UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño electrónico e implementación de un vehículo de exploración espacial y su programación para navegación autónoma

Trabajo Fin de Grado

Grado en Ingeniería Aeroespacial

AUTOR/A: Folch Company, Andreu

Tutor/a: Masot Peris, Rafael

Cotutor/a: Alcañiz Fillol, Miguel

CURSO ACADÉMICO: 2022/2023

Agradecimientos

Agradecer, en primer lugar, a mi tutor del presente Trabajo Final de Grado, Rafael Masot, por plantearme la idea de realizar este proyecto, facilitarme los medios para su ejecución y estar comprometido y dispuesto a ayudar durante todo el proceso.

A mis padres, por quererme y apoyarme en todos los aspectos de mi vida, también en mis estudios, y por educarme e inculcarme la curiosidad científica de siempre querer saber más sobre cómo funciona lo que nos rodea.

A mis compañeros de carrera, Dani, Maci y Urru, a los que estoy convencido que les espera un brillante futuro, por ser mi apoyo durante estos últimos cuatro años y haberme permitido el privilegio de entablar una gran amistad con personas tan buenas y capaces.

Por último, a mi hermano mayor, Fran, por ser mi principal referencia en la vida, a mis primos, Miguel, Carlos, María, Jorge, Elena y Juan, y a todos mis amigos.

Resumen

El presente Trabajo Final de Grado aborda el diseño e implementación de un vehículo de exploración espacial y su programación para navegación autónoma. El proyecto se desarrolla mediante la placa ESP32 que controla y extrae información de un sensor de proximidad, un sensor de medidas inerciales, una cámara, dos motores de corriente continua y un ordenador que asiste al vehículo en la detección del objetivo. Durante la realización del proyecto se adquieren y aplican conocimientos sobre el diseño e implementación en sistema digital de un controlador PID, el uso redes neuronales convolucionales para la detección de objetos, el entrenamiento e implementación del modelo de detección YOLOv5, el desarrollo en Python de una Interfaz Gráfica, el desarrollo, en el lenguaje de Arduino IDE, del algoritmo de navegación de un vehículo, etc. Como resultado se obtiene un vehículo capaz de alcanzar y detenerse frente a una referencia visual estática o móvil incluso sobre superficies irregulares.

Palabras clave: Controlador PID; Redes neuronales convolucionales; YOLOv5; Vehículo de exploración espacial; Detección de objetos; ESP32

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

1.1. Motivación

El ser humano, desde el inicio de su existencia, se ha dedicado a explorar nuevos territorios, continentes, mares, océanos, y en última instancia, nuevos satélites o planetas, como la Luna o Marte.

Con este objetivo se han desarrollado numerosas tecnologías que alejan la exploración humana del planeta Tierra, como cohetes, naves espaciales o vehículos de exploración espacial que permiten la incursión en territorios hostiles para la supervivencia sin poner en peligro vidas humanas.

Estos vehículos de exploración espacial o *rovers* son capaces de navegar por la superficie de planetas a millones de kilómetros de la Tierra gracias a la tecnología que equipan. Es por ello por lo que el proyecto realizado pretende adentrarse en el apasionante mundo de los *rovers*, para entender mejor sus sistemas de navegación y sentirse partícipe del sentimiento de exploración que caracteriza al hombre.

1.2. Objetivos

Por lo tanto, el objetivo será diseñar e implementar un vehículo de exploración espacial que sea capaz de navegar de manera autónoma hacia un objetivo visual, atendiendo a las limitaciones de presupuesto y complejidad que supone un proyecto tan ambicioso y flexible. Se define un objetivo visual para la navegación debido a las limitaciones que suponen navegar por la superficie de un planeta lejano. Un ejemplo de misión que el vehículo objetivo debería ser capaz de llevar a cabo sería navegar por la superficie de Marte hasta encontrar visualmente un mar de hielo y avanzar hasta él para realizar las extracciones y análisis pertinentes.

Para ello, durante la ejecución del proyecto se pretende adquirir y aplicar conocimientos de las siguientes disciplinas:

1) Diseño y ensamblaje de sistemas empotrados para la navegación autónoma.
2) Detección de objetos mediante el uso de redes neuronales.
3) Diseño e implementación de controladores PID para la navegación de vehículos.
4) Desarrollo de interfaces gráficas en Python para facilitar la operación de robots.

2. Estado del arte

A lo largo del presente capítulo se introducen los conceptos claves en el desarrollo del proyecto: los vehículos de conducción autónoma, la detección de objetos mediante redes neuronales convolucionales y los controladores PID.

2.1. Vehículos de conducción autónoma

Los vehículos de conducción completamente autónoma son automóviles que se desplazan de manera independiente sin la intervención directa de una persona. Estos vehículos están diseñados para realizar todas las tareas requeridas para una conducción segura y eficiente.

Con el fin de cumplir su función, los vehículos autónomos están equipados con una combinación de unidades de procesamiento y sensores, como cámaras, radares, lidares, sonares y sensores GNSS. Estos componentes trabajan en conjunto para captar y procesar información sobre el estado y el entorno circundante del vehículo en tiempo real, y a partir de dicha información, realizar un plan de movimiento y ejecutarlo a través de los actuadores [1]. La arquitectura del software se muestra en la Figura 1.

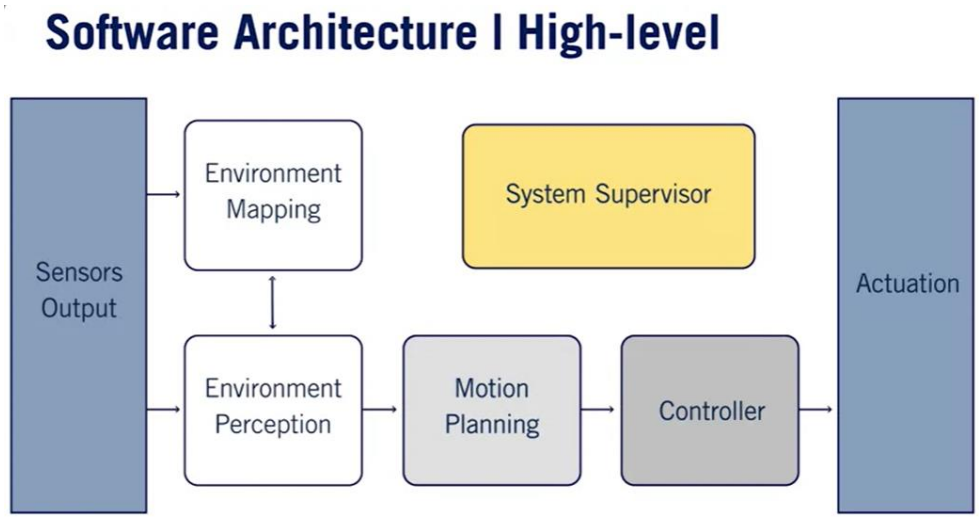

Figura 1.- Arquitectura del software de los vehículos de conducción autónoma [1].

Como se ve en la Figura 1, el bloque de percepción del entorno recibe información de los sensores y envía la información resultante a los bloques de planificación del movimiento y de mapeado del entorno. En concreto, a partir de la información de los sensores previamente mencionados, calcula la posición del vehículo, de objetos estáticos en el entorno y predice el movimiento de los objetos dinámicos del entorno.

Por otro lado, el bloque de mapeado del entorno realiza mapas que posibilitan la tarea del bloque de planificación del movimiento y mejoran la estimación de la localización del vehículo calculada por el bloque de percepción del entorno. En primer lugar, a partir de la información del lidar y del seguimiento de los objetos del entorno, se elabora el

mapa de rejilla de ocupación, en el que cada celda de la rejilla contiene una probabilidad de estar ocupado por algún elemento externo. Por otro lado, a partir de la información del lidar se elabora el mapa de localización, que mejora la estimación de la posición y velocidad del vehículo calculando la velocidad y la posición respecto al entorno. Por último, se calcula el mapa detallado de la carretera a partir del mapa de la carretera por la que se circula y añadiéndoles la información del seguimiento de objetos.

Tras recibir la información de los bloques anteriores, el bloque de planificación del movimiento establece una ruta a través de la red de carreteras para llegar al objetivo. Durante el seguimiento de la ruta se incorpora la información del seguimiento de objetos estáticos y dinámicos, de la posición y velocidad del vehículo y del mapa de rejilla de ocupación para calcular la trayectoria deseada.

Por último, el bloque de control calcula la aceleración y la dirección del volante necesarias para realizar la trayectoria, y manda las ordenes resultantes a los actuadores.

2.1.1. Vehículos de exploración espacial

En cuanto a la navegación de los vehículos de exploración espacial, esta se ve afectada por la ausencia de carreteras y de mapas precisos, además de la ausencia de satélites GNSS. Es por ello por lo que la información para la navegación se ve limitada a la recibida por los sensores de navegación inercial y las cámaras, que capturan imágenes del terreno circundante.

A esta información se le añade la percepción del entorno recibida de los lidares y radares, que permiten mapear el terreno y detectar la distancia a los objetos del entorno. Con base en la información mencionada, el rover utiliza algoritmos de planificación de trayectorias para determinar la ruta óptima para alcanzar un objetivo específico.

2.2. Detección de objetos

La detección de objetos es un problema fundamental en Visión Artificial que consiste en la identificación y localización de múltiples objetos sobre imágenes digitales. El enfoque tradicional a la hora de abordar dicho problema, basado en métodos más artesanales y en algoritmos clásicos de *machine learning*, se veía limitado a la hora de detectar distintas clases de objetos en diferentes condiciones y escenarios. A lo largo de los últimos años las redes neuronales convolucionales han revolucionado la detección de objetos, consiguiendo la realización de la tarea a partir de información sin procesar. Es decir, mientras los algoritmos tradicionales necesitan realizar una extracción previa de las características de la imagen, las redes neuronales son capaces de resolver el problema a partir de la imagen en crudo.

El vehículo diseñado basa su navegación en alcanzar un objetivo visual, para lo que empleará una red neuronal convolucional previamente diseñada y entrenada, y reentrenada por el propio usuario para la detección del objeto deseado.

A continuación, se explica brevemente qué es una red neuronal convolucional y como se puede medir su rendimiento en la detección de objetos.

2.2.1. Redes neuronales convolucionales

Una red neuronal convolucional es un tipo de modelo de *Deep learning* diseñado específicamente para el procesamiento y análisis de imágenes y otros tipos de datos similares. Estas redes utilizan la operación matemática de convolución con el objetivo de extraer información de las propias imágenes o, de forma más general, cualquier tensor de datos de entrada. Esta operación de convolución, como se muestra en la Figura 2, opera sobre cada región de la matriz de la imagen mediante filtros cuyos valores se establecen a través del entrenamiento [2].

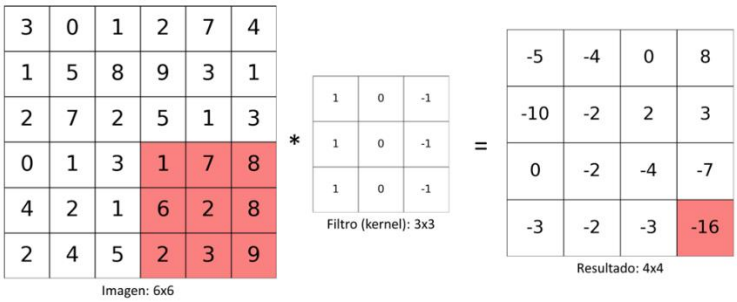

Figura 2.- Operación de convolución [3].

La operación marcada en rojo sobre la Figura 2 se desarrolla en la Ecuación (1).

$$1 \cdot 1 + 7 \cdot 0 + 8 \cdot (-1) + 6 \cdot 1 + 2 \cdot 0 + 8 \cdot (-1) + 2 \cdot 1 + 3 \cdot 0 + 9 \cdot (-1) = -16 \quad (1)$$

Las redes neuronales convolucionales están compuestas por múltiples capas que se aplican en cascada con el fin de extraer características relevantes de las imágenes recibidas. Se destacan las siguientes capas:

1) Capa de convolución: Esta capa es el núcleo de las redes neuronales convolucionales. Aplica un conjunto de filtros al tensor de entrada, que es la salida de la capa anterior. La entrada de la primera capa convolucional es la propia imagen a analizar. La salida de una capa convolucional consiste en un conjunto de mapas de características que resaltan patrones relevantes, como bordes, texturas o formas.
2) Capa de agrupación: Esta capa reduce la dimensión espacial del mapa de características, reduciendo la cantidad de parámetros y computaciones en el modelo. La operación de *max-pooling* selecciona el valor máximo en pequeñas áreas del mapa de características.

La sucesión de estas capas junto con otras más específicas, en las que no es necesario entrar en detalle, forman la arquitectura de las redes neuronales convolucionales tal y como se muestra en la Figura 3.

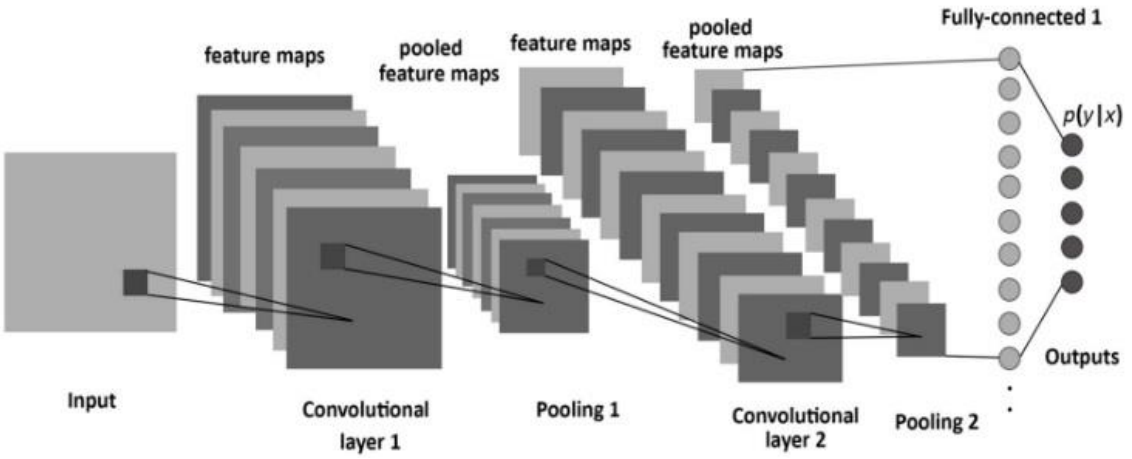

Figura 3.- Capas de las redes neuronales convolucionales [4].

Finalmente, como se observa en la Figura 3, se introduce el vector de características resultante en la capa totalmente conectada, obtenida de la agrupación de los últimos mapas de características, como entrada a la arquitectura de una red neuronal tradicional cuyas salidas contendrán finalmente la información correspondiente a las detecciones.

2.2.2. Calidad de un detector

La calidad de un detector se mide a partir de ciertas métricas específicas. Para definirlas se emplea la matriz de confusión, mostrada en la Tabla 1, que permite mostrar los aciertos y errores del modelo de detección. Las columnas representan la predicción realizada por el detector sobre una clase, mientras que las filas representan la condición real de la propia clase [5].

		Predicción de la condición	
		Positiva (PP)	Negativa (PN)
Condición real	Positiva (P)	Verdadero positivo (VP)	Falso negativo (FN)
	Negativa (N)	Falso positivo (FP)	Verdadero negativo (VN)

Tabla 1.- Matriz de confusión [5].

En la detección de objetos hay una gran cantidad de muestras negativas, es decir, todas aquellas posibles ventanas o marcos que no enmarcan ninguno de los objetos que se pretende detectar. Se excluye pues la condición de verdadero negativo de las métricas

de calidad de los detectores, para evitar indicadores de calidad de detección no representativos.

La primera métrica empleada, denominada precisión, relaciona los verdaderos positivos, obtenidos por el detector, sobre el total de predicciones de condición positiva que realiza el propio detector, tal y como se muestra en la Ecuación (2).

$$precisión = \frac{VP}{PP} \tag{2}$$

Por otro lado, la métrica definida como sensibilidad (*recall* en inglés), en la Ecuación (3) relaciona los verdaderos positivos sobre el total de las condiciones positivas reales.

$$sensibilidad = \frac{VP}{P} \tag{3}$$

A partir de estas métricas se obtiene, para la detección de una única clase de objetos, el valor del *Average Precision*. Además, en el caso de un modelo de detección con varias clases, es interesante obtener la *Mean Average Precision* o mAP, pero puesto que en el desarrollo del vehículo solamente se detecta una única clase no se entrará en más detalle.

Por último, se definen dos nuevos parámetros de calidad del detector: la pérdida de marco o *box loss* y la pérdida de objetividad o *objectness loss*. La pérdida del marco representa la precisión con la que el detector localiza el centro del objeto, y la precisión con la que el marco cubre el propio objeto. La pérdida de objetividad indica la precisión con la que se predice la probabilidad de que un objeto exista en una región de interés determinada. Ambos parámetros reflejarán un mayor rendimiento del detector cuanto menor sea su valor [6].

2.3. Control PID

El controlador PID es sin duda alguna el algoritmo de control más utilizado en el mundo. La mayoría de los bucles cerrados o de realimentación de la salida se controlan mediante este algoritmo o versiones de este con pequeñas variaciones. En esta sección se introduce al control PID, su algoritmo básico y alguna representación alternativa [7].

En la Figura 4 se muestra el diagrama de bloques de un bucle cerrado donde se controla un sistema, descrito por una función de transferencia $G(s)$, mediante un controlador PID para el seguimiento de la referencia $R(s)$. Este se encuentra en el dominio de la frecuencia, ya que las variables se encuentran expresadas en función de la variable imaginaria s.

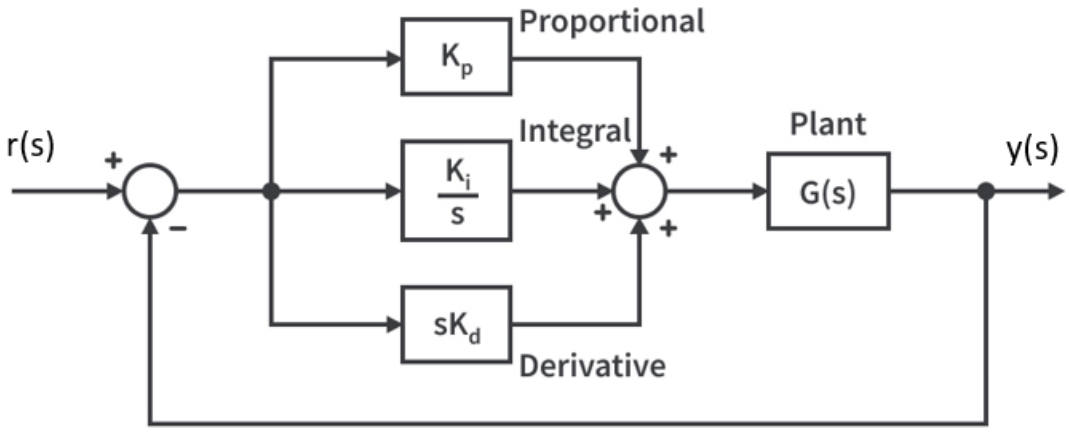

Figura 4.- Bucle cerrado con controlador PID [8].

El controlador PID ordena una acción de control u que, como se observa en la Ecuación (4), es la suma de las tres acciones que incluye el PID: la acción proporcional, la acción integral y la acción derivativa.

$$u(s) = K_p\big(r(s) - y(s)\big) + \frac{K_i}{s}\big(r(s) - y(s)\big) + K_d\, s\big(r(s) - y(s)\big) \tag{4}$$

Acción Proporcional

La acción proporcional, que se muestra en la Ecuación (5), consiste en tomar la diferencia entre la referencia r y el estado actual y de la variable controlada, es decir, el error, y multiplicarlo por una constante K_p. De este modo se obtiene una acción de control proporcional u_p que aparece en oposición al error. Es posible regular la magnitud de u_p ante un mismo error cambiando el valor de la constante K_p.

$$u_p(s) = K_p(r(s) - y(s)) \tag{5}$$

Acción Integral

La principal función de la acción integral es asegurarse de que la salida del proceso coincida con la referencia en la fase estacionaria. Con el control proporcional normalmente existe un error en dicha fase. En cambio, con la acción integral el más mínimo error, sin importar cuan pequeño sea, prolongado en el tiempo conlleva al crecimiento de la acción de control [7].

El término integral, como su propio nombre indica, integra el error y lo multiplica por la constante K_i. Expresando este concepto en el dominio de la frecuencia), donde el integrador es $\frac{1}{s}$, se obtiene la Ecuación (6).

$$u_i(s) = \frac{K_i}{s}\big(r(s) - y(s)\big) \tag{6}$$

Acción Derivativa

El propósito de la acción derivativa es mejorar la estabilidad en bucle cerrado. El mecanismo de inestabilidad se puede describir de un modo intuitivo como sigue. Debido a la dinámica del proceso, pasará algo de tiempo hasta que un cambio en la acción u de control haga efecto sobre la salida del proceso y, por lo que el controlador llega tarde a corregir el error. En cambio, con la introducción de la acción derivativa, el control se realiza sobre la predicción del error del proceso [7].

Para ello, el término derivativo, como su propio nombre indica, deriva el error y lo multiplica por la constante K_d. Expresando este concepto en el dominio de la frecuencia, donde la derivada es s, se obtiene la Ecuación (7).

$$u_d(s) = K_d\, s\big(r(s) - y(s)\big) \tag{7}$$

Sin embargo, la acción derivativa puede causar problemas en el caso de que la medición del estado de la variable controlada introduzca ruido de alta frecuencia. Este ruido puede provocar que la señal de control derivativo sea arbitrariamente amplia si la frecuencia del ruido es lo suficientemente alta. La ganancia de alta frecuencia del término derivativo, por lo tanto, se limita para evitar este problema [7]. Con este objetivo, el término derivativo se modifica por el mostrado en la Ecuación (8).

$$u_d(s) = \frac{K_d\, s}{T_f\, s + 1}\big(r(s) - y(s)\big) \tag{8}$$

En la Ecuación (8) se puede apreciar como al término derivativo ideal s se le ha añadido un filtro paso bajo que, como su nombre indica, deja pasar las frecuencias bajas y atenúa las altas. Los filtros paso bajo en el dominio de Laplace toman la forma $\frac{N}{N\, s+1}$, donde el umbral entre las frecuencias alta y baja está definido por la constante N, o por T_f en este caso.

Filtro de ponderación de la referencia

La última modificación que se introduce es el filtro de ponderación de la referencia. Añadiendo el filtro F aumentan a dos los grados de libertad del controlador PID. Los controladores PID de dos grados de libertad son capaces de rechazar rápidamente las perturbaciones sin sufrir un incremento significativo de la sobreoscilación en el seguimiento de la referencia. Además, son útiles a la hora de mitigar el efecto de los cambios de la referencia sobre la señal de control [9].

El filtro de ponderación de la referencia F multiplica a la referencia antes de introducirla en el bucle cerrado como se muestra en el diagrama de bloques de la Figura 5.

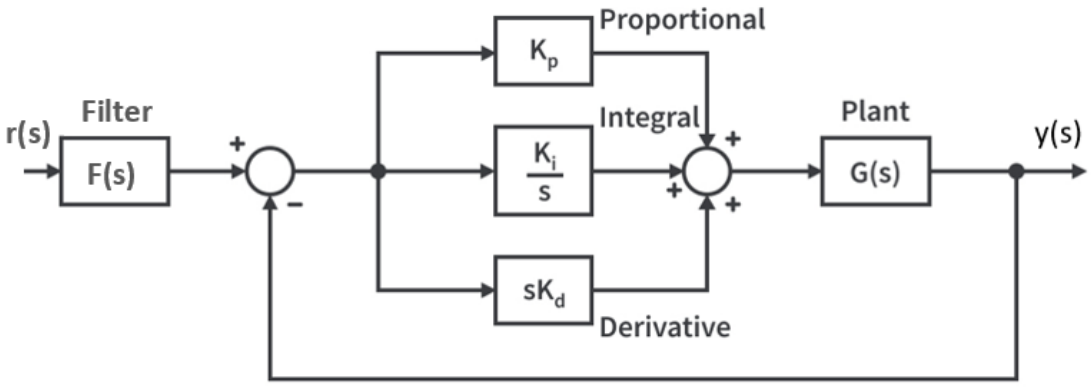

Figura 5.- Bucle cerrado con controlador PID y filtro de ponderación de la referencia [8].

El filtro F, como se muestra en la Ecuación (9), se define a partir de las constantes utilizadas en el controlador PID de un grado de libertad, además de dos nuevas constantes: b y c [9].

$$X(s) = \frac{(b\,K_p\,T_f + c\,K_d)s^2 + \left(b\,K_p + K_i\,T_f\right)s + K_i}{(K_p\,T_f + K_d)s^2 + \left(K_p + K_i\,T_f\right)s + K_i} \tag{9}$$

La relación entre la acción de control salida del PID de dos grados de libertad u, y sus dos entradas, la referencia r y el valor de la variable controlada y, se puede representar como la Ecuación (10). Se puede apreciar que en los términos proporcional y derivativo se multiplica la referencia por las constantes b y c [9].

$$u(s) = K_p(b\,r(s) - y(s)) + \frac{K_i}{s}(r(s) - y(s)) + \frac{K_d\,s}{T_f\,s + 1}(c\,r(s) - y(s)) \tag{10}$$

La Ecuación (10) muestra la forma final del controlador PID que, con las correspondientes modificaciones para ser implementado en el sistema real, se emplea en el vehículo para el seguimiento de la referencia definida más adelante.

3. Materiales y métodos de la solución

A lo largo de este capítulo se describen los componentes y métodos seleccionados con el objetivo de desarrollar una solución ante el problema planteado. La solución diseñada se basa, en parte, en la disponibilidad de los distintos elementos, ya que se limita a la proporcionada por el tutor del trabajo. Además, se tienen en cuenta también las limitaciones propias de la navegación de los vehículos de exploración espacial descritas en el capítulo anterior, con la adición de que no se dispone de mapas del terreno en el que se navega, ni se emplean lidares ni radares para mapear el terreno por la complejidad que sumarían al desarrollo de la solución, por lo que se añadirían en futuras mejoras del modelo.

Por lo tanto, la solución tomada basa la navegación del vehículo en referencias visuales capturadas por una cámara, un sensor de medidas inerciales, suponiendo que el planeta explorado dispone de campo magnético, y un sensor de proximidad para evitar la colisión frontal con elementos externos.

Sobre las imágenes captadas por la cámara del robot se realizan las detecciones de objetos, en busca del objetivo del vehículo, mediante una red neuronal convolucional preentrenada seleccionada más adelante en este capítulo.

3.1. Hardware

A continuación, se describen en detalle los elementos del hardware que componen la solución final y los motivos de su selección frente a otras alternativas. La imagen de cada uno de los componentes seleccionados se muestra en la Figura 11.

3.1.1. Placa de desarrollo

La placa de desarrollo es la unidad de procesamiento de la navegación del vehículo, extrae información de los sensores, ordena las acciones de control a los motores y se conecta vía Bluetooth con el ordenador para recibir información relacionada con la detección de objetos.

Algunas de las alternativas que ofrece el mercado son: Raspberry Pi, ATmega328p, Arduino Uno Rev3, entre otras.

Sin embargo, la placa seleccionada, por ser muy económica y por haber aprendido a utilizarla a lo largo de la carrera, es la placa ESP32 NodeMCU, de la empresa Az-Delivery. Esta equipa un microcontrolador de 32 bits, el ESP32, que dispone de una gran capacidad de memoria, buena capacidad de procesamiento, conexión WIFI y Bluetooth, canales de comunicación UART, I2C, etc. En definitiva, es perfectamente válida para el desarrollo de la solución [10].

El firmware cargado en la placa se desarrolla en el entorno de programación Arduino IDE.

3.1.2. Sensores

Los sensores equipados por el robot son un sensor de medidas inerciales, y un sensor de distancia.

Sensor de medidas inerciales

El sensor de medidas inerciales o IMU seleccionado por disponibilidad, precio y funcionalidad es el modelo BNO055. Este sensor fusiona la información de los tres sensores que lo componen: un acelerómetro, que proporciona información de la aceleración del vehículo, un giróscopo, que mide la velocidad angular, y un magnetómetro, que mide la orientación respecto al campo magnético del planeta explorado. La fusión de los sensores permite al BNO055 proporcionar al microcontrolador información de los ángulos de Euler del vehículo, que especifican la orientación del sistema de referencia móvil del vehículo respecto a un sistema de referencia fijo.

Sensor de distancia VL35L0

El sensor de distancia seleccionado por su disponibilidad y precio es el VL53L0. Se trata de un sensor de distancia infrarrojo láser, capaz de medir distancias de 50 mm a 2000 mm de forma precisa. Su funcionamiento se basa en enviar un pulso láser de luz infrarroja y medir el tiempo que tarda en regresar al sensor [11].

Al equipar este sensor, el vehículo es capaz de detectar obstáculos frontales en el rango especificado y detenerse evitando la colisión. También posibilita la detención del robot una vez alcanza el objetivo.

3.1.3. Cámara

La cámara seleccionada es la equipada por el teléfono inteligente de uso personal, ya que mediante la aplicación IP Webcam es capaz de crear un servidor en el que carga la imagen capturada en tiempo real, al que accede el ordenador para tomar las imágenes proporcionadas y realizar las detecciones de objetos.

3.1.4. Chasis

El chasis elegido para contener al resto de elementos ensamblados en el vehículo es el *Rover 5*. Se trata de un chasis para robot equipado con dos motores de corriente continua que mueven a las dos ruedas oruga de las que dispone. El uso de este tipo de ruedas es especialmente útil sobre terrenos irregulares como los que se podrían encontrar sobre un planeta rocoso inexplorado.

3.1.5. Controlador de motores L298N

El controlador de motores elegido para ser equipado por el robot es el L298N, mostrado en la Figura 6, ya que es capaz de controlar la velocidad y la dirección de giro de hasta dos motores de corriente continua como los equipados en el chasis.

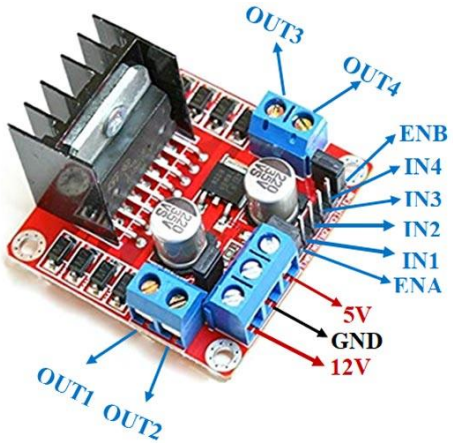

Figura 6.- Controlador de motores L298N [12].

Los pines ENA y ENB sirven para controlar la velocidad de los motores A y B respectivamente, mientras que IN1 e IN2 controlan el sentido de giro del Motor A, y los pines IN3 e IN4 la del Motor B [12].

Velocidad de los motores

El control de la velocidad de los motores DC se realiza introduciendo una señal PWM a través de los pines ENA y ENB ya mencionados. Una señal PWM es una onda cuadrada periódica como la que se muestra en la Figura 7, que se mantiene a nivel alto durante un tiempo t_{ON} [10].

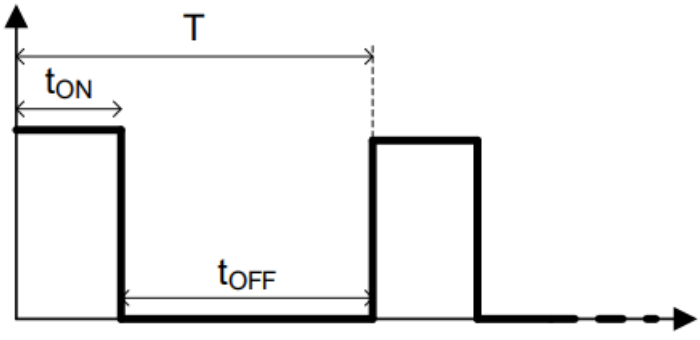

Figura 7.- Señal PWM [10].

La relación entre t_{ON} y el periodo de la onda T describe el ciclo de trabajo δ, que tomará un valor entre 0 y 1:

$$\delta = \frac{t_{ON}}{T}$$

Para generar una onda de este tipo mediante la placa de desarrollo ESP32 se dispone de la librería LEDC. En primer lugar, será necesario configurar los siguientes parámetros:

- Canal. Se debe seleccionar uno de los dieciséis canales de los que dispone el ESP32 a través del que se generará la señal PWM.
- Frecuencia de la señal PWM (en Hz).
- Resolución de la temporización (en bits).

La resolución res y el ciclo de trabajo δ definen $cton$, el número de cuentas del temporizador que corresponden al nivel alto de la señal. Este valor será introducido en la función *ledcWrite(canal,cton)* para establecer el ciclo de trabajo de la onda generada.

$$cton = \delta(2^{res} - 1) \tag{11}$$

Variando el ciclo de trabajo de la señal, varía la tensión media aplicada al motor y, por tanto, su velocidad de giro. Esta relación se aproxima con la distribución lineal de la Ecuación (12), aunque en el análisis de los resultados se comprobará que no se ajusta exactamente a esta distribución y que, además, la velocidad angular depende de más factores.

$$w = \delta\, w_{max} \tag{12}$$

Sentido de giro de los motores

La Tabla 2 muestra en qué sentido girarán las ruedas como resultado de las posibles combinaciones de activación de los pines IN1 e IN2 (o IN3 e IN4) [12].

	Adelante	Atrás	Freno
IN1 (o IN3)	HIGH	LOW	LOW
IN2 (o IN4)	LOW	HIGH	LOW

Tabla 2.- Sentido de giro de las ruedas como resultado de las posibles combinaciones de activación de los pines IN1 e IN2 (o IN3 e IN4) [12].

3.1.6. Fuentes de alimentación

Se requieren dos fuentes de alimentación. La primera de ellas, de 3.7 voltios, alimenta la placa seleccionada para el vehículo. Además, se equipa otra batería, de 7.7 voltios que alimenta al controlador de motores y a los propios motores. Para su implementación en un vehículo de exploración espacial real se debería añadir una fuente de carga para las baterías como, por ejemplo, placas fotovoltaicas.

3.1.7. Ordenador

Puesto que la placa seleccionada no dispone de la capacidad de implementar el modelo de detección de imágenes seleccionado en la siguiente sección, se requiere el uso de una unidad de procesamiento externa a la placa de desarrollo que cumpla dicha tarea.

La opción seleccionada consiste en el ordenador de uso personal, ya que es capaz de acceder al servidor creado por la aplicación IP Webcam, tomar la imagen, ejecutar el programa de Python en el que se implementa la red neuronal y enviar, a la placa ESP32 vía Bluetooth, la información resultante de las detecciones.

Debido al reducido tamaño del vehículo, no es posible ensamblar el ordenador y el router WIFI, con sus fuentes de alimentación correspondientes, al propio robot. Este debería ser el caso en un vehículo de exploración espacial real, ya que las comunicaciones con la tierra debido a las grandes distancias son extremadamente lentas, requiriendo que el sistema embebido por el vehículo sea totalmente independiente de elementos externos. En este caso, al tratarse de un ejercicio didáctico, se encuentran como elementos externos del vehículo.

Además, en el propio ordenador equipado con Matlab se realiza el diseño del controlador PID mediante la herramienta simulink como se verá en el capítulo del desarrollo del vehículo.

3.2. Red neuronal para la detección de imágenes

La arquitectura de red neuronal convolucional preentrenada seleccionada para realizar la detección de objetos en el vehículo es YOLO, que es un modelo gratuito y sencillo de entrenar e implementar, capaz de predecir los marcos delimitadores de objetos y la probabilidad de las clases directamente sobre las imágenes en una sola evaluación. Esta arquitectura está construida en PyTorch, y destaca por su versatilidad, facilidad de uso y gran rendimiento [13].

Torch es una biblioteca de código abierto que es especialmente útil para desarrollar proyectos de aprendizaje automático. Por otro lado, PyTorch es un entorno de programación especializado en el aprendizaje profundo. Se basa en el lenguaje de programación Python y aprovecha la funcionalidad proporcionada por la biblioteca Torch mencionada anteriormente. PyTorch se utiliza ampliamente por la comunidad gracias a su enfoque en la simplicidad y flexibilidad, lo que permite a los desarrolladores implementar modelos de aprendizaje profundo de manera eficiente directamente desde Python, como es el caso del trabajo desarrollado [14].

La versión del modelo seleccionado que se emplea en la realización del proyecto es Utralytics YOLOv5, que es uno de los lanzamientos de la familia de modelos YOLO, capaz de realizar tareas de detección, segmentación y clasificación de imágenes. Se escoge esta versión por su gran facilidad a la hora de ser entrenada e implementada, y como se observa en la Figura 8, no es significativamente peor que las versiones más nuevas, que en el momento de la realización del trabajo el autor aún no ha aprendido a utilizarlas por ser lanzamientos más recientes.

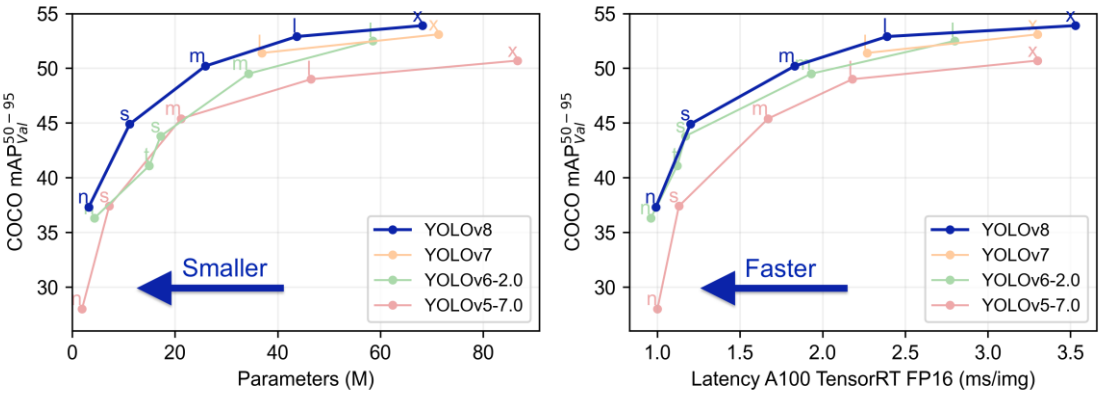

Figura 8.- Comparación del tamaño y la velocidad de las últimas versiones de YOLO [15].

En concreto se emplea la versión YOLOv5s, que es la segunda versión de YOLOv5 más rápida y sencilla, como se ve representado en naranja en la Figura 9. Esta elección se basa en que, al realizar detecciones en tiempo real para la navegación de un vehículo, es crucial que las detecciones sean rápidas además de precisas para no comprometer el rendimiento y la seguridad del vehículo y del entorno en el que circula.

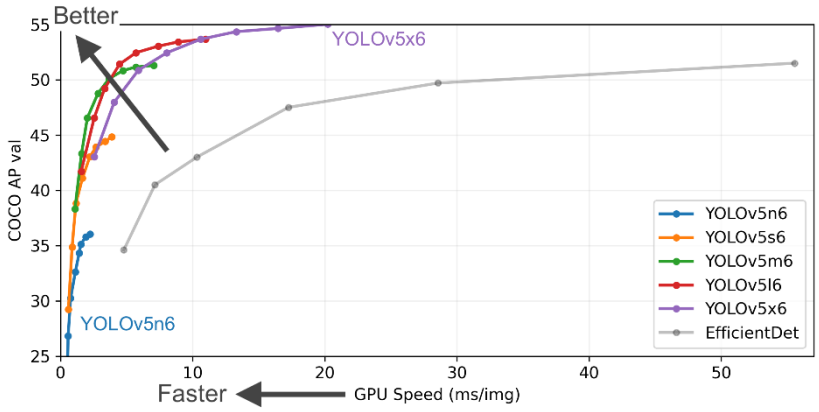

Figura 9. Comparativa del rendimiento y la velocidad de las distintas versiones de YOLOv5 [15].

Más adelante en el presente proyecto, el modelo se entrena con una base de datos personalizada, con el fin de que sea capaz de detectar el objeto que se desee, y se implementará en Python para realizar la detección sobre las imágenes que se reciban en tiempo real desde la cámara.

4. Desarrollo y Resultados

A lo largo del presente capítulo se desarrolla la solución del problema planteado y se analizan los resultados. El desarrollo se basa en la realización de los siguientes pasos:

1) En ensamblar el hardware definido en el capítulo anterior.
2) Entrenar la red neuronal seleccionada para la detección de un objeto personalizado y e implementar el modelo resultante en un algoritmo de detección ejecutado en el ordenador.
3) Diseñar e implementar en el sistema real un controlador PID con el objetivo de realizar el seguimiento de la referencia, establecida por el algoritmo de detección entre otros factores.
4) Integrar el resultado de los pasos anteriores en un algoritmo de navegación, programado en Arduino IDE y cargado en la placa del ESP32, que posibilite al vehículo alcanzar el objetivo.
5) Desarrollar una interfaz gráfica en Python que facilite la experiencia del usuario al conectar, configurar y activar el vehículo.

Por último, se analizan los resultados comprobando el funcionamiento del vehículo en diferentes situaciones.

4.1. Ensamblaje del hardware

Una vez se han seleccionado y adquirido los componentes que forman el vehículo se procede a su ensamblaje. El robot obtenido como resultado se muestra en la Figura 10.

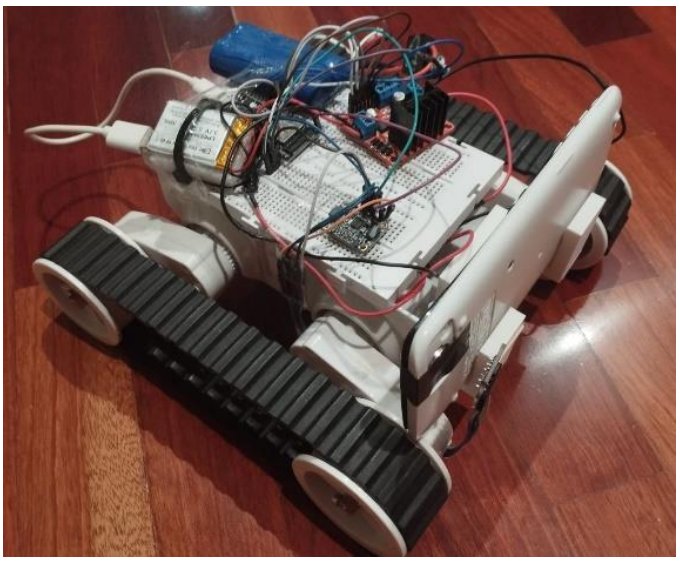

Figura 10.- Vehículo obtenido como resultado del ensamblaje del hardware.

En la Figura 11 se ha representado la unión de todos los componentes mediante un diagrama de bloques. A la izquierda, sobre un cuadro gris que representa el chasis del robot, se muestran todos los componentes que van montados sobre este, y a la derecha

se representa el ordenador, donde se realizan las tareas de detección de imagen y la configuración del robot mediante una interfaz gráfica creada en Python que se mostrará más adelante.

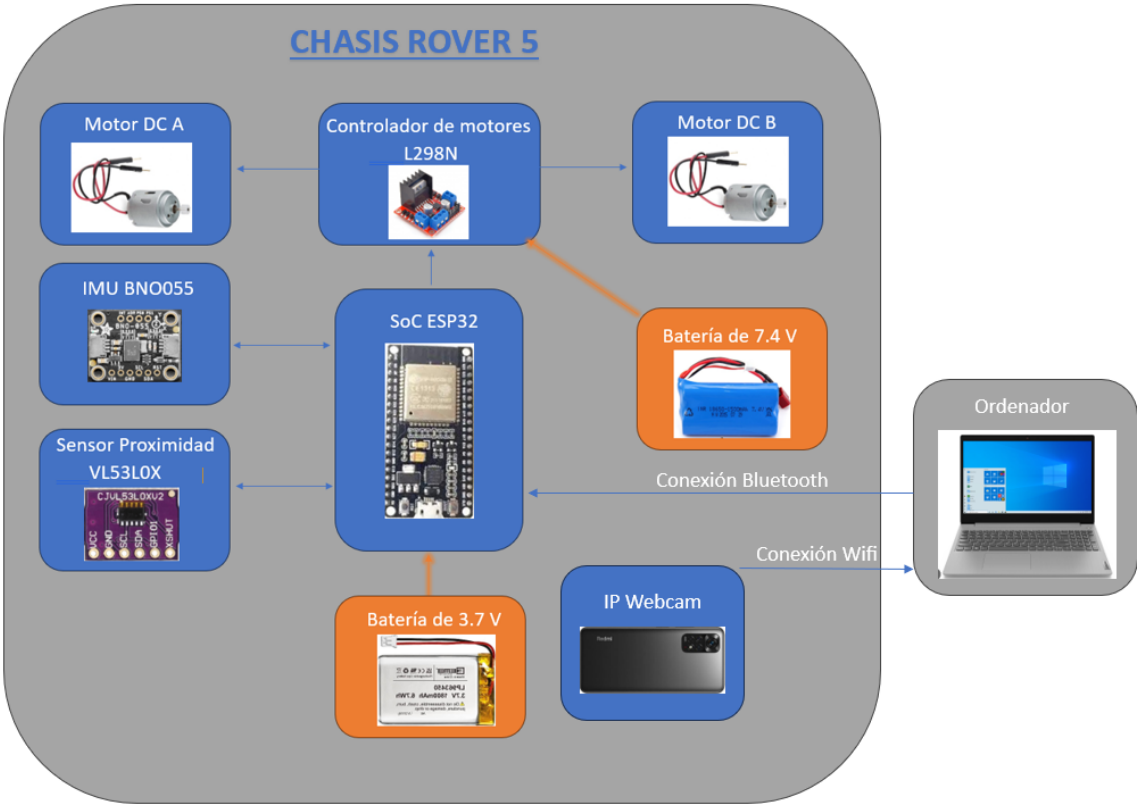

Figura 11.- Diagrama de bloques de las conexiones establecidas entre el hardware.

La placa de desarrollo ESP32 es el "cerebro" del vehículo, por lo que todos los componentes se conectan directa o indirectamente a este, algunos para enviarle información, como los sensores y el ordenador, y otros para recibirla, como el controlador de motores e, indirectamente, los propios motores. La placa se alimenta mediante una batería de 3.7 voltios a través del puerto USB.

A continuación, se comenta con más detalle las conexiones que se han establecido entre los elementos.

4.1.1. Conexión sensores

La conexión de los sensores se realiza mediante el protocolo de comunicación I2C, que permite la comunicación entre la placa ESP32, que cumple el rol de maestro, y varios sensores, que tienen el rol de esclavos. Este protocolo emplea dos cables para compartir información. Uno se utiliza para transmitir la señal del reloj (SCL) y el otro para enviar y recibir información (SDA) [16]. Como se muestra en la Figura 12, estos cables se conectan a las entradas G22 y G21 respectivamente, que son las entradas establecidas por defecto en el ESP32 para realizar este tipo de comunicación.

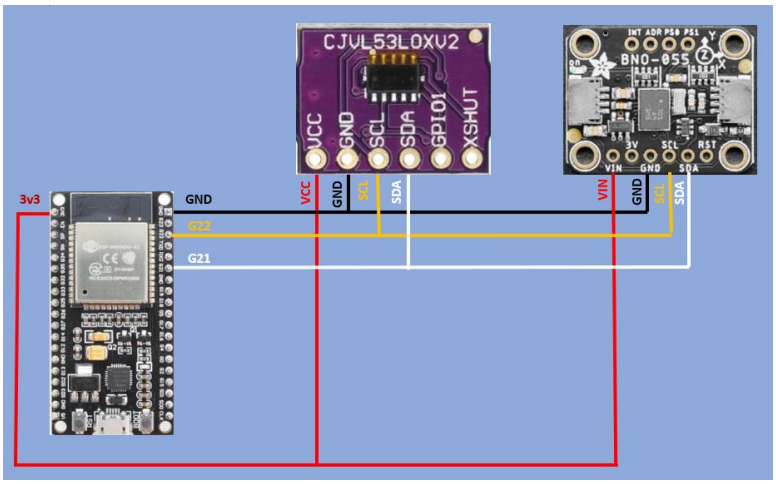

Figura 12.- Diagrama de conexiones entre los sensores BNO055, VL53L0 y la placa ESP32.

Puesto que ambos sensores se han conectado al mismo canal I2C, es necesario indicar, en el código de Arduino, la dirección de cada sensor para que la placa sea capaz de extraerles información. En el caso la IMU empleado, el BNO-055, la dirección es 0x28, mientras que en el caso del sensor de proximidad VL53L0, la dirección es 0x29.

La alimentación de los sensores se realiza mediante el pin 3v3 del ESP32, que suministra un voltaje de 3.3 voltios.

4.1.2. Conexión controlador y motores

En la Figura 13 se han ilustrado las conexiones necesarias para controlar los motores desde la placa ESP32.

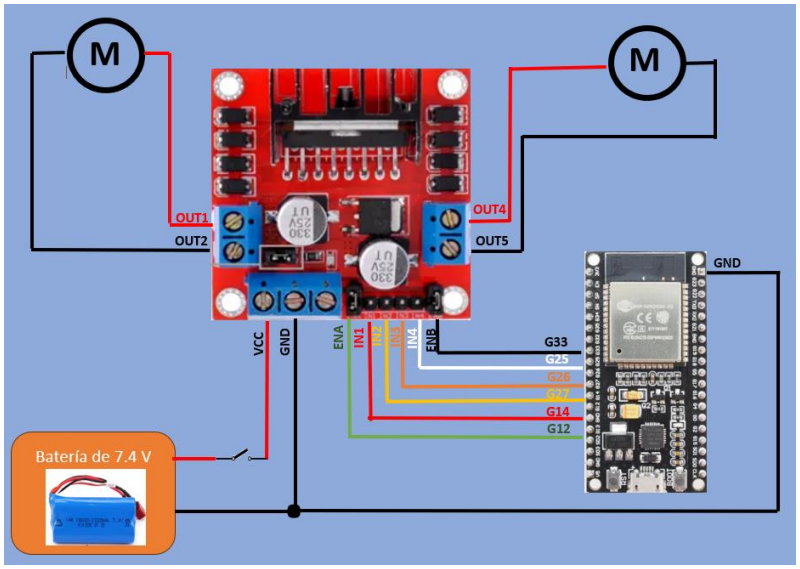

Figura 13.- Diagrama de conexiones entre el controlador de motores, los motores, la fuente de alimentación de 7.4 V y la placa ESP32.

Los motores y el controlador se alimentan conectando el polo positivo de la batería de 7.4 voltios, que es el cable rojo, al que se le ha añadido un interruptor por la seguridad de los componentes, a la entrada Vcc del controlador. El polo negativo se conecta tanto

al *ground* de la placa ESP32 como al del controlador. Los pines G12, G14, G27, G26, G25 y G33 se establecen como pines de salida y se conectan a los pines del controlador ENA, IN1, IN2, IN3, IN4 y ENB respectivamente. Como se ha explicado anteriormente, la velocidad de los motores A y B se controla introduciendo señales PWM a través de los pines ENA y ENB respectivamente. Por otro lado, la dirección de giro de los motores se controla estableciendo un estado alto o bajo en los demás pines, como se explicó en la Tabla 2. Por último, las salidas del controlador OUT1, OUT2, OUT3 y OUT4 se conectan a los motores y estas les transmiten las diferencias de voltaje necesarias para el funcionamiento deseado.

4.1.3. Conexión cámara y ordenador

La conexión entre la cámara del móvil y el ordenador se establece a partir de instalar la aplicación para móviles inteligentes, IP Webcam. Dicha aplicación es capaz de crear un servidor en el que carga la imagen captada en tiempo real por la cámara del teléfono. La dirección del servidor se introduce en el código de Python y el programa adquiere la imagen sobre la que se realizará la detección del objeto que se describe en el capítulo próximo. Una vez se extrae la información que se concreta en dicho capítulo, esta se envía a la placa ESP32 a través del puerto serie del ordenador conectado vía Bluetooth con un puerto serie de la propia placa.

Con el objetivo de sujetar teléfono al chasis se emplea una pieza que, como se ve en la Figura 10, se adhiere a la parte frontal del vehículo y sirve como soporte para el móvil.

4.2. Algoritmo de detección. Entrenamiento e implementación de YOLOV5

Una vez se ha seleccionado el modelo de detección equipado por el vehículo se procede a su entrenamiento e implementación en el algoritmo de detección de objetos que ejecuta el ordenador.

4.2.1. Creación de la base de datos

El primer paso del entrenamiento del modelo de detección de imágenes YOLOv5s consiste en crear una base de datos personalizada. Esta base de datos se compone de una gran cantidad de imágenes del objeto a detectar, como la mostrada en la Figura 14. Concretamente se han tomado 109 imágenes, ya que entrenando la red neuronal con esta cantidad se ha obtenido un modelo que funciona, aunque si se busca más fiabilidad y precisión se debe aumentar el tamaño de la base de datos. En este caso, el objeto es un aro, aunque podría ser cualquier objeto, como por ejemplo un tipo de piedra que se busque en marte o un océano helado.

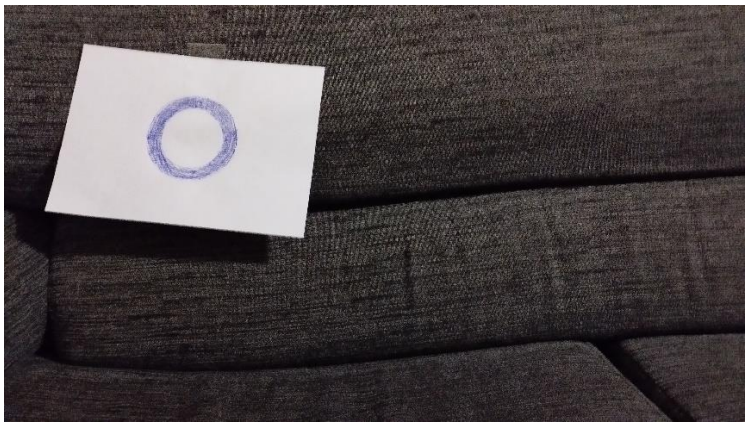

Figura 14.- Imagen ejemplo de la base de datos personalizada.

Cada una de las imágenes está acompañada por un documento, de extensión *.txt,* que contiene información de la ubicación del objeto dentro de la imagen. Cada documento contiene tantas filas de números como objetos a detectar contenga la imagen vinculada. Cada fila contiene, a su vez, cinco números separados por espacios como se observa en la Figura 15, que muestra el documento adjunto a la imagen de la Figura 14.

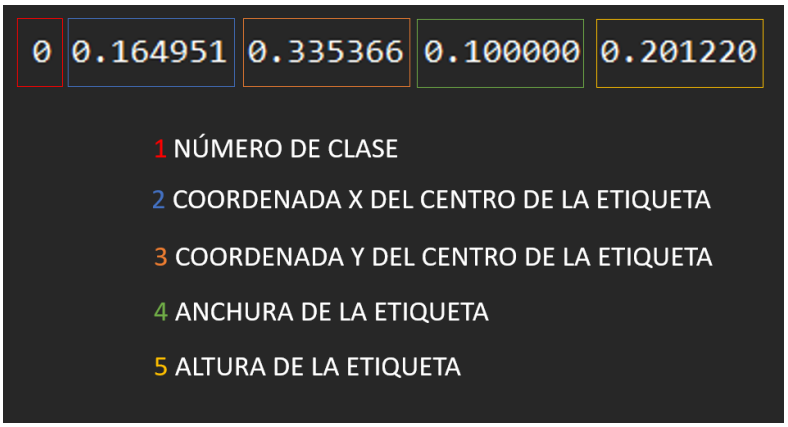

Figura 15.- Documento adjunto a cada imagen de la base de dato. Información de cada dígito.

El primero indica el número de la clase, ya que un mismo modelo puede ser entrenado para detectar múltiples clases. Una clase es el objeto por detectar, por ejemplo, un gato, un coche o, como es el caso de estudio, un aro. En este caso se entrena a la red neuronal para detectar una única clase, el aro ya mencionado, a la que se le asigna al número cero. El segundo número es la coordenada en el eje horizontal del centro de la etiqueta, siendo cero cuando se encuentra en el primer píxel desde la izquierda y uno en el último. El tercer número es la coordenada en el eje vertical del centro de la etiqueta, siendo cero cuando se encuentra en el primer píxel desde la arriba y uno en el último. El cuarto número es el ancho de la etiqueta y el quinto y último número, el alto.

El proceso de generar el documento de cada imagen, o etiquetado, se ha realizado con la ayuda de la herramienta *labelImg*, donde se introducen las imágenes y manualmente se recuadra el aro en cada una de ellas. Una vez realizado este paso el programa genera

los documentos automáticamente y los almacena en una carpeta, con lo que se da por finalizada la creación de la base de datos.

Las carpetas de las imágenes y las etiquetas se unen en una única carpeta, se comprimen y se suben a Google Drive para acceder a ellas en la fase de entrenamiento.

4.2.2. Entrenamiento de YOLOV5

El entrenamiento se realiza mediante el fichero tutorial que se proporciona en la propia documentación de YOLOv5. Este fichero se ejecuta en Google Colab, una herramienta que permite programar y ejecutar Python en el navegador con la ventaja de que proporciona acceso a GPUs (Graphics Processing Unit) de manera gratuita. Esto es de gran ayuda si no se dispone de una GPU, ya que es un hardware caro y en este caso acelera en gran medida el proceso de entrenamiento del modelo de detección de objetos.

El primer paso que se ejecuta en el entorno de Google Colab consiste en instalar los requerimientos para poder realizar el entrenamiento, los cuales se encuentran con más detalle en la documentación de YOLOv5 [15]. Después, se importa al directorio de trabajo de Google Colab la carpeta de las etiquetas y las imágenes desde Google Drive, donde se había subido anteriormente, y se descomprime.

A continuación, se confecciona un archivo al que se nombra como *custom.yalm* y se muestra en la Figura 16. Este archivo contiene información sobre la dirección de la carpeta que contiene la base de datos personalizada, la dirección de las imágenes de entrenamiento y de validación, además del número de clases, una en este caso, y su nombre, *circle*. Una vez creado, este archivo se sube al directorio de Google Colab en el que se está trabajando.

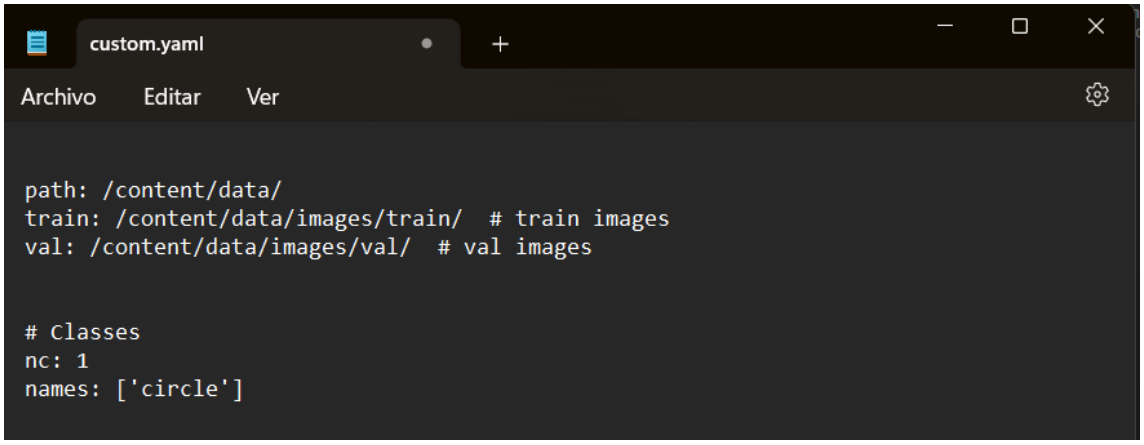

Figura 16.- Documento custom.yalm introducido en el entrenamiento en Google Colab.

Por último, se ejecuta la línea de código que inicia el entrenamiento. En esta línea se introduce la dirección del archivo *custom.yalm*, la versión de YOLOv5 seleccionada para

ser entrenada, en este caso YOLOv5s, y el número de épocas de entrenamiento, entre otros parámetros.

En un primer entrenamiento, se entrena al modelo durante un total de 100 épocas y se analiza el resultado. Puesto que los indicadores de la calidad del detector, definidos anteriormente, no han dejado de mejorar antes de la finalización del entrenamiento, se entrena una segunda vez durante 200 épocas.

4.2.3. Extracción del modelo y análisis de los resultados

En este segundo intento, donde se entrena a la red neuronal durante 200 épocas, sí que se aprecia, en la Figura 17, un estancamiento de los indicadores con el paso de las épocas, cuyos últimos valores se muestran en la Tabla 3, por lo que se da por bueno el número de épocas establecido.

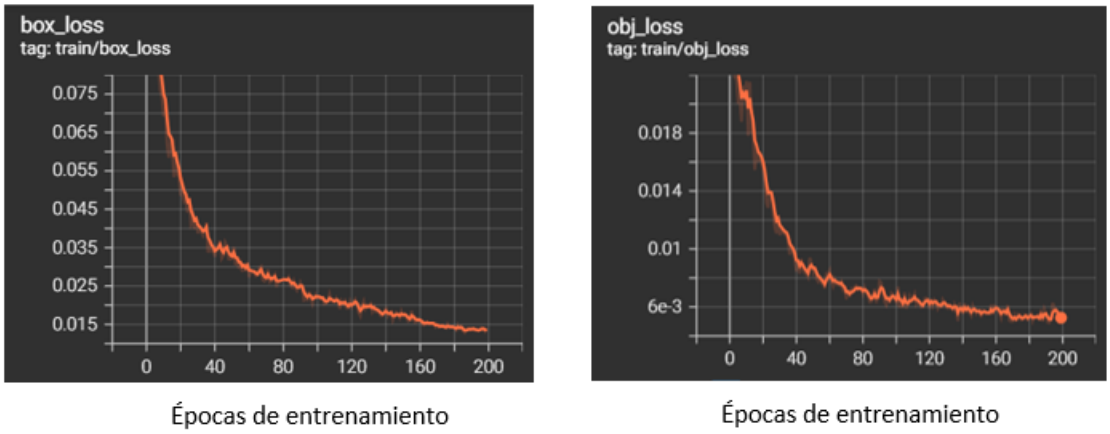

Figura 17.- Evolución con el aumento de las épocas de entramiento de box loss y objectness loss.

Como se muestra en la Tabla 3, los valores de las pérdidas de marco y de objetividad son significativamente bajos por lo que se considera que el modelo de detección resultante es lo suficientemente preciso.

Épocas	*Box loss*	*Objectness loss*
200	0.0138	0.0051

Tabla 3.- Valores de box loss y objectness loss obtenidas como resultado tras el entrenamiento durante 200 épocas.

Además, en la Figura 18, donde se muestra la evolución de los parámetros *recall* y precisión a lo largo de las épocas de entrenamiento, se aprecia que ambos alcanzan practicamente el rendimiento máximo sobre las imágenes de entrenamiento, que no implica necesariamente que se alcance el mismo rendimiento sobre nuevas imágenes, pero es un indicador de un entrenamiento correcto.

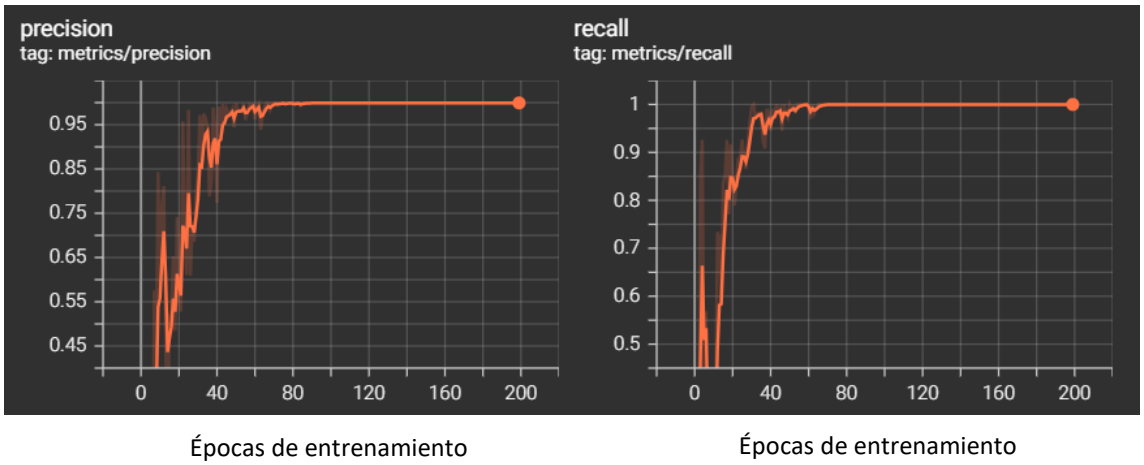

Figura 18.- Evolución de la precisión y el recall de la red neuronal entrenada con el aumento de las épocas de entrenamiento.

Como resultado, se obtiene un archivo, al que se ha nombrado *bestcircle200.pt*, que contiene modelo ya entrenado para detectar el aro en cualquier imagen que reciba. Para comprobar su correcto funcionamiento, se prueba sobre una imagen y, como se muestra en la Figura 19, se realiza la detección con éxito recuadrando en rojo el objeto.

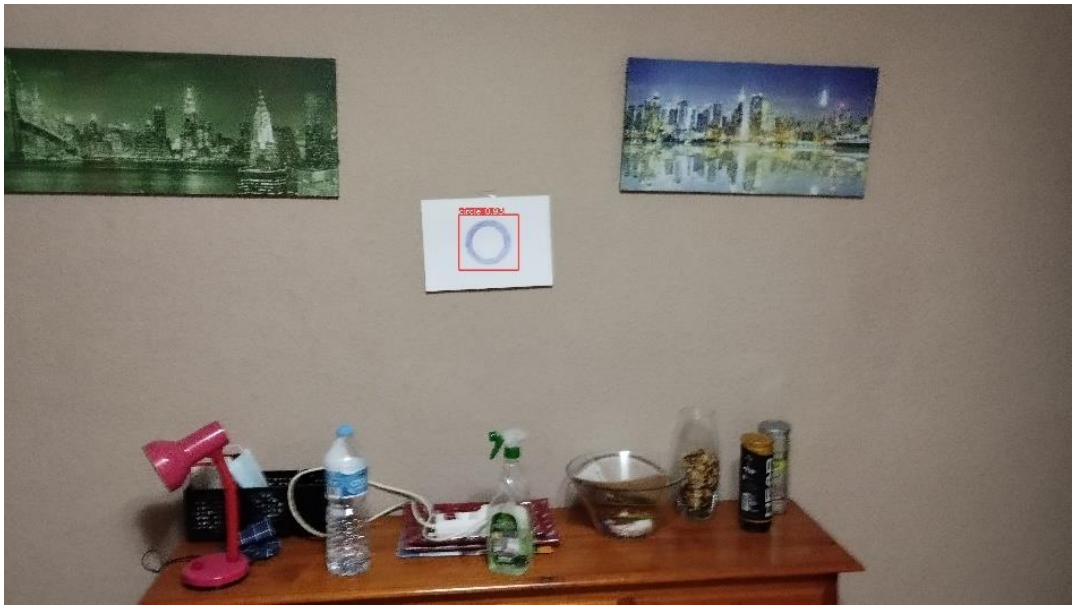

Figura 19.- Resultado de la detección del objeto sobre una imagen aleatoria de la base de datos.

4.2.4. Implementación en Python

Una vez obtenido el modelo entrenado, se implementa en el código de Python para realizar detecciones del objetivo sobre las imágenes que recibe el ordenador desde la cámara. El modelo de detección, una vez cargado en el código de Python, se emplea en el siguiente algoritmo que se describe más adelante, y cuyo código se ve en detalle en el Anexo I.I.

La información que es interesante extraer, aparte de si hay o no detección, es la ubicación del objeto dentro de la imagen. Esta información determinará, como se

explica a continuación, la orientación de referencia θ_{ref} que el controlador PID del vehículo tratará de alcanzar.

Al realizar las detecciones en tiempo real se obtiene información de la ubicación del recuadro que contiene la imagen en el eje horizontal. Las coordenadas obtenidas son x_{min} y x_{max}, que toman un valor entre el cero, que corresponde al extremo izquierdo de la imagen, y el valor del ancho de la imagen, que corresponde al extremo derecho. En este caso se ha configurado un ancho de imagen de 700 píxeles. Una vez se tiene ambas coordenadas se realiza el promedio para obtener la coordenada horizontal del centro del recuadro. A esta, como se muestra en la Ecuación (13), se le resta la mitad del ancho de la imagen, y de este modo se consigue la distancia al centro de la imagen d_{obj}, siendo la mitad del ancho cuando se encuentra en el extremo izquierdo y el mismo valor, pero con signo negativo, cuando está en el extremo derecho.

$$d_{obj} = -\left(\frac{(x_{max} + x_{min})}{2} - \frac{Ancho}{2}\right) \tag{13}$$

A partir de esta distancia d_{obj} será posible calcular el ángulo del objetivo respecto al centro de la imagen θ_{obj}, representado en la Figura 20, aunque antes es necesario conocer en ángulo de visión α de la cámara del vehículo.

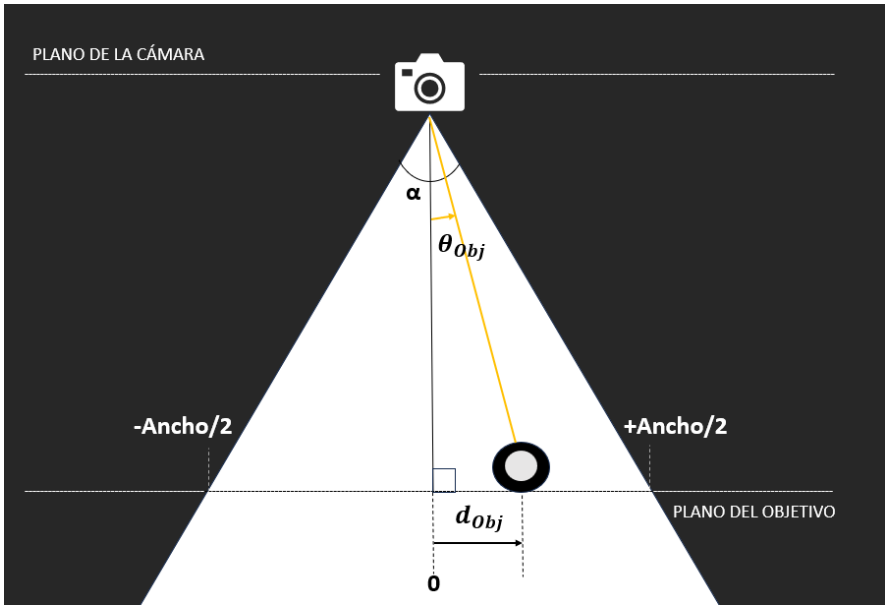

Figura 20.- Representación de los parámetros que definen la posición del objetivo.

Ángulo de visión de la cámara

El ángulo de visión α de la cámara determina qué porción del panorama se refleja sobre el sensor de la cámara. Esta magnitud se muestra gráficamente en la Figura 21, donde se representa con la letra griega alfa.

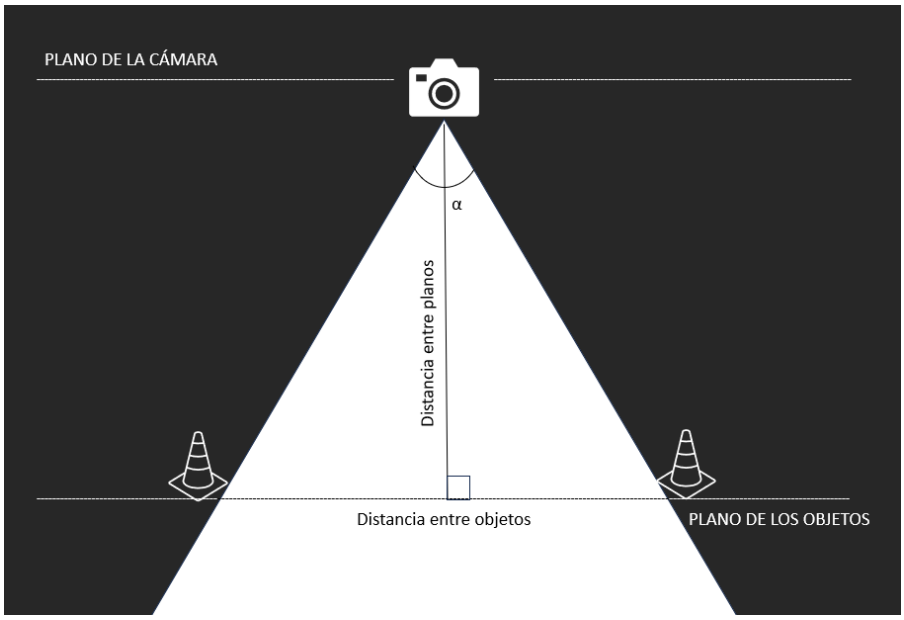

Figura 21.- Metodología empleada para calcular el ángulo de visión.

Con el fin de calcular su valor, se realiza el montaje representado en la Figura 21, que consiste en ubicar dos objetos en un plano paralelo al plano de la cámara, ubicados en los extremos horizontales de la imagen. Midiendo la distancia entre ambos objetos $d_{objetos}$ y la distancia d_{planos} entre el plano de la cámara y el plano de los objetos, y sustituyendo los valores en la Ecuación (14), se alcanza el valor del ángulo de visión α que se buscaba.

$$\alpha = 2\arctan\left(\frac{\frac{1}{2}d_{objetos}}{d_{planos}}\right) = 60.96° \tag{14}$$

A partir de este valor y de aplicar trigonometría sobre los triángulos rectángulos mostrados en la Figura 20 se tiene como resultado la Ecuación (15) del ángulo del objetivo respecto al centro de la cámara θ_{obj}.

$$\theta_{obj} = \arctan\left(\frac{d_{obj}\ \tan(\alpha/2)}{\frac{1}{2}Ancho}\right) \tag{15}$$

En la Ecuación (16), este ángulo se añade a la orientación actual del vehículo θ, medida por la IMU, para formar el ángulo de referencia que se envía vía Bluetooth a la placa ESP32.

$$\theta_{ref} = \theta + \theta_{obj} \tag{16}$$

El método descrito no es exacto, ya que en la Figura 22 se puede apreciar que debido a que la cámara y el eje del vehículo no están alineados, la orientación calculada, del objetivo respecto a la cámara, no es exactamente igual a la orientación del objetivo respecto al eje del vehículo. Es posible corregir dicha imprecisión si se conoce la distancia a la que se encuentra el objetivo, pero ya que no se dispone de dicha información se desprecia este efecto.

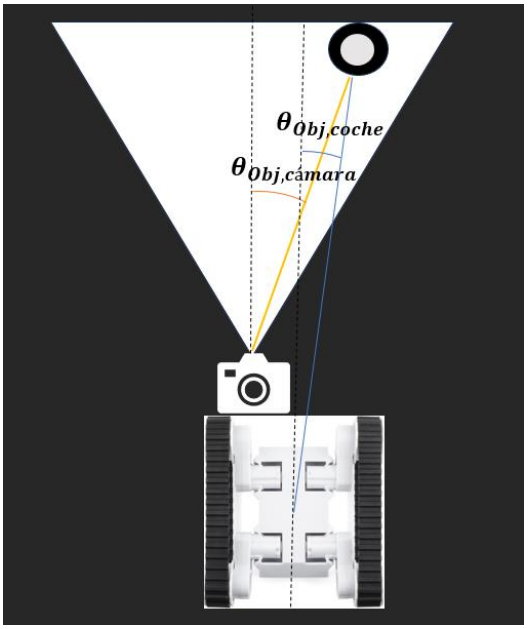

Figura 22.- Diferencia del ángulo del objetivo respecto al vehículo y a la cámara.

Corrección del retraso por la detección

Un efecto que es sí posible considerar y corregir es el que se describe a continuación. Durante el tiempo que transcurre desde que se toma la imagen hasta que se envía la información de la detección a la placa de desarrollo, el vehículo se encuentra en movimiento. Esto provoca que se introduzca un error angular que es posible calcular mediante la Ecuación (17), que multiplica el tiempo de retraso originado por la detección, $t_{retraso}$, por la velocidad angular media durante este tiempo ω_{media}.

$$\theta_{rot} = \omega_{media}\, t_{retraso} \tag{17}$$

El tiempo de retraso es el resultado de calcular la diferencia entre el tiempo en el momento de tomar la imagen de la cámara, y el tiempo tras realizar la detección. Los tiempos se obtienen mediante la función *time()* del módulo homónimo de Python. Dicha función devuelve el número de segundos transcurridos desde la época (1 de enero de 1970, a las 00:00:00 UTC).

El tiempo de retraso se envía, junto al ángulo de referencia, a la placa ESP32. En la memoria del ESP32 se almacenan, en un vector, las últimas velocidades angulares medidas por la IMU, además del periodo con el que se toman las mediciones, que será

prácticamente constante. Una vez la placa ESP32 recibe el tiempo de retraso, se divide entre el periodo de mediciones y el resultado, redondeado a un número entero, será el número de las últimas medidas de las velocidades angulares que se promedian para obtener ω_{media}. Esta corrección se aplica sobre la Ecuación (16), dando como resultado la Ecuación (18).

$$\theta_{ref} = \theta + \theta_{obj} - \theta_{rot} \tag{18}$$

El resultado de la Ecuación (18) es el ángulo de referencia que se introduce como referencia en el bucle de control, cuyo controlador PID se desarrolla más adelante.

Algoritmo de detección del objeto

El algoritmo de detección que se ejecuta en Python se representa gráficamente en el diagrama de la Figura 23.

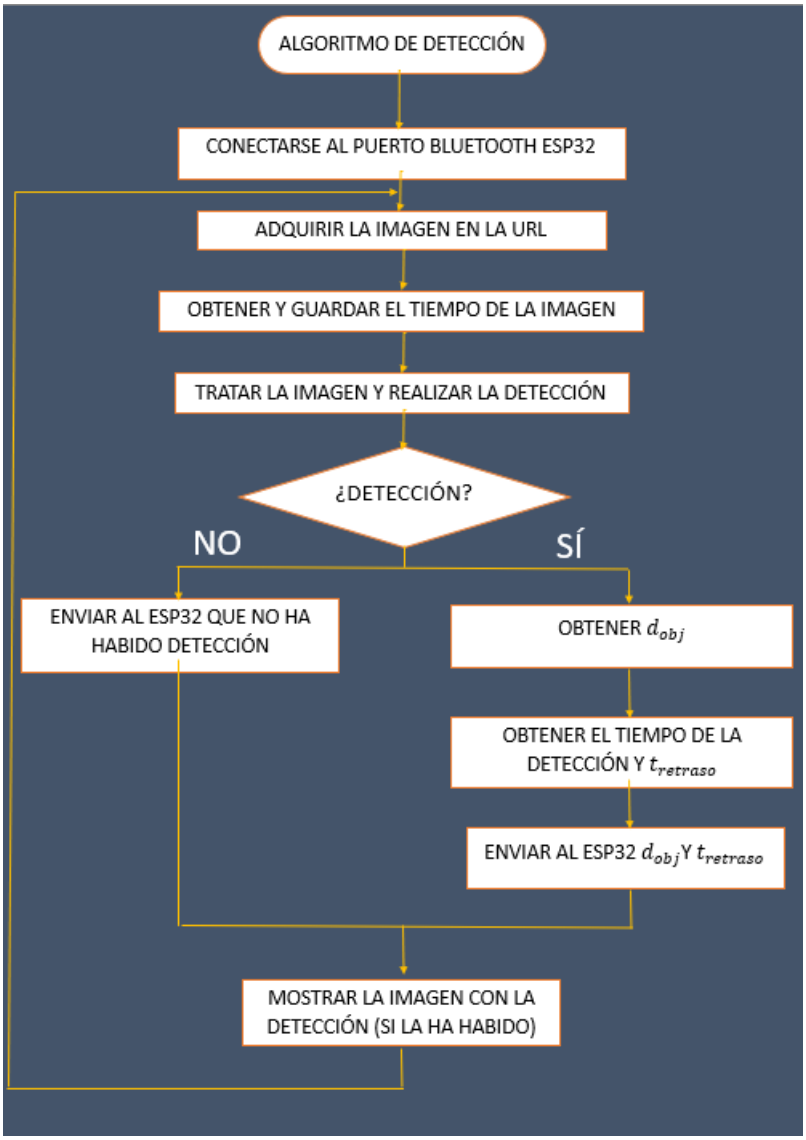

Figura 23.- Algoritmo de detección del objeto.

El primer paso consiste en conectarse al puerto asignado a la conexión vía Bluetooth con el ESP32. Después, se obtiene la última imagen, captada por la cámara, mediante el servidor creado en IP Webcam. Esta imagen se trata y se le realiza la detección del objeto. En el caso de detectar el objetivo, se extrae la información necesaria para calcular d_{obj} y $t_{retraso}$ y los resultados se envían separados por comas en un *string* a el ESP32 a través del puerto previamente asignado. En cambio, si no se produce la detección del objeto se informará de ello al ESP32. El último paso consiste en mostrar la imagen, con la detección remarcada en el caso de haberla habido, y repetir el proceso desde el segundo paso.

4.3. Diseño e implementación del PID

Un buen diseño de las constantes del controlador es indispensable para obtener una respuesta adecuada ante la entrada de un cambio en la referencia. En primer lugar, se define cuál será la variable controlada, es decir, la que se busca controlar ya que describe el comportamiento del vehículo, y cuál es la variable manipulada, sobre la que se podrá actuar para influir en la variable controlada. El objetivo del vehículo es ir hacia una referencia visual, por lo que la dirección de avance debe estar alineada con el propio objetivo. Dicha dirección de avance corresponde con el ángulo de orientación del vehículo θ, ya que las ruedas están fijas, por lo que la velocidad del vehículo siempre es tangencial a su orientación θ. Por lo tanto, la orientación del robot θ, que será medida por la IMU, se establece como variable controlada.

La única variable manipulable capaz de influir sobre la variable controlada definida es la diferencia de velocidad angular entre las ruedas izquierda y derecha Δw. Esta será entonces la variable manipulada.

A continuación, se obtiene un modelo cinemático que relaciona las variables definidas.

4.3.1. Modelo cinemático

Con el objetivo de desarrollar un modelo cinemático del robot que relacione θ y Δw, se establecen una serie de simplificaciones. En primer lugar, se considera el vehículo como un robot con dos ruedas sencillas de dirección fija, como el que se puede observar en la Figura 24.

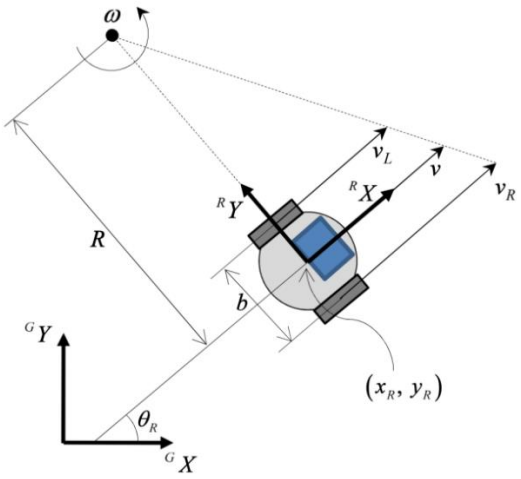

Figura 24.- Representación del vehículo como un robot con dos ruedas sencillas de dirección fija [1].

Además, no se considera que exista deslizamiento entre la rueda y el suelo, por lo que la velocidad de cada de una de las ruedas, izquierda v_L y derecha v_R, viene dada por la Ecuación (19), siendo w_i la velocidad angular de cada una de las dos ruedas y r su radio.

$$v_i = r\, w_i \tag{19}$$

Cuando existe una diferencia de velocidades entre las ruedas se produce un giro alrededor del centro instantáneo de rotación con una velocidad angular ω. La velocidad angular se puede definir como la derivada de la velocidad tangencial respecto al radio de giro. Puesto que la velocidad angular aumenta linealmente a lo largo del radio, o lo que es lo mismo, la velocidad angular es constante, esta se calcula como la relación entre el incremento entre las ruedas izquierda y derecha de la velocidad tangencial y del radio:

$$\dot{\theta} = \omega = \frac{v_R - v_L}{b} \tag{20}$$

Como se observa en la Figura 24, b es la distancia entre las ruedas. Sustituyendo la Ecuación (19) en (20) finalmente da como resultado la Ecuación (21).

$$\dot{\theta} = \frac{r\,(w_R - w_L)}{b} = \frac{r\,\Delta w}{b} \tag{21}$$

Se han agrupado las variables $w_R \, y \, w_L$ en una única variable, la diferencia de las velocidades angulares Δw. De este modo será más sencillo realizar el diseño del controlador ya que se trata de un sistema de una sola entrada Δw y una sola salida θ.

4.3.2. Diseño del PID de 2 grados de libertad

Una vez se ha desarrollado el modelo cinemático y se ha obtenido la ecuación diferencial (21), se convierte al dominio de la frecuencia aplicando la transformada de Laplace, tomando condición inicial nula $\theta(0) = 0$, como se muestra en la Ecuación (22).

$$L[\dot{\theta}] = L\left[\frac{r\,\Delta w}{b}\right]$$

$$s\,\theta(s) = \frac{r\,\Delta w(s)}{b} \tag{22}$$

Reordenando y sustituyendo en la Ecuación (22) los valores propios del Rover 5 del radio de las ruedas r y de la distancia entre ellas b, queda la función de transferencia $G(s)$ que relaciona las variables manipulada y controlada:

$$G(s) = \frac{\theta(s)}{\Delta w(s)} = \frac{0.1579}{s} \tag{23}$$

Con el fin de realizar el diseño del controlador, se implementa el bucle mostrado en la Figura 5, en la herramienta de Matlab, *simulink*. De esta manera será posible simular la respuesta del sistema con diferentes controladores, referencias, perturbaciones, etc. Además, se empleará la herramienta de Matlab, *pidTuner*, para ajustar las constantes del controlador PID y conseguir la respuesta deseada.

Como se puede observar en la Ecuación (23), la función de transferencia únicamente tiene un polo en $s = 0$. Esto indica que es constante en la fase estacionaria, aunque ante la introducción de una perturbación, el sistema no es capaz de volver a la referencia establecida. Para demostrarlo, se ha realizado una simulación sin el filtro de ponderación de la referencia y un controlador que únicamente contará con la parte proporcional. La respuesta del sistema se muestra en la Figura 25, donde el filtro F y el PID utilizados han sido los siguientes:

$$F = 1$$

$$PID = 16.1$$

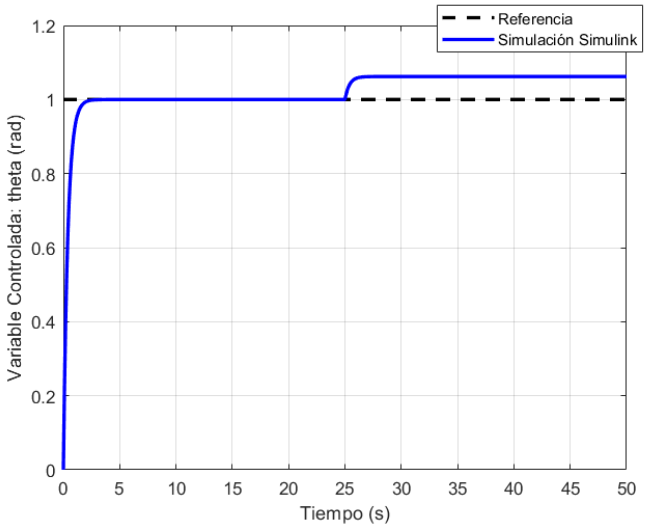

Figura 25.- Evolución temporal de la respuesta debida a la acción proporcional.

Como se observa en la Figura 25, inicialmente el controlador proporcional funciona correctamente en el seguimiento de la referencia hasta la entrada de una perturbación a los veinticinco segundos de empezar la simulación. Es entonces cuando el controlador es incapaz devolver a la variable controlada θ al valor de la referencia. Es por ello por lo que se introduce la acción del integrador, cuyo efecto se muestra en la Figura 26. También se ha añadido la acción del término derivativo en esta segunda simulación, que se opone a la variación del error, consiguiendo que el pico del error ante la perturbación sea menor. El filtro F y el PID utilizados en la simulación de la Figura 26 son los siguientes:

$$F = 1$$

$$PID = 16.1 + \frac{7.59}{s} + \frac{s}{0.103\,s + 1}$$

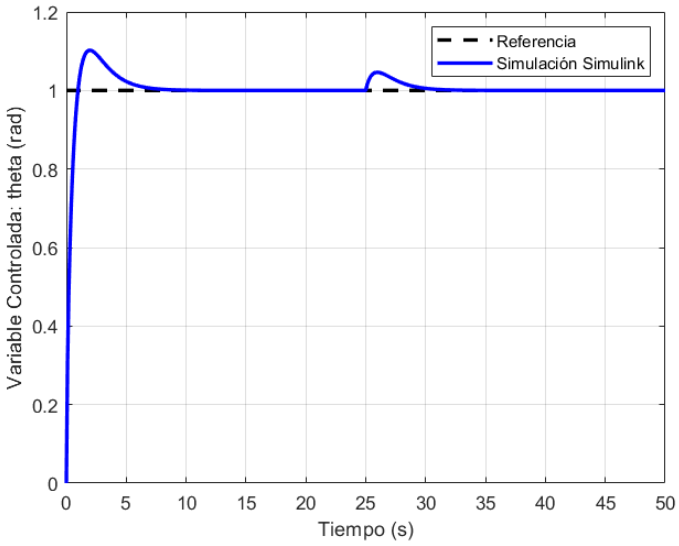

Figura 26.- Evolución temporal de la respuesta debida a las acciones proporcional, derivativa e integral.

Analizando la diferencia entre las Figuras 25 y 26, se puede apreciar el efecto de las acciones integral y derivativa, que consiguen una buena respuesta ante la perturbación. El problema es que, al introducir estas acciones, la respuesta ante los cambios en la referencia empeora ya que aparece una sobreoscilación. Para mejorar la respuesta del sistema se introduce el filtro de ponderación de la referencia, que multiplica a la referencia en los términos proporcional y derivativo. Este controlador PID de dos grados de libertad permite rechazar las perturbaciones sin incrementar la sobreoscilación en el seguimiento de la referencia [9]. El filtro F y el PID utilizados en la simulación de la Figura 27 son los siguientes:

$$F = \frac{0.35\ (s + 10.27)(s + 0.87)}{(s + 5.93)(s + 0.52)}$$

$$PID = 16.1 + \frac{7.59}{s} + \frac{s}{0.103\ s + 1}$$

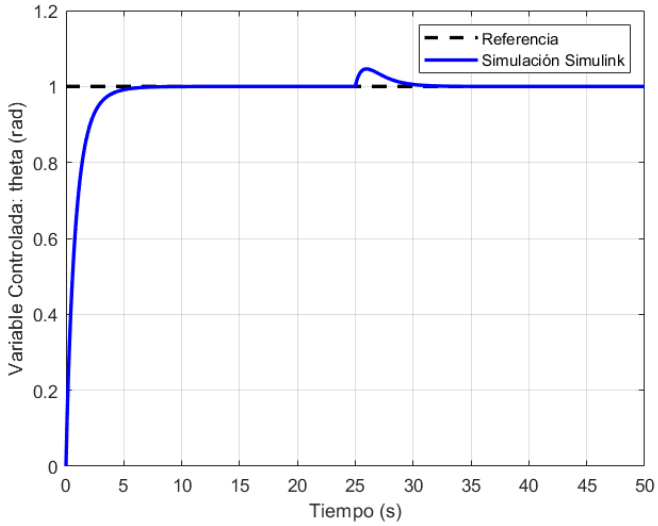

Figura 27.- Evolución temporal de la respuesta debida a la acción del controlador PID de dos grados de libertad.

En la Figura 27 se observa un comportamiento del sistema idóneo, capaz de realizar un seguimiento de la referencia sin sobreoscilación y, a los veinticinco segundos, de rechazar la perturbación con éxito. Puesto que se ha alcanzado un resultado satisfactorio se procede a implementar el controlador diseñado en el vehículo real.

4.3.3. Discretización

El controlador se implementa en un entorno digital como lo es el equipado por el robot, por lo que las operaciones de control se efectúan en instancias de tiempo discreto. La secuencia de operaciones que se realizará en el bucle implementado en el código de Arduino que se carga a la placa de desarrollo ESP32 es la siguiente:

1) Leer la medida del sensor de la variable controlada θ y establecerla como entrada.
2) Computar la señal de control.
3) Ejecutar la acción de control.
4) Actualizar las variables del controlador.
5) Realizar una espera que define el periodo de muestreo.

Con el fin de implementar el PID de dos grados de libertad y poder computar la señal de control, es necesario aproximar la Ecuación (10), que se trata de una función continua, a una función discreta. Se emplean las aproximaciones desarrolladas en el trabajo de Amströn et. al. [7]:

Acción Proporcional

Como se ha visto anteriormente, el término proporcional expresado con las variables propias del caso de estudio es:

$$\Delta w_p = K_p\left(b\,\theta_{ref} - \theta\right)$$

Este término se implementa reemplazando las variables continuas por sus versiones discretas, en función del instante de muestreo t_k, resultando en la Ecuación (24).

$$\Delta w_p(t_k) = K_p(b\ \theta_{ref}(t_k) - \theta(t_k)) \tag{24}$$

Acción Integral

Extrayendo el término integral de la Ecuación (10) y sustituyendo las variables por las propias del caso de estudio:

$$\Delta w_i(s) = \frac{K_i}{s}\big(\theta_{ref}(s) - \theta(s)\big)$$

Aplicando a ambos lados de la igualdad la transformada inversa de Laplace, se transfiere del dominio de la frecuencia al dominio temporal:

$$\Delta w_i = K_i \int_0^t \big(\theta_{ref} - \theta\big)\, dt$$

Derivando respecto al tiempo a ambos lados de la igualdad:

$$\frac{d\Delta w_i}{dt} = K_i\big(\theta_{ref} - \theta\big)$$

Se aproxima la derivada como el incremento de la función Δw_i entre un instante de muestreo t_k y un instante previo t_{k-1}, en relación con el tiempo transcurrido entre ambas medidas T. También se reemplazan las variables continuas θ_{ref} e θ por sus versiones discretas $\theta_{ref}(t_k)$ e $\theta(t_k)$:

$$\frac{\Delta w_i(t_k) - \Delta w_i(t_{k-1})}{T} = K_i\left(\theta_{ref}(t_k) - \theta(t_k)\right)$$

Reordenando se obtiene la Ecuación (25), que será la introducida en el código y calcula la aportación del término integral Δw_i a la acción de control en instancias de tiempo discreto.

$$\Delta w_i(t_k) = \Delta w_i(t_{k-1}) + T\ K_i(\ \theta_{ref}(t_k) - \theta(t_k)) \tag{25}$$

El periodo entre iteraciones T se calcula mediante la función de Arduino, *millis()*. Esta devuelve el número de milisegundos transcurridos desde el inicio de la ejecución del código de Arduino. Al albergar el valor de este tiempo y realizar la diferencia respecto al de la iteración anterior se obtiene el tiempo entre ambas repeticiones del bucle.

Acción Derivativa

De manera análoga a los apartados anteriores se extrae el término derivativo de la Ecuación (10):

$$\Delta w_d(s) = \frac{K_d\ s}{T_f\ s + 1}(c\ \theta_{ref}(s) - \theta(s))$$

De nuevo, se transfiere al dominio temporal aplicando a ambos lados la transformada inversa de Laplace:

$$T_f \frac{d\Delta w_d}{dt} + \Delta w_d = K_d \frac{d\big(c\ \theta_{ref} - \theta\big)}{dt}$$

Aplicando la aproximada de la derivada y reemplazando las variables continuas nuevamente:

$$T_f \frac{\Delta w_d(t_k) - \Delta w_d(t_{k-1})}{T} + \Delta w_d(t_k) =$$

$$= K_d \left(c \left(\frac{\theta_{ref}(t_k) - \theta_{ref}(t_{k-1})}{T} \right) - \left(\frac{\theta(t_k) - \theta(t_{k-1})}{T} \right) \right)$$

Reordenando se obtiene la Ecuación (26), que se introduce en el código y calcula la aportación de la acción derivativa en la acción de control:

$$\Delta w_d(t_k) = \tag{26}$$

$$= \frac{T_f}{T_f + T}\Delta w_d(t_{k-1}) + \frac{K_d}{T_f + T}\Big(c\big(\theta_{ref}(t_k) - \theta_{ref}(t_{k-1})\big) - (\theta(t_k) - \theta(t_{k-1}))\Big)$$

Finalmente, en la Ecuación (27) se suman las acciones de control de cada uno de los tres términos, lo que da como resultado la acción de control total que se ejercerá sobre el sistema.

$$\Delta w(t_k) = \Delta w_p(t_k) + \Delta w_i(t_k) + \Delta w_d(t_k) \tag{27}$$

Con el propósito de comprobar la validez de las aproximaciones que han resultado en la función discreta de la Ecuación (27), se realiza una nueva simulación en Matlab que grafique la respuesta temporal de la implementación de dicha aproximación en código. Esta respuesta es comparada en la misma gráfica con la respuesta temporal obtenida en

la simulación que emplea el mismo controlador, pero en su forma continua, implementado en el diagrama de *simulink* mostrado en la Figura 39. La simulación se realiza para un periodo de muestreo $T = 0.1\ s$, ya que será el utilizado en el sistema real. La gráfica resultante se muestra en la Figura 28.

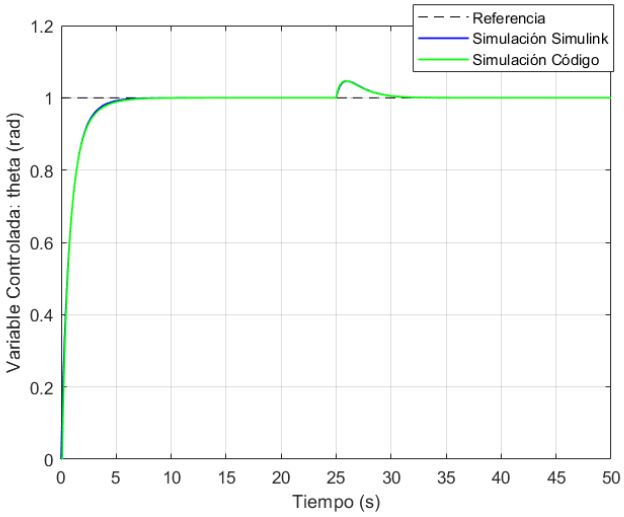

Figura 28.- Comparación de la respuesta del sistema contínuo (simulink) vs el sistema en forma discreta (código).

Analizando la Figura 28 apenas se aprecian diferencias entre ambas simulaciones. Además, calculando el error absoluto en cada iteración se obtiene un error absoluto promedio de 0.0015 radianes, por lo que se concluye que las aproximaciones extraídas del trabajo se Amström et. al. [7] son válidas para ser implementadas en el robot real con el periodo de muestreo establecido. En el caso de buscar más exactitud con respecto al modelo continuo sería necesario aumentar el periodo. Por ejemplo, con $T = 0.01\ s$ se tendría un error absoluto promedio de $1.50 \cdot 10^{-4}$ radianes.

4.3.4. Ejecución de la acción de control

Una vez se ha obtenido la acción de control saliente del PID de dos grados de libertad, se debe aplicar en el vehículo. Se recuerda que la acción de control en el caso de estudio consiste en variar la diferencia de velocidad angular entre las ruedas izquierda y derecha del robot, es decir, la variable manipulada elegida.

Recuperando la Ecuación (12) y expandiendo la diferencia de velocidad angular Δw:

$$\Delta w = w_R - w_L = \delta_R\, w_{max} - \delta_L\, w_{max} \tag{28}$$

Como se observa en la Ecuación (28) quedan tres incógnitas: los ciclos de trabajo de las señales suministradas a los motores δ_R y δ_L, y la velocidad angular máxima w_{max} de las

ruedas. En primer lugar, se mide experimentalmente w_{max} suministrando un ciclo de trabajo del 100% a ambos motores y cronometrando el tiempo que tardan las ruedas en realizar diez vueltas completas. De esta manera se diluye el error de medición entre las diez vueltas. Se repite el proceso de medición varias veces y se toma el tiempo promedio, que es de 7.15 segundos. Por lo tanto, las ruedas dan una vuelta cada 0.715 segundos. La velocidad angular resulta:

$$w_{max} = \frac{2\pi}{T_{vuelta}} = 8.79\ \frac{rad}{s} \tag{29}$$

En cuanto a los ciclos de trabajo, dependiendo de si la acción de control es positiva o negativa, se fijará en un valor constante a δ_R o δ_L respectivamente. Aplicando esta condición en la Ecuación (28) se obtienen las Ecuaciones (30) y (31).

Si $\Delta w > 0$:

$$\delta_L = \frac{\delta_{fijo}\, w_{max} - \Delta w}{w_{max}} \tag{30}$$

Si $\Delta w < 0$:

$$\delta_R = \frac{\delta_{fijo}\, w_{max} + \Delta w}{w_{max}} \tag{31}$$

El valor constante δ_{fijo} dictará a qué velocidad avanza el vehículo hacia el objetivo cuando lo tiene orientado de manera estable, ya que cuando la acción de control es nula, el ciclo de trabajo variable debe ser igual al constante. En cambio, si la acción de control es no nula, el ciclo de trabajo variable se reduce, hasta un mínimo de detener la rueda, con el fin de cumplir la acción de control. Es decir, con este método se garantiza no solo que el robot se oriente al objetivo, sino que también avance hacia el mismo a una velocidad ajustable con el ciclo de trabajo fijo δ_{fijo}.

Por último, se configuran los parámetros explicados en la Sección 5.1.4. con el fin de generar las señales PWM con los ciclos de trabajo obtenidos de las Ecuaciones (30) y (31). Los canales asignados serán el Canal 0 para el Motor A y el Canal 1 para el Motor B. La frecuencia seleccionada es de 1000 Hz, ya que suele ser adecuada para motores de baja potencia como lo son los equipados por el robot. Para acabar, la resolución elegida es de 15 bits.

Los parámetros anteriores se introducen en el código de Arduino mediante las funciones de la librería LEDC, aplicando así la acción de control al vehículo.

4.3.5. Saturación y *Anti-windup*

A pesar de que muchos aspectos del control de un sistema se pueden entender a partir de la teoría lineal, hay algunos efectos no lineales que se deben tener en cuenta. Todos los actuadores tienen limitaciones, incluidos los motores, que son los actuadores en el caso de estudio. Los motores tienen un límite de velocidad que no son capaces de sobrepasar, y este hecho se debe tener en cuenta a la hora de diseñar el controlador, ya que la variable manipulada se puede ver limitada por esta condición [7].

En un sistema lineal ideal como el que se ha implementado en Matlab para realizar el diseño del controlador, esta limitación no se produce, por lo que se puede incurrir en el error de diseñar un controlador que constantemente ordene acciones de control demasiado elevadas como para que los actuadores en el sistema real puedan efectuarlas. Por lo tanto, para considerar este efecto se introduce la saturación de la variable manipulada Δw en el código de Matlab, de tal modo que, si dicha variable supera el valor de saturación, no lo sobrepasará.

Para ello, en primer lugar, se define la condición de saturación de la variable Δw. Recapitulando la sección anterior, el ciclo de trabajo se fija en una rueda, y en la otra varía desde el valor del ciclo de trabajo fijado en la otra rueda hasta el límite de ser nulo. Sustituyendo en la Ecuación (28) esta condición límite se llega a los valores máximo y mínimo de la variable Δw:

$$\Delta w_{max} = \delta_{fijo}\, w_{max} \tag{32}$$

$$\Delta w_{min} = -\delta_{fijo}\, w_{max} \tag{33}$$

De aquí en adelante, las simulaciones y el diseño del controlador se realizan para un ciclo de trabajo fijo de 0.6. En la Figura 29, se muestra la simulación en Matlab del efecto de la saturación en la respuesta con un controlador demasiado rápido que satura la acción de control al tratar de alcanzar la referencia.

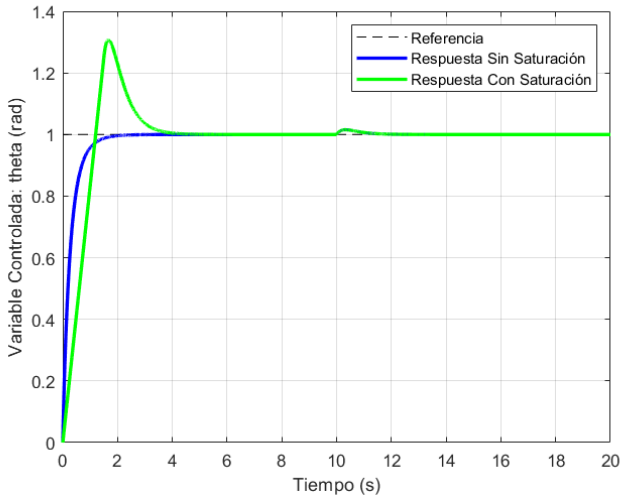

Figura 29.- Evolución temporal de la respuesta con la introducción del efecto de saturación.

En la Figura 29 se aprecia como la saturación provoca una sobreoscilación en el seguimiento de la referencia que antes no se producía. Esto se debe a que el error sigue siendo integrado a pesar de la saturación, causando el crecimiento de la acción integral hasta valores muy elevados. Como consecuencia, el error debe ser del signo contrario a la acción integral durante un tiempo prolongado para que el integrador vuelva a su estado normal. Este fenómeno se conoce como *windup* [7].

La solución consiste en introducir un método *Anti-windup* que consiste en la integración condicional. Este método consiste en desactivar la acción integral cuando el controlador está saturado, y, además, la acción integral provoca que la señal de control se sature incluso más [7]. Por ejemplo, suponiendo que el controlador toma el valor máximo descrito en la Ecuación (32), la acción integral se desactiva si el error es negativo y no se desactiva si es positivo. Introduciendo esta condición en la simulación anterior se obtiene la respuesta mostrada en la Figura 30.

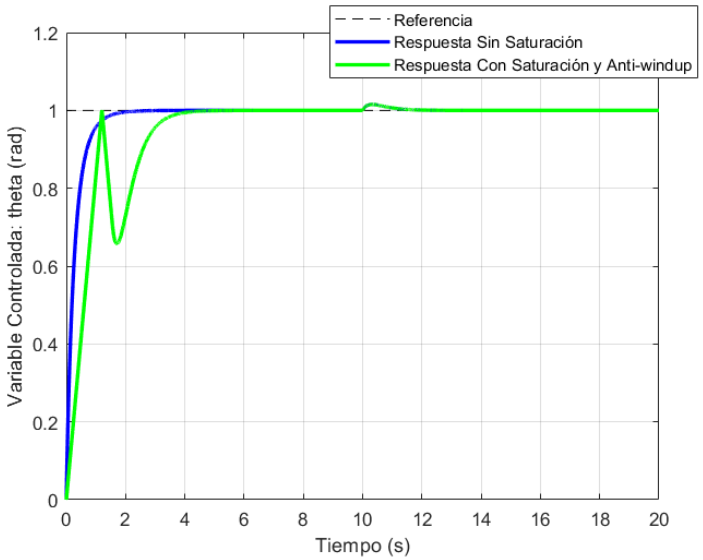

Figura 30.- Evolución temporal de la respuesta con el efecto de saturación y la introducción de método Anti-windup.

Como se observa en la Figura 30, cuando la variable controlada θ alcanza el valor de referencia, la integración se desactiva por la acción del *Anti-windup,* ya que la acción de control está saturada en el límite superior y el error pasa a ser negativo. El problema es que, cuando esto sucede, la acción de control pasa a ser negativa por la acción proporcional calculada en la Ecuación (24). Esto se debe a que la constante b, que es un número menor que uno, multiplica a la referencia. Como solución, cuando se cumpla la condición del *Anti-windup*, además de desactivar la integración, también se desactivará el filtro de ponderación de la referencia, dándole a la constante b un valor de uno. Esta constante tomará su valor original cuando haya un nuevo cambio de la referencia, reactivando el filtro de ponderación de la referencia con los beneficios que conlleva. La Figura 31 muestra la simulación del sistema incluyendo la última modificación explicada.

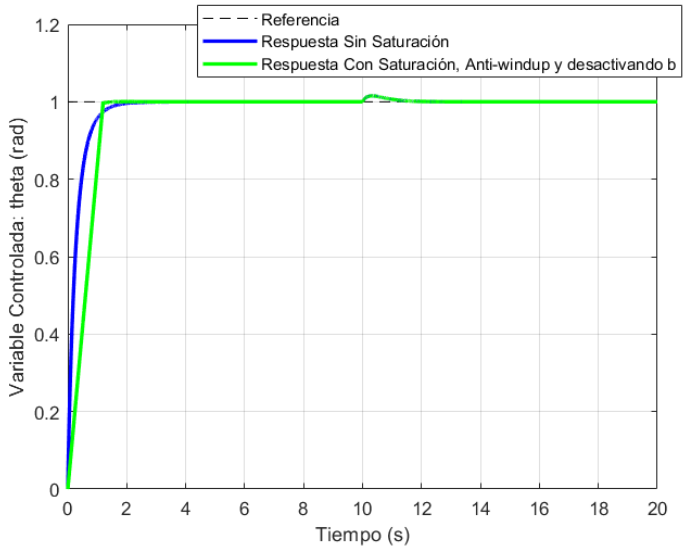

Figura 31.- Evolución temporal de la respuesta con el efecto de saturación y la introducción de nuevo método Anti-windup.

De la Figura 31 se concluye que el comportamiento del sistema mejora considerablemente, aunque no es idéntico al sistema de diseño cuya respuesta se muestra en azul. Con el objetivo de equipararlos se reduce el tiempo de respuesta introducido en la herramienta *pidTuner* para así obtener un controlador más lento que no sature constantemente la acción de control. Para comprobar si el sistema se satura, se introduce una variable booleana que valdrá cero hasta que se alcance la saturación y pase a valer uno. Entonces, se reduce el tiempo de respuesta hasta los 0.75 segundos, cuando se obtiene un controlador que, para el seguimiento de una referencia de treinta grados, que es el máximo aumento de referencia para el campo de visión calculado anteriormente, no satura la acción de control. Las constantes del controlador obtenido tras este proceso se muestran en la Tabla 4.

Constante	K_p	K_i	K_d	T_f	b	c
Valor	16.1092	7.5877	1.1472	0.10283	0.58195	0.011078

Tabla 4.- Valores de las constantes del controlador PID resultante del diseño.

4.3.6. Análisis del controlador PID en el sistema real

El último paso del diseño del controlador PID consiste en analizar y validar su rendimiento en el vehículo real. Con dicho objetivo, se desarrollan los códigos del Anexo III. El código de Arduino implementa la Ecuación (27), correspondiente al PID discretizado, donde introduce la orientación θ, medida por la IMU, y una orientación de referencia θ_{ref} de treinta grados. Por último, ejecuta la acción de control y repite el proceso en bucle. Durante cada iteración de este proceso, además, envía, a través del puerto Bluetooth del ESP32, la información de la orientación θ y el tiempo t transcurrido desde la primera iteración.

Por otro lado, el código de Matlab ejecuta las siguientes instrucciones. En primer lugar, se conecta al puerto mediante el que se establece conexión Bluetooth con el ESP32. A continuación, abre el puerto y lee la información recibida, que consiste en un *string* de los valores θ y t separados por una coma. Separa los valores, los transforma al formato numérico *double* y los almacena en dos vectores. Realiza este proceso hasta que el tiempo t supere los diez segundos y grafica los vectores resultantes junto con los vectores guardados de la simulación en código del apartado anterior. La gráfica resultante se muestra en la Figura 32.

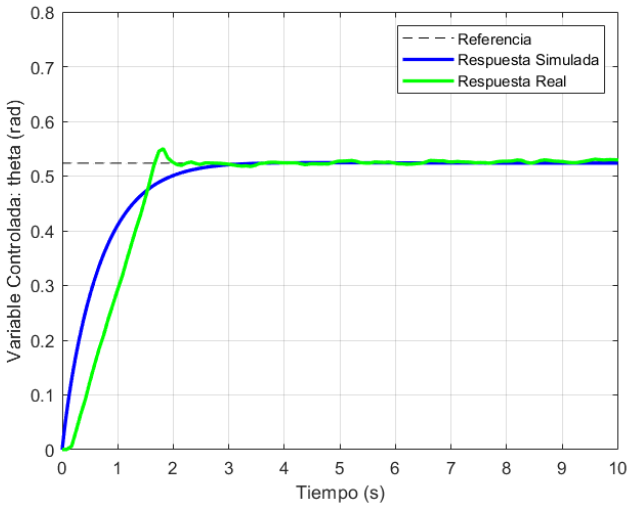

Figura 32.- Comparación de las respuestas real y simulada.

En la Figura 32 se observa cómo, pese a no ser igual a la simulada, la respuesta real cumple el objetivo del seguimiento de la referencia. El principal factor que provoca la discrepancia entre las respuestas real y simulada es la inexactitud en la aplicación de la acción de control. El error se introduce al suponer que la relación entre la velocidad angular de la rueda y el ciclo de trabajo de la señal PWM introducida en el motor se ajusta a la relación lineal de la Ecuación (12). En contraste, empíricamente la rueda no inicia su movimiento hasta introducir un ciclo de trabajo de aproximadamente el 15%, dependiendo este valor de la carga de las baterías y de la velocidad de la otra rueda entre otros factores.

Por otra parte, la rueda en cuyo motor se establece el ciclo de trabajo fijo también varía su velocidad angular dependiendo de la velocidad de la otra rueda. Cuando la otra rueda está parada, esta genera una resistencia al movimiento por rozamiento, por lo que el motor que recibe el ciclo de trabajo fijo consigue suministrar una velocidad inferior que en el caso de que ambas ruedas estén en movimiento.

Estos, junto a otros factores como el peso, la inexactitud del modelo cinemático, el resbalamiento de las ruedas y la imprecisión de los sensores, provocan la discrepancia entre los resultados del controlador ideal y del real. Pese a ello, puesto que el seguimiento de la referencia se efectúa con éxito, se da por bueno el resultado.

4.4. Algoritmo de Navegación

Anteriormente se han desarrollado con detalle el algoritmo de detección del objeto y el módulo del controlador PID, por lo que ya es posible comprender el algoritmo de navegación del vehículo en su totalidad. Para facilitar su explicación, se realiza el diagrama mostrado en la Figura 33.

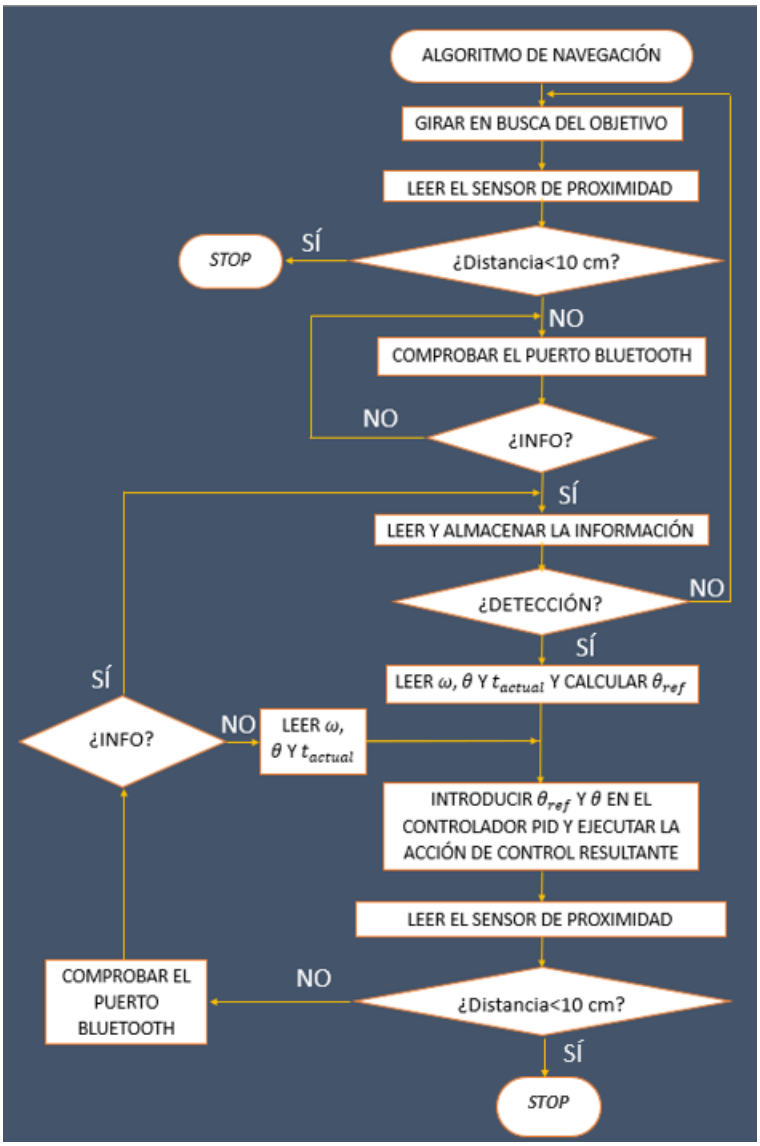

Figura 33.- Algoritmo de navegación.

El algoritmo de navegación, ejecutado en la placa ESP32, se inicia suministrando a los motores ciclos de trabajo arbitrarios y distintos entre sí con el objetivo de que el vehículo rote en busca del objetivo. Después se lee la información del sensor de distancia para no colisionar con el entorno y si la distancia leída es mayor de diez centímetros se continúa, en el caso contrario se detiene el coche.

El siguiente paso consiste en comprobar el puerto Bluetooth de la placa hasta que haya información disponible, proveniente del algoritmo de detección ejecutado simultáneamente en el ordenador. Una vez se reciba la información se lee y se almacena. Si no se produce ninguna detección, se regresa al primer paso. En el caso de haberse producido una detección, la cadena de caracteres recibida contendrá el tiempo de retraso de la detección $t_{retraso}$ y la distancia del objetivo al centro de la imagen d_{obj}. Estos datos junto con los resultados de las lecturas de la orientación θ y la velocidad

angular ω y el tiempo actual se utilizan en la Ecuación (18) para calcular la orientación de referencia θ_{ref}.

La θ_{ref} calculada y la orientación θ se introducen en el controlador PID, dando como resultado una acción de control que se ejecuta a través de aplicar el ciclo de trabajo pertinente al controlador de los motores.

Para finalizar el algoritmo se lee el sensor de proximidad de nuevo para evitar colisionar. Si la distancia es menor que diez centímetros se detiene el vehículo, y si es mayor se comprueba el puerto Bluetooth en busca de nueva información acerca de la detección del objetivo. En caso de no recibirla, se vuelve a leer la orientación actual θ para introducirla junto con la θ_{ref}, calculada anteriormente, en el controlador PID. Además, se leen y almacenan la velocidad angular ω y el tiempo actual, este último necesario para calcular el periodo de muestreo de la velocidad angular, ya que cuando haya una nueva detección se utilizarán para calcular la corrección correspondiente al movimiento y al retraso en la detección. En cambio, si hay información disponible se regresa al punto del bucle donde se leía la información como se puede ver en la Figura 33.

4.5. Interfaz gráfica

Con el fin de facilitar la experiencia del usuario que opere con el robot, se ha creado una interfaz gráfica en Python que posibilita la configuración, la conexión, la visualización de la imagen en tiempo real, y el control del vehículo. La interfaz en cuestión se crea mediante, la librería de Python, *tkinter.* La interfaz se muestra en la Figura 34 y consiste en una raíz que contiene los seis marcos o *frames* que se describirán a continuación y cumplen distintas funciones. Cada uno de estos marcos contiene hasta tres tipos de objetos diferentes: botones, entradas y letreros.

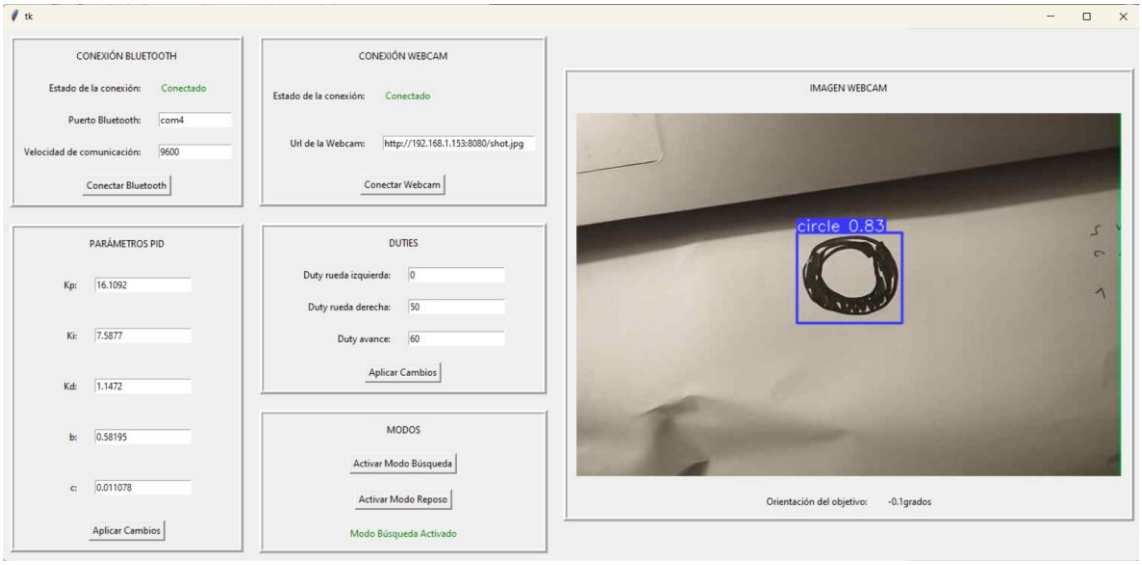

Figura 34.- Interfaz gráfica.

El primer tipo de objeto, los botones, ejecuta una acción asociada al ser pulsado por el usuario. Las entradas sirven para que el usuario introduzca texto en ellas. Cuando se pulsa el botón correspondiente, el código accede a dicho texto, lo guarda en la variable asociada a la entrada y lo emplea en la acción pertinente. Por último, los letreros muestran texto o imágenes, pudiendo variar su contenido cambiando la variable asociada al letrero. Los marcos de la interfaz gráfica creada son los siguientes.

4.5.1. Marcos de conexión

Los marcos de conexión, mostrados en la Figura 35, facilitan la conexión Bluetooth con el ESP32 y la conexión a la dirección del servidor de donde se obtiene la imagen captada en tiempo real por la cámara.

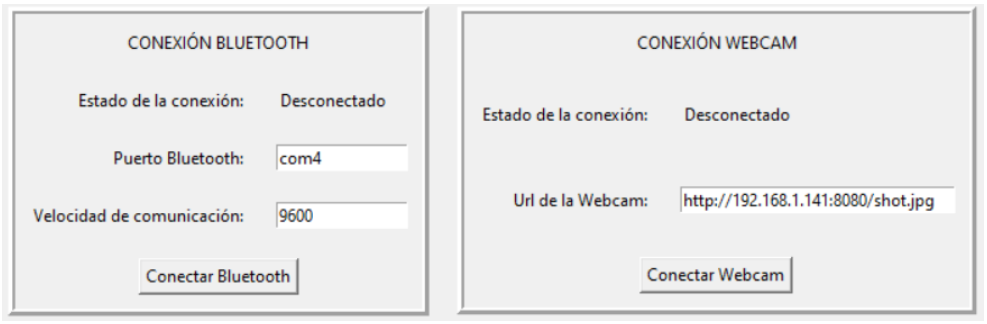

Figura 35.- Marcos de conexión.

En el caso de la conexión bluetooth se debe introducir, en las entradas blancas, el puerto y la velocidad de la comunicación establecida con el ESP32. Por otro lado, para conectarse a la webcam se debe introducir en la entrada pertinente la dirección *url* del servidor. Para intentar las conexiones se debe pulsar en los botones "Conectar Bluetooth" y "Conectar Webcam" respectivamente. Si una conexión se efectúa con éxito, el estado de la conexión cambiará a "Conectado" en verde como se observa en la Figura 34. En cambio, si el intento de conexión fracasa el estado será "Conexión fallida" en rojo.

4.5.2. Marcos de configuración

Los marcos de configuración de la Figura 36 permiten cambiar las constantes del controlador PID y los ciclos de trabajo o *duties* que se envían a los motores cuando el vehículo está girando en busca del objetivo, y el ciclo de trabajo fijo δ_{fijo} o "*Duty* de avance". Las entradas contienen, por defecto, los valores establecidos en el diseño. En el caso de cambiar algún parámetro se debe pulsar el botón "Aplicar Cambios" y los nuevos parámetros se envían por el puerto Bluetooth al ESP32.

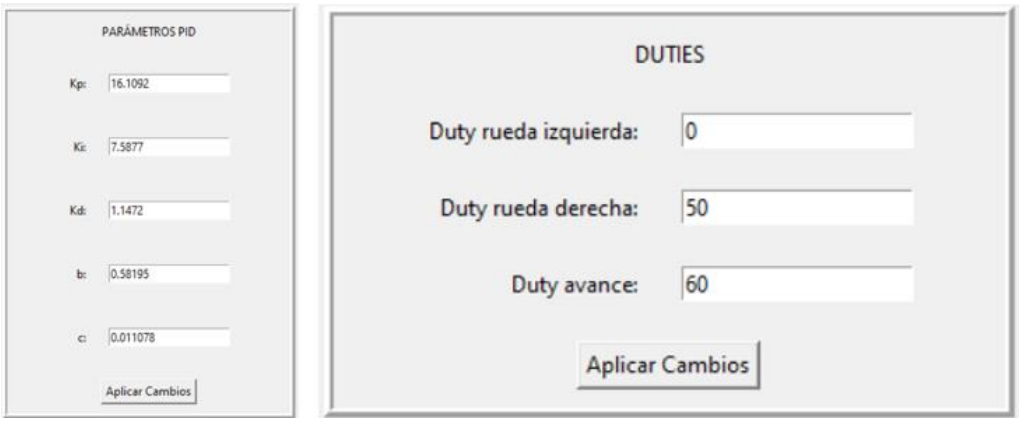

Figura 36.- Marcos de configuración.

4.5.3. Marco de control

El marco de control de la Figura 37 permite seleccionar el modo del vehículo pulsando el botón correspondiente.

El Modo Búsqueda activa el algoritmo de detección del objetivo que se ejecuta en el propio código de Python, y el algoritmo de navegación ejecutado en el código de Arduino, cargado en el ESP32.

El Modo Reposo detiene el vehículo y las detecciones. Para activar alguno de los dos modos deberán haberse efectuado con éxito las conexiones mediante los marcos de conexión.

Figura 37.- Marco de selección de modos.

El ESP32 distinguirá, a partir del primer dígito de la cadena de texto recibida, si la información proveniente del ordenador corresponde a nuevos parámetros del PID, nuevos ciclos de trabajo o al modo seleccionado. Si el primer dígito es un cero, la información recibida corresponde a nuevas constantes del controlador PID. Si es un uno, se trata de nuevos ciclos de trabajo. Si es un dos significa que se ha seleccionado el modo de detección y si es un tres el modo seleccionado es el Modo Reposo. De este modo, el ESP32 es capaz de distinguir la información recibida y de actuar en consecuencia.

Los siguientes dígitos contendrán diferente información en cada caso. La correspondencia entre el dígito y la información contenida para cada caso se representa en la Tabla 5.

DÍGITO	1º	2º	3º	4º	5º	6º
PID	0	K_p	K_p	K_p	b	c
DUTIES	1	δ_L	δ_R	δ_{fijo}		
MODO DETECCIÓN	2	$t_{retraso}$	d_{obj}			
MODO REPOSO	3					

Tabla 5.- Codificación de la información enviada desde Python a la placa ESP32.

4.5.4. Marco de detección del objeto

El último marco, como se muestra en la Figura 38, muestra las detecciones sobre la imagen en tiempo real e indica, en el texto ubicado bajo la imagen, la orientación en grados del objetivo respecto al centro de la imagen θ_{obj}.

Figura 38.- Marco de detección del objeto.

4.6. Prueba de funcionamiento final

La prueba de funcionamiento final se muestra en el video al que da acceso el siguiente enlace:

https://youtu.be/Fvn3CFXy2R8

En el vídeo se muestra el comportamiento del vehículo en diferentes circunstancias, como el seguimiento de un objetivo fijo, de un objetivo móvil, gracias al sistema de navegación que actualiza constantemente la posición del objetivo como referencia, o del funcionamiento en terrenos irregulares, gracias a las ruedas oruga y al controlador PID que corrige constantemente las perturbaciones que introduce el terreno.

Se concluye que el resultado es satisfactorio, ya que se cumple el objetivo del proyecto, y se da por concluido el desarrollo de este.

5. Conclusiones

5.1. Conclusiones del trabajo realizado

El objetivo inicial del proyecto consistía en diseñar e implementar un vehículo de exploración espacial que fuera capaz de alcanzar una referencia visual. Para ello, a lo largo del presente proyecto se ha desarrollado una solución cuya navegación se dirige partir de la placa de desarrollo ESP32 y del ordenador de uso personal, que reciben información y dirigen la acción del resto de componentes que componen el robot.

El seguimiento de la orientación de referencia, cuyo valor se calcula a partir de la detección del objetivo mediante la red neuronal personalizada y de las medidas inerciales, se realiza mediante un controlador PID. Este posibilita, además, el rechazo de las perturbaciones introducidas en el sistema durante su funcionamiento, consiguiendo así que el vehículo sea capaz de funcionar sobre terrenos irregulares como los que se podrían encontrar en la superficie de un planeta rocoso inexplorado. Además, actualiza constantemente la orientación de referencia del objetivo, posibilitando el seguimiento de objetivos móviles.

En el vídeo mostrado en la sección de prueba de funcionamiento final se ha observado como el vehículo resultante del desarrollo de la solución cumple el objetivo inicial del proyecto, aunque queda un amplio margen de mejora en los aspectos que se mencionan a continuación.

5.2. Mejoras futuras

Con el objetivo de mejorar la estimación del estado del vehículo y la percepción del entorno se debe añadir en un futuro las siguientes mejoras:

1) Incorporación de *encoders*: La estimación de la velocidad del vehículo requiere de la introducción de una unidad que proporcione información relativa a la velocidad angular de las ruedas. Un elemento que otorga esta información en motores de corriente continua, como los equipados por en el chasis, es el *encoder*. Fusionando esta información con la proporcionada por el acelerómetro sería posible tener una estimación de la velocidad del vehículo.
2) Incorporación de un lidar: La odometría es la estimación de la posición del vehículo a lo largo de la navegación. Combinando la estimación de la velocidad del vehículo, junto con la información proporcionada por la incorporación de un dispositivo lidar posibilitaría la estimación de la posición del vehículo en el entorno de exploración. Además, el lidar percibe el relieve del entorno y la distancia a posibles obstáculos.

Mediante la incorporación de las mejoras anteriores sería posible desarrollar algoritmos de navegación más seguros, complejos y precisos que el desarrollado en el proyecto, aunque requiere de un mayor presupuesto y dedicación.

6. Pliego de condiciones, presupuesto y relación con los ODS

A lo largo de este último capítulo de la memoria del proyecto realizado, se establece el pliego de condiciones, el presupuesto, y la relación del trabajo con los objetivos de desarrollo sostenible de la Agenda 2030.

6.1. Pliego de Condiciones

Entre la Propiedad, la Universitat Politècnica de València, en adelante "la Propiedad", y el Contratista, Andreu Folch Company, en adelante "el Contratista", se establecen las siguientes cláusulas que constituyen el pliego de condiciones para la ejecución del Trabajo de Fin de Grado acordado.

1) Objeto:

El objeto del presente pliego de condiciones es definir las especificaciones técnicas y los requisitos necesarios para la construcción y desarrollo de un Vehículo Robótico como parte del Trabajo de Fin de Grado del estudiante, Andreu Folch Company. El vehículo de exploración espacial debe ser capaz de alcanzar un objetivo detectado visualmente mediante inteligencia artificial, a partir de la información aportada por los sensores, de medidas inerciales y de proximidad.

2) Condiciones de los materiales:

a) Descripción de los materiales: El Contratista utilizará materiales adecuados y de calidad para la construcción del vehículo. La lista detallada de los materiales y componentes necesarios será aprobada y proporcionada por la Propiedad.

b) Control de calidad de los materiales: El Contratista se compromete a cumplir con los estándares de calidad establecidos por la Propiedad para la selección y uso de materiales en la construcción del vehículo. Será responsabilidad del Contratista garantizar que todos los materiales utilizados cumplan con los criterios de calidad especificados.

3) Condiciones de la ejecución:

a) Descripción de la ejecución: El Contratista llevará a cabo la construcción y desarrollo del vehículo de acuerdo con los diseños y especificaciones técnicas previamente aprobados por la Propiedad. Se espera que el Contratista cumpla con los plazos y etapas establecidos para la ejecución del proyecto.

b) Control de calidad de la ejecución: Durante el proceso de construcción, el Contratista deberá llevar a cabo un riguroso control de calidad para asegurar que el vehículo se ajuste a las especificaciones requeridas y funcione de manera óptima.

4) Pruebas y ajustes finales o de servicio:

Una vez completado el diseño y ensamblaje del robot, el Contratista realizará las pruebas y ajustes necesarios para verificar su correcto funcionamiento y cumplimiento de las especificaciones técnicas establecidas. Estas pruebas deberán ser documentadas y presentadas a la Propiedad para su evaluación.

En caso de que el Vehículo Robótico no cumpla con las especificaciones técnicas acordadas o presente problemas de funcionamiento, el Contratista se compromete a realizar las correcciones y ajustes necesarios hasta alcanzar el nivel de calidad requerido y aprobado por la Propiedad.

En caso de incumplimiento de cualquiera de las cláusulas establecidas en este pliego de condiciones, la Propiedad se reserva el derecho de tomar las medidas que considere oportunas, incluida la rescisión del contrato y la no aprobación del Trabajo de Fin de Grado.

6.2. Presupuesto

En esta sección se presenta el presupuesto de la realización del proyecto, considerando el coste de los materiales, el software utilizado y los recursos humanos.

6.2.1. Coste de los materiales y software utilizados

En la Tabla 6 se muestra el desglose del precio del hardware utilizado y la suma de su coste total. En los casos de los precios del ordenador y del teléfono móvil, se ha introducido su coste de amortización equivalente a un semestre, que se supone del 10% del precio inicial de cada uno. A este presupuesto se le ha añadido también las aproximadamente 30 horas de licencia de Matlab que se han requerido para la realización del proyecto. Nótese que las licencias de los programas más utilizados, Python y Arduino IDE, son gratuitos.

Componente	**Precio [€]**
Placa ESP32	10.99
BNO055	40.54
VL53L0	12.99
Rover 5	10
L298N	3.99
Ordenador	45
Teléfono inteligente	18
15 horas de licencia de Matlab	6
Total	**147.51**

Tabla 6.- Presupuesto hardware y software.

6.2.2. Coste del personal implicado

El coste del personal implicado, que se recoge en la Tabla 7 se compone de las aproximadamente 350 horas de trabajo del alumno, considerando un sueldo de becario en la UPV que es de 13 €/h.

Personal	Tiempo [h]	Salario [€/h]	Precio [€]
Ingeniero Aeroespacial	350	13	4550
Total			**4550**

Tabla 7.- Presupuesto del personal implicado.

6.2.3. Presupuesto total del proyecto

Por último, en la Tabla 8 se realiza la suma del presupuesto total.

Concepto	Precio [€]
Coste del personal	4550
Coste del hardware y software	147.51
Subtotal sin IVA	5147.51
IVA (21%)	1080.98
Total	**5783.49**

Tabla 8.- Presupuesto total del proyecto.

Como se observa en la Tabla 8, el presupuesto total del proyecto realizado se encuentra en los 6228.49 €.

6.3. Relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030

En la Tabla 9 se representa el grado de relación del proyecto realizado con los objetivos de desarrollo sostenible de la agenda 2030.

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 terrestres**				X
ODS 16.	**Paz, justicia e instituciones sólidas**				X
ODS 17.	**Alianzas para lograr objetivos**				X

Tabla 9.- Relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

7. Bibliografía

[1] S. Waslander y J. Kelly, «Introduction to Self-Driving Cars [MOOC],» (s. f.). Coursera. Available: https://www.coursera.org/learn/intro-self-driving-cars/home/info

[2] F. Berzal , Redes Neuronales & Deep Learning, Edición independiente, 2018.

[3] «Codificando Bits,» [En línea]. Available: https://www.codificandobits.com/img/posts/2019-03-30/resultado-convolucion-ultima-iteracion.png

[4] IA-Latam, «IA-Latam,» [En línea]. Available: https://ia-latam.com/wp-content/uploads/2019/02/n8-1.jpg

[5] T. Fawcett, «An Introduction to ROC Analysis,» de *Pattern Recognition Letters*, vol. 27, 2006, pp. 861-874.

[6] M. Kasper-Eulaers, N. Hahn, S. Berger, T. Sebulonsen, Ø. Myrland y P. E. Kummervold, «Short Communication: Detecting Heavy Good Vehicles in Rest Areas in Winter Conditions Using YOLOv5,». Available: https://doi.org/10.3390/a14040114

[7] K. Amström y T. Hägglund, PID Controllers: Theory, Design and Tuning. (2ª ed.), Instrument Society of America, 1995.

[8] Circuit Bread, «Circuit Bread,» [En línea]. Available: https://www.circuitbread.com/tutorials/proportional-integral-and-derivative-control-4-2 . [Último acceso: 2023 Junio 16].

[9] 1994-2023 The MathWorks, Inc., «MathWorks,» [En línea]. Available: https://es.mathworks.com/help/control/ug/two-degree-of-freedom-2-dof-pid-controllers.html. [Último acceso: 26 Junio 2023].

[10] R. Masot, Apuntes de Tecnología Electrónica, 2022.

[11] L. LLamas, «Medir distancia con precisión con Arduino y sensor láser VL53L0X y VL6180X,» *Luis LLamas. Ingeniería, informática y diseño*. Available: https://www.luisllamas.es/arduino-sensor-distancia-vl53l0x/, 9 de enero de 2018.

[12] L. Llamas, «Controlar motores de corriente continua con Arduino y L298N,» *Luis Llamas. Ingeniería, informática y diseño.* Available:

https://www.luisllamas.es/arduino-motor-corriente-continua-l298n/, 26 de mayo 2016.

[13] J. Redmon, «YOLO: Real time object detection,» 2015.

[14] Meta , «Meta AI,» [En línea]. Available: https://ai.meta.com/tools/pytorch/. [Último acceso: 29 Junio 2023].

[15] 2023 Ultralytics Inc., «Comprehensive Guide to Ultralytics YOLOv5,» 29 Marzo 2023. [En línea]. Available: https://docs.ultralytics.com/yolov5/. [Último acceso: 7 Mayo 2023].

[16] Random Nerd Tutorials, «RansomNerdTutorials.com,» [En línea]. Available: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/. [Último acceso: 14 Julio 2023].

[17] F. L. Lewis, «UTA - The University of Texas at Arlington,» [En línea]. Available: https://lewisgroup.uta.edu/ee4314/lectures/Lecture%203.pdf. [Último acceso: 27 Junio 2023].

ANEXOS

Anexo I. Código fuente del vehículo

Anexo I.I. Código de Python

El código de Python ejecutado en el ordenador con el fin de ejecutar la interfaz gráfica y realizar las detecciones de objetos y enviar la información al ESP32 es el siguiente:

```
# Se importan las librerías
import requests
import torch
import cv2
import numpy as np
import imutils
import pandas
import serial
import time
from tkinter import *
from PIL import Image, ImageTk
import math
“””
La información enviada por puerto serial: el primer dígito indica si es información
de: 0-parámetros PID, 1-duties, 2-detección, 3-modo reposo

Parámetros pid: 0-0, 1-Kp, 2-Ki, 3-Kd, 4-b, 5-c
Parámetros duties: 0-1, 1-dutya, 2-dutyb, 3-dutyavance
Datos detección (modo 1): 0-2, 1-delay, 2-posición target
Modo 2: 0-3
“””
model = torch.hub.load('ultralytics/yolov5', 'custom', path =

'C:/Users/andre/OneDrive/Escritorio/4º/TFG/Pruebaredcol/model/bestcircle200.pt')

# Se crea la raíz y los marcos de la interfaz
raiz=Tk()

FrameConex=Frame(raiz, bd=5, relief="ridge")
FrameConex.grid(row=1, column=1, padx=10, pady=10)

FrameWebcam=Frame(raiz, bd=5, relief="ridge")
```

```
FrameWebcam.grid(row=1, column=2, padx=10, pady=10)

FrameParam=Frame(raiz, bd=5, relief="ridge")
FrameParam.grid(row=2, column=1, padx=10, pady=10, rowspan=2)

FrameDuties=Frame(raiz, bd=5, relief="ridge")
FrameDuties.grid(row=2, column=2, padx=10, pady=10)

FrameModos=Frame(raiz, bd=5, relief="ridge")
FrameModos.grid(row=3, column=2, padx=10, pady=10)

FrameImagen=Frame(raiz, bd=5, relief="ridge")
FrameImagen.grid(row=1, column=3, padx=10, pady=10, rowspan=3)
#----------Se definen las variables----------
modo="0"
conex=0
url="0"
# La variable conex es número de conexiones establecidas
puerto=StringVar()
velocidad=StringVar()
estadoBlue=StringVar()
estadoWebcam=StringVar()
urlintro=StringVar()
modotext=StringVar()
kptext=StringVar()
kitext=StringVar()
kdtext=StringVar()
btext=StringVar()
ctext=StringVar()
dutya=StringVar()
dutyb=StringVar()
dutyavance=StringVar()
gradostext=StringVar()

#----------Funciones----------
# Función que se ejecuta al pulsar el botón de conectar Bluetooth
def conectarbluetooth():
    global ad
    global conex
    try:
```

```
        ad=serial.Serial(puerto.get(),velocidad.get())
        estadoBlue.set("Conectado")
        EstadoConex.config(fg="green")

        conex+=1
        if conex==2:
            modotext.set("Seleccione un modo")
    except:

        estadoBlue.set("Conexión fallida")
        EstadoConex.config(fg="red")

# Función que se ejecuta al pulsar el botón de conectar Webcam
def conectarwebcam():
    global conex
    global url
    try:
        url=urlintro.get()
        requests.get(url)
        estadoWebcam.set("Conectado")
        EstadoWebcam.config(fg="green")
        conex+=1
        if conex==2:
            modotext.set("Seleccione un modo")

    except:

        estadoWebcam.set("Conexión fallida")
        EstadoWebcam.config(fg="red")
# Función que se ejecuta al pulsar el botón de aplicar parámetros PID
def aplicarPID():
    global ad

pidstr="0,"+kptext.get()+","+kitext.get()+","+kdtext.get()+","+btext.get()+","+ctext.get(
)+"\n"
    ad.write(pidstr.encode())
# Función que se ejecuta al pulsar el botón de aplicar duties
def aplicarDuties():
    global ad
    dutystr="1,"+dutya.get()+","+dutyb.get()+","+dutyavance.get()+"\n"
```

```
    ad.write(dutystr.encode())

# Función que se ejecuta al pulsar el botón de activar modo
def modoPulsado(num):
    global ad
    global conex
    global modo
    global url
    global model

    modo=num
    if conex==2:

        if modo=="1":
            modotext.set("Modo Búsqueda Activado")
            modoLabel.config(fg="green")

        while modo=="1":
            img_resp = requests.get(url)
            tini = time.time()
            # Procesar la imagen
            img_arr = np.array(bytearray(img_resp.content), dtype=np.uint8)
            img = cv2.imdecode(img_arr, -1)
            img = imutils.resize(img, width=700, height=1260)


            # Realizar las detecciones
            detect = model(img)
            info=detect.pandas().xyxy[0]

            tf=time.time()
            dt= str(tf-tini)


            if info.empty:
                print('DataFrame is empty!') [5]
                dt= "2,"+dt[0:6] + ",0\n"
                ad.write(dt.encode())
```

```
            else:

                xmaxima=info.at[0, 'xmax']
                xminima=info.at[0, 'xmin']
                xmedia=-((xmaxima+xminima)/2-350)
                xmedstr= str(xmedia)
                detectstr= "2,"+ dt[0:6] + "," + xmedstr [0:6] + "\n"
                ad.write(detectstr.encode())

                gradosstr=str(math.atan(xmedia/(594.6237))*180/math.pi)
                gradostext.set(gradosstr[0:4]+ "grados")


            # Mostrar FPS
            frame=cv2.cvtColor(np.squeeze(detect.render()),cv2.COLOR_BGR2RGB)
            imagenpil=Image.fromarray(frame)
            photo=ImageTk.PhotoImage(imagenpil)
            ImgLabel.config(image=photo)
            ImgLabel.image=photo

            raiz.update_idletasks()
            raiz.update()

        if modo=="2":

            modotext.set("Modo Reposo Activado")
            modoLabel.config(fg="green")
            serialstr="3\n"
            ad.write(serialstr.encode())

    else:
        modoLabel.config(fg="red")
#-----------------Frame Conexión Bluetooth----------------------


TituloConex=Label(FrameConex,text="CONEXIÓN BLUETOOTH")
TituloConex.grid(row=1, column=1, padx=10, pady=10, columnspan=2)

TituloConexBlue=Label(FrameConex,text="Estado de la conexión:")
TituloConexBlue.grid(row=2, column=1, sticky="e", padx=10, pady=10)
```

```
EstadoConex=Label(FrameConex,textvariable=estadoBlue)
EstadoConex.grid(row=2, column=2, sticky="w", padx=10, pady=10)
estadoBlue.set("Desconectado")

LablelPuerto=Label(FrameConex,text="Puerto Bluetooth:")
LablelPuerto.grid(row=3, column=1, sticky="e", padx=10, pady=10)

EntryPuerto=Entry(FrameConex, textvariable=puerto,width=15)
EntryPuerto.grid(row=3, column=2, sticky="w", padx=10, pady=10)
puerto.set('com4')

LablelVelocidad=Label(FrameConex,text="Velocidad de comunicación:")
LablelVelocidad.grid(row=4, column=1, sticky="e", padx=10, pady=10)

EntryVelocidad=Entry(FrameConex, textvariable=velocidad,width=15)
EntryVelocidad.grid(row=4, column=2, sticky="w", padx=10, pady=10)
velocidad.set(9600)

BotonConexBlue=Button(FrameConex, text="Conectar Bluetooth",
command=conectarbluetooth)
BotonConexBlue.grid(row=5, column=1, padx=10, pady=10, columnspan=2)

#-----------------Frame Conexión Web Cam----------------------
TituloWebcam=Label(FrameWebcam,text="CONEXIÓN WEBCAM")
TituloWebcam.grid(row=1, column=1, padx=10, pady=10, columnspan=2)

TituloConexWebcam=Label(FrameWebcam,text="Estado de la conexión:")
TituloConexWebcam.grid(row=2, column=1, sticky="e", padx=10, pady=20)

EstadoWebcam=Label(FrameWebcam,textvariable=estadoWebcam)
EstadoWebcam.grid(row=2, column=2, sticky="w”, padx=10,pady=20)
estadoWebcam.set("Desconectado")

Lablelurl=Label(FrameWebcam,text="Url de la Webcam:")
Lablelurl.grid(row=3, column=1, sticky="e", padx=10, pady=20)

Entryurl=Entry(FrameWebcam, textvariable=urlintro, width=32)
Entryurl.grid(row=3, column=2, sticky="w", padx=10, pady=20)
urlintro.set("http://192.168.1.141:8080/shot.jpg")
```

```
BotonConexWebcam=Button(FrameWebcam, text="Conectar Webcam",
command=conectarwebcam)
BotonConexWebcam.grid(row=4, column=1, padx=10, pady=10, columnspan=2)


#-----------------Frame Parámetros PID----------------------------

TituloParam=Label(FrameParam,text="PARÁMETROS PID")
TituloParam.grid(row=1, column=1, padx=94, pady=10, columnspan=2)

kpLabel=Label(FrameParam,text="Kp:")
kpLabel.grid(row=2, column=1, sticky="e”, padx=10, pady=22)

kpEntry=Entry(FrameParam,textvariable=kptext)
kpEntry.grid(row=2, column=2, sticky="w”, padx=10, pady=22)
kptext.set("16.1092")

kiLabel=Label(FrameParam,text="Ki:")
kiLabel.grid(row=3, column=1, sticky="e”, padx=10, pady=22)

kiEntry=Entry(FrameParam,textvariable=kitext)
kiEntry.grid(row=3, column=2, sticky="w”, padx=10, pady=22)
kitext.set("7.5877")

kdLabel=Label(FrameParam,text="Kd:")
kdLabel.grid(row=4, column=1, sticky="e”, padx=10, pady=22)

kdEntry=Entry(FrameParam,textvariable=kdtext)
kdEntry.grid(row=4, column=2, sticky="w”, padx=10, pady=22)
kdtext.set("1.1472")

bLabel=Label(FrameParam,text="b:")
bLabel.grid(row=5, column=1, sticky="e", padx=10, pady=22)

bEntry=Entry(FrameParam,textvariable=btext)
bEntry.grid(row=5, column=2, sticky="w”, padx=10, pady=22)
btext.set("0.58195")
```

```
cLabel=Label(FrameParam,text="c:")
cLabel.grid(row=6, column=1, sticky="e”, padx=10, pady=22)

cEntry=Entry(FrameParam,textvariable=ctext)
cEntry.grid(row=6, column=2, sticky="w”, padx=10, pady=22)
ctext.set("0.011078")

BotonPID=Button(FrameParam, text="Aplicar Cambios", command=aplicarPID)
BotonPID.grid(row=7, column=1, padx=10, pady=10, columnspan=2)

#----------------Frame Duties---------------------------

TituloDuties=Label(FrameDuties,text="DUTIES")
TituloDuties.grid(row=1, column=1, padx=158, pady=10, columnspan=2)

dutyaLabel=Label(FrameDuties,text="Duty rueda izquierda:")
dutyaLabel.grid(row=2, column=1, sticky="e", padx=10, pady=10)

dutyaEntry=Entry(FrameDuties,textvariable=dutya)
dutyaEntry.grid(row=2, column=2, sticky="w", padx=10, pady=10)
dutya.set("0")

dutybLabel=Label(FrameDuties,text="Duty rueda derecha:")

dutybLabel.grid(row=3, column=1, sticky="e", padx=10, pady=10)

dutybEntry=Entry(FrameDuties,textvariable=dutyb)
dutybEntry.grid(row=3, column=2, sticky="w", padx=10, pady=10)
dutyb.set("50")

dutyavLabel=Label(FrameDuties,text="Duty avance:")
dutyavLabel.grid(row=4, column=1, sticky="e", padx=10, pady=10)

dutyavEntry=Entry(FrameDuties,textvariable=dutyavance)
dutyavEntry.grid(row=4, column=2, sticky="w", padx=10, pady=10)
dutyavance.set("60")

BotonDuties=Button(FrameDuties, text="Aplicar Cambios", command=aplicarDuties)
BotonDuties.grid(row=5, column=1, padx=10, pady=10, columnspan=2)
```

```
#-----------------Frame Modos-----------------------------------
TituloModos=Label(FrameModos,text="MODOS")
TituloModos.grid(row=1, column=1, padx=156, pady=10)

botonm1=Button(FrameModos,text="Activar Modo Búsqueda",
command=lambda:modoPulsado("1"))
botonm1.grid(row=2, column=1, padx=10, pady=10)

botonm2=Button(FrameModos,text="Activar Modo Reposo",
command=lambda:modoPulsado("2"))
botonm2.grid(row=3, column=1, padx=10, pady=10)

modoLabel=Label(FrameModos, textvariable=modotext)
modoLabel.grid(row=4, column=1, padx=10, pady=10)
modotext.set("Conecte el Bluetooth y la Webcam")

#--------------Frame Imagen----------------------------------------

TituloImagen=Label(FrameImagen, text="IMAGEN WEBCAM")
TituloImagen.grid(row=1, column=1, padx=156, pady=10, columnspan=2)

ImgLabel=Label(FrameImagen)
ImgLabel.grid(row=2, column=1, padx=10, pady=10, columnspan=2)

GradosLabelI=Label(FrameImagen, text="Orientación del objetivo:")
GradosLabelI.grid(row=3, column=1, padx=10, pady=10, sticky="e")

GradosLabelD=Label(FrameImagen, textvariable=gradostext)
GradosLabelD.grid(row=3, column=2, padx=10, pady=10, sticky="w")
gradostext.set("-")

raiz.mainloop()
```

Anexo I.II. Código de Arduino

El código desarrollado en Arduino IDE cargado en el ESP32 para el funcionamiento del vehículo a partir de la información recibida del ordenador, que ejecuta el código anterior, es el siguiente:

```
//Bluetooth
#include "BluetoothSerial.h"
```

```
BluetoothSerial SerialBT;

//LIBRERIAS
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <math.h>
#include "Adafruit_VL53L0X.h"

//->MOTORES
//MOTOR A O IZQUIERDA
int motorAPin1 = 14;
int motorAPin2 = 27;
int enableAPin = 12;

//MOTOR B O DERECHA
int motorBPin3 = 26;
int motorBPin4 = 25;
int enableBPin = 33;

// Setting PWM properties
#define FREQ 1000
#define CHANNELA 0
#define CHANNELB 1
#define RES 15

double DUTYfijo=60, DUTYArot=0, DUTYBrot=50;
double DUTYA, TonA;
double DUTYB, TonB;
double thetarefabs, thetarefPrev, thetaPrev=M_PI, yawi=M_PI, thetaref;

//BNO055
#define BNO055_SAMPLERATE_DELAY_MS (100)
Adafruit_BNO055 myIMU = Adafruit_BNO055(0x28);

//VL53L0x
Adafruit_VL53L0X lox = Adafruit_VL53L0X();

//Prestaciones motor
```

```
double wmax=8.8; //rad/s

//Constantes del controlador PIDF2
double Kp=16.1092, Ki=7.5877, Kd=1.1472, b=0.58195, c=0.011078,Tf=0.10283;;
double br=0.58195;
//Variables controlador
double currentTime, previousTime;
double elapsedTime;
double error;
double ui, uiPrev=0, up, ud,udPrev;
double deltaomegamin, deltaomegamax, output;


int it=0, sat;

//variables para extraer los datos del str estado
char* token;
double miArray[6]; //python envía hasta 6 datos
int cont;
const char *delimiter =",";

//vector que almacena las últimas velocidades angulares
int sizeVectorTHD=7;
double THDprevias[7];
double THDtotal[7];
int i=0;
double prevtimeTHD;
double timeTHD;
double TTHD;
int indicesTHD;
double sumaTHD;
double promedioTHD;


void setup() {

  Serial.begin(9600); // Se inicializa el canal serie
  delay(1000);

  SerialBT.begin("ESP32BTnuevo");
```

```
  delay(1000);

  // sets the pins as outputs:
  pinMode(motorAPin1, OUTPUT);
  pinMode(motorAPin2, OUTPUT);
  pinMode(enableAPin, OUTPUT);
  pinMode(motorBPin3, OUTPUT);
  pinMode(motorBPin4, OUTPUT);
  pinMode(enableBPin, OUTPUT);

  // configure PWM functionalitites
  ledcSetup(CHANNELA, FREQ, RES);
  ledcSetup(CHANNELB, FREQ, RES);
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enableAPin, CHANNELA);
  ledcAttachPin(enableBPin, CHANNELB);

  //SETUP BNO055
  myIMU.begin();
  delay(1000);
  int8_t temp=myIMU.getTemp();
  myIMU.setExtCrystalUse(true);

  //SETUP VL53L0
  lox.begin(0x29);
  delay(1000);


  //DUTIES
  DUTYA=DUTYfijo;
  TonA=DUTYA*32767/100;
  DUTYB=DUTYfijo;
  TonB=DUTYB*32767/100;
  ledcWrite(CHANNELA,TonA);
  ledcWrite(CHANNELB,TonB);
}

void loop() {
/*
  uint8_t system, gyro, accel, mg = 0;
```

```
 myIMU.getCalibration(&system, &gyro, &accel, &mg);
 SerialBT.print(system);
 SerialBT.print(",");
 SerialBT.print(gyro);
 SerialBT.print(",");
 SerialBT.print(accel);
 SerialBT.print(",");
 SerialBT.println(mg);
 delay(100);
*/

 if (SerialBT.available()>0)
 {
  String estado = SerialBT.readStringUntil('\n');
  // pasamos estado de string a char
  char estadochar[estado.length()+1];
  estado.toCharArray(estadochar, estado.length()+1);

  token=strtok(estadochar, delimiter);
  cont=0;
  while (token != NULL){
   miArray[cont]=atof(token);
   token=strtok(NULL, delimiter);
   cont=cont+1;
  }

  if(miArray[0]==0){

   Kp=miArray[1];
   Ki=miArray[2];
   Kd=miArray[3];
   b=miArray[4];
   c=miArray[5];
  }

  if(miArray[0]==1){

   DUTYArot=miArray[1];
   DUTYBrot=miArray[2];
   DUTYfijo=miArray[3];
```

```
}

if(miArray[0]==2){

 if(miArray[2] == 0)
 {
  Serial.println("Sin detección");
  DUTYB=DUTYBrot;
  TonB=DUTYB*32767/100;
  ledcWrite(CHANNELB,TonB);

  DUTYA=DUTYArot;
  TonA=DUTYA*32767/100;
  ledcWrite(CHANNELA,TonA);

  digitalWrite(motorAPin1, HIGH);
  digitalWrite(motorAPin2, LOW);
  digitalWrite(motorBPin3, HIGH);
  digitalWrite(motorBPin4, LOW);
  delay(100);

  double dist = readDist();
  if(dist!=0 && dist<=200){
   digitalWrite(motorBPin3, LOW);
   while(1==1){
   }
  }

 }
 else
 {
  imu::Vector<3> gyr =myIMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  digitalWrite(motorAPin1, LOW);
  digitalWrite(motorAPin2, LOW);
  digitalWrite(motorBPin3, LOW);
  digitalWrite(motorBPin4, LOW);

  previousTime=millis()/1000.0;
  prevtimeTHD=previousTime;
```

```
double xmed=miArray[2];
double yaw=readYaw();
thetaref=yaw+atan(xmed/594.6237)-yawi-gyr.z()*miArray[1]*M_PI/180;

while (miArray[0]==2){

 double yaw=readYaw();
 double deltayaw=yaw-yawi;

 double deltaomega=computePID(deltayaw);

 if (deltaomega>0){
  DUTYB=DUTYfijo;
  TonB=DUTYB*32767/100;
  DUTYA=(DUTYB*wmax/100-deltaomega)*100/wmax;
  TonA=DUTYA*32767/100;
  ledcWrite(CHANNELA,TonA);
  ledcWrite(CHANNELB,TonB);
 }
 if (deltaomega<=0){
  DUTYA=DUTYfijo;
  TonA=DUTYA*32767/100;
  DUTYB=(DUTYA*wmax/100+deltaomega)*100/wmax;
  TonB=DUTYB*32767/100;
  ledcWrite(CHANNELA,TonA);
  ledcWrite(CHANNELB,TonB);
 }
 digitalWrite(motorAPin1, HIGH);
 digitalWrite(motorAPin2, LOW);
 digitalWrite(motorBPin3, HIGH);
 digitalWrite(motorBPin4, LOW);

 delay(100);
 Serial.print("referencia en>: ");
 Serial.println(atan(xmed/594.6237));

 double dist = readDist();
 if (dist!=0 && dist<100){
  digitalWrite(motorAPin1, LOW);
```

```
 digitalWrite(motorBPin3, LOW);
 while(1==1){
 }
}

imu::Vector<3> gyr =myIMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
THDtotal[0]=gyr.z();
for (i=0; i<sizeVectorTHD;i++){
 THDtotal[1+i]=THDprevias[i];
}

for (i=0; i<sizeVectorTHD;i++){
 THDprevias[i]=THDtotal[i];
}
timeTHD=millis()/1000.0;
TTHD=timeTHD-prevtimeTHD;
prevtimeTHD=timeTHD;


if (SerialBT.available()>0){
 estado = SerialBT.readStringUntil('\n');
 char estadochar[estado.length()+1];
 estado.toCharArray(estadochar, estado.length()+1);

 token=strtok(estadochar, delimiter);
 cont=0;
 while (token != NULL){
  miArray[cont]=atof(token);
  token=strtok(NULL, delimiter);
  cont=cont+1;
 }
 if (miArray[2]==0){
  it=it+1;
  if (it>5) {
   it=0;
   break;
  }
 }else{
  b=br;
  xmed= miArray[2];
```

```
          it=0;
          indicesTHD=int(miArray[1]/TTHD);
          Serial.println(indicesTHD);
          for(i=0;i<indicesTHD;i++){
            sumaTHD=sumaTHD+THDtotal[i];
          }
          promedioTHD=sumaTHD/indicesTHD;
          SerialBT.println(promedioTHD);
          SerialBT.println(miArray[1]);
          SerialBT.println(promedioTHD*miArray[1]);
          sumaTHD=0;
          thetaref=yaw+atan(xmed/594.6237)-promedioTHD*M_PI*miArray[0]/180-
yawi;
        }
       }
      }
     }
   }
   if(miArray[0]==3){
    digitalWrite(motorAPin1, LOW);
    digitalWrite(motorAPin2, LOW);
    digitalWrite(motorBPin3, LOW);
    digitalWrite(motorBPin4, LOW);
   }
  }
}

double computePID(double inp){
  currentTime = millis()/1000.000;
  elapsedTime = currentTime- previousTime;

  error=thetaref-inp;

  up=Kp*b*thetaref-Kp*inp;
  ui=uiPrev+Ki*elapsedTime*error;
  ud=Tf*udPrev/(Tf+elapsedTime)+Kd*(c*(thetaref-thetarefPrev)-
inp+thetaPrev)/(Tf+elapsedTime);

  output=up+ui+ud;
  //Introducimos la saturación debida a la limitación de los motores
```

```
  deltaomegamin=-DUTYfijo*wmax/100; //dutyb 0
  deltaomegamax=DUTYfijo*wmax/100; //dutya 0
  sat=0;
  if(output>=deltaomegamax){
    output=deltaomegamax;
    sat=1;
  }
  if(output<=deltaomegamin){
    output=deltaomegamin;
    sat=1;
  }
  //Introducimos el CLAMPING ANTIWINDUP
  if (sat==1&&(((error>0)&&(output<0))||((error<0)&&(output>0)))){
    ui=0;
    b=1;
    SerialBT.println("ui anulado");
  }

  previousTime=currentTime;
  uiPrev=ui;
  thetarefPrev=thetaref;
  thetaPrev=inp;
  udPrev=ud;

  return output;

}
double readYaw(){

  imu::Quaternion quat=myIMU.getQuat();
  double outputyaw=atan2(2*(quat.w()*quat.z()+quat.x()*quat.y()),1-
2*(quat.y()*quat.y()+quat.z()*quat.z()))+M_PI;
  return outputyaw;

}

double readDist(){
  double outdist;
  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false);
```

```
 if (measure.RangeStatus != 4){

  if (measure.RangeMilliMeter == 8191){
   outdist=0;
  }else{
   outdist=measure.RangeMilliMeter;
  }

 }
 else{
  Serial.println("  Fuera de rango ");
  outdist=0;
 }
 return outdist;
}
```

Anexo II. Código para el diseño e implementación en código del controlador PID

Para realizar el diseño del controlador PID en Matlab en primer lugar se introduce el siguiente diagrama de bloques en simulink.

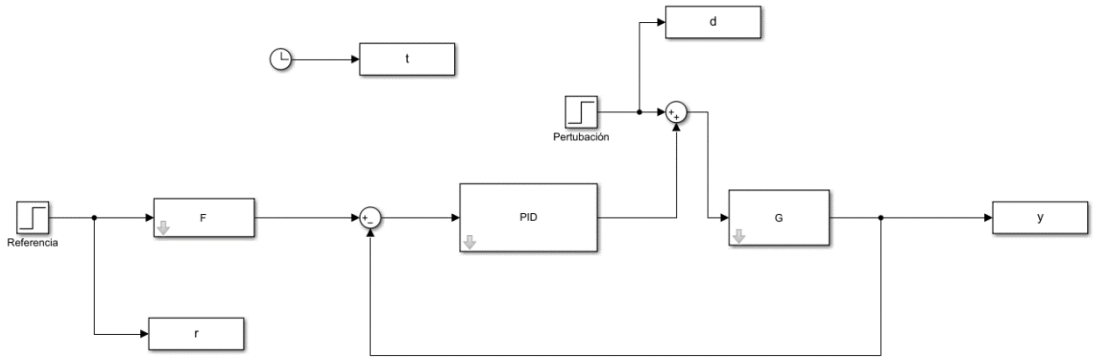

Figura 39.- Diagrama de bloques implementado en simulink.

Una vez se ha completado el primer paso, ya es posible ejecutar el siguiente código, que simula la respuesta del controlador PID diseñado mediante la función pidTuner:

```
s=tf('s');
tiempo=10;
timeD=0;
timeR=0;
T=0.01;
```

```
R=30*pi/180;
disturbance=1;
G=0.15789/s;
%rltool(G);
%pidTuner(G,'pidf2');
%pause


[Cfr,Xfr]= getComponents(C,'filter');
PID=Cfr;
F=Xfr;
sim('simula_pl3_prefiltro')
kp=16.1092;
ki=7.5877;
kd=1.1472;
ti=kp/ki;
td=kd/kp;
b=0.58195;
c=0.011078;
Tf=0.10283;
N=td/Tf;

wmax=8.79;
dutyfijo=0.6;
satur=0;
saturever=0;
%Implementación código
z=tf('z',T);
gpz=c2d(G,T,'zoh');
[Az,Bz,Cz,Dz]=ssdata(gpz);

n=length(Az);
ek=0;
yk=0;
ys=[];
us=[];
uk=0;
rk=R;

x=zeros(n,length(t));
```

```
ui_1=0;
ud_1=0;
rk_1=0;
yk_1=0;
sumdif=0;
diferencia=[];
ys=zeros(1,length(t));
temps=0;

for i=1:length(t)

   %Medida de la variable controlada
   ys(i)= Cz*x(:,i)+Dz*uk+Dz*d(i);
   yk = ys(i);

   diferencia(i)=abs(ys(i)-y(i));
   sumdif=sumdif+diferencia(i);
   ek=rk-yk;

   up=kp*b*rk-kp*yk;
   ui=ui_1+(kp*T/ti)*ek;
   %ud=(kp*td/T)*c*(rk-rk_1)-(kp*td/T)*(yk-yk_1);
   %ud=td*ud_1/(td+N*T)-kd*N*(yk-yk_1)/(td+N*T);
   ud=Tf*ud_1/(Tf+T)+kd*(c*(rk-rk_1)-yk+yk_1)/(Tf+T);
   u=up+ui+ud;
   satur=0;


   if u>=wmax*dutyfijo
      u=wmax*dutyfijo;
      satur=1;
      saturever=1;

   end

   if u<=-wmax*dutyfijo

      u=-wmax*dutyfijo;
```

```
      satur=1;
      saturever=1;

   end
   if satur==1&&(((ek<0)&&(u>0))||((ek>0)&&(u<0)))
      ui=0;
      b=1;
   end

   ui_1=ui;
   ud_1=ud;
   rk_1=rk;
   yk_1=yk;

   uk=u;
   x(:,i+1) = Az*x(:,i)+Bz*uk+Bz*d(i);
   temps=temps+T;
   if temps>(timeR-T)
      rk=R;
   end


end

promdif=sumdif/length(t)

figure
plot(t,r,'k--','LineWidth',0.5)
hold on
plot(t,y,'b','LineWidth',2)
plot(t,ys,'g','linewidth',2)
plot(tiemporeal,yawreal,'g','linewidth',2)

xlabel('Tiempo (s)')
axis([0 tiempo 0 0.8])
grid
%'Simulación Simulink con filtro derivativo'
legend('Referencia','Respuesta Simulink','Respuesta Simulación código')
```

Anexo III. Código para realizar las mediciones del PID real

Para realizar las mediciones de la respuesta del PID en el vehículo real se carga el siguiente código en la placa ESP32:

```
//Bluetooth
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;

//LIBRERIAS
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <math.h>

//->MOTORES
//MOTOR A O IZQUIERDA
int motorAPin1 = 14;
int motorAPin2 = 27;
int enableAPin = 12;

//MOTOR B O DERECHA
int motorBPin3 = 26;
int motorBPin4 = 25;
int enableBPin = 33;

// Setting PWM properties
#define FREQ 1000
#define CHANNELA 0
#define CHANNELB 1
#define RES 15

//Definimos el DTUTY de la rueda fija
double DUTYfijo=60;
double DUTYA;
double TonA;
double DUTYB;
double TonB;
double thetarefabs=210*M_PI/180, thetarefPrev, thetaPrev=M_PI, yawi=M_PI,
thetaref,udPrev;
```

```
//BNO055
#define BNO055_SAMPLERATE_DELAY_MS (100)
Adafruit_BNO055 myIMU = Adafruit_BNO055();

//Prestaciones motor
double wmax=8.8; //rad/s

//Constantes del controlador PIDF2
double Kp=16.1092, Ki=7.5877, Kd=1.1472, b=0.58195, c=0.011078, Tf=0.10283;
double br=b;
//Variables prefiltro

//Variables controlador
double currentTime, previousTime=0;
double elapsedTime;
double error;
double ui, uiPrev=0, up, ud;
double deltaomegamin, deltaomegamax, output, sat;
int it=0;
double timei;

void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600); // Se inicializa el canal serie
  delay(1000);
  SerialBT.begin("ESP32BTnUEVO");
  delay(1000);
  // sets the pins as outputs:
  pinMode(motorAPin1, OUTPUT);
  pinMode(motorAPin2, OUTPUT);
  pinMode(enableAPin, OUTPUT);
  pinMode(motorBPin3, OUTPUT);
  pinMode(motorBPin4, OUTPUT);
  pinMode(enableBPin, OUTPUT);

  // configure LED PWM functionalitites
  ledcSetup(CHANNELA, FREQ, RES);
  ledcSetup(CHANNELB, FREQ, RES);
```

```
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enableAPin, CHANNELA);
  ledcAttachPin(enableBPin, CHANNELB);

  //SETUP BNO055
  myIMU.begin();
  delay(1000);
  int8_t temp=myIMU.getTemp();
  myIMU.setExtCrystalUse(true);

  //DUTIES
  DUTYA=DUTYfijo;
  TonA=DUTYA*32767/100;
  DUTYB=DUTYfijo;
  TonB=DUTYB*32767/100;
  ledcWrite(CHANNELA,TonA);
  ledcWrite(CHANNELB,TonB);
  delay(10000);
  timei = millis()/1000.000;
  digitalWrite(motorAPin1, HIGH);
  digitalWrite(motorAPin2, LOW);
  digitalWrite(motorBPin3, HIGH);
  digitalWrite(motorBPin4, LOW);

}

void loop() {
  // put your main code here, to run repeatedly:
    uint8_t system, gyro, accel, mg = 0;
  myIMU.getCalibration(&system, &gyro, &accel, &mg);

  imu::Quaternion quat=myIMU.getQuat();
  double yaw=atan2(2*(quat.w()*quat.z()+quat.x()*quat.y()),1-
2*(quat.y()*quat.y()+quat.z()*quat.z()))+M_PI;
  double yawg=yaw*180/M_PI;

  thetaref=thetarefabs-M_PI;
  double deltayaw=yaw-yawi;
```

```
  double deltaomega=computePID(deltayaw);
  SerialBT.print(deltayaw,4);
  SerialBT.print(",");
  SerialBT.println(currentTime-timei);

  if (deltaomega>0){
    DUTYB=DUTYfijo;
    TonB=DUTYB*32767/100;
    DUTYA=(DUTYB*wmax/100-deltaomega)*100/wmax;
    TonA=DUTYA*32767/100;
    ledcWrite(CHANNELA,TonA);
    ledcWrite(CHANNELB,TonB);
    Serial.println(DUTYA);
  }
  if (deltaomega<=0){
    DUTYA=DUTYfijo;
    TonA=DUTYA*32767/100;
    DUTYB=(DUTYA*wmax/100+deltaomega)*100/wmax;
    TonB=DUTYB*32767/100;
    ledcWrite(CHANNELA,TonA);
    ledcWrite(CHANNELB,TonB);
    Serial.println(DUTYA);
  }
  digitalWrite(motorAPin1, HIGH);
  digitalWrite(motorBPin3, HIGH);
  delay(80);


}

double computePID(double inp){
  currentTime = millis()/1000.000;
  elapsedTime = currentTime- previousTime;


  error=thetaref-inp;

  up=Kp*b*thetaref-Kp*inp;
  ui=uiPrev+Ki*elapsedTime*error;
  //ud=Kd/elapsedTime*c*(thetaref-thetarefPrev)-Kd/elapsedTime*(inp-thetaPrev);
```

```
  ud=Tf*udPrev/(Tf+elapsedTime)+Kd*(c*(thetaref-thetarefPrev)-
inp+thetaPrev)/(Tf+elapsedTime);

  output=up+ui+ud;
//Introducimos la saturación debida a la limitación de los motores
  deltaomegamin=-DUTYfijo*wmax/100; //dutyb 0
  deltaomegamax=DUTYfijo*wmax/100//dutya 0
  sat=0;
  if(output>=deltaomegamax){
    output=deltaomegamax;
    sat=1;
  }
  if(output<=deltaomegamin){
    output=deltaomegamin;
    sat=1;
  }
  return output;
//Introducimos el CLAMPING ANTIWINDUP
  if (sat==1&&(((error>0)&&(output<0))||((error<0)&&(output>0)))){
    ui=0;
    b=1;
  }
  previousTime=currentTime;
  uiPrev=ui;
  thetarefPrev=thetaref;
  thetaPrev=inp;
  udPrev=ud;
}
```

El anterior código envía, vía Bluetooth al ordenador, información relativa a la evolución temporal de la orientación, y para leerla y graficarla se ejecuta el siguiente código en Matlab:

```
s=serialport("COM4",9600)
tiemporeal=[];
yawreal=[];
tiempor=0;
indice=1;
while tiempor<=10
   fopen(s);
```

```
    str=fscanf(s);
    valores=strsplit(str,',');
    yawreal(indice)=str2double(valores{1});
    tiemporeal(indice)=str2double(valores{2});
    tiempor=tiemporeal(indice)
    indice=indice+1;

end
figure
plot(t,r,'k--','LineWidth',0.5)
hold on
plot(t,ys,'b','linewidth',2)
```