Cada momento hace referencia al tipo de coordenada al que afecta. Con estos tres momentos se puede pasar a calcular el centroide del contorno estudiado:

$$x = \frac{M_{10}}{M_{00}} \qquad y = \frac{M_{01}}{M_{00}}$$

Con toda la teoría explicada, la implementación en Python para la utilización de la función quedaría de la siguiente forma:

```python
#Funcion para cálcular el centroide de la detección
def Calculo_Centroide  (Contorno):

    M = cv.moments(Contorno)
    if (M['m00']==0): M['m00'] = 1
    x = int(M['m10']/M['m00'])
    y = int(M['m01']/M['m00'])
    return x, y
```

Figura 81. Función para el cálculo del centroide

Elaboración propia

La declaración de esta función (*Fig. 81*) estará presente en los tres bucles de estudio, uno para cada color, y se obtendrán las coordenadas “x” e “y” de las detecciones, que corresponderían a la posición del objetivo (azul) y del robot seguidor (rojo), y a un color extra (verde) que permitirá estimar la orientación. Los bucles tendrán la estructura siguiente:

```python
#Para no dibujar todos los contornos y evitar ruidos:
for c in contornos_guia:     #Miramos los contornos uno por uno (azul)
    area = cv.contourArea(c)
    if area > 1000:         #comprobamos que el area es mayor a 1000
        nuevoContorno = cv.convexHull(c)       #Suaviza los contornos
        cv.drawContours(frame,[nuevoContorno], 0,(255,0,0), 3) #Dibujamos los contornos

        #Calculo del centro de gavedad
        x_guia, y_guia = Calculo_Centroide (nuevoContorno)

        #Dibujar el centro de gravedad y sus coordenadas
        cv.circle(frame, (x_guia,y_guia),7,(0,255,0),-1)
        font = cv.FONT_HERSHEY_SIMPLEX
        cv.putText(frame,'{},{}'.format(x_guia,y_guia),(x_guia+10,y_guia), font, 0.75, (0,255,0),1,cv.LINE_AA)
```

Figura 82. Estructura del bucle de análisis de los contornos

Elaboración propia

Como se puede observar en la *Fig. 82*, también se han añadido diversas funciones que permitirán realizar el dibujado de los contornos (*drawContours()*) en la ventana donde se está visualizando la aplicación, además de escribir las coordenadas calculadas anteriormente para cada elemento (*putText()*). Se puede observar en la *Fig. 83*:

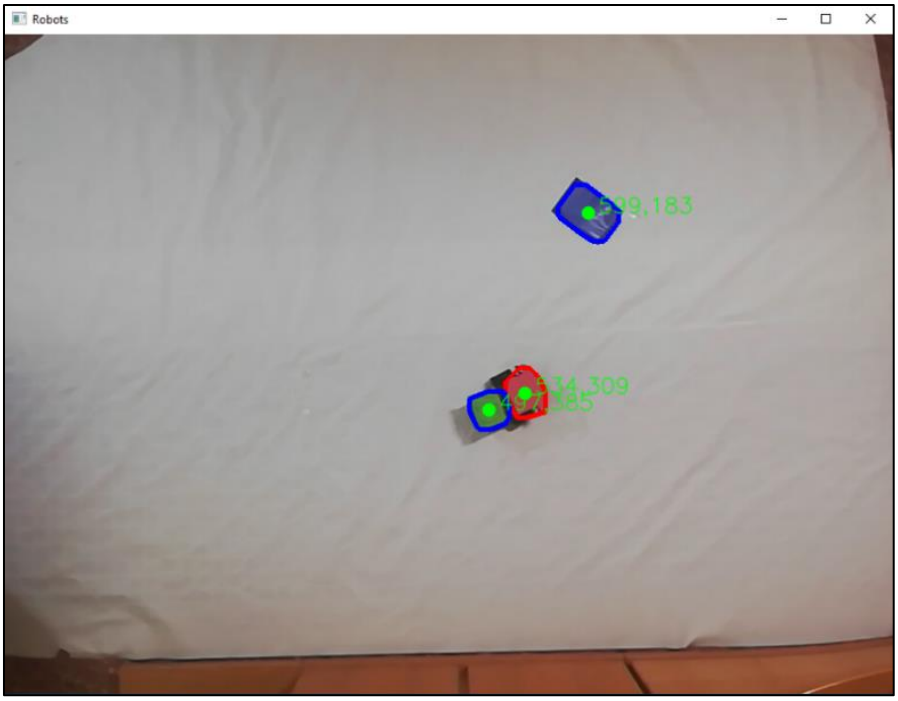


Figura 83. Visualización de las posiciones de los elementos

Elaboración propia

5.3 Control del robot

La última parte que tratar del proyecto está relacionada con el algoritmo de control que se ha seguido para que, una vez detectadas las posiciones de las pegatinas, el robot se aproxime al objetivo y se posicione lo más cercanamente posible a él.

De esta forma, se explicará la metodología que se va a seguir, los cálculos que se han necesitado para llevar a cabo el seguimiento y el código que se ha implementado, teniendo en cuenta también la comunicación entre la plataforma de procesamiento y el robot.

5.3.1 Planteamiento de la metodología

Durante el planteamiento inicial del modelo de control que se iba a seguir, surgieron diversas opciones, entre las cuales destacaba un control PID para los motores y un algoritmo de control basado en Lyapunov. Este tipo de algoritmo se implementaría realizando un control en cascada y permitiendo llevar al robot al objetivo a partir de las ecuaciones calculadas.

Sin embargo, durante las pruebas de este algoritmo se vieron números problemas relacionados principalmente con el hardware del robot:

- **Encoders imprecisos:** Se intentó implementar el uso de encoders ópticos para llevar a cabo mediciones de velocidad, ya que los motores escogidos no lo tenían incorporado. Durante la realización de pruebas se diagnosticaron imprecisiones y ruidos durante la implementación.
- **Zona muerta de los motores:** Aunque los motores alcanzan velocidades de 160 RPM, es cierto que en velocidades más bajas se encuentran limitados por su reducido rango de voltaje de funcionamiento (3 V – 6 V), por lo que en valores de PWM reducidos el motor entraba en lo que se conoce como zona muerta o de no funcionamiento, quedándose completamente parado.
- **Procesamiento de datos:** El microcontrolador debe poner en marcha los motores y recibir datos de velocidades de manera simultánea, teniendo por lo tanto una carga de trabajo muy superior y, en ocasiones, dejando de realizar mediciones de velocidades.

Con todos los problemas y limitaciones del hardware explicadas, se buscaron otras opciones que pudieran conseguir llevar a cabo la aplicación de seguimiento con éxito.

La metodología escogida consiste en la **alineación del robot con el objetivo**, donde haciendo uso únicamente del procesamiento de imagen en tiempo real, se tratará de alinear el robot con el objetivo para que únicamente deba avanzar hasta alcanzarlo. Para realizar este algoritmo de seguimiento, se deben seguir diversos pasos:

1. El robot deberá girar sobre sí mismo hasta alinearse con el objetivo, siempre teniendo en cuenta un rango de error introducido por el retraso de la cámara y que se tratará de solventar mediante código (*Fig. 84*).

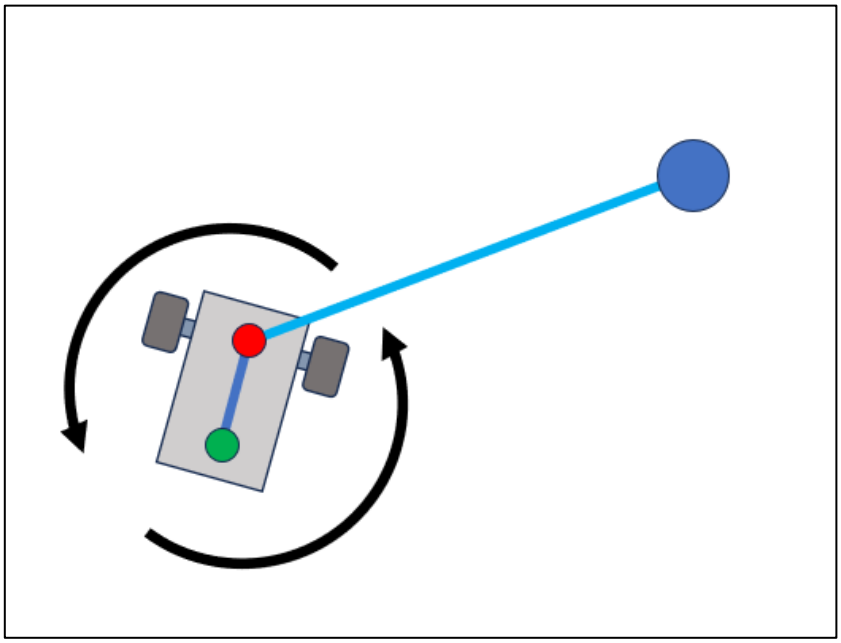


Figura 84. Paso 1 del algoritmo de control

Elaboración propia

2. Una vez se haya alineado con el objetivo, es decir, los ángulos de orientación del robot y el ángulo que forma con el objetivo sean iguales (dentro del rango estipulado), el robot deberá avanzar siguiendo un control P dependiente de la distancia (*Fig. 85*).

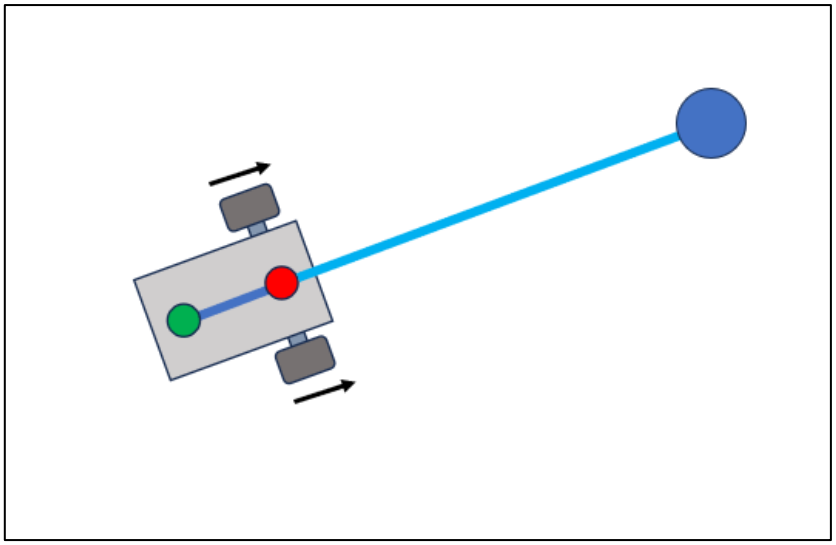


Figura 85. Paso 2 del algoritmo de control

Elaboración propia

3. Una vez alcance el objetivo, se detendrá hasta que la cámara detecte nuevamente un objetivo guía (*Fig. 86*).

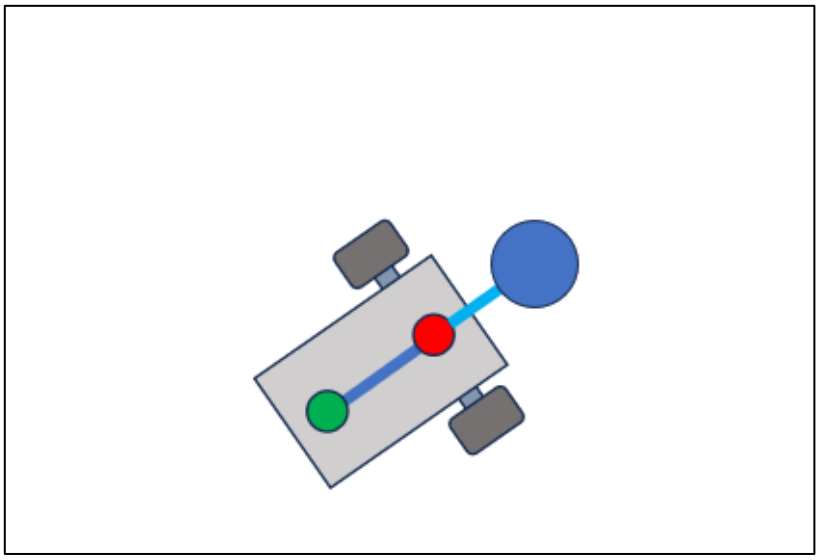


Figura 86. Paso 3 del algoritmo de control

Elaboración propia

En caso de que el robot se desvíe, volverá al paso 1 para corregir la trayectoria y alinearse de nuevo al objetivo. Se deberán enviar las velocidades correspondientes al robot utilizando la comunicación inalámbrica.

De esta forma, en los siguientes apartados se van a explicar los cálculos realizados para llevar a cabo el algoritmo explicado, además de su implementación en código y la comunicación con el robot.

5.3.2 Cálculo del algoritmo de control

Como se ha comentado anteriormente, en este apartador se tratarán todos los cálculos necesarios para llevar a cabo el control del robot con éxito. Los parámetros que se necesitan para realizar el seguimiento son: la orientación del robot, el ángulo que forma el robot con el objetivo y la distancia que los separa (*Fig. 87*).

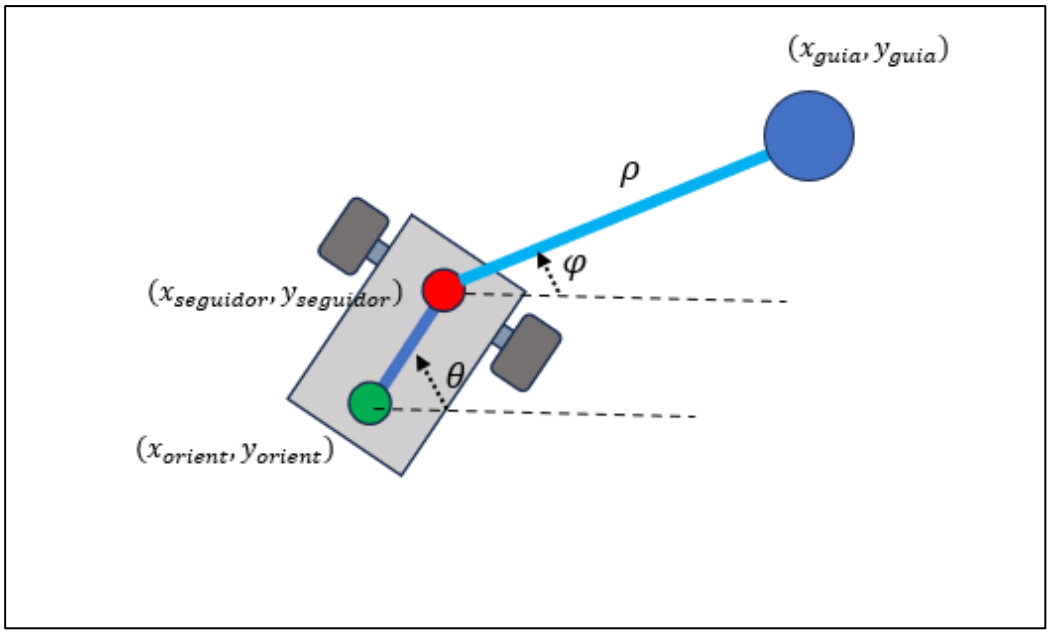


Figura 87. Representación de los parámetros a calcular en la aplicación

Elaboración propia

La estrategia principal que se ha seguido para poder extraer los parámetros del control ha sido la utilización de las técnicas de procesamiento de imagen, utilizando las detecciones anteriormente explicadas (filtrado de color). De esta manera, para llevar a cabo el cálculo de la orientación del robot, se trazará una recta entre ambas marcas presentes en la parte superior del dispositivo, que establecerán el ángulo en el que está mirando el vehículo móvil.

Una de las primeras metodologías que se pensó utilizar fue la diferenciación de formas. La estrategia consistía en la colocación de un cuadrado y un círculo verde en línea, de esta manera, una vez ya se había detectado el color a estudiar, se conseguiría diferenciar ambas formas y trazar la recta de orientación. Finalmente se decidió no implementar esta solución debido a que la elevación de la cámara dificultaba la correcta diferenciación de las figuras, detectando en ocasiones que ambas marcas eran iguales.

De esta forma, la metodología que finalmente se implementó fue la utilización de marcas de colores diferentes (*Fig. 88*), que permitiría realizar una detección mucho más sencilla para el algoritmo y donde la distancia de la cámara no supondría un problema durante la ejecución de la aplicación. La elección de colores para las marcas del robot serían por una parte el rojo, que estará posicionado en el eje de las ruedas y permitirá también la extracción de la posición del robot; y el color verde, que será la pegatina extra con la que se calculará la orientación.

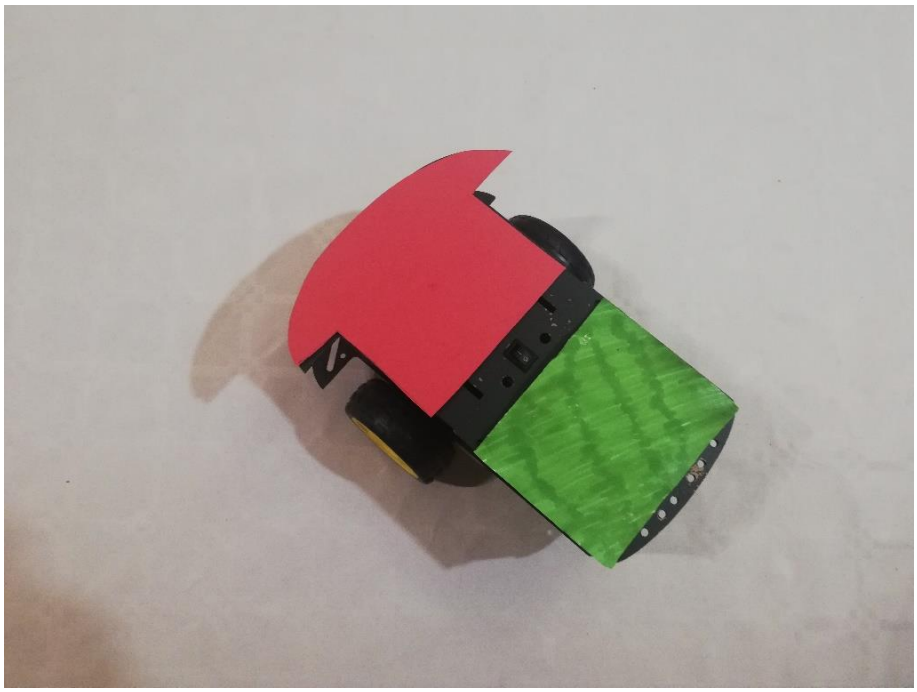


Figura 88. Robot con las pegatinas de detección colocadas

Elaboración propia

La detección de ambos colores se realizaría con el método explicado en el apartado 5.2.7, utilizando un bucle por cada color e implementando la función para calcular el centroide a los contornos correspondientes. Una vez se han obtenido las coordenadas (x, y) de ambas marcas posicionadas en el robot, se hace uso de la trigonometría para extraer el ángulo que forman ambas pegatinas. En este caso, la operación utilizada es la tangente, más específicamente *atan2()*, que permite realizar el cálculo de cualquier ángulo respecto de la horizontal, independientemente del cuadrante en el que éste se encuentre.

```
#Cálculo orientación del robot
if x_seguidor and x_orientacion != 0:
    theta = math.atan2(y_orientacion-y_seguidor, x_seguidor-x_orientacion) #Ángulo robot
    if theta < 0:
        theta = theta + (2*math.pi) #Ajuste para ángulos negativos
    theta = round (theta,2) #Ajuste a 2 decimales
    cv.line(frame,(x_seguidor,y_seguidor),(x_orientacion,y_orientacion),(255,0,0),2) #Se dibuja
```

Figura 89. Cálculo de la orientación del robot

Elaboración propia

Como se observa en la *Fig. 89*, el cálculo del ángulo de orientación theta (θ) consiste en establecer un punto final, que serán las coordenadas del color rojo (x_seguidor, y_seguidor), y un punto inicial, que corresponderá con el color verde (x_orientacion, y_orientacion). El cálculo tiene la misma base que el arco tangente, es decir, la variación en la coordenada “y” dividida entre la variación de la coordenada “x”:

$$\theta = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right)$$

Si se observa detenidamente la implementación en el código, se puede observar que el cálculo de la diferencia en la coordenada “y” estaría al revés, ya que se está restando la posición final (y_seguidor) a la inicial (y_orientacion). Este cambio se debe al posicionamiento de los ejes de la cámara, donde el eje Y está invertido, lo que a su vez resulta que los movimientos en dirección positiva a lo largo del eje Y se traducen en un desplazamiento hacia abajo en el plano de la imagen (*Fig. 90*).

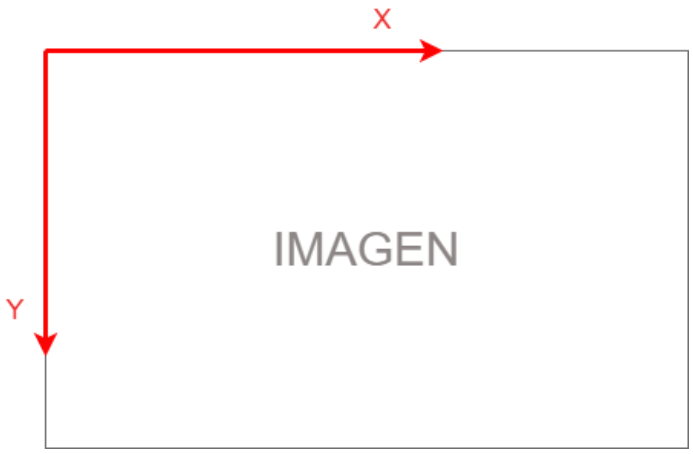


Figura 90. Representación de los ejes de captura de la imagen

Elaboración propia

Por otra parte, cabe destacar la implementación de un condicional para ángulos negativos, ya que la función *atan2()* devuelve un valor negativo si supera los 180º, y de esta forma se evitan ángulos menores a cero durante la ejecución.

De esta manera, el siguiente parámetro a calcular sería el ángulo que formaría el robot con el objetivo y al que se le ha denominado phi (φ). Para llevar a cabo este cálculo, únicamente se necesitarán las coordenadas de la posición del robot (x_seguidor, y_seguidor) y del objetivo (x_guia, y_guia), y aplicando nuevamente la función *atan2()* se conseguiría el ángulo mencionado (*Fig. 91*).

```
#Cálculo del ángulo al objetivo
if x_seguidor and x_guia != 0:
    phi = math.atan2(y_seguidor-y_guia, x_guia-x_seguidor) #Ángulo robot
    if phi < 0:
        phi = phi + (2*math.pi)
    phi = round(phi,2) #Ajuste a 2 decimales
    cv.line(frame,(x_seguidor,y_seguidor),(x_guia,y_guia),(255,255,0),2)
```

Figura 91. Cálculo del ángulo entre el robot y el objetivo

Elaboración propia

Nuevamente se observa que la variación en la coordenada “y” se encuentra invertida por la forma de los ejes de cámara, y también se ha implementado la condición para evitar ángulos negativos.

Por otra parte, el siguiente cálculo necesario para llevar a cabo el control sería la distancia que separa el robot y el objetivo, y que se ha denominado rho (ρ). En este caso únicamente se ha tenido que aplicar un sencillo cálculo para vectores, que se corresponde a su módulo.

$$|\vec{v}| = \sqrt{x^2 + y^2}$$

Primeramente, se deberá calcular el vector que forman ambos elementos, realizando la diferencia entre sus coordenadas en ambos ejes, y finalmente calcular el módulo de ese vector siguiendo la fórmula anterior:

$$rho = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

De esta manera, la implementación en el código consiste únicamente en seguir las fórmulas explicadas, teniendo en cuenta el nombre de las variables que se han utilizado durante todo el proyecto:

```
#Cálculo de la distancia al objetivo
rho = math.sqrt (pow(x_seguidor-x_guia,2)+pow(y_seguidor-y_guia,2))
rho = round (rho,2) #Ajuste a 2 decimales
```

Figura 92. Cálculo de la distancia entre el robot y el objetivo

Elaboración propia

Finalmente, para el cálculo de rho (*Fig. 92*), se ha tenido en cuenta que en la operación se obtiene un valor en píxeles y, para evitar complicaciones posteriores, se ha aplicado un factor de escala obtenido mediante un factor de conversión. En este caso, si se sabe que la dimensión horizontal de la imagen es de 912 píxeles (teniendo en cuenta el recorte de la imagen durante la calibración), se ha realizado una medida en la realidad del ancho de la aplicación, y por lo tanto el factor de escala quedaría de la siguiente forma:

$$Factor\ de\ escala = \frac{Medición\ en\ la\ realidad\ (m)}{Ancho\ imagen\ (píxeles)}$$

Su implementación es sencilla, ya que únicamente se introduce como un factor de conversión, realizando el cambio de unidades de píxeles a metros en el valor calculado de rho.

5.3.3 Implementación del algoritmo de control

Con los ángulos necesarios calculados y la distancia entre ellos determinada, se va a llevar a cabo la implementación del control del robot. La metodología que se va a seguir es sencilla, el robot girará sobre sí mismo hasta que *phi* y *theta* (ángulos calculados anteriormente) sean iguales o dentro de cierto valor de tolerancia, cuando eso suceda, el robot avanzará hasta llegar al objetivo.

Primeramente, se ha implementado una estructura *if ()…else*, ya que el robot solo avanzará si se cumple la condición relacionada con los ángulos, en caso negativo, el robot seguirá tratando de alinearse con el objetivo.

Una vez el robot haya cumplido la condición angular estipulada, se realizará una pequeña parada de los motores para posteriormente implementar un control P dependiente de la distancia que separa ambos elementos (*rho*) (*Fig. 93*). De esta forma, los pasos seguidos para diseñar el control son:

1. Se calcula el error entre el valor de referencia (se ha puesto un valor mínimo de 15 cm) y el valor de rho extraído del procesamiento de imagen, realizando la diferencia entre ambos.
2. Se establecen unos valores para las ganancias proporcionales de ambos motores, que en este caso se ha puesto un valor de 1.5 para ambos motores. Hay que destacar que se ha invertido el signo para establecer un criterio positivo en ambos actuadores.
3. Finalmente se ha implementado la función *map_rango()* para realizar una relación del resultado del control a PWM.

```python
#Ley de control
if (theta == phi) or (1 >= phi-theta >= -1):
    if parada < 5: #Espera de 5 ticks
        PWM_L = 0
        PWM_R = 0
        parada = parada + 1
        enviar_datos(PWM_R,PWM_L)
    else:
        e = ref_rho-(rho*factor_de_escala) #Error dependiente de la distancia

        #Cálculo de los valores de velocidad con el control P
        v_L = -k1 * e
        v_R = -k2 * e

        if (v_L<0.20):
            v_L=0

        if (v_R<0.20):
            v_R=0

        #Mapeo de los valores de PWM
        v_L = map_rango(v_L,0,4.29,70,255)
        v_R = map_rango (v_R,0,4.29,70,255)

        if (rho*factor_de_escala < 0.20):
            v_L=0
            v_R=0

        v_R = round (v_R,2)
        v_L = round (v_L,2)

        PWM_R = v_R
        PWM_L = v_L
        inicio = 0
```

Figura 93. Control P dependiente de la distancia rho

Elaboración propia

La función *map_rango()* es una función que permite establecer una relación entre máximos y mínimos de dos magnitudes distintas. Aunque es una función presente en Arduino, en Python se ha implementado como se puede observar en la *Fig. 94:*

```python
def map_rango(x, in_min, in_max, out_min, out_max):
  return (x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min
```

Figura 94. Función para mapear el valor de PWM

Elaboración propia

Si el robot no se encuentra correctamente colocado, el código estará constantemente en la sentencia *else* (*Fig. 95*), enviando al robot las velocidades pertinentes para girar sobre sí mismo, añadiendo un impulso inicial para vencer la fricción, pero teniendo en cuenta que, si la distancia entre ambos es menor de 20 cm, el robot deberá detenerse.

```
else: #Gira sobre si mismo hasta igualar ambos ángulos

    if inicio < 4 and rho*factor_de_escala > 0.20:
        PWM_L = 0
        PWM_R = 120
        inicio = inicio + 1
        enviar_datos(PWM_R,PWM_L)
    else:
        PWM_L = 0
        PWM_R = 65

    if rho*factor_de_escala < 0.20:
        PWM_L = 0
        PWM_R = 0

    parada = 0
```

Figura 95. Código de giro perpetuo para realizar la alineación

Elaboración propia

5.3.4 Comunicación inalámbrica: envío y recepción de datos

El control realiza los procedimientos necesarios para calcular y determinar las velocidades en PWM necesarias para cada rueda, pero el último paso para llevar a cabo la aplicación sería enviar los datos de forma inalámbrica al robot.

Para establecer la comunicación serial con el robot se ha inicializado el puerto serie del módulo Bluetooth, leyendo el puerto en el que accede el ordenador una vez se ha vinculado con el dispositivo. En este caso, se ha utilizado el COM7 con una velocidad de transmisión de 9600 baudios (*Fig. 96*).

```python
# Configuración del puerto serial
puerto = "COM7"
velocidad = 9600
ser = serial.Serial(puerto, velocidad)
```

Figura 96. Inicialización del puerto serial Bluetooth

Elaboración propia

El envío de datos se ha llevado a cabo utilizando la librería json explicada en el apartado 5.2.2. De esta manera, se ha creado una función en Python cuya implementación se puede ver en la *Fig. 97*:

```python
# Función para enviar los datos a Arduino
def enviar_datos(PWM_R, PWM_L):
    datos = {
        "PWM_R": PWM_R,
        "PWM_L": PWM_L,
    }
    mensaje = json.dumps(datos)
    ser.write(mensaje.encode())
    print("Datos enviados: PWM_R={}, PWM_L={}".format(PWM_R, PWM_L))
```

Figura 97. Función utilizada para enviar datos al robot

Elaboración propia

La función *enviar_datos()* recibe los valores de las velocidades de cada rueda en PWM a partir del control y los serializa en un mensaje de tipo json. Finalmente, este mensaje se envía por la comunicación serial inicializada anteriormente.

Por otra parte, se debe estudiar el código que se ha introducido en el interior del robot y que se ha programa en el lenguaje de Arduino. Este código únicamente debe leer el archivo json de forma que pueda extraer ambos datos de velocidad y finalmente aplicarlos a los motores por las salidas PWM correspondientes.

Primeramente, se deben declarar los pines de los motores como salida (*Figs. 98 y 99*), teniendo en cuenta que como se ha realizado la conexión, deben ser cuatro pines PWM y cada uno controla una rueda y un sentido de giro. Hay que destacar que, por el control que se ha implementado, los pines que llevan a cabo el retroceso de robot no se han utilizado.

```
#define DDel 5 //IN1
#define DAtr 6 //IN2
#define IDel 9 //IN3
#define IAtr 10 //IN4
```

Figura 98. Pines utilizados para los motores

Elaboración propia

```
void setup() {
  Serial.begin(9600);
  pinMode(DDel, OUTPUT);
  pinMode(DAtr, OUTPUT);
  pinMode(IDel, OUTPUT);
  pinMode(IAtr, OUTPUT);
}
```

Figura 99.Setup del código del robot

Elaboración propia

El siguiente paso que se debe llevar a cabo consiste en la lectura y descomposición del archivo json, que debe llegar del programa de Python. Para ello, se ha realizado el siguiente conjunto de código (*Fig. 100*):

```
void loop() {
  if (Serial.available() > 0) {
    char byteRecibido = Serial.read();

    if (byteRecibido == '{') {
      bytesRecibidos = 0; // Reiniciar el contador de bytes recibidos
      buffer[bytesRecibidos] = byteRecibido;
      bytesRecibidos++;
    } else if (byteRecibido == '}') {
      buffer[bytesRecibidos] = byteRecibido;
      buffer[bytesRecibidos + 1] = '\0'; // Agregar el carácter nulo al final del buffer
      bytesRecibidos++;

      // Analizar el JSON y obtener los valores
      DeserializationError error = deserializeJson(jsonDocument, buffer);
```

Figura 100. Código de lectura del archivo json en Arduino

Elaboración propia

La estrategia que se ha seguido es sencilla: cuando en el byte recibido llegue el carácter “{“, significa que ha llegado un conjunto de nuevos datos y reinicia el contador, y cuando llega en el byte el símbolo “}”, significa que los datos ya se han terminado y añade el carácter nulo. Con este procedimiento separa ambos valores de PWM para finalmente deserializar el archivo y guardar los valores en una variable *jsonDocument*.

A partir de la variable anterior y, en caso de que no haya habido errores durante el proceso de deserialización del archivo json, el proceso que queda consiste únicamente en extraer las velocidades y aplicarlas a los motores mediante un *analogWrite()* (*Fig. 101*).

```
if (!error) {
  float PWM_R = jsonDocument["PWM_R"];
  float PWM_L = jsonDocument["PWM_L"];

  //Introducción de los valores PWM a los motores
  Serial.print("Valor PWM derecha=");
  Serial.println (PWM_R);
  Serial.print("Valor PWM izquierda=");
  Serial.println (PWM_L);

  analogWrite (DDel,PWM_R);
  analogWrite (IDel,PWM_L);


} else {
  Serial.println("Error al decodificar el JSON");
```

Figura 101. Extracción e implementación de las velocidades en Arduino

Elaboración propia

5.3.5 Creación y visualización de gráficas

Una vez finalice la ejecución de la aplicación, se llevará a cabo el trazado de una serie de gráficas que permitirán visualizar la evolución de las velocidades de ambas ruedas y de la distancia que separa al robot del punto objetivo.

Para ello, el primer paso será crear los vectores que almacenarán los datos durante la ejecución del programa (*Fig. 102*):

```
#Arrays para las gráficas
distancias = []
tiempos = []
vel_R = []
vel_L = []
```

Figura 102. Vectores para almacenar los datos de las gráficas

Elaboración propia

Como se puede observar, los datos que se van a estar guardando corresponden con las distancias a las que se encuentra el robot del objetivo, los tiempos en los que se cogen los datos y las velocidades de ambas ruedas.

La toma de datos se deberá realizar una vez se haya aplicado el control diseñado, es decir, después de enviar los datos. De esta forma, a cada interacción del bucle *while()* se actualizarán los datos y se añadirán a los vectores anteriormente creados (*Fig. 103*).

```
# Agregar el valor actual a las listas
tiempo_actual = time.time()
distancias.append(rho*factor_de_escala)
vel_R.append(PWM_R)
vel_L.append(PWM_L)
tiempos.append(tiempo_actual - tiempo_inicio)  # Usar time.time() para obtener el tiempo actual
```

Figura 103. Actualización de datos en los vectores para graficar

Elaboración propia

Finalmente, se lleva a cabo la declaración de la función *dibujar_graficas()* que recoge los vectores completos y que se ejecuta al final del programa (una vez se ha salido del bucle *while()*). Esta función realizará las dos gráficas y las mostrará a la vez cuando se haya acabado la ejecución de la aplicación (*Fig. 104*).

```
#Función para dibujar las gráficas
def dibujar_graficas ():

    # Crear la primera figura y los ejes
    fig1, ax1 = plt.subplots()
    ax1.plot(tiempos, distancias) #Crea la gráfica con los datos actualizados
    ax1.set_xlabel('Tiempo (s)') #Titulo eje x
    ax1.set_ylabel('Distancia (m)') #Titulo eje Y
    ax1.set_title('Gráfica de evolución de la distancia') #Título

    # Crear la segunda figura y los ejes
    fig2, ax2 = plt.subplots()
    ax2.plot(tiempos, vel_R, label='PWM Derecha')
    ax2.plot(tiempos, vel_L, label='PWM Izquierda')
    ax2.set_xlabel('Tiempo (s)') #Título eje x
    ax2.set_ylabel('PWM') #Título eje Y
    ax2.set_title('Gráfica de evolución de velocidades PWM') #Título
    ax2.legend()  # Muestra la leyenda con los nombres de las variables

    # Mostrar ambas figuras en ventanas separadas
    plt.show()
```

Figura 104. Función para llevar a cabo las gráficas

Elaboración propia

6 Resultados y conclusiones

Para finalizar completamente la explicación del proyecto, se debe probar de manera experimental la aplicación para poder observar su funcionamiento y visualizar si cumple los objetivos propuestos inicialmente, es decir, que el robot alcance un punto objetivo utilizando técnicas de procesamiento de imagen.

6.1 Resultados experimentales

Con toda la parte de programación lista, se empiezan las pruebas de funcionamiento del proyecto para poder analizar los resultados.

Primeramente, se deberá llevar a cabo el montaje del escenario y del robot, teniendo en cuenta todas las explicaciones vistas anteriormente y las indicaciones seguidas en el pliego de condiciones (*Figs. 105 y 106*).

Figura 105. Trípode fijado con el móvil en posición cenital

Elaboración propia

Figura 106. Visualización del escenario con el mantel blanco colocado

Elaboración propia

Con todos los montajes listos, la primera parte que se debe tener en cuenta antes de comenzar es que el dispositivo móvil y el ordenador en el que se está llevando a cabo el procesamiento de imagen deben estar conectados a la misma red. Una vez se ha realizado la conexión de internet correspondiente, se activa la aplicación IP WebCam y se inicia la transmisión de imagen.

Cabe destacar también los resultados de la calibración de la cámara, que esta implementado en un código aparte, pero que se introduce en el código principal mediante un archivo de extensión *.pkl* de la librería *pickle*. Los resultados del parámetro *ret* son de 0.372, que como se encuentra dentro del intervalo 0.1 y 1.0, se considera una calibración correcta.

De esta manera, una vez comience a ejecutarse el programa de Python, que tardará algo de tiempo ya que debe conectarse al puerto serial del robot y verificar la conexión con la cámara, aparecerá la siguiente imagen en la pantalla:

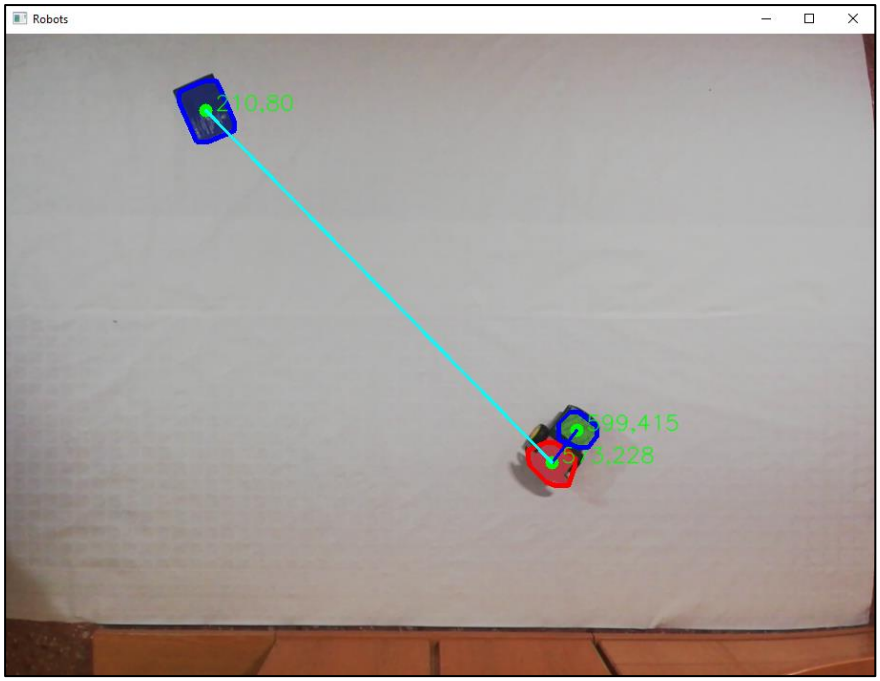


Figura 107. Ventana de visualización de la aplicación

Elaboración propia

En la imagen vista en la *Fig. 107* se observarán por una parte los tres elementos a detectar con sus respectivos colores: azul para el objetivo, rojo para la posición del robot y verde para realizar el cálculo de la orientación. Además, se pueden observar las posiciones en píxeles de las tres marcas en tiempo real y como varían, teniendo en cuenta que se han dibujado los contornos correspondientes a las tres detecciones. Por otra parte, también se ha preparado para que marque las líneas que forman los ángulos que se han calculado para llevar a cabo el control: azul oscuro para la orientación del robot y azul cielo para la recta que forma el ángulo entre el robot y el objetivo.

De esta manera, el algoritmo de control se pondrá en marcha realizando el envío de las velocidades al robot según los cálculos que se estén realizando durante la ejecución del programa. Inicialmente el robot comenzará a girar sobre sí mismo con el objetivo de alinearse, viéndose en pantalla como realiza las acciones de giro (*Fig. 108*).

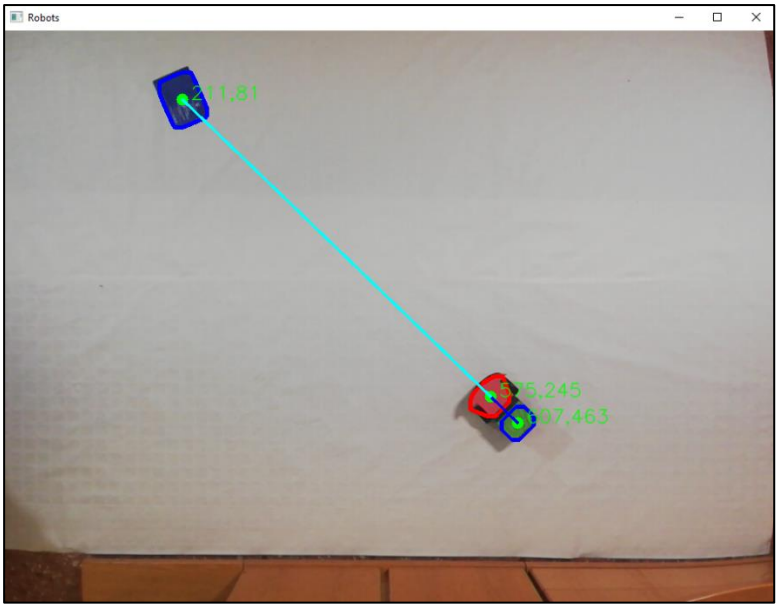


Figura 108. Visualización del robot alineado con el objetivo

Elaboración propia

Finalmente, el robot comenzará a avanzar siguiendo el control P establecido, hasta que la distancia con el objetivo sea lo suficientemente baja, obteniendo finalmente el siguiente resultado que se puede observar en la *Fig. 109:*

Figura 109. Visualización del robot en el objetivo

Elaboración propia

Para representar esto se ha implementado una gráfica (*Fig. 110*) que representa la disminución de la distancia del robot al objetivo a medida que la aplicación se lleva a cabo:

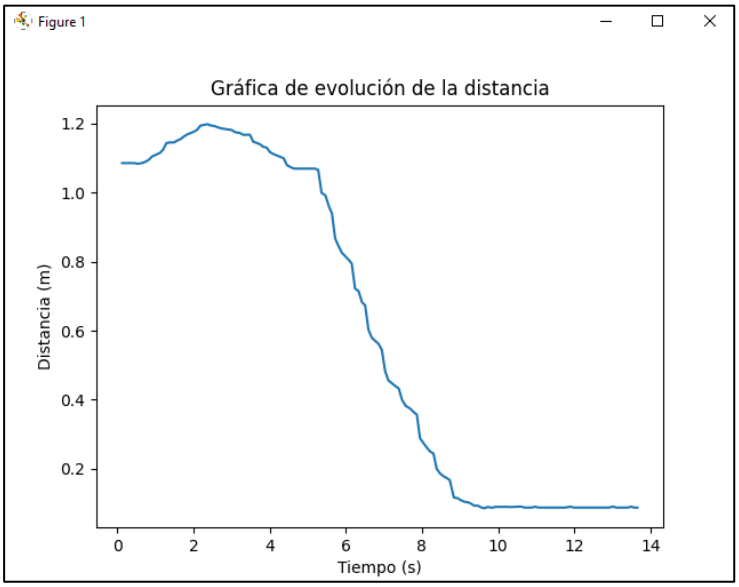


Figura 110. Gráfica de evolución de la distancia entre el robot y el objetivo

Elaboración propia

También se ha representado una gráfica (*Fig. 111*) donde se visualiza la evolución de las velocidades en PWM durante la ejecución control P y el giro de alineación:

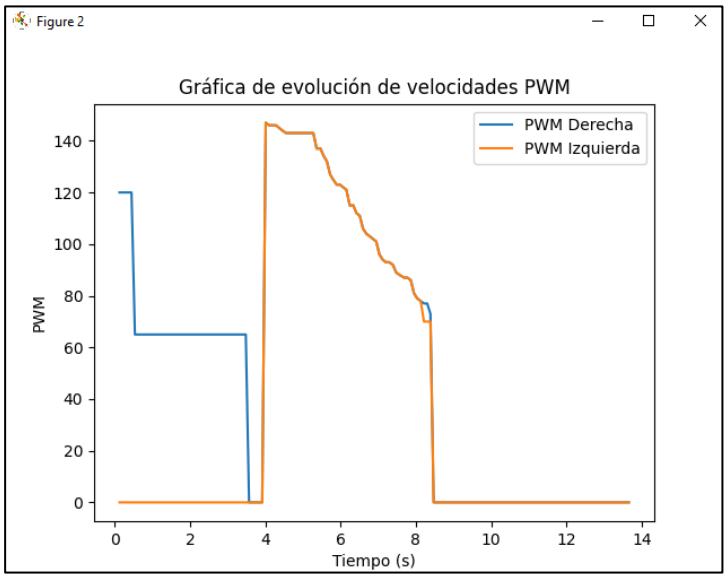


Figura 111. Gráfica de evolución de las velocidades de las ruedas

Elaboración propia

En esta gráfica se puede ver como inicialmente se introduce un impulso en la rueda derecha de 120, que tiene como objetivo arreglar una limitación importante que se presentaba durante la ejecución de la aplicación. Esta limitación ocurría constantemente al inicio del movimiento del robot, donde debido a la fricción del suelo y al bajo PWM de giro para la alineación, el robot no comenzaba a moverse; sin embargo, con un elevado impulso inicial, las ruedas obtenían inercia y la fricción se veía disminuida, consiguiendo el giro a velocidades bajas. Tras el impulso, las velocidades continúan siendo constantes en 0 para la rueda izquierda y bajan a 65 para la rueda derecha, hasta que el robot esta alineado, en ese momento se activa el control y se igualan las velocidades para avanzar. Finalmente, cuando el robot haya llegado al objetivo, las velocidades bajaran a cero y el robot parará.

Una vez el robot haya llegado al objetivo, se puede repetir el proceso tantas veces como se desee, ya que el programa se mantendrá activo hasta que el usuario quiera finalizar la ejecución y, por lo tanto, estará realizando procesamiento de imagen en tiempo real de manera continuada. Esto significa que, durante la ejecución, se puede ir moviendo el objetivo a donde el usuario lo prefiera y el robot lo irá siguiendo continuamente, independientemente de la localización en la que se coloque. De esta forma, se obtendrían gráficas como las siguientes:

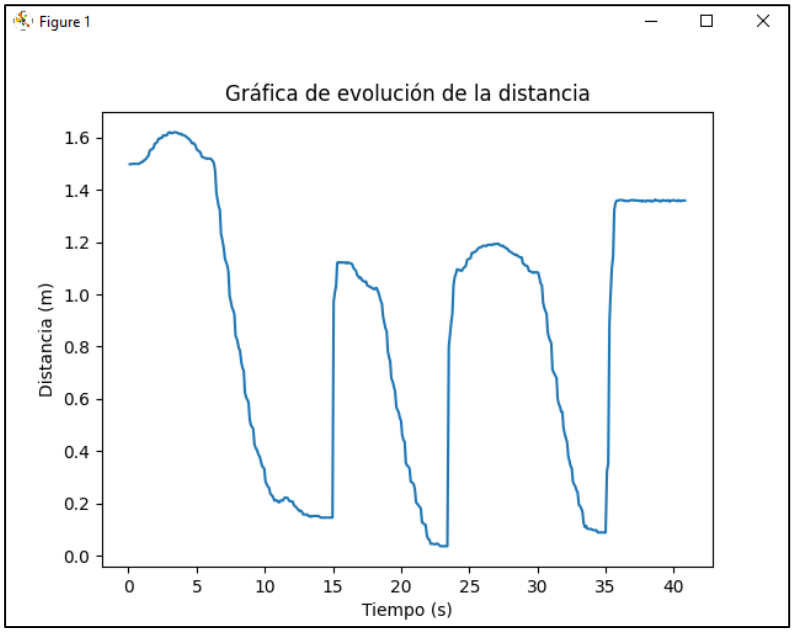


Figura 112. Visualización del funcionamiento de la aplicación de manera continuada (gráfica de distancia)

Elaboración propia

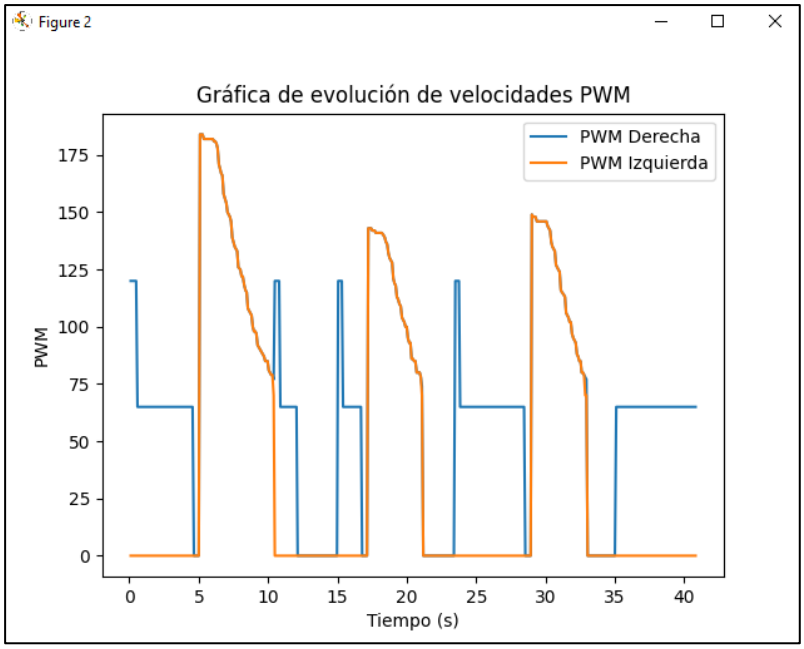


Figura 113. Visualización del funcionamiento de la aplicación de manera continuada (gráfica de velocidades)

Elaboración propia

Como se puede observar en las *Figs. 112 y 113,* tanto la distancia como las velocidades varían varias veces durante la ejecución, pero siempre mantienen la misma estructura, ya que se visualizan las diferentes zonas de la gráfica donde se lleva a cabo la alineación y los picos de velocidades donde el robot avanzará hasta el objetivo.

6.2 Conclusiones

Para finalizar el proyecto se van a comentar algunas conclusiones, en las que se incluirán aspectos relacionados con los resultados obtenidos, el hardware utilizado, el software y los problemas que se han tenido durante la ejecución del proyecto en los diferentes ámbitos.

Respecto a los resultados, se han podido cumplir con éxito los objetivos que se habían planteado inicialmente, consiguiendo llevar al robot móvil hasta el objetivo utilizando principalmente recursos relacionados con el procesamiento de imágenes. Por una parte, se ha llevado a cabo el montaje de un robot terrestre basado en un microcontrolador Arduino y diferentes módulos y elementos mecánicos. También se ha conseguido diseñar un sistema de visión y procesamiento de imagen capaz de transmitir imagen de forma inalámbrica, con el objetivo de llevar a cabo los procedimientos de detección de colores y aplicarles las técnicas de filtrado correspondientes para extraer los datos relevantes para la aplicación. Y finalmente, la implementación de un algoritmo de control que fuera capaz de calcular las velocidades necesarias para que el robot llegará al objetivo de manera precisa, estableciendo de forma simultánea una comunicación inalámbrica por Bluetooth entre el robot y el sistema de procesamiento de imagen.

Respecto al hardware utilizado, se han podido observar algunos problemas derivados de la calidad de los elementos mecánicos que se han utilizado. Los motores que se han escogido tenían cierta limitación a la hora de adaptar la velocidad, debido a su bajo rango de tensiones de alimentación, provocando que entrará en zona muerta muy fácilmente. Por esta limitación, en el inicio del giro del robot, se le debía aplicar un pequeño impulso para que los motores comenzarán a moverse. También debido a problemas en el hardware, no se han podido implementar soluciones más sofisticadas, sobre todo en el apartado del control del robot, donde se habían planteado opciones más complejas como un control basado en Lyapunov. Respecto al microcontrolador escogido, aunque es suficiente para la aplicación que se ha llevado a cabo, se podrían haber planteado otras opciones más adecuadas que habrían facilitado la comunicación con el programa de visión.

En el apartado del software, se debe destacar el reto que ha supuesto programar en Python, que es un lenguaje que no se había estudiado y que se ha tenido que aprender a utilizar durante la realización del proyecto. Sin embargo, la elección de este lenguaje y la utilización de OpenCV (una biblioteca ampliamente utilizada en aplicaciones de visión y detección) ha permitido obtener resultados ampliamente exitosos observándose estos en el apartado anterior.

Aunque en el proyecto ha habido numerosas complicaciones, tales como puede ser la transmisión de datos entre el robot y Python, que en ocasiones no se realizaba de manera correcta o surgían fallos; el problema principal que se ha observado ha sido el retraso de la imagen durante su transmisión al programa de procesamiento. Este retraso era verdaderamente complicado de eliminar, ya que se originaba a partir de la resolución (que debía ser elevada para obtener un gran rango de visión en la aplicación) y la cantidad de procesamiento que se le debía aplicar a la imagen. Por lo tanto, se decidió tenerlo en cuenta a la hora de implementar el control añadiendo un intervalo durante el proceso de alineación.

Las posibilidades del trabajo son prácticamente infinitas, ya que se podrían haber llevado a cabo incorporaciones como la detección y esquive de obstáculos mediante procesamiento, o incluso el seguimiento de trayectorias por medio de visión. No obstante, el proyecto cumple satisfactoriamente con los apartados estipulados teniendo en cuenta los conocimientos aprendidos y el tiempo disponible.

De esta manera, este proyecto ha permitido destacar la importancia y el futuro potencial que tiene la combinación de disciplinas como la robótica, la visión y el control, teniendo infinidad de aplicaciones en el campo de la robótica autónoma.

7 Bibliografía

7.1 Fuentes de información

Harmonic Drive SE (s. f.). *Robótica móvil.* Recuperado el 30/05/23.

https://harmonicdrive.de/es/glosario/robotica-movil

C. Márquez-Sánchez, R. Silva-Ortigoza, M. Marcelina-Aranda, M. Antonio-Cruz, C. Y. Sosa-Cervantes y J. R. García-Sánchez (2014, febrero 1). *Robots móviles de ruedas. Generalidades.* Recuperado el 30/05/23.

https://www.boletin.upiita.ipn.mx/index.php/ciencia/593-cyt-numero-45/1081-robots-moviles-de-ruedas-generalidades

INGENIERÍA Y MECÁNICA AUTOMOTRIZ (2019, octubre 21). *¿Qué es y cómo funciona el principio de Ackerman?* Recuperado el 30/05/23.

https://www.ingenieriaymecanicaautomotriz.com/que-es-y-como-funciona-el-principio-de-ackerman/

Ángel Eduardo Gil Pérez (2022, julio 8). *Róbotica móvil: Qué es y sus aplicaciones.* Recuperado el 30/05/2023.

https://openwebinars.net/blog/robotica-movil-que-es-y-sus-aplicaciones/

HiSoUR Arte Cultura Historia (s. f.). Robot patas. Recuperado el 30/05/2023.

https://www.hisour.com/es/legged-robot-43178/R

Héctor A. Moreno , Roque Saltaren , Lisandro Puglisi , Isela Carrera , Pedro Cárdenas, Cesar Álvarez (2014, marzo 19). *Robótica Submarina: Conceptos, Elementos, Modelado y Control.* Recuperado 29/05/2023.

https://digital.csic.es/bitstream/10261/111496/1/Saltaren_R_Robotica_Submarina_Revista_Iberoamericana_Automatica_Informatica_industrial_11_2014_3%E2%80%9319.pdf

CSCAZORLA (2011, mayo 11). *Robots móviles (IV)*. Recuperado el 30/05/23

https://www.xatakaciencia.com/robotica/robots-moviles-iv

Antonio González Sorribes (s. f.). *Asignatura de robótica aérea, «Introducción».*

Universidad Politécnica de Valencia; Presentaciones y PDF.

Beatriz Borella Petisco (2022, septiembre). *Introducción a la visión artificial. Procesos y aplicaciones.* Recuperado el 30/05/2023.

https://eprints.ucm.es/id/eprint/74914/1/beatriz_borrella_introduction.pdf

Infaimon (2020, enero 20). *Historia de la Visión Artificial y su evolución.* Recuperado 30/05/2023.

https://infaimon.com/blog/vision-2d-3d/historia-evolucion-vision-artificial/

IBM (s. f.). *¿Qué es la visión artificial?* Recuperado el 30/05/2023.

https://www.ibm.com/es-es/topics/computer-vision#:~:text=La%20visi%C3%B3n%20artificial%20es%20un,en%20funci%C3%B3n%20de%20esa%20informaci%C3%B3n

Arduino (s. f.). *Arduino nano características técnicas.* Recuperado el 02/06/23.

https://store.arduino.cc/products/arduino-nano

NayLampMechatronics (s. f.). *Driver puente H L298N 2A.* Recuperado el 03/06/23.

https://naylampmechatronics.com/drivers/11-driver-puente-h-l298n.html#:~:text=El%20driver%20puente%20H%20L298N,paso%20a%20paso%20bipolar%2Funipolar

Wikipedia (s. f.). *Puente H (electrónica).* Recuperado el 03/06/23.

https://es.wikipedia.org/wiki/Puente_H_(electr%C3%B3nica)#:~:text=Un%20Puente%20en%20H%20es,y%20como%20convertidores%20de%20potencia

Ferretrónica (s. f.). *Modulo encoder HC-020K. Sensor de Velocidad B83609.* Recuperado el 03/06/23.

https://ferretronica.com/products/modulo-encoder-hc-020k-sensor-de-velocidad-b83609

Robótica fácil (s. f.). *SPP-C Bluetooth (compatible con HC-06).* Recuperado el 04/06/23.

https://roboticafacil.es/prod/spp-c-bluetooth/

Electrocrea (s. f.). *Modulo bluetooth HC05 HC06 SPPC Reemplazo.* Recuperado el 04/06/23.

https://electrocrea.com/products/modulo-bluetooth-hc05-sppc-reemplazo

Bolaños Daniel J. (2019, septiembre 14). *Motores con rueda para Arduino.* Recuperado el 05/06/23.

https://www.bolanosdj.com.ar/MOVIL/ARDUINO2/MotoresConRuedasArd.pdf

Programarfacil (s. f.). *ESP32 CAM introducción y primeros pasos.* Recuperado el 05/06/23.

https://programarfacil.com/esp32/esp32-cam/#Caracteristicas_ESP32_CAM

Xataka móvil (2022, junio 13). *Huawei P20 Lite, análisis: review con características, precio y especificaciones.* Recuperado el 10/06/23.

https://www.xatakamovil.com/analisis/huawei-p20-lite-analisis-review-con-caracteristicas-precio-y-especificaciones

Gabriela Solano (2020, marzo 13). *¿Instalar Python? ¿Y OpenCV? En Windows [Fácil y rápido]*. Recuperado el 10/06/23

https://omes-va.com/instalacion-de-python-y-opencv-en-windows/

Visual Studio Code (s. f.). *Descarga del programa Visual Studio Code*. Recuperado el 10/06/23.

https://code.visualstudio.com/

Python (s. f.). *Descarga del lenguaje de programación Python.* Recuperado el 10/06/23.

https://www.python.org/downloads/release/python-3114/

Immune Institute (2022, marzo 17). *Librerías de Python, ¿qué son y cuáles son las mejores?* Recuperado el 11/06/23.

https://immune.institute/blog/librerias-python-que-son/

Kaustubh Sadekar, Satya Mallick (2020, febrero 25). *Camera Calibration using OpenCV.* Recuperado el 12/06/23.

https://learnopencv.com/camera-calibration-using-opencv/#disqus_thread

Ángel Rodas Jordá (s. f.). *Asignatura de robótica inteligente, «Segmentación de imágenes».*

Universidad Politécnica de Valencia; Presentaciones y PDF.

Gastón Di Giuseppe (2019, agosto 29). *Detección de objetos por colores en imágenes con Python y OpenCV.* Recuperado el 12/06/23.

https://medium.com/@gastonace1/detecci%C3%B3n-de-objetos-por-colores-en-im%C3%A1genes-con-python-y-opencv-c8d9b6768ff

7.2 Fuentes de las imágenes

Figura 1:

https://www.timetoast.com/timelines/la-robotica-3318d484-0f7b-4fe4-baf5-11bdbe097059

Figura 2:

https://www.boletin.upiita.ipn.mx/index.php/ciencia/593-cyt-numero-45/1081-robots-moviles-de-ruedas-generalidades

Figura 3:

https://www.redalyc.org/pdf/4026/402640448003.pdf

Figura 4:

https://www.ecorfan.org/handbooks/Handbooks_Proyectos_de_Ingenieria_Aplicada_TI/Handbooks_Proyectos_de_Ingenieria_Aplicada_TI_1.pdf

Figura 5:

https://www.cienciamx.com/index.php/tecnologia/robotica?start=45

Figura 6:

https://www.hispadrones.com/principiantes/aprendizaje-consejos/tipos-de-drones/

Figura 7:

https://www.timetoast.com/timelines/vision-artificial-ce1040a2-5f17-4f6b-a431-ffc710196b22

Figura 8:

https://www.edsrobotics.com/blog/sistemas-de-vision-artificial-tipos-aplicaciones/

Figura 10:

https://store.arduino.cc/products/arduino-nano

Figura 11:

https://store.arduino.cc/products/arduino-nano

Figura 12:

https://proyectosconarduino.com/modulos/arduino-nano-shield/

Figura 13:

https://createc3d.com/es/modulos-y-sensores/345-comprar-modulo-l298n-puente-h-para-arduino-controlo-motores-paso-a-paso-precio-oferta.html

Figura 14:

https://electronoobs.com/circuitos_tut1.php

Figura 15:

https://es.banggood.com/5pcs-SPPC-bluetooth-Serial-Adapter-Module-Wireless-Serial-Communication-from-Machine-AT-05-Replace-HC-05-HC-06-p-1465907.html?cur_warehouse=CN

Figura 16:

https://www.leroymerlin.ro/produse/baterii-si-acumulatori/878/baterie-alcalina-lexman-9v/40981

Figura 17:

https://es.aliexpress.com/item/4000419476085.html

Figura 18:

https://www.amazon.es/BeMatik-Portapilas-Conector-Recto-6LF22/dp/B07NDRN8WP/ref=sr_1_19?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=3Q1KEZPU8KI5H&keywords=conectores+pila+9v&qid=1685881841&sprefix=conectores+pila+9v%2Caps%2C123&sr=8-19

Figura 19:

https://encrypted-tbn3.gstatic.com/shopping?q=tbn:ANd9GcSvOH4-6xKjvTEvjVixywtzyLkphSD4bnZ8GDZz_Qc6mgt5dHXgZWd-z17ziVV-eQAIlaZUkSc27cLZDnmKCa00YxaIh_lz4KV9-RRrq1vfEt_3ygKoq01P4Q&usqp=CAE

Figura 21:

https://www.amazon.es/UMTMedia%C2%AE-Chasis-inteligente-Arduino-Raspberry/dp/B096XZ9VK2/ref=sr_1_3?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=EBL26W4FOLZK&keywords=AptoFun+2WD+Smart+Robot+Car&qid=1685921978&sprefix=aptofun+2wd+smart+robot+car%2Caps%2C93&sr=8-3

Figura 22:

https://robot.com.ve/product/motorreductor-amarillo-481-para-carrito-con-caja-reductora/

Figura 23:

https://createc3d.com/es/mas-componentes/657-comprar-rueda-para-construir-tu-robot-con-arduino-precio-oferta.html

Figura 24:

https://www.zamux.co/rueda-loca-tipo-rodachin

Figura 25:

https://www.az-delivery.de/es/products/esp32-cam-modul-esp32-wifi-bluetooth-modul-inklusive-kamera

Figura 26:

https://www.amazon.es/Wansview-vigilancia-bidireccional-Movimiento-Compatible/dp/B07QKVPF5C

Figura 27:

https://www.amazon.es/Huawei-P20-Lite-Memoria-Pantalla/dp/B07BHDC9V6

Figura 28:

https://www.amazon.es/TARION-tel%C3%A9fono-articulado-videograf%C3%ADa-fotograf%C3%ADa/dp/B08JCG4V5S/ref=sr_1_11?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=5F1JREWUWPPV&keywords=tripode+de+movil+silla&qid=1654775756&s=electronics&sprefix=tripode+de+movil+silla%2Celectronics%2C77&sr=1-11

Figura 44:

https://www.python.org/downloads/release/python-3114/

Figura 45:

https://www.mclibre.org/consultar/python/otros/python-instalacion.html

Figura 46:

https://www.ochobitshacenunbyte.com/2020/05/21/como-instalar-visual-studio-code-en-linux/

Figura 47:

https://code.visualstudio.com/

Figura 54:

https://www.xatakafoto.com/guias/distorsion-de-lente-vs-distorsion-de-la-perspectiva

Figura 65:

https://medium.com/@gastonace1/detecci%C3%B3n-de-objetos-por-colores-en-im%C3%A1genes-con-python-y-opencv-c8d9b6768ff

Figura 66:

https://omes-va.com/deteccion-de-colores/

ANEXOS

Anexo I. Justificación de los Objetivos y metas de Desarrollo sostenible

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.			X	
ODS 2. Hambre cero.			X	
ODS 3. Salud y bienestar			X	
ODS 4. Educación de calidad		X		
ODS 5. Igualdad de género			X	
ODS 6. Agua limpia y saneamiento			X	
ODS 7. Energía asequible y no contaminante	X			
ODS 8. Trabajo decente y crecimiento económico	X			
ODS 9. Industria, innovación e infraestructuras	X			
ODS 10. Reducción de las desigualdades.			X	
ODS 11. Ciudades y comunidades sostenibles.		X		
ODS 12. Producción y consumo responsables.		X		
ODS 13. Acción por el clima.			X	
ODS 14. Vida submarina.			X	
ODS 15. Vida de ecosistemas.		X		
ODS 16. Paz, justicia e instituciones sólidas.			X	
ODS 17. Alianzas para lograr objetivos.			X	

ANEXOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo II. Código para la verificación del funcionamiento de los motores

```
#define DDel 5 //IN1
#define IDel 9 //IN3

//Asignación de variables
unsigned long Tiempo_ref = 0;
unsigned int pwm = 0;
unsigned long tiempoSegundos = 0;
volatile unsigned muestreoActual = 0;

void setup() {
  Serial.begin(9600);
  Serial.print("Tiempo");Serial.print("        "); //Para observar los datos en el puerto serie
  Serial.println("PWM");

  pinMode(DDel, OUTPUT);
  pinMode(IDel, OUTPUT);

  pwm=10;
}

void loop() {
  analogWrite (DDel,pwm);   //Puesta en marcha del motor derecho
  analogWrite (IDel,pwm);   //Puesta en marcha del motor izquierdo

  muestreoActual = millis();                    // se asigna el tiempo de ejecuciòn a el muestreo actual

  if(muestreoActual>Tiempo_ref+10000)   //Contador de tiempo
  {
    Tiempo_ref = millis();
    tiempoSegundos = Tiempo_ref/1000;

    Serial.print ((int)tiempoSegundos); Serial.print ("            "); //Impresion de datos para verificació
    Serial.print(pwm);Serial.print("          ");


    if (pwm<250)
    {
      pwm=pwm+10;
    }else
    {
      pwm = 255;
    }

    Tiempo_ref = millis();

  }
}
```

ANEXOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo III. Código para la calibración de la cámara en Python

```python
import numpy as np
import cv2 as cv
import glob
import pickle


################ CÓDIGO PARA CALIBRACIÓN DE CÁMARA ##########################################
# Dimensiones del tablero y resolución de la cámara
Dimension_Tablero = (6,4)
Dimension_Frame = (960,720)


# Criterio de terminación
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)


# Definición de las coordenadas del mundo, como (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((Dimension_Tablero[0] * Dimension_Tablero[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:Dimension_Tablero[0],0:Dimension_Tablero[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 30
objp = objp * size_of_chessboard_squares_mm


# Vectores para almacenar los puntos objeto y los puntos imagen de todas las imagenes
P_objeto = [] # Vector para almacenar los puntos en 3D para cada imagen del tablero
P_imagen = [] # Vector para almacenar los puntos en 2D para cada imagen del tablero

# Extracción de las imagenes del directorio
imagenes = glob.glob('./imagenes/*.png')

for image in imagenes:

    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Busca las esquinas del tablero
    # Si se encuentra el número de esquinas necesarias, ret = true
    ret, corners = cv.findChessboardCorners(gray, Dimension_Tablero, None)

    """
    Si se encuentran las esquinas en cada imagen,
    se refinan las coordenadas en pixeles para
    conseguir mayor precisión y las mostramos
    en las imagenes del tablero
    """

    # Si se han encontrado esquinas, se añaden los puntos objeto y los puntos imagen
    if ret == True:

        P_objeto.append(objp)

        # Se refinan las coordenadas de los píxeles para los puntos en 2D encontrados
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        P_imagen.append(corners2)

        # Se dibujan las esquinas y se muestran por pantalla
        cv.drawChessboardCorners(img, Dimension_Tablero, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)


cv.destroyAllWindows()
```

```
############## CALIBRACIÓN ####################################################

"""
Realización de la calibración de la cámara
pasando el valor de los puntos en 3D (P_objeto)
y las correspondientes coordenadas de píxeles de
las esquinas detectadas (P_imagen)
"""

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(P_objeto, P_imagen, Dimension_Frame, None
, None)

print("ret : \n")
print(ret)
print("Camera matrix : \n")
print(cameraMatrix)
print("dist : \n")
print(dist)
print("rvecs : \n")
print(rvecs)
print("tvecs : \n")
print(tvecs)

# Guardar las variables (parámetros de la cámara) en un archivo para poder usarla en otro código
with open('parametros_camara.pkl', 'wb') as archivo:
    pickle.dump((cameraMatrix,dist), archivo)

# Reprojection Error
mean_error = 0

for i in range(len(P_objeto)):
    P_imagen2, _ = cv.projectPoints(P_objeto[i], rvecs[i], tvecs[i], cameraMatrix, dist)
    error = cv.norm(P_imagen[i], P_imagen2, cv.NORM_L2)/len(P_imagen2)
    mean_error += error

print( "total error: {}".format(mean_error/len(P_objeto)) )
```

ANEXOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo IV. Programa completo en Python y OpenCV para el control del robot mediante procesamiento de imagen

```
import cv2 as cv
import numpy as np
import pickle
import math
import serial
import json
import time
import matplotlib.pyplot as plt

#Resolución 960x720
tamaño_x = 960 #Pixeles en horizontal (w)
tamaño_y = 720 #Pixeles en vertical (h)
centro_x = 480 #Después de la calibración 456
centro_y = 360 #Después de la calibración 339

#Después de la calibración
tamaño_x_dc = 912 #Pixeles en horizontal (w)
tamaño_y_dc = 677 #Pixeles en vertical (h)
centro_x_dc = 456
centro_y_dc = 339

#Inicialización valores de locacalización y angulos
x_guia = 0
y_guia = 0
x_seguidor = 0
y_seguidor = 0
x_orientacion = 0
y_orientacion = 0
theta = 0
phi = 0
rho = 0

#Velocidades iniciales
PWM_R = 0
PWM_L = 0

#Valores de control
ref_rho = 0.15
k1 = 1.5
k2 = 1.5
inicio = 0
parada = 0

#Arrays para las gráficas
distancias = []
tiempos = []
vel_R = []
vel_L = []
```

```python
#Factor de escala para conversión pixeles a m
Medición_realidad = 2.50 #Medición horizontal en m
factor_de_escala = Medición_realidad/tamaño_x_dc

# Configuración del puerto serial
puerto = "COM7"
velocidad = 9600
ser = serial.Serial(puerto, velocidad)

# Inicializar el objeto de captura de video (IP cam viewer)
url = "http://192.168.43.1:8080/video"
cap = cv.VideoCapture(url)

#Funcion para cálcular el centroide de la detección
def Calculo_Centroide  (Contorno):

    M = cv.moments(Contorno)
    if (M['m00']==0): M['m00'] = 1
    x = int(M['m10']/M['m00'])
    y = int(M['m01']/M['m00'])
    return x, y

# Función para enviar los datos a Arduino
def enviar_datos(PWM_R, PWM_L):
    datos = {
        "PWM_R": PWM_R,
        "PWM_L": PWM_L,
    }
    mensaje = json.dumps(datos)
    ser.write(mensaje.encode())
    print("Datos enviados: PWM_R={}, PWM_L={}".format(PWM_R, PWM_L))

#Función para mapear el valor a PWM
def map_rango(x, in_min, in_max, out_min, out_max):
  return (x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

#Función para dibujar las gráficas
def dibujar_graficas ():

    # Crear la primera figura y los ejes
    fig1, ax1 = plt.subplots()
    ax1.plot(tiempos, distancias) #Crea la gráfica con los datos actualizados
    ax1.set_xlabel('Tiempo (s)') #Titulo eje x
    ax1.set_ylabel('Distancia (m)') #Titulo eje Y
    ax1.set_title('Gráfica de evolución de la distancia') #Título

    # Crear la segunda figura y los ejes
    fig2, ax2 = plt.subplots()
    ax2.plot(tiempos, vel_R, label='PWM Derecha')
    ax2.plot(tiempos, vel_L, label='PWM Izquierda')
    ax2.set_xlabel('Tiempo (s)') #Titulo eje x
    ax2.set_ylabel('PWM') #Título eje Y
    ax2.set_title('Gráfica de evolución de velocidades PWM') #Título
    ax2.legend()  # Muestra la leyenda con los nombres de las variables

    # Mostrar ambas figuras en ventanas separadas
    plt.show()
```

```
# Cargar las variables (parámetros de distorsión) desde el archivo
with open('parametros_camara.pkl', 'rb') as archivo:
   cameraMatrix,dist = pickle.load(archivo)


#Procesado de imagen mediante filtrado de color (robot guía)
azulBajo = np.array([100,100,70],np.uint8)
azulAlto = np.array([125,255,255],np.uint8)

#Procesado de imagen mediante filtrado de color (robot seguidor (posición))
rojoBajo = np.array([170,50,40],np.uint8)
rojoAlto = np.array([185,255,255],np.uint8)

#Procesado de imagen mediante filtrado de color (robot seguidor (orientación))
verdeBajo = np.array([45,25,70],np.uint8)
verdeAlto = np.array([75,255,255],np.uint8)

#Kernel para realizar un filtrado de ruido
kernel = np.ones((10,10),np.uint8)

tiempo_inicio = time.time()  # Tiempo actual en milisegundos

# Crear un objeto VideoWriter
fourcc = cv.VideoWriter_fourcc(*'XVID')  # Especificar el códec de video
fps = 30.0  # Fotogramas por segundo
width, height = tamaño_x_dc, tamaño_y_dc  # Tamaño de la ventana a grabar
video_out = cv.VideoWriter('output4.avi', fourcc, fps, (width, height))


while True:

     # Leer cada cuadro del video de streaming
    ret, frame = cap.read()

    # Corregir distorsión de la imagen
    h, w = frame.shape[:2]

    newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,
h), 1, (w,h))
    frame = cv.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)

    # Recorte de imagen
    x, y, w_final, h_final = roi
    frame = frame[y:y+h_final, x:x+w_final]
```

```
if ret == True:
        #Procesado para el robot (guía) (azul)
        frameHSV_1 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
        mask_azul = cv.inRange(frameHSV_1,azulBajo, azulAlto) #Filtrado de color

        mask_close_azul = cv.morphologyEx(mask_azul, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
        mask_def_azul = cv.morphologyEx(mask_close_azul, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
        contornos_guia,_ = cv.findContours(mask_def_azul,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)

        #Procesado para el robot seguidor (posición) (amarillo)
        frameHSV_2 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
        mask_rojo = cv.inRange(frameHSV_2,rojoBajo, rojoAlto) #Filtrado de color

        mask_close_rojo = cv.morphologyEx(mask_rojo, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
        mask_def_rojo = cv.morphologyEx(mask_close_rojo, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
        contornos_seguidor_pos,_ = cv.findContours(mask_def_rojo,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)

        #Procesado para el robot seguidor (orientación) (verde)
        frameHSV_3 = cv.cvtColor(frame,cv.COLOR_BGR2HSV) #Se pasa la imagen de BGR a HSV
        mask_verde = cv.inRange(frameHSV_3,verdeBajo, verdeAlto) #Filtrado de color

        mask_close_verde = cv.morphologyEx(mask_verde, cv.MORPH_CLOSE, kernel) #Eliminación ruido interno
        mask_def_verde = cv.morphologyEx(mask_close_verde, cv.MORPH_OPEN, kernel) #Eliminación ruido externo
        contornos_seguidor_orient,_ = cv.findContours(mask_def_verde,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_SIMPLE)


#-----------------------------Procesamiento de imagen-----------------------------------------------

        #Para no dibujar todos los contornos y evitar ruidos:
        for c in contornos_guia:     #Miramos los contornos uno por uno (azul)
            area = cv.contourArea(c)
            if area > 1000:         #comprobamos que el area es mayor a 1000
                nuevoContorno = cv.convexHull(c)       #Suaviza los contornos
                cv.drawContours(frame,[nuevoContorno], 0,(255,0,0), 3) #Dibujamos los contornos

                #Calculo del centro de gavedad
                x_guia, y_guia = Calculo_Centroide (nuevoContorno)

                #Dibujar el centro de gravedad y sus coordenadas
                cv.circle(frame, (x_guia,y_guia),7,(0,255,0),-1)
                font = cv.FONT_HERSHEY_SIMPLEX
                cv.putText(frame,'{},{}'.format(x_guia,y_guia),(x_guia+10,y_guia), font, 0.75, (0,255,0),1,cv
.LINE_AA)
```

```
for c in contornos_seguidor_pos:     #Miramos los contornos uno por uno (rosa)(EJE DE LAS RUEDAS)
    area = cv.contourArea(c)
    if area > 1000:         #comprobamos que el area es mayor a 1000
        nuevoContorno = cv.convexHull(c)       #Suaviza los contornos
        cv.drawContours(frame,[nuevoContorno], 0,(0,0,255), 3) #Dibujamos los contornos

        #Calculo del centro de gavedad
        x_seguidor, y_seguidor = Calculo_Centroide (nuevoContorno)

        #Dibujar el centro de gravedad y sus coordenadas
        cv.circle(frame, (x_seguidor,y_seguidor),7,(0,255,0),-1)
        font = cv.FONT_HERSHEY_SIMPLEX
        cv.putText(frame,'{},{}'.format(x_seguidor,h_final-y_seguidor),(x_seguidor+10,y_seguidor),
font, 0.75, (0,255,0),1,cv.LINE_AA)

for c in contornos_seguidor_orient:     #Miramos los contornos uno por uno (verde)
    area = cv.contourArea(c)
    if area > 500:         #comprobamos que el area es mayor a 500
        nuevoContorno = cv.convexHull(c)       #Suaviza los contornos

        cv.drawContours(frame,[nuevoContorno], 0,(255,0,0), 3) #Dibujamos los contornos

        #Calculo del centro de gavedad
        x_orientacion, y_orientacion = Calculo_Centroide (nuevoContorno)

        #Dibujar el centro de gravedad y sus coordenadas
        cv.circle(frame, (x_orientacion,y_orientacion),7,(0,255,0),-1)
        font = cv.FONT_HERSHEY_SIMPLEX
        cv.putText(frame,'{},{}'.format(x_orientacion,y_orientacion),(x_orientacion+10,y_orientacion
), font, 0.75, (0,255,0),1,cv.LINE_AA)

#Cálculo orientación del robot
if x_seguidor and x_orientacion != 0:
    theta = math.atan2(y_orientacion-y_seguidor, x_seguidor-x_orientacion) #Ángulo robot
    if theta < 0:
        theta = theta + (2*math.pi) #Ajuste para ángulos negativos
    theta = round (theta,2) #Ajuste a 2 decimales
    cv.line(frame,(x_seguidor,y_seguidor),(x_orientacion,y_orientacion),(255,0,0),2) #Se dibuja

#Cálculo del ángulo al objetivo
if x_seguidor and x_guia != 0:
    phi = math.atan2(y_seguidor-y_guia, x_guia-x_seguidor) #Ángulo robot
    if phi < 0:
        phi = phi + (2*math.pi)
    phi = round(phi,2) #Ajuste a 2 decimales
    cv.line(frame,(x_seguidor,y_seguidor),(x_guia,y_guia),(255,255,0),2)

#Cálculo de la distancia al objetivo
rho = math.sqrt (pow(x_seguidor-x_guia,2)+pow(y_seguidor-y_guia,2))
rho = round (rho,2) #Ajuste a 2 decimales
```

```
#-----------------------Cálculos del control del robot----------------------

        #Visualización de los valores de control
        print("phi:")
        print (phi)
        print("\n")
        print("theta:")
        print (theta)
        print("\n")
        print("rho:")
        print (rho*factor_de_escala)
        print("\n")

        #Ley de control
        if (theta == phi) or (1 >= phi-theta >= -1):
            if parada < 5: #Espera de 5 ticks
                PWM_L = 0
                PWM_R = 0
                parada = parada + 1
                enviar_datos(PWM_R,PWM_L)
            else:
                e = ref_rho-(rho*factor_de_escala) #Error dependiente de la distancia

                #Cálculo de los valores de velocidad con el control P
                v_L = -k1 * e
                v_R = -k2 * e

                if (v_L<0.20):
                    v_L=0

                if (v_R<0.20):
                    v_R=0

                #Mapeo de los valores de PWM
                v_L = map_rango(v_L,0,4.29,70,255)
                v_R = map_rango (v_R,0,4.29,70,255)

                if (rho*factor_de_escala < 0.20):
                    v_L=0
                    v_R=0

                v_R = round (v_R,2)
                v_L = round (v_L,2)

                PWM_R = v_R
                PWM_L = v_L
                inicio = 0
```

```
        else: #Gira sobre si mismo hasta igualar ambos ángulos

            if inicio < 4 and rho*factor_de_escala > 0.20:
                PWM_L = 0
                PWM_R = 120
                inicio = inicio + 1
                enviar_datos(PWM_R,PWM_L)
            else:
                PWM_L = 0
                PWM_R = 65

            if rho*factor_de_escala < 0.20:
                PWM_L = 0
                PWM_R = 0

            parada = 0

        print("PWM_R:")
        print (PWM_R)
        print("\n")
        print("PWM_L:")
        print (PWM_L)
        print("\n")

        #Envio de los valores PWM al Arduino
        enviar_datos(PWM_R,PWM_L)

        # Agregar el valor actual a las listas
        tiempo_actual = time.time() # Usar time.time() para obtener el tiempo actual
        distancias.append(rho*factor_de_escala)
        vel_R.append(PWM_R)
        vel_L.append(PWM_L)
        tiempos.append(tiempo_actual - tiempo_inicio)

        # Mostrar la imagen procesada en una ventana
        cv.imshow('Robots',frame)
        video_out.write(frame)

        #cv.imshow('Mascara azul',mask_azul)
        #cv.imshow('Mascara azul',mask_rojo)
        #cv.imshow('Mascara azul',mask_verde)
        #cv.imshow('Mascara con close',mask_close_azul)
        #cv.imshow('Mascara con open',mask_def_azul)

        # Salir del bucle si se presiona la tecla 's'
        if cv.waitKey(1) & 0xFF == ord('s'):
            break


#Dibujar las gráficas correspondientes
dibujar_graficas()

# Liberar el objeto de captura de video y cerrar las ventanas
ser.close() #Se cierra la conexión serial
video_out.release() #Acaba la grabación
cap.release()
cv.destroyAllWindows() #Destruye todas las ventantas
```

ANEXOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo V. Programa completo en Arduino para controlar el robot móvil

```
#include <ArduinoJson.h>

#define DDel 5 //IN1
#define DAtr 6 //IN2
#define IDel 9 //IN3
#define IAtr 10 //IN4

int PWM_R = 0;
int PWM_L = 0;

const int capacidadJson = JSON_OBJECT_SIZE(3); // Tamaño del objeto JSON
const int capacidadBuffer = 64; // Tamaño del buffer de recepción

StaticJsonDocument<capacidadJson> jsonDocument;
char buffer[capacidadBuffer];
int bytesRecibidos = 0;

void setup() {
  Serial.begin(9600);
  pinMode(DDel, OUTPUT);
  pinMode(DAtr, OUTPUT);
  pinMode(IDel, OUTPUT);
  pinMode(IAtr, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    char byteRecibido = Serial.read();

    if (byteRecibido == '{') {
      bytesRecibidos = 0; // Reiniciar el contador de bytes recibidos
      buffer[bytesRecibidos] = byteRecibido;
      bytesRecibidos++;
    } else if (byteRecibido == '}') {
      buffer[bytesRecibidos] = byteRecibido;
      buffer[bytesRecibidos + 1] = '\0'; // Agregar el carácter nulo al final del buffer
      bytesRecibidos++;

      // Analizar el JSON y obtener los valores
      DeserializationError error = deserializeJson(jsonDocument, buffer);


      if (!error) {
        float PWM_R = jsonDocument["PWM_R"];
        float PWM_L = jsonDocument["PWM_L"];

        //Introducción de los valores PWM a los motores
        Serial.print("Valor PWM derecha=");
        Serial.println (PWM_R);
        Serial.print("Valor PWM izquierda=");
        Serial.println (PWM_L);

        analogWrite (DDel,PWM_R);
        analogWrite (IDel,PWM_L);


      } else {
        Serial.println("Error al decodificar el JSON");
      }
    } else if (bytesRecibidos < capacidadBuffer - 1) {
      buffer[bytesRecibidos] = byteRecibido;
      bytesRecibidos++;
    }
  }
}
```

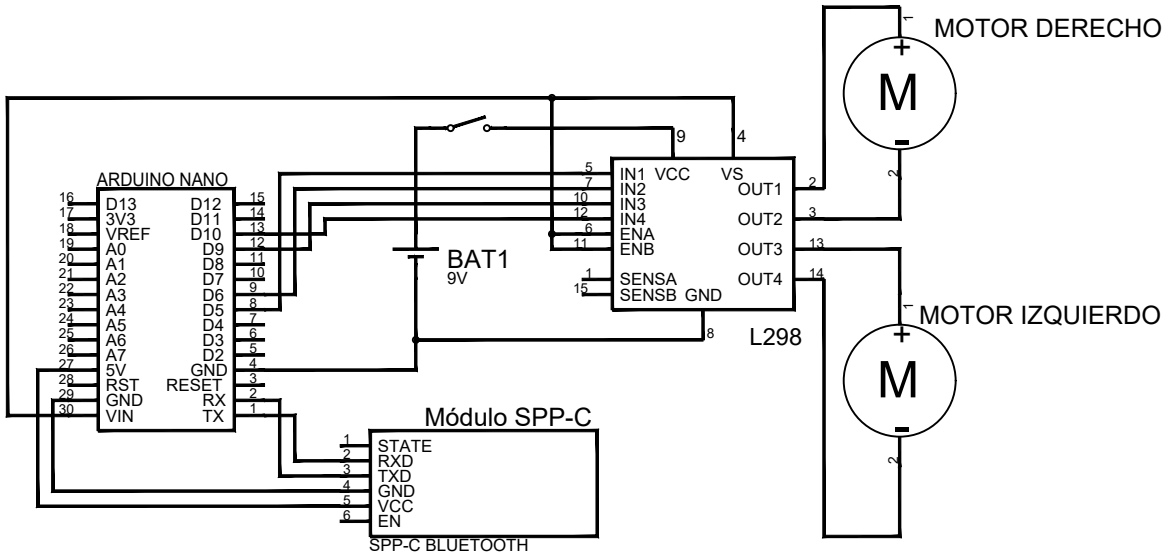
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO DE FIN DE GRADO

DOCUMENTO N.º 2. PLANOS

ÍNDICE



PROYECTO

TITULO: Circuito electrónico del robot			FECHA: 07/07/2023	
UNIVERSITAT POLITÈCNICA DE VALÈNCIA Escuela Técnica Superior de Ingeniería del Diseño	AUTOR: David Pérez Martínez	FIRMA:	ESCALA: -	UNIDADES: -
	REVISADO POR: Antonio Gonzalez Sorribes	FIRMA:	Nº DE PLANO: 1	

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO DE FIN DE GRADO

DOCUMENTO Nº 3.
PLIEGO DE CONDICIONES

ÍNDICE

1 Objeto

La presente especificación se refiere a la construcción y fabricación del robot móvil para una aplicación de seguimiento de robots móviles mediante un cámara IP y procesamiento de imagen. El robot cuenta con un chasis prefabricado, un microcontrolador y un controlador para motores DC, además de los correspondientes elementos mecánicos como los motores y las ruedas. El procesamiento de imagen se realizará utilizando Python y OpenCV, y la comunicación inalámbrica que se deberá establecer con el robot se llevará a cabo mediante un módulo Bluetooth SPP-C y el propio microcontrolador Arduino. La cámara IP utilizada corresponde con un dispositivo móvil Huawei P20 lite, que enviará la imagen directamente a Python. Finalmente, se establece un algoritmo de control para llevar al robot al objetivo.

2 Normativa de Carácter General

Se deberá realizar el proyecto acogiéndose a la siguiente normativa:

- **Norma UNE-EN ISO 13482:2014** sobre robots y dispositivos robóticos. Se tratan los requisitos relacionados con la seguridad de robots no industriales. Robots de asistencia personal no médica. Esta norma se encuentra limitada a robots únicamente terrestres.
- **Directiva 2003/108/CE** del Parlamento Europeo y del Consejo, de 8 de diciembre de 2003, por la que se modifica la Directiva 2002/96/CE sobre residuos de aparatos eléctricos y electrónicos (RAEE). Esta directiva modifica el artículo 9 de la Directiva 2002/96 (Financiación relativa a los RAEE no procedentes de hogares particulares). **Directiva 2002/96/CE** del Parlamento Europeo y del Consejo, de 27 de enero de 2003, sobre residuos de aparatos eléctricos o electrónicos.
- **Real Decreto 1801/2003**, de 26 de diciembre, sobre seguridad general de los productos. BOE-A-2004-511.

3 Materiales

3.1 Electrónica

3.1.1 Módulos

3.1.1.1 Arduino Nano

El microcontrolador que se ha utilizado corresponde a un Arduino Nano, basado en un chip de Atmega328 de 8 bits. Ofrece una disponibilidad de 14 pines digitales de entrada/salida, con 6 pines con posibilidad de ser utilizados con la tecnología PWM y 8 pines analógicos. Dispone de un puerto serie por hardware comunicado mediante los pines 0 y 1. Se puede alimentar por USB mini o por un pin de entrada V_{in} , teniendo en cuenta que funciona con un voltaje de trabajo de 5V.

El proceso de control de calidad de la placa electrónica se lleva a cabo por parte de la empresa distribuidora, que realiza las pruebas pertinentes para determinar su correcto funcionamiento.

3.1.1.2 Driver L298N

Para realizar el control de la velocidad de los motores DC se ha utilizado un controlador L298N de OcioDual. Este módulo basa su funcionamiento principalmente en la tecnología en puente H, permitiendo el control de la velocidad en ambos sentidos de giro para dos motores funcionando de manera simultánea utilizando PWM. Funciona con un voltaje de trabajo de 5-35 V y posee un consumo de corriente de 0-36 mA. Para su utilización se emplean unos jumpers que permiten elegir el modo de alimentación que se va a usar dependiendo del voltaje de entrada que se le va a suministrar.

El control de calidad del módulo corre a cargo del fabricante de la placa, que asegura su correcto funcionamiento antes de la venta.

3.1.1.3 SPP-C Bluetooth

Para la comunicación inalámbrica de los diferentes dispositivos del proyecto se ha utilizado un módulo bluetooth SPP-C. Este componente compatible con los módulos HC-05 tiene un soporte bluetooth V2.1+EDR con una banda de frecuencias de trabajo incluidas en un intervalo de 2.4 GHz - 2.48 GHz. Respecto a la alimentación, soporta valores de 5.6 V y dispone de seis pines, de los cuales se utilizarán únicamente cuatro, que corresponden con los pines de alimentación y los de comunicación serial (RX y TX).

El componente se ha sometido a los procesos de calidad correspondientes por la empresa de fabricación, asegurando el correcto funcionamiento.

3.1.2 Alimentación

3.1.2.1 Pila 9V

Para alimentar el robot móvil se ha utilizado una pila convencional de 9V de la marca LEXMAN, con las dimensiones establecidas para este tipo de pila (45 mm x 26 mm x 18 mm). La fuente de alimentación se conectará directamente a la entrada de voltaje del driver y se utilizará el jumper del que este dispone para voltajes de alimentación de 6-12 V. La pila consta de sus dos bornes, correspondientes al positivo y negativo, que se conectarán al circuito mediante un cable adaptado a ambos contactos.

El fabricante sería nuevamente el encargado de realizar el control de calidad asegurando tanto el voltaje como la capacidad de la pila.

3.1.2.2 Cableado

Para el conexionado de los diferentes componentes electrónicos se han usado cables Dupont de 20 cm de longitud. Son unos cables recubiertos con plástico ABS y con extremos en forma de jumper. Se han utilizado los tres tipos disponibles que combinan macho y hembra. También se ha empleado un cableado específicamente adaptado para la pila utilizada como fuente de alimentación, y un cable mini USB para el traspaso de código al microcontrolador del robot.

Los controles de calidad de los diferentes cables utilizados vienen determinados por la empresa fabricante.

3.1.2.3 Interruptor

Para poder apagar el robot sin tener que desconectar la correspondiente fuente de alimentación, se ha utilizado un interruptor de dos vías de dimensiones 14 mm x 9 mm. Su conexión se lleva a cabo en serie y mediante el accionador mecánico abre o cierra el circuito dependiendo de la posición en la que se deje colocado.

Las pruebas de funcionamiento se determinan por el fabricante.

3.2 Elementos mecánicos

3.2.1 Chasis

Los elementos que componen el chasis se corresponden con la base, las paredes y la parte superior. Tanto la base del robot como el techo son parte de un kit de construcción de robots de la marca DollaTek, que incluye una base perforada fabricada a partir de plástico acrílico con unas dimensiones de 21.2 cm x 15 cm. Las perforaciones que contiene la placa se utilizarán principalmente para pasar los cables del conexionado y para atornillar la electrónica utilizada en el robot.

El control de calidad viene dado por la empresa fabricante del kit.

3.2.2 Motores DC

Para el movimiento del robot se han utilizado como actuadores unos motores de continua. Su alimentación se corresponde con un intervalo de 3 V a 6 V, y está constituido con una reductora de 1:48, llegando a tener un torque de salida de 1.1 kg · cm. Su peso ronda los 50 g por cada motor, teniendo cada uno unas dimensiones de 70 mm x 22 mm x 18 mm. Este tipo de actuadores pueden alcanzar una velocidad máxima sin carga de 200 RPM y de 160 RPM con carga. El conexionado se realizará mediante dos pines que poseen los motores y un proceso de soldadura.

Las pruebas de funcionamiento se realizan por parte de la empresa fabricante de los motores.

3.2.3 Ruedas

Las ruedas utilizadas en el robot son de dos tipos diferentes. Por una parte, tenemos dos ruedas convencionales de un diámetro de 65 mm y un exterior de goma antideslizante, teniendo en cuenta que estarán completamente unidas a los actuadores. Por otra parte, se instalará una rueda loca en la parte trasera del robot utilizando separadores de placa de 1 cm y las perforaciones de la propia base.

El fabricante lleva a cabo los procesos pertinentes relacionados con el control de calidad.

3.3 Composición del escenario

3.3.1 Cámara

Para el proyecto se ha utilizado una cámara trasera de un teléfono móvil Huawei P20 lite, que cuenta con un sensor principal de 16 MP con una apertura focal de 2.2 y uno secundario de 2 MP con una apertura focal de 2.4. Su grabación de video alcanza el Full HD (1080p) a 30 fotogramas por segundo.

Por parte de Huawei, deben de realizar los controles de calidad necesarios para asegurar tanto el funcionamiento de las diferentes partes del dispositivo como su resistencia a diferentes situaciones.

3.3.2 Trípode

Par conseguir una posición elevada y un rango de funcionamiento de la aplicación lo suficientemente amplio, se hizo uso de un trípode de la marca TARION. Este soporte consta de un enganche mecánico que le permite agarrarse a un gran número de superficies, y una articulación de movimiento libre para ajustar el teléfono móvil en la posición adecuada para la aplicación, que sería una configuración de cámara cenital.

El control de calidad se lleva a cabo a manos de la empresa fabricante.

3.3.3 Escenario

Para el montaje del escenario, se ha utilizado principalmente dos gatos para la fijación de una madera de tamaño 45.7 cm x 11.3 cm, permitiendo extender el trípode hasta alcanzar una posición centrada y amplia para la aplicación. El teléfono móvil se colocará en el trípode de forma que mire recto hacía abajo. También se ha utilizado un mantel de papel completamente blanco cubriendo toda la superficie que se observa desde el plano de la cámara, teniendo unas dimensiones de 2.42 m x 1.90 m

Cada producto utilizado para el escenario ha pasado los controles de calidad de las empresas fabricantes.

4 Condiciones de ejecución

4.1 Montaje del robot

Primeramente, se realizará el proceso de soldadura del cableado pertinente en los motores de continua. Para ello se hará uso de cuatro cables macho-macho, un soldador y estaño de diámetro 1 mm. Se calentará el pin donde se va a realizar la conexión para posteriormente aplicar el estaño y fijar el cable al motor. Este proceso de repetirá tres veces más.

Con los motores cableados, se deberán instalar ambos motores en la base perforada haciendo uso de las cuatro piezas de madera en forma de “T” y los tornillos que vienen con el kit. De esta forma, se insertarán las piezas de 32 mm x 10 mm x 2 mm en las perforaciones cuadradas de 10 mm x 2 mm. Una vez se hayan colocado los soportes de madera (que constan de dos perforaciones para atornillar), dos piezas para cada motor, se colocarán ambos actuadores, y se procederá a introducir los tornillos M3 x 3 cm. Los tornillos deben atravesar los dos soportes y los motores, para finalmente fijarse mediante dos tuercas M3.

El siguiente paso consiste en instalar la rueda trasera que, como se ha comentado anteriormente, es una rueda loca. Para realizar este procedimiento, se van a utilizar un total de ocho tornillos M3 x 6 mm con arandela incorporada y cuatro separadores de metal de 11 mm. La rueda loca consta de un soporte con cuatro agujeros para llevar a cabo el proceso de instalación en la base perforada.

Primeramente, se fijarán los separadores de mental en el soporte de la rueda, para ello únicamente se deben utilizar cuatro de los tornillos con arandela y atornillar los separadores al soporte. Posteriormente, se utilizarán los cuatro tornillos restantes para finalmente fijar el conjunto de los separadores y la rueda a la base perforada, teniendo en cuenta que los agujeros en los que se debe atornillar el soporte deben estar alineados con los de los separadores y la rueda loca. Para finalizar el montaje de las ruedas, se instalarán las dos ruedas convencionales, que irán conectadas a los actuadores para proporcionarle la capacidad de movimiento al robot.

Con el chasis prácticamente montado, se va a llevar a cabo la instalación de la electrónica antes de sellar el robot, para facilitar el montaje. Primeramente, la fijación del shield del Arduino Nano se ha llevado a cabo utilizando tres separadores de metal de 1 cm, tres tornillos de metal M3 x 6 mm y otros tres tornillos de las mismas medidas, pero de plástico, evitando así malos contactos con la electrónica. De esta forma, únicamente se han buscado en la base perforada los agujeros que mejor cuadraban con las dimensiones y los agujeros que posee la shield, y se ha fijado utilizando los separadores y los tornillos mencionados.

Respecto al controlador L298N, su fijación se llevaba a cabo en la parte superior del robot, colocando el módulo hacia abajo. Para realizar este procedimiento se ha seguido una estrategia similar a la vista en el microcontrolador, observando los agujeros que mejor cuadraban para realizar la fijación del módulo. En este caso, el montaje se ha llevado a cabo utilizando dos separadores de 1 cm, cuatro tornillos de 6 mm y dos arandelas. De esta forma, se únicamente se escogen los agujeros de la placa perforada que se va a utilizar como parte superior y se atornilla el componente.

En el interior del robot estarán colocadas la fuente de alimentación y el módulo Bluetooth. La pila que suministra la energía al robot no se ha fijado de ninguna forma, ya que únicamente se ha colocado de forma horizontal, proporcionando estabilidad suficiente para que no se mueva en exceso y facilitando así el cambio de pila en caso de que la existente se gaste. Por otra parte, el módulo Bluetooth se ha fijado utilizando cinta de doble cara, de esta manera se evitarán posibles desconexiones del componente.

Para finalizar, se tratará el sellado del robot, y para llevar esto acabo primeramente se deben atornillar los separadores de placa que permitirán atornillar el techo. Para realizar este procedimiento se necesitarán cuatro separadores de placa de una longitud de 4.5 cm y cuatro tuercas de M3. De esta forma, se colocarán los separadores en las cuatro esquinas del robot utilizando agujeros ya presentes en la base y fijándolas con las tuercas.

Una vez se tienen los soportes de la parte superior, se puede implementar la caja de madera que protegerá al robot cubriéndole las zonas laterales, frontales y traseras. Para el montaje de la caja únicamente se han unido cuatro piezas de madera: dos de ellas con unas dimensiones de 17 cm x 4.5 cm, y las otras dos con unas dimensiones de 9 cm x 4.5cm. Para realizar la unión se han utilizado bastones de madera de 1 cm^2 y una altura igual que las paredes, además de pegamento de contacto para realizar la fijación. La caja se deberá colocar de forma que los separadores de placa delanteros queden en su interior y los traseros queden fuera.

Con la protección del robot fijada y colocada, únicamente quedaría colocar la parte superior, que se debe recordar que es exactamente igual que la base del robot. Para ello se han utilizado cuatro tornillos M3 x 6 mm de longitud. El proceso es sencillo, ya que únicamente hay que alinear los agujeros de la tapa superior con los de los separadores de placa anteriormente colocados y realizar el proceso de fijación utilizando los tornillos y las arandelas.

Únicamente queda insertar el interruptor en la parte superior del robot, donde hay una perforación para instalarlo de manera precisa. Para llevar a cabo el conexionado se deberá realizar un proceso de soldadura en la conexión serie que hay entre el polo positivo de la pila y la entrada de voltaje del controlador L298N. De esta forma, el robot quedaría finalmente montado y listo para llevar a cabo la aplicación.

4.2 Montaje del escenario

El montaje del escenario se lleva a cabo en un espacio amplio y bien iluminado. La primera parte es buscar un lugar de fijación elevado para colocar de manera estable el soporte para el teléfono móvil.

El procedimiento comienza fijando el tablón de madera en la posición escogida utilizando dos gatos de carpintero. Tras esto, se colocará el trípode lo más alto posible, con el objetivo de aumentar el rango de visión de la cámara, fijándolo con su agarre mecánico al tablón de madera. Finalmente se colocará el teléfono móvil en el extremo del trípode mirando hacia abajo de forma recta (posición cenital).

Por otra parte, se colocará la superficie completamente blanca en el espacio de la aplicación. Esta se extenderá lo máximo posible evitando posibles arrugas que empeoren el funcionamiento del robot, y se fijará utilizando cinta adhesiva transparente.

5 Prueba de servicio

5.1 Conexión cámara – Python

Se procederá a probar la conexión de la aplicación utilizada para establecer la cámara IP y la plataforma que se utilizará para llevar a cabo el procesamiento de imagen. Para ello el procedimiento consiste en realizar una conexión directa al servidor que crea la aplicación para transmitir la imagen. Esta prueba permitirá a su vez ajustar el espacio que visualiza la cámara para modificar si fuera necesario la altura de la cámara o cambiar la resolución de la cámara.

5.2 Prueba de funcionamiento de los motores DC

Se procederá a realizar una prueba de funcionamiento de los motores. Utilizando el código visto en el Anexo 2 se irá aumentando el valor PWM de los motores para visualizar su funcionamiento y los valores en los que el motor se encuentra en la zona muerta, utilizando un cable USB para monitorear la velocidad por el puerto serie. El procedimiento consiste en aumentar cada diez segundos los valores de velocidad que se introducen en ambos motores en un incremento de diez en diez , teniendo en cuenta que el robot debe estar colocado en la superficie en la que se va a llevar a cabo la aplicación, para que cuente con el factor de rozamiento. Además, este procedimiento permitirá observar irregularidades en los motores (que uno de ellos vaya más rápido que el otro), para poder llevar a cabo ajustes en el código.

5.3 Prueba de comunicación Bluetooth

Se llevará a cabo un procedimiento de comprobación de la comunicación Bluetooth entre el programa de Python y el robot. Para la realización de esta prueba de servicio se conectará el robot al puerto serie del ordenador que se esté utilizando para procesar la imagen y, mientras se envíen los datos de las velocidades al módulo Bluetooth del robot, se imprimirán de forma simultánea por el puerto serial del ordenador, verificando que efectivamente el envío de datos se ha llevado a cabo con éxito.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO DE FIN DE GRADO

DOCUMENTO Nº 4. PRESUPUESTO

ÍNDICE

1 Tabla de precios elementales

La mayoría de los materiales utilizados en el proyecto han sido reutilizados de otros trabajos previos o eran elementos que ya se poseían con anterioridad. En el apartado de herramientas no se considerarán los precios (0 euros) pero al final se añadirá un porcentaje de gastos auxiliares o directos haciendo referencia a estos elementos. Cabe destacar que, aunque gran parte de los elementos se comprarán en pack, los precios son unitarios.

1. Cuadro de precios elementales			
Ref.	**Unidad**	**Descripción**	**Precio (€)**
Materiales			
m1	ud.	Microcontrolador Arduino Nano	5,50
m2	ud.	Shield para Arduino Nano	4,50
m3	ud.	Driver L298N	3,50
m4	ud.	Módulo Bluetooth SPP-C	5,50
m5	ud.	Pila de 9V	2,50
m6	ud.	Cables Dupont	0,08
m7	ud.	Cable adaptador para pila de 9V	0,60
m8	ud.	Cable Mini USB	8,00
m9	ud.	Kit de robótica (Base+motores+ruedas+interruptor+tornilleria)	17,00
m10	ud.	Base perforada (utilizada de tapa)	3,00
m11	ud.	Madera MDF 50 x 50 cm	10,00
m12	ud.	Bastón de madera 1 cm2 x 4,5 cm	0,50
m13	g.	Pegamento de contacto	5,77
m14	ud.	Kit de tornillos, arandelas y separadores M3 metal y plástico	8,00
m15	ud.	Teléfono móvil Huawei P20 lite	200,00
m16	ud.	Trípode para el teléfono móvil	22,98
m17	ud.	Mantel blanco 7 x 1,20 m	2,50
m18	ud.	Tablón de madera de 45,7 x 11,3 cm	1,00
m19	ud.	Plástico termo retráctil Ø3,2	0,69
m20	ud.	Hoja de tamaño A4 (para pegatinas)	0,01
m21	ud.	Cinta adhesiva transparente	1,35
m22	g.	Bobina de estaño para soldadura electrónica 0,8 mm (20 g)	0,82
m23	ud.	Gato carpintero	5,40
Herramientas			
e1	ud.	Multímetro digital	0,00
e2	ud.	Soldador de estaño	0,00
e3	ud.	Ordenador	0,00
e4	ud.	Destornillador	0,00
MOD			
h1	h	Ingeniero	15,00
h2	h	Ayudante	12,00
Medios auxiliares			
	%	M.A sobre costes directos	10,00%

2 Cuadro de precios descompuestos

Para la realización del presupuesto total de la aplicación se ha realizado un cuadro de precios descompuestos, donde se podrán visualizar todos los montajes y procesos de los que ha estado compuesto el proyecto y el coste de cada uno de ellos. Los medios auxiliares únicamente se aplicarán en el montaje final y en el diseño del código, ya que los demás procesos se van concatenando en las diferentes partidas.

2. Cuadro de precios descompuestos					
Ref.	**Unidad**	**Descripción**	**Precio**	**Cantidad**	**Parcial**
d1	ud.	Montaje de la base y los elementos mecánicos del robot incluyendo el soldado de los cables de los motores			
		Materiales			
m9	ud.	Kit de robótica (Base+motores+ruedas+interruptor+tornilleria)	17,00	1	17,00
m6	ud.	Cables Dupont	0,08	4	0,32
m22	g.	Bobina de estaño para soldadura electrónica 0,8 mm (20 g)	0,82	5	4,10
		MOD			
h1	h	Ingeniero	15,00	1	15,00
		Medios auxiliares			
	%	M.A sobre costes directos	10,00%	66,42	6,64
				Precio e.m	**86,35**

Ref.	**Unidad**	**Descripción**	**Precio**	**Cantidad**	**Parcial**
d2	ud.	Montaje de la electrónica en la base del robot			
		Materiales			
d1	ud.	Montaje de la base y los elementos mecánicos del robot incluyendo el soldado de los cables de los motores	86,35	1	86,35
m1	ud.	Microcontrolador Arduino Nano	5,50	1	5,50
m2	ud.	Shield para Arduino Nano	4,50	1	4,50
m3	ud.	Driver L298N	3,50	1	3,50
m4	ud.	Módulo Bluetooth SPP-C	5,50	1	5,50
m5	ud.	Pila de 9V	2,50	7	17,50
m6	ud.	Cables Dupont	0,08	16	1,28
m7	ud.	Cable adaptador para pila de 9V	0,60	1	0,60
m8	ud.	Cable Mini USB	8,00	1	8,00
m14	ud.	Kit de tornillos, arandelas y separadores M3 metal y plástico	8,00	1	8,00
		MOD			
h1	h	Ingeniero	15	2	30,00
				Precio e.m	**440,73**

Ref.	Unidad	Descripción	Precio	Cantidad	Parcial
d3	ud.	Montaje de la caja protectora			
		Materiales			
m11	ud.	Madera MDF 50 x 50 cm	10	1	10,00
m12	ud.	Bastón de madera 1 cm2 x 4,5 cm	0,5	4	2,00
m13	g.	Pegamento de contacto	5,77	0,02	0,12
		MOD			
h1	h	Ingeniero	15	0,25	3,75
				Precio e.m	**23,37**

Ref.	Unidad	Descripción	Precio	Cantidad	Parcial
d4	ud.	Sellado del robot (robot completo)			
		Materiales			
d2	ud.	Montaje de la electrónica en la base del robot	440,73	1	440,73
d3	ud.	Montaje de la caja protectora	23,37	1	23,37
m10	ud.	Base perforada (utilizada de tapa)	3,00	1	3,00
m19	ud.	Plástico termo retráctil Ø3,2	0,69	1	0,69
		MOD			
h1	h	Ingeniero	15	0,5	7,50
				Precio e.m	**489,59**

Ref.	Unidad	Descripción	Precio	Cantidad	Parcial
d5	ud.	Montaje del escenario para realizar la aplicación			
		Materiales			
m15	ud.	Teléfono móvil Huawei P20 lite	200	1	200,00
m16	ud.	Trípode para el teléfono móvil	22,98	1	22,98
m17	ud.	Mantel blanco 7 x 1,20 m	2,5	1	2,50
m18	ud.	Tablón de madera de 45,7 x 11,3 cm	1	1	1,00
m20	ud.	Hoja de tamaño A4 (para pegatinas)	0,01	1	0,01
m21	ud.	Cinta adhesiva transparente	1,35	1	1,35
m23	ud.	Gato carpintero	5,4	1	5,40
		MOD			
h1	h	Ingeniero	15	1	15,00
h2	h	Ayudante	12	1	12,00
				Precio e.m	**422,24**

Ref.	Unidad	Descripción	Precio	Cantidad	Parcial
d6	ud.	Unión de los montajes para llevar a cabo la aplicación completa			
		Materiales			
d4	ud.	Sellado del robot (robot completo)	489,59	1	489,59
d5	ud.	Montaje del escenario para realizar la aplicación	422,24	1	422,24
		MOD			
h1	h	Ingeniero	15	2	30,00
		Medios auxiliares			
0	%	M.A sobre costes directos	10,00%	971,83	97,18
				Precio e.m	**1166,20**

Ref.	Unidad	Descripción	Precio	Cantidad	Parcial
d7	ud.	Desarrollo del código completo para el correcto funcionamiento de la aplicación			
		MOD			
h1	h	Ingeniero	15	100	1500,00
		Medios auxiliares			
0	%	M.A sobre costes directos	10,00%	1500,00	150,00
				Precio e.m	**1650,00**

3 Estado de mediciones

3. Estado mediciones			
Referencia	**Unidad**	**Descripción de la partida**	**Cantidad**
d6	ud.	Unión de los montajes para llevar a cabo la aplicación completa	1
d7	ud.	Desarrollo del código completo para el correcto funcionamiento de la aplicación	1

4 Resumen del presupuesto

4. Valoración Total				
Referencia	**Descripción**	**Precio**	**Cantidad**	**Total**
d6	Unión de los montajes para llevar a cabo la aplicación completa	1166,20	1	1166,198
d7	Desarrollo del código completo para el correcto funcionamiento de la aplicación	1650,00	1	1650,00
TOTAL (PEM)				2816,20
Gastos generales		6,00%		168,97
Beneficio industrial		13,00%		21,97
TOTAL (PEC)				3007,14
IVA		21%		631,50
TOTAL PRESUPUESTO				**3638,63**

El presupuesto final del diseño y el montaje de la aplicación de seguimiento asciende a:

TRES MIL SEISCIENTOS TREINTA Y OCHO CON SESENTA Y TRES EUROS