UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema de guiado autónomo de un Rover basado en una matriz de sensores de distancia ToF

Trabajo Fin de Grado

Grado en Ingeniería Aeroespacial

AUTOR/A: Gómez Pol, Víctor

Tutor/a: Alcañiz Fillol, Miguel

Cotutor/a: Masot Peris, Rafael

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y desarrollo de un sistema de guiado autónomo de un Rover basado en una matriz de sensores de distancia ToF

TRABAJO DE FIN DEL
Grado en Ingeniería Aeroespacial

AUTOR
Víctor Gómez Pol

TUTORES
Miguel Alcañiz Fillol
Rafael Masot Peris

CURSO ACADÉMICO: 2022/2023

Resumen

La exploración espacial supone uno de los retos más complicados de la actualidad. Uno de los campos cruciales en este tema es la exploración terrestre de cuerpos celestes tales como planetas, satélites o asteroides. Los vehículos de exploración espacial capaces de recorrer y analizar las superficies extraterrestres disponen de gran cantidad de sistemas muy sofisticados, entre los que se incluye el sistema de guiado, navegación y control (GNC).

Este sistema de navegación, guiado y control de un *rover* se alimenta de datos recopilados por distintos tipos de sensores. Entre ellos, los sensores de percepción captan el espacio tridimensional que rodea el vehículo. Existen varias líneas de investigación para desarrollar técnicas avanzadas que permitan mejorar la precisión y el coste computacional de este tipo de sensores y sus algoritmos de control asociados. Este proyecto pretende estudiar una alternativa a los sensores de percepción utilizados hasta ahora en los vehículos de exploración espacial.

En este Trabajo de Fin de Grado se ha diseñado, fabricado e implementado un sistema de navegación, guiado y control para un vehículo de exploración espacial basado en un sensor *Time of Flight* (ToF) de última generación que genera una matriz de distancias de 64 casillas que representa el espacio delante al *rover*. Este tipo de sensores tienen algunas ventajas frente a los sensores utilizados hasta ahora, siendo especialmente compactos, ligeros y con un consumo de energía muy bajo.

Resum

L'exploració espacial suposa un dels reptes més complicats de l'actualitat. Un dels camps crucials en aquest tema és l'exploració terrestre de cossos celestes com planetes, satèl·lits o asteroides. Els vehicles espacials capaços de recórrer i analitzar les superfícies extraterrestres disposen de gran quantitat de sistemes molt sofisticats, entre els quals s'inclou el sistema de navegació, guiatge i control (GNC).

Aquest sistema de navegació i guiatge d'un rover s'alimenta de dades recopilades per diferents tipus de sensors. Entre ells, els sensors de percepció capturen l'espai tridimensional que envolta el vehicle. Hi ha diverses línies d'investigació per desenvolupar tècniques avançades que permetin millorar la precisió i el cost computacional d'aquest tipus de sensors i els algoritmes de control associats. Aquest projecte pretén estudiar una alternativa als sensors de percepció utilitzats fins ara en els vehicles d'exploració espacial.

En aquest Treball de Fi de Grau s'ha dissenyat, fabricat i implementat un sistema de navegació i guiatge per a un vehicle d'exploració espacial basat en un sensor *Time of Flight* (ToF) d'última generació que genera una matriu de distàncies de 64 caselles que representa l'espai davant del rover. Aquest tipus de sensors tenen algunes avantatges davant dels sensors utilitzats fins ara, essent especialment compactes, lleugers i amb un consum d'energia molt baix.

Abstract

Space exploration is one of the most challenging endeavors of our time. One crucial field in this area is the terrestrial exploration of celestial bodies such as planets, satellites, or asteroids. Extraterrestrial vehicles capable of traversing and analyzing foreign surfaces are equipped with a wide range of highly sophisticated systems, including the guidance, navigation and control system (GNC).

This navigation and guidance system of a rover is powered by data collected from various types of sensors. Among them, perception sensors capture the three-dimensional space surrounding the vehicle. There are several lines of research aimed at developing advanced techniques to improve the precision and computational cost of these types of sensors and their associated control algorithms. This project aims to study an alternative to the perception sensors used in space exploration vehicles thus far.

In this Degree's Thesis, a navigation and guidance system based on a state-of-the-art Time of Flight (ToF) sensor has been designed, developed, and implemented for a space exploration vehicle. The ToF sensor generates a 64-cell distance matrix representing the space in front of the rover. These types of sensors offer several advantages over the sensors used thus far, being particularly compact, lightweight, and energy-efficient.

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

Unidades empleadas

^{o}	Grados
cm	Centímetro
g	Gramo
h	Hora
Hz	Hercio
kB	Kilobyte
kB	Megabyte
mA	Miliamperio
mAh	Miliamperio hora
MHz	Megahercio
mm	Milímetro
ud	Unidad
V	Voltio
€	Euro

Magnitudes físicas

$\Delta\theta$	Ángulo de giro

Siglas

ACK	Acknowledgement
CCW	Counter-Clockwise
CW	Clockwise
DC	Direct Current
DEM	Digital Elevation Model
DSP	Digital Signal Processor
GNC	Guidance, Navigation and Control
GND	Ground
$I2C$	Inter-Integrated Circuit

IMU Inertial Measurement Unit

IVA Impuesto sobre el Valor Añadido

Lidar Light Detection and Ranging

$LIPO$ Lithium Polymer

MER Mars Exploration Rover

PID Proportional, Integral and Differential

PWM Pulse Wide Modulation

RGB Red-Green-Blue

SCL Serial Clock Line

SDA Serial Data Line

SFR Sample Fetch Rover

$SRAM$ Static Random Access Memory

ToF Time of Flight

$VCSEL$ Vertical Cavity Surface Emitting Laser

1. Introducción

Este Trabajo de Fin de Grado tiene el objetivo de desarrollar un sistema de guiado autónomo para un *rover* basado en el uso de un novedoso sensor ToF (*Time of Flight*) que genera un mapeado del espacio a través de una matriz 8x8 de distancias.

El trabajo está estructurado en siete capítulos.

En el capítulo *1. Introducción* se presenta la justificación y relevancia del tema seleccionado para el desarrollo de este proyecto. Se describe el tema y el trabajo desarrollado y se plantean los objetivos del presente proyecto.

En el capítulo *2. Estado del Arte* se realiza un estudio bibliográfico de los sensores de percepción más utilizados en los sistemas de navegación, guiado y control de los vehículos espaciales. Posteriormente, se expone la relevancia de este este proyecto en referencia a las necesidades de mejora que estos sistemas necesitan.

En el capítulo *3. Estudio y selección de los componentes* se presentan las diferentes alternativas que se han tenido en cuenta durante el proceso de selección de los componentes electrónicos que componen el sistema de guiado. En él se justifica la elección de cada componente y se presentan algunas características de los sensores seleccionados.

En el capítulo *4. Implementación hardware* se presenta el proceso seguido para la conexión física de todos los componentes del sistema, desde los elementos físicos como el chasis y los motores hasta los componentes electrónicos como el microcontrolador y los sensores. Se detallan además todas las conexiones realizadas y las piezas auxiliares que han sido diseñadas y fabricadas para el ensamblaje del prototipo.

En el capítulo *5. Implementación software* se detallan los algoritmos utilizados en los distintos módulos *software* que permiten la correcta toma de decisiones de control por parte del sistema de navegación. Se detallan también los protocolos de comunicación que utilizan los componentes electrónicos para la transmisión de datos y el flujo de datos que se produce en cada ejecución del sistema.

En el capítulo *6. Resultados y líneas futuras* se hace un análisis detallado de los resultados obtenidos y se proponen futuras líneas de investigación que pueden mejorar el sistema de navegación desarrollado utilizando el presente trabajo como base teórica y práctica.

Finalmente, en el capítulo *7. Presupuesto* se calcula una estimación de los costes totales de desarrollo del prototipo y del sistema de navegación, haciendo distinción entre los costes materiales y los costes asociados a la mano de obra ingenieril.

1.1. Justificación

La exploración espacial ha sido uno de los retos más fascinantes y desafiantes emprendidos por la humanidad en la reciente historia moderna. A medida que avanza el siglo XXI, la importancia de la exploración espacial sigue creciendo, especialmente en los últimos años en los que se ha reavivado la ilusión y esperanza de la población ligados a la aparición de nuevas empresas y proyectos que pretenden llegar a otros cuerpos celestes nunca alcanzados antes por la humanidad.

La capacidad de enviar misiones a otros planetas y lunas con el objetivo de descubrir y comprender mejor el universo que nos rodea ha impulsado avances significativos en la tecnología y la ciencia. Uno de los elementos clave de estas misiones son los *rovers*, vehículos robóticos diseñados para operar en entornos extraterrestres y recopilar datos científicos valiosos.

A medida que la exploración espacial avanza y las misiones se vuelven más ambiciosas, surge la necesidad de desarrollar sistemas de navegación más sofisticados y autónomos para los *rovers*. En particular, el diseño y desarrollo de un sistema de navegación autónomo capaz de evitar obstáculos en tiempo real es esencial para garantizar la eficiencia y seguridad de estas misiones, permitiendo a estos vehículos espaciales tomar decisiones instantáneas y responder a situaciones inesperadas sin depender completamente de los comandos enviados desde la Tierra, que sufren el retardo de la señal asociado al tiempo de viaje en el espacio.

El objetivo principal de este Trabajo de Fin de Grado es abordar este desafío y proponer el uso de una tecnología desarrollada recientemente que permita gestionar la toma de decisiones en el ámbito local, en el que las maniobras de un vehículo de exploración espacial deben ir enfocadas a evitar los obstáculos cercanos. De esta forma, este proyecto se centrará en la creación de algoritmos y técnicas de control automático que permitan a los *rovers* detectar y sortear obstáculos de manera eficiente y efectiva.

El sistema de navegación propuesto se basará en un sensor de distancias ToF (*Time of Flight*) que realiza un mapeado del espacio y genera una matriz de 64 casillas donde cada valor representa la distancia al objeto presente en dicha zona del espacio.

1.2. Objetivos

El objetivo del presente Trabajo Final de Grado es el diseño e implementación de un sistema de guiado autónomo que permita a un *rover* evitar obstáculos gracias a un sensor ToF que genera una matriz de distancias de 64 celdas. Para ello, es necesario el desarrollo de varios módulos de *software* que sean capaces de interconectar y gestionar los distintos componentes *hardware* tales como sensores y actuadores a través de un microcontrolador.

Los objetivos concretos que se han ido cumpliendo durante le desarrollo del trabajo son los siguientes:

- Selección de los componentes *hardware* y *software* y estudio de su compatibilidad.
- Diseño e impresión 3D de piezas auxiliares necesarias para el correcto ensamblaje del *rover* y los sensores
- Conexión de los componentes electrónicos con los elementos mecánicos del *rover* y el módulo microcontrolador ESP32
- Desarrollo en Matlab de un algoritmo generador de señales PWM (*Pulse Wide Modulation*) para el control de los motores DC basado en la información de la matriz de sensores de distancia ToF.
- Desarrollo de un algoritmo de control tipo PID para realizar de forma precisa las órdenes de control de giro y avance
- Realización de ensayos y pruebas de validación del sistema.

2. Estado del Arte

La exploración espacial ha experimentado un notable avance en las últimas décadas, convirtiéndose en un campo de investigación de gran relevancia y trascendencia para la humanidad. A medida que el ser humano ha ampliado sus fronteras más allá de la Tierra, el deseo de comprender y descubrir los secretos del universo se ha convertido en una búsqueda apasionante y sin precedentes.

En este contexto, los *rovers*, vehículos robóticos especialmente diseñados para operar en entornos extraterrestres, han surgido como herramientas esenciales en las misiones espaciales. Estos ingeniosos dispositivos permiten a los científicos explorar y recopilar datos valiosos sobre planetas, lunas y otros cuerpos celestes de nuestro sistema solar.

En particular, los *rovers* han desempeñado un papel crucial en la exploración espacial al permitirnos estudiar detalladamente la superficie de planetas y satélites como Marte o la Luna. Estos vehículos autónomos cada vez más sofisticados y pesados están equipados con una gran variedad de instrumentos científicos y tecnológicos que les permiten recolectar datos geológicos, atmosféricos y astrobiológicos, entre otros. Numerosos *rovers* forman parte de importantes misiones espaciales como los cinco que exploraron la superficie de Marte: *Sojourner*, *Spirit*, *Opportunity*, *Curiosity* y *Perseverance*.

La importancia de los *rovers* en las misiones espaciales radica en su capacidad para superar las limitaciones humanas y llevar a cabo tareas complejas en entornos hostiles. Estos vehículos son capaces de explorar regiones inaccesibles para los astronautas, operar en condiciones extremas de temperatura y radiación, y enfrentar terrenos difíciles e irregulares. Para ello, disponen de varios sistemas que interactúan con el medio en el que se encuentran y que recopilan información para que complejos algoritmos tomen las decisiones adecuadas que permitan a estos vehículos operar de forma segura y eficiente en estas tierras hostiles.

Es necesario hacer una distinción de los tipos de sistemas y algoritmos que permiten esta correcta operación de los vehículos de exploración espacial. Por un lado, existen los algoritmos de planificación de rutas y, por otro lado, existen los algoritmos de evasión de obstáculos. Ambos sistemas son clave para garantizar el éxito de una misión y la seguridad de estos vehículos. Aunque ambos algoritmos comparten el objetivo general de guiar a los *rovers* a través de un entorno desconocido, existen diferencias significativas en su enfoque y aplicación.

Un algoritmo de planificación de rutas se centra en determinar la trayectoria óptima para alcanzar un destino específico. Este tipo de algoritmo tiene en cuenta diversos factores, como la distancia, la pendiente del terreno, los recursos disponibles y los objetivos científicos de la misión. El objetivo principal es encontrar la ruta más eficiente y segura que permita al *rover* alcanzar su destino deseado y completar sus tareas científicas de manera exitosa.

En contraste, un algoritmo de evasión de obstáculos, en el que se centra el presente trabajo, se enfoca en detectar y evitar obstáculos cercanos en tiempo real. A diferencia del algoritmo de planificación de ruta, que considera un horizonte de planificación más amplio,

el algoritmo de evasión de obstáculos se ocupa de situaciones inmediatas y locales. Su objetivo es permitir al *rover* evitar colisiones y sortear obstáculos imprevistos que puedan surgir en su entorno inmediato. Para lograr esto, un algoritmo de evasión de obstáculos se basa en datos en tiempo real recopilados por los sensores del *rover*. Estos datos permiten al algoritmo detectar la presencia de obstáculos y calcular trayectorias alternativas que eviten colisiones.

Es importante destacar que estos dos tipos de algoritmos no son mutuamente excluyentes y, de hecho, suelen trabajar en conjunto para garantizar la navegación exitosa de un *rover* espacial. El algoritmo de planificación de ruta establece la ruta general que el vehículo debe seguir, teniendo en cuenta los objetivos de la misión y las características del terreno. Mientras tanto, el algoritmo de evasión de obstáculos se activa en situaciones locales y dinámicas, permitiendo que el *rover* tome decisiones en tiempo real para evitar obstáculos inmediatos. Ambos sistemas deben operar de forma conjunta y con el mismo nivel de precisión para poder crear un sistema de navegación preciso y consistente [1].

El algoritmo de evasión de obstáculos se nutre de información recopilada por diferentes sensores que tratan de captar la realidad del entorno de la forma más precisa posible para que el algoritmo de toma de decisiones calcule la maniobra óptima de operación. Este proyecto se centra en el estudio y la propuesta de un nuevo sensor que aporta al sistema información referente al mapeo tridimensional del entorno por el que transita el *rover*. Este tipo de sensores reciben el nombre de sensores de percepción.

La percepción es el término comúnmente utilizado en robótica para referirse a la capacidad de mapeo o, en otras palabras, la modelización del entorno. Los mapas pueden ser de diferentes tipos dependiendo de los datos que codifican, pero en general hacen referencia a una representación en 3D del terreno, típicamente una nube de puntos o un modelo digital del terreno (DEM) una vez que la nube de puntos está estructurada. El modelo del terreno luego se procesa aún más en la cadena de percepción para detectar obstáculos o peligros y extraer de ellos la información de transitabilidad del terreno [2].

Existe gran variedad de sensores de percepción utilizados (y combinaciones de ellos) que permiten obtener una representación tridimensional del entorno del *rover*. Muchos de ellos han sido probados y utilizados con éxito en varias misiones de exploración espacial.

Uno de los sensores más utilizados son las cámaras ópticas, que han formado parte de diversas misiones planetarias como la conocida *Mars Science Laboratory* con su *rover Curiosity*, que disponía de doce cámaras ópticas [3], o la reciente misión *Mars 2020*, cuyo *rover Perseverance* dispone de dieciséis cámaras ópticas de nueva generación [4]. El uso de dos cámaras ópticas en configuración estéreo permite inferir profundidad reconstruyendo el entorno, correlacionando píxeles de imágenes de izquierda a derecha gracias a procesos de calibración precisos[5]. Un par de imágenes estéreo rectificadas está calibrado de manera precisa cuando un píxel de la cámara izquierda puede encontrar su píxel correspondiente en la cámara derecha en la misma fila de la imagen, correspondiente a la línea epipolar. Esto permite estimar la disparidad y, como consecuencia, la profundidad de cada píxel en el marco de la cámara y eventualmente obtener una nube de puntos [6]. Éstos sistemas de cámaras estéreo deben combinarse con múltiples técnicas de medición y otros sensores para incrementar la fiabilidad del sistema. Especialmente es interesante la combinación de el sistema estéreo de cámaras con sensores láser que permitan detectar peligros que las cámaras no sean capaces de captar [7].

Sin embargo, tanto las cámaras ópticas como los algoritmos de odometría visual (que determinan la posición y la orientación de los *rovers* a partir de las imágenes generadas por las cámaras) no son capaces de inferir la profundidad de la imagen que generan de

forma precisa y rápida. El proceso de estimar la profundidad de cada píxel para crear un mapa en 3D del entorno requiere de recursos computacionales intensivos que resultan difíciles de incorporar en los módulos de procesamiento de los *rovers* espaciales. En la literatura se han estudiado soluciones para abordar la estimación de la transitabilidad del espacio frente al sensor sin necesidad de generar un modelo digital del terreno (DEM). Estas soluciones se basan en ajustar planos potenciales del suelo utilizando las disparidades proporcionadas por una cámara estéreo, y los peligros se detectan al medir las desviaciones de la altura promedio en el plano del suelo ajustado. Aunque esta solución de cómputo puede ser potencialmente más eficiente y rápida, se vuelve menos adecuada al aplicarse en aplicaciones espaciales debido a la dificultad de ajustar un plano del suelo en superficies planetarias irregulares y no estructuradas [8], [9].

Analizando las misiones espaciales reales en las que han participado *rovers*, el *Sojourner* inicial (1997) tenía funcionalidades de navegación limitadas. Fue con la llegada de los Rovers de Exploración de Marte (MERs) el momento en el que un *rover* implementó por primera vez capacidades de mapeo [10]. Sin embargo, el tiempo que tomaba el proceso cada vez que se utilizaban las funciones de mapeo y análisis de transitabilidad era del orden de minutos, reduciendo significativamente la velocidad neta de recorrido del *rover*. Debido al tiempo que tomaba ejecutarlas, las operaciones del *rover* siempre intentaban priorizar el uso del Modo Ciego de navegación. En este modo, los controladores del *rover* en Tierra proporcionaban un camino seguro para que el vehículo lo siguiera en forma de una secuencia de puntos de referencia, *waypoints*, sin necesidad de detenerse cada pocos metros para realizar el ciclo de mapeo, reduciendo así el aprovechamiento y potencial del vehículo. Los *rovers Curiosity* y *Perseverance* de la NASA se han beneficiado de módulos de cómputo más rápidos para reducir los tiempos de ejecución de estas funciones y aumentar el uso de modos de navegación más autónomos [11].

Para la prevista misión en 2026 *Mars Sample Return*, en la que el *Sample Fetch Rover* (SFR) recogerá muestras marcianas para traerlas de vuelta a la Tierra, se consideró y se realizó una prueba de concepto basada en los algoritmos SPARTAN acelerados por *hardware* con el objetivo reducir el tiempo de ejecución y procesamiento de los algoritmos mencionados [12]. A pesar de la mejora en el tiempo de ejecución, se ha descartado esta solución y, finalmente, el enfoque de diseño del SFR se basará en la reutilización de los algoritmos implementados en la misión *EuMars* [13]. También se ha seguido investigando en la técnica de odometría visual para mejorar su precisión y eficiencia computacional y favorecer su inclusión en los sistemas de navegación autónoma de futuras misiones espaciales con *rovers* [14].

Una alternativa a los métodos de mapeado y percepción del espacio mencionados son las técnicas de aprendizaje automático que se emplean cada vez más fuera del ámbito espacial para clasificar directamente el terreno en áreas seguras o peligrosas. Algunas líneas de investigación se centran en combinar las imágenes del terreno que tiene el vehículo por delante con datos de sensores propioceptivos durante el proceso de desplazamiento del *rover* sobre el terreno. Esto conduce a aumentar el conocimiento general del sistema de adquisición de datos y aprender las características de tracción de dicho terreno para eventualmente poder predecirlas antes de atravesarlo [15]. Sin embargo, estas técnicas para la clasificación del terreno y detección de obstáculos aún están lejos de implementarse en el espacio, ya que el no determinismo de la tecnología de aprendizaje automático dificulta su adopción en misiones espaciales. Otros métodos todavía en estudio y desarrollo que deben aún probar su efectividad y fiabilidad para el guiado de un vehículo espacial son los campos potenciales o los mapas de ocupación [16].

Otra alternativa para el mapeo tridimensional del espacio alrededor del *rover* es el uso

sensores Lidar (*Light Detection and Ranging*). Estos sensores emiten pulsos de luz láser y miden el tiempo que tarda en regresar después de reflejarse en los objetos. Esto permite obtener información detallada sobre la distancia, posición y forma de los obstáculos. Los sensores Lidar son altamente precisos y pueden proporcionar un mapeo tridimensional del entorno. Sin embargo, a pesar de que se han utilizado en tareas espaciales como el acoplamiento de la Estación Espacial Internacional [17], no son aún indicados para su uso en *rovers* debido a su gran peso y consumo de energía. Este alto consumo de energía es debido a la naturaleza activa del sensor que, a diferencia de las cámaras ópticas que son pasivas, el Lidar está constantemente emitiendo luz. Actualmente, las tecnologías emergentes de Lidar de estado sólido y flash podrían potencialmente abrir una nueva era de sensores para el espacio, pero es difícil prever cuándo podría suceder.

Finalmente, se dispone también de otra alternativa que ha sido poco utilizada en el ámbito de la exploración espacial. Se trata de los sensores *Time of Flight* (ToF). Éstos sensores funcionan de forma similar al Lidar, basados en el tiempo que tarda la señal emitida en volver al receptor. Sin embargo, este tipo de sensores utiliza haces de luz láser infrarroja. Estos sensores, a pesar de generar mediciones de distancias con una precisión algo menor, tienen un consumo de energía menor y un diseño compacto y poco pesado que les hace susceptibles de ser considerados como alternativas para la inferencia de la profundidad en misiones espaciales.

Debido a la falta de sensores ToF realmente fiables y precisos, las alternativas mencionadas anteriormente han sido utilizados preferentemente en los *rovers* espaciales. La reciente aparición de un nuevo sensor ToF que genera una matriz 8x8 que representa el mapeado del espacio ha motivado el desarrollo de este trabajo, que pretende estudiar la viabilidad del uso de este tipo de sensores en un *rover*. Este nuevo sensor realiza un mapeo del espacio tridimensional hacia el que apunta y genera una matriz de 64 casillas en la que cada celda indica la distancia hasta los objetos presentes en dicha región del espacio. Este sensor podría ser integrado en el complejo sistema de sensores fusionados que tienen los *rovers* para contribuir a la percepción tridimensional del espacio y permitir al sistema de evasión de obstáculos tomar mejores decisiones o contrastar la información recibida por otros sensores integrados.

3. Estudio y selección de los componentes

En esta sección se plasma el estudio realizado para determinar los componentes *hardware* idóneos para el desarrollo del presente trabajo. Para cada tipo de componente se han analizado diversas alternativas y se han seleccionado los componentes más adecuados en base a criterios de calidad, simplicidad y compatibilidad.

3.1. Microcontrolador

Un microcontrolador es un dispositivo integrado en un solo chip que combina las funciones de un microprocesador, memoria, periféricos de entrada/salida y unidades de control en un solo paquete. Es un componente esencial en la mayoría de los sistemas electrónicos modernos y se utiliza para controlar y coordinar diversas operaciones y funciones. Los microcontroladores están diseñados para ser programables, lo que significa que se les puede cargar un conjunto de instrucciones o un programa específico para realizar una tarea determinada.

Durante el proceso de selección del microcontrolador para el desarrollo de este trabajo, dos alternativas se analizaron debido a su popularización en el campo de la electrónica y a sus buenas características de coste y rendimiento: El microcontrolador ESP32 y la placa de desarrollo Arduino UNO , que tiene integrado el microcontrolador Atmega328P.

Para determinar el microcontrolador más adecuado al proyecto, se llevó a cabo un análisis comparativo entre ambas placas de desarrollo. Por un lado, la placa Arduino UNO presenta un consumo de energía menor que el ESP32 debido a su procesador de baja velocidad y menor cantidad de periféricos integrados. Además, Arduino UNO tiene una amplia comunidad de usuarios y una gran cantidad de recursos y bibliotecas disponibles que facilitan el desarrollo de proyectos. Sin embargo, en términos de potencia y conectividad el ESP32 presenta grandes ventajas. Por un lado, dispone de una velocidad de procesamiento de hasta 240 MHz, muy superior a los 16 MHz de su competidor. En cuanto a memoria disponible, el ESP32 tiene una memoria flash integrada de 4 MB y una memoria SRAM de hasta 512 KB. En contraste, la placa Arduino UNO tiene 32 KB de memoria flash y 2 KB de memoria SRAM. Esta diferencia permite que el ESP32 maneje programas y datos más grandes y complejos. Finalmente, el ESP32 tiene soporte incorporado para Wi-Fi y Bluetooth, siendo este último imprescindible para el desarrollo del proyecto. La placa Arduino UNO no presenta conectividad inalámbrica, por lo que sería necesario agregar módulos externos que implementen esta conectividad.

La gran diferencia de memoria y potencia de procesamiento, junto con la mayor conectividad inalámbrica del ESP32, hace evidente la elección de este microcontrolador para el desarrollo del proyecto. En cuanto a la placa de desarrollo sobre la que el ESP32 está embedido, se ha seleccionado la ESP32 NodeMCU, de la empresa Az-Delivery, ya que en el departamento de Ingeniería Electrónica de la Universitat Politècnica de València trabajan

con esta placa desde hace años y se disponía de existencias. En la *Figura 3.1* se presenta la placa de desarrollo ESP32 NodeMCU.

Figura 3.1: Placa de desarrollo ESP32 NodeMCU
fuente: www.fruugo.es

3.2. Motores y chasis

Para la implementación de dispositivos electrónicos de tamaño relativamente pequeño, se utilizan pequeños motores de corriente continua. Éstos convierten la energía eléctrica en energía mecánica mediante la interacción de campos magnéticos generados por imanes permanentes y electroimanes en el rotor y el estator del motor. Existen dos tipos de motores de corriente continua comúnmente utilizados: los motores con escobillas y los motores sin escobillas.

En los motores de corriente continua con escobillas (*DC brushed motors*) el rotor está compuesto por un eje central y un conjunto de bobinas llamadas devanados, que están conectadas a través de escobillas. Las escobillas son contactos de grafito o carbón que hacen contacto con las partes conductoras del rotor. El estator, por otro lado, consta de imanes permanentes dispuestos en una configuración fija alrededor del rotor. Estos imanes generan un campo magnético constante. Cuando se aplica una corriente eléctrica al rotor a través de las escobillas, se crea un campo magnético en el rotor que interactúa con el campo magnético constante del estator. Esto causa una fuerza electromagnética que hace que el rotor gire.

En los motores de corriente continua sin escobillas (*DC brushless motors*) el rotor está compuesto por imanes permanentes, mientras que el estator contiene bobinas de alambre de cobre dispuestas en forma de U o estrella. Además, éstos disponen de unos controladores electrónicos que están encargados de controlar el flujo de corriente en las bobinas y el tiempo de conmutación. A medida que los imanes permanentes del rotor son atraídos o repelidos por los campos magnéticos generados por las bobinas del estator, el rotor gira.

Los sensores o controladores electrónicos monitorizan continuamente la posición del rotor y ajustan la conmutación de las bobinas para mantener el movimiento suave y controlado. Este proceso de conmutación y control se repite rápidamente para mantener la rotación del motor en la dirección deseada y a la velocidad requerida.

Para el desarrollo de este trabajo se han seleccionado los motores de corriente continua con escobillas, ya que presentan un coste más bajo y un control más sencillo al no necesitar un controlador electrónico para funcionar correctamente.

En cuanto al chasis del *rover*, existe una gran cantidad de productos similares que se pueden adquirir para desarrollar proyectos electrónicos de este tipo. Cada uno de ellos presenta diferentes ventajas y desventajas, por lo que es necesario estipular las características que el chasis de este proyecto requiere y seleccionar la opción más adecuada. Por un lado, la baja velocidad de avance requerida y la necesidad de tener una buena tracción requieren un chasis que disponga de orugas en lugar en ruedas. Además, el pequeño tamaño de los componentes electrónicos y el requisito de poder maniobrar correctamente en espacios pequeños llenos de obstáculos requiere un *rover* con un chasis compacto y poco pesado. Finalmente, el requisito económico de ser un producto asequible para un Trabajo de Fin de Grado hace que el *Black Gladiator* de la empresa DFRobot sea la opción más adecuada. Además, el kit que se puede adquirir desde la página web de DFRobot incluye lo motores de corriente continua con escobillas.

Este chasis de aluminio de tan sólo 470 g tiene unas medidas de 19.3x16.3x6 cm, siendo muy compacto y montando unas orugas de plástico que le permiten operar en terreno montañoso. Además, como se puede ver en la *Figura 3.2* dispone de una placa plana sobre la que se pueden colocar los componentes electrónicos necesarios para su funcionamiento.

Figura 3.2: Chasis Black Gladiator de DFRobot
fuente: www.dfrobot.com

3.3. Sensor de distancias ToF

Como se ha mencionado en la *Sección 1.1*, la motivación del desarrollo de este Trabajo de Fin de Grado es la de estudiar la adecuación del nuevo sensor de distancias ToF VL53L5CX para un sistema de guiado autónomo de exploración espacial. Este novedoso sensor desarrollado por la empresa STMicroelectronics mantiene el funcionamiento de los sensores ToF, que emite pulsos de luz láser invisible hacia el objetivo y mide el tiempo

que tarda la luz en viajar hasta el objeto y regresar al sensor para determinar la distancia a la que se encuentra el objeto. Sin embargo, tiene un aspecto diferencial con todos sus predecesores, tiene capacidad de medida multizonal y genera una matriz de 8x8 en la que cada celda marca la distancia correspondiente a una zona del espacio mapeado.

El sensor utiliza un sistema de detección de luz que incluye un emisor láser VCSEL (*Vertical Cavity Surface Emitting Laser*), una matriz de detectores de fotones y un procesador digital de señales (DSP) para analizar y calcular las mediciones de distancia. Además, el VL53L5CX incorpora un algoritmo de detección de objetos y un procesamiento avanzado de señales que mejoran la precisión y la fiabilidad de las mediciones. Éstas son algunas de sus características principales:

- Matriz de detección: El VL53L5CX cuenta con una matriz de 8x8 detectores de fotones, lo que permite la medición simultánea de múltiples puntos en una escena. Esto proporciona una gran cantidad de información espacial y permite detectar objetos con mayor precisión.

- Rango de medición: El sensor tiene un rango de medición de hasta 4 metros en condiciones óptimas. Esto lo hace adecuado para aplicaciones que requieren mediciones de distancias en un rango moderado.

- Alta precisión: El sensor ofrece una alta precisión en las mediciones de distancia, con una resolución de hasta 4 mm. Esto permite detectar y medir objetos pequeños con precisión.

- Modos de funcionamiento configurables: El sensor ofrece diferentes modos de funcionamiento que se pueden configurar según los requisitos de la aplicación. Esto incluye la configuración de la tasa de muestreo, el rango de medición y el número de puntos de medición, lo que aporta flexibilidad y adaptabilidad en diferentes escenarios.

- Interfaz de comunicación: El VL53L5CX se comunica a través de una interfaz I2C, lo que permite una fácil integración con microcontroladores y otros dispositivos.

- Detección de movimiento y seguimiento: El VL53L5CX tiene la capacidad de detectar y rastrear objetos en movimiento. Esto lo hace adecuado para aplicaciones como robots autónomos, sistemas de seguimiento de personas, control de gestos y más.

En cuanto a las características técnicas del sensor, éste dispone de un campo de visión que genera una zona de detección de obstáculos de $45^{\underline{o}}$ en vertical y en horizontal, mientras que el receptor láser tiene un campo de visión de $55^{\underline{o}}$ en horizontal y $61^{\underline{o}}$ en vertical. Esto se puede ver en la *Figura 3.3*.

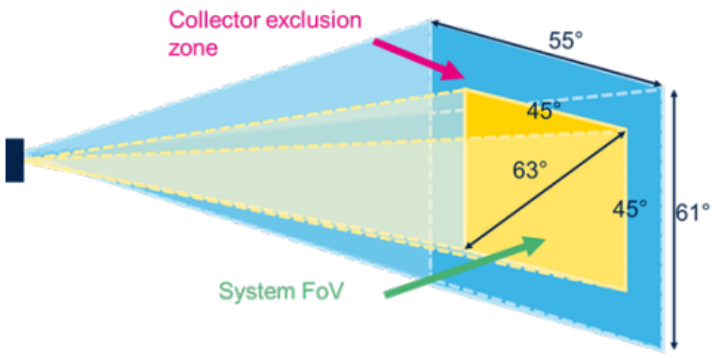


Figura 3.3: Campo de visión del sensor ToF VL53L5CX
fuente: VL53L5CX datasheet

Este campo de visión está dividido en 64 casillas (si se utiliza la resolución 8x8), de las cuales 12 de ellas (3 por cada esquina) se consideran medidas de esquina y el resto se consideran medidas internas como se puede observar en la *Figura 3.4*. Con esta resolución de medidas la tasa de muestreo mínima es de 1 Hz y la máxima de 15 Hz.

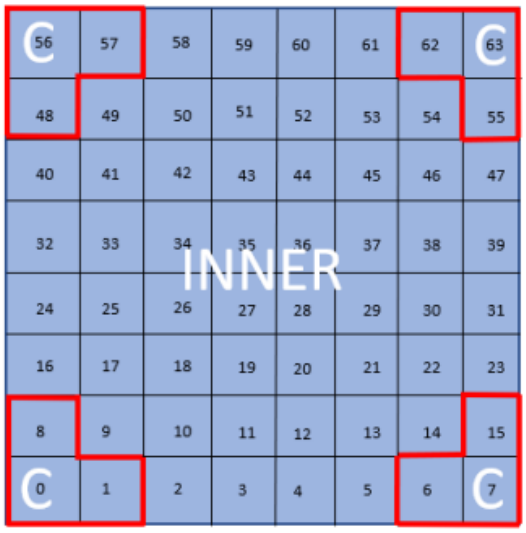


Figura 3.4: Disposición de las celdas de la matriz del sensor ToF VL53L5CX
fuente: VL53L5CX datasheet

Además, se debe tener en cuenta que debido al posicionamiento del emisor láser y el receptor del sensor, la imagen almacenada en la matriz está invertida frente a la imagen real con respecto al eje horizontal como se puede ver en la *Figura 3.5*.

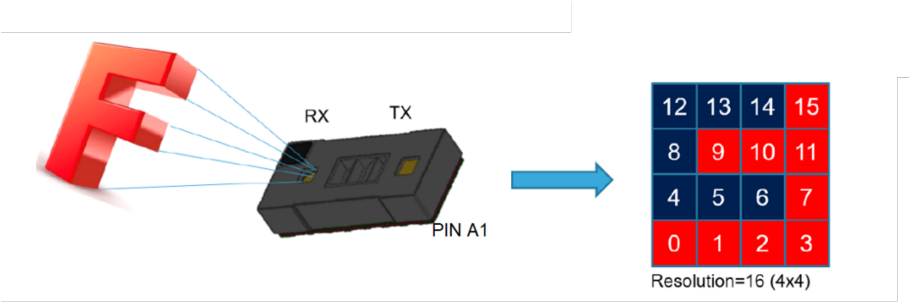


Figura 3.5: Inversión de la imagen real del sensor ToF VL53L5CX
fuente: VL53L5CX datasheet

Finalmente, es necesario también tener en cuenta que, a pesar de que el rango máximo del sensor es de 4000 mm, esto solo ocurre en unas condiciones óptimas de oscuridad con obstáculos blancos. Por tanto, para las condiciones de luminosidad ambiental a las que se operará el dispositivo, el rango máximo se reduce según la *Figura 3.6*

Target reflectance level. Full FoV (reflectance %)	Zone	Dark (0 klux)	Ambient light (5 klux)
White target (88 %)	Inner	Typical: 3500 mm Minimum: 2600 mm	Typical: 1100 mm Minimum: 950 mm
	Corner	Typical: 3100 mm Minimum: 1700 mm	Typical: 1000 mm Minimum: 800 mm
Grey target (17 %)[1]	Inner:	Typical: 1300 mm Minimum: 900 mm	Typical: 800 mm Minimum: 600 mm
	Corner	Typical: 1100 mm Minimum: 600 mm	Typical: 650 mm Minimum: 400 mm

1. *measured 13 % in IR at 940 nm*

Figura 3.6: Variación del rango máximo en función de la luminosidad ambiental
fuente: VL53L5CX datasheet

Una vez realizadas varias pruebas en condiciones ambientales se ha obtenido un rango máximo alrededor de los 800 mm.

Es importante también determinar cuándo una medida es válida y cuándo ha sido alterada y, por tanto, no debe considerarse como válida. Para ello, el sensor además de devolver la matriz de distancias de 64 celdas, genera otra matriz de estados de igual tamaño en la que se asigna un valor de estado a cada celda indicando la validez de la medida. De esta forma, a partir de las descripciones mostradas en la *Figura 3.7*, se debe comprobar si el *status* de cada medida es distinto de 5, en cuyo caso la medida no será fiable y se deberá descartar dicho valor en el algoritmo de control.

Target status	Description
0	Ranging data are not updated
1	Signal rate too low on SPAD array
2	Target phase
3	Sigma estimator too high
4	Target consistency failed
5	Range valid
6	Wrap around not performed (Typically the first range)
7	Rate consistency failed
8	Signal rate too low for the current target
9	Range valid with large pulse (may be due to a merged target)
10	Range valid, but no target detected at previous range
11	Measurement consistency failed
12	Target blurred by another one, due to sharpener
13	Target detected but inconsistent data. Frequently happens for secondary targets.
255	No target detected (only if number of target detected is enabled)

Figura 3.7: Descripción del valor de estado de las medidas del sensor ToF VL53L5CX
fuente: VL53L5CX user manual

La placa de desarrollo utilizada es la *SparkFun Qwiic ToF Imager - VL53L5CX*, que tiene integrado dicho sensor y proporciona un fácil acceso a los pines de configuración como se puede ver en la *Figura 3.8*

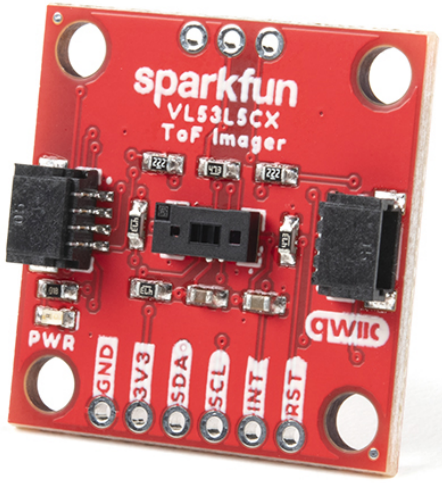


Figura 3.8: Placa de desarrollo SparkFun Qwiic ToF Imager - VL53L5CX
fuente: www.sparkfun.com

3.4. Unidad de medida inercial

La correcta operación y control de un vehículo de exploración espacial y, por tanto, de una réplica como la desarrollada en este trabajo, exige disponer de sensores que permitan

determinar la actitud del vehículo en todo momento. Las unidades de medida inerciales (IMU) son la mejor opción para ello, ya que combinan varios componentes, como un acelerómetro, un giroscopio y un magnetómetro, que permite conocer la orientación y actitud espacial del vehículo en un espacio tridimensional. Existen multitud de placas de desarrollo que incluyen estos sensores. Se ha decidido utilizar la placa de desarrollo Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055, desarrollada por la empresa Adafruit y utilizada con éxito en distintos proyectos anteriores en la universidad. Las características principales de esta placa son las siguientes:

- Sensor de 9 ejes: La IMU BNO055 integra un acelerómetro de 3 ejes, un giroscopio de 3 ejes y un magnetómetro de 3 ejes en un solo chip. Esto permite medir la aceleración lineal, la velocidad angular y la orientación magnética en los tres ejes espaciales.

- Fusión de sensores interna: La IMU BNO055 cuenta con un procesador de fusión de sensores interno que combina los datos de los diferentes sensores para proporcionar una salida de orientación precisa y estable. Esto simplifica el procesamiento de datos y reduce la carga en el microcontrolador o el sistema al que está conectado.

- Orientación absoluta: La IMU BNO055 ofrece la capacidad de calcular la orientación absoluta en relación con el norte magnético de la Tierra. Esto permite determinar la dirección y la posición relativa del dispositivo en relación con su entorno.

- Calibración automática: La IMU BNO055 incorpora un sistema de calibración automática que ajusta los sensores internos para compensar las variaciones y las influencias ambientales. Esto simplifica el proceso de configuración y mejora la precisión de las mediciones.

- Interfaz sencilla: La IMU BNO055 se comunica a través de una interfaz I2C, lo que facilita su integración con microcontroladores y otros dispositivos. Además, cuenta con bibliotecas de *software* y ejemplos de código disponibles para varios microcontroladores, lo que simplifica su programación y uso.

- Estabilidad de temperatura: La IMU BNO055 tiene una excelente estabilidad de temperatura, lo que reduce los errores causados por cambios de temperatura en los sensores y mejora la precisión de las mediciones a lo largo del tiempo.

- Modos de funcionamiento seleccionables: La IMU BNO055 ofrece diferentes modos de funcionamiento seleccionables, como el modo de orientación, el modo de detección de movimiento y el modo de brújula. Esto permite adaptar su uso a diferentes aplicaciones y requisitos específicos.

- Gran variedad de magnitudes medidas: La IMU BNO055 puede generar medidas de las siguientes magnitudes físicas: Orientación absoluta con vectores de Euler, orientación absoluta con cuaterniones, vector de velocidad angular, vector de aceleración, vector de fuerza del campo magnético, vector de aceleración lineal, vector de gravedad y temperatura.

Cómo se observa en la *Figura 3.9*, esta placa proporciona también facil acceso a los pines de entrada y salida que permiten conectarla con el microcontrolador.

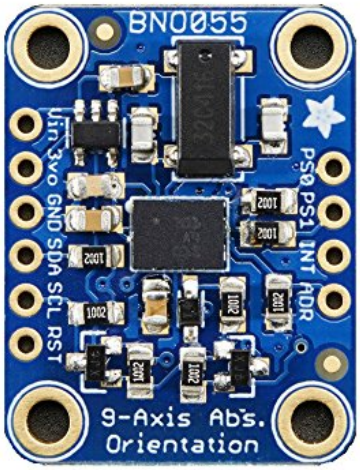


Figura 3.9: Unidad de medida inercial Adafruit - BNO055
fuente: www.learn.adafruit.com

3.5. Controlador de los motores

La velocidad de giro de los motores de corriente continua con escobillas se controla mediante las señales PWM (Pulse Wide Modulation) generadas y enviadas por el microcontrolador. Sin embargo, hay diversas razones por las que es necesario incluir un controlador de motores que actúe como un puente H, recibiendo la señal PWM del microcontrolador y transformándola para enviarla a los motores de corriente continua. Por ello, se ha decidido utilizar el módulo controlador de motores Driver L298 por ser ampliamente utilizado en proyectos electrónicos similares como vehículos teledirigidos y por haber sido utilizado con éxito en diferentes proyectos previos en la universidad. Éstas son algunas de las características y funciones que realiza el Driver L298

- Potencia y corriente: Los motores de corriente continua requieren niveles de potencia y corriente más altos que los que un microcontrolador puede proporcionar directamente. Los pines de salida de un microcontrolador generalmente no están diseñados para manejar corrientes significativas. En cambio, el Driver L298 actúa como un puente H, que es capaz de amplificar la señal de control del microcontrolador y proporcionar la potencia y corriente adecuadas para el motor

- Control de dirección: Los motores de corriente continua tienen la capacidad de girar en dos direcciones (hacia adelante y hacia atrás). El Driver L298 proporciona una interfaz para controlar la dirección del motor. Utilizando una combinación adecuada de señales de control, el driver L298 puede invertir la polaridad aplicada al motor, permitiendo así controlar la dirección del giro.

- Protección de circuitos: El Driver L298 también ofrece protección a los circuitos del microcontrolador y al propio motor. Incorpora diodos de protección de retroceso (*flyback diodes*) en su diseño, lo que evita daños causados por corrientes inducidas cuando el motor se apaga o cambia de dirección. Esto protege tanto el microcontrolador como el driver en caso de sobretensiones o voltajes inversos.

- Capacidad de manejar cargas inductivas: Los motores de corriente continua son cargas inductivas, lo que significa que generan una contracorriente al cambiar el estado de la corriente. El Driver L298 está diseñado específicamente para manejar cargas inductivas como los motores DC, gracias a su capacidad para manejar corrientes de retroceso y regular la velocidad de respuesta del motor.

- Flexibilidad de voltaje: El Driver L298 es capaz de trabajar con una amplia gama de voltajes de alimentación, lo que lo hace adecuado para diferentes tipos de motores de corriente continua. Puede manejar voltajes de alimentación de hasta 46 V, lo que proporciona una mayor flexibilidad en la selección de motores y en la adaptación a diferentes aplicaciones.

En la *Figura 3.10* se presenta el *Driver L298*.

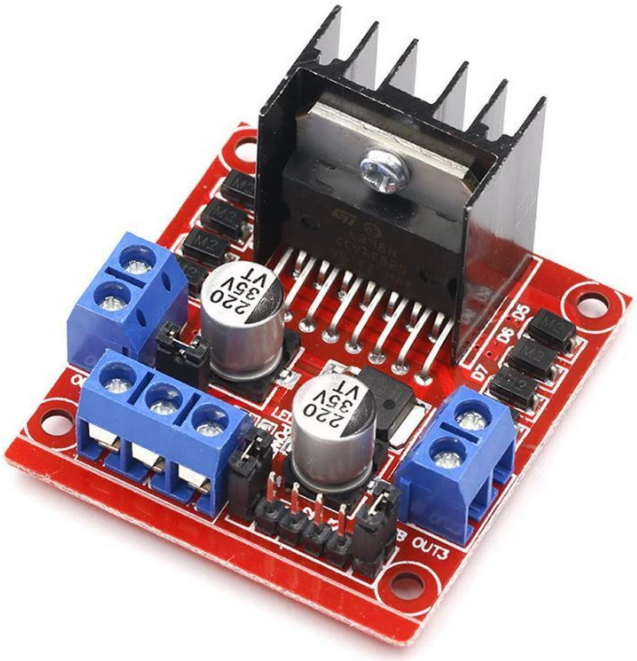


Figura 3.10: Controlador de motores Driver L298
fuente: www.prometec.net

4. Implementación hardware

En esta sección se describe detalladamente la interconexión de los componentes *hardware* seleccionados así como el ensamblaje final del prototipo que se utilizará en este trabajo como simplificación de *rover* espacial.

4.1. Diagrama de conexiones

El primer paso realizado previo a la interconexión de los componentes electrónicos fue el desarrollo de un diagrama de conexiones. Como se puede ver en la *Figura 4.1*, el microcontrolador ESP32 representa el punto común donde los sensores y actuadores están conectados.

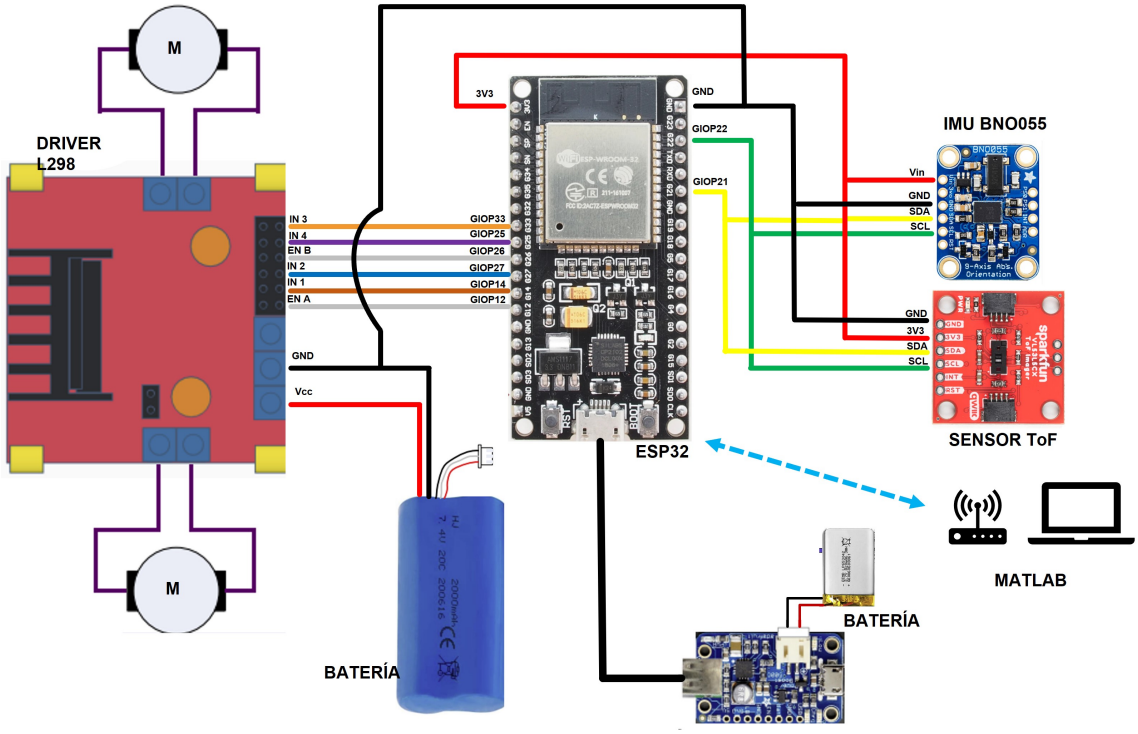


Figura 4.1: Diagrama de conexiones del sistema
fuente: elaboración propia

Por un lado, tanto la unidad inercial de medida como el sensor de distancias ToF transmiten datos al microcontrolador a través de la conexión SDA (*Serial Data Line*), estando ambos conectados al puerto GIOP21, que es el puerto del ESP32 destinado a la comunicación de datos I2C como se puede ver en la distribución de pines de la *Figura 4.2*. Ambos sensores están también conectados a través de la conexión SCL (*Serial Clock Line*), que representa la señal de reloj que sincroniza el sistema. Esta conexión se realiza a

través del puerto GIOP22, que es el destinado a tal fin. Finalmente, ambos sensores están conectados también a las señales de 3.3V y GND (*Ground*) del microcontrolador.

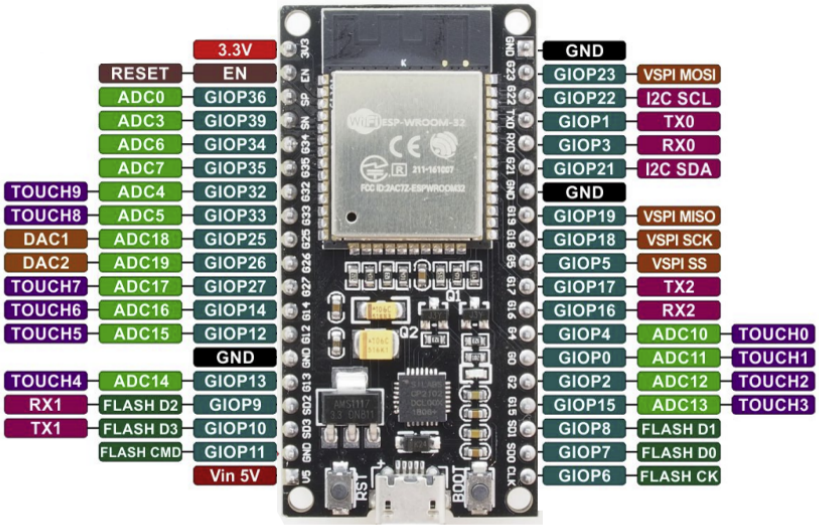


Figura 4.2: Esquema PINOUT del microcontrolador ESP32
fuente: transparencias Departamento Ingeniería Electrónica - UPV

Por otro lado, el Driver L298 dispone de cuatro pines de conexión IN1, IN2, IN3 e IN4 que se conectan a los puertos GIOP14, GIOP27, GIOP33 y GIOP25 respectivamente. A través de ellos se controla la dirección de giro de los dos motores. Además, dispone de dos pines ENA y ENB que se conectan a los puertos GIOP12 y GIOP26 respectivamente y cuya función es la de recibir la señal PWM generada por el microcontrolador que controla la velocidad de giro de los dos motores.

Es relevante comentar que, como se puede observar en la *Figura 4.1*, se han utilizado dos tipos de baterías para alimentar el sistema. Por un lado, se ha usado una batería LIPO con capacidad de 2000 mAh que alimenta al controlador de los motores con 7.4V. Por otro lado, se alimenta al microcontrolador con una batería más pequeña, de 820 mAh que opera a 3.7V. Esta batería pasa por una placa *DC-DC boost* (que también hace la función de cargador) que eleva la tensión de 3.7V a 5V. La salida del *DC-DC boost* de 5V es la que alimenta al módulo del ESP32, que tiene un regulador interno para convertir esos 5V a 3.3V, que es el la tensión a la que trabaja el microcontrolador.

4.2. Ensamblaje del prototipo

Una vez el diagrama de conexiones fue revisado y su funcionamiento fue comprobado, se procedió al ensamblaje del prototipo.

El uso del sensor ToF como principal fuente de información de los obstáculos alrededor del *rover* hace necesaria la introducción de alguna superficie que permita elevar su posición con respecto al chasis. De esta forma, el campo de visión del sensor será más amplio. Además, es necesario también incluir un segundo nivel sobre el que colocar los componentes electrónicos ya que la placa del chasis es demasiado pequeña para colocarlos todos juntos. Esto, si se realiza una distribución a consciencia, favorece la disminución del ruido que

podrían tener las señales de los sensores debido a la cercanía de la batería y los motores al microcontrolador.

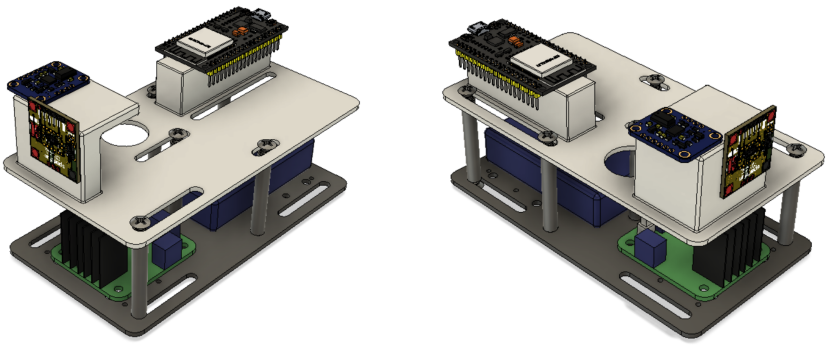


Figura 4.3: Modelo 3D del sistema
fuente: elaboración propia

Por todo ello, se ha diseñado un modelo 3D del prototipo (ver *Figura 4.3*) y se ha diseñado y fabricado mediante impresión 3D una placa que se situará unos 5 cm por encima de la placa del chasis. Esta placa, que se puede observar en la *Figura 4.4* dispone de unos agujeros por los que se pasarán los cables que conectan los sensores y los actuadores con el microcontrolador. Además, dispone de unas ranuras en las que se colocarán los tornillos que ajustan unas torretas que permiten elevarla por encima de la placa del chasis como se puede ver en la *Figura 4.3*.

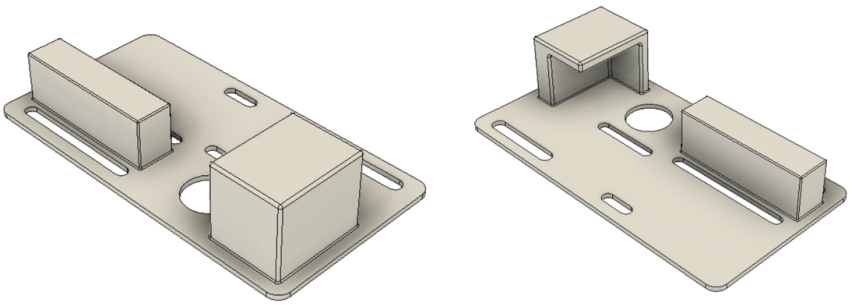


Figura 4.4: Modelo 3D de la placa superior del ensamblaje
fuente: elaboración propia

En cuanto a la disposición de los componentes mostrada en la *Figura 4.3*, se ha colocado la batería principal y el Driver L298 en el nivel inferior, para mantener el centro de gravedad lo más bajo posible y favorecer las conexiones con los motores. En el nivel superior se han colocado el microcontrolador, la IMU y el sensor de distancias ToF. De esta forma, se consigue separar estos elementos de la batería y el controlador de los motores que podrían introducir algo de ruido en las señales. En cuanto a la batería más pequeña, ésta se introduce solamente cuando el *rover* está en funcionamiento y se coloca encima de la batería principal, de forma que se tenga un acceso fácil al interruptor de encendido y al puerto microUSB del microcontrolador. En la *Figura 4.5* se puede observar el aspecto final del prototipo.

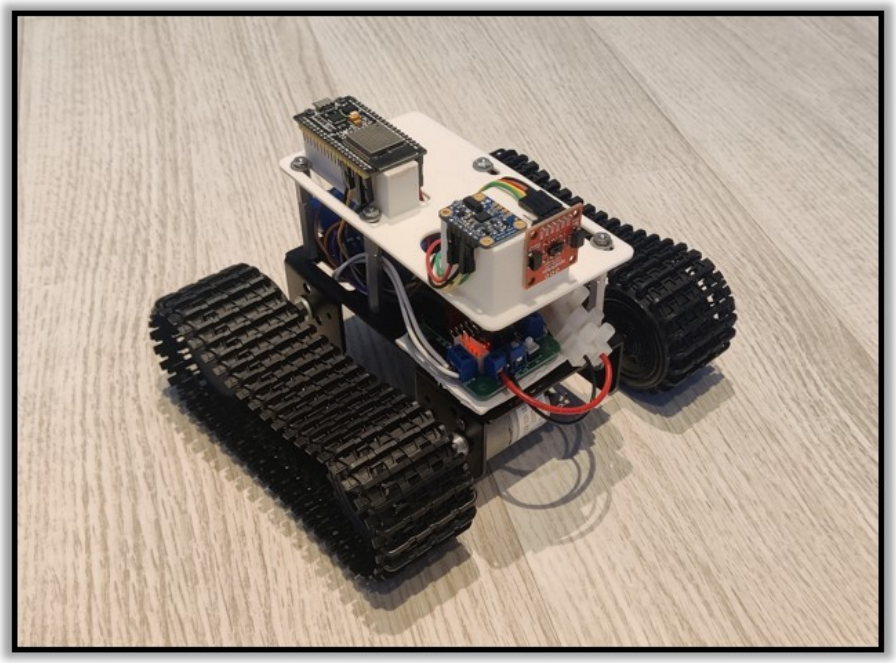


Figura 4.5: Ensamblaje final del prototipo
fuente: elaboración propia

Cabe mencionar que, debido a que el sensor de distancias ToF tiene un campo de visión con un ángulo vertical de 45^{o}, se ha inclinado la posición del sensor apuntando hacia arriba un pequeño ángulo para evitar que se generen medidas que correspondan al suelo sobre el que se desplaza el vehículo.

5. Implementación software

En esta sección se describe detalladamente la implementación *software* desarrollada, desde los protocolos de comunicación entre cada módulo hasta el código fuente de cada módulo *software* que compone el sistema de guiado y control del *rover*.

5.1. Protocolos de comunicación

La comunicación entre los distintos componentes electrónicos utilizados en el desarrollo del trabajo se ha basado en dos protocolos principalmente. El protocolo de comunicación en serie I2C (*Inter-Integrated Circuit*) se ha utilizado para la comunicación entre los sensores y actuadores con el microcontrolador. Por otro lado, el protocolo de comunicación inalámbrica Bluetooth se ha utilizado para la comunicación del microcontrolador con el *software* Matlab.

5.1.1. Comunicación serie I2C

El protocolo de comunicación I2C (Inter-Integrated Circuit) es un estándar ampliamente utilizado en Arduino y otros dispositivos electrónicos para permitir la comunicación entre diferentes componentes. Funciona como un bus de datos bidireccional, lo que significa que varios dispositivos pueden compartir la misma línea de comunicación.

El funcionamiento del protocolo I2C en Arduino se basa en dos líneas de comunicación principales: SDA (*Serial Data Line*) y SCL (*Serial Clock Line*). Estas líneas son utilizadas para transmitir datos entre el microcontrolador y los dispositivos periféricos conectados.

El funcionamiento básico del protocolo I2C es el siguiente:

- Inicio de la comunicación: El maestro (en este caso, el microcontrolador) inicia la comunicación enviando una señal de inicio. Esta señal consiste en una transición de alto a bajo en la línea SDA mientras que la línea SCL se mantiene en alto.

- Dirección del dispositivo: El maestro envía la dirección del dispositivo al que desea comunicarse. Cada dispositivo en el bus I2C tiene una dirección única asignada. La dirección puede ser de 7 bits (para la mayoría de los dispositivos) o de 10 bits (para dispositivos especiales). El maestro especifica si desea leer o escribir en el dispositivo mediante el bit de lectura/escritura.

- Transmisión de datos: Una vez que se ha establecido la comunicación con el dispositivo objetivo, se pueden transmitir los datos. Tanto el maestro como el dispositivo objetivo pueden enviar y recibir datos. El maestro envía los datos a través de la línea SDA, y el dispositivo objetivo responde a través de la misma línea.

- Confirmación de recepción: Después de cada byte de datos enviado, el dispositivo objetivo envía una señal de ACK (*Acknowledgement*) para confirmar que ha recibido correctamente los datos. Si el maestro recibe el ACK, procede a enviar el siguiente byte de datos. Si no recibe el ACK, puede tomar medidas correctivas o finalizar la comunicación.

- Finalización de la comunicación: El maestro finaliza la comunicación enviando una señal de parada. Esta señal consiste en una transición de bajo a alto en la línea SDA mientras que la línea SCL se mantiene en alto. Después de la señal de parada, el bus I2C vuelve a su estado inactivo y otros dispositivos pueden iniciar nuevas comunicaciones.

  Una de las ventajas del protocolo I2C es que permite conectar múltiples dispositivos a un solo bus utilizando solo dos líneas de comunicación. Esto permite conectar de forma más sencilla la unidad de medida inercial y el sensor de distancias ToF. Además, el protocolo I2C es relativamente fácil de implementar en Arduino gracias a las bibliotecas disponibles que simplifican la configuración y el manejo de la comunicación.

5.1.2. Comunicación inalámbrica Bluetooth

El protocolo de comunicación Bluetooth es una tecnología inalámbrica ampliamente utilizada en dispositivos electrónicos para establecer conexiones de corto alcance y transmitir datos de manera inalámbrica. En el caso de Arduino, el uso del protocolo Bluetooth permite la comunicación con otros dispositivos, como teléfonos inteligentes, tabletas u ordenadores, brindando una forma conveniente de controlar y monitorizar proyectos. Para este trabajo se ha utilizado la librería *BluetoothSerial* disponible en arduino para gestionar la conexión Bluetooth entre el microcontrolador y Matlab.

5.2. Control de los motores. Señal PWM

El control de velocidad de los motores se realiza mediante una señal PWM, que se controla a través del ciclo de trabajo (*Duty Cycle*). El *Duty Cycle*, es un parámetro utilizado en la modulación de ancho de pulso (PWM) para controlar la velocidad y la potencia de los motores de corriente continua.

La técnica de control mediante PWM consiste en generar una señal digital cuadrada, donde el tiempo de encendido y apagado del pulso se modifica para controlar la cantidad de potencia suministrada al motor. El *Duty Cycle* se refiere a la proporción de tiempo que la señal está en estado activo (encendido) en comparación con el período completo del ciclo.

El *Duty Cycle* se expresa como un porcentaje, que varía de 0 % a 100 %. Un *Duty Cycle* del 0 % indica que la señal está siempre en estado apagado, mientras que un *Duty Cycle* del 100 % indica que la señal está siempre en estado encendido. Tener un valor del 50 % significa que la señal está encendida la mitad del tiempo y apagada la otra mitad.

Al controlar el *Duty Cycle* de la señal PWM que se envía al motor DC, se puede ajustar la cantidad promedio de potencia suministrada. Por ejemplo, si se utiliza un *Duty*

Cycle del 50 %, el motor recibirá la mitad de la potencia máxima, lo que resultará en una velocidad reducida. Si se utiliza un *Duty Cycle* del 100 %, el motor recibirá la potencia máxima y funcionará a máxima velocidad.

En la *Figura 5.1* se muestra un diagrama que ilustra diferentes valores de *Duty Cycle* para una señal con el mismo periodo.

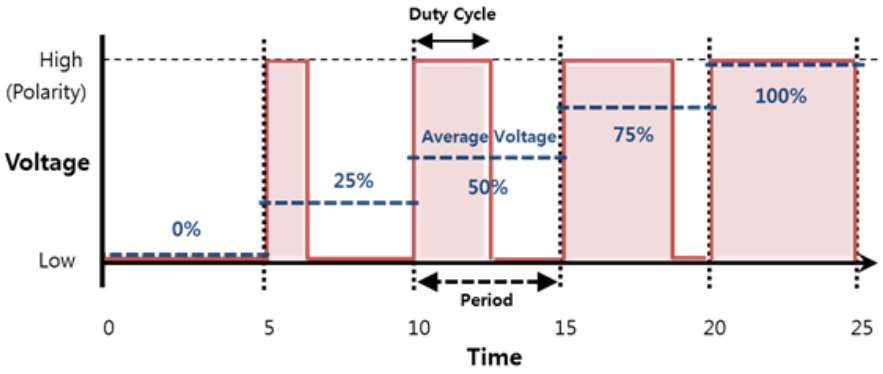


Figura 5.1: Varios *Duty Cycle* para una señal con el mismo periodo
fuente: docs.tizen.org

Las señales PWM son generadas por el microcontrolador y enviadas a los motores a través del controlador de motores. Las señales originales tienen una tensión de 3.3V y una corriente de salida máxima de 20 mA. El controlador de motores se encarga de aumentar la tensión y corriente de dichas señales hasta valores adecuados a los motores que tiene conectados. Además, como se ha mencionado en la *Sección 3.5* el Driver L298 también permite realizar el cambio de sentido de giro en los motores.

5.3. Sistema de referencia del rover

Un *rover* inmerso en una misión de reconocimiento espacial debe realizar maniobras que permiten modificar su posición y orientación en las tres dimensiones del espacio. Sin embargo, la simplificación de este modelo se hace de forma que se considera que el vehículo se desplaza por una superficie plana constantemente. Además, la posición bidimensional no es una variable que se utiliza en este sistema de guiado y control autónomo, por lo que la única variable espacial relevante es la posición angular del *rover* con respecto al eje longitudinal original. De esta forma, simplemente con conocer la variación relativa del ángulo de orientación del vehículo se puede operar este sistema de guiado. Para ello, debido a que el objetivo es conseguir un sistema de control que guíe al vehículo en línea recta evitando obstáculos en el camino, se ha limitado la rotación del *rover* a 90^{o} hacia cada dirección con respecto a la posición inicial del eje longitudinal. Es decir, si se considera un sistema de referencia como el mostrado en la *Figura 5.2*, el vehículo parte con un ángulo inicial de 0^{o} y puede conseguir valores angulares en el rango $[0^{o}, 90^{o}]$ si se produce una rotación hacia la derecha (CW) y valores en el rango $[270^{o}, 360^{o})$ si se produce una rotación hacia la izquierda(CCW).

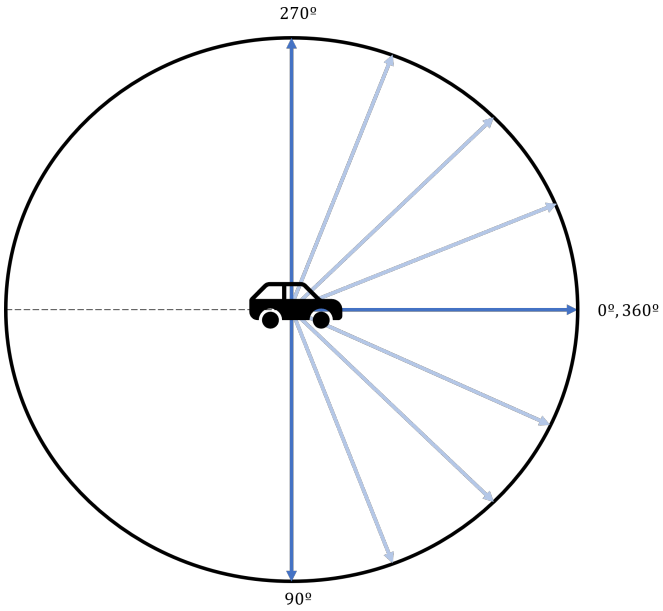


Figura 5.2: Sistema de referencia del *rover*
fuente: elaboración propia

5.4. Algoritmo de control PID

Antes de entrar en detalle en la implementación y el funcionamiento del código, se debe entender qué es un control PID y cómo funciona, ya que es uno de los bloques *software* principales del sistema de guiado y control.

El PID (*Proportional-Integral-Derivative*) es un controlador ampliamente utilizado en sistemas de control automático para mantener una variable controlada en un valor deseado. El funcionamiento del controlador PID se basa en el cálculo de una señal de control utilizando tres componentes principales: la ganancia proporcional (Kp), la ganancia integral (Ki) y la ganancia derivativa (Kd). Cada una de estas ganancias tiene un significado y una influencia específica en el comportamiento del sistema.

- La ganancia proporcional Kp está relacionada directamente con el error presente en el sistema en un instante dado. El control proporcional proporciona una señal de control que es proporcional al error actual entre la variable controlada y el valor deseado. Una ganancia proporcional alta puede producir una respuesta rápida, pero también puede causar oscilaciones o inestabilidad si se establece demasiado alta. Por otro lado, una ganancia proporcional baja puede resultar en una respuesta lenta o insuficiente del sistema.

- La ganancia integral Ki tiene en cuenta el error acumulado a lo largo del tiempo. El término integral se encarga de corregir cualquier desviación persistente o error de estado estacionario en el sistema. La acción integral del controlador aumenta con el tiempo y compensa los errores acumulados. Esto permite eliminar el error constante o de *offset* presente en el sistema. Una ganancia integral alta puede ayudar a eliminar rápidamente el error de estado estacionario, pero también puede causar una respuesta lenta y sobrepasar el objetivo deseado si se configura incorrectamente.

- La ganancia derivativa Kd tiene en cuenta la tasa de cambio del error. La acción derivativa del controlador predice la tendencia futura del error y proporciona una señal de control que es proporcional a la velocidad de cambio del error. La ganancia derivativa ayuda a mejorar la estabilidad del sistema y a reducir el tiempo de respuesta a cambios rápidos. Una ganancia derivativa alta puede proporcionar una respuesta rápida y amortiguar las oscilaciones, pero también puede amplificar el ruido presente en la señal y causar una respuesta inestable si se establece demasiado alta.

La señal de control se calcula como la suma aritmética del producto de cada ganancia por su correspondiente error. El esquema del controlador PID se presenta en la *Figura 5.3*. En el caso particular de este proyecto, la señal de control que se quiere obtener es la señal PWM que actúa sobre los motores de corriente continua. Normalmente, en este tipo de aplicaciones se utiliza la velocidad actual de de los motores y la velocidad deseada para ajustar esta señal de control. Sin embargo, debido a que los motores utilizados no tienen *encoders* (que son los que permiten medir la velocidad de giro) y que la velocidad de avance que transmiten las orugas al suelo no es la misma que la velocidad de giro de los motores, se debe buscar otra magnitud para generar la señal de control. La magnitud que permite medir la diferencia entre la orientación deseada del vehículo y la orientación actual es la posición angular que se obtiene de la IMU. Por tanto, los errores se calcularán a partir de las variables *current_angle*, que almacena la orientación actual del *rover*, y *desired_angle*, que almacena la orientación deseada del *rover*.

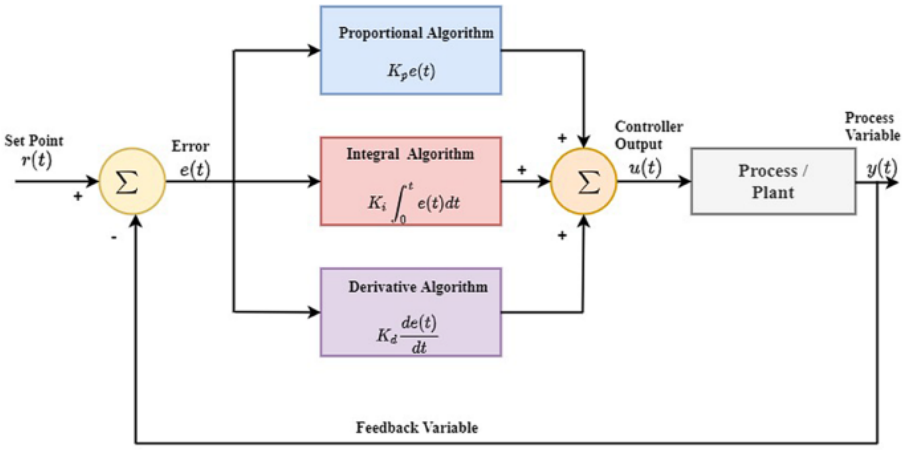


Figura 5.3: Diagrama del controlador PID
fuente: link.springer.com

5.5. Sistema de control y guiado

El sistema de control y guiado del *rover* diseñado está formado por diferentes módulos *software* que interactúan entre ellos para tomar la mejor decisión posible sobre las órdenes que el microcontrolador debe mandar a los motores de corriente continua. De esta forma, el programa principal consiste en un bucle en el que en cada iteración el microcontrolador recibe los datos de los sensores y los manda a través del protocolo Bluetooth a Matlab, dónde los datos son procesados y se determina la acción de control necesaria sobre el

vehículo. Esta orden es mandada desde Matlab al microcontrolador, que es el que se encarga de comunicarse con el controlador de los motores.

En la *Figura 5.4* se presenta el diagrama de flujo del sistema en el que está representado el flujo de datos entre los diferentes módulos *hardware*.

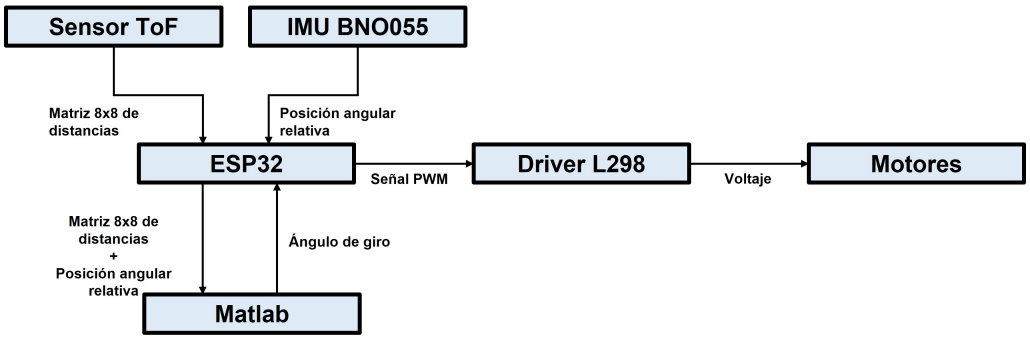


Figura 5.4: Diagrama de flujo de datos del sistema
fuente: elaboración propia

Como se puede observar en la *Figura 5.4*, el sensor de distancias ToF envía la matriz 8x8 de distancias al microcontrolador, mientras que la IMU envía los datos medidos en sus nueve grados de libertad. Sin embargo, el único dato utilizado por el algoritmo es la posición angular relativa del vehículo con respecto al eje longitudinal inicial. Cabe mencionar que, además de la matriz 8x8 de distancias, también se envía otra matriz que indica la validez de cada una de las 64 medidas de distancia. Esto no se ha incluido en el diagrama de flujo por simplicidad.

Posteriormente, el microcontrolador envía estos datos al módulo de *software* en Matlab a través de una conexión Bluetooth. El *script* de Matlab procesa los datos recibidos y determina el ángulo y la dirección hacia la que debe rotar el vehículo para proseguir su movimiento sin colisionar con los obstáculos que puedan haber en su camino.

Una vez determinado el ángulo de giro, el microcontrolador aplica un algoritmo de cálculo que determina la señal PWM que debe enviar a los motores, pasando por el Driver L298.

Para conocer en más detalle la comunicación entre los diferente módulos *software* del sistema y la toma de decisiones que se realiza, se presenta en la *Figura 5.5* un diagrama de bloques que representa la interconexión e interacción de los módulos *software* que componen el sistema.

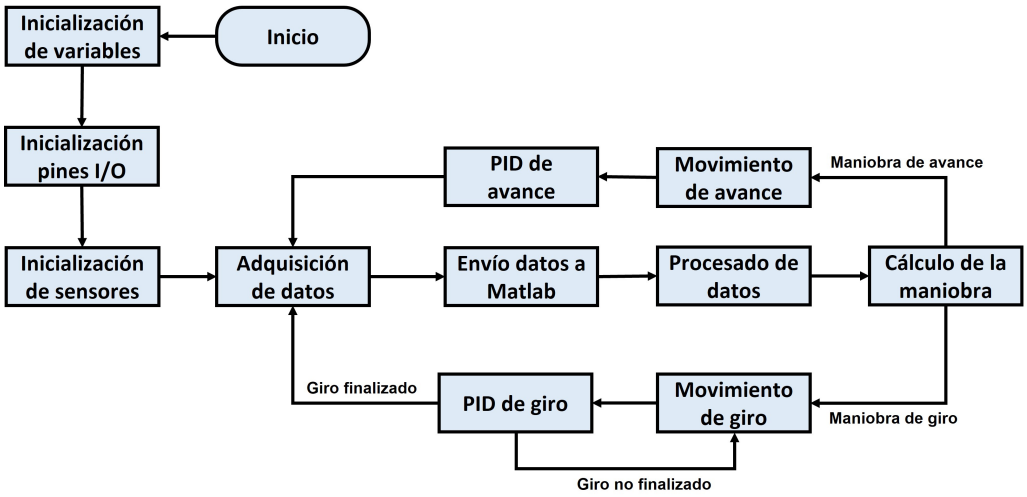


Figura 5.5: Diagrama de módulos software del sistema
fuente: elaboración propia

5.5.1. Inicialización de variables, pines y sensores

El código creado para el microcontrolador está compilado y ejecutado en Arduino. Al inicio del *script* de Arduino se incluyen las librerías necesarias y se declaran e inicializan algunas de las variables y los pines de salida y entrada del microcontrolador como se presenta en el *Código 5.1*. Las librerías incluídas permiten trabajar con objetos que facilitan enormemente la manipulación y comunicación de los módulos *software*. El código es sencillo y está suficientemente comentado para su correcta interpretación, por lo que solamente se van a hacer un par de comentarios acerca de esta porción de código.

```
1  // Include needed lybraries
2  #include <BluetoothSerial.h>
3  #include <Wire.h>
4  #include <Adafruit_Sensor.h>
5  #include <Adafruit_BNO055.h>
6  #include <utility/imumaths.h>
7  #include <SparkFun_VL53L5CX_Library.h>
8
9  // Initialize Bluetooth, IMU and ToF sensor objects object
10 BluetoothSerial SerialBT;
11 Adafruit_BNO055 bno = Adafruit_BNO055(55);
12 SparkFun_VL53L5CX myImager;
13
14
15 // Left DC motor pins
16 int motor1Pin1 = 25;
17 int motor1Pin2 = 33;
18 int enable1Pin = 12;
19
20 // Right DC motor pins
21 int motor2Pin1 = 14;
22 int motor2Pin2 = 27;
23 int enable2Pin = 26;
24
25 // Set PWM properties
26 const int freq = 30000;
27 const int pwmChannel1 = 0;
```

```
28 const int pwmChannel2 = 1;
29 const int resolution = 8;
30 int dutyCycle = 250;
31 int dutyCycle_forward = 0.78*dutyCycle;
32 int dutyCycle_left;
33 int dutyCycle_right;
34
35 // ToF sensor variables
36 VL53L5CX_ResultsData measurementData; // Result data class structure,
       1356 byes of RAM
37 SF_VL53L5CX_RANGING_MODE ranging_mode;  // Sets ToF sensor ranging mode
38 int imageResolution = 0;
39 int imageWidth = 0;
40
41 // IMU variables
42 sensors_event_t event;        // Get a new sensor event
43 float current_angle;          // Current IMU angle
44
45 // PID variables
46 bool PID_working = false;
47 float desired_angle = 0;      // Desired angle
48 float kp = 0;                 // Proportional gain for making a turn
49 float ki = 0;                 // Integral gain for making a turn
50 float kd = 0;                 // Derivator gain for making a turn
51 float kp_forward = 7;         // Proportional gain for forward movement
52 float ki_forward = 10;        // Integral gain for forward movement
53 float kd_forward = 0;         // Derivator gain for forward movement
54
55 float error;                  // Error between desired and current angle
56 float last_error;             // Error in previous iteration
57 long prevT;                   // Time variable to compute integral error
58 float eintegral = 0;          // Integral error
59 float ederivator = 0;         // Derivator error
60 float u;                      // Control action
61 long currT;                   // Time variable to compute integral error
62 float deltaT;                 // Time elapsed between two iterations
63
64 // Other needed variables
65 int flag = 0;                 // Flag indicating rover start
66 int data;                     // Data received from Matlab
```

5.1: Inicialización de variables y objetos

Como se observa en la sección *Set PWM properties*, se definen varios *DutyCycles*. El primero hace referencia al *DutyCycle* máximo al que operarán los motores, que es de 250. La variable *DutyCycle_forward* hace referencia al valor de la señale PWM que reciben los motores cuando el *rover* está avanzando en línea recta. Las dos últimas variables hacen referencia a los valores de la señal PWM que tiene cada motor durante la aplicación del PID en una maniobra de giro.

En la sección *ToF sensor variables*, se declara la variable *ranging_mode*, que modifica el modo de operación del sensor ToF, que operará en régimen continuo.

En la sección *PID variables* se declaran variables que hacen referencia a las ganancias de un PID para dos versiones distintas, cuando se requiere hacer un giro y cuando se requiere avanzar en línea recta. Ambos PID son independientes y por eso tienen ganancias diferentes, siendo las ganancias del PID de avance fijas y las ganancias del PID de giro variables en función del giro que se realizará.

En la sección *setup* del código se inicializan las variables y los sensores que no han sido

inicializados anteriormente y se determina la configuración inicial de los objetos que hacen referencia a los motores y a los sensores. El *Código 5.2* muestra la implementación de esta parte del sistema.

```
/* Initial configuration */
void setup() {
  SerialBT.begin("ESP32BT");  // Inicialize Bluetooth object
  Serial.begin(115200);       // Inicialize serial communication
  Wire.begin();               // Resets to 100kHz I2C
  Wire.setClock(400000);      // Sets max I2C freq to 400kHz

  // Initialise the IMU
  if(!bno.begin())
  {
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }

  // Initialise the ToF Sensor
  if (myImager.begin() == false)
  {
    Serial.println(F("Sensor not found - check your wiring. Freezing"));
    while (1);
  }

  // Initial configuration of ToF Sensor
  myImager.setResolution(8*8);                    // Enable all 64 pads
  imageResolution = myImager.getResolution();     // Query sensor for current resolution - either 4x4 or 8x8
  imageWidth = sqrt(imageResolution);             // Calculate printing width
  myImager.setRangingFrequency(10);               // Sets ranging frequency to 10Hz
  ranging_mode = SF_VL53L5CX_RANGING_MODE::CONTINUOUS;  // Sets operating mode to CONTINUOUS
  myImager.setRangingMode(ranging_mode);
  myImager.setSharpenerPercent(10);               // Sets sharpener percentage to 10
  myImager.startRanging();                        // Start ranging operation

  // Configure the motor pins as outputs
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(enable2Pin, OUTPUT);

  // Configure LED PWM functionalitites
  ledcSetup(pwmChannel1, freq, resolution);
  ledcSetup(pwmChannel2, freq, resolution);

  // Attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, pwmChannel1);
  ledcAttachPin(enable2Pin, pwmChannel2);

  delay(1000);

```

```
51   // Use external crystal as clock (better timing accuracy for IMU)
52   bno.setExtCrystalUse(true);
53 }
```

5.2: Función *setup* de Arduino

Como se puede observar, inicialmente se configuran e inicializan las comunicaciones Bluetooth y serie. También se comprueba la correcta inicialización de la IMU y del sensor de distancias ToF.

En cuanto a la inicialización del sensor de distancias ToF, se configura la resolución de 8x8 y la frecuencia de muestreo de 10 Hz. Además, se configura el modo de operación en *continuous*. En este modo el sensor realiza un muestreo constante y no solamente cuando el microcontrolador se lo pide. Esto mejora la precisión y el alcance máximo del sensor a expensas de consumir una mayor energía. Finalmente, se configura el porcentaje del *sharpener* al 10 %. Este parámetro permite al sensor realizar un ajuste de las medidas de distancia de forma que se produce un mayor contorneo de los objetos detectados afilando las esquinas de los objetos. El ajuste de este parámetro debe hacerse en base a realizar varias pruebas y comprobar el valor que mejor funciona en las condiciones ambientales en las que se opera y en función de los objetos que debe detectar. El valor del 10 % fue deteminado realizando este tipo de pruebas hasta que se consiguió el resultado que más se ajustaba a la realidad. Cómo se puede observar en la *Figura 5.6*, hay que determinar el valor óptimo del parámetro ya que un valor superior o inferior a éste puede reducir la precisión y aportar una información distorsionada de la realidad medida.

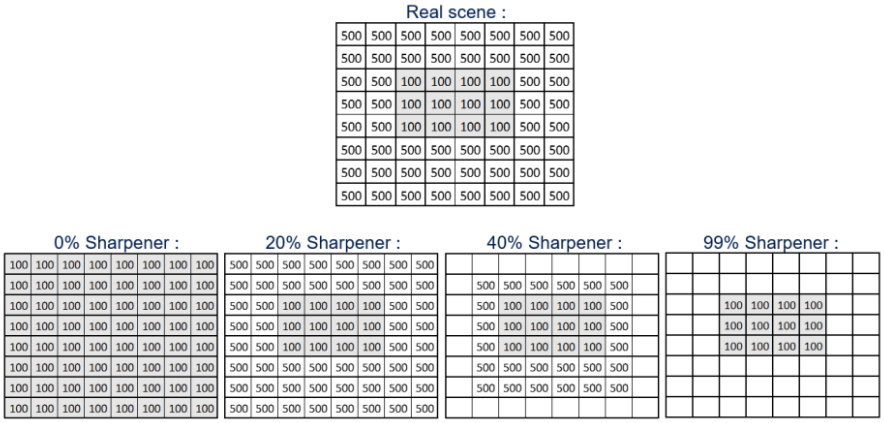


Figura 5.6: Ajuste del parámetro *sharpener*
fuente: VL53L5CX user manual

5.5.2. Adquisición y envío de datos a Matlab

Una vez inicializados los objetos y las variables necesarias, el sistema entra en el bucle de ejecución cuyo primer paso consiste en la adquisición de los datos medidos por los sensores. Estos datos son directamente enviados al módulo de *software* en Matlab para su procesado. En el *Código 5.3* se puede ver la implementación de esta parte del código.

```
1 /* Acquire data measured from sensors and send it to Matlab */
2 void data_acquisition(){
```

```
3   if (myImager.getRangingData(&measurementData)){   //Read distance
      data into array
4     // Send ToF data with decreasing y (sensor mounted upside down),
      decreasing x to reflect reality
5     for (int y = imageWidth * (imageWidth - 1); y >= 0 ; y -=
      imageWidth)
6     {
7       for (int x = imageWidth - 1 ; x >= 0 ; x--)
8       {
9         // Send ToF distances to Matlab
10        SerialBT.write((int) measurementData.distance_mm[x + y]/256);
11        SerialBT.write((int) measurementData.distance_mm[x + y]%256);
12
13        // Send ToF distances status to Matlab
14        SerialBT.write((int) measurementData.target_status[x + y]/256);
15        SerialBT.write((int) measurementData.target_status[x + y]%256);
16      }
17    }
18    // Send IMU data to Matlab
19    SerialBT.write( (int) current_angle/256);
20    SerialBT.write( (int) current_angle%256);
21
22    data = 0;    // Reset data value
23  }
24 }
25
26
27 /* System execution loop */
28 void loop() {
29   // Matlab sends flag = 1 to start the operation of the system
30   if(flag == 0 & SerialBT.available()){
31     flag = SerialBT.read();
32   }
33
34   // Control algorithm execution
35   if(flag == 1){
36
37     // Matlab sends one byte indicating the operation to execute
38     if(SerialBT.available()){
39       data = SerialBT.read();
40     }
41
42     // Read data from IMU
43     bno.getEvent(&event);
44     current_angle = event.orientation.x;
45
46     // Execute operation indicated by Matlab
47     if(data == 3){     // Acquire sensor data and send it to Matlab
48       data_acquisition();
49     }
50   }
51 }
```

5.3: Adquisición de datos

Como se observa, el bucle de ejecución comprueba primero si Matlab ha enviado un byte con valor 1 indicando que el proceso de operación del *rover* debe iniciarse. A partir de recibir esa señal (*flag*), empieza el algoritmo de control. Este algoritmo se basa en la comunicación entre el microcontrolador y Matlab de forma que Matlab le envía un byte indicando la operación que el microcontrolador debe realizar. Este byte se almacena en la variable *data*. En caso de que el valor sea 3, se llama a la función *data_acquisition()*, que es

la encargada de recoger la información del sensor ToF y mandarla a Matlab. Sin embargo, antes de ello se lee la información medida por la IMU correspondiente a la posición angular relativa con respecto al eje x (eje logitudinal del vehículo). Este valor se corresponde con la dirección del *rover*, de forma que se almacena en la variable *current_angle*.

Una vez se ha llamado a la función *data_acquisition()*, ésta recoge las medidas producidas por el sensor de distancias ToF y envía cada dato de la matriz de distancias y de la matriz de estado (*status*) a Matlab. Cabe mencionar que los índices de los bucles *for* son decrecientes ya que, como se ha visto en la *Sección 3.3*, la imagen generada por el sensor ToF está invertida con respecto al eje longitudinal y la colocación del sensor en el *rover* hace que también esté invertida con respecto al eje vertical. Una vez enviadas las dos matrices de 64 celdas, se envía también la posición angular actual del vehículo.

Es necesario mencionar que, por simplicidad, se han omitido algunas partes del código correspondiente al bucle de ejecución del sistema que se corresponden con otros módulos *software*. A continuación se van a ir introduciendo paulatinamente dichas porciones de código para facilitar su comprensión.

5.5.3. Procesado de datos

El bloque *software* del procesado de datos corresponde con una parte de la implementación de un *script* en Matlab. Esta porción de código se muestra en el *Código 5.4*.

```
%% Create rover data
rover.m = eye(8);    % Distance 8x8 matrix
rover.status = eye(8);
rover.angle = 0;     % (Current angle measured by IMU)

%% Connect Matlab with ESP32 to send PWM duty
bt = bluetooth("ESP32BT",1, "Timeout", 30);

%% Process data from ToF sensor and IMU

% Initialize loop
i = 1;
write(bt,1);
while(1 )
    % Read from ToF Sensor
    write(bt,3)      % Tells ESP32 that Matlab is ready to
    read data
    for row = 1:8
        for col = 1:8
            byte_ToF_1 = read(bt,1);
            byte_ToF_2 = read(bt,1);

            byte_status_1 = read(bt,1);
        Procesado_datos    byte_status_2 = read(bt,1);

            m(row,col) = byte_ToF_1*256 + byte_ToF_2;
            rover.status(row,col) = byte_status_1*256 +
```

```
    byte_status_2;
27          end
28      end
29
30      % Read from IMU
31      byte_imu_1 = read(bt,1);
32      byte_imu_2 = read(bt,1);
33
34      % Fix incorrect data values
35      for row = 1:8
36          for col = 1:8
37              if(rover.status(row,col) ~= 5)
38                  m(row,col) = 800;
39              end
40          end
41      end
42
43      % Set rover variables
44      rover.m = m;
45      rover.angle= byte_imu_1*256 + byte_imu_2;
46
47      % the script continues....
```

5.4: Procesado de datos en Matlab

Como se observa, inicialmente se crea una estructura que representa al *rover*. Ésta dispone de tres variables: la matriz *m* (correspondiente a la matriz de distancias ToF), la matriz *status* (correspondiente a la matriz de estado de que indica la validez de las medidas de distancia) y la variable *angle*, que hace referencia al valor actual de la orientación angular del vehículo. Es equivalente a la variable *current_angle* utilizada en el código Arduino anterior. Una vez inicializada la estructura del *rover*, se establece la conexción Bluetooth con el ESP32 y se envía un uno antes de entrar en el bucle de ejecución para indicar el inicio del funcionamiento del vehículo. Dentro del bucle de ejecución se leen los datos de la matriz de distancias y la matriz de estado junto con el ángulo de la orientación del vehículo. Posteriormente se corrigen los valores de aquellas medidas cuyo estado asociado es distinto de 5. Esto implica que la medida no se ha realizado de forma correcta como se indica en el manual de usuario del sensor de distancias ToF y se puede ver en la *Figura 3.7*.

En caso de que la medida no sea correcta, se ha decidido obviar dicha medida asignándole un valor de 800 mm, que es el rango de medida fiable que tiene el sensor en las condiciones de luminosidad sobre las que ha sido probado y operado como se explica posteriormente en la *Sección 6.1*. Por tanto, se considera que dicha medida errónea no aporta información al sistema y se considera que no hay ningún obstáculo en ese punto. La alta tasa de muestreo hace que considerar como despreciable esta medida no provoque errores de detección de obstáculos presentes que se consideren inexistentes. Una vez se han corregido dichas medidas, se asigna la matriz obtenida a la matriz *m* de la estructura del *rover*. Una vez más, se ha cortado aquí la implementación de este módulo para facilitar el entendimiento del código y comprender las partes de código que corresponden a cada módulo de *software*.

5.5.4. Cálculo de la maniobra

Para el cálculo de la maniobra que el *rover* debe realizar en cada iteración se ha creado una función en Matlab llamada *calculateAngle* que recibe como parámetro la estructura de datos que representa al *rover* y que devuelve dos datos: el ángulo hacia el que tiene que rotar el *rover* (*desired_angle*) y una variable booleana *make_turn*, que indican si el vehículo debe realizar un giro o, en caso de devolver *false*, significa que el *rover* puede seguir avanzando en línea recta. La implementación de esta función se muestra en el *Código 5.5*.

```
1 % Function that calculates the turn angle
2 function [desired_angle, make_turn] = calculateAngle (
    rover)
3     % Angle that rover must turn if needed
4     turn_angle = 25;
5      % Set limit distance to make turn
6     limit = 550;
7     %% Check if rover can move forward
8     min_forward = min(min(rover.m(:, 3:6)));
9
10    % Move forward if possible
11    if(min_forward >= limit)
12        desired_angle = rover.angle;
13        make_turn = false;
14        return;
15    end
16
17    %% Check left and right (columns 1,2 and 7,8)
18    min_cols(1) = min(rover.m(:,1));   max_cols(1) = max(
    rover.m(:,1));
19    min_cols(2) = min(rover.m(:,2));   max_cols(2) = max(
    rover.m(:,2));
20    min_cols(3) = min(rover.m(:,7));   max_cols(3) = max(
    rover.m(:,7));
21    min_cols(4) = min(rover.m(:,8));   max_cols(4) = max(
    rover.m(:,8));
22
23    % Find best direction to turn if both sides are
    possible
24    max_distance = 0;
25    col = 0;
26    for i = 1:4
27        if( (min_cols(i) >= limit || max_cols(i) >= 800 )
    && max_cols(i) > max_distance)
28            max_distance = max_cols(i);
29            col = i;
30        end
31    end
32
33    % Turn left
```

```
34      if(col <= 2 && col > 0)
35          make_turn = true;
36          if(rover.angle >= 270)    % was pointing to left
37              desired_angle = rover.angle - turn_angle;
38              if(desired_angle < 270)
39                  desired_angle = 270;
40              end
41          else                      % was pointint to right
42              desired_angle = rover.angle - turn_angle;
43              if(desired_angle < 0)
44                  desired_angle = 360 + desired_angle;
45              end
46          end
47          return;
48      % Turn right
49      elseif (col >= 3 && col > 0 )
50          make_turn = true;
51          if(rover.angle <= 90)    % was pointint to right
52              desired_angle = rover.angle + turn_angle;
53              if(desired_angle > 90)
54                  desired_angle = 90;
55              end
56          else                      % was pointint to left
57              desired_angle = rover.angle + turn_angle;
58              if(desired_angle > 360)
59                  desired_angle = desired_angle - 360;
60              end
61          end
62          return;
63      end
64
65      %% All directions are blocked
66      disp(" All directions blocked");
67      make_turn = true;
68      if(rover.angle <= 90)        % was pointing to right
69          desired_angle = rover.angle - turn_angle;
70          if(desired_angle < 0)
71              desired_angle = 360 + desired_angle;
72          end
73      elseif(rover.angle >= 270)  % was pointing to right
74          desired_angle = rover.angle + turn_angle;
75          if(desired_angle > 360)
76              desired_angle = desired_angle - 360;
77          end
78      end
79      return;
80
81      %% Other cases
82      disp("Error. Case not expected");
83
84  end
```

5.5: Cálculo de la maniobra en Matlab

Inicialmente, se define el ángulo de giro que, después de realizar varias pruebas se fijó el valor a 25º por ser el ángulo de giro que mejor funcionaba con la distancia límite de 550 mm a partir de la cual se debe girar para evitar un obstáculo. Esto permite al *rover* evitar un obstáculo de 256 mm de ancho a cada lado de su eje con un solo giro como se muestra en la *Figura 5.7*.

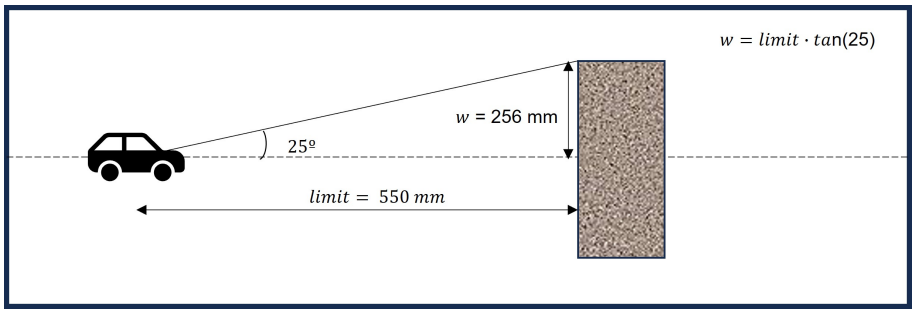


Figura 5.7: Anchura máxima del obstáculo que se evita con un solo giro
fuente: elaboración propia

Antes de seguir con el análisis del *Código 5.5*, es necesario mencionar que la matriz 8x8 de distancias ToF se ha dividido en tres secciones. Cómo se puede observar en la *Figura 5.8*. La primera sección agrupa las dos primeras columnas de la matriz y contiene los datos que permitirán determinar si se puede realizar un giro hacia la izquierda. Similarmente, las dos últimas columnas de la matriz contienen los datos de distancias que permitirán determinar si se puede rotar hacia la derecha. El resto de columnas centrales se corresponden con los datos que permitirán decidir si el *rover* puede seguir avanzando en linea recta.

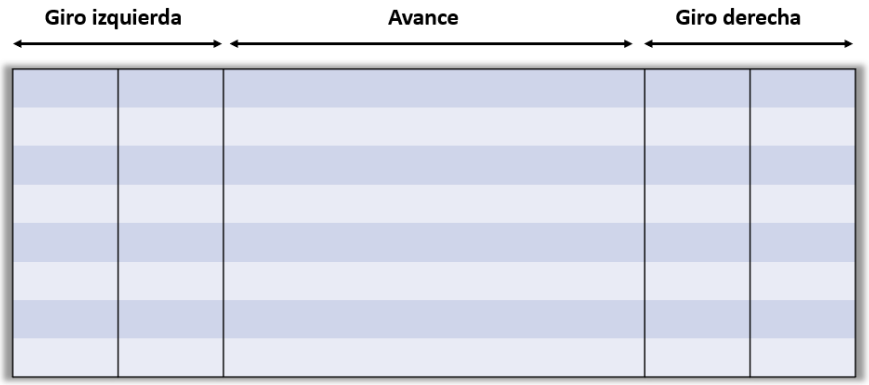


Figura 5.8: División de la matriz de distancias por direcciones
fuente: elaboración propia

Siguiendo con el *Código 5.5*, se calcula el valor mínimo de distancias en la zona central de la matriz y se comprueba si dicho valor es mayor que el límite mínimo de distancia hasta el que se puede seguir avanzando. En caso afirmativo, se puede seguir avanzando en línea recta, por lo que el ángulo deseado será el mismo que tenía el vehículo en la iteración anterior y la variable booleana *make_turn* se pone a *false*.

En caso de no poder seguir avanzando en línea recta, se calculan los valores mínimos y máximos de cada una de las columnas correspondientes a los giros a izquierda y derecha. Posteriormente se analizan dichos datos y, en caso de que una o más columnas tengan un valor mínimo por encima de la distancia límite o su máximo valor supere los 800 mm, se selecciona la columna que cumple estas condiciones y cuyo valor máximo es el mayor de todas las columnas. Esto permite seleccionar la dirección cuya distancia máxima a los objetos sea mayor en caso de que ambas direcciones fuesen válidas. De esta forma, el *rover* elegirá siempre la dirección en la que los obstáculos están más lejos. Una vez seleccionada la dirección se configura *make_turn* a *true* y se calcula el ángulo deseado, corrigiendo los valores obtenidos si fuese necesario para que el valor esté siempre en el rango [0,90] U [270,360).

Finalmente, en caso de que todas las direcciones estén bloqueadas por objetos cercanos, se realiza un giro hacia la dirección en la que está el origen del eje longitudinal del *rover*. De esta forma, si el vehículo se encuentra apuntando hacia la derecha (ha realizado virajes anteriores que le han dejado en esa posición), realizará un giro hacia la izquierda y viceversa. De esta forma se consigue que el *rover* tienda a volver a la orientación inicial en caso de verse bloqueado y en cada giro pueda volver a evaluar el espacio tridimensional enfrente y encontrar una salida.

5.5.5. Comunicación de la maniobra al microcontrolador

Posteriormente al cálculo de la maniobra, se debe comunicar desde Matlab a Arduino la maniobra determinada para que éste pueda mandar las órdenes a los motores. Para ello, se continúa el *script* mostrado en el *Código 5.4*, que se incluye completo en el *Código 5.6*.

```
clear; clc;
%% Create rover data
rover.m = eye(8);    % Distance 8x8 matrix
rover.status = eye(8);
rover.angle = 0;     % (Current angle measured by IMU)

%% Connect Matlab with ESP32 to send PWM duty
bt = bluetooth("ESP32BT",1, "Timeout", 30);

%% Recieve ToF matrix and IMU data from ESP32
% Initialize data for PID
prevT = tic;
error_integral = 0;
last_error = 0;
current_angle = 0;
PID_working = false;

% Initialize loop
i = 1;
write(bt,1);
while(1 )
    % Read from ToF Sensor
    write(bt,3)      % Tells ESP32 that Matlab is ready to
    read data
```

```
24      for row = 1:8
25          for col = 1:8
26              byte_ToF_1 = read(bt,1);
27              byte_ToF_2 = read(bt,1);
28
29              byte_status_1 = read(bt,1);
30              byte_status_2 = read(bt,1);
31
32              m(row,col) = byte_ToF_1*256 + byte_ToF_2;
33              rover.status(row,col) = byte_status_1*256 +
    byte_status_2;
34          end
35      end
36
37      % Read from IMU
38      byte_imu_1 = read(bt,1);
39      byte_imu_2 = read(bt,1);
40
41      % Fix incorrect data values
42      for row = 1:8
43          for col = 1:8
44              % Correct invalid values
45              if(rover.status(row,col) ~= 5)
46                  m(row,col) = 800;
47              end
48          end
49      end
50
51      rover.m = m;      % ToF Data
52      rover.m
53      % rover.status
54
55      current_angle = byte_imu_1*256 + byte_imu_2;
56      rover.angle = current_angle;
57
58      % Calculate desired angle (only if PID not working)
59      % [desired_angle, turn_angle, make_turn] =
    calculateAngle6(rover);
60      [desired_angle, make_turn] = calculateAngle7(rover);
61
62      disp(current_angle + "  " + desired_angle);
63
64      % Adjust PID for new turn
65      if(make_turn)
66           write(bt, 4);   % Tells rover that needs to turn
67           write(bt,fix(desired_angle/256));
68           write(bt,mod(desired_angle,256));
69           flag = read(bt,1);
70           % break;
71      end
72
```

```
73      % Update variables
74      i = i + 1;
75
76  end
```

5.6: Comunicación de la maniobra a Arduino

Como se puede observar, una vez procesados los datos y calculada la maniobra se le manda a Matlab un byte que vale 4 en caso de que haya que realizar un giro y se le envía el ángulo deseado. En caso de no mandarle nada, Arduino asume que el *rover* puede seguir operando en modo avance en línea recta. Cabe mencionar que por motivos de sincronización entre ambos lenguajes, Matlab se debe poner a la espera de que Arduino realice la operación completa de giro antes de volver a entrar en la siguiente iteración del bucle de ejecución. Para ello se incluye la línea *finished = read(bt,1);*, ya que se ha configurado un *timeout* para la operación *read()* de 30 segundos, siendo este muy superior a los 2-3 segundos máximos que tarda el vehículo en realizar el giro. Una vez ha realizado el giro, Arduino manda un byte a Matlab y ambos proceden a ejecutar la próxima iteración del bucle de ejecución.

5.5.6. Operación algoritmos de control PID

Los dos últimos módulos *software* restantes de la *Figura 5.5* son aquellos que utilizan un PID para avanzar en línea recta o para realizar un giro. Se analizará primero el módulo *software* que permite realizar un giro. Éste se muestra en el *Código 5.7*.

```
1  /* Read desired angle from Matlab and adjust PID gain values */
2  void adjustPID(float *ki, float *kp, float *kd, float *desired_angle,
       float current_angle){
3    float delta_angle;  // Delta angle that the rover must turn
4    float byte1;        // Desired angle
5    float byte2;        // Desired angle
6    delay(500);
7
8    // Read desired angle from Matlab
9    if(SerialBT.available()){
10     byte1 = (float) SerialBT.read();
11     byte2 = (float) SerialBT.read();
12     *desired_angle = byte1*256 + byte2;
13
14     // Find delta angle (correction for differences > 180  )
15     if( fabs(*desired_angle - current_angle) <= 180 ){
16       delta_angle = fabs(*desired_angle - current_angle);
17     }else{
18       delta_angle = 360 - fabs(*desired_angle - current_angle);
19     }
20
21     // Adjust gain values depending on delta angle
22     if( delta_angle >= 15 && delta_angle <= 80){
23       *kp = 7;
24       *ki = -(4.0/15.0)*delta_angle + 25;
25     }else if(delta_angle < 15){
26       *kp = 10;
27       *ki = 22;
28     }else{
29       *kp = 7;
```

```
30        *ki = 2;
31      }
32      *kd = 0;
33      delay(500);
34    }else{
35      Serial.println("Desired angle was not read from Matlab");
36    }
37
38  }
39
40  /* Resets PID variables to start making a turn */
41  void reset_PID_variables(){
42    PID_working = true;
43    prevT = micros();   //Re-start time to avoid delays due to adjustPID
        () function
44    eintegral = 0;      // Re-set the integral error to 0
45    ederivator = 0;     // Re-set the derivator error to 0
46    data = 0;
47  }
48
49  /* Operates the PID to make a turn */
50  void operate_PID(){
51    // Compute error between desired and current angle
52    if(desired_angle > current_angle ){
53      if( fabs(desired_angle - current_angle) <= 180){
54        error = fabs(desired_angle - current_angle);          // CW turn (
        right)
55      }else{
56        error = fabs(desired_angle - current_angle) - 360;  // CCW turn (
        left)
57      }
58    }else{
59      if( fabs(desired_angle - current_angle) <= 180){
60        error = -fabs(desired_angle - current_angle);         // CCW turn (
        left)
61      }else{
62        error = 360 - fabs(desired_angle - current_angle);    // CW turn (
        right)
63      }
64    }
65
66    // If it is 1st iteration, update prevT to avoid delays
67    if(prevT == 0){
68      prevT = micros();
69    }
70
71    // Compute time elapsed between two iterations
72    currT = micros();
73    deltaT = ((float) (currT-prevT))/1.0e6;
74
75    //Compute integral error
76    eintegral = eintegral + error*deltaT;
77
78    // Compute derivator error
79    ederivator = (error - last_error)/deltaT;
80
81    // Update variables
82    prevT = currT;
83
84    // Compute control signal u
85    u = kp*error + ki*eintegral + kd*ederivator;
86
87    // Set motor duty
```

```
88    dutyCycle = (int) fabs(u);
89    if(dutyCycle > 250){
90      dutyCycle = 250;      // Maximum Duty = 250
91    }
92
93    // Operate motors to make turn
94    make_turn();
95  }
96
97  /* Operate motors to make turn */
98  void make_turn(){
99    // Turn CW (right)
100   if( u >= 0){
101     //Serial.println("Turn right");
102     digitalWrite(motor1Pin1, HIGH);
103     digitalWrite(motor1Pin2, LOW);
104     digitalWrite(motor2Pin1, LOW);
105     digitalWrite(motor2Pin2, HIGH);
106
107     ledcWrite(pwmChannel1, dutyCycle);
108     ledcWrite(pwmChannel2, dutyCycle );
109   }
110
111   // Turn CCW (left)
112   if( u < 0){
113     //Serial.println("Turn left");
114     digitalWrite(motor1Pin1, LOW);
115     digitalWrite(motor1Pin2, HIGH);
116     digitalWrite(motor2Pin1, HIGH);
117     digitalWrite(motor2Pin2, LOW);
118
119     ledcWrite(pwmChannel1, dutyCycle);
120     ledcWrite(pwmChannel2, dutyCycle );
121   }
122   print_data();
123 }
124
125 /* Stop rover */
126 void stop(){
127   // Stop
128   Serial.println("Stop");
129   digitalWrite(motor1Pin1, LOW);
130   digitalWrite(motor1Pin2, LOW);
131   digitalWrite(motor2Pin1, LOW);
132   digitalWrite(motor2Pin2, LOW);
133 }
134
135
136 /* System execution loop */
137 void loop() {
138   // Matlab sends flag = 1 to start the operation of the system
139   if(flag == 0 & SerialBT.available()){
140     flag = SerialBT.read();
141   }
142
143   // Control algorithm execution
144   if(flag == 1){
145
146     // Matlab sends one byte indicating the operation to execute
147     if(SerialBT.available()){
148       data = SerialBT.read();
149       Serial.println(data);
150     }
```

```
151
152     // Read data from IMU
153     bno.getEvent(&event);
154     current_angle = event.orientation.x;
155
156     // Execute operation indicated by Matlab
157     if(data == 3){     // Acquire sensor data and send it to Matlab
158       data_acquisition();
159
160     }else if(data == 4) {    // Adjust PID to start making a turn
161       stop();
162       adjustPID(&kp, &ki, &kd, &desired_angle, current_angle);
163       reset_PID_variables();
164     }
165
166     // Operate PID
167     while( PID_working ){
168       operate_PID();
169       last_error = error;
170
171       // Read data from IMU
172       bno.getEvent(&event);
173       current_angle = event.orientation.x;
174
175       // Condition to finish PID operation
176       if( fabs(desired_angle - current_angle) < 3 || dutyCycle < 100){
177         Serial.println("PID finished");
178         PID_working = false;
179         stop();
180         SerialBT.write(1);     // Tell matlab that turn is finished
181         break;
182       }
183     }
184
185     delay(1);
186   }
187 }
```

5.7: Controlador PID para realizar un giro

Una vez Arduino ha recibido el byte indicando que que se debe realizar un giro, detiene el vehículo, ajusta las variables del PID y hace un *reset* de las variables que influyen en la operación del PID para que queden inicializadas a cero.

En la función *adjust_PID()* se leen los dos bytes correspondientes al ángulo final que ha sido obtenido en el módulo de cálculo de la maniobra de Matlab. Una vez se tiene el ángulo, se aplica un algoritmo que ajusta los valores de las ganancias del PID dependiendo del ángulo que se desea realizar, ya que durante el desarrollo del trabajo se observó que dependiendo de la magnitud del ángulo de giro, el PID funcionaba mejor con distintas ganancias. En el código de la versión final el ángulo girado es siempre de 25º, por lo que este código sería innecesario ya que las ganancias del PID serían siempre las mismas. Sin embargo, a lo largo del desarrollo del proyecto se probó la opción de realizar giros de diferente magnitud en función de los datos de la matriz de distancias. Ambas opciones resultaron en comportamientos similares y satisfactorios del vehículo, por lo que se ha dejado este algoritmo por si se desea modificar el algoritmo de cálculo de maniobras para que genere giros de diferente magnitud. Los valores de las ganancias se midieron y comprobaron experimentalmente hasta que se obtuvo una relación que optimizaba la precisión y rapidez del giro. Los valores óptimos se muestran en la *Tabla 5.1*. Como se puede observar, la ganancia derivativa es siempre nula ya que las pruebas que se hicieron

indicaron que el giro se realiza de forma precisa y más rápida eliminando esta ganancia. Por tanto, en realidad se tiene un PI en lugar de un PID, ya que la parte derivativa no influye.

Magnitud del giro	**Kp**	**Ki**	**Kd**
$15^o \leq \Delta\theta \leq 80^o$	7	$-4/15 * \Delta\theta + 25$	0
$\Delta\theta < 15^o$	10	22	0
$\Delta\theta \geq 80^o$	2	5	0

Tabla 5.1: Ganancias del PID de giro

Siguiendo con el análisis del *Código 5.7*, una vez se ha ajustado el PID y se han inicializado a cero de nuevo las variables involucradas, se indica que el PID está funcionando y se entra en un bucle *while*, que llama a la función *operate_PID()* y del que sólamente se sale si la diferencia entre el ángulo deseado y el actual es menor de 3º (se asumen esos 3º de error) o el *dutyCycle* de los motores es menor que 100, ya que para ese valor los motores están parados y, por tanto, el PID ha terminado su función. En ese caso, se vuelve a parar el vehículo y se indica a Matlab que se ha terminado la operación del PID. La función *operate_PID()* calcula primero la diferencia entre el ángulo actual y el ángulo deseado para posteriormente actualizar los valores de los errores proporcional, integral y derivativo. Una vez calculada la acción de control (con un valor máximo de 250) a partir de los errores y las ganancias correspondientes se llama a la función *make_turn()*, que es la encargada de enviar la señal PWM correspondiente a los motores dependiendo de si se quiere girar hacia la derecha (acción de control positiva) o hacia la izquierda (acción de control negativa).

El último módulo *software* es el que se encarga del PID de avance. Este módulo se asegura que el *rover* no se desvía hacia ninguna dirección debido a las desigualdades de potencia que los motores tienen ni a las desigualdades en las tensiones de las cadenas de ambas orugas. Las ganancias proporcional e integral son 7 y 10 respectivamente. El funcionamiento es similar al PID de giro como se puede observar en el *Código 5.8*.

```
1  /* Operate PID while moving forward */
2  void PID_forward(){
3    // Compute error between desired and current angle
4    if(desired_angle > current_angle ){
5      if( fabs(desired_angle - current_angle) <= 180){
6        error = fabs(desired_angle - current_angle);          // CW turn (
       right)
7      }else{
8        error = fabs(desired_angle - current_angle) - 360;  // CCW turn (
       left)
9      }
10   }else{
11     if( fabs(desired_angle - current_angle) <= 180){
12       error = -fabs(desired_angle - current_angle);         // CCW turn (
       left)
13     }else{
14       error = 360 - fabs(desired_angle - current_angle);    // CW turn (
       right)
15     }
16   }
17
18   // If it is 1st iteration, update prevT to avoid delays
19   if(prevT == 0){
20     prevT = micros();
```

```
21    }
22
23    // Compute time elapsed between two iterations
24    currT = micros();
25    deltaT = ((float) (currT-prevT))/1.0e6;
26
27    //Compute integral error
28    eintegral = eintegral + error*deltaT;
29
30    // Compute derivator error
31    ederivator = (error - last_error)/deltaT;
32
33    // Update variables
34    prevT = currT;
35
36    // Compute control signal u
37    u = kp_forward*error + ki_forward*eintegral + kd_forward*ederivator;
38
39    if( u <= 0){
40      dutyCycle_left = dutyCycle_forward;
41      dutyCycle_right = dutyCycle_forward + (int) fabs(u);
42      if(dutyCycle_right > 250){
43        dutyCycle_right = 250;
44      }
45    }else{
46      dutyCycle_right = dutyCycle_forward;
47      dutyCycle_left = dutyCycle_forward + (int) fabs(u);
48      if(dutyCycle_left > 250){
49        dutyCycle_left = 250;
50      }
51    }
52
53  }
54
55  /* Move forward */
56  void move_forward(){
57    digitalWrite(motor1Pin1, HIGH);
58    digitalWrite(motor1Pin2, LOW);
59    digitalWrite(motor2Pin1, HIGH);
60    digitalWrite(motor2Pin2, LOW);
61
62    prevT = micros();    // Re-start time to avoid delays due to adjustPID
        () function
63    eintegral = 0;       // Re-set the integral error to 0
64    ederivator = 0;      // Re-set the derivator error to 0
65
66    PID_forward();       // Calculate control action
67    ledcWrite(pwmChannel1, dutyCycle_left);
68    ledcWrite(pwmChannel2, dutyCycle_right);
69    //print_data();
70  }
71
72  /* Print data on serial monitor to check values */
73  void print_data(){
74    Serial.print(desired_angle);
75    Serial.print(" ");
76    Serial.print(current_angle);
77    Serial.print(" ");
78    Serial.print(kp);
79    Serial.print(" ");
80    Serial.print(ki);
81    Serial.print(" ");
82    Serial.print(error);
```

```
  Serial.print(" ");
  Serial.print(eintegral);
  Serial.print(" ");
  Serial.print(ederivator);
  Serial.print(" ");
  Serial.print(dutyCycle);
  Serial.print(" ");
  Serial.print(dutyCycle_left);
  Serial.print(" ");
  Serial.print(dutyCycle_right);
  Serial.println();
  delay(1);
}

/* System execution loop */
void loop() {
  // Matlab sends flag = 1 to start the operation of the system
  if(flag == 0 & SerialBT.available()){
    flag = SerialBT.read();
  }

  // Control algorithm execution
  if(flag == 1){

    // Matlab sends one byte indicating the operation to execute
    if(SerialBT.available()){
      data = SerialBT.read();
      Serial.println(data);
    }

    // Read data from IMU
    bno.getEvent(&event);
    current_angle = event.orientation.x;

    // Execute operation indicated by Matlab
    if(data == 3){     // Acquire sensor data and send it to Matlab
      data_acquisition();

    }else if(data == 4) {    // Adjust PID to start making a turn
      stop();
      adjustPID(&kp, &ki, &kd, &desired_angle, current_angle);
      reset_PID_variables();
    }

    while( PID_working ){
      operate_PID();
      last_error = error;

      // Read data from IMU
      bno.getEvent(&event);
      current_angle = event.orientation.x;

      // Condition to finish PID operation
      if( fabs(desired_angle - current_angle) < 3 || dutyCycle < 100){
        Serial.println("PID finished");
        PID_working = false;
        stop();
        SerialBT.write(1);     // Tell matlab that turn is finished
        break;
      }
    }

    // Move forward if no turn is going on
```

```
146       if(PID_working == false){
147         move_forward();
148       }
149
150       delay(1);
151
152     }
153 }
```

5.8: Controlador PID para avanzar en línea recta

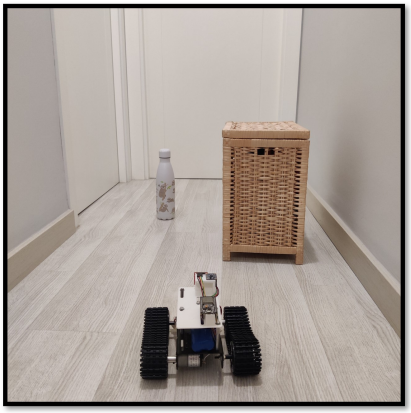
En esta sección se ha hecho un desglose de los diferentes módulos *software* que componen el sistema de guiado del vehículo y se ha analizado cada uno de forma independiente. El código completo del sistema se encuentra en el *Anexo A*.

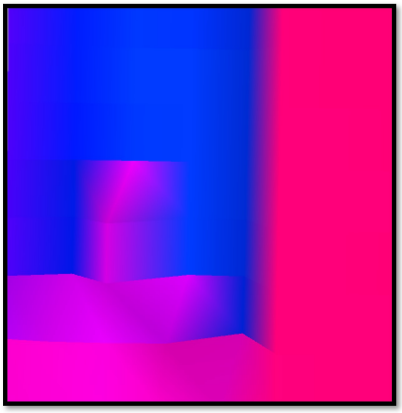
6. Resultados

Este Trabajo de Fin de Grado se ha centrado en el diseño y desarrollo de una primera aproximación a un sistema de guiado y control autónomo novedoso de un vehículo terrestre no tripulado que simula y simplifica el comportamiento de un *rover* dedicado a la exploración espacial. En esta sección se recogen y analizan los resultados obtenidos durante el desarrollo del presente trabajo.

6.1. Lectura del sensor ToF

Durante la creación del presente proyecto se consiguió de forma satisfactoria la configuración y recopilación de datos del sensor de distancias ToF VL53L5CX. Previamente a la implementación del algoritmo de control completo se realizaron diversas pruebas para comprobar el buen funcionamiento del sensor y la correlación que existía entre la matriz de datos generada y la distribución de obstáculos real frente al sensor. En la *Figura 6.1* se muestra un ejemplo de las pruebas llevadas a cabo y que confirman el buen funcionamiento del sensor, que es capaz de mapear el espacio tridimensional frente a él de forma correcta. La imagen superior izquierda muestra una fotografía de la distribución espacial real de la escena. A su derecha se muestra un mapeado en código RGB que se ha creado a partir de la matriz de distancias generada por el sensor ToF (mostrada en la parte inferior de la figura). Esta codificación de colores que permite comparar la escena real y la generada por el sensor de forma sencilla y visual se ha realizado utilizando el *script* desarrollado por la empresa SparkFun, responsables de la creación y comercialización de la placa de desarrollo SparkFun Qwiic ToF Imager - VL53L5CX.





1067	1404	1473	1484	1435	305	310	305
1097	1403	1471	1479	1456	317	310	310
1128	1409	1477	1494	1462	318	311	308
1143	1400	1496	1495	1476	322	314	310
1140	1405	714	1491	1466	318	315	312
1123	1374	714	1496	1466	322	319	316
779	744	715	748	1482	320	318	315
516	577	599	507	568	323	321	321

Figura 6.1: Correlación de los datos generados por el sensor ToF y la escena real
fuente: elaboración propia

Como se puede observar en las *Figura 6.1*, existe una correlación positiva entre la escena real y los datos generados por el sensor ToF. A la derecha de la imagen se detecta el baúl de mimbre en un tono rojizo más fuerte, que indica que dicho objeto está más cerca que la cantimplora, que tiene un color rosado más suave. El resto de la escena está en un tono azul, indicando que existe una mayor distancia hasta la pared izquierda y la puerta del fondo de la imagen. La línea rojiza horizontal en la parte inferior del mapeado se debe a la detección del suelo por parte del sensor ya que, como se ha mencionado en la *Sección 3.3*, el sensor tiene un campo de visión con un ángulo vertical de 45º, que le hace detectar el suelo a cierta distancia. Por tanto, se puede afirmar que se ha configurado y utilizado de forma satisfactoria el novedoso sensor ToF VL53L5CX.

Cabe mencionar que el desarrollo de estas pruebas de correlación ha permitido establecer el límite de distancia a partir del cual las medidas del sensor no son fiables y coherentes con la realidad. En las condiciones de luz ambiental probadas este límite se encuentra en los 800 mm. Por ello, como se ha comentado en la *Sección 5.5.3* se ha asignado este valor de distancia a las medidas cuyo valor de estado asociado indicaba que no era una medida válida.

6.2. Validación experimental del PID

Las maniobras de giro realizadas por el *rover* se han realizado utilizando técnicas de control PID como se explica en la *Sección 5.4*. Las ganancias de este algoritmo se han ajustado en base a resultados experimentales como se presentó en la *Tabla 5.1*. Previamente a la obtención de estas ganancias óptimas para cada rango de giro se realizaron diversas pruebas en las que se ordenaba al vehículo girar un cierto ángulo. Una vez el algoritmo PID realizaba la acción de control necesaria y el *rover* se estabilizaba, se tomaban datos de la posición angular inicial y final. Este proceso se realizó con diversas magnitudes de

ángulos y diversas ganancias del PID con el objetivo de hallar las ganancias óptimas para cada magnitud de giro. En la *Tabla 6.1* se muestran los datos recogidos en este tipo de pruebas para las ganancias óptimas. La primera columna se corresponde con la magnitud del ángulo que el vehículo debía girar. La segunda columna presenta el ángulo girado indicado por la IMU. La tercera columna presenta el ángulo girado realmente. Este ángulo se ha medido a mano, colocando unas marcas en el suelo que han permitido medirlo de forma manual, evitando así el posible pequeño error que la IMU puede tener. Finalmente, en la cuarta columna se indica el error cometido, tomando como valor final la media entre el indicado por la IMU y el medido manualmente.

Como se puede observar, el error para valores de giro inferiores a 30^{o} es menor a 2^{o} y, para valores inferiores a 90^{o} es siempre menor de 3.3^{o}. Estos errores son asumibles para el grado de precisión buscado en el desarrollo de este trabajo. Debido a que en la implementación final el ángulo girado es siempre de 25^{o}, se asume que se estará cometiendo un error de unos 1.6^{o} en cada giro. Este valor es muy pequeño y no afecta al correcto funcionamiento del vehículo ya que está constantemente realizando mediciones y, en caso de necesitar un giro mayor, generará otra señal de giro en un intervalo de tiempo muy pequeño. Para valores superiores a 90^{o} el error aumenta, indicando que sería necesario un ajusta algo más preciso. Sin embargo, el funcionamiento de este sistema de guiado está diseñado para funcionar de forma óptima girando ángulos menores a 60^{o}, por lo que este error no afecta al sistema implementado.

$\Delta\theta_{teórico}$	$\Delta\theta_{IMU}$	$\Delta\theta_{experimental}$	$Error$
2 º	2.87 º	2.8 º	0.8 º
5 º	6.0 º	6.1 º	1.1 º
10 º	9.2 º	9.2 º	0.8 º
15 º	14.5 º	14.3 º	0.6 º
20 º	21.3 º	21.5 º	1.4 º
25 º	27.0 º	26.2 º	1.6 º
30 º	32.3 º	31.6 º	1.9 º
60 º	57.6 º	58.3 º	2.0 º
90 º	93.4 º	93.2 º	3.3 º
135 º	139.1 º	138.7 º	3.9 º
180 º	188.2 º	187.9 º	8.1 º

Tabla 6.1: Verificación del funcionamiento del PID de giro

Cabe mencionar que se ha considerado como finalizado el proceso de control PID en el primer momento en el que los motores se paran. En realidad, si se dejase ejecutar el algoritmo unos segundos posteriores a la detención de los motores, el error integral aumentaría y se volvería a generar una señal PWM lo suficientemente grande como para que los motores se muevan y el vehículo se acerque más al ángulo deseado. Sin embargo, en este proyecto se ha buscado el diseño de un PID efectivo y rápido, que permita una navegación fluida sin que el *rover* se detenga unos segundos para finalizar el giro en caso de que el error sea pequeño.

6.3. Validación de la navegación autónoma

Los distintos módulos software del sistema de navegación y guiado autónomo imple-

mentado han sido probados y operados de forma independiente. El último paso en la validación del sistema es la creación de diversas pruebas de navegación que permitan la comprobación del funcionamiento del sistema completo. Para ello se han diseñado diversos circuitos con diferentes configuraciones de obstáculos.

Inicialmente se diseñaron circuitos cerrados en los que solo existe un camino por el que el *rover* debe circular para superar la prueba. Estos circuitos están compuestos por cajas que simulan las paredes de un canal curvilíneo por el que el vehículo debe circular sin tocar estas cajas.

Posteriormente se diseñaron circuitos semi-abiertos en los que se colocaron obstáculos anchos que el *rover* debía evitar realizando una trayectoria sinusoidal amplia. De esta forma se comprobó el funcionamiento del sistema en un espacio más amplio en el que sigue existiendo solamente un camino para superar el circuito.

Finalmente, se han diseñado varios circuitos con obstáculos de menor tamaño distribuidos de forma aleatoria. En estos circuitos existen varias trayectorias posibles que el *rover* puede seguir para completar el circuito completo.

El sistema de evasión de obstáculos ha sido capaz de superar de forma satisfactoria todos los circuitos planteados, desde los más simples con trayectorias cerradas hasta los más complejos con múltiples trayectorias. En estos últimos circuitos, el *rover* ha demostrado ser capaz de recorrer distintos caminos para superarlos debido a pequeñas variaciones de la posición y orientación inicial del vehículo. En el siguiente enlace a Drive se han subido varios vídeos que demuestran el buen funcionamiento del sistema en los diversos circuitos mencionados.

Repositorio con vídeos del rover superando diversos circuitos de obstáculos

6.4. Conclusiones y líneas futuras

Gracias al desarrollo del presente trabajo se ha demostrado que es posible utilizar sensores de distancias ToF para el guiado de un *rover*. Estos sensores generan matrices de datos que mapean la realidad tridimensional del entorno en el que se encuentra el vehículo autónomo. Además, aportan una mayor cantidad de información al sistema de navegación ya que aumenta el número de puntos espaciales de los cuales se tiene información, enriqueciendo así el conocimiento del entorno del que dispone el algoritmo de control que determina la siguiente maniobra que debe llevarse a cabo.

El correcto funcionamiento de este sistema de navegación y guiado ha permitido al prototipo construido superar diversos circuitos con obstáculos tomando decisiones de forma autónoma en cada momento, eligiendo siempre la maniobra óptima para superar dichos obstáculos. Todas estas decisiones vienen determinadas simplemente por los datos del sensor ToF. Esto indica que los datos que genera este sensor son lo suficientemente fiables y precisos como para implementar un algoritmo de estas características, al menos en aplicaciones de este estilo tales como la robótica recreacional. Sin embargo, para poder considerar este tipo de sensores ToF como alternativas reales a los sensores de percepción utilizados actualmente en *rovers* espaciales, la tecnología de éstos debe mejorar considerablemente para aumentar su precisión y, sobretodo, su robustez. Este factor es crucial en entornos en los que el vehículo no puede ser controlado o manipulado, tales como Marte, La Luna u otros cuerpos celestes aún por explorar. Sería inadmisible utilizar estos sensores si no se

confía plenamente en la robustez de sus medidas ya que la generación de datos erróneos o poco fiables en entornos tan hostiles podría ser fatal para la misión. Por ello, considero que queda aún un largo camino de desarrollo tecnológico en el ámbito de los sensores ToF para que puedan ser empleados en misiones de exploración espacial. Además de mejoras en el sensor, se requeriría también un algoritmo de control mucho más sofisticado que el diseñado en este trabajo para que se puedan llevar a cabo maniobras más complejas y precisas basadas en diferentes tipos de sensores como cámaras o sensores de ultrasonidos.

Como posibles líneas de trabajo futuras relacionadas con este proyecto se proponen las siguientes:

- Migración del código completo a Arduino para conseguir una autonomía absoluta y no depender de una conexión inalámbrica estable con Matlab.
- Sofisticación y mejora del algoritmo de cálculo de maniobras para considerar situaciones concretas que hagan más eficiente el funcionamiento del algoritmo de guiado y control.
- Sofisticación y mejora del algoritmo completo para incluir un sistema de referencia que permita al vehículo rotar los 360^{o}.
- Sofisticación y mejora del algoritmo completo para permitir el funcionamiento en terreno tridimensional, considerando las tres coordenadas espaciales y angulares.
- Estudio de la viabilidad y utilidad de combinar diversos sensores de distancias ToF para cubrir un mayor campo de visión angular.
- Estudio de la viabilidad y utilidad de combinar este sensor de distancias ToF con otros sensores de captación de información del medio como cámaras, sensores de infrarrojos o tecnología Lidar.
- Estudio de la viabilidad y utilidad de introducir módulos de tomas de decisiones basados en Inteligencia Artificial para optimizar el cálculo de las maniobras que hagan más eficiente el funcionamiento del algoritmo de guiado y control.

Todas estas interesantes propuestas que se escapan del ámbito de estudio de este proyecto pueden ser abordadas tomando como base teórica y práctica el presente trabajo.

7. Presupuesto

Para concluir este Trabajo de Fin de Grado acerca del diseño y desarrollo de un sistema de guiado y control de un *rover* dedicado a la exploración espacial, se calculan y se detallan a continuación los costes asociados al proyecto.

Dentro de los costes totales del proyecto se pueden diferenciar entre aquellos asociados a la adquisición de los componentes electrónicos y el material utilizado y aquellos asociados al desarrollo del prototipo por parte de un ingeniero.

Coste de los componentes y el material

Los costes asociados a los componentes y el material utilizados hacen referencia a la placa de desarrollo ESP32 NodeMCU, el kit de chasis y motores Black Gladiator de DFRobot, la placa de desarrollo SparkFun Qwiic ToF Imager - VL53L5CX, la Unidad de medida inercial Adafruit - BNO055, el controlador de motores Driver L298 y otros materiales como cables y baterías. El resto de componentes tales como torretas de montaje y tornillos no se han incluído en el presupuesto ya que se tuvo acceso a ellos de forma gratuita en el Taller de Ingeniería Electrónica de la Universitat Politècnica de València. Además, la placa diseñada e impresa en 3D no conlleva tampoco costes de material asociados ya que se tuvo acceso gratuito a la impresora 3D de la misma universidad. Tampoco se han tenido en cuenta los gastos de envío de los componentes disponibles en comercios online, ya que dependen del plazo en el que se necesiten los componentes o de la empresa que realiza el envío.

En cuanto a las licencias de software, no se han incluido en el presupuesto ya que el IDE de Arduino es de código abierto y la licencia de uso de Matlab fue proporcionada de forma gratuita por la universidad al disponer de un acuerdo entre ambas entidades.

El desglose completo de los costes asociados a los componentes y el material se muestra en la *Tabla 7.1*. En ella se incluyen los precios finales, estando el 21 % de IVA ya aplicado.

Material	Fuente	Cantidad [ud]	Coste unidad [€/ud]	Importe [€]
ESP32 NodeMCU	Amazon	1	11.49	11.49
Kit Black Gladiator de DFRobot	DFRobot	1	22.83	22.83
SparkFun Qwiic ToF Imager - VL53L5CX	Sparkfun	1	24.95	24.95
IMU BNO055	Adafruit	1	31.91	31.91
Driver L298	Prometec	1	9.68	9.68
Batería LiPo 2000mAh	Amazon	1	2.50	2.50
Batería 2000mAh	AliExpress	1	0.57	0.57
Cable USB A-B	Amazon	1	2.80	2.80
TOTAL				**106.73**

Tabla 7.1: Costes de los componentes y material utilizados

Coste de mano de obra

Para calcular el coste de la mano de obra, se ha hecho una estimación del tiempo invertido en el desarrollo completo del proyecto, desde la búsqueda de información y recursos hasta la realización de pruebas de validación del prototipo. Este tiempo se ha estimado alrededor de las 200 horas de trabajo.

Para determinar el precio por hora de la mano de obra se ha buscado información en diversas fuentes sobre el salario medio de un Ingeniero Aeroespacial recién graduado. El salario medio ronda los 25 000 € anuales. Considerando una jornada laboral de ocho horas diarias, el salario por hora ronda los 15 €. En la *Tabla 7.2* se recoge el presupuesto asociado a la mano de obra.

Mano de obra	Horas [h]	Coste por hora [€/h]	Importe [€]
Graduado en Ingeniería Aeroespacial	200	15	3000
TOTAL			**3000**

Tabla 7.2: Costes de mano de obra de un graduado en Ingeniería Aeroespacial

Sumando ambos costes se puede estimar un coste total de desarrollo del prototipo alrededor de los 3100 €.

A. Código completo

En este anexo se muestra el código completo del sistema de guiado autónomo del *rover* formado por un *script* de Arduino y dos *scripts* de Matlab.

***Script* de Arduino. Comunicación con los periféricos y bucle de ejecución:**

```
// Include needed lybraries
#include <BluetoothSerial.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <SparkFun_VL53L5CX_Library.h>

// Initialize Bluetooth, IMU and ToF sensor objects object
BluetoothSerial SerialBT;
Adafruit_BNO055 bno = Adafruit_BNO055(55);
SparkFun_VL53L5CX myImager;


// Left DC motor pins
int motor1Pin1 = 25;
int motor1Pin2 = 33;
int enable1Pin = 12;

// Right DC motor pins
int motor2Pin1 = 14;
int motor2Pin2 = 27;
int enable2Pin = 26;

// Set PWM properties
const int freq = 30000;
const int pwmChannel1 = 0;
const int pwmChannel2 = 1;
const int resolution = 8;
int dutyCycle = 250;
int dutyCycle_forward = 0.78*dutyCycle;
int dutyCycle_left;
int dutyCycle_right;

// ToF sensor variables
VL53L5CX_ResultsData measurementData; // Result data class structure,
    1356 byes of RAM
SF_VL53L5CX_RANGING_MODE ranging_mode;  // Sets ToF sensor ranging mode
int imageResolution = 0;
int imageWidth = 0;

// IMU variables
sensors_event_t event;       // Get a new sensor event
float current_angle;         // Current IMU angle

// PID variables
```

```
46 bool PID_working = false;
47 float desired_angle = 0;     // Desired angle
48 float kp = 0;                // Proportional gain for making a turn
49 float ki = 0;                // Integral gain for making a turn
50 float kd = 0;                // Derivator gain for making a turn
51 float kp_forward = 7;        // Proportional gain for forward movement
52 float ki_forward = 10;        // Integral gain for forward movement
53 float kd_forward = 0;        // Derivator gain for forward movement
54
55 float error;                 // Error between desired and current angle
56 float last_error;            // Error in previous iteration
57 long prevT;                  // Time variable to compute integral error
58 float eintegral = 0;         // Integral error
59 float ederivator = 0;        // Derivator error
60 float u;                     // Control action
61 long currT;                  // Time variable to compute integral error
62 float deltaT;                // Time elapsed between two iterations
63
64 // Other needed variables
65 int flag = 0;                // Flag indicating rover start
66 int data;                    // Data received from Matlab
67
68 /* Acquire data measured from sensors and send it to Matlab */
69 void data_acquisition(){
70   //Read distance data into array
71   if (myImager.getRangingData(&measurementData)){
72     // Send ToF data with decreasing y (sensor mounted upside down),
     decreasing x to reflect reality
73     for (int y = imageWidth * (imageWidth - 1); y >= 0 ; y -=
     imageWidth)
74     {
75       for (int x = imageWidth - 1 ; x >= 0 ; x--)
76       {
77         // Send ToF distances to Matlab
78         SerialBT.write((int) measurementData.distance_mm[x + y]/256);
79         SerialBT.write((int) measurementData.distance_mm[x + y]%256);
80
81         // Send ToF distances status to Matlab
82         SerialBT.write((int) measurementData.target_status[x + y]/256);
83         SerialBT.write((int) measurementData.target_status[x + y]%256);
84       }
85     }
86     // Send IMU data to Matlab
87     SerialBT.write( (int) current_angle/256);
88     SerialBT.write( (int) current_angle%256);
89
90     data = 0;    // Reset data value
91   }
92 }
93
94
95 /* Read desired angle from Matlab and adjust PID gain values */
96 void adjustPID(float *ki, float *kp, float *kd, float *desired_angle,
     float current_angle){
97   float delta_angle;  // Delta angle that the rover must turn
98   float byte1;        // Desired angle
99   float byte2;        // Desired angle
100  delay(500);
101
102  // Read desired angle from Matlab
103  if(SerialBT.available()){
104    byte1 = (float) SerialBT.read();
105    byte2 = (float) SerialBT.read();
```

```
106     *desired_angle = byte1*256 + byte2;
107
108     // Find delta angle (correction for differences > 180  )
109     if( fabs(*desired_angle - current_angle) <= 180 ){
110       delta_angle = fabs(*desired_angle - current_angle);
111     }else{
112       delta_angle = 360 - fabs(*desired_angle - current_angle);
113     }
114
115     // Adjust gain values depending on delta angle
116     if( delta_angle >= 15 && delta_angle <= 80){
117       *kp = 7;
118       *ki = -(4.0/15.0)*delta_angle + 25;
119     }else if(delta_angle < 15){
120       *kp = 10;
121       *ki = 22;
122     }else{
123       *kp = 2;
124       *ki = 5;
125     }
126     *kd = 0;
127     delay(500);
128   }else{
129     Serial.println("Desired angle was not read from Matlab");
130   }
131 }
132
133 /* Resets PID variables to start making a turn */
134 void reset_PID_variables(){
135   PID_working = true;
136   prevT = micros();   //Re-start time to avoid delays due to adjustPID
        () function
137   eintegral = 0;      // Re-set the integral error to 0
138   ederivator = 0;     // Re-set the derivator error to 0
139   data = 0;
140 }
141
142 /* Operates the PID to make a turn */
143 void operate_PID(){
144   // Compute error between desired and current angle
145   if(desired_angle > current_angle ){
146     if( fabs(desired_angle - current_angle) <= 180){
147       error = fabs(desired_angle - current_angle);         // CW turn (
        right)
148     }else{
149       error = fabs(desired_angle - current_angle) - 360;  // CCW turn (
        left)
150     }
151   }else{
152     if( fabs(desired_angle - current_angle) <= 180){
153       error = -fabs(desired_angle - current_angle);        // CCW turn (
        left)
154     }else{
155       error = 360 - fabs(desired_angle - current_angle);   // CW turn (
        right)
156     }
157   }
158
159   // If it is 1st iteration, update prevT to avoid delays
160   if(prevT == 0){
161     prevT = micros();
162   }
163
```

```
164   // Compute time elapsed between two iterations
165   currT = micros();
166   deltaT = ((float) (currT-prevT))/1.0e6;
167
168   //Compute integral error
169   eintegral = eintegral + error*deltaT;
170
171   // Compute derivator error
172   ederivator = (error - last_error)/deltaT;
173
174   // Update variables
175   prevT = currT;
176
177
178   // Compute control signal u
179   u = kp*error + ki*eintegral + kd*ederivator;
180
181   // Set motor duty
182   dutyCycle = (int) fabs(u);
183   if(dutyCycle > 250){
184     dutyCycle = 250;     // Maximum Duty = 250
185   }
186
187   // Operate motors to make turn
188   make_turn();
189 }
190
191 /* Operate motors to make turn*/
192 void make_turn(){
193   // Turn CW (right)
194   if( u >= 0){
195     digitalWrite(motor1Pin1, HIGH);
196     digitalWrite(motor1Pin2, LOW);
197     digitalWrite(motor2Pin1, LOW);
198     digitalWrite(motor2Pin2, HIGH);
199
200     ledcWrite(pwmChannel1, dutyCycle);
201     ledcWrite(pwmChannel2, dutyCycle );
202   }
203
204   // Turn CCW (left)
205   if( u < 0){
206     digitalWrite(motor1Pin1, LOW);
207     digitalWrite(motor1Pin2, HIGH);
208     digitalWrite(motor2Pin1, HIGH);
209     digitalWrite(motor2Pin2, LOW);
210
211     ledcWrite(pwmChannel1, dutyCycle);
212     ledcWrite(pwmChannel2, dutyCycle );
213   }
214   print_data();
215 }
216
217 /* Stop the rover */
218 void stop(){
219   digitalWrite(motor1Pin1, LOW);
220   digitalWrite(motor1Pin2, LOW);
221   digitalWrite(motor2Pin1, LOW);
222   digitalWrite(motor2Pin2, LOW);
223 }
224
225 /* Operate PID while moving forward */
226 void PID_forward(){
```

```
227   // Compute error between desired and current angle
228   if(desired_angle > current_angle ){
229     if( fabs(desired_angle - current_angle) <= 180){
230       error = fabs(desired_angle - current_angle);          // CW turn (
      right)
231     }else{
232       error = fabs(desired_angle - current_angle) - 360;   // CCW turn (
      left)
233     }
234   }else{
235     if( fabs(desired_angle - current_angle) <= 180){
236       error = -fabs(desired_angle - current_angle);         // CCW turn (
      left)
237     }else{
238       error = 360 - fabs(desired_angle - current_angle);    // CW turn (
      right)
239     }
240   }
241
242   // If it is 1st iteration, update prevT to avoid delays
243   if(prevT == 0){
244     prevT = micros();
245   }
246
247   // Compute time elapsed between two iterations
248   currT = micros();
249   deltaT = ((float) (currT-prevT))/1.0e6;
250
251   //Compute integral error
252   eintegral = eintegral + error*deltaT;
253
254   // Compute derivator error
255   ederivator = (error - last_error)/deltaT;
256
257   // Update variables
258   prevT = currT;
259
260
261   // Compute control signal u
262   u = kp_forward*error + ki_forward*eintegral + kd_forward*ederivator;
263
264   if( u <= 0){
265     dutyCycle_left = dutyCycle_forward;
266     dutyCycle_right = dutyCycle_forward + (int) fabs(u);
267     if(dutyCycle_right > 250){
268       dutyCycle_right = 250;
269     }
270   }else{
271     dutyCycle_right = dutyCycle_forward;
272     dutyCycle_left = dutyCycle_forward + (int) fabs(u);
273     if(dutyCycle_left > 250){
274       dutyCycle_left = 250;
275     }
276   }
277 }
278
279 /* Move forward */
280 void move_forward(){
281   digitalWrite(motor1Pin1, HIGH);
282   digitalWrite(motor1Pin2, LOW);
283   digitalWrite(motor2Pin1, HIGH);
284   digitalWrite(motor2Pin2, LOW);
285
```

```
286   prevT = micros();   // Re-start time to avoid delays due to adjustPID
        () function
287   eintegral = 0;      // Re-set the integral error to 0
288   ederivator = 0;     // Re-set the derivator error to 0
289
290   PID_forward();      // Calculate control action
291   ledcWrite(pwmChannel1, dutyCycle_left);
292   ledcWrite(pwmChannel2, dutyCycle_right);
293   //print_data();
294 }
295
296 /* Print data on serial monitor to check values */
297 void print_data(){
298   Serial.print(desired_angle);
299   Serial.print(" ");
300   Serial.print(current_angle);
301   Serial.print(" ");
302   Serial.print(kp);
303   Serial.print(" ");
304   Serial.print(ki);
305   Serial.print(" ");
306   Serial.print(error);
307   Serial.print(" ");
308   Serial.print(eintegral);
309   Serial.print(" ");
310   Serial.print(ederivator);
311   Serial.print(" ");
312   Serial.print(dutyCycle);
313   Serial.print(" ");
314   Serial.print(dutyCycle_left);
315   Serial.print(" ");
316   Serial.print(dutyCycle_right);
317   Serial.println();
318   delay(1);
319 }
320
321 /* Initial configuration */
322 void setup() {
323   SerialBT.begin("ESP32BT");  // Inicialize Bluetooth object
324   Serial.begin(115200);       // Inicialize serial communication
325   Wire.begin();               // Resets to 100kHz I2C
326   Wire.setClock(400000);      // Sets max I2C freq to 400kHz
327
328   // Initialise the IMU
329   if(!bno.begin())
330   {
331     Serial.print("Ooops, no BNO055 detected ... Check your wiring or
        I2C ADDR!");
332     while(1);
333   }
334
335   // Initialise the ToF Sensor
336   if (myImager.begin() == false)
337   {
338     Serial.println(F("Sensor not found - check your wiring. Freezing"))
        ;
339     while (1);
340   }
341
342   // Initial configuration of ToF Sensor
343   myImager.setResolution(8*8);                    // Enable all 64 pads
344   imageResolution = myImager.getResolution();     // Query sensor for
        current resolution - either 4x4 or 8x8
```

```
345   imageWidth = sqrt(imageResolution);             // Calculate printing
        width
346   myImager.setRangingFrequency(10);               // Sets ranging
        frequency to 10Hz
347   ranging_mode = SF_VL53L5CX_RANGING_MODE::CONTINUOUS;  // Sets
        operating mode to CONTINUOUS
348   myImager.setRangingMode(ranging_mode);
349   myImager.setSharpenerPercent(10);               // Sets sharpener
        percentage to 10
350   myImager.startRanging();                        // Start ranging
        operation
351
352   // Configure the motor pins as outputs
353   pinMode(motor1Pin1, OUTPUT);
354   pinMode(motor1Pin2, OUTPUT);
355   pinMode(enable1Pin, OUTPUT);
356
357   pinMode(motor2Pin1, OUTPUT);
358   pinMode(motor2Pin2, OUTPUT);
359   pinMode(enable2Pin, OUTPUT);
360
361   // Configure LED PWM functionalitites
362   ledcSetup(pwmChannel1, freq, resolution);
363   ledcSetup(pwmChannel2, freq, resolution);
364
365   // Attach the channel to the GPIO to be controlled
366   ledcAttachPin(enable1Pin, pwmChannel1);
367   ledcAttachPin(enable2Pin, pwmChannel2);
368
369   delay(1000);
370
371   // Use external crystal as clock (better timing accuracy for IMU)
372   bno.setExtCrystalUse(true);
373 }
374
375 /* System execution loop */
376 void loop() {
377   // Matlab sends flag = 1 to start the operation of the system
378   if(flag == 0 & SerialBT.available()){
379     flag = SerialBT.read();
380   }
381
382   // Control algorithm execution
383   if(flag == 1){
384
385     // Matlab sends one byte indicating the operation to execute
386     if(SerialBT.available()){
387       data = SerialBT.read();
388       Serial.println(data);
389     }
390
391     // Read data from IMU
392     bno.getEvent(&event);
393     current_angle = event.orientation.x;
394
395     // Execute operation indicated by Matlab
396     if(data == 3){     // Acquire sensor data and send it to Matlab
397       data_acquisition();
398
399     }else if(data == 4) {    // Adjust PID to start making a turn
400       stop();
401       adjustPID(&kp, &ki, &kd, &desired_angle, current_angle);
402       reset_PID_variables();
```

```
403     }
404
405     while( PID_working ){
406       operate_PID();
407       last_error = error;
408
409       // Read data from IMU
410       bno.getEvent(&event);
411       current_angle = event.orientation.x;
412
413       // Condition to finish PID operation
414       if( fabs(desired_angle - current_angle) < 3 || dutyCycle < 100){
415         Serial.println("PID finished");
416         PID_working = false;
417         stop();
418         SerialBT.write(1);     // Tell matlab that turn is finished
419         break;
420       }
421     }
422
423     // Move forward if no turn is going on
424     if(PID_working == false){
425       move_forward();
426     }
427
428     delay(1);
429
430   }
431 }
```

A.1: Script de Arduino que controla la ejecución del sistema de guiado

***Script* 1 de Matlab. Procesamiento de datos y comunicación con Arduino:**

```
%% Create rover data
rover.m = eye(8);    % Distance 8x8 matrix
rover.status = eye(8);
rover.angle = 0;     % (Current angle measured by IMU)

%% Connect Matlab with ESP32 to send PWM duty
bt = bluetooth("ESP32BT",1, "Timeout", 30);

%% Process data from ToF sensor and IMU

% Initialize loop
i = 1;
write(bt,1);
while(1 )
    % Read from ToF Sensor
    write(bt,3)      % Tells ESP32 that Matlab is ready to
    read data
    for row = 1:8
        for col = 1:8
            byte_ToF_1 = read(bt,1);
            byte_ToF_2 = read(bt,1);

            byte_status_1 = read(bt,1);
            byte_status_2 = read(bt,1);

            m(row,col) = byte_ToF_1*256 + byte_ToF_2;
            rover.status(row,col) = byte_status_1*256 +
   byte_status_2;
        end
    end

    % Read from IMU
    byte_imu_1 = read(bt,1);
    byte_imu_2 = read(bt,1);

    % Fix incorrect data values
    for row = 1:8
        for col = 1:8
            if(rover.status(row,col) ~= 5)
                m(row,col) = 800;
            end
        end
    end

    % Set rover variables
    rover.m = m;
    rover.angle= byte_imu_1*256 + byte_imu_2;

    % Calculate desired angle
```

```
48      [desired_angle, make_turn] = calculateAngle(rover);
49
50      % Adjust PID for new turn
51      if(make_turn)
52           write(bt, 4);   % Tells rover that needs to turn
53           write(bt,fix(desired_angle/256));
54           write(bt,mod(desired_angle,256));
55           finished = read(bt,1);
56      end
57
58      % Update variables
59      i = i + 1;
60
61  end
```

A.2: Script de Matlab que procesa los datos de las distancias ToF

***Script* 2 de Matlab. Cálculo de la maniobra requerida:**

```
% Function that calculates the turn angle
function [desired_angle, make_turn] = calculateAngle (rover)
    % Angle that rover must turn if needed
    turn_angle = 25;
     % Set limit distance to make turn
    limit = 550;
    %% Check if rover can move forward
    min_forward = min(min(rover.m(2:4, 3:6)));

    % Move forward if possible
    if(min_forward >= limit)
        desired_angle = rover.angle;
        make_turn = false;
        return;
    end

    %% Check left and right (columns 1,2 and 7,8)
    min_cols(1) = min(rover.m(2:4,1));   max_cols(1) = max(rover.m(2:4,1));
    min_cols(2) = min(rover.m(2:4,2));   max_cols(2) = max(rover.m(2:4,2));
    min_cols(3) = min(rover.m(2:4,7));   max_cols(3) = max(rover.m(2:4,7));
    min_cols(4) = min(rover.m(2:4,8));   max_cols(4) = max(rover.m(2:4,8));

    % Find best direction to turn if both sides are possible
    max_distance = 0;
    col = 0;
    for i = 1:4
        if( (min_cols(i) >= limit || max_cols(i) >= 800 ) && max_cols(i) > max_distance)
            max_distance = max_cols(i);
            col = i;
        end
    end

    % Turn left
    if(col <= 2 && col > 0)
        make_turn = true;
        if(rover.angle >= 270)    % was pointing to left
            desired_angle = rover.angle - turn_angle;
            if(desired_angle < 270)
                desired_angle = 270;
            end
        else                      % was pointint to right
            desired_angle = rover.angle - turn_angle;
```

```
43              if(desired_angle < 0)
44                  desired_angle = 360 + desired_angle;
45              end
46          end
47          return;
48      % Turn right
49      elseif (col >= 3 && col > 0 )
50          make_turn = true;
51          if(rover.angle <= 90)   % was pointint to right
52              desired_angle = rover.angle + turn_angle;
53              if(desired_angle > 90)
54                  desired_angle = 90;
55              end
56          else                     % was pointint to left
57              desired_angle = rover.angle + turn_angle;
58              if(desired_angle > 360)
59                  desired_angle = desired_angle - 360;
60              end
61          end
62          return;
63      end
64
65      %% All directions are blocked
66      disp(" All directions blocked");
67      make_turn = true;
68      if(rover.angle <= 90)       % was pointing to right
69          desired_angle = rover.angle - turn_angle;
70          if(desired_angle < 0)
71              desired_angle = 360 + desired_angle;
72          end
73      elseif(rover.angle >= 270)  % was pointing to right
74          desired_angle = rover.angle + turn_angle;
75          if(desired_angle > 360)
76              desired_angle = desired_angle - 360;
77          end
78      end
79      return;
80
81      %% Other cases
82      disp("Error. Case not expected");
83
84  end
```

A.3: Script de Matlab que calcula la maniobra requerida

B. Planos

En este anexo se presentan los planos de la pieza diseñada e impresa en 3D que sirve como base superior del ensamblaje del *rover*.

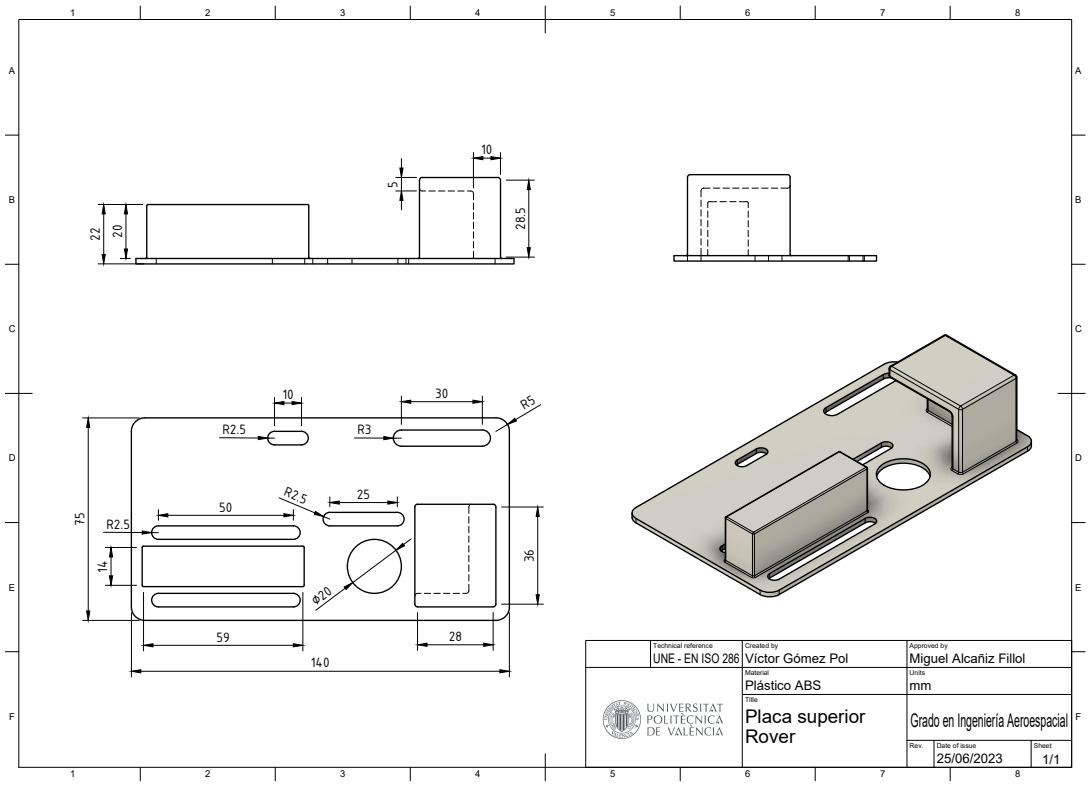


Figura B.1: Planos de la placa superior del *rover*

C. Fichas técnicas

En este anexo se presentan las fichas técnicas de los distintos componentes electrónicos utilizados en el desarrollo del presente trabajo.

Microcontrolador ESP32

IMU BNO055

Sensor de distancias ToF VL53L5CX (Ficha técnica)

Sensor de distancias ToF VL53L5CX (Manual de usuario)

Controlador de motores Driver L298

D. Pliego de condiciones

En este apéndice se hace referencia a la normativa relativa a seguridad e higiene que se ha seguido durante el desarrollo del presente trabajo. Además, se exponen las condiciones bajo las que se ha comprobado el funcionamiento del prototipo desarrollado y se presentan recomendaciones de uso.

D.1. Normativa de Seguridad e Higiene

El desarrollo del presente Trabajo (desde el ensamblaje del prototipo hasta la realización de las pruebas de verificación del funcionamiento) se ha realizado en las instalaciones de la Escuela Técnica Superior de Ingeniería del Diseño (ETSID), perteneciente a la Universitat Politècnica de València. Por ello, durante el desarrollo del proyecto se ha cumplido la Normativa General de Seguridad e Higiene en el Trabajo y Ensayos en Laboratorio, publicada en el BOE número 64 el 16 de marzo de 1971.

Obligaciones y derechos de los trabajadores

Según el Artículo 11, los trabajadores tienen la responsabilidad de colaborar en la prevención de riesgos laborales en la empresa y mantener un alto nivel de higiene en el lugar de trabajo. Para cumplir con estos objetivos, deben seguir los requisitos de esta ordenanza y sus instrucciones complementarias. También deben seguir las órdenes e instrucciones que se les den por parte de sus superiores.

Los trabajadores tienen la obligación expresa de:

- Recibir formación sobre seguridad, higiene y primeros auxilios en los centros de trabajo, proporcionada por la empresa o instituciones autorizadas.
- Utilizar correctamente los equipos de protección personal y asegurarse de que estén en buen estado y conservación.
- Informar de inmediato a sus superiores sobre cualquier avería o deficiencia que pueda causar peligro en el centro de trabajo.
- Mantener una buena higiene personal para evitar enfermedades contagiosas o incomodidades a sus compañeros de trabajo.
- Someterse a los exámenes médicos requeridos y recibir las vacunas o inmunizaciones ordenadas por las autoridades sanitarias competentes o por el servicio médico de la empresa.

- No introducir bebidas u otras sustancias no autorizadas en el lugar de trabajo, ni presentarse o permanecer en estado de embriaguez o intoxicación.

- Colaborar en la extinción de incendios y en el rescate de las víctimas de accidentes laborales, según las condiciones que sean razonablemente exigibles en cada caso.

Seguridad estructural

Según el artículo 13, Se requiere que todos los edificios, ya sean permanentes o temporales, estén construidos de manera segura y estable para evitar riesgos de derrumbe y daños causados por condiciones atmosféricas. Esto implica que los cimientos, pisos y otros elementos de los edificios deben tener la resistencia adecuada para soportar y suspender de manera segura las cargas para las que han sido diseñados.

Además, es necesario indicar mediante rótulos o inscripciones las cargas máximas que los espacios puedan soportar o suspender, y está prohibido sobrecargar los pisos y plantas de los edificios.

Superficies y cubicación

Según el artículo 14, los espacios de trabajo deben cumplir con los siguientes requisitos mínimos:

- Una altura de tres metros desde el piso hasta el techo.

- Una superficie de dos metros cuadrados por cada trabajador.

- Un volumen de diez metros cúbicos por cada trabajador.

Sin embargo, en establecimientos comerciales, de servicios, oficinas y despachos, la altura mencionada anteriormente (tres metros) puede ser reducida a 2,50 metros, siempre y cuando se respete el volumen por trabajador establecido y se garantice una adecuada renovación del aire. A la hora de calcular la superficie y el volumen, no se deben tener en cuenta los espacios ocupados por máquinas, equipos, instalaciones y materiales.

Suelos, techos y paredes

Según el artículo 15, el suelo será uniforme, plano y liso, sin interrupciones, y estará hecho de un material resistente, que no sea resbaladizo o que pueda volverse resbaladizo con el uso, y que sea fácil de limpiar. Si hay desniveles, se utilizarán rampas con una pendiente no superior al 10 por ciento para nivelarlos.

En cuanto a las paredes, estas deben ser lisas, revestidas o pintadas en colores claros y que puedan ser lavadas o blanqueadas. Los techos deben ser adecuados para proteger a los trabajadores de las inclemencias del tiempo. Si los techos tienen que soportar cargas, deben cumplir con las mismas condiciones establecidas para los pisos en el artículo 13.

Iluminación

Según el artículo 25, todos los espacios de trabajo o áreas de paso contarán con una iluminación adecuada, ya sea natural, artificial o una combinación de ambas, según las actividades que se realicen. Se dará preferencia al uso de la iluminación natural siempre que sea posible. Se deberá aumentar la intensidad de la iluminación en máquinas peligrosas, zonas de paso con riesgo de caídas, escaleras y salidas de emergencia.

Ventilación, humedad y temperatura

Según el artículo 30, en los lugares de trabajo y sus áreas complementarias, se mantendrán condiciones atmosféricas apropiadas mediante medios naturales o artificiales, evitando la presencia de aire viciado, excesos de calor o frío, humedad o sequedad, y olores desagradables. En los espacios de trabajo cerrados, se garantizará un suministro mínimo de aire fresco y limpio de 30 a 50 metros cúbicos por hora y por trabajador, a menos que se realice una renovación total del aire varias veces por hora, no siendo inferior a seis veces en trabajos sedentarios y diez veces en trabajos que requieran esfuerzo físico superior al normal.

En los centros de trabajo expuestos a temperaturas extremas, se evitarán cambios bruscos mediante el uso del método más eficaz. Si hay una gran diferencia de temperatura entre diferentes áreas de trabajo, se deberán proporcionar espacios de transición para que los trabajadores se adapten gradualmente.

Se establecen los siguientes límites normales de temperatura y humedad en los espacios y para diferentes tipos de trabajos, siempre que el proceso de fabricación lo permita:

- Para trabajos sedentarios: de 17ºC a 22ºC.
- Para trabajos regulares: de 15ºC a 18ºC.
- Para trabajos que requieran un esfuerzo físico intenso: de 12ºC a 15ºC.

La humedad relativa del ambiente oscilará entre el 40 % y el 60 %, excepto en instalaciones donde pueda generarse electricidad estática, donde deberá ser superior al 50 %. Las instalaciones que generen calor o frío se ubicarán a una distancia adecuada de los espacios de trabajo para evitar riesgos de incendio o explosión, la liberación de gases nocivos, la radiación directa de calor o frío, y corrientes de aire perjudiciales para los trabajadores. Además, en los trabajos que se realicen en espacios cerrados con temperaturas extremadamente altas o bajas, se limitará la permanencia de los trabajadores estableciendo turnos adecuados, si es necesario.

Soldadura eléctrica

Según el artículo 54, en la instalación y uso de soldadura eléctrica, se deben cumplir las siguientes regulaciones obligatorias:

Los terminales de conexión para los circuitos de alimentación de los equipos manuales de soldadura deben estar cuidadosamente aislados.

Las carcasas de cada equipo de soldadura deben estar conectadas a tierra, al igual que uno de los conductores del circuito de soldadura. Siempre y cuando no se generen

corrientes peligrosas debido a la conexión a tierra, se permite conectar uno de los polos del circuito de soldadura a estas carcasas. De lo contrario, el circuito de soldadura debe estar conectado a tierra en el lugar de trabajo.

Cuando se realicen trabajos de soldadura en espacios altamente conductores, no se deben utilizar tensiones superiores a 50 voltios. En caso contrario, la tensión entre el electrodo y la pieza a soldar no debe superar los 90 voltios en corriente alterna o los 150 voltios en corriente continua. El equipo de soldadura debe colocarse en el exterior del área donde trabaja el operador.

La superficie exterior de los portaelectrodos manuales, y en la medida de lo posible, sus mandíbulas, deben estar aisladas.

El soldador y sus ayudantes deben disponer y utilizar visores, gorros o pantallas para proteger su vista, así como discos o guantes para proteger sus manos. Además, deben usar delantales de cuero y botas que cumplan con las características especificadas en el capítulo XIII de esta ordenanza.

D.2. Condiciones de uso

Durante el desarrollo del prototipo de *rover* espacial a escala se ha sometido dicho modelo a diversas condiciones externas para las que se puede garantizar el correcto funcionamiento del sistema. Sin embargo, si se opera el prototipo bajo condiciones externas distintas se podrían producir efectos no deseados que no han sido detectados y que inhabiliten el sistema.

Condiciones de funcionamiento comprobadas

El prototipo ha sido sometido a pruebas exhaustivas bajo las siguientes condiciones de funcionamiento:

- Luz ambiental: El funcionamiento del *rover* se ha comprobado en condiciones de luz ambiental típica. Se ha verificado su capacidad para detectar obstáculos y tomar medidas evasivas de manera adecuada.
- Terreno plano y firme: Las pruebas se han realizado en superficies planas y firmes para garantizar un desplazamiento suave y preciso del *rover*. Se ha comprobado su capacidad para evitar obstáculos y mantener una trayectoria estable en estas condiciones.
- Obstáculos estáticos: El sistema de evasión de obstáculos ha sido evaluado con objetos estáticos colocados en la trayectoria del *rover*. Se ha verificado su capacidad para detectar y evitar dichos obstáculos de manera efectiva.

Limitaciones de funcionamiento

Es importante tener en cuenta las siguientes limitaciones y condiciones específicas del prototipo:

- Condiciones climáticas: No se ha comprobado el funcionamiento del prototipo en condiciones climáticas adversas, como lluvia intensa. El contacto directo con la humedad o la exposición prolongada a condiciones húmedas puede afectar el correcto funcionamiento de los componentes electrónicos. Se recomienda evitar su operación en condiciones climáticas adversas.
- Nivel de luminosidad: El prototipo ha sido probado en condiciones de luz ambiental, pero no se ha evaluado su funcionamiento bajo incidencia directa de la luz solar u otras fuentes de luz intensa sobre el sensor ToF. Se advierte que la exposición directa a la luz intensa puede afectar la precisión de las mediciones y el rendimiento del sistema de evasión de obstáculos.
- Superficies irregulares: El prototipo ha sido diseñado y probado para su operación en terrenos planos y firmes. Se debe tener precaución al operar el *rover* en superficies irregulares o inestables, ya que esto puede afectar su estabilidad y capacidad para evitar obstáculos de manera efectiva.

- Condiciones de prueba limitadas: Las pruebas realizadas se han llevado a cabo en un entorno controlado y bajo condiciones específicas. No se garantiza el correcto funcionamiento del prototipo en todas las situaciones posibles. El rendimiento puede variar en función de factores ambientales, características del terreno y otros elementos imprevistos.

Recomendaciones de uso

Para garantizar un funcionamiento óptimo y seguro del prototipo de *rover* espacial a escala, se recomienda seguir las siguientes pautas:

- Operar en condiciones favorables: Preferiblemente, utilizar el *rover* en condiciones de luz ambiental moderada y terrenos planos y firmes.

- Evitar condiciones climáticas adversas: No utilizar el prototipo en condiciones de lluvia intensa o exposición prolongada a la humedad.

- Supervisión constante: Mantener una supervisión constante durante la operación del prototipo para detectar cualquier anomalía o comportamiento inesperado.

- Protección del sensor ToF: Evitar la exposición directa del sensor ToF a fuentes de luz intensa, como el sol o luces brillantes, para evitar posibles interferencias en su funcionamiento.

- Mantenimiento regular: Realizar revisiones periódicas del prototipo para garantizar el correcto estado de los componentes electrónicos y realizar cualquier reparación o reemplazo necesario.

E. Objetivos de Desarrollo Sostenible, Agenda 2030

En el campo de la exploración espacial, el desarrollo de sistemas avanzados de guiado y navegación es fundamental para garantizar el éxito de las misiones. Un aspecto clave en este sentido es la capacidad de los *rovers* espaciales para sortear obstáculos en su camino y realizar exploraciones precisas en cuerpos celestes. En línea con los Objetivos de Desarrollo Sostenible de la Agenda 2030, se ha enfocado el desarrollo de un sistema de guiado de obstáculos para un *rover* espacial que busca combinar la eficiencia y la sostenibilidad en la exploración espacial.

Objetivo 7: Energía asequible y no contaminante

En la exploración espacial, la eficiencia energética es fundamental debido a las limitaciones de recursos en el espacio. Al desarrollar un sistema de guiado de obstáculos basado en un sensor ToF con un bajo consumo de energía, se promueve la utilización eficiente de los recursos. Este sistema también reduce el peso en comparación con sistemas de guiado más tradicionales, permitiendo optimizar el espacio y peso que ocupa el sistema en las naves espaciales que llevan los *rovers* a otros cuerpos celestes.

Objetivo 9: Industria, innovación e infraestructura

El desarrollo de sistemas avanzados de guiado y navegación para *rovers* espaciales basados en el uso de sensores ToF implica una mejora en la infraestructura tecnológica y fomenta la innovación en el sector espacial. Estas innovaciones pueden tener aplicaciones tanto en el ámbito espacial como en la Tierra, impulsando el desarrollo de nuevas industrias y mejorando la empleabilidad y calidad de vida de las personas que viven en zonas próximas a estas industrias.

Objetivo 11: Ciudades y comunidades sostenibles

Aunque el enfoque principal del desarrollo de sistemas de guiado de obstáculos para *rovers* espaciales no está directamente relacionado con las ciudades, sus aplicaciones tienen un impacto indirecto en la sostenibilidad urbana. La tecnología y los conocimientos adquiridos en la exploración espacial pueden utilizarse para mejorar la gestión de infraestructuras urbanas, la planificación del transporte y la seguridad en las ciudades. Numerosos ejemplos de avances tecnológicos desarrollados para la exploración espacial han terminado siendo útiles en muchos otros campos y han contribuido al desarrollo de ciudades y comunidades sostenibles tales como las células de energía solar o los sensores CMOS.

Objetivo 13: Acción por el clima

El desarrollo sostenible de sistemas de guiado de obstáculos para *rovers* espaciales

fomenta la conciencia sobre la importancia de la preservación del medio ambiente. Al minimizar el consumo de energía y optimizar los recursos utilizados con la introducción de sensores ToF, se reduce la huella ambiental de las misiones espaciales y se promueve la adopción de prácticas más sostenibles en el ámbito espacial.

Objetivo 17: Alianzas para lograr los objetivos

El desarrollo de sistemas de guiado de obstáculos para *rovers* espaciales requiere la colaboración y la cooperación entre diferentes actores, como agencias espaciales, instituciones de investigación, empresas y organizaciones internacionales. Estas alianzas fomentan el intercambio de conocimientos, recursos y tecnología, impulsando el desarrollo sostenible en el ámbito espacial y promoviendo la cooperación global.

Conclusiones

El desarrollo de un sistema de guiado de obstáculos para un *rover* espacial en el marco de los Objetivos de Desarrollo Sostenible de la Agenda 2030 muestra la interconexión entre la exploración espacial y el desarrollo sostenible en la Tierra. Al buscar la eficiencia y la sostenibilidad en la exploración espacial, se generan beneficios tangibles en términos económicos, medioambientales y sociales. Este enfoque impulsa la investigación y la innovación, promueve el uso eficiente de los recursos, fomenta la colaboración global y sienta las bases para un futuro sostenible tanto en el espacio como en nuestro planeta.

Bibliografía

[1] T. Barfoot, P. Furgale, B. Stenning et al., "Field testing of a rover guidance, navigation, and control architecture to support a ground-ice prospecting mission to Mars," *Robotics and Autonomous Systems*, 2011. dirección: https://www.sciencedirect.com/science/article/pii/S0921889011000388.

[2] E. Zereik, A. Biggio, A. Merlo y G. Casalino, "VISION-BASED PERCEPTION AND SENSOR DATA INTEGRATION FOR A PLANETARY EXPLORATION ROVER," 2011. dirección: http://robotics.estec.esa.int/ASTRA/Astra2011/Papers/06B/FCXNL-11A06-2145465-1-2145465zereik.pdf.

[3] J. Maki, D. Thiessen, A. Pourangi et al., "The Mars Science Laboratory Engineering Cameras," *Space Science Reviews*, 2012. dirección: https://doi.org/10.1007/s11214-012-9882-4.

[4] J. N. Maki, D. Gruel, C. McKinney et al., "The Mars 2020 Engineering Cameras and Microphone on the Perseverance Rover: A Next-Generation Imaging System for Mars Exploration," *Space Science Reviews*, 2020. dirección: https://doi.org/10.1007/s11214-020-00765-9.

[5] M. Azkarate, L. Gerdes, T. Wiese et al., "Concept, Development and Testing of Mars Rover Prototypes for ESA Planetary Exploration," 2021. dirección: https://riuma.uma.es/xmlui/bitstream/handle/10630/22480/IEEE_RAM_ExoTeR_MaRTA-compressed.pdf.

[6] M. Azkarate Vecilla, "Autonomous navigation of planetary rovers," 2022. dirección: https://riuma.uma.es/xmlui/handle/10630/24220.

[7] R. Simmons, L. Henriksen, L. Chrisman y G. Whelan, "Obstacle avoidance and safeguarding for a lunar rover," 1996. dirección: https://www.cs.cmu.edu/afs/cs/usr/reids/www/home/papers/AIAAobsAvoid.pdf.

[8] T. Williamson y C. Thorpe, "A specialized multibaseline stereo technique for obstacle detection," 1998. dirección: https://www.ri.cmu.edu/pub_files/pub1/williamson_todd_1998_2/williamson_todd_1998_2.pdf.

[9] R. Mandelbaum, L. McDowell, L. Bogoni, B. Reich y M. Hansen, "Real-time stereo processing, obstacle detection, and terrain estimation from vehicle-mounted stereo cameras," 1998. dirección: https://ieeexplore.ieee.org/document/732909.

[10] S. Goldberg, M. Maimone y L. Matthies, "Stereo vision and rover navigation software for planetary exploration," 2002. dirección: https://www2.ece.ohio-state.edu/ion/documents/IEEE_aero.pdf.

[11] A. Rankin, M. Maimone, J. Biesiadecki, N. Patel, D. Levine y O. Toupet, "Mars Curiosity Rover Mobility Trends During the First Seven Years," dirección: https://www-robotics.jpl.nasa.gov/media/documents/ROB-20-0040_R3.pdf.

[12] G. Lentaris, K. Maragos, D. Soudris, X. Zabulis y M. Lourakis, "Single- and Multi-FPGA Acceleration of Dense Stereo Vision for Planetary Rovers," *ACM Transactions on Embedded Computing Systems*, 2019. dirección: https://www.researchgate.net/profile/Konstantinos-Maragos/publication/331853349_Single-_and_Multi-FPGA_Acceleration_of_Dense_Stereo_Vision_for_Planetary_Rovers/links/5cb05b614585156cd7918421/Single-and-Multi-FPGA-Acceleration-of-Dense-Stereo-Vision-for-Planetary-Rovers.pdf.

[13] P. Weclewski, R. Marc, B. Brayzier et al., *Sample Fetch Rover Guidance, Navigation and Control Subsystem - An Overview*. 2022. dirección: https://www.researchgate.net/profile/Robert-Marc/publication/361892678_Sample_Fetch_Rover_Guidance_Navigation_and_Control_Subsystem_-_An_Overview/links/62cb13f0cab7ba7426e3723f/Sample-Fetch-Rover-Guidance-Navigation-and-Control-Subsystem-An-Overview.pdf.

[14] I. Kostavelis, E. Boukas, L. Nalpantidis y A. Gasteratos, "Stereo-Based Visual Odometry for Autonomous Robot Navigation," *International Journal of Advanced Robotic Systems*, 2016. dirección: https://doi.org/10.5772/62099.

[15] M. Dimastrogiovanni, F. Cordes y G. Reina, "Terrain Estimation for Planetary Exploration Robots," *Applied Sciences*, 2020. dirección: https://www.mdpi.com/2076-3417/10/17/6044.

[16] D. Green, J. Sasiadek y G. Vukovich, "Guidance and control of an autonomous planetary rover," 1993. dirección: https://ieeexplore.ieee.org/document/585690.

[17] J. A. Christian y S. Cryan, "A Survey of LIDAR Technology and its Use in Spacecraft Relative Navigation," 2013. dirección: https://arc.aiaa.org/doi/10.2514/6.2013-4641.