UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES
GRADO EN INGENIERÍA ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA

Diseño y construcción de un robot explorador de entornos y generador de mapas

Design and construction of a robot to explore environments and build maps

Realizado por
Antonio Roldán Gómez
Tutorizado por
Manuel Roldán Castro
Cotutorizado por
Vicente Jesús Benjumea García
Departamento
Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, 13 de enero de 2023

Resumen

En este documento se presenta el desarrollo de un robot móvil semiautónomo, capaz de explorar entornos y generar mapas representativos. El proyecto abarca las fases necesarias de análisis, diseño y construcción del sistema.

Inicialmente se realizará un análisis de los requisitos necesarios para lograr el objetivo, tanto a nivel de comportamiento como a nivel de construcción del robot. Este análisis dará paso al diseño de hardware, donde se seleccionan los componentes del robot, entre los que se encontrará un sensor *LIDAR* y se realiza el montaje del mismo. Además, se llevará a cabo una calibración de los sensores y actuadores que forman el robot, con el objetivo de conseguir un sistema controlable.

Posteriormente, se desarrolla la programación del sistema, donde se distinguen tres procesos principales: la exploración del robot, que se realizará en tiempo real usando FreeRTOS en C++; la construcción y generación del mapa, programada en Python en una Raspberry Pi; y el control del sistema, usando una aplicación Android de desarrollo propio en Java.

Palabras claves: ESP32, FreeRTOS, MQTT, Android, Raspberry Pi, Mapa de ocupación, LIDAR

Abstract

This document presents the development of a semi-autonomous mobile robot capable of exploring environments and generating representative maps. The project covers the necessary phases for the design and construction of the system.

Initially, an analysis of the requirements necessary to achieve the objective will be carried out, both in terms of the behaviour and the construction of the robot.

This analysis will give way to the hardware design, where the components of the robot will be defined, including a LIDAR sensor of our own design to obtain measurements of the environment. In addition, a calibration of the sensors and actuators that make up the robot will be carried out, with the aim of achieving a controllable system.

Subsequently, the programming of the system will be developed. There are three main processes: the exploration of the robot, which will be carried out in real time using FreeRTOS in C++; the construction and generation of the map programmed in Python on a Raspberry Pi; and the control of the system, using a self-developed Android application in Java.

Keywords: ESP32, FreeRTOS, MQTT, Android, Raspberry Pi, Occupancy Grid Map, LIDAR

Índice de contenidos

Índice de figuras

CAPÍTULO 1

Introducción

1.1. Antecedentes y objeto

Actualmente, el avance de la robótica está creciendo exponencialmente y una parte fundamental de ella es la robótica móvil. Un robot móvil es un sistema capaz de moverse de forma autónoma y realizar determinadas acciones en tiempo real. Una aplicación interesante de los robots móviles es la navegación y creación de mapas de entornos. Numerosas empresas hacen uso de estas técnicas para obtener el mapa de un entorno desconocido, desde las más conocidas como Roomba, para llevar a cabo la limpieza de una zona, hasta robots de rescate, para el mapeado de entornos de alto riesgo y difícil acceso humano.

El Grado de Ingeniería enfrenta al alumnado a las diferentes áreas de conocimiento de forma aislada. Este proyecto une muchos de estos conocimientos con el fin de construir desde cero un robot capaz de explorar un entorno y generar un mapa en tiempo real, abordando desde el diseño y construcción del hardware, con la calibración necesaria del material, hasta el desarrollo del software de navegación y la representación gráfica del resultado.

El desarrollo del proyecto llevará a afianzar los conocimientos de control y programación de robots, en concreto en el sector de robótica móvil, así como la electrónica y el empleo de protocolos de comunicaciones. Además, se adquirirán conocimientos de nuevos lenguajes de programación.

1.2. Objetivos

El objetivo general del proyecto es afrontar todas las etapas necesarias para construir un robot semiautónomo capaz de generar mapas de un entorno, sin centrarse por completo en una de las tareas necesarias.

La robótica actual ofrece herramientas de alto nivel para facilitar el trabajo con robots a diferentes niveles, como por ejemplo *ROS (Robot Operating System)*[1], que ofrece un marco en el que compartir experiencias entre desarrolladores, y facilita diferentes tareas asociadas al trabajo con robots, permitiendo así especializar el trabajo en alguna fase del ciclo de trabajo con un robot. Sin embargo, nuestro objetivo es afrontar todas las etapas necesarias para construir un robot semiautónomo capaz de generar mapas de su entorno,

sin centrarnos únicamente en una de las tareas para ello. Por ese motivo, además, no se estudiaran todas las posibles estrategias de exploración, métodos de representación o protocolos de comunicación disponibles, sino que centraremos el proyecto en realizar el trabajo necesario para obtener el resultado final, en concreto:

- Diseñar y construir un robot móvil que pueda explorar el entorno, obteniendo y transmitiendo sus lecturas al proceso encargado de generar el mapa. Se elegirán los componentes que formarán el robot y se diseñaran aquellos que sean necesarios.
- Calibrar todos sus elementos de forma que su comportamiento sea fiable y controlable.
- Desarrollar un algoritmo que permita la exploración y obtención de medidas del entorno en tiempo real.
- Representar las medidas procedentes del robot en un mapa de probabilidad que reproduzca el entorno explorado.
- Crear una aplicación que muestre el resultado del mapa, transmitido por el servidor, en un dispositivo de control.
- Comunicar los distintos procesos entre sí usando un protocolo de mensajería.

Para la exploración del entorno se diseñará un robot terrestre dotado de una tecnología de desarrollo propio, alternativa al *LIDAR (Laser Imaging Detection And Ranging)* para el sensorizado. Se programará en *C++* sobre el microcontrolador *ESP32*, usando un sistema operativo en tiempo real (*FreeRTOS*). Las medidas tomadas por el robot se transmitirán por un servidor, en concreto una *Raspberry Pi 3B+*, usando el protocolo *MQTT*. Estas lecturas se usarán para crear un mapa de probabilidad, programado en *Python*, que se almacenará también en la *Raspberry Pi* y se retransmitirán a un dispositivo móvil de control, que mostrará el resultado sobre un aplicación de desarrollo propio, sobre *Java* en *Android Studio*, con la que se podrá también dirigir al robot.

1.3. Estructura de la Memoria

En esta memoria se recogen las distintas fases de trabajo que se han llevado a cabo en el proyecto. En el capítulo 2 se detalla cómo se pretende afrontar el propósito del proyecto: la metodología, tecnologías y herramientas empleadas en el proceso. En el capítulo 3 se describe el análisis de los requisitos que debe cumplir el robot, de su comportamiento y la arquitectura del sistema. En el capítulo 4 se expone el diseño del hardware empleado y la construcción del robot, la calibración se desarrolla en el capítulo 5 y la programación del sistema en el capítulo 6. Por último, en el capítulo 7 se presentan las pruebas y resultados obtenidos junto a las conclusiones y líneas futuras.

Los apéndices muestran el detalle completo de algunos de los elementos que conforman el proyecto, como son la programación de las distintas partes o el código de programación para la calibración.

CAPÍTULO 2

Metodología, Tecnologías y Herramientas Utilizadas

2.1. Metodología

El objetivo principal del proyecto es desarrollar las distintas etapas que se llevan a cabo para la construcción de un robot semiautónomo de exploración y representación de entornos, para los que se establecen tres etapas principales:

1. Definir el diseño del hardware que permita la exploración.
2. Probar el funcionamiento de sus componentes y realizar la calibración necesarias para que las medidas sean suficientemente fiables.
3. Elaborar los algoritmos necesarios que permitan explorar de forma semiautónoma un entorno, representarlo en un mapa probabilístico y poder actuar sobre su comportamiento por medio de una aplicación móvil.

A continuación, se detallarán los pasos a seguir, y posteriormente, se mostrarán las tecnologías y herramientas utilizadas y su justificación.

Primera fase

En primer lugar se construirá un robot capaz de obtener las mediciones adecuadas de la zona en exploración. Ello supone decidir el tipo de movimiento, los motores a utilizar y el resto de componentes del mismo, entre los que se incluirá un *LIDAR* propio de bajo coste, que se desarrollará en base a componentes tanto comerciales como desarrollados expresamente mediante *SolidWorks*. Se deberán seleccionar los sensores necesarios y un microcontrolador que permita la ejecución de tareas en tiempo real. Una vez decididos los componentes, se procederá a elaborar el esquema de conexiones, usando *Fritzing* para el posterior montaje y cableado.

Segunda fase

Este proyecto combina la parte hardware y software, y ello supone que sea imprescindible comprobar el funcionamiento real de los componentes del robot y calibrarlos para que el sistema sea controlable. Para poder generar un mapa, el robot debe conocer su posición y tomar medidas con precisión. Si el sistema no esta calibrado correctamente, el resultado no será fiable y a medida que siga explorando la probabilidad de error aumentará. Por tanto, se debe conocer la relación entre la realidad y los valores obtenidos por el robot, tratando

de reducir el error al mínimo. Es decir, si no es posible reducir el error a cero, se debe conocer y compensar el error sistemático de forma que se tenga control sobre el sistema. Por tanto, en esta fase se desarrolla el código necesario para el control de los sensores y actuadores del robot y se realizan pruebas para determinar y ajustar el comportamiento.

Tercera fase

En esta fase se desarrollarán los tres procesos principales del proyecto: la exploración, la construcción del mapa y el control del robot. Éstos se desarrollaran en tres dispositivos diferentes que se comunicarán usando el protocolo de mensajería *MQTT* implementado por *Eclicpse Mosquittto* sobre una *Raspberry Pi*.

Para la exploración se programará un algoritmo que permite al robot navegar por el entorno, tomando decisiones por si mismo, si el modo de navegación elegido por el usuario es navegación autónoma, o por el contrario ejecutar la órdenes que le indique el usuario, si el modo es manual. La selección del modo la recibirá el robot desde el proceso de control. Además, el algoritmo debe tomar medidas del entorno realizando un barrido 360º del *LIDAR*, que transmitirá la posición del robot junto a su lectura de forma concurrente. El manejo de distintas tareas en tiempo real nos lleva a usar una tecnología que lo permita, eligiendo así como sistema operativo *FreeRTOS* en *C++*.

Estas medidas se utilizaran como entrada del proceso de construcción del mapa, elaborado en *Pyhton* en el entorno de *Spyder*. Para ello, se utilizará un método probabilístico que represente el entorno explorado en función de la probabilidad de que exista o no un obstáculo. El resultado se transmitirá al proceso de control.

Para el proceso de control del robot, se creará una aplicación sobre *Android Studio* en *Java* para un dispositivo móvil, de manera que el usuario pueda visualizar el resultado mientras se realiza la exploración e intervenir en ella. El usuario podrá decidir entre los modos de exploración, "autónoma" o "manual", e incluso cancelar el movimiento del robot y devolver al *LIDAR* a su origen, si lo desea. Además, tendrá la capacidad de decidir si es necesaria la generación del mapa o se prefiere explorar la zona sin realizar una representación del entorno.

2.2. Tecnologías Empleadas

2.2.1. MQTT

Para la comunicación del sistema se ha decidido utilizar el protocolo *MQTT*[2], un protocolo de mensajería M2M (Machine-to-machine)[3], ejecutado sobre *TCP/IP*. Este sistema de comunicación es muy ligero, ya que requiere un ancho de banda mínimo (ideal para conexiones de Internet con baja calidad), sencillo y de bajo consumo. Además, se basa en la topología *Publish/Subscribe*, aportando las ventajas de escalabilidad, asincronismo y desacoplamiento entre clientes. Esto lo convierte en una de los principales protocolos de comunicación de dispositivos de *IoT (Internet of Things)* y por tanto, la opción elegida para transferir la información entre la exploración del entorno, la construcción del mapa y el control del robot.

Esta compuesto por el *broker* y los clientes. El *broker* funciona como intermediario entre los clientes, es decir, recibe los mensajes de los clientes remitentes, llamados *publisher*, y

los retransmite a los clientes destinatarios correspondientes, llamados *subscriber*. El *broker* solo actuará como intermediario, sin almacenar los mensajes recibidos. Para poder filtrar los mensajes y solo retransmitir al cliente correspondiente, éstos deben indicarle al *broker* que se han subscrito a los temas o *topics*[4] de los que desean obtener mensajes. Por tanto, si un cliente envía un mensaje éste debe publicarlo al *broker* sobre un *topic* y todos aquellos clientes que se hayan suscrito recibirán su contenido, manteniendo una conexión "'abierta" que se reutiliza en cada comunicación.

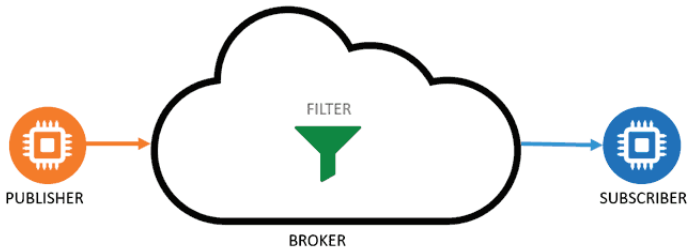


Figura 2.1: Comunicación en MQTT.

2.2.2. FreeRTOS

La exploración del robot se realizará en tiempo real, siendo posible realizar de forma concurrente la lectura del entorno, el desplazamiento y la transmisión de datos. Por tanto, el sistema operativo que se empleará será *FreeRTOS (Real Time Operting System, RTOS)*.

FreeRTOS[5] es un sistema operativo en tiempo real y *open-source*, figura 2.2. Su código esta programado en lenguaje C y proporciona métodos de concurrencia en base a prioridades, es decir, permite ejecutar tareas a la vez. El número de tareas simultáneas está limitado únicamente por el hardware y el número de núcleos que contenga.

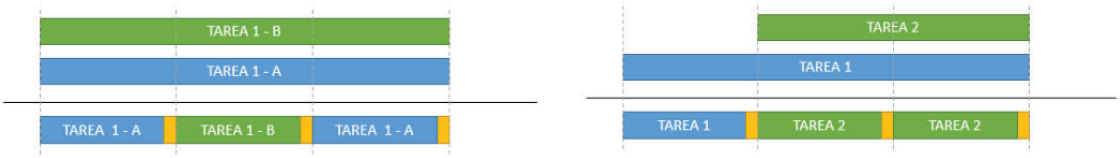
Figura 2.2: FreeRTOS.

La principal característica es la planificación, encargada de decidir la tarea que debe ejecutarse en cada momento, entre las que se encuentren con estado "listo". Para ello se basa en el orden de prioridad de las tareas involucradas, un número entero asignado en su creación. Si existen varias tareas con el mismo nivel de prioridad se sigue el modelo "*Round-Robin*"[6], que permite que cada proceso se ejecute durante un periodo de tiempo finito llamado "cuanto de tiempo". Las características de éste modelo son:

- Los procesos se gestionan en colas circulares, agregándose siempre al final de la misma.
- La cantidad de tiempo que se ejecuta un proceso está limitada, evitando que una tarea no llegue a ejecutarse por falta de recursos.

- Transcurrido el "cuanto de tiempo" del proceso, éste vuelve al final de la cola.
- Si el proceso se ha completado se elimina de la cola, disminuyendo así el tiempo de espera.

Un ejemplo de planificación es el siguiente, figura 2.3 a). Ambas tareas poseen la misma prioridad y se ejecutan alternativamente. Otro ejemplo se observa en la figura 2.3 b), ahora la tarea 2 tiene mayor prioridad por lo que se ejecutará antes. El tiempo en amarillo de ambas imágenes es el tiempo necesario para que el planificador cambie la tarea.



(a) (b)

Figura 2.3: Ejemplos planificación FreeRTOS

2.2.3. Representación del entorno

En robótica móvil de exploración, si el entorno es desconocido y se desea representar aparece el fenómeno del *SLAM (Simultaneous Localization And Mapping)*[7]. Se trata de una técnica muy usada para generar la representación de un entorno desconocido a la vez que se navega en él estimando la posición del robot. Su objetivo es resolver el problema de la incertidumbre en la lectura del entorno dependiente de la incertidumbre de la posición desde la que se realiza. Para ello se aplican métodos probabilísticos[8] basados en el *teorema de Bayes*[9], como los *mapas de ocupación de rejilla* que se utilizarán en este proyecto. Además, estos métodos estiman la posición del robot a partir de la información procedente de sus sensores. Los sensores principales que se utilizan son las cámaras (*Visual Odometry)* y los *LIDAR (Light Detection and Ranging*. Este proyecto se centrará en este último.

Occupancy Grid Map

Para la representación del entorno existen diferentes técnicas, en función de la interpretación[10]: mapas métricos, por descripción basaba en geometría, y mapas topológicos, basados en una descripción cualitativa. Un mapa de ocupación, *Occupancy Grid Map*[11] (figura 2.4), se clasifica como un mapa métrico de descomposición espacial. Se trata de la representación espacial del entorno de trabajo del robot dividida en particiones iguales, formando una matriz bidimensional. Los valores de cada una de las celdas de la matriz representan la probabilidad de ocupación, es decir, de la existencia de un obstáculo (*Teorema de Bayes*). Las celdas son independientes entre sí y su probabilidad toma un valor entre 0 y 1. Para ello se hace uso de la información recogida de sensores y es necesario conocer la posición del robot.

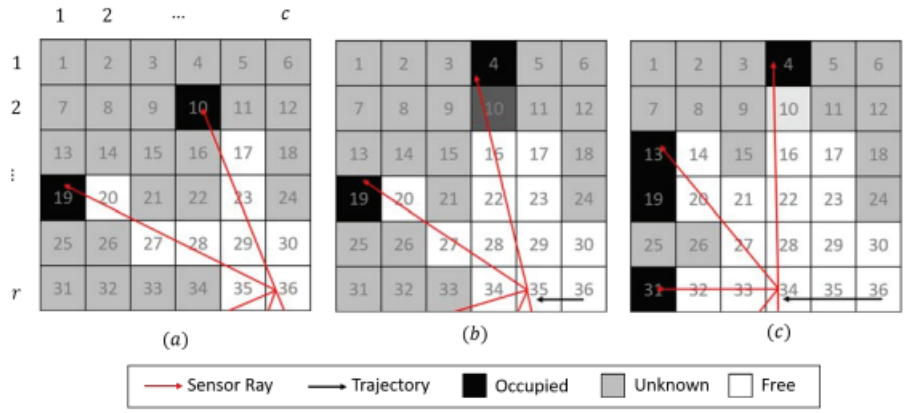


Figura 2.4: Mapa de ocupación.

LIDAR

Un *LIDAR (Laser Imaging Detection And Ranging)*[12] (figura 2.5) es un dispositivo de medición y detección de objetos mediante un láser infrarrojos. Consiste en un foco emisor de haces de rayos láser infrarrojos y una lente receptora. El haz de luz rebota al encontrar un objeto y es captado por el receptor. El retraso desde que se emite hasta que se recibe se conoce como *tiempo de vuelo* y a partir de éste el sensor obtiene la distancia, al igual que lo hacen los sensores ultrasonidos. Los sensores infrarrojos se basan en la propagación de la luz, en lugar del sonido. Ello los que los convierte en una opción mucho más rápida, además de tener mayor precisión. Entre ellos existen *LIDAR* fijos y 360, que realiza un barrido de 360º sobre si mismo. De esta forma permite trabajar en aplicaciones de alta velocidad. No obstante, sus principales inconvenientes son la distancia máxima del láser y su elevado precio.

Figura 2.5: Ejemplo de LIDAR comercial.

2.3. Herramientas Empleadas

2.3.1. Arduino IDE

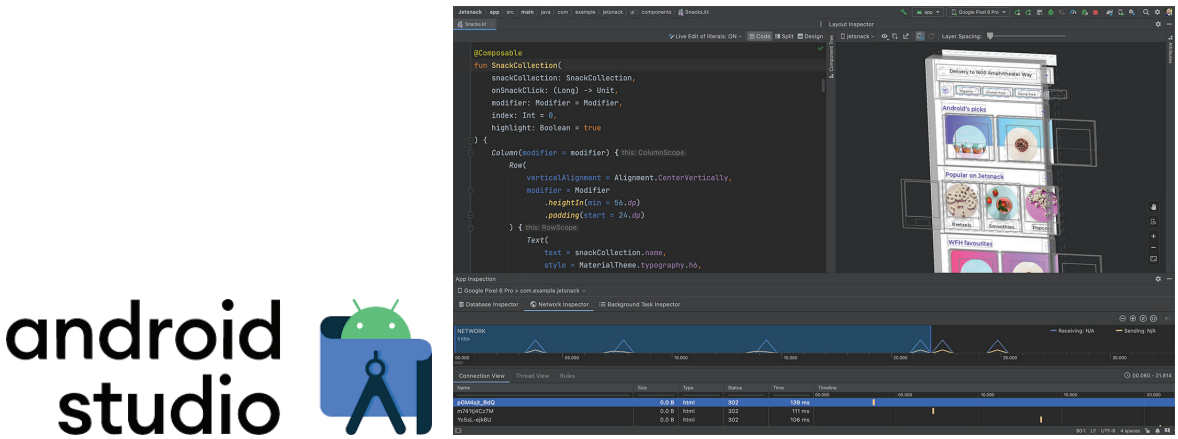
Para el desarrollo del robot se ha hecho uso del entorno de programación *Arduino IDE (Integrated Development Evironment)*[13] (Figura 2.6). Se trata de una aplicación *open-source* multiplataforma, que utiliza como lenguaje *C* o *C++*. Permite compilar y cargar el programa en un microcontrolador compatible usando comunicación serie y admite también *FreeRTOS* por lo que es la opción elegida.

(a) (b)

Figura 2.6: Arduino IDE.

2.3.2. Android Studio

Entorno de desarrollo integrado oficial para la creación de aplicaciones Android[14] (remplazando a *Eclipse*[15]), sobre el que se programará la aplicación de control del robot (Figura 2.7). Como lenguaje principal de programación utiliza *Kotlin*, no obstante admite otros como *Java* o *C++*. Proporciona un editor de diseño de la interfaz gráfica y plantillas. Además, este IDE permite simular el comportamiento en un dispositivo virtual.

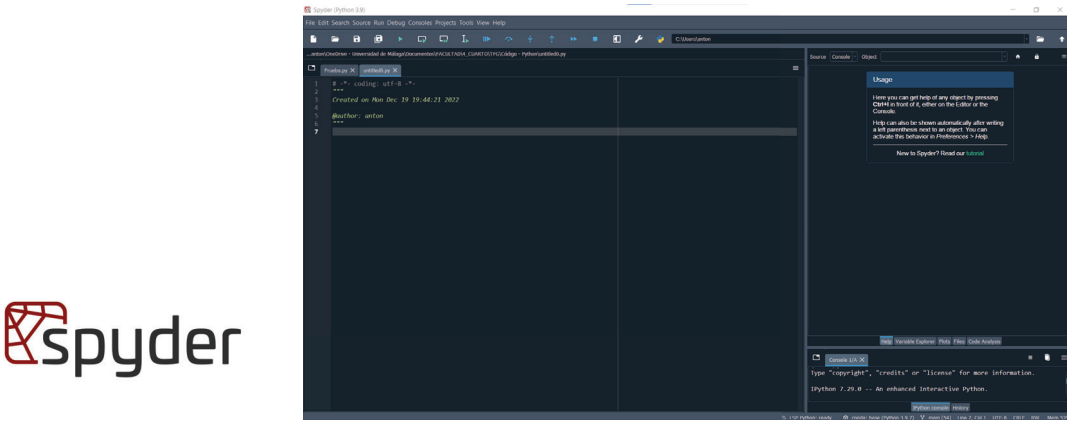


(a) (b)

Figura 2.7: Android Studio.

2.3.3. Spyder

Se trata de un entorno de programación[16] *open-source* multiplataforma que utiliza como lenguaje de programación *Pyhton* (figura 2.8). Se usará para desarrollar el algoritmo de construcción del mapa, debido a que este lenguaje contiene diferentes librerías de procesamiento de imágenes como *Numpy, OpenCV, SciPy* o *Python Imaging Library (PIL)*[17].



(a) (b)

Figura 2.8: Sypder IDE.

2.3.4. Mosquitto

Eclipse Mosquitto[18] (figura 2.9) es un broker, un intermediario, de mensajería *open-source* y multiplataforma, que implementa el protocolo *MQTT*. Una de sus grandes ventajas es que debido a su ligereza es muy apropiado para realizar la comunicación entre diferentes dispositivos *IoT*. Su instalación se recoge en el apéndice B.

Figura 2.9: Bróker Mosquitto MQTT.

2.3.5. Fritzing

Programa de diseño electrónico[19] que permite crear prototipos basados en Arduino y circuitos impresos (figura 2.10). Se empleará para construir el circuito del robot.

Figura 2.10: Logo Fritzing

2.3.6. SolidWorks

Software de modelado y diseño CAD[20] de piezas mecánicas que permitirá diseñar las piezas 3D que usarán en el proyecto (figura 2.11).

Figura 2.11: Logo SolidWorks

2.3.7. Raspberry Pi

Se trata de un SBC[21] (Single Board Computer), un ordenador de placa única que contiene todos o la mayor parte de los componentes de un ordenador entre las cuales destacan el microprocesador, la memoria y las entradas y salidas. Estos dispositivos son atractivos por su reducido tamaño y bajo precio. El sistema operativo se carga en una tarjeta MicroSD, apéndice A, posee conectividad USB, HDMI, Ethernet, WIFi doble banda y bluetooth y numerosos pines de entrada y salida. En este proyecto se empleará una Raspberry Pi versión 3B+, figura 2.12, como dispositivo sobre el que se instalará el servidor de *MQTT*

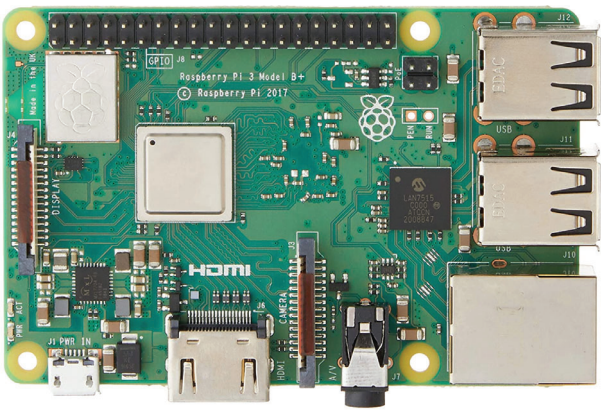


Figura 2.12: Raspberry Pi 3B+.

CAPÍTULO 3

Análisis del sistema

Como trabajo previo al diseño del hardware del robot y al desarrollo software, se realiza un análisis del sistema a construir. En primer lugar, se determinan las características de los componentes necesarios para su funcionamiento, una vez definidas se realiza el esquema de arquitectura del sistema y por último se realiza un análisis del comportamiento del sistema, con los requisitos y casos de uso.

3.1. Análisis de componentes

En esta sección se realiza un análisis de los distintos características que tiene que cumplir el robot para lograr el objetivo:

- El robot debe contener actuadores con la capacidad de trasladar al robot por el entorno.
- El sistema de movimiento del robot debe permitir el desplazamiento sobre superficies terrestres planas.
- El sistema de locomoción debe asegurar la estabilidad del robot.
- El movimiento del robot debe permitir conocer la posición del robot de forma precisa.
- El robot debe contener sensores que permitan tomar medidas de distancias del entorno en todo el plano.
- El microcontrolador del robot tiene que permitir la ejecución en tiempo real de procesos.
- Se usarán componentes de bajo coste para la construcción del robot.
- La alimentación del robot debe ser portátil y recargable.

3.2. Arquitectura del sistema

Un sistema es un conjunto de elementos y procesos relacionados entre sí, cuyo comportamiento tiene como objetivo satisfacer unas necesidades[22]. Para ello se opera sobre entradas, obteniendo unas salidas procesadas e interactuando con el entorno.

Los procesos y elementos físicos principales del sistema son: la exploración del entorno, realizada por el robot con un microcontrolador que trabaje en tiempo real y procese la lectura de sensores (*LIDAR*) y la operación de sus actuadores, que corresponde a las entradas; la construcción y representación del mapa por un cartógrafo, que será la salida; y la teleoperación del robot, es decir, su control y visualización del resultado, realizado sobre una aplicación en un dispositivo móvil. Su comunicación se realizará vía WiFi, que será el núcleo del sistema, siguiendo el protocolo *MQTT* formado por un broker alojado en un microcontrolador. La arquitectura del proyecto se representa en la figura 3.1.

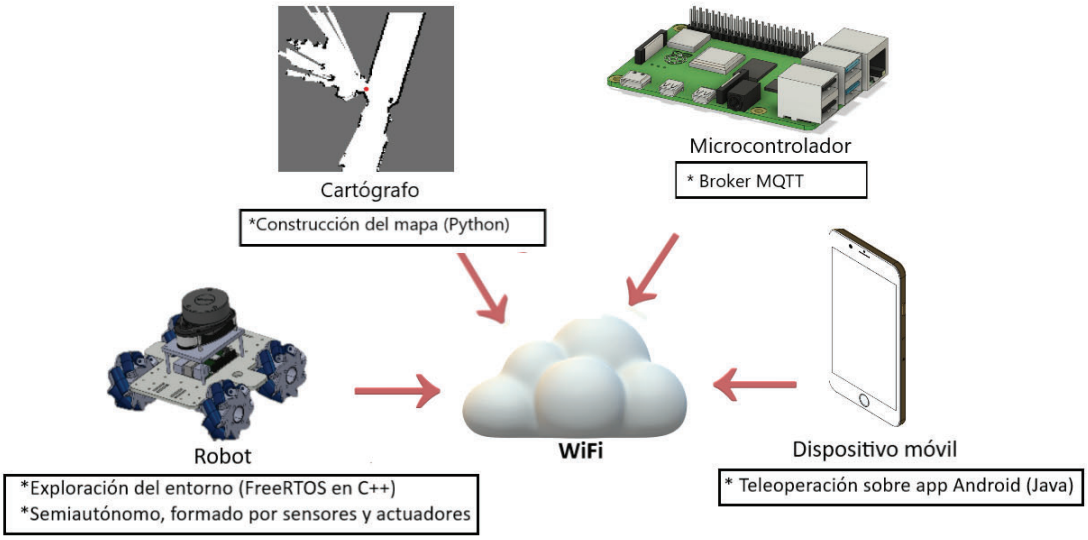


Figura 3.1: Arquitectura básica del sistema.

3.3. Análisis del comportamiento

El comportamiento del sistema de organiza en dos modos de operación diferentes y en ambos, el robot tomará medidas del entorno, que reflejará en un mapa de probabilidades si el usuario lo desea:

- Modo autónomo, donde el robot toma sus propias decisiones respecto a los cambios de dirección que debe realizar durante la exploración.
- Modo guiado (manual), donde el usuario es el encargado de indicar la dirección mediante comandos desde el dispositivo móvil.

A continuación, se exponen los documentos realizados para el análisis de su comportamiento: los casos de uso, requisitos funcionales y no funcionales del sistema.

3.3.1. Casos de Uso

Un caso de uso es la descripción de una actividad realizada por un actor para llevar a acabo un proceso[23]. Es decir, la secuencia de interacciones que se desarrollan entre el sistema y sus actores en respuesta a un evento iniciado por un actor principal. Para ello se emplean descripciones textuales, conocidas como especificaciones de casos de uso, o gráficas, llamadas diagramas de casos de uso. En este proyecto se ha optado por la especificación de casos de uso.

El objetivo es detallar la comunicación y el comportamiento, la relación entre los actores y la funcionalidad del sistema.. Este sistema contempla dos actores: el operador, la persona encargada de inicializar el sistema que no necesariamente será la que controle la exploración, y el usuario, la persona que sí manejará la exploración desde el dispositivo de control.

Por último, el desarrollo de los casos de uso será fundamental para definir los requisitos funcionales del sistema.

Caso 0 - Inicialización del sistema (CU00)

Actor Operador.

Precondición Sistema apagado.

Postcondición Sistema encendido e inicializado.

Curso Normal

1. *Operador* - Sitúa el robot en el punto de partida.
2. *Operador* - Enciende todos los dispositivos implicados
3. *Sistema* - Se inicializa y queda listo para comenzar a operar

Caso 1 - Inicio de la exploración (CU01)

Actor Usuario.

Precondición El sistema debe estar inicializado (CU00).

Postcondición El robot está explorando su entorno.

Curso Normal

1. *Usuario* - Elige el modo deseado.
2. *Usuario* - Pulsa botón de comienzo de exploración
3. *Sistema* - Comienza la exploración. El robot recorre la zona atendiendo a la configuración de la exploración y el sistema va confeccionando el mapa del área.

Caso 2 - Pausa de la exploración (CU02)

Actor Usuario

Postcondición El sistema está en pausa.

Curso Normal

1. *Usuario* - Solicita una pausa de la exploración.
2. *Sistema* - El robot detiene sus actuadores y entra en pausa, sin cancelar la exploración actual.

Caso 3 - Reanudación de la exploración (CU03)

Actor Usuario

Precondición El sistema está en pausa.

Postcondición El robot está explorando su entorno.

Curso Normal

1. *Usuario* - Solicita la reanudación de la exploración.
2. *Sistema* - El robot reactiva sus actuadores y continúa la exploración en el modo indicado.

Caso 4 - Cancelación de la exploración (CU04)

Actor Usuario.

Postcondición El robot está parado, el *LIDAR* vuelve a la posición inicial y la generación de mapas queda dehabilitada.

Curso Normal

1. *Usuario* - Solicita abortar la exploración.
2. *Sistema* - Detiene su movimiento, devuelve el LIDAR a la posición inicial de 0^{o}. Desactiva el modo actual y desactiva la generación de mapa.

Caso 5 - Cambio de modo (CU05)

Actor Usuario

Postcondición El sistema muestra el modo actual y el robot está detenido.

Curso Normal

1. *Usuario* - Pulsa el modo deseado.
2. *Sistema* - El robot detiene sus actuadores y entra en pausa, a espera de indicación de inicio del modo indicado (CU01).

Comentarios El robot se detiene independientemente de si el usuario cambia de modo sin realizar una pausa previa.

Caso 6 - Teleoperación (CU06)

Actor Usuario.

Precondición Modo navegación seleccionado.

Postcondición El sistema muestra el modo navegación y el robot se encuentra desplazado en la dirección indicada.

Curso Normal

1. *Usuario* - Pulsa el comando de movimiento.
2. *Sistema* - El robot realiza el movimiento en la dirección solicitada mientras realiza la lectura del entorno.

Caso 7 - Activar generación de mapa (CU07)

Actor Usuario.

Postcondición El sistema dibuja el resultado de la lectura del entorno actual.

Curso Normal

1. *Usuario* - Activa la generación de mapa.
2. *Sistema* - Comienza a dibujar en el mapa la lecturas tomadas sobre el entorno y la posición actual del robot.

Caso 8 - Desactivar generación de mapa (CU08)

Actor Usuario.

Precondición El sistema debe estar inicializado (CU00).

Postcondición El sistema detiene la creación del mapa.

Resumen El usuario desactiva la generación de mapa. El sistema solo marca la posición actual del robot sobre el último mapa guardado.

Curso Normal

1. *Usuario* - Desactiva la generación de mapa
2. *Sistema* - Muestra únicamente sobre el mapa la posición actual del robot y el robot continúa tomando medidas únicamente para evitar obstáculos.

3.3.2. Requisitos funcionales

Un requisito funcional es la descripción del servicio o función que debe cumplir el software, detalles técnicos, interacción con el entorno y su estado[24]. Para obtener los requisitos funcionales es necesario el desarrollo de los casos de uso del sistema.

Sistema

RF01: La comunicación entre los elementos del sistema se realizará mediante el protocolo MQTT.

Robot

RF02: Debe ser capaz de identificar el comando de movimiento enviado por el usuario.

RF03: Debe permitir al usuario devolver a la posición inicial al *LIDAR*.

Mapa

RF04: Se debe construir el mapa de ocupación probabilístico de la zona actual en la que se encuentra el robot.

RF05: Distinguir entre un obstáculo, una zona libre de obstáculos y zona desconocida en el mapa de ocupación.

Control

RF06: El sistema debe permitir al usuario modificar el modo de funcionamiento entre navegación (manual) y exploración (automático).

RF07: En modo navegación el sistema debe permitir al usuario seleccionar la dirección deseada.

RF08: El sistema debe permitir al usuario abortar la actividad actual.

RF09: El sistema debe permitir elegir al usuario si desea o no generar mapa.

RF10: El sistema debe permitir al usuario pausar y reanudar la actividad.

3.3.3. Requisitos no funcionales

Se define como requisito no funcional, aquel que describe las características generales de funcionamiento y restricciones del sistema[25].

RFN01: Se construirá un robot de bajo coste.

RFN02: El número de usuarios que podrá influir en el control del robot estará limitado a uno.

RFN03: Debe ser fácilmente escalable, como permitir la incorporación de nuevos modos de funcionamiento.

RFN04: El entorno de funcionamiento del robot se limita a zonas interiores con terreno regular.

RFN05: La diferencia entre tonalidades del mapa debe ser clara y fiable, representando así la probabilidad de obstáculos.

RFN06: La aplicación de control debe ser sencilla e intuitiva para el usuario.

RFN07: Los giros en modo navegación deben permitir al usuario alcanzar cualquier zona accesible.

RFN08: Se debe evitar el sobrecalentamiento de los dispositivos, como la Raspberry Pi y los controladores de los motores A4988.

RFN09: El control del movimiento del robot y del *LIDAR* debe tener un control de precisión exigente.

CAPÍTULO 4

Diseño Hardware

En esta fase se decidirán los componentes del robot, se diseñarán los elementos necesarios y se realizará el conexionado y montaje del robot. En el capítulo 2 se han expuesto las tecnologías que se emplearán para el desarrollo del proyecto, a continuación se detallarán los componentes específicos del robot

4.1. Movimiento

Se define la locomoción como la facultad para desplazarse de un lugar a otro[26]. Para un robot terrestre existen 3 grandes grupos de locomoción: con ruedas, patas y orugas. Los robots con orugas tienen como ventaja la posibilidad de moverse sobre cualquier superficie, incluso subiendo y bajando escaleras. No obstante, debido a su alto coste en este proyecto se ha descartado esta opción. Por otro lado, los robots con patas se usan en terrenos irregulares, debido a la gran estabilidad que proporcionan, sin embargo, conllevan un mayor consumo. Este proyecto se desarrollará para un terreno regular, descartando la opción anterior, por lo que el tipo de locomoción elegida en este proyecto es un robot con ruedas.

Dentro de este grupo existen distintos tipos de ruedas[27] (Figura 4.1):

- Motriz: Unidas al motor proporcionando la tracción para el avance del robot.
- Directriz: Encargadas del control de la dirección de orientación del movimiento.
- Fijas: No aportan tracción y solo giran en torno a su eje.
- Suecas: Ruedas omnidireccionales, es decir, permiten tanto el avance hacia delante y detrás como hacia los lados debido a los cilindros que forman la rueda.
- Castor: Orientables, centradas o descentradas, sin control de dirección ni tracción.

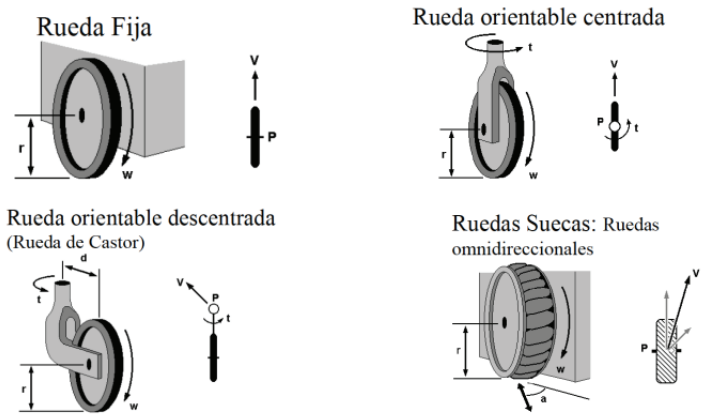


Figura 4.1: Tipos de ruedas.

Para este proyecto se usarán 2 ruedas fijas motrices y se empleará, como apoyo para mantener la estabilidad y para facilitar el giro, una rueda orientable centrada también conocida como "rueda loca".

Motores

Uno de los objetivos del sistema es controlar la posición del robot con gran exactitud. Ésta será el origen de las medidas y por tanto necesaria para no encontrarse perdidos dentro de un entorno desconocido. Se plantean dos posibles opciones para el desplazamiento del robot.

Por un lado, el motor de corriente continua *(CC)*[28] con un reductor interno incorporado (Figura 4.2) que proporciona alta velocidad de giro y un bajo torque. El efecto del reductor provocará que aumente el par y disminuya la velocidad. Su principal desventaja es el control de posición, por lo que suelen incorporar un encoder, que permite conocer las vueltas que da el motor. No obstante, el propósito de este proyecto, como ya se ha comentado, es la exploración de un entorno, para la que es muy importante la precisión, por lo que se descarta esta opción.

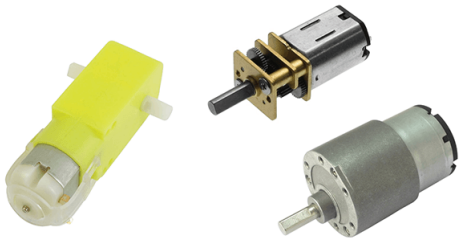


Figura 4.2: Motor DC.

Por otro lado, el motor paso a paso[29] (Figura 4.3) esta formado por un estátor con dos bobinas desfasadas 90^{o} y un imán permanente soldado al eje como rotor. El giro del motor se realiza aplicando pulsos eléctricos en una secuencia correcta en las bobinas que provocará que el imán se oriente realizando giros o "pasos". Además, con esta secuencia también se controla la dirección de giro, y en función de la frecuencia de los pulsos eléctricos de entrada, se controla la velocidad de giro. Para aplicar esta secuencia, estos motores emplean un dispositivo de control, que también le aporta la alimentación, y los

convierte en motores muy precisos y controlables. Por tanto, a pesar de que no devuelvan información del giro, su gran precisión en lazo abierto los convierte en la opción elegida para este proyecto.

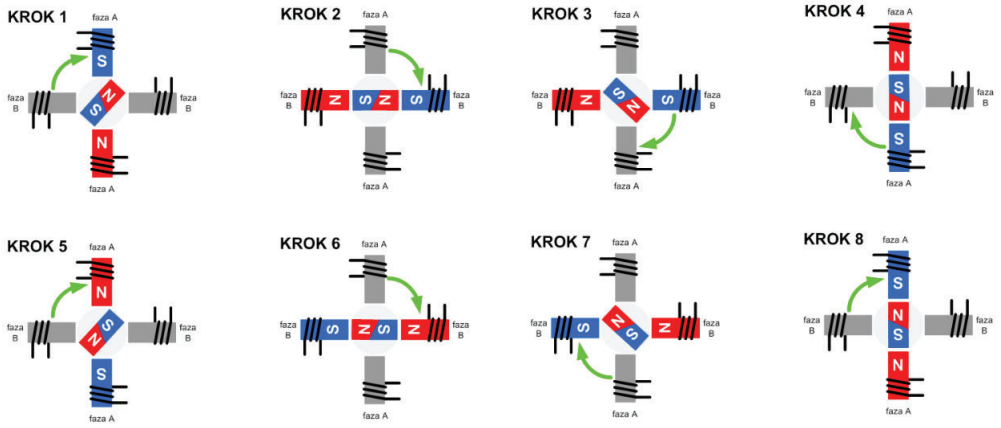
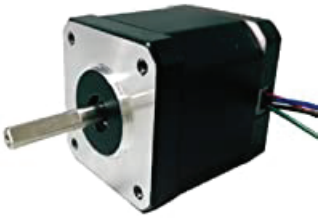


Figura 4.3: Motor paso a paso.

Por tanto, el tipo de motor motor paso a paso elegido es el motor bipolar Nema 17HS16-2004S (figura 4.4), con un paso de 1.8º y una precisión en el ángulo de paso del 5 %.

1.8° 42MM High Torque Hybrid Stepping Motor

Item	Specifications
Step Angle	1.8°
Step Angle Accuracy	±5% (full step, no load)
Resistance Accuracy	±10%
Inductance Accuracy	±20%
Temperatru Rise	80°CMax. (rated current,2 phase on)
Ambient Temperatuar	-20°C~+50°C
Insulation Resistance	100M?Min.,500VDC
Dielectric Strength	500VAC/ for one minute
Shaft Radial Play	0.02Max. (450 g-load)
Shaft Axial Play	0.08Max. (450 g-load)
Max. radial force	28N (20mm from the flange)
Max.axial force	10N



● 42MM Hybrid Stepping Motor Specificatiors

Model No	Rated Voltage	Current /phase	Resistance /phase	Inductance /phase	Holding Torque	#of Leads	Moment of inertia	Weinght	Orientation Torque	Length
	V	A	Ω	mH	Kg-cm		g-cm²	kg	g-cm	mm
17HS16-2004S	2.2	2	1.1	2.6	4.0	4	54	0.28	150	40

Figura 4.4: Motor paso a paso Nema 17HS16-2004S.

Como se ha comentado antes, estos motores paso a paso necesitan un controlador que ajuste la intensidad que le otorga al motor, basándose en *Puentes-H*. En este proyecto se emplea un A4988 (Figura 4.7), que posee un regulador de corriente con el que se limitará la corriente que pasa por la bobinas del motor. La intensidad máxima que se suministra se calcula siguiendo la "Ley de Ohm"[30], en función del voltaje de referencia (V_{ref}) y la resistencia del controlador, (R_s), fórmula 4.1.

$$I_{max} = \frac{V_{ref}}{8 \cdot R_s} \tag{4.1}$$

Para modificar la corriente, el A4988 contiene un potenciómetro con el que se controla la V_{ref}. Por tanto, sustituyendo los valores de I_{max} por 2A y el valor de R_s del controlador por 100mΩ, se obtiene la V_{ref} buscada (Fórmula 4.2).

$$V_{ref} = 8 \cdot R_s \cdot I_{max} = 8 \cdot 0{,}1 \cdot 2 = 1{,}6V \tag{4.2}$$

Finalmente, a este valor se le aplica un porcentaje de seguridad del 77.5 %, como se observa en la figura 4.5.

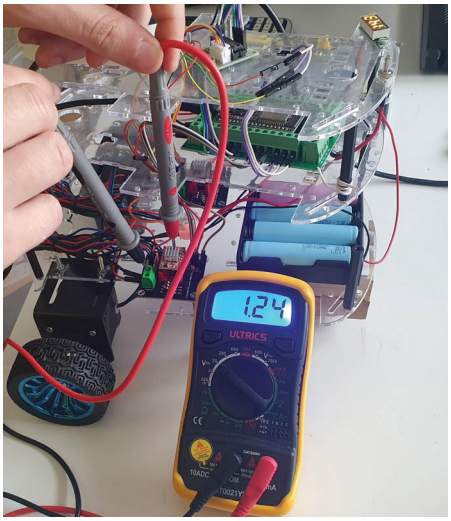


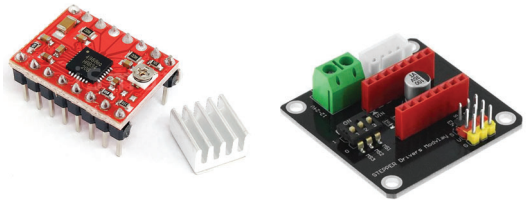
Figura 4.5: V_{ref} potenciómetro A4988 .

Por otro lado, el A4988 permite definir el sentido de giro y la cantidad de paso por sus entradas DIR y STEP. Y, además, cuenta con la técnica del *"microstepping"* (Figura 4.6)[31] que permite realizar pasos inferiores al paso nominal reduciéndolo 2^n partes con n∈[0,4], en función del estado de sus pines M1, M2 y M3, que provoca un aumento en la precisión en el paso.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Figura 4.6: Microstepping A49988

Un inconveniente del A4998 es la elevada temperatura que alcanza durante su funcionamiento. Por ello, se han acoplado aletas térmicas que disminuyan este problema.



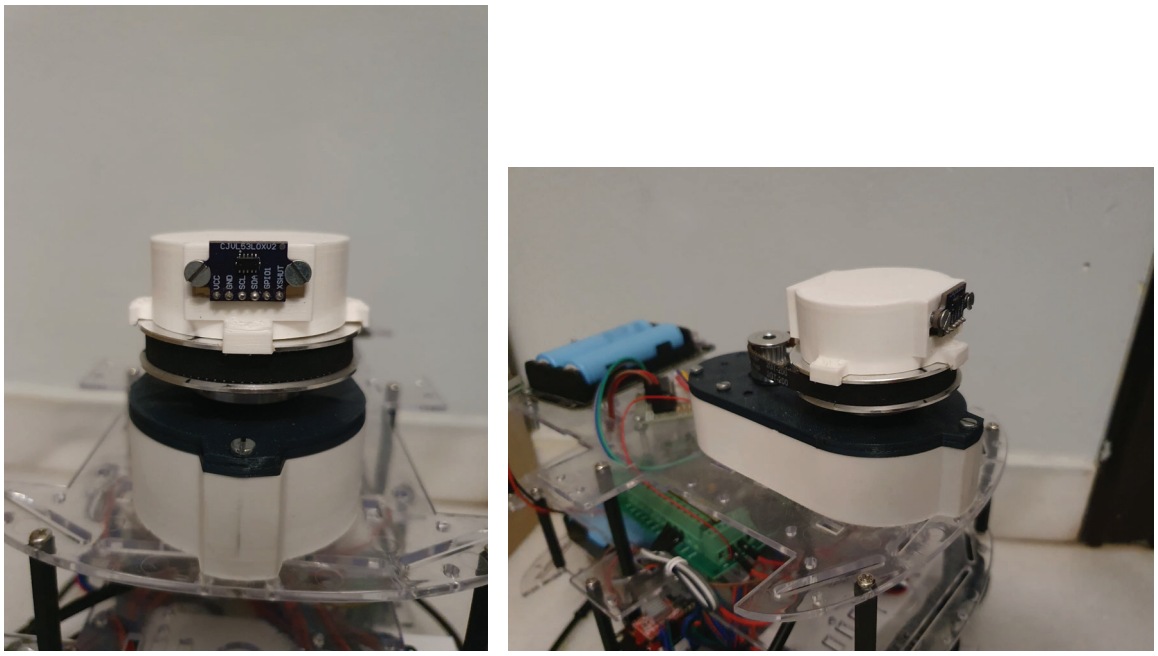
(a) (b)

Figura 4.7: A49988 y Módulo de control

4.2. Sensores

El objetivo del robot en este proyecto es la exploración del entorno evitando obstáculos de forma que pueda navegar por un entorno desconocido obteniendo información del mismo. Debido al alto coste de los dispositivos comerciales de esta tecnología, se ha decidido diseñar y construir un prototipo de diseño propio de *LIDAR* (Figura 4.8) que realizará un barrido 360º tomando medidas del entorno.

Si bien se podrían haber dotado de distintos sensores láser alrededor del robot, este prototipo de *LIDAR* permite realizar un barrido 360º con un sólo sensor lo que supone un ahorro en el coste de la construcción del robot y la elaboración de un sistema más sencillo. Este prototipo consta de varios componentes que se detallan a continuación.



(a) (b)

Figura 4.8: LIDAR diseño propio.

Estructura giratoria

En primer lugar se decidió tomar un diseño 3D ya creado por un autor[32], cuyo movimiento de barrido se transmite a través de una relación de poleas, como se observa en la figura 4.10. Este modelo permite tener un barrido completo de 360º, aislando por un lado el motor y por otro el giro de los cables del sensor, frente a modelos de otros autores[33] que realizan el giro sobre el propio motor creando una zona ciega en el barrido (Figura 4.9).

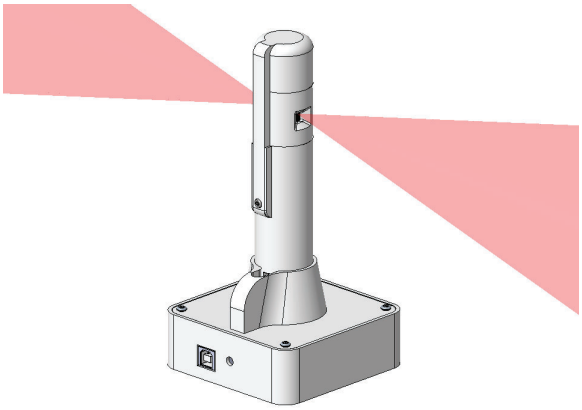
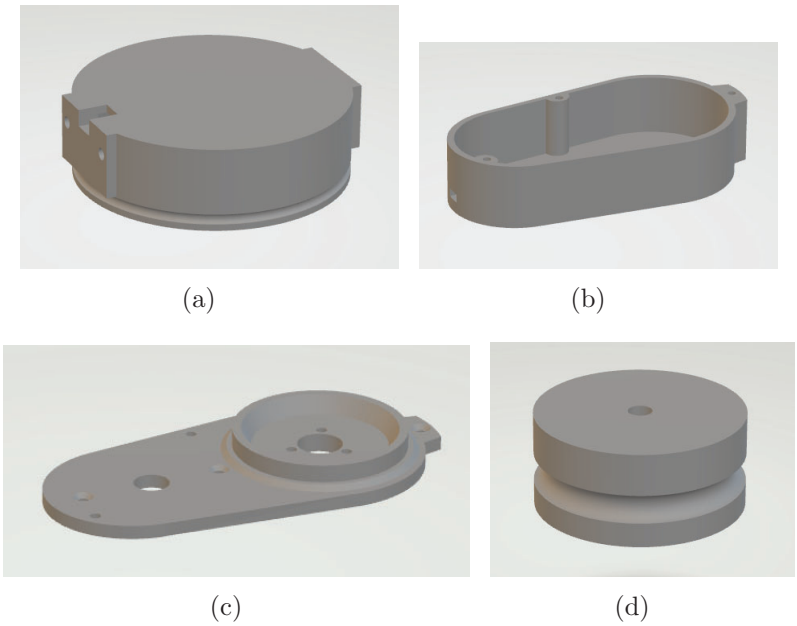


Figura 4.9: Estructura 3D LIDAR alternativa.



(a) (b)

(c) (d)

Figura 4.10: Estructura 3D LIDAR escogida.

Sin embargo, en este modelo se ha modificado la transmisión de movimiento (realizada por las poleas 3D), que provocaba imprecisiones acumulando error a medida que se aplicaban pasos al motor. La decisión tomada fue sustituir las poleas 3D por un juego de engranajes y correa, figura 4.11. De esta forma, con el empleo de dientes se consigue un movimiento más preciso basado en la relación de número de dientes, que para este juego de engranajes tiene valor de 4:1.

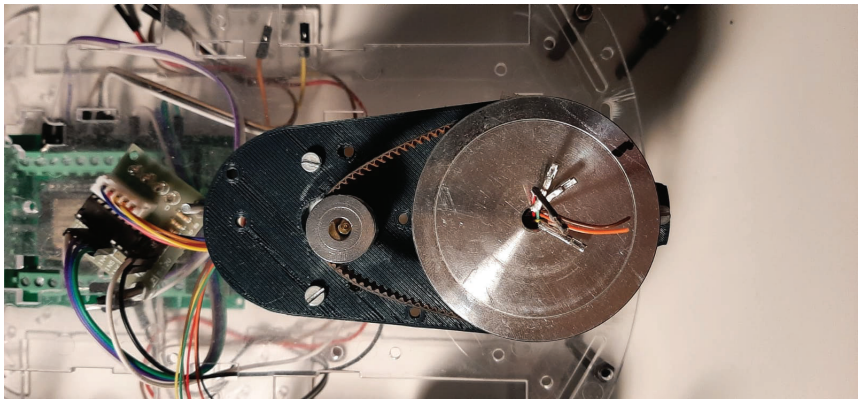
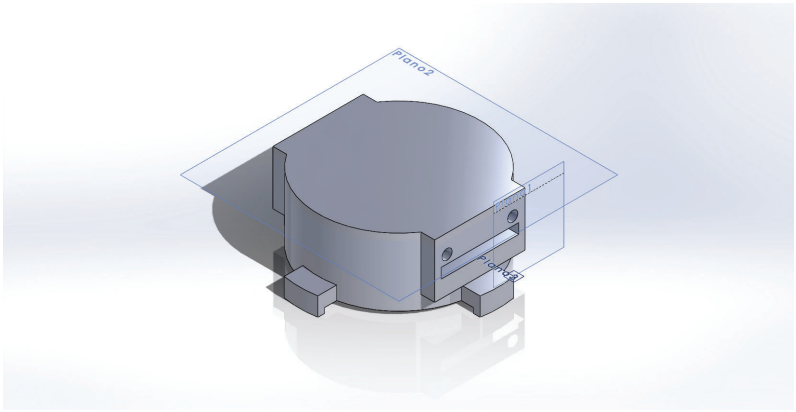
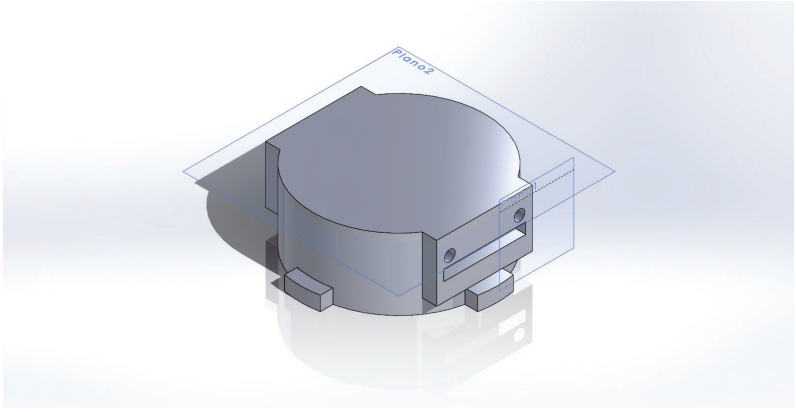


Figura 4.11: Transmisión mecánica LIDAR.

A la reductora se le acoplará un diseño 3D propio elaborado en *SolidWorks* (Figura 4.12) elaborado a partir de la idea original de la figura 4.10 a), al que se le ha reducido su tamaño y añadido dos caras para una posible ampliación a dos sensores. Se han diseñado dos alternativas: con pestañas de sujeción como se muestra en la figura 4.12 a) y sin pestañas como se observa en la figura 4.12 b), aunque la opción elegida es la opción con pestañas.



(a)



(b)

Figura 4.12: Diseños propios 3D.

Motor

De nuevo es imprescindible tener un control y precisión sobre la rotación del *LIDAR*, de modo que el motor seleccionado para ello es de nuevo el *motor paso a paso*. De esta forma, dada la relación de transmisión para la rotación del *LIDAR*, se puede determinar que 4 pasos del motor supone 1 paso para el *LIDAR*.

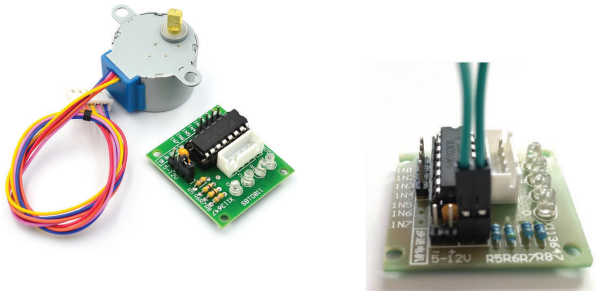
Debido a su reducido tamaño se empleará el motor paso a paso 28BYJ-48 (Figura 4.13 a), que permite mayor control y precisión frente a los motores *(CC)*. Se trata de un motor unipolar de 4 bobinas (con resistencia de 50Ω y consumo de 55mA) con un paso de 11'25º y que incluye un reductora 1:64[34]. No obstante, la relación real de reducción se calcula con el número de dientes de los engranajes que la forman (Fórmula 4.3).

$$\frac{32 \cdot 22 \cdot 26 \cdot 31}{9 \cdot 11 \cdot 9 \cdot 10} \approx 63{,}6839 \tag{4.3}$$

Por tanto, existe un error aproximadamente del 0.49 % (Fórmula 4.4) y un paso completo del motor supone un ángulo final del $\frac{11{,}25}{63{,}6739} = 0{,}1767º$, que para el *LIDAR* se convierte en $\frac{0{,}1767}{4} = 0{,}044º$.

$$Error = (1 - \frac{63{,}6839}{64}) \cdot 100\,\% = 0{,}49\,\% \tag{4.4}$$

Su alimentación y control se realiza con el dispositivo ULN2003(Figura 4.13 b), compuesto por 7 drivers, cada uno con 2 transistores en configuración Darlington[35] . Posee 4 entradas para el control de las bobinas y entradas para la alimentación del dispositivo y del motor. Si se alimenta la etapa con 5 voltios se debe puentear las conexiones con un jumper. Además, posee 4 leds que indican la bobina que se excita en cada momento.



(a) (b)

Figura 4.13: Motor 28BYJ-48 y jumpers ULN2003

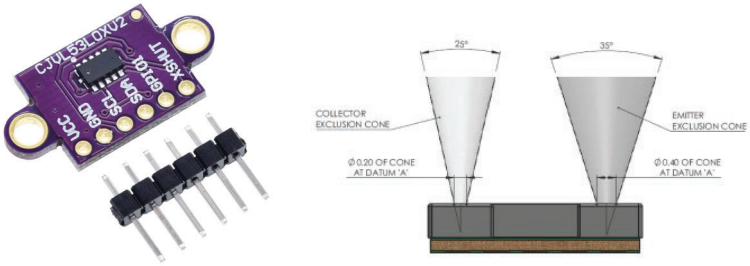
Anillo rodante

Además, es necesario un elemento que permita el giro de los cables sin que éstos se enreden y lleguen a provocar daños. En este proyecto se ha elegido un anillo rodante como se observa en la figura 4.14.

Figura 4.14: Anillo rodante.

Sensor de medición

El sensor de medición es un sensor láser infrarrojos VL53L0X (Figura 4.15). Su voltaje de funcionamiento entre 2.6V y 3.5V y su conexión al *ESP32* se realiza mediante bus I2C. Su campo de visión es de 25º con un emisor infrarrojos de 940 nm (el cono de emisión es de 35º mientras que el de colección es de 25º). El nivel de reflectancia para objetos blancos es del 88 % y para grises del 17 % y su características de precisión se observan en la figura 4.17. Tiene 4 modos de funcionamiento: modo por defecto, alta velocidad, alta precisión y larga distancia (Figura 4.16).



(a) (b)

Figura 4.15: VL53L0X

Range profile	Range timing budget	Typical performance	Typical application
Default mode	30 ms	1.2 m, accuracy as per *Table 12*	Standard
High accuracy	200 ms	1.2 m, accuracy < +/- 3 %	Precise measurement
Long range	33 ms	2 m, accuracy as per *Table 12*	Long ranging, only for dark conditions (no IR)
High speed	20 ms	1.2 m, accuracy +/- 5 %	High speed where accuracy is not priority

Figura 4.16: Perfiles de medición.

	Indoor (no infrared)			Outdoor		
Target reflectance level (full FOV)	Distance	33 ms	66 ms	Distance	33 ms	66 ms
White Target (88%)	At 120 cm	4 %	3 %	At 60 cm	7 %	6 %
Grey Target (17%)	At 70 cm	7 %	6%	at 40 cm	12 %	9 %

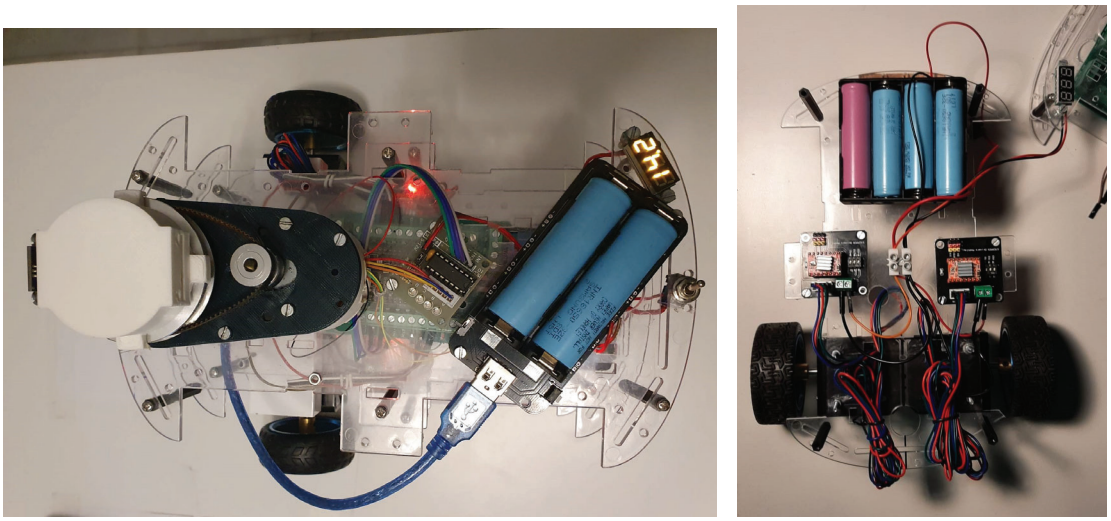
Figura 4.17: Tabla 12 Datasheet VL53L0X.

4.3. Alimentación

Como fuente de alimentación del robot se ha optado por usar pilas 18650. Por un lado, una de las razones de su elección es su capacidad de recarga. Por otro, se contaba con la adquisición previa de este tipo de alimentación, por lo que se decidió integrar al proyecto como ahorro de coste.

La alimentación se divide en dos partes (Figura 4.18):

- Alimentación del microcontrolador y del motor del *LIDAR*.
- Alimentación de los motores de las ruedas



(a) (b)

Figura 4.18: Alimentación del robot

Para los motores de las ruedas se van a emplear 4 baterías 18650 (Figura 4.19) pudiendo aportar hasta unos 14,8 voltios en total, voltaje suficiente para los motores *NEMA 17*. Además, se añadirá un interruptor para un menor consumo.

Para el microcontrolador *ESP32* se emplean dos baterías 18650 y un *Shield 18650 V8* (Figura 4.20), que permite obtener un voltaje de USB de salida regulado de 5V. Cuenta además con puertos de salida 5V a las que se han soldado los cables que alimentarán la etapa de potencia del motor paso a paso del *LIDAR*. Este Shield cuenta con un interruptor y un modo ahorro de energía que detiene el suministro si detecta que no se está demandando batería.

Samsung INR18650-32E 3100mAh - 6.4A

Sea el primero en dejar una reseña para este artículo

Información adicional

EAN / GTIN	7417940525839
Marca	Samsung
Modelo	INR18650-32E
Tamaño de la batería	18650
Química de la batería	Li-ion
Batería	Recargable
Voltaje	3.6V
Min. capacidad en mAh	3.100,00
Versión de batería	Superficie plana
Corriente de descarga	6,40
Protección del circuito	Desprotegido
Altura en mm	65,20
Diámetro	18,50

Figura 4.19: Baterías 18650.

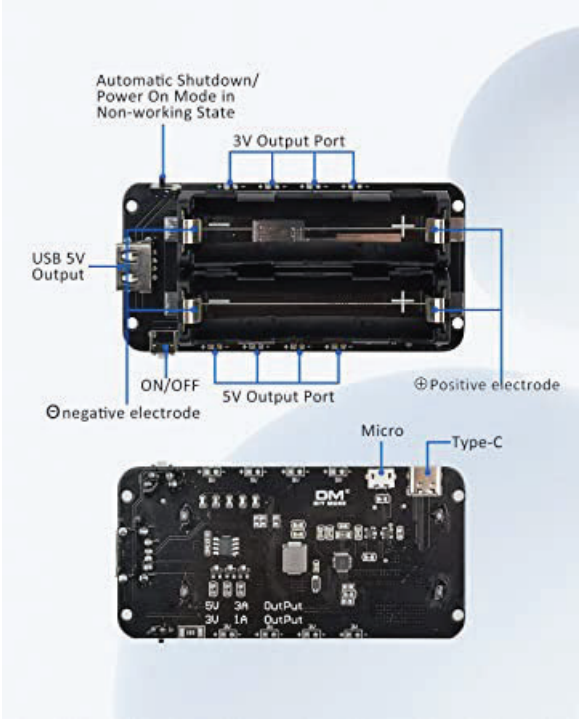


Figura 4.20: Shield 18650 V8.

4.4. Microcontrolador ESP32

FreeRTOS solo es compatible con una lista de dispositivos, aquellos con memoria flash superior al MB y RAM de más de 128kB. y entre ellos, se ha optado por utilizar una *ESP32* (Figura 4.21). Se trata de un *SoC* (System on Chip), un sistema completo integrado en un solo chip incluyendo memoria, microprocesadores y periféricos. Esta basado en la arquitectura *Xtensa LX6 Tensilica* y posee dos núcleos, que permitirán realizar las comunicación WiFi del robot así como controlar el comportamiento del mismo[36]. Sus principales especificaciones de interés para el proyecto son las siguientes:

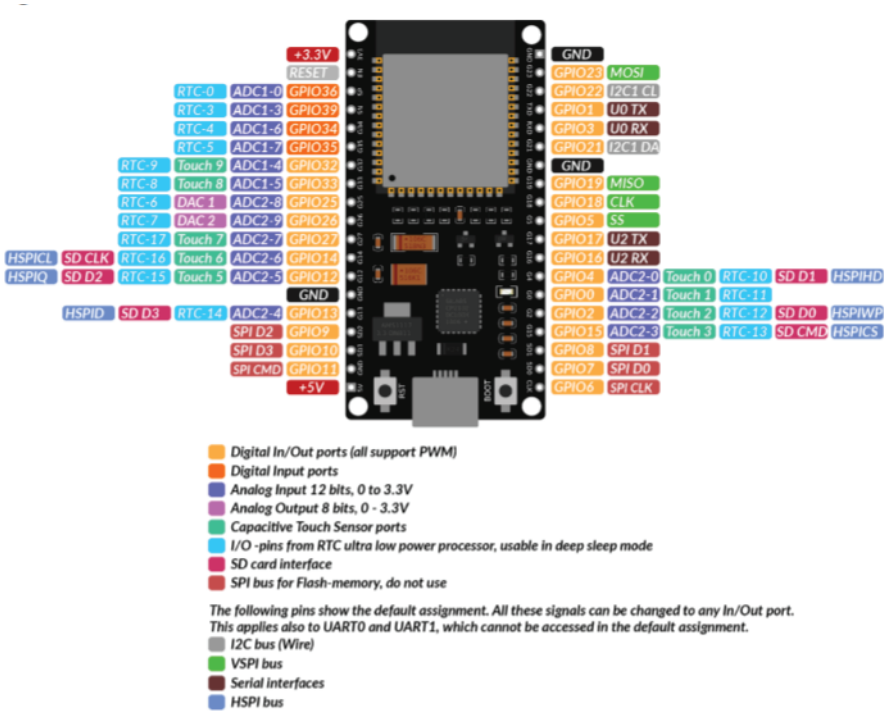


Figura 4.21: Pinout ESP32.

- SoC ESP32-WROOM 32.
- Procesador dual core Xtens LX6 de 32 bits, 800MHz/240MHz.
- 512 kB de RAM.
- Memoria flash externa de 4MB.
- 34 pines de entrada y salida, 18 canales ADC de 12 bits de resolución y 2 canales DAC de 8 bits, Figura 4.18.
- Protocolos SPI, I2C, I2s, CAN y UART.
- Conectividad WiFi a 2.5GHz y Bluettoth.
- Contiene un regulador de voltaje que permite su alimentación por USB.
- Los pines de entrada y salida trabajan a 3.3 V
- Alimentación a 5V.
- Consumo energético mínimo requerido de 500 mA.

Otras características de interés respecto a los pines son la siguientes:

- Los pines GPIO 34 a 39 son solo de entrada ya que no poseen resistencias pull-up y pull-down.
- Los pines GPIO 6 a GPIO 11 están conectados al SPI integrado y no se recomiendan para otros usos: GPIO 6 (SCK/CLK), GPIO 7 (SDO/SD0), GPIO 8 (SDI/SD1), GPIO 9 (SHD/SD2), GPIO 10 (SWP/SD3), GPIO 11 (CSC/CMD).
- Posee 16 canales independientes que pueden ser configurados para generar señales PWM de diferentes características. Todos los pines que pueden actuar como salida pueden ser usados como pines PWM

En este proyecto, se acoplará la *ESP32* a una placa de conexiones (Figura 4.22), que aportará más puertos de alimentación necesarios para los elementos de control y sensores.

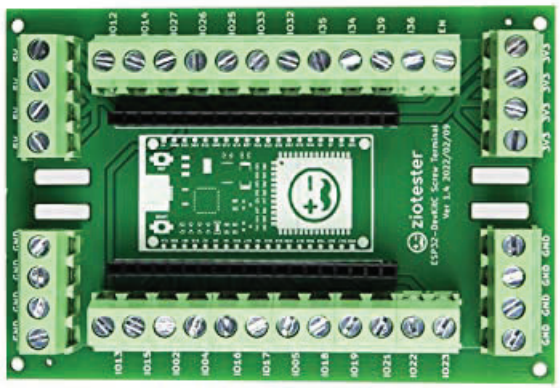


Figura 4.22: Placa conexiones ESP32.

4.5. Cartógrafo

En este proyecto el *broker* debe estar en constante disponibilidad, manteniendo el servidor siempre encendido. Por tanto, debido al bajo consumo y coste se ha elegido instalar el servidor en una *Raspberry Pi 3B+*. Además, la construcción del mapa se realizará sobre *Python*, que es uno de los lenguaje aceptados por la *Raspberry Pi*. Otra de las razones que ha motivado la elección de este dispositivo es su capacidad de almacenar el resultado de la exploración, manteniendo un histórico de las ejecuciones.

Por otro lado, para evitar las altas temperaturas que alcanza este dispositivo, se han instalado aletas térmicas para contrarrestar este efecto.

4.6. Conexionado

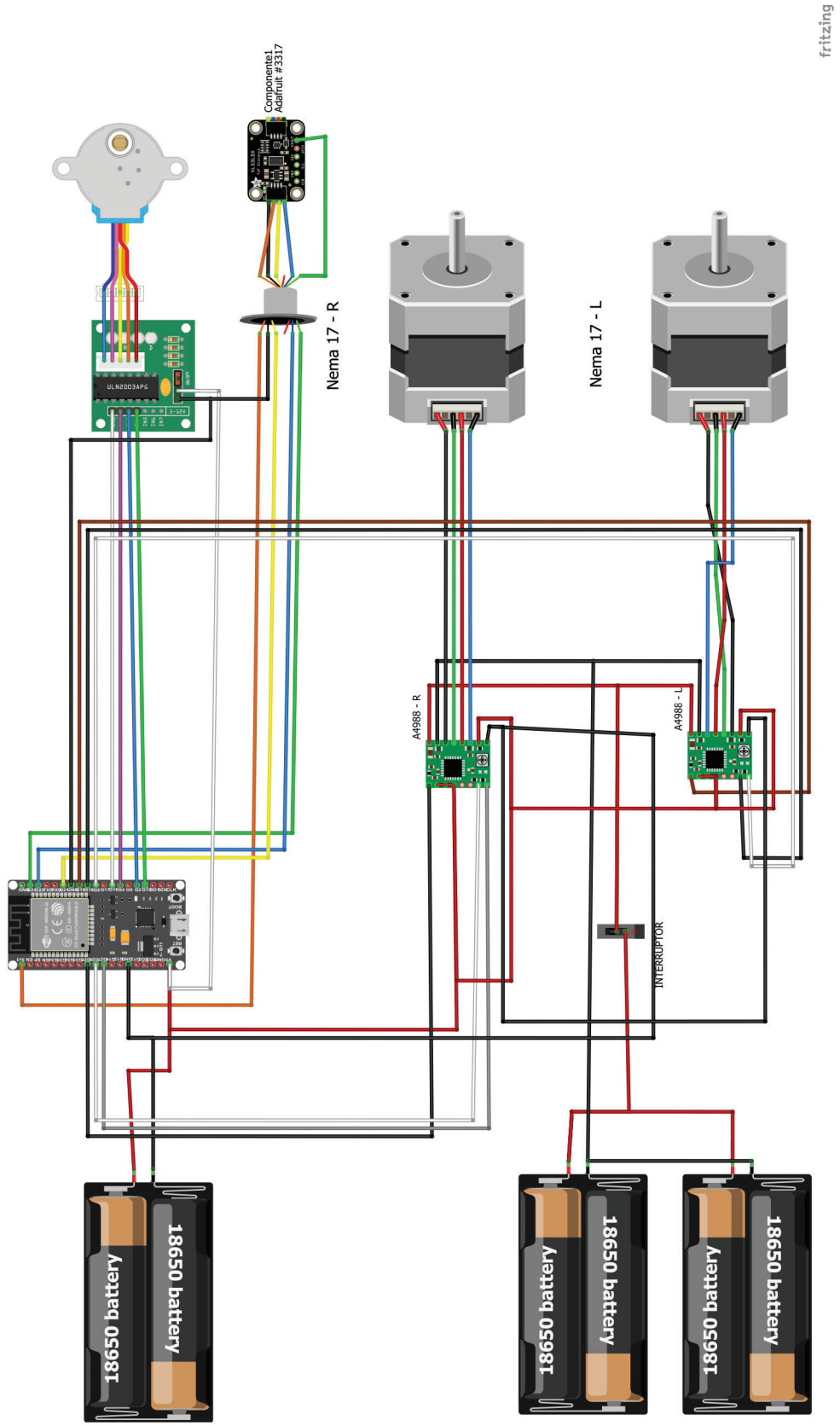


Figura 4.23: Esquema básico.

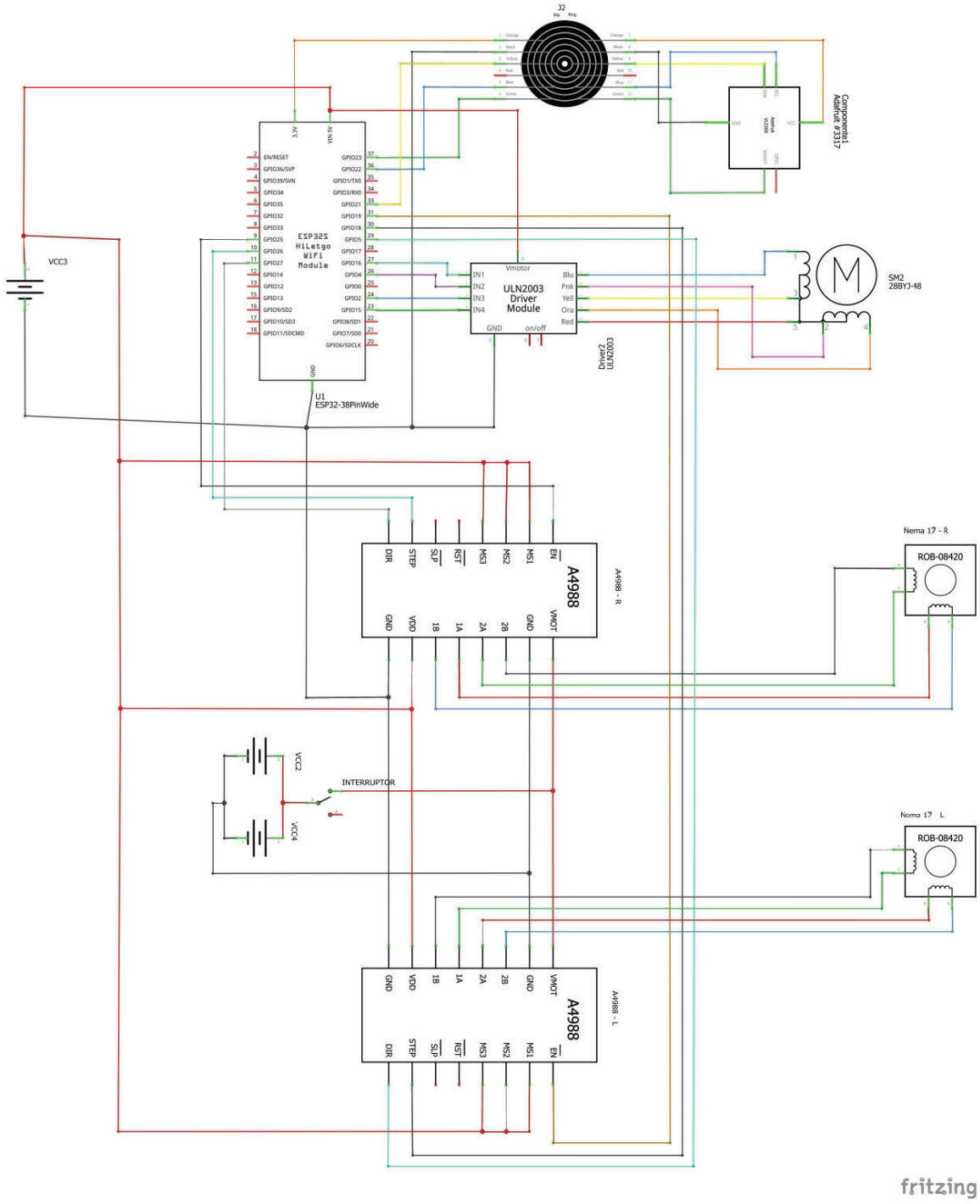


Figura 4.24: Esquema de conexiones.

El conexionado del robot se ha realizado usando *Fritzing*. Por un lado se muestra la vista de prototipado en la figura 4.23 y por otro lado, su esquema de conexiones en la figura 4.24.

CAPÍTULO 5

Calibración

Una vez que se dispone del robot y del sistema completo, a continuación pasamos a calibrar sus elementos de manera que éste sea fiable y se obtenga un control sobre él. Cuando se integra hardware en un sistema se debe comprobar su funcionamiento real respecto al teórico proporcionado por sus hojas de datos. No podemos suponer que el comportamiento del sistema coincida con el esperado, se deben hacer las pruebas y ajustes necesarios para garantizar que el comportamiento real coincida con el teórico. Nuestro sistema consta de tres elementos principales: la *Raspberry Pi*, el dispositivo y el robot. Los dos primeros son elementos comerciales, que suponemos se comportan según sus especificaciones y, además, no proporcionan mediciones críticas, por lo que su control se hará mediante software, sin plantear mayores escollos. Sin embargo, el robot ha sido construido desde cero, y sus sensores proporcionan las mediciones críticas para nuestro trabajo, por lo que habrá que comprobar que los datos que proporciona, tanto su posición como las medidas recabadas por sus sensores son suficientemente precisas.

En el capítulo 4 se han presentado los componentes del robot. Es necesario tener control sobre la posición del robot, actuando sobre su desplazamiento real, y sobre las mediciones del entorno. Así pues, deberemos estudiar y controlar el comportamiento del motor paso a paso y del *LIDAR*. A continuación se describen las pruebas y actuaciones realizadas para ello.

5.1. Desplazamiento

El desplazamiento depende de diferentes factores, que analizaremos a continuación. El código completo empleado en su calibración se encuentra en el apéndice F.

1. Calibración del chasis
2. Calibración del arranque
3. Calibración del movimiento en línea recta
4. Calibración de los giros

5.1.1. Calibración del chasis

En la primera versión del prototipo de robot construido el peso de los motores producía un cierto grado de combado del chasis y para solucionarlo se ha añadido un perfil de aluminio anclado al chasis y a los motores, reduciendo el efecto combado que se producía. Además, se añaden unos tirantes en la zona inferior de los motores para reducir aún más dicho efecto (Figura 5.1).

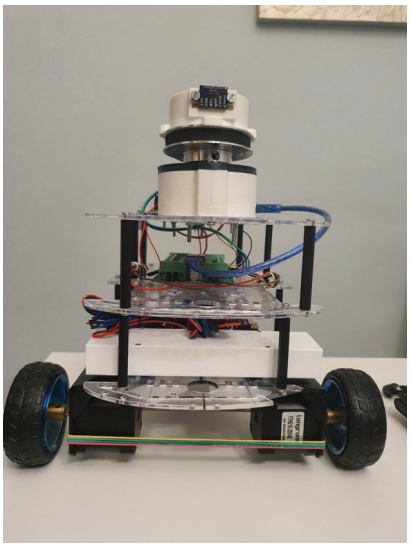


Figura 5.1: Calibración del chasis.

5.1.2. Calibración del arranque

Se observó que el robot realizaba un pequeño impulso en el arranque inicial del movimiento, tras la colocación manual en el entorno. Se ha comprobado si este tirón se producía en cada pausa del robot y se concluyó que solo se realiza si, a consecuencia de una manipulación del mismo, se rota el eje del motor lo que provoca el desencaje de un paso del mismo.

Para evitar este posible efecto, se aplicará un pequeño paso al inicio de la exploración. Ello no supone ningún efecto ya que, equivale a 3.2μm

5.1.3. Calibración del movimiento en línea recta

El método de calibración consiste en avanzar desde un punto fijo de partida una distancia conocida. Para conseguir esto, el robot calculará el número de pasos necesarios para recorrer la distancia teórica. Con ello, se marca la referencia desde punto de inicio a la distancia deseada, se habilita el movimiento del robot y se toma la medida de posición y orientación final de las ruedas del robot a dicha referencia.

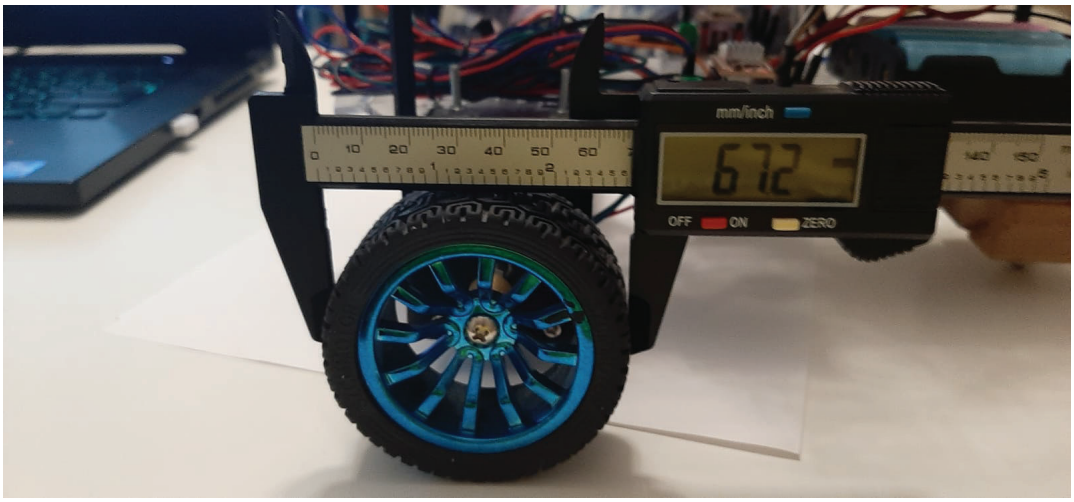
En primer lugar, se calcula la distancia de avance del robot en función de los pasos efectuados por el motor. La distancia se obtiene como la media entre la longitud andada por la rueda derecha e izquierda (Fórmula 5.1).

$$d = \frac{s_{drch} + s_{izq}}{2} \tag{5.1}$$

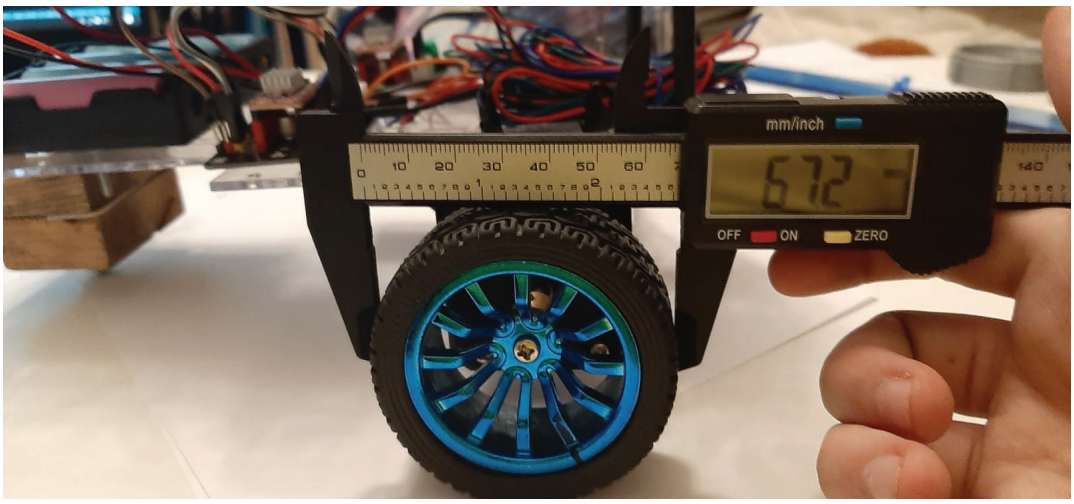
Para hallar la longitud de cada rueda se usa la fórmula 5.2 que describe el arco de una circunferencia. El valor de ϕ de cada rueda se calcula como la proporción de pasos para completar una vuelta completa por 2π radianes.

$$s = r \cdot \phi = r \cdot \frac{Pasos}{PasosporVuelta} \cdot 2\pi \tag{5.2}$$

Se mide el diámetro de cada rueda (Figura 5.2) y conociendo que cada vuelta son 3200 pasos se obtiene la fórmula 5.3.



(a)



(b)

Figura 5.2: Diámetro ruedas

$$d = \frac{s_{drch} + s_{izq}}{2} = \frac{\frac{6{,}72mm}{2} \cdot \frac{Pasos_{Izq} + Pasos_{Drch}}{3200pasos} \cdot 2\pi}{2} \tag{5.3}$$

En segundo lugar, se calcula el giro del robot en función de los pasos aplicados. En este caso se calcula como el cociente de la diferencia de longitud recorrida de la rueda derecha

menos la rueda izquierda, dividido por la distancia entre los puntos de apoyo de las ruedas (L) (Fórmula 5.4).

$$\theta = \frac{s_{drch} - s_{izq}}{L} \tag{5.4}$$

Se mide la distancia entre las ruedas (en nuestro caso 225.5 mm) y combinando la fórmula 5.2. se obtiene la fórmula 5.5.

$$\theta = \frac{s_{drch} - s_{izq}}{L} = \frac{\frac{6{,}72mm}{2} \cdot \frac{Pasos_{Drch} - Pasos_{Izq}}{3200pasos} \cdot 2\pi}{225{,}5mm} \tag{5.5}$$

Una vez conocido como afectan los pasos al desplazamiento y rotación del robot, se realizan distintos ensayos para comprobar el número de pasos reales necesarios frente al número teórico para alcanzar 1 metro de longitud en línea recta.

Ensayo 1

En este ensayo se comprueba el funcionamiento del robot sin aplicar ningún ajuste, es decir, se obtiene el número de pasos para avanzar 1 metro y se aplica la función de avance.

```
double metros = 1;
int pasos_recto = round((metros * 2 * STEPS_X_REV) / (2 * 2 * Pi * r))↩
    ;
avanzar(pasos_recto);

void avanzar(int pasos) {
  Serial.println("Avanzar");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  for (int i = 0; i < pasos; i++)
  {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
  }
  digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
  digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
}
```

Los resultados de las distintas pruebas son:

- **Primera prueba:** Se aprecia como la rueda derecha se adelanta a la rueda izquierda provocando un giro de un 1º respecto a la rueda izquierda. Con un error de -7mm a la marca de 1m respecto a la rueda izquierda.

- **Segunda prueba:** En este caso se adelanta más la rueda derecha con giro de 3º y un error de -9mm a la señal de 1m respecto a la rueda izquierda.
- **Tercera prueba:** Se obtienen los mismos resultados que la primera prueba.

Ensayo 2

En el ensayo anterior se obtuvo que la rueda derecha se adelantaba a la rueda izquierda, por lo que se probó de manera arbitraria el efecto de retroceder la rueda derecha cada 100 pasos.

Se realizaron dos pruebas con resultados muy similares:

- **Primera prueba:** Se aprecia como la rueda izquierda se adelanta a la rueda derecha provocando un giro de un -1º respecto a la rueda izquierda. Con un error de -5mm a la marca de 1m respecto a la rueda izquierda.
- **Segunda prueba:** Se obtiene el mismo efecto reduciendo el error a -4mm a la señal de 1m respecto a la rueda izquierda.

Ensayo 3

Se prueba a recorrer una mayor distancia teórica para obtener la distancia real de 1m.

```
double metros = 1.01;
int pasos_recto = round((metros * 2 * STEPS_X_REV) / (2 * 2 * Pi * r))↩
    ;
avanzar(pasos_recto);
```

Se realizaron dos pruebas con resultados:

- **Primera prueba:** Avance teórico de 1.01m, se observa un avance de la rueda derecha respecto a la rueda izquierda, provocando un ángulo de 2º y un error de +2mm de la rueda izquierda respecto a la marca de 1m.
- **Segunda prueba:** Avance teórivo de 1.005m, se observa un ángulo de 1.5 º, sin embargo, la rueda izquierda tiene un error de -3mm respecto a la marca de 1m mientras que la derecha de +5mm.

Ensayo 4

En este ensayo se trata de compensar el desajuste del avanza de una rueda respecto a la otra. Para ello, se obtiene el número de pasos necesarios para rotar 1 grado. El robot realiza los giros respecto al centro entre ambas ruedas, por lo que se aplica el mismo número de pasos para los motores en sentidos opuestos, que para girar 1º para cada rueda sigue la fórmula 5.6.

$$\frac{1^o \cdot \frac{\pi}{180} \cdot 225{,}5mm \cdot 3200}{\frac{6{,}72mm}{2} \cdot 2\pi \cdot 2} = Pasos_{Rueda} \approx 298pasos \tag{5.6}$$

Por otro lado, se calcula la cantidad de pasos para alcanzar 1m (Fórmula 5.7). De nuevo, siendo línea recta ambas ruedas dan el mismo número de pasos, por lo que implica que $Pasos_{Izq} = Pasos_{Drch}$.

$$\frac{1m \cdot 2 \cdot 3200}{\frac{6{,}72mm}{2} \cdot 2\pi \cdot 2} = Pasos_{Rueda} = 151576pasos \tag{5.7}$$

Se concluye que cada $\frac{151576}{298} = 508$ pasos se debe retroceder 1 paso la rueda derecha. En este ensayo se prueba a retroceder cada 500 pasos.

```
void avanzar(int pasos) {
  Serial.println("Avanzar");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  for (int i = 0; i < pasos; i++)
  {

    if (i % 500 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
    } else {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }

  digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
  digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
}
```

Los resultados obtenidos son los siguientes:

- **Primera prueba:** Se prueba a avanzar 1m y se obtiene de nuevo un ángulo de 0.5º respecto a la rueda izquierda con un error de -6mm a la marca de 1m.
- **Segunda prueba:** Se prueba a combinar el ensayo 3, avanzando teóricamente 1.005m. Se observa un ángulo de 0.5º respecto a la rueda izquierda y un error de -1mm respecto la rueda izquierda.
- **Tercera prueba:** Se repite la segunda prueba y se obtiene un error de -0.5º respecto a la rueda izquierda. No se detecta error en la distancia de la rueda izquierda a la marca. Es decir, solo se produce un error de -2mm de la rueda derecha a la marca.

Ensayo 5

En este ensayo se trata de eliminar el error de 0.5º que se produce por el avance de una rueda un cantidad de pasos mayor respecto a la otra. Por tanto, se utiliza la fórmula 5.7 para calcular cada cuantos pasos se debe proceder a un retroceso de la rueda derecha (Fórmula 5.7).

$$\frac{0{,}5^{o} \cdot \frac{\pi}{180} \cdot 225{,}5mm \cdot 3200}{\frac{6{,}72mm}{2} \cdot 2\pi \cdot 2} = Pasos_{Rueda} \approx 149pasos \tag{5.8}$$

Se obtiene que, cada $\frac{151576}{149} = 1017$ pasos se debe retroceder 1 paso la rueda derecha. En este ensayo se prueba a retroceder cada 1010 pasos.

```
void avanzar(int pasos) {
  Serial.println("Avanzar");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  for (int i = 0; i < pasos; i++)
  {

    if (i % 500 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
    } else if (i % 1010 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    } else {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }

  digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
  digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
}
```

Se prueba este efecto recorriendo una distancia teórica de 1m y se obtiene un error de ángulo respecto a la rueda izquierda menos a 0.5^{o} y un error en distancia a la marca de 1m de -3mm.

A este efecto se le aplican diferentes correcciones de distancia:

- **Primera prueba:** Se aplica 1.01, se obtiene un error de la rueda izquierda de -2mm y de la rueda derecha de +1mm.
- **Segunda prueba:** Se aplica 1.012, se obtiene un error de la rueda izquierda de -1mm. La rueda derecha no presenta error.

Ensayo 6
Se realiza el ensayo 4 aplicando un factor de 1.5 %, es decir recorriendo 1.015m. Los resultados obtenidos son:

- **Primera prueba:** Se obtiene un error de la rueda izquierda de +2mm aproximadamente y de la rueda derecha de +0.5mm aproximadamente.
- **Segunda prueba:** Mismos resultados que la primera prueba.
- **Tercera prueba:** Se obtiene un error de la rueda izquierda de +0.5mm aproximadamente y de la rueda derecha de +0.25mm aproximadamente.
- **Cuarta prueba:** Mismos resultados que la tercera prueba .

En estas cuatro pruebas se observa como puede impactar el efecto derivado del error en la posición inicial, ya que el error en la rueda derecha 2mm de la primera prueba desaparece al colocar el robot con mayor precisión. Como resultado, conseguimos reducir notablemente el error tanto, tanto en distancia como en orientación, en la prueba realizada. A continuación se realizan ensayos para diferentes distancias.

Ensayo 7
Para este ensayo se recorre una distancia real de 0.5m aplicando distintos factores de distancia. Los resultados son:

- **Primera prueba:** Se aplica un valor teórico de 0.5. Se obtiene un error de la rueda izquierda de -4mm y de la rueda derecha de -3.5mm aproximadamente
- **Segunda prueba:** Se corrige con un factor de 1 %, es decir, 0.505m. Se produce un error en la rueda izquierda de +1mm y de la rueda derecha de -1.5mm aproximadamente
- **Tercera prueba:** Ahora un factor de 1.5 %, es decir, 0.5075m el mismo factor que el Ensayo 6. Se produce un error en la rueda izquierda de +1mm y de la rueda derecha de +0.5mm aproximadamente

Ensayo 8

En este caso se recorre una distancia real equivalente a media vuelta del motor, es decir, de 0.1056m, aplicando el factor de 1.5 %. Los resultados son:

- **Primera prueba:** Se obtiene un error de la rueda izquierda menor a 0,5mm y de la rueda derecha menor a 0.25mm aproximadamente
- **Segunda prueba:** Se obtienen los mismos errores excepto en la rueda izquierda que en este caso es negativo, -0.5mm.
- **Tercera prueba:** Mismo resultado que la primera prueba

Conclusión

Como resultado de los ensayos de calibración, se puede concluir que se ha logrado controlar el avance en línea recta del robot consiguiendo un error inferior a 1mm y se ha demostrado que éste no se acumula para diferentes distancias.

Para aplicar el control, se ha comprobado que la distancia de avance real del robot es inferior a la distancia teórica que debería recorrer en función de los pasos, por lo conlleva a calcular la distancia de avance real aplicando un factor de corrección. Además, se ha detectado que la rueda derecha recorría una distancia mayor a la rueda izquierda y ésto provocaba un desviación en el avance del robot, por lo que se ha determinado la cantidad de paso que se deben realizar para aplicar un control sobre el motor derecho. La programación de esta calibración se puede consultar en el apéndice F.

5.1.4. Calibración de los giros

Esta prueba de calibración consiste en realizar un giro de más de 360º con pequeños fracciones de giros de 11.25º. Ello equivale al giro básico que realizará el robot en la exploración y se corresponde con 337 pasos. Por un lado, se probará a realizar un giro únicamente de más 360º y por otro, a realizar más de 5 giros seguidos completos.

```
    if (derecha) {
      for (int i = 1; i <= 33; i++){giroDerecha(337);}
    }else{
      for (int i = 1; i <= 33; i++){giroIzquierda(337);}
    }
    digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
    digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
```

Para poder cuantificar el resultado, se coloca un rotulador a una distancia conocida del centro de giro del robot, 30mm. Si el robot realiza un giro correctamente, se observará como el inicio y fin de la circunferencia dibujada coinciden y se cumple un radio de 30mm.

Giro a la izquierda

Ensayo 1

En primer lugar se prueba un giro sin aplicar ningún cambio tras de la calibración del chasis.

```
void giroIzquierda(int giroLsteps) {
  Serial.printf("Girar %d pasos hacia la izquierda", giroLsteps);
  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
  digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede

  for (int steps = 0; steps < giroLsteps; steps++){
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }
```

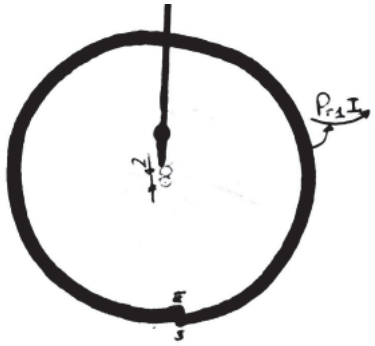


Figura 5.3: Ensayo 1 Giro izquierda.

En el resultado mostrado en la figura 5.3 se observa un cierto desplazamiento con un desfase de 2mm. Además, a medida que el robot se acercaba al final este tendía hacia el interior de la circunferencia.

Ensayo 2

El primer ensayo nos mostró que el robot tiene tendencia a desviarse hacia el interior, por lo que a continuación veremos como compensar este error. Para ello, estudiamos el efecto obtenido al retroceder la rueda derecha un paso de cada cada $\frac{337}{4} \approx 84$ pasos. La cantidad de pasos se ha decidido de forma arbitraria, para comprobar el efecto y actuar a continuación.

```
void giroIzquierda(int giroLsteps) {
  Serial.printf("Girar %d pasos hacia la izquierda", giroLsteps);
  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
  digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede

  for (int steps = 0; steps < giroLsteps; steps++)
  {
    if (steps % 84 == 0) {
```

```
            digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
            digitalWrite(STEPPER_R_STEPPIN, HIGH);
            digitalWrite(STEPPER_L_STEPPIN, HIGH);
            delayMicroseconds(STEP_SPEED);
            digitalWrite(STEPPER_R_STEPPIN, LOW);
            digitalWrite(STEPPER_L_STEPPIN, LOW);
            delayMicroseconds(STEP_SPEED);
        }else {
            digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
            digitalWrite(STEPPER_R_STEPPIN, HIGH);
            digitalWrite(STEPPER_L_STEPPIN, HIGH);
            delayMicroseconds(STEP_SPEED);
            digitalWrite(STEPPER_R_STEPPIN, LOW);
            digitalWrite(STEPPER_L_STEPPIN, LOW);
            delayMicroseconds(STEP_SPEED);
        }
    }
}
```

Figura 5.4: Ensayo 2 Giro izquierda.

El resultado de este ajuste se muestra en la figura 5.4, donde se observa como aumenta a la tendencia e incluso se produce un desfase mayor, de 3mm.

Ensayo 3

En esta prueba se trata de reducir el efecto de la tendencia al interior del ensayo 1. Si bien en el ensayo 2 se ha probado retrocediendo la rueda derecha, en este ensayo se comprobará el efecto de avanzar la rueda izquierda cada 84 pasos.

```
void giroIzquierda(int giroLsteps) {
  Serial.printf("Girar %d pasos hacia la izquierda", giroLsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
  digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede

  for (int steps = 0; steps < giroLsteps; steps++)
  {
    if (steps % 84 == 0) {
        digitalWrite(STEPPER_L_DIRPIN, HIGH); //Avanza
```

```
            digitalWrite(STEPPER_R_STEPPIN, HIGH);
            digitalWrite(STEPPER_L_STEPPIN, HIGH);
            delayMicroseconds(STEP_SPEED);
            digitalWrite(STEPPER_R_STEPPIN, LOW);
            digitalWrite(STEPPER_L_STEPPIN, LOW);
            delayMicroseconds(STEP_SPEED);
        }else {
            digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede
            digitalWrite(STEPPER_R_STEPPIN, HIGH);
            digitalWrite(STEPPER_L_STEPPIN, HIGH);
            delayMicroseconds(STEP_SPEED);
            digitalWrite(STEPPER_R_STEPPIN, LOW);
            digitalWrite(STEPPER_L_STEPPIN, LOW);
            delayMicroseconds(STEP_SPEED);
        }
    }
}
```

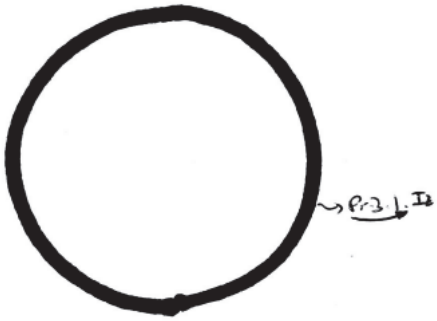
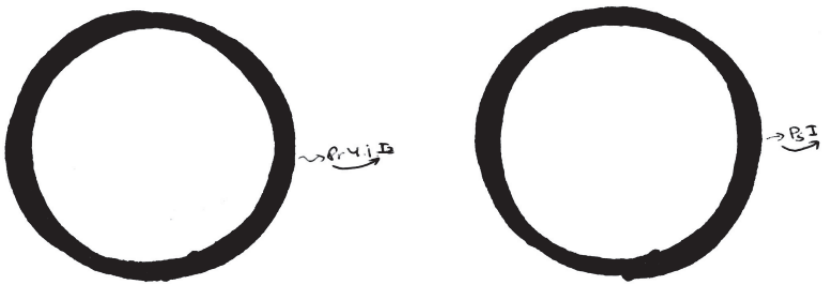


Figura 5.5: Ensayo 3 Giro izquierda.

En el resultado obtenido sE muestra en la figura 5.5 donde se puede apreciar que no hay desfase. Por tanto, se realizan nuevo ensayos para comprobar el resultado tras 5 giros.

Ensayo 4

A continuación se procede a realizar un ensayo en el que queremos comprobar si el ajuste se mantiene estable después de un número de giros. La figura 5.6 muestra el resultado tras 5 giros consecutivos. Como se puede observar, el robot sufre una desviación en el primer y tercer cuadrante, mientras que en el segundo y cuarto permanece estable.



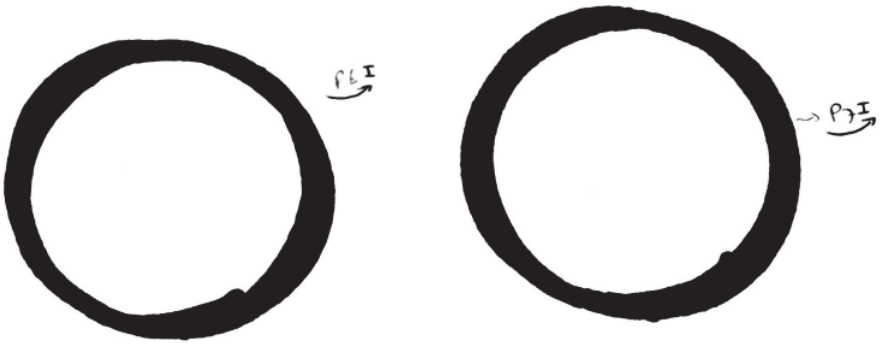
(a) (b)

Figura 5.6: Ensayo 4 Giro izquierda.

Ensayo 5

A continuación, se procede a estudiar el resultado de diferentes ajustes con los que compensar el desajuste detectado en el ensayo 4. Para ello, se avanzará la rueda izquierda en distintos intervalos de pasos.

- **Primera prueba:** Para $\frac{337}{2} \approx 168$ pasos, el resultado obtenido muestra como el desfase del primer cuadrante y tercer cuadrante es mayor (figura 5.7 a)). La conclusión obtenida es que no se debe aumentar el intervalo de pasos para avanzar la rueda izquierda.
- **Segunda prueba:** Para $\frac{337}{7} \approx 48$ pasos, el resultado se mantiene el desfase del primer cuadrante y tercer cuadrante (Figura 5.7 b). Se descarta reducir el intervalo de pasos para avanzar la rueda izquierda.

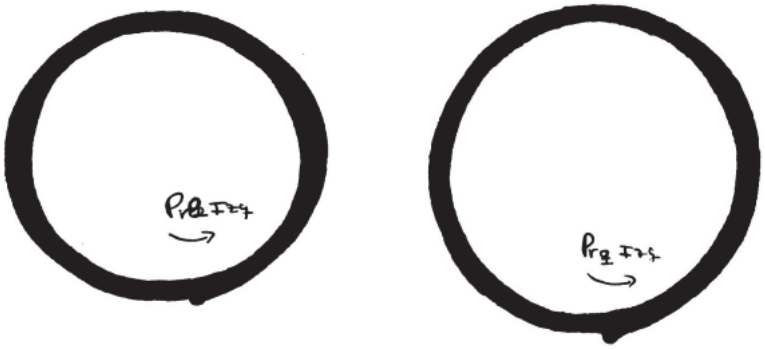


(a) (b)

Figura 5.7: Ensayo 7 Giro izquierda.

Ensayo 6

Dado que en los ensayos anteriores se mantiene el efecto del desfase, se prueba en este caso a dejar inmovilizada durante un paso la rueda derecha. Este efecto de nuevo se prueba cada cierta cantidad de pasos, comenzando la prueba, de forma arbitraria, en $\frac{337}{6} \approx 56$ pasos.



(a) (b)

Figura 5.8: Ensayo 6 Giro izquierda.

El resultado obtenido muestra como el efecto del desfase de los cuadrantes 1 y 3 desaparece notablemente, figura 5.8 a). No obstante, se aprecia un pequeño desfase en el 2 cuadrante. Por tanto, se realiza de nuevo la prueba cambiando el periodo de pausa de la rueda derecha en $\frac{337}{5} \approx 67$ pasos, figura 5.8 b).

Finalmente, se observa como tras 5 giros el robot permanece sobre la misma circunferencia. Tomando su radio tiene un valor de 30mm, igual a la distancia del rotulador al centro de giro.

Giro a la derecha

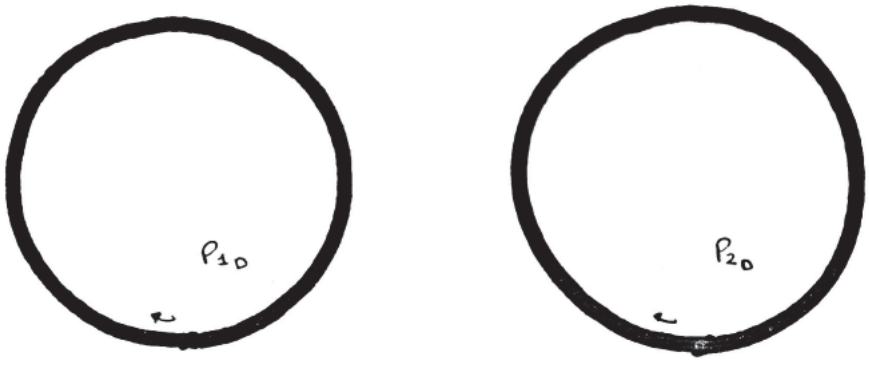
Ensayo 1

En primer lugar, se comprueba el funcionamiento tras la calibración del chasis para un solo giro.

```
void giroDerecha(int giroRsteps) {
  Serial.printf("Girar %d pasos hacia la derecha", giroRsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
  digitalWrite(STEPPER_L_DIRPIN, HIGH);//Avanza
  for (int steps = 0; steps < giroRsteps; steps++)
  {
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
  }
}
```

El resultado obtenido parece correcto, pero posee un radio de 34mm (Figura 5.9 a). Aunque no se observa desplazamiento, se repite de nuevo la prueba para comprobar el funcionamiento.



(a) (b)

Figura 5.9: Ensayo 1 Giro derecha.

En este caso, aunque de nuevo el resultado es correcto, a media que el robot se acerca al final del primer giro se advierte una tendencia hacia el exterior (Figura 5.9 b).

Ensayo 2

En este ensayo se comprueba la tendencia del robot para 5 giros consecutivos. De nuevo, tal y como ocurría para el giro a la izquierda, se observa un desplazamiento de 2mm aproximadamente en el primer y tercer cuadrante (Figura 5.10).

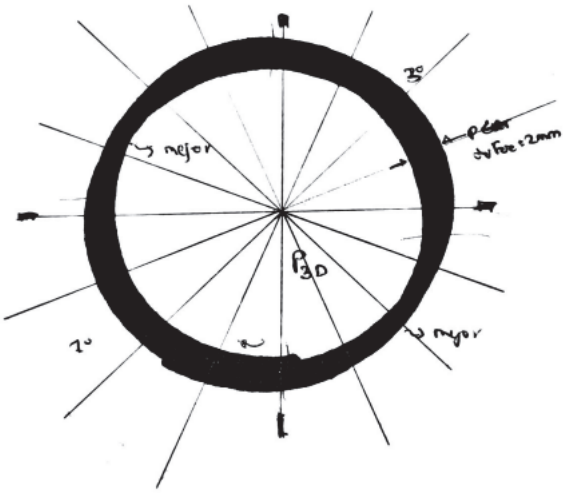


Figura 5.10: Ensayo 2 Giro derecha.

Ensayo 3

Debido a la similitud con el giro a la izquierda, se comprueba el efecto de avanzar la rueda derecha 1 paso cada cierta cantidad de pasos, en concreto cada $\frac{337}{6} \approx 56$ pasos (Figura 5.11).

```
void giroDerecha(int giroRsteps) {
  Serial.printf("Girar %d pasos hacia la derecha", giroRsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
  digitalWrite(STEPPER_L_DIRPIN, HIGH);//Avanza

  for (int steps = 0; steps < giroRsteps; steps++)
  {
    if (steps % 56 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);

    } else {
      digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
```

```
        digitalWrite(STEPPER_R_STEPPIN, HIGH);
        delayMicroseconds(STEP_SPEED);
        digitalWrite(STEPPER_L_STEPPIN, LOW);
        digitalWrite(STEPPER_R_STEPPIN, LOW);
        delayMicroseconds(STEP_SPEED);
      }
    }
  }
```

Figura 5.11: Ensayo 3 Giro derecha.

De nuevo, se mantiene el comportamiento en el primer y tercer cuadrante.

Ensayo 4

Puesto que el modo de proceder para el giro a la derecha es simétrico al giro a la izquierda, en este ensayo se comprueba el efecto de inmovilizar la rueda izquierda 1 paso cada número de pasos tras 5 giros consecutivos (Figura 5.12).

```
  void giroDerecha(int giroRsteps) {
    Serial.printf("Girar %d pasos hacia la derecha", giroRsteps);

    digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
    digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido
    digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
    digitalWrite(STEPPER_L_DIRPIN, HIGH);//Avanza

    for (int steps = 0; steps < giroRsteps; steps++)
    {
      if (steps % 56 == 0) {
        digitalWrite(STEPPER_R_STEPPIN, HIGH);
        delayMicroseconds(STEP_SPEED);
        digitalWrite(STEPPER_R_STEPPIN, LOW);
        delayMicroseconds(STEP_SPEED);
      } else {
        digitalWrite(STEPPER_L_STEPPIN, HIGH);
        digitalWrite(STEPPER_R_STEPPIN, HIGH);
        delayMicroseconds(STEP_SPEED);
        digitalWrite(STEPPER_L_STEPPIN, LOW);
        digitalWrite(STEPPER_R_STEPPIN, LOW);
```

```
            delayMicroseconds(STEP_SPEED);
        }
    }
}
```

Se realizan diferentes pruebas para distintos periodos de pausa de la rueda izquierda:

- **Primera prueba:** Se inmoviliza la rueda izquierda cada $\frac{337}{6} \approx 56$ pasos. Se observa una mejora notable con un error en el radio de giro de 2mm aproximadamente, 5.12 a) .

- **Segunda prueba:** En este caso se realiza cada $\frac{337}{8} \approx 42$ pasos. A pesar de realizar un giro con un radio de 30mm aproximadamente, sí se aprecia un desplazamiento en el primer y tercer cuadrante, figura 5.12 b).

- **Tercera prueba:** Para $\frac{337}{2} \approx 168$ pasos. En este caso el error es claramente notable, con una tendencia hacia el interior en el tercer cuadrante provocando un desplazamiento de hasta 4mm, figura 5.12 c).

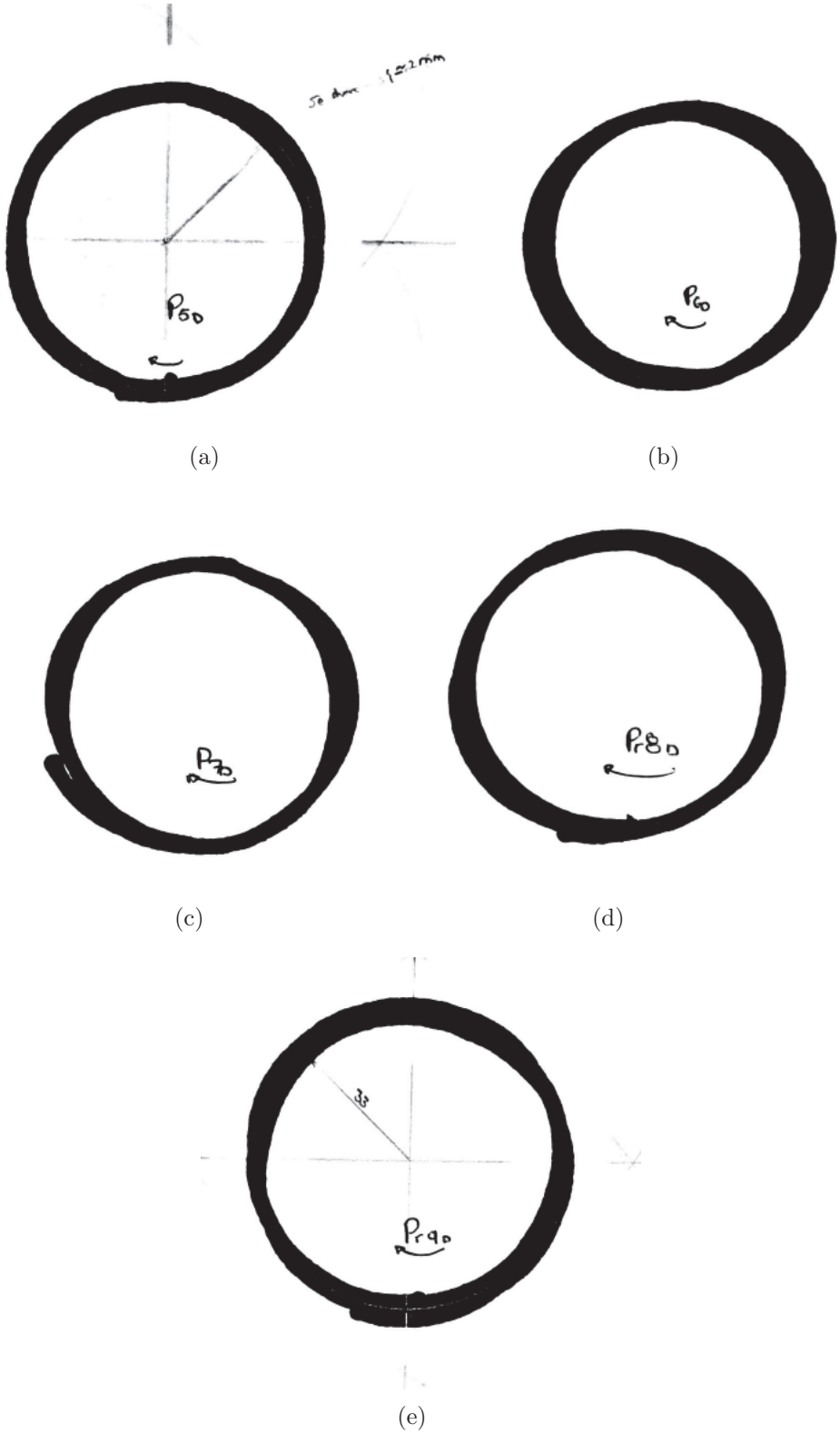
- **Cuarta prueba:** Cada $\frac{337}{7} \approx 48$ pasos. Se nota una mejora respecto a la segunda prueba pero se mantiene un leve desplazamiento en el primer y tercer cuadrante, figura 5.12 d).

- **Quinta prueba:** En esta prueba se realiza una pausa cada $\frac{337}{7} \approx 67$ pasos. Se aprecia un leve desplazamiento en el segundo y cuarto cuadrante con un error en el radio de 3mm aproximadamente, 5.12 e). Sin embargo, la primera prueba ha obtenido los mejores resultados.

Finalmente se concluye con la primera prueba de este ensayo, es decir, una pausa de la rueda izquierda cada $\frac{337}{6} \approx 56$ pasos.

Conclusión

Los resultados de los ensayos de calibración han demostrado que el comportamiento del robot para la rotación en ambos sentidos es simétrico, con errores en el primer y tercer cuadrante para cada uno. Por ello, también se comprueba como el control aplicado para ambos movimiento es también simétrico, inmovilizando la rueda interior al sentido del giro una vez realizados una cantidad de pasos. La programación de esta calibración se puede consultar en el apéndice F.



(a) (b)

(c) (d)

(e)

Figura 5.12: Ensayo 4 Giro derecha.

5.2. LIDAR

La calibración del *LIDAR* requiere actuar sobre sus dos componentes fundamentales: : el sensor láser y el motor paso a paso.

5.2.1. Calibración de la rotación

En este ensayo se han realizado dos pruebas independientes. En una se ha comprobado la precisión del motor individual durante 15 minutos y en otra la precisión de la transmisión completa durante otros 15 minutos.

En ambas pruebas se ha marcado una referencia en el punto de partida. En el caso de la prueba de la transmisión, la referencia será el punto de partida de la rueda reductora. Se han realizado vueltas completas consecutivas durante 15 minutos, con una pequeña espera entre ellas para visualizar la precisión resultante.

El número de pasos para dar una vuelta completo se ha obtenido de la hoja de datos del motor paso a paso. Para calibrar la transmisión, relación del número de dietes es de 4:1, es decir, es 4 veces el necesario para una vuelta del motor. Finalmente, ambas pruebas han sido exitosas sin necesidad de aplicar cambios.

5.2.2. Calibración de la distancia

Para esta prueba se han realizado diferentes ensayos a distintas distancias del objeto. En ellos se tomó la lectura máxima, mínima y la media y tras esto se calculó la desviación estándar, la varianza de todas las medidas tomadas y el error relativo. Los resultados se muestran en la figura 5.13. Las pruebas se realizaron para medidas inferiores a 1 metro, distancia a partir de la cual se tomaban medidas espúreas con mayor recurrencia. El código completo se encuentra en el apéndice G.

	950,00	900	800	700	600	500	400	300
REAL	950,00	900,00	800,00	700,00	600,00	500,00	400,00	300,00
MÁXIMO	1007,00	956,00	851,00	748,00	642,00	540,00	434,00	333,00
MEDIA	998,07	946,21	842,99	740,49	637,41	534,83	430,81	329,76
MÍNIMO	989,00	938,00	830,00	732,00	631,00	529,00	427,00	327,00
DESV	2,89233	2,84116	2,40081	2,17019	1,78621	1,35080	1,20495	0,97418
VARIANZA	8,36556	8,07219	5,76390	4,70974	3,19055	1,82466	1,45190	0,94902
ERROR ABSOLUTO	48,07	46,21	42,99	40,49	37,41	34,83	30,81	29,76
ERROR RELATIVO	5,0600%	5,1344%	5,3738%	5,7843%	6,2350%	6,9660%	7,7025%	9,9200%
ERROR MÁXIMO	56	55	50	48	42	38	34	32

Figura 5.13: Datos medidas láser en mm.

El análisis de los resultados obtenidos permite observar que en las medidas cortas, el láser mejora su precisión, ya que la desviación estándar inferior a 1. Sin embargo, el error relativo es mayor que para distancias largas, cuya desviación estándar es menor a 3. Para las medidas a distancias medias la desviación baja a 2 con un error relativo parecido al de las distancias largas. Si se mide la repetición de los errores en las medidas para cada distancia, se observa que las medidas siguen una distribución gaussiana[37] (Figura 5.14).

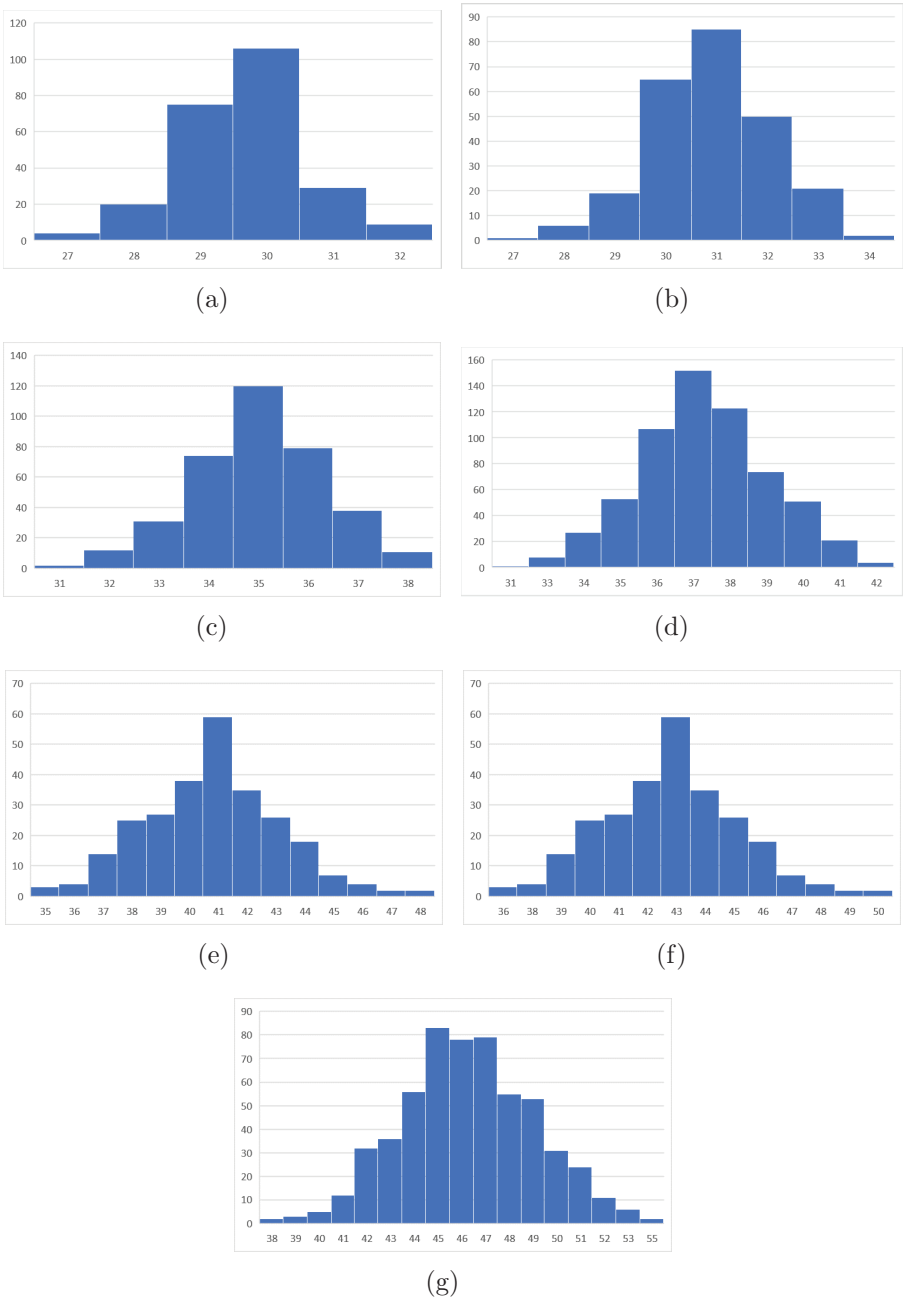


Figura 5.14: Gaussiana del error en las medidas láser de 300mm a 900mm.

Una vez analizado el error existente, estudiamos cómo ajustarlo. En primer lugar, la figura 5.15 muestra el resultado de aplicar un factor de corrección con el error relativo promedio (6.5 %). es decir, $medida_{Corregida} = medida \cdot (1 - 6,52\,\%)$.

REAL	950,00	900,00	800,00	700,00	600,00	500,00	400,00	300,00
MÁXIMO	941,32	893,65	795,50	699,22	600,13	504,78	405,69	311,28
MEDIA	932,98	884,50	788,01	692,20	595,84	499,95	402,71	308,25
MÍNIMO	924,50	876,82	775,87	684,26	589,85	494,50	399,15	305,67
ERROR ABSOLUTO	**17,02**	**15,50**	**11,99**	**7,80**	**4,16**	**0,05**	**2,71**	**8,25**
ERROR RELATIVO	1,792%	1,722%	1,499%	1,115%	0,694%	0,010%	0,678%	2,751%

Figura 5.15: Calibración láser por error relativo promedio.

A continuación, la figura 5.16 muestra el efecto de restar el error absoluto promedio (38 mm.) a las medidas obtenidas.

REAL	950,00	900,00	800,00	700,00	600,00	500,00	400,00	300,00
MÁXIMO	969,00	918,00	813,00	710,00	604,00	502,00	396,00	295,00
MEDIA	960,07	908,21	804,99	702,49	599,41	496,83	392,81	291,76
MÍNIMO	951,00	900,00	792,00	694,00	593,00	491,00	389,00	289,00
ERROR ABSOLUTO	**10,07**	**8,21**	**4,99**	**2,49**	**0,59**	**3,17**	**7,19**	**8,24**
ERROR RELATIVO	1,060%	0,912%	0,624%	0,356%	0,098%	0,634%	1,798%	2,747%

Figura 5.16: Calibración láser por error absoluto promedio.

Finalmente, la figura 5.17 muestra el efecto de una ajuste en el que se resta un error constante de 30mm, aplicando posteriormente un factor de corrección de 0.985.

REAL	950,00	900,00	800,00	700,00	600,00	500,00	400,00	300,00
MÁXIMO	962,35	912,11	808,69	707,23	602,82	502,35	397,94	298,46
MEDIA	953,55	902,47	800,80	699,83	598,30	497,26	394,80	295,26
MÍNIMO	944,62	894,38	788,00	691,47	591,99	491,52	391,05	292,55
ERROR ABSOLUTO	**3,55**	**2,47**	**0,80**	**0,17**	**1,70**	**2,74**	**5,20**	**4,74**
ERROR RELATIVO	0,374%	0,274%	0,099%	0,024%	0,284%	0,548%	1,301%	1,579%

Figura 5.17: Calibración láser mixta.

La calibración de la figura 5.16 da lugar a buenos resultados reduciendo el error relativo para todas las distancias. La calibración de la figura 5.15 muestra resultados aceptables a distancias medias, sin embargo para distancias largas y cortas el error es mucho mayor. El método de la figura 5.17 obtiene los mejores resultados sin superar el centímetro de error y con un error relativo inferior al 1,5 %, por lo que será el método de ajuste seleccionado.

Conclusión
Inicialmente, las medias tomadas cumplen con las especificaciones del producto (véase la figura 4.17). No obstante, éstas no cumplen el nivel de exigencia de precisión para el proyecto, por lo que se decidió realizar un calibración. Tras varios ensayos, se determinó, que el láser aplica un *offset* a cada medida, por lo que se aplicó el error absoluto promedio para anular este efecto. Además, se realizó un ajuste proporcional, ya que se observaba que el error dependía de la distancia de observación. Por tanto, con estos ajustes se ha conseguido reducir 10 veces el error absoluto máximo original y tener un error relativo máximo del 1.5 %. La programación de esta calibración se puede consultar en el apéndice G

5.2.3. Calibración del campo de visión LIDAR

Este ensayo se realiza colocando el robot en un punto y comprobando a distintas distancias el ancho máximo de su campo de visión, es decir, el ancho a partir del cual el robot no detecta el obstáculo. Una vez marcados distintos puntos se puede trazar la recta que contiene a todos ellos. El campo de visión teórico del robot es de 25º, sin embargo, el resultado obtenido es de 15.25º.

En conclusión, este efecto implica que sea necesario contemplar la medida del *LIDAR* para el ángulo máximo y mínimo durante la construcción del mapa.

CAPÍTULO 6

Diseño Software

Nuestro sistema consta de tres componentes fundamentales conectados mediante WiFi:

- Robot
- Raspberry Pi
- Aplicación Android

Cada componente está especializado en una tarea y está programado con herramientas adecuadas para ella: el control del robot durante la exploración se implementa en *C++* sobre *FreeRTOS* y se ejecuta sobre una *ESP32*; la construcción del mapa está programada en *Python* y se ejecuta sobre una Raspberry Pi y la aplicación móvil se ha desarrollado para *Android* en *Java*. Todos estos componentes interactúan y se comunican entre si usando el protocolo de comunicaciones *MQTT*. A continuación se describirán los aspectos de desarrollo software de cada uno de estos componentes. El código concreto de cada uno de ello puede consultarse en los apéndices correspondientes.

6.1. Comunicación del sistema

Los diferentes elementos se comunican mediante el protocolo *MQTT (Message Queuing Telemtry Transport)*. Se trata de un protocolo de mensajería ligero M2M (Machine-to-machine), que se ejecuta sobre TCP/IP utilizando la topología *PUBLISH/SUBSCRIBE* y usando el puerto 1883. Además, MQTT usa la mínima cantidad de ancho de banda, ya que únicamente transmite información cuando los dispositivos envían datos, es decir, está controlado por eventos. En este tipo de comunicación se diferencian dos componentes:

- **Clientes:** los dispositivos que se comunican entre sí. El intercambio de mensajes no es directo, sino que deben realizarse con el *broker*. Un cliente puede actuar como emisor de mensajes (publicador), como receptor de mensajes (suscriptor), o realizar ambas funciones. En este proyecto se definen **tres clientes**: el **robot**, cuyo dispositivo es la *ESP32* el **cartógrafo o builder**, alojado en la *Raspberry Pi*; y la **aplicación *Android* de control** del dispositivo móvil.
- **Broker:** actúa como servidor y los clientes se conectan a él. Su función es tomar los mensajes procedentes de los clientes publicadores y filtrarlos para retransmitirlos a los clientes destinatarios correspondientes, los suscriptores. En este proyecto se ha

decidido usar como servidor el software ***Mosquitto Eclipse*, instalado en la *Raspberry Pi***. Dicha instalación se puede consultar en el apéndice B.

Para que el *broker* sea capaz de filtrar los mensajes y distribuir a los suscriptores correspondientes, éstos publican sobre un tema o *topic*, de manera que, el mensaje que reciba sobre dicho *topic* será retransmitido a los clientes suscritos al mismo. Un *topic* es una cadena de texto *UTF-8* con una estructura jerárquica, utilizando el carácter / como delimitador para declarar los diferentes niveles. De esta forma, el suscriptor puede especificar el nivel de profundidad del topic al que desea suscribirse. Además, en caso de recibir una publicación sobre un topic que no existía, este se crea automáticamente. En este proyecto se declaran los siguientes topics:

- *robot/medidaObs* - Contiene la información de la pose (posición X, Y y orientación respecto al eje z) en la que se encuentra el robot y de la lectura del *LIDAR* en distancia y ángulo respecto al robot.
- *builder/mapa* - Se trata de la representación del mapa generada por el cliente cartógrafo o builder.
- *control/genMapa* - Indica si se debe realizar la representación de obstáculos sobre el mapa actual.
- *control/accion* - Comando que debe realizar el robot durante la exploración.

6.2. Programación del robot

La programación del robot se realiza sobre una *ESP32* usando el sistema operativo de *FreeRTOS* en el entorno de *Arduino IDE* en *C++* y se encuentra en el apéndice C . *FreeRTOS* es un sistema operativo que permite la ejecución de distintas tareas de forma concurrente. El número de tareas ejecutadas a la vez está limitado únicamente por la cantidad de núcleos que posea el hardware sobre el que se ejecuta. La *ESP32* dispone de dos núcleos: núcleo 0, también llamado de protocolo que es el encargado de las comunicaciones y, el núcleo 1, de aplicación.

FreeRTOS dispone de en un planificador o *TaskScheduler*, que se encarga de decidir qué tarea debe ejecutarse en cada momento según la prioridad de cada una de ellas (número entero cuyo nivel más bajo es el 0 y representa a una tarea inactiva). Estas tareas pueden encontrarse en diferentes estados (Figura 6.1): **eliminado**, **suspendido** (que la tarea está "parada"), **listo** (que la tarea está disponible para ser ejecutada), **bloqueado** (que está esperando a que ocurra algún evento para pasar al estado de "listo"), y **en ejecución** (que está siendo ejecutada por el procesador).

Las tareas se ejecutan en bucles infinitos, por lo que es necesario aplicar mecanismos que interrumpan la tarea actual para lpermitir la ejecución de tareas de menor prioridad. Para conseguir esto, se bloquean o se suspenden estas tareas aplicando *llamadas* como `vTaskSDelay`, que bloquea una tarea una cantidad de tiempo y tras esto la desbloquea. Por este motivo, en la declaración de funciones manejadoras de las tareas se añadirá una llamada a `vTaskSDelay`.

La ejecución de varias tareas concurrentemente puede provocar que actúen sobre un valor compartido, creando conflictos. Para ello, *FreeRTOS* dispone de métodos que evitan esta casuística: los *semáforos* y los *mutex*[38]. Otro aspecto importante son las *colas*[39],

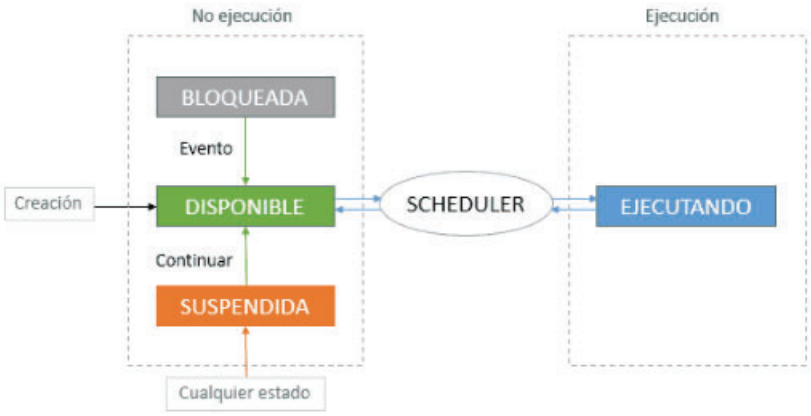


Figura 6.1: Estados FreeTOS.

que siguen el modelo *FIFO (First Input First Output)*, es decir el primer dato en recibirse será el primero en salir. En *FreeRTOS* a creación de una la se hace con `xQueueCreate` utilizando dos parámetros, la longitud máxima (número de elementos en cola) y el tamaño de los elementos cola. Este proyecto se utilizan todas estas primitivas, y su uso se describirá más adelante.

6.2.1. Modelo del proyecto

El proyecto sigue el modelo de Arduino con sus dos partes: `setup()` y `loop`. Esta última no se utiliza, ya que es el propio *FreeRTOS* se encarga de planificar la ejecución de tareas concurrentes. En el `setup()` se establece la conexión WiFi, llamando a la función `conecta_wifi()` y al servidor *MQTT* usando las funciones de la librería de *MQTT*, (`PubSubClient.h`), declarando también su función de `callback`. Este cliente solo se suscribe a los comandos de la aplicación (topic *control/accion*) y publica sobre un topic la pose y medidas obtenidas (topic *robot/medidasObs*). Tras esto, se declara un mutex y una cola, cuyo uso se explicará más adelante, se crean las tareas del robot y se eliminan las tareas actuales (setup y loop) para que el propio *FreeRTOS* comience con la planificación de tareas.

Las librerías que utilizará este proyecto son:

- Arduino.h: Contiene diversas funciones como operaciones matemáticas o funciones de manejo de los pines.
- Wire.h: Para habilitar la comunicación I2C del sensor *VL53L0X* del *LIDAR*
- Stepper.h: Para el control de motores paso a paso. Se usará para el motor de rotación del *LIDAR*.
- VL53L0X.h: Para el control del sensor *VL53L0X* del *LIDAR*.
- Wifi.h: Para la conexión WiFi del *ESP32*.
- PubSubCliente.h: para la comunicación *MQTT*.
- Arduino.JSON: para el manejo de formato *JSON* que se usará para estructurar los datos a enviar por *MQTT*.

6.2.2. Tareas del robot

Se definen 4 tareas: *movimientoTask*, para movimiento del robot; *lidarTask* para la rotación del *LIDAR* y medida del entorno; *sendMQTTTask*, para la publicación en MQTT; y *receiveMQTTTask* para la recepción de mensajes en MQTT. Estas dos últimas tareas se han divido en dos, debido a que, al ser un robot semiautónomo que recibe comandos de un usuario, éstos deben tener una prioridad diferente de la exploración del robot.

Tarea movimientoTask

Es la encargada del movimiento del robot y de calcular su pose actual. Antes de iniciar el bucle infito de esta tarea se deben configurar los pines de los motores paso a paso como salida. Estos pines son (para el motor derecho):

- `STEPPER_R_ENABLEPIN`, pin de enable para encender o apagar el motor
- `STEPPER_R_DIRPIN`, pin de dirección para indicar el sentido de giro
- `STEPPER_R_STEPPIN`, pin de step sobre el que se le aplica una señal PWM para realizar un paso del motor

Para el motor de la izquierda se indica como `STEPPER_L`. Además, se realiza un paso para calibrar el arranque y se inicializa a 0 la pose, que es una estructura formada por 3 elementos de tipo doble que representan la posición X, Y y θ actual del robot.

El movimiento del robot se realiza por cinco funciones básicas: avanzar(), retroceder(), giroDerecha(int giroRsteps), girIzquierda(int giroLsteps), parar(). Para cada movimiento, excepto parada, se encienden los motores poniendo a nivel bajo los pines de enable. Se indica el sentido de giro del motor aplicando un nivel alto o bajo según el sentido deseado, y se aplica una señal PWM. Esta señal no se generará llamando a la librería `Stepper.h`, sino que, se obtiene aplicando un valor alto y bajo al pin creando la señal deseada y ajustando el periodo y tamaño del pulso. Un periodo de la señal PWM equivale a un paso del motor y la duración de cada pulso controla la velocidad en la que se da el paso.

El avance y retroceso del robot realiza una cantidad de pasos constante y superior a la cantidad mínima para calibrar el movimiento. Los giros se realizan respecto al centro del eje de las ruedas del robot, de forma que, el robot gira sobre un punto. Para ello se gira una rueda en un sentido mientras la otra se gira en el sentido opuesto. Además, tienen como parámetro el número de pasos a efectuar, dando como resultado diferentes ángulos de giro. Por otro lado, en cada ejecución de movimiento se actualiza el valor de una variable global que contiene el número de pasos efectuados por cada rueda en el último movimiento.

La decisión de la función de movimiento a realizar depende de los parámetros de entrada de la función `void ControlMotor(''Comando'', ''Angulo LIDAR'', ''Distancia a objeto'')`. Ésta será la función que llame la tarea en cada iteración del bucle infinito, y tiene como entrada el comando recibido por *MQTT* de la aplicación de control del robot. En caso de modo "guiado" el robot recibirá los comandos de movimiento a realizar, es decir, recibirá la orden de avance, giro a la derecha o izquierda, retroceso o pausa y realizará la función correspondiente. En caso de modo exploración o autónomo, el robot debe realizar un algoritmo en función de la entrada del *LIDAR*. Si la distancia al objeto es inferior a 30cm éste continuará avanzando en línea recta; y en caso contrario, comprobará que el objeto se encuentre delante de su dirección de movimiento, es decir, que el ángulo

del *LIDAR* esté entre 60º y -60º. Si el ángulo se encuentra entre ese rango el robot realiza un movimiento de evasión: retrocede y gira aleatoriamente un ángulo entre 60 y 120º a la izquierda o derecha.

Tras esto, se calcula la pose del robot según los pasos aplicados, llamando a `estimar_pose()`. Para ello se calcula la distancia recorrida mediante la fórmula 6.1 y la variación de giro mediante la fórmula 6.2.

$$dist = \frac{s_{drch} + s_{izq}}{2} = \frac{\frac{6{,}72mm}{2} \cdot \frac{Pasos_{Izq} + Pasos_{Drch}}{3200pasos} \cdot 2\pi}{2} \tag{6.1}$$

$$\Delta\theta = \frac{s_{drch} - s_{izq}}{L} = \frac{\frac{6{,}72mm}{2} \cdot \frac{Pasos_{Drch} - Pasos_{Izq}}{3200pasos} \cdot 2\pi}{225{,}5mm} \tag{6.2}$$

El avance del robot se hará sobre el eje en X, por tanto se obtiene la pose del robot como indican las fórmulas 6.3, 6.4 y 6.5.

$$x_{act} = x_{prev} + dist \cdot \cos(\theta_{prev} + \frac{\Delta\theta}{2}) \tag{6.3}$$

$$y_{act} = y_{prev} + dist \cdot \sin(\theta_{prev} + \frac{\Delta\theta}{2}) \tag{6.4}$$

$$\theta_{act} = \theta_{prev} + \Delta\theta \tag{6.5}$$

Una vez calculado, se escala el ángulo de forma que se mantenga entre 0 y 2π y se guarda la pose actual como la pose previa del robot. El valor que devuelve esta función es el de la estructura de pose actual, compartido con la tarea de *lidarTask*, por lo que se utilizará un `mutex` para que no se modifique el valor mientras se usa el recurso en la tarea del *LIDAR*.

Tarea lidarTask

La misión de esta tarea es obtener la medida del entorno para construir el mensaje que se enviará por *MQTT*. Antes de comenzar el bucle infinito de la tarea se crea un objeto para el motor paso a paso, `LIDAR_STEPPER`, y otro para el sensor VL53L0X, `SENSOR_MEDICION`, usando las librerías `Stepper.h` y `VL53L0X.h`. Para ello se definen los parámetros del motor:

- Pasos por vuelta.
- Velocidad de rotación.
- Pines.
- Modo de medición del láser, que para este proyecto será el modo de alta precisión.

En cada iteración la tarea comprueba si el usuario ha activado el botón de "CANCELAR". En dicho caso, el motor paso a paso volverá a la posición inicial de 0º aplicando el número de pasos necesarios en función de su número de pasos actual, la función "retornoLIDAR()".

En caso contrario, se ejecuta el control del *LIDAR* (`controlLIDAR()`) que devuelve una estructura con los datos de el ángulo de giro del *LIDAR* y la medida de distancia

del entorno. En primer lugar, el motor realiza un número de pasos que equivale a 11,25º (realizando 32 medidas por vuelta) usando la función `step(Número de pasos)` de la librería y actualiza el valor del número actual de pasos. Se obtiene el valor del ángulo del *LIDAR* aplicando la fórmula 6.6. Tras esto se obtiene la lectura del láser, con la calibración aplicada, y si su valor es superior al máximo de distancia, el valor final es el máximo y se construye la estructura que devuelve la función.

$$\alpha = 360 \cdot \frac{PasosActuales}{PasosPorVuelta} \tag{6.6}$$

Conocida la pose del robot, que está bloqueada por el `mutex`, y la lectura del *LIDAR* se crea una estructura *JSON* que se serializa en un String. Este `String` se añade a una cola, `Medidas_msg_queue`, que se usará para publicar las medidas del entorno respecto a la posición del robot por *MQTT*.

Tarea sendMQTTTask

En esta tarea se comprueba la conexión con el cliente y se reconecta en caso necesario. Tras esto, se comprueba si hay datos en la cola `Medidas_msg_queue` y se publica el valor del elemento de la cola, las medidas del robot respecto a su pose, sobre el topic *robot/medidasObs*.

Tarea receiveMQTTTask

En esta tarea se comprueba también la conexión con el cliente y se reconecta si es necesario. Tras esto, se ejecuta el `loop` del cliente *MQTT*, función de la biblioteca de `PubSubClient.h`, que permite que el cliente procese, si se han recibido, mensajes entrantes del `callback`. En esta función se comprueba si el topic es al que se encuentra suscrito, *control/accion*, y toma el valor del comando recibido que almacena en la variable global "Accion".

6.3. Construcción del mapa

La generación del mapa se realiza sobre *Pyhton* en la *Raspberry Pi*, y su programación se encuentra en el apéndice D. Su objetivo es construir un mapa de rejilla a partir de los datos recibidos por *MQTT* con la pose estimada del robot y distancia y ángulo de la observación del *LIDAR* respecto a la posición del robot. El mapa será una representación probabilística en escala de grises, donde cada celda toma un valor entre 0 y 1, siendo 0 el valor de una celda ocupada y 1 el valor de una celda libre. El resultado se transmitirá de nuevo por *MQTT*. Otro requisito es poder generar el mapa indicando solo la posición de robot sin alterar el valor del mapa, es decir, sin representar nuevos obstáculos. Para llevar a cabo este proceso se han hecho uso de las siguientes librerías:

- paho.mqtt.client: Para la comunicación *MQTT*
- json: Para el manejo de JSON recibido por *MQTT*
- numpy: Para el cálculo numérico y uso de matrices
- math: Para operaciones matemáticas
- Io: Para manejo de entrada y salida y almacenar datos en memoria

- Matplotlib.Pyplot: Para la representación de imágenes
- Datetime: Para obtener la fecha del día actual y nombrar el mapa
- PIL.Image: Para manejo de imágenes.

6.3.1. Clase Mapa

Para la construcción del mapa se ha desarrollado una clase llamada "Mapa". Sus atributos y métodos son:

- Atributos
  - LogOdd_map: matriz con el valor de cada celda del mapa.
  - Centro: Centro del mapa.
  - TamCelda: Tamaño de la celda que corresponde a $30cm^2$.
  - FOV: Ángulo del campo de visión del láser en radianes.
  - MaxDist: Máxima distancia medida por el láser.
  - PosLidar: Posición del *LIDAR* en el mapa.
  - PosObjeto: Posiciones del objeto en el mapa, matriz de 2 filas que con el ángulo máximo y mínimo del *LIDAR* según el FOV.
  - logOdd_Libre: Logaritmo neperiano de la razón entre probabilidad de que la celda este libre y la probabilidad de que este ocupada.
  - loogOdd_Ocupado: Logaritmo neperiano de la razón entre probabilidad de que la celda este ocupada y la probabilidad de que este libre.
  - distCentroLidar: Distancia del centro del robot al centro del *LIDAR* en cm.
  - distLaser: Distancia desde el centro del *LIDAR* al láser en cm.
- Métodos
  - GenerarPosiciones: Se obtienen la posición del *LIDAR* y de los objetos en función de las medidas del robot.
  - ActualizarMapa: Se obtiene el mapa actualizando el valor de las celdas que hay entre el objeto y el *LIDAR*.
  - PublicarMapa: Generación de un byte array a partir de la imagen del mapa y publicación del mismo.

Al crear un objeto de esta clase se indica como parámetro de entrada el tamaño del mapa, que será siempre cuadrado, por lo que solo se especifica una dimensión y el tamaño de la celda.

6.3.2. Modelo de ejecución

Inicialmente se importan las librerías y se indican los parámetros del servidor *MQTT*, así como los valores de los topics. Este cliente publicará sobre el topic *builder/mapa* el mapa generado y se suscribe a: *control/genMapa*, que indica si se desea representar nuevos obstáculos, y *robot/medidasObs*, con el valor de la pose del robot y lectura del *LIDAR*. Tras esto, se declara un objeto mapa, se crea un cliente *MQTT*, se definen sus funciones de recepción de mensajes, "on_message" y de conexión al servidor, "on_connect" y se realiza la conexión.

Los mensajes procedentes del topic *builder/mapa* son JSON que se han serializado en un String, por tanto, si se recibe un dato de este topic se deserializa usando la librería "JSON". Tras esto, según si se desea o no representar los nuevos obstáculos (valor recibido por *control/genMapa* es .ªctivar.º "Desactivar") se obtiene las posiciones del *LIDAR* y del objeto, se actualiza el mapa y se publica a *builder/mapa*.

Generar posiciones

Este método obtiene la celda a la que corresponde la posición del LIDAR y del objeto en el mapa. Inicialmente, el robot se centra en el mapa y su posición avanzará una celda por cada distancia recorrida en una dirección igual al tamaño de la celda. El avance del robot se realiza en X con el eje Z saliente del plano. Para construir la matriz del mapa hay que tener en cuenta que una matriz tiene sus filas en orden ascendentes de arriba a abajo y las columnas en orden ascendente de izquierda a derecha. Por tanto, un aumento en (X,Y) del robot supone una disminución en la posición fila y columna respectivamente. Para obtener la posición del *LIDAR* se siguen las fórmulas 6.7 y 6.8

$$X_{Lidar} = Centro - floor(\frac{X_{robot}}{TamCelda}) \tag{6.7}$$

$$Y_{Lidar} = Centro - floor(\frac{Y_{robot}}{TamCelda}) \tag{6.8}$$

El valor de la posición del objeto depende de si se desea o no representar las nuevas observaciones. En caso afirmativo, se obtiene a partir de la pose del robot y de la distancia de observación (d_{objeto}) y del ángulo (α) del *LIDAR*. Se crean dos posiciones por cada medida, que corresponden a los haces del láser con el ángulo máximo y mínimo de del *LIDAR*. Es decir, el ángulo (α) del *LIDAR* $\pm\frac{FOV}{2}$. Así, se siguen las fórmulas 6.11 y 6.12 para su posición en X e Y. Para poder visualizar las fórmulas correctamente se definen unos coeficiente que se muestran en las fórmulas 6.9 y 6.10

Estas fórmulas se desarrollan a partir de la aplicación de la composición de transformadas homogéneas[40] teniendo en cuenta que, el centro del *LIDAR* no coincide con el centro del robot, sino que está desplazado 17mm (distCentroLidar) en el eje X de la coordenada del robot, y que el láser que compone el *LIDAR* está a su vez desplazado respecto al centro del mismo 24mm (distLaser) en el eje X del centro del *LIDAR* (véase Figura 6.2). En caso de que ambas posiciones den lugar a la misma celda esto hace que el valor de la misma se estime con mayor certeza.

$$C1 = d_{CentroLidar} + (d_{Laser+d_{Objeto}}) \cdot \cos(\alpha \pm \frac{FOV}{2}) \tag{6.9}$$

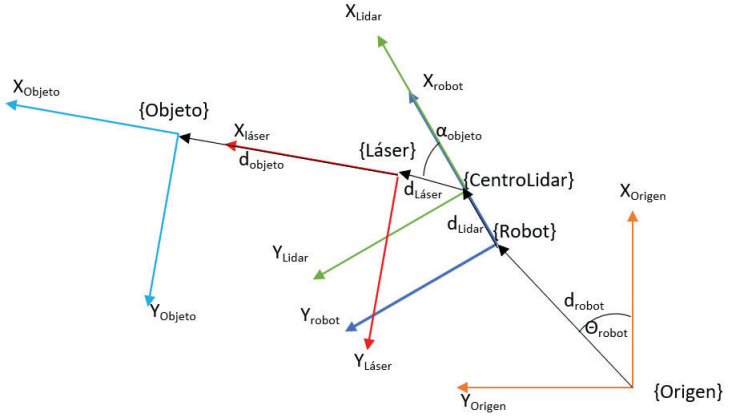


Figura 6.2: Composición de transformadas homogéneas.

$$C2 = (d_{Laser} + d_{Obj}) \cdot \sin(\alpha \pm \frac{FOV}{2}) \tag{6.10}$$

$$X_{obejto} = X_{Lidar} - floor(\frac{\cos(\theta) \cdot C1 - \sin(\theta) \cdot C2}{TamCelda}) \tag{6.11}$$

$$Y_{Objeto} = Y_{Lidar} - floor(\frac{\sin(\theta) \cdot C1 + \cos(\theta) \cdot C2}{TamCelda}) \tag{6.12}$$

Actualizar mapa

La actualización del mapa se realiza tomando el camino de celdas de la matriz desde la posición del robot al objeto. Este camino se obtiene aplicando el *algoritmo de Bresenham*[41] (Figura 6.3). Este algoritmo traza la recta entre dos puntos teniendo en cuenta la distancia entre ellos en ambos ejes. Tras esto, realiza un bucle con un avance unitario sobre el eje con mayor distancia entre puntos. En cada iteración se comprueba si se debe avanzar en el eje menor. En caso contrario solo se avanza en el eje mayor, es decir se realiza un llamado "avance recto" y en caso afirmativo se avanza en ambos ejes, conocido como "avance inclinado", obteniendo así el camino de celdas a recorrer.

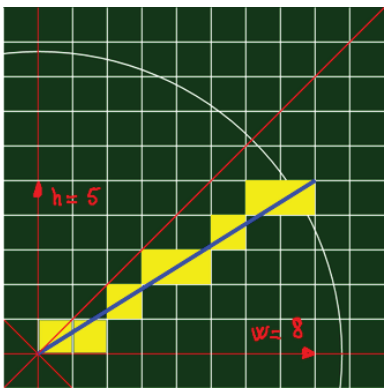


Figura 6.3: Algoritmo de Bresenham.

A continuación, se actualiza el valor de las celdas del mapa pertenecientes al camino, y se obtiene el mapa usando la operación matemática conocida como *Log Odds*[42]. Se trata del logaritmo de la proporción de probabilidades, es decir, la razón entre la probabilidad de que algo sea cierto y la probabilidad de que sea falso (Fórmula 6.13).

$$logOdds = \log(\frac{p}{1-p}) \tag{6.13}$$

Por lo que la probabilidad de que una celda este ocupada se obtiene según la fórmula 6.14, manteniendo siempre esta probabilidad entre 0 y 1 (Figura 6.4).

$$1 - p = 1 - \frac{1}{1 + e^{logOdds}} \tag{6.14}$$

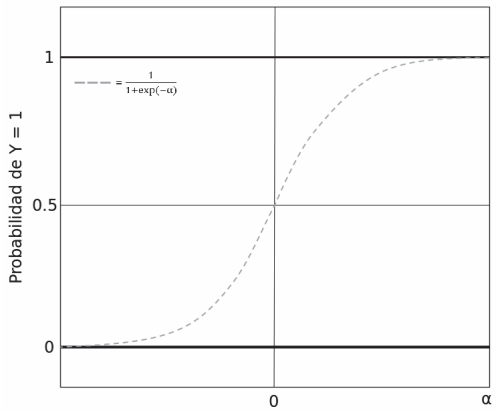


Figura 6.4: Probabilidad de ocupación.

.

Para cada celda se comprueba la distancia al objeto y se estima el valor en función de la observación anterior y el "Log Odd" de ocupación de observación actual. Por tanto, el proceso es el siguiente:

- Si la distancia es la máxima significa que no ha detectado ningún obstáculo. Se actualiza el valor de la celda como: el resultado anterior de la celda más el "Log Odds" de que se encuentre libre, (logOdd_Libre).
- Si la distancia es inferior:
  - Si la celda corresponde al camino hasta el obstáculo, se actualiza el valor como el resultado anterior de la celda más el "Log Odds" de que se encuentre libre, (logOdd_Libre)
  - Si corresponde al obstáculo, se actualiza el valor como el resultado anterior de la celda más el "Log Odds" de que se encuentre ocupada (logOdd_Ocupado).

Finalmente, el mapa se representa como la probabilidad de ocupación, aplicando la fórmula 6.14 a la matriz completa. Esta probabilidad se refleja en una escala de grises (Figura 6.5) donde el valor 1 indica ocupación y se muestra con una celda negra y el valor 0 indica una zona libre de obstáculos y se muestra en color blanco. Las zonas desconocidas, sin explorar, mantendrán su valor en 0.5, que se simboliza con el nivel de gris correspondiente. Si una celda que el robot ha leído previamente tiene el mismo valor

que la zona desconocida, esto quiere decir que su probabilidad de ocupación es de 50 % y se han obtenido el mismo número de lecturas de ocupación que de zona libre, por lo que se considera desconocido al anularse entre sí su información.

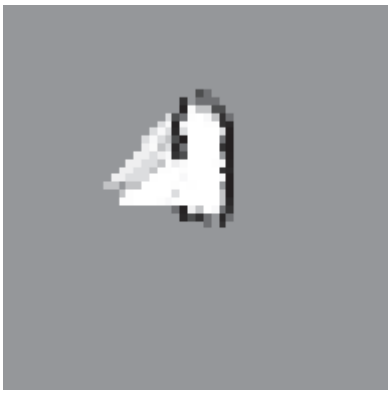


Figura 6.5: Ejemplo de generación de mapa.

Publicar mapa

En esta función se lee la imagen creada por el cartógrafo y se almacena en un espacio de memoria. Se toma el valor del *byte array* del espacio ocupado y se publica por *MQTT* al topic *builder/mapa*.

6.4. Desarrollo de la aplicación de control

Se ha desarrollado una aplicación que permite al usuario tomar decisiones sobre la exploración, es decir, le proporciona control sobre el robot. Para ello, se ha usado el entorno de *Android Studio* que proporciona las herramientas para el diseño gráfico y para la programación de los elementos de la aplicación, apéndice E. Las funcionalidades que tiene que permitir la aplicación son:

- Comunicación por *MQTT*.
- La pausa y reanudación de la exploración.
- La selección de modo: autónomo o "guiado" por el usuario.
- El modo guiado debe permitir al usuario dirigir al robot en las distintas direcciones.
- La visualización del mapa.
- La posibilidad de activar o desactivar la generación del mapa.
- La opción de cancelar el movimiento del robot y la generación del mapa.

6.4.1. Interfaz gráfica

En primer lugar, era necesario diseñar una interfaz clara e intuitiva para el usuario. Se probaron diferentes modelos en los que se podía cambiar entre diferentes pantallas y acceder a algún menú, sin embargo, la opción final fue utilizar una única pantalla en la que el usuario tenga toda la información, como se aprecia en la figura 6.6.

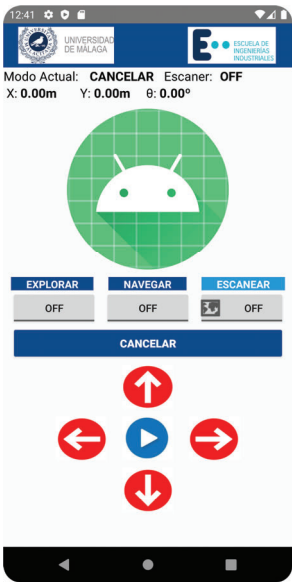


Figura 6.6: Diseño aplicación.

En la zona superior se encuentra la información de la exploración, así el usuario tendrá un resumen claro de la situación actual de la exploración. En esta zona se indica el tipo de exploración, para el modo manual o "guiado" se corresponde con el nombre de 'NAVEGAR' y para el modo automático con el nombre "EXPLORAR". Además, también se muestra si se encuentra activada la generación de mapa, nombrado como "ESCANER" y la pose del robot. Todos estos elementos corresponden al tipo de *widget* llamado *TextView* que muestra el texto que indique su propiedad "text".

A continuación, se encuentra el mapa que recibe la aplicación desde MQTT. Este tipo de *widget* se denomina *ImageView* y tomará el valor de su propiedad "foreground" que irá cambiando a medida que se actualiza el mapa.

En la zona media se encuentra una botonera de activación o *ToggleButton*, es decir, presentará dos estados: activado o desactivado. Por un lado, están los botones de modos: "EXPLORAR" y "NAVEGAR" y por otro lado, el botón de generación de mapa o "ESCANER". Además, se dispone de un botón de pulsación (*Button*) para abortar la exploración, "CANCELAR".

Por último, en la zona inferior está la botonera de control. Al iniciar la aplicación, o si se aborta, se muestra la botonera completa. Si se activa el modo "EXPLORAR" (modo automático) ésta cambia y solo se visualiza y se puede usar el botón en azul de "play", ya que este modo solo tiene opciones de exploración o pausa. Si el modo es "NAVEGAR" (modo manual) los únicos botones disponibles son los de color rojo, los cuales indican al robot la dirección de movimiento.

6.4.2. Programación de la aplicación

Una actividad en *Android Studio* representa una ventana sobre una pantalla en la que se representa la interfaz gráfica del usuario (Figura 6.7). En esta aplicación se dispone de una única actividad que cubrirá toda la pantalla de inicio.

Antes de crear la actividad, se disponen las distintas variables que se representan los elementos visibles de la aplicación. Entre ellas están las correspondientes a los botones, la imagen del mapa, el texto y los datos de *MQTT*.

Este proceso realiza dos publicaciones: una para comandos del robot, *control/accion*, y otra para el escaneo de obstáculos, *control/genMapa*, y dos suscripciones: para obtener el mapa, *builder/mapa*, y para mostrar la pose del robot, *medidas/Obs*. Así, se definen las funciones de conexión, callback, suscripción y publicación para *MQTT*. En el callback se obtienen los datos de los topics suscritos. Por un lado, del topic *medidas/Obs* se obtiene un *JSON* que se deserializa para obtener los valores X, Y y θ de la pose del robot y se asigna su valor a los *TextView* correspondientes a esta información. Por otro lado, del topic *builder/mapa* se obtiene un *byte array* cuyo contenido se decodifica en mapa de bits que actualizarán la propiedad "foreground", es decir el valor del la imagen del *ImageView* del mapa.

Al crear la actividad se ejecuta la conexión *MQTT* y se definen métodos de "escucha" para cada botón. En ellos se modificarán sus propiedades como: la visibilidad, para mostrar las partes de la botonera de desplazamiento en función del modo actual; la posibilidad para pulsar el botón, es decir, la botonera se mantendrá deshabilitada mientras no se active un modo; y el estado de cada modo, si activa un modo se debe "'apagar" otro de forma automática y "enceder" el correspondiente . Además, en función del modo activado, el valor del elemento de texto que lo indica se actualiza.

Estas acciones conllevan a la publicación del comando deseado por el usuario, por tanto, cuando se active o desactive un botón se publicará la actuación deseada . No obstante, los botones de giro funcionan según eventos de pulsación, es decir, mientras éstos se mantengan pulsados publicarán periódicamente el comando y cuando se liberen publicarán la pausa del robot.

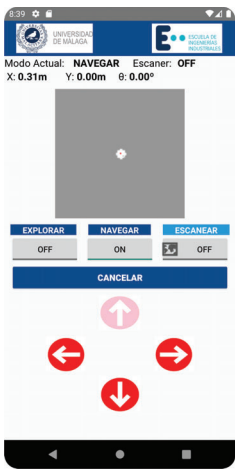


Figura 6.7: Navegación robot en la App.

CAPÍTULO 7

Pruebas y resultados

Para la comprobación del correcto funcionamiento del robot se han llevado a cabo tres pruebas. Dos de ellas se ha realizado en un mismo entorno, pero usando un modo de exploración diferente para comparar los resultados obtenidos respecto al entorno real. Para la tercera de ellas, se ha utilizado un entorno diferente combinando ambos modos de funcionamiento. En todas ellas se ha ejecutado el robot durante 30 minutos. Los resultados obtenidos servirán para evidenciar la correcta calibración del hardware junto a la programación del software de control del sistema.

Los testeos realizados demostrarán el comportamiento del robot en sus dos modalidades de uso. Sus resultados representarán en el mapa las zonas con mayor probabilidad de ocupación con un nivel de gris cercano al negro (0) y las zonas con menor probabilidad, con un nivel de gris cercano al blanco (255). Así, en función de la probabilidad, se observarán diferentes tonos de grises. Para las zonas desconocidas se mantiene el mismo nivel de gris (127). Además, se representa la posición del robot con un píxel de color rojo.

7.1. Primer entorno

Por un lado, se ha realizado una prueba en un entorno determinado bajo el control manual de la exploración (modo "navegación" en la aplicación) y, por otro lado, se realiza una segunda prueba con un control automático de la exploración (modo "exploración" en la aplicación). En ambas, se ha seleccionado una habitación cuyas dimensiones se reflejan en la figura 7.1 sobre un mapa de dimensiones de 225m^2 y un tamaño de celda de 30 cm^2.

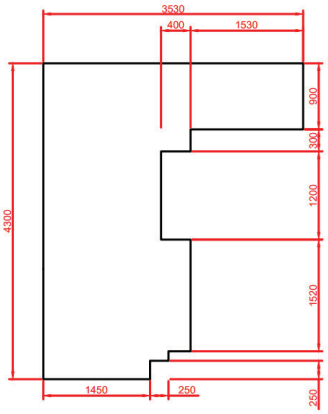


Figura 7.1: Entorno 1 de pruebas.

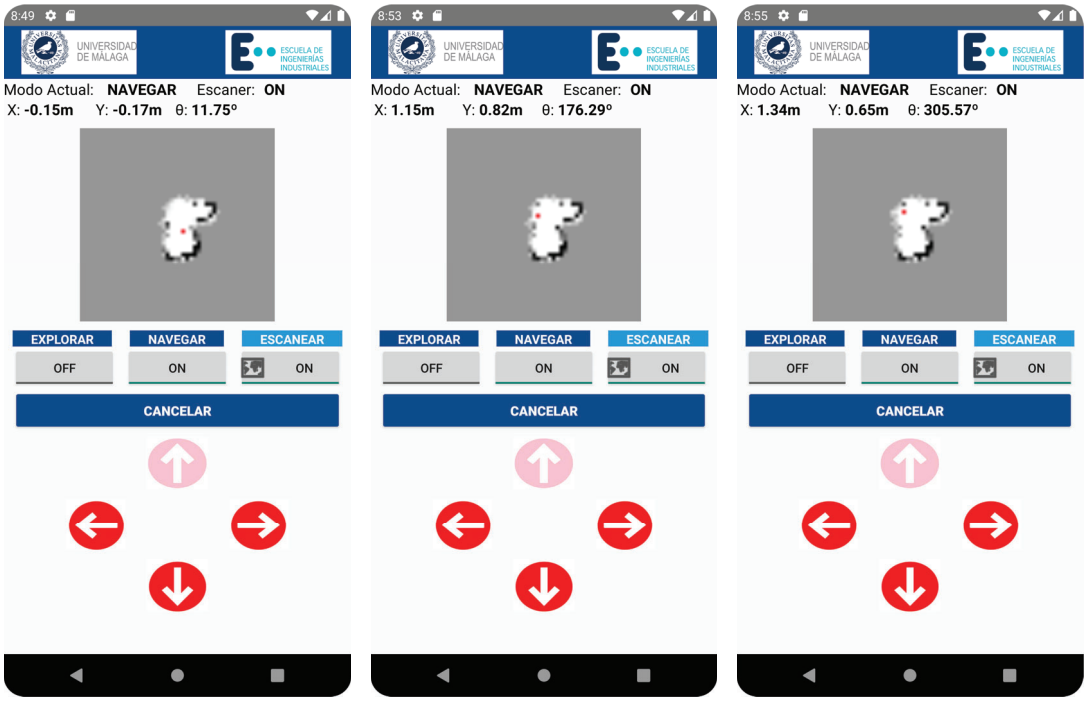
A continuación, se procederá a explicar con mayor detalle cada una de las modalidades realizadas.

7.1.1. Prueba manual

El modo seleccionado es "Navegación", es decir, bajo teleoperación del usuario. Para iniciar el movimiento, se decidió colocar el robot en el centro del mapa, de manera que se pudieran obtener medidas tanto para coordenadas positivas como negativas. Tras esto, se decidió arbitrariamente la dirección para iniciar el movimiento, dirigiendo al robot en dirección del eje negativo.

En la siguiente (Figura 7.2) se muestran los resultados obtenidos durante el progreso de la navegación una vez transcurridos 20 minutos desde su inicio hasta su finalización a los 30 minutos. Si se compara la representación del entorno de estas figuras, se observa:

- En la zona superior izquierda, se puede apreciar como, en la figura 7.2 a) se refleja la posibilidad de una zona libre de obstáculos. Transcurridos 4 minutos (Figura 7.2 b), el mapa refleja la posibilidad de un obstáculo que no se contemplaba en la figura anterior. Transcurridos 2 minutos (Figura 7.2 c), esta posibilidad aumenta simbolizándola con un tono más oscuro.
- En la zona izquierda, la figura 7.2 a) muestra una mayor probabilidad de ocupación y ésta disminuye hasta un nivel de gris que representa una probabilidad menor que la inicial, como se observa en la figura 7.2 c).
- Para el resto de las zonas se mantiene el resultado en las distintas progresiones temporales de la exploración



(a) (b) (c)

Figura 7.2: Prueba manual del primer entorno

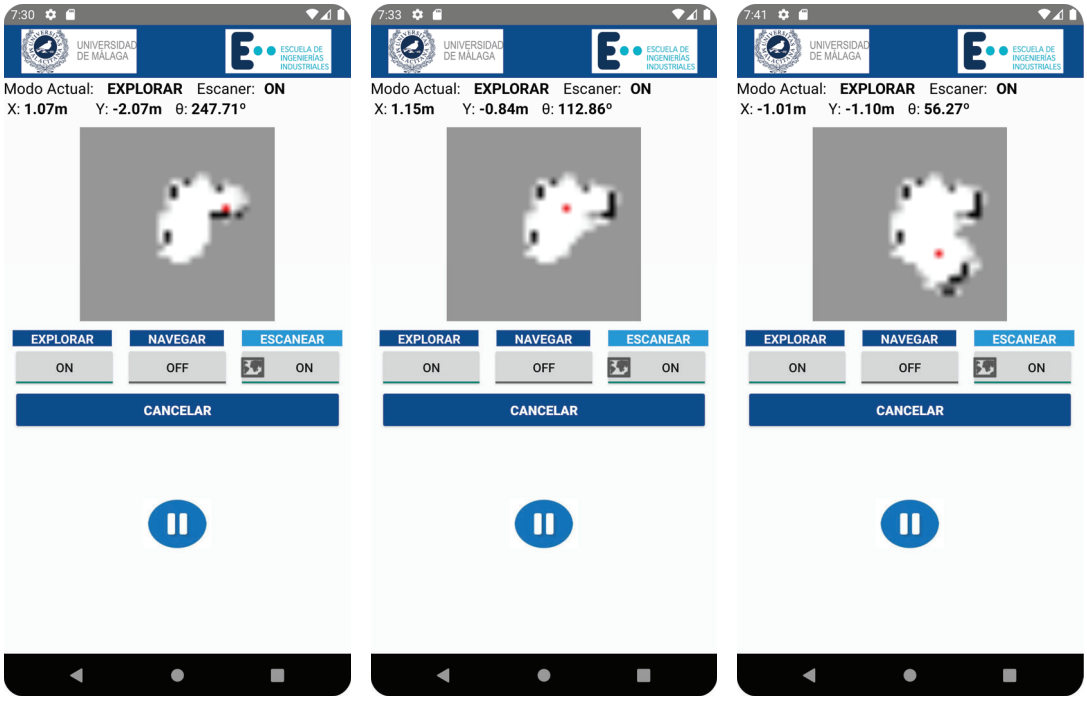
7.1.2. Prueba autónoma

En esta prueba el modo seleccionado es "Exploración", es decir, el robot tomará las decisión de movimiento, desplazándose por el entorno según su algoritmo. Para esta prueba, el mapa de ocupación representará una superficie con dimensiones de $81m^2$, de nuevo con un tamaño de celda de $30cm^2$ y, del mismo modo que se hizo en la prueba manual, el robot se ubicará inicialmente en el centro del mapa. Por defecto, el robot comenzará el movimiento en línea recta hasta detectar el primer obstáculo, que provocará un giro aleatorio tanto en dirección como en ángulo.

En las figuras 7.3 y 7.4 se muestran los resultados obtenidos. A continuación, se procede a comentar el resultado de estas figuras:

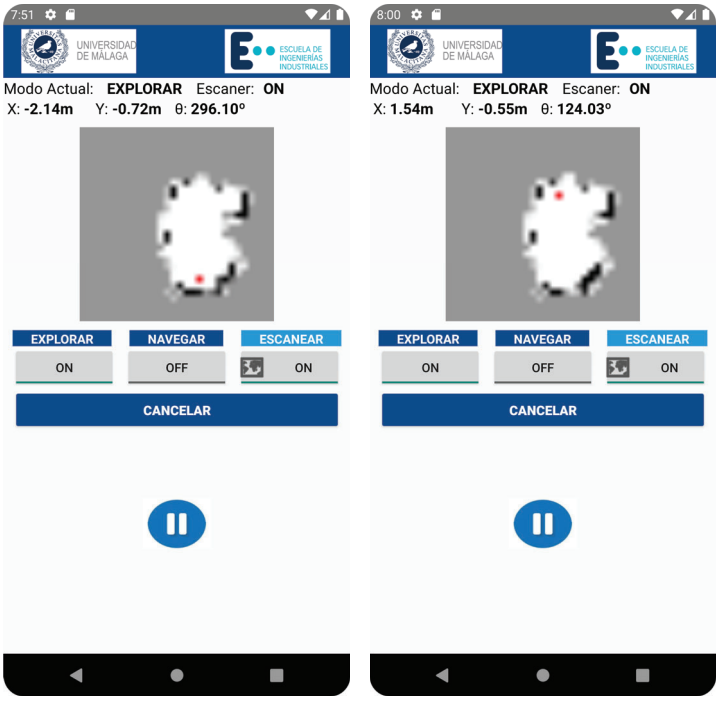
- Tras 10 minutos del inicio de la exploración, en la figura 7.2 a) se observa como el robot ha detectado un obstáculo durante su camino en línea recta, esto ha provocado que el robot modifique su trayectoria encontrándose en la zona superior derecha.
- Transcurridos 3 minutos de la observación anterior, en la figura 7.2 b) se obsrva que el robot ha detectado más obstáculos en la zona superior derecha, y éstos, han provocado que cambie de dirección situándose cerca del origen de la exploración.
- Pasados 8 minutos del caso anterior, en la figura 7.2 c) se observa que el robot se encuentra en la zona inferior, donde aparecen nuevos obstáculos. Además, se observan distintos tonos de grises en la zona derecha del mapa que indican la posibilidad de que exista un obstáculo con menor certeza.
- Tras 10 minutos más, en la figura 7.4 a) se observa que el robot expande el mapa por la zona inferior, mapeando más obstáculos.

- Finalmente, completados los 30 minutos, en la figura 7.4 b) se observa que el robot se encuentra en la zona superior del mapa, mientras que éste no parece sufrir cambios notables.



(a) (b) (c)

Figura 7.3: Prueba autónoma. Primer entorno tras 10 minutos



(a) (b)

Figura 7.4: Prueba autónoma. Primer entorno, últimos 20 minutos

7.2. Segundo entorno

En esta tercera prueba se ha seleccionado un entorno diferente al anterior, cuyas dimensiones se observan en la figura 7.5. El reconocimiento del entorno se realiza combinando el modo "Exploración", modo automático del robot, y el modo "Navegación", modo guiado. El mapa de ocupación mantendrá sus dimensiones en $81m^2$, con un tamaño de celda de $30cm^2$, y al igual que en el anterior entorno, el robot se colocará inicialmente en el centro del mapa.

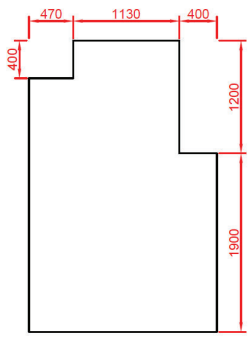


Figura 7.5: Entorno 2 de pruebas.

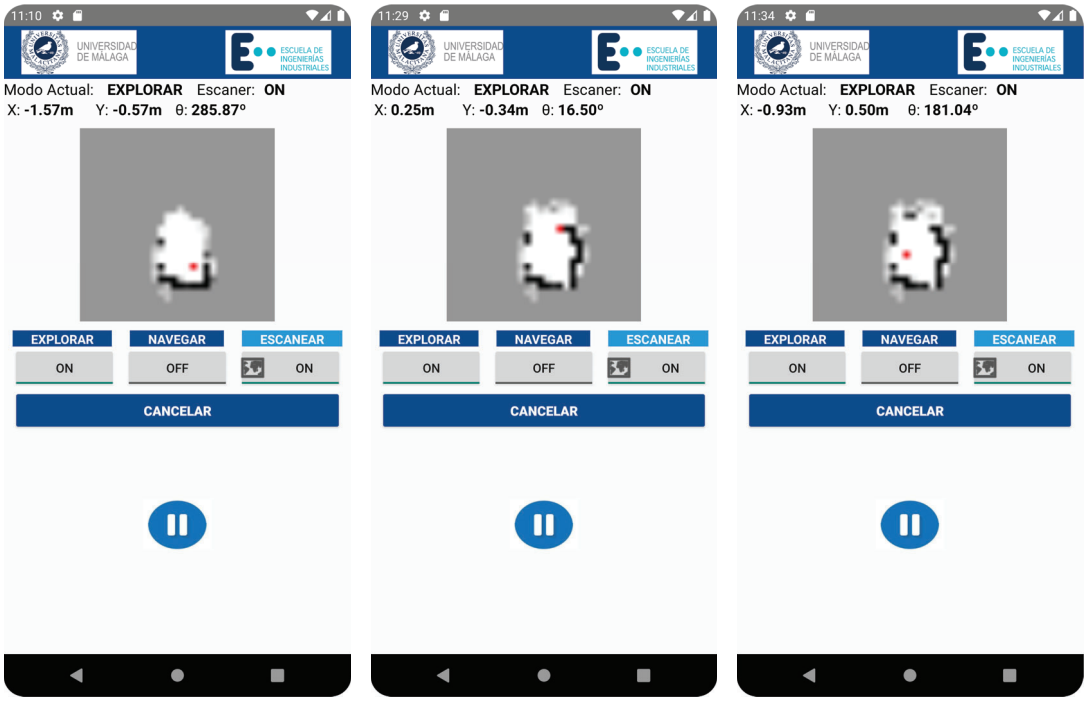
En un primer momento, el robot se dirigió a la zona inferior del mapa utilizando el modo "Navegación". Detectados los primeros obstáculos se cambió al modo "Exploración" alternando con el modo "Navegación" para colocar al robot en las zonas en las que se detectaba la posibilidad de un obstáculo. De igual forma se procede durante la evolución de la prueba.

En las figura 7.6 se pueden ver los resultados obtenidos en esta ocasión. Comparando la representación del entorno de cada progreso, se extrae que:

- Transcurridos 10 minutos del inicio, en la figura 7.6 a) se observa que el robot ha mapeado la zona inferior, por tanto, se procede a orientar al robot, usando el modo "Navegación" a una zona libre superior en busca de nuevos obstáculos y se vuelve al modo autónomo de "Exploración".

- Tras 20 minutos más, en figura 7.6 b) se puede ver que se han detectado nuevos obstáculos y se mantiene al robot por esta zona de manera que, tras varias lecturas del entorno, se conozca con mayor certeza su ocupación.

- Finalmente, en los últimos 5 minutos, se dirige al robot a la zona izquierda del entorno para detectar nuevos obstáculos, que durante su desplazamiento manual provoca que los obstáculos de la zona superior tengan ahora una mayor probabilidad de ocupación. Transcurrido este tiempo, en la figura 7.6 c) se contempla que el mapa prácticamente no sufre alteraciones respecto a la observación anterior. Se concluye la exploración alcanzados los 30 minutos de duración.



(a) (b) (c)

Figura 7.6: Prueba combinada del segundo entorno

De la pruebas resultantes se concluye que el modo exploración cumple con su misión, no obstante, se observan zonas que el robot no ha llegado a explorar. En el modo manual, se comprueba como, dirigiendo al robot a la zonas de interés, se mejora el resultado. Por tanto, se puede resumir, que la que combinación de ambos modos da lugar a un resultado más óptimo para realizar la exploración.

CAPÍTULO 8

Conclusiones y líneas futuras

8.1. Conclusiones

El desarrollo del Trabajo Fin de Grado ha supuesto un reto y un gran esfuerzo. La construcción del robot ha implicado tener que lidiar con numerosos problemas y obstáculos, como la selección de los componentes adecuados para su montaje y la construcción del *LIDAR*, que ha conllevado tener que probar distintos prototipos hasta alcanzar el más adecuado.

Por otro lado, la fase de la calibración, también, ha requerido un esfuerzo para superar las dificultades surgidas en la precisión del control del movimiento del robot, así como, un estudio del comportamiento del *LIDAR* para el ajuste de sus medidas.

Por último, el desarrollo del software ha permitido afianzar y avanzar en distintos lenguajes de programación y la adquisición de conocimientos, como el desarrollo de aplicaciones y la programación en tiempo real.

Por otro lado, el trabajo realizado ha cumplido con los objetivos del sistema:

- Se ha construido un robot móvil que, junto con sus actuadores y sensores, tiene la capacidad de desplazarse sobre un entorno y leer la distancia hasta sus obstáculos. La calibración aplicada a sus componentes se ha cumplido con éxito, lo que permite conocer con precisión la localización del robot de acuerdo con los pasos ejecutados por los motores.
- Además, se ha logrado ejecutar distintas tareas en tiempo real, de manera que el robot ha sido capaz de moverse por el entorno mientras se tomaban lecturas y se transmitían al servidor. Se podría haber desarrollado un algoritmo de exploración autónoma más complejo, que permitiese la navegación por el entorno más eficiente, sin embargo, el algoritmo aplicado cumple con la función de movimiento del robot evitando obstáculos.
- La generación del mapa ha permitido reflejar el entorno en función de probabilidades, basadas en las distintas lecturas que ha tomado el robot sobre un mismo obstáculo, que utiliza una escala de grises para representarlas. De esta forma se ha conseguido diferenciar las zonas ocupadas entre las zonas libres

- La aplicación de control desarrollada es capaz de mostrar el resultado del mapa generado y permite el control del sistema, habilitando al usuario distintos modos de exploración y modificar su dirección de movimiento.
- Así mismo, se ha logrado una comunicación eficiente de todos los componentes del sistema, obteniendo el mapa en la aplicación a partir de los datos robot.

Si bien es cierto que el objetivo propuesto en el trabajo se ha alcanzado positivamente, la utilización de componentes de mayor calidad y aplicación de métodos más avanzados conllevaría a obtener un resultado de mayor eficiencia.

8.2. Líneas futuras

Como líneas de expansión para la escalabilidad y mejora, este proyecto tiene distintas posibilidades:

- Incluir diferentes algoritmos para la exploración autónoma, que sean seleccionables desde la aplicación.
- Añadir nuevas funcionalidades seleccionables desde la aplicación, como la navegación punto a punto de forma autónoma.
- Incluir un control de batería, de manera que el robot regrese al inicio cuando se encuentre a un nivel bajo de batería.
- Añadir más sensores al *LIDAR*, de forma que se obtenga información de distintas fuentes por cada rotación.
- Añadir la posibilidad de configurar el sistema en tiempo real desde la aplicación.
- Incluir diferentes algoritmos para la representación del entorno.

Bibliografía

[1] *ROS- ROS Wiki*. Ros.org, 2021. URL: http://wiki.ros.org/es/ROS/Introduccion#:~:text=En\%20cambio\%2C\%20el\%20objetivo\%20primario,acoplable\%20a\%20los\%20dem\%C3\%A1s\%20procesos..

[2] *MQTT - Guía con Toda la información que debes saber de este sistema*. URL: https://descubrearduino.com/mqtt-que-es-como-se-puede-usar-y-como-funciona/.

[3] *Protocolos de comunicación para IoT*. URL: https://www.luisllamas.es/protocolos-de-comunicacion-para-iot/.

[4] *Qué son y cómo usar los Topics en MQTT correctamente*. URL: https://www.luisllamas.es/que-son-y-como-usar-los-topics-en-mqtt-correctamente/.

[5] *FreeRTOS - frwiki.wiki*. Frwiki.wiki, dic. de 2020. URL: https://es.frwiki.wiki/wiki/FreeRTOS.

[6] *Round-robin (informática) - frwiki.wiki*. Frwiki.wiki, 2023. URL: https://es.frwiki.wiki/wiki/Round-robin_(informatique).

[7] Colaboradores de. *técnica de navegación utilizada por robots y vehículos autónomos*. Wikipedia.org, mayo de 2008. URL: https://es.wikipedia.org/wiki/Localizaci\%C3\%B3n_y_modelado_simult\%C3\%A1neos.

[8] Franklin Pineda Torres. «Técnicas de slam con filtros probabilísticos; caracterización y resultados en robots móviles». En: *Mundo FESC* 9 (2019), págs. 7-15. DOI: https://dialnet.unirioja.es/descarga/articulo/7452802.pdf. URL: https://dialnet.unirioja.es/servlet/articulo?codigo=7452802.

[9] Victor Roman. *Algoritmos Naive Bayes: Fundamentos e Implementación*. Medium, abr. de 2019. URL: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci\%C3\%B3n-4bcb24b307f.

[10] Cipriano Galindo Andrades. «Robot Programming and Control Mobile Robotics (III)». En: ().

[11] Jugesh Sundram, Harry Duong, Gim Song Soh y Kristin Lee Wood. *Development of a Miniature Robot for Multi-robot Occupancy Grid Mapping*. ResearchGate, jul. de 2018. URL: https://www.researchgate.net/publication/325809276_Development_of_a_Miniature_Robot_for_Multi-robot_Occupancy_Grid_Mapping.

[12] Ibáñez. *Qué es un LIDAR, y cómo funciona el sensor más caro de los coches autónomos*. Motorpasion.com, sep. de 2017. URL: https://www.motorpasion.com/tecnologia/que-es-un-lidar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser.

[13] *About Arduino*. Arduino.cc, 2015. URL: https://www.arduino.cc/en/about.

[14] *Introducción a Android Studio — Desarrolladores de Android — Android Developers*. Android Developers, 2022. URL: https://developer.android.com/studio/intro.

[15] Uptodown Technologies SL. *Eclipse IDE (Windows)*. Uptodown.com, mar. de 2021. URL: https://eclipse.uptodown.com/windows.

[16] Spyder Team. *Home — Spyder IDE*. Spyder-ide.org, 2022. URL: https://www.spyder-ide.org/.

[17] jolhfred. *¿Porqué son considerado las 10 mejores bibliotecas de procesamiento de imágenes en Python?* Somoslibres.org, mayo de 2022. URL: https://www.somoslibres.org/index.php/48-nieuws/programacion/python/11153-porque-son-considerado-las-10-mejores-bibliotecas-de-procesamiento-de-imagenes-en-python.

[18] jecrespom. *Mosquitto*. Aprendiendo Arduino, nov. de 2018. URL: https://aprendiendoarduino.wordpress.com/2018/11/19/mosquitto/.

[19] *Welcome to Fritzing*. Fritzing.org, 2022. URL: https://fritzing.org/.

[20] *SOLIDWORKS - Qué es y para qué sirve*. SolidBI, nov. de 2022. URL: https://solid-bi.es/solidworks/.

[21] Redacción. *Qué es una placa SBC o Single Board Computer — Descubrearduino.com*. Descubrearduino.com, feb. de 2020. URL: https://descubrearduino.com/sbc/.

[22] Colaboradores de. *Arquitectura del sistema*. Wikipedia.org, mayo de 2002. URL: https://es.wikipedia.org/wiki/Sistema#:~:text=Un\%20sistema\%20es\%20un\%20conjunto,generar\%20salidas\%2C\%20resultados\%20del\%20sistema..

[23] *Caso de uso - EcuRed*. Ecured.cu, 2023. URL: https://www.ecured.cu/Caso_de_uso.

[24] admin. *¿Qué son los requisitos funcionales? Especificación, tipos, EJEMPLOS*. Ebooks Online, 2020. URL: https://ebooksonline.es/que-es-un-requisito-funcional-especificacion-tipos-ejemplos/.

[25] pmoinformatica.com. *Requerimientos no funcionales: Ejemplos*. Pmoinformatica.com, 2015. URL: http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html.

[26] *1.1.1 Locomoción*. Iearobotics.com, 2023. URL: http://www.iearobotics.com/personal/juan/doctorado/tea/html/node8.html#:~:text=La\%20locomoci\%C3\%B3n\%20es\%20la\%20facultad,locomotiva\%20se\%20llaman\%20robots\%20m\%C3\%B3viles.&text=La\%20complejidad\%20depende\%20del\%20tipo,haga\%20un\%20robot\%20con\%20patas..

[27] *Modelado de seguimiento de caminos*. idp.uma.es. URL: https://eii.cv.uma.es/pluginfile.php/347662/mod_resource/content/12/Robots\%20moviles.\%20Modelado\%20de\%20vehiculos\%20y\%20seguimiento\%20de\%20caminos_2021-2022.pdf.

[28] *Motores: Tipos y Características*. Murkyrobot.com, 2023. URL: https://www.murkyrobot.com/guias/actuadores/motores-tipos.

[29] *Motor paso a paso – tipos y ejemplos del uso de motores paso a paso*. Www.tme.eu, sep. de 2020. URL: https://www.tme.eu/es/news/library-articles/page/41861/Motor-paso-a-paso-tipos-y-ejemplos-del-uso-de-motores-paso-a-paso/.

[30] *2.4.- La ley de Ohm*. Xunta.gal, 2023. URL: https://www.edu.xunta.gal/espazoAbalar/sites/espazoAbalar/files/datos/1464947843/contido/24_la_ley_de_ohm.html.

[31] *DESCRIPCIÓN DEL DRIVER A4988 – Electrónica Práctica Aplicada*. Diarioelectronicohoy.com, mar. de 2020. URL: https://www.diarioelectronicohoy.com/blog/descripcion-del-driver-a4988.

[32] *Arduino homemade LIDAR sensor distance Infrared*. Electronoobs.com, 2022. URL: https://electronoobs.com/eng_arduino_tut110_sch1.php.

[33] Instructables. *Project Lighthouse - 360° Mini Arduino LiDAR*. Instructables, dic. de 2020. URL: https://www.instructables.com/Project-Lighthouse-360-Mini-Arduino-LiDAR/.

[34] *MOTOR DE PASOS 28BYJ-48*. Puntoflotante.net, 2023. URL: https://www.puntoflotante.net/MOTOR-DE-PASOS-28BYJ-48.htm.

[35] inventable. *El ULN2003. driver de salida para microcontroladores — Inventable.eu*. Inventable, feb. de 2018. URL: https://www.inventable.eu/2018/02/09/uln2003-driver-salida-microcontroladores/.

[36] https://www.facebook.com/jose.guerracarmenate. *ESP32*. Programar fácil con Arduino, feb. de 2021. URL: https://programarfacil.com/esp8266/esp32/.

[37] *DISTRIBUCIÓN NORMAL*. URL: https://www.uv.es/ceaces/pdf/normal.pdf.

[38] *Exclusión mutua - frwiki.wiki*. URL: https://es.frwiki.wiki/wiki/Exclusion_mutuelle.

[39] *Colas - frwiki.wiki*. Frwiki.wiki, 2023. URL: https://es.frwiki.wiki/wiki/File_(structure_de_donn\%C3\%A9es).

[40] Ricardo-Franco Mendoza-Garcia y R Mendoza-Garcia. *Fundamentos de Robótica Herramientas Matemáticas para la Localización Espacial Matrices de Transformación Homogéneas*. 2014. URL: http://www.eudim.uta.cl/rmendozag/courses/fundamentos_robotica/lectures/matrices_homogeneas.pdf.

[41] *Bresenham*. Chello.at, 2016. URL: http://members.chello.at/~easyfilter/bresenham.html.

[42] Francisco Urdinez. *Capítulo 8 Modelos logísticos — AnalizaR Datos Políticos*. Github.io, 2019. URL: https://arcruz0.github.io/libroadp/logit.html.

Apéndices

APÉNDICE A

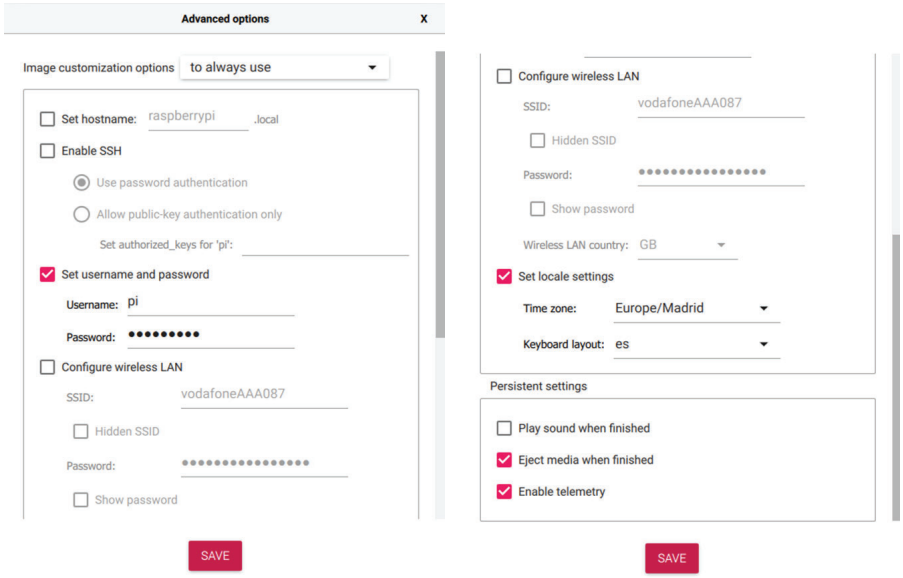
Instalación de Raspbian

En este proyecto se va a emplear una Raspberry Pi como el dispositivo para cliente cartógrafo y el servidor del sistema. En primer lugar, será necesario instalar el sistema operativo del dispositivo y en este caso, se ha optado por usar el sistema de Linux - aspbian".

Raspbian es el sistema operativo recomendado por Raspberry Pi y se basa en una distribución de GNU/Linux llamada "Debian". Para su instalación existen dos versiones: una con interfaz gráfica y otra en modo consola de comando. En este caso se ha elegido por la versión con entorno gráfico. Además, su instalación se puede llevar acabo con un asistente, llamado "NOOBS", o mediante la imagen del sistema operativo, figura A.1. En este caso se ha decidido instalar la imagen desde la página oficial y montarla usando cualquier programa para escribir imágenes ISO o IMG. Una de las opciones es "Balena Etcher", no obstante, la página oficial de Raspberry ofrece su propio programa, "Raspberry Pi Imager", por lo que se ha elegido esta última opción.

Figura A.1: Raspberry Pi Imager

Tras la instalación, se ejecuta el programa, que ofrece elegir las opciones para montar la imagen. Entre ellas se elige el sistema operativo, en concreto el recomendado "Raspberry PI OS (32-bit)", la unidad de almacenamiento, una micro SD de 128 Gb, y se abren los ajustes figura , figura A.2. En ellos, se puede modificar parámetros como la red por defecto, contraseña y usuario e idioma.



(a) (b)

Figura A.2: Opciones de la instalación de la imagen

Una vez elegidos los parámetros se pulsa "Write" y se espera. Montada la imagen en la unidad, se introduce en la Raspberry y se inicia comprobando su correcta instalación. Sin embargo, se producen un error al iniciar el sistema, figura A.3 y es necesario formatear la unidad. En este segundo intento se instala el sistema operativo "Raspberry PI OS FULL (32-bit)", se verifica y se comprueba que ha sido realizada con éxito.

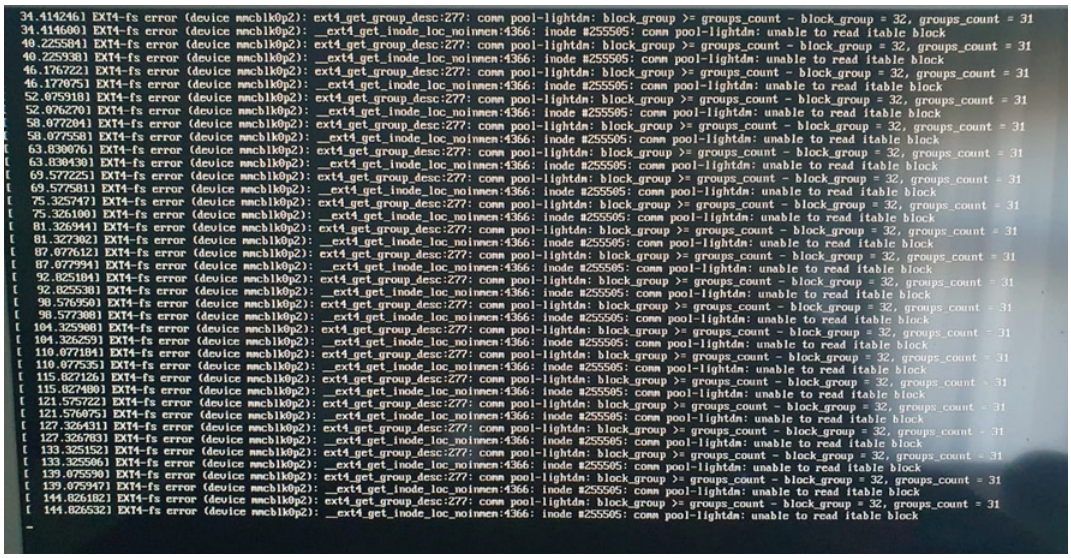


Figura A.3: Error en la escritura de la imagen

APÉNDICE B

Instalación de Mosquitto

Una vez instalado el sistema operativo en la Raspberry, se pretende instalar el servidor del sistema en ella. Para ello, se ha elegido el servidor "Eclipse Mosquitto". Se trata de un servidor de mensajes open-source que implementa el protocolo MQTT actuando como bróker o intermediario de mensajes. La implementación de Mosquitto tiene un ejecutable de 120 kB que consume aproximadamente 3MB de RAM y se han realizado pruebas exitosas con incluso 100.000 clientes conectados.

Antes de la instalación de Mosquito se comprueba el estado de la conexión de la raspberry usando el comando "ping", conociendo así la velocidad y calidad de la red. Se realiza un ping a la IP de la raspberry, que se ha conocido realizando el comando "ifconfig" en su terminal. Se observa que la conexión es estable y rápida, figura B.1.

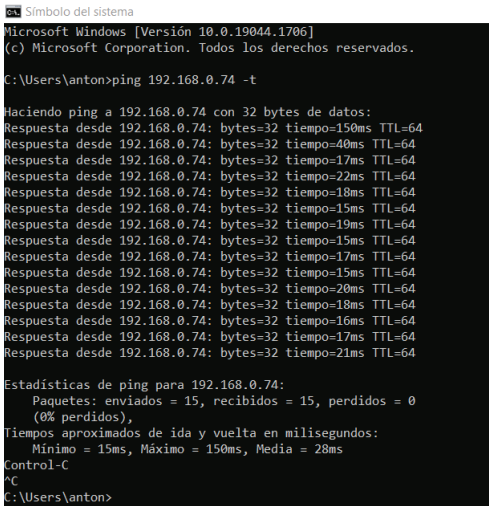


Figura B.1: Ping Raspberry

A continuación, se realiza la preparación de la instalación comprobando la versión de la Raspberry, figura B.2. Conocida esta, se accede al repositorio de mosquitto de nuestra versión Raspbian importando la llave y se actualiza la base de datos.

```
pi@raspberrypi:~ $ lsb_release -cdr
Description:    Raspbian GNU/Linux 11 (bullseye)
Release:        11
Codename:       bullseye
pi@raspberrypi:~ $ wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
--2022-05-31 19:39:03--  http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
Resolviendo repo.mosquitto.org (repo.mosquitto.org)... 85.119.83.194, 2001:ba8:1f1:f271::2
Conectando con repo.mosquitto.org (repo.mosquitto.org)[85.119.83.194]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 3167 (3,1K) [application/octet-stream]
Grabando a: «mosquitto-repo.gpg.key»

mosquitto-repo.gpg.key                  100%[=======================================================================================>]   3,09K  --.-KB/s    en 0,00

2022-05-31 19:39:04 (926 KB/s) - «mosquitto-repo.gpg.key» guardado [3167/3167]

pi@raspberrypi:~ $ cd /etc/apt/sources.list.d/
pi@raspberrypi:/etc/apt/sources.list.d $ sudo wget http://repo.mosquitto.org/debian/mosquitto-bulleyse.list
--2022-05-31 19:40:53--  http://repo.mosquitto.org/debian/mosquitto-bulleyse.list
Resolviendo repo.mosquitto.org (repo.mosquitto.org)... 85.119.83.194, 2001:ba8:1f1:f271::2
Conectando con repo.mosquitto.org (repo.mosquitto.org)[85.119.83.194]:80... conectado.
Petición HTTP enviada, esperando respuesta... 404 Not Found
2022-05-31 19:40:53 ERROR 404: Not Found.

pi@raspberrypi:/etc/apt/sources.list.d $ sudo wget https://repo.mosquitto.org/debian/mosquitto-bullseye.list
--2022-05-31 19:43:49--  https://repo.mosquitto.org/debian/mosquitto-bullseye.list
Resolviendo repo.mosquitto.org (repo.mosquitto.org)... 85.119.83.194, 2001:ba8:1f1:f271::2
Conectando con repo.mosquitto.org (repo.mosquitto.org)[85.119.83.194]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 52 [application/octet-stream]
Grabando a: «mosquitto-bullseye.list»

mosquitto-bullseye.list                 100%[=======================================================================================>]      52  --.-KB/s    en 0s

2022-05-31 19:43:50 (17,6 MB/s) - «mosquitto-bullseye.list» guardado [52/52]

pi@raspberrypi:/etc/apt/sources.list.d $
```

Figura B.2: Preparación Raspberry

Tras el comando 'sudo apt-get update' se obtiene el siguiente error: 'Las firmas siguientes no se pudieron verificar porque su llave pública no está disponible'. Para resolverlo se debe obtener la llave pública ejecutando los comandos de la figura B.3.

```
pi@raspberrypi:~ $ gpg --keyserver keyserver.ubuntu.com --recv 61611AE430993623
gpg: creado el directorio '/home/pi/.gnupg'
gpg: caja de claves '/home/pi/.gnupg/pubring.kbx' creada
gpg: /home/pi/.gnupg/trustdb.gpg: se ha creado base de datos de confianza
gpg: clave 61611AE430993623: clave pública "Mosquitto Apt Repository <repo@mosquitto.org>" importada
gpg: Cantidad total procesada: 1
gpg:               importadas: 1
pi@raspberrypi:~ $ gpg --armor --export 61611AE430993623|sudo apt-key add-
Usage: apt-key [--keyring file] [command] [arguments]

Manage apt's list of trusted keys

  apt-key add <file>          - add the key contained in <file> ('-' for stdin)
  apt-key del <keyid>         - remove the key <keyid>
  apt-key export <keyid>      - output the key <keyid>
  apt-key exportall           - output all trusted keys
  apt-key update              - update keys using the keyring package
  apt-key net-update          - update keys using the network
  apt-key list                - list keys
  apt-key finger              - list fingerprints
  apt-key adv                 - pass advanced options to gpg (download key)

If no specific keyring file is given the command applies to all keyring files.
pi@raspberrypi:~ $ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 61611AE430993623
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
Executing: /tmp/apt-key-gpghome.pdLPnXH7gX/gpg.1.sh --keyserver keyserver.ubuntu.com --recv-keys 61611AE430993623
gpg: clave 61611AE430993623: clave pública "Mosquitto Apt Repository <repo@mosquitto.org>" importada
gpg: Cantidad total procesada: 1
gpg:               importadas: 1
pi@raspberrypi:~ $ sudo apt-get update
Obj:1 http://raspbian.raspberrypi.org/raspbian bullseye InRelease
Obj:2 http://archive.raspberrypi.org/debian bullseye InRelease
Des:3 https://repo.mosquitto.org/debian bullseye InRelease [12,5 kB]
Des:4 https://repo.mosquitto.org/debian bullseye/main armhf Packages [4.976 B]
Des:5 https://repo.mosquitto.org/debian bullseye/main all Packages [1.208 B]
Descargados 18,6 kB en 2s (12,0 kB/s)
Leyendo lista de paquetes... Hecho
pi@raspberrypi:~ $
```

Figura B.3: Error de update Raspberry

Después de la preparación se inicia la instalación. De nuevo surge un error y se sugiere un comando que devolverá el motivo del error. Se trata de un paquete que es necesario actualizarse, no obstante, ya que no se va a usar, se elimina y se continúa con la instalación, figura B.4.

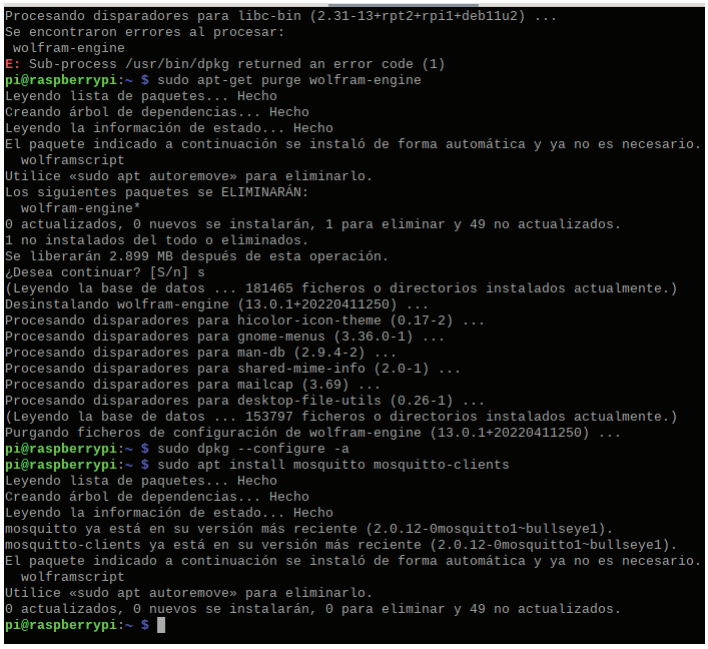


Figura B.4: Instalación Mosquitto

Se habilita el inicio automático de MQTT con el reinicio de la Raspeberry usando el comando: "sudo systemctl enable mosquitto". Después se realiza una prueba de funcionamiento: se inicia MQTT y luego se comprueba su estado, figura B.5.

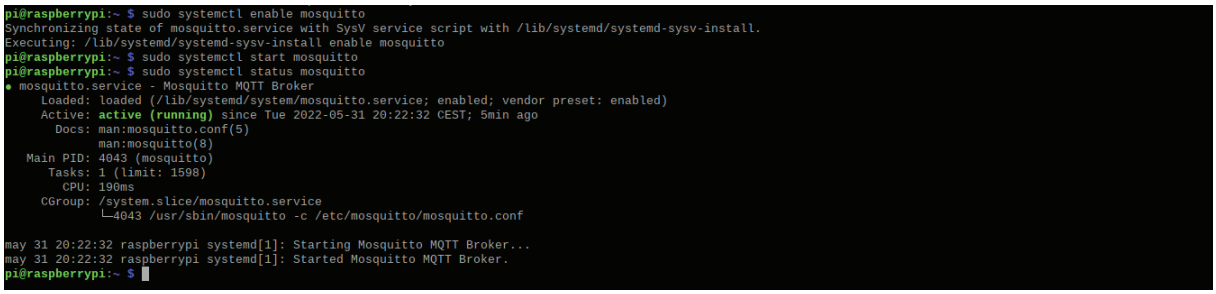


Figura B.5: Estado MQTT

Para realizar simulaciones desde consola Mosquitto proporciona 2 clientes para la publicación y suscripción: mosquitto_pub y mosquitto_sub. En las pruebas que se observan en las figura B.6 y B.7 se observan distintos flags que se explican a continuación:

- -d permite ejecutar mosquitto en segundo plano
- -h establece la dirección IP
- -p especifica el puerto de la conexión
- -t identifica el topic
- -m identifica el mensaje a enviar al broker

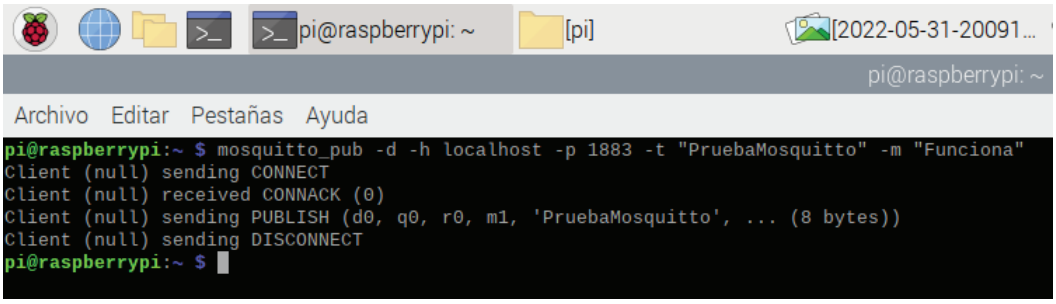


Figura B.6: Ejemplo de publicación

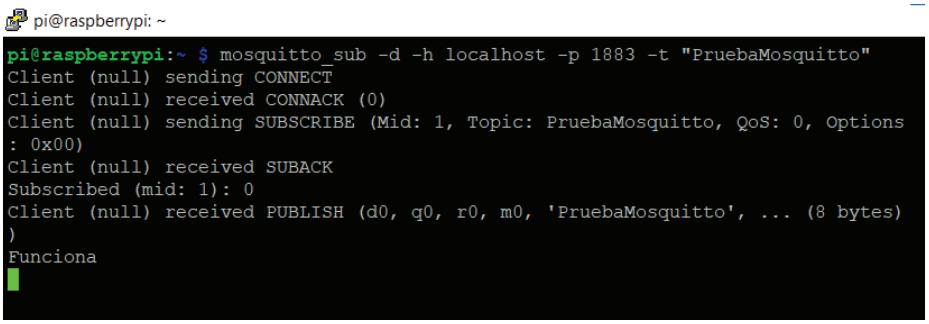


Figura B.7: Ejemplo de suscripción

APÉNDICE C

Código del robot

C.0.1. Robot.h

```
#ifndef ROBOT_H
#define ROBOT_H

//------------------------------
//          LIBRERIAS
//------------------------------
#include <Arduino.h>
#include "Lidar.h"
#include "Movimiento.h"
#include <WiFi.h>           //Comunicación Wifi ESP32
#include <PubSubClient.h> //Mqtt
#include <ArduinoJson.h>   //JSON

//-----------------------------------------
//            DATOS Y DECLARACIONES
//-----------------------------------------
//COMUNICACIONES
//WiFi
const char* const red_ssid = "_____";
const char* const red_password = "_____";

//VALORES DE LA RED
const char* const mqtt_server = "_____";

//MQTT
const char* const mqtt_user = "Antonio";
const char* const mqtt_pass = "Antonio";

//TOPICS
const char* const tPub_medidaObs= "robot/medidaObs";
const char* const tSub_accion="control/accion";

//CONSTANTES DE ACCIONES
const char Cancelar='C';          //Cancelar mvto motor y mvto LIDAR

//NUCLEOS ESP32
const BaseType_t pro_cpu0 = 0; //Núcleo para comunicaciones o CPU de ↩
    protocolo, se ejecuta el SO y se controla el WiFi. No se debe ↩
```

```
   interrumpir
const BaseType_t app_cpu1 = 1; //Núcleo para aplicaciones o CPU de ↩
   aplicación, se ejecuta el loop

//FUNCIONES
void conecta_wifi();
void conecta_mqtt();
void callback(char* topic, byte* payload, unsigned int length);
void subscribeMQTT();

//TAREAS
void lidarTask(void *pvParameters);          //Tarea control y lectura del↩
    lidar
void movimientoTask(void *pvParameters);     //Tarea control del ↩
   movimiento del robot
void sendMQTTTask(void *pvParameters);       //Tarea que envía por mqtt
void receiveMQTTTask(void *pvParameters);    //Tarea que recibe por mqtt

//PRIORIDADES TAREAS
//La mayor prioridad tendrá el valor más alto
#define LIDAR_TASK_PRIORITY                  2
#define MOVIMIENTO_TASK_PRIORITY             2
#define SEND_MQTT_TASK_PRIORITY              3
#define RECEIVE_MQTT_TASK_PRIORITY           1

//TAMAÑO MEMORIA TAREA
#define LIDAR_TASK_MEM_SIZE                  3072
#define MOVIMIENTO_TASK_MEM_SIZE             2048
#define SEND_MQTT_TASK_MEM_SIZE              4096
#define RECEIVE_MQTT_TASK_SIZE               2048

//PERIODO TAREA
#define LIDAR_TASK_MS                80
#define MOVIMIENTO_TASK_MS           500
#define SEND_MQTT_TASK_MS            30
#define RECEIVE_MQTT_TASK_MS         50


//COLAS
static const uint8_t Medidas_msg_queue_len = 10;

#endif // Fin de guarda
```

C.0.2. Robot.ino

```
//------------------------------

//          LIBRERIAS
//------------------------------
#include "Robot.h"

//-----------------------------------------
//            VARIABLES GLOBALES
//-----------------------------------------
//LIDAR
```

```
datosLIDAR scanLIDAR;

//MVTO
pose poseRobot;

//COMUNICACIONES
WiFiClient wClient;
PubSubClient mqtt_client(mqtt_server, 1883, wClient); //Iniciamos la ↩
    libreria con el cliente que se obtine de mqtt

//SEMAFOROS
static SemaphoreHandle_t mutex;

//COLAS
static QueueHandle_t Medidas_msg_queue;

//OTRAS
char Accion = Cancelar;


//↩
    ------------------------------------------------------------------------------------------------

//                                                  SET UP
//↩
    ------------------------------------------------------------------------------------------------

void setup() {

  Serial.begin(115200);
   //Espera para comenzar (para que no perder la salida en serie)
  vTaskDelay(1000 / portTICK_PERIOD_MS);
  Serial.println();
  Serial.println("Empieza setup...");

  //INICIALIZAR COMUNICACIONES WiFi y MQTT----------
  conecta_wifi();
  mqtt_client.setServer(mqtt_server, 1883);
  mqtt_client.setBufferSize(512);
  mqtt_client.setCallback(callback);
  conecta_mqtt();

  //MUTEX-------------------------------------------

  if ( mutex == NULL ) //Nos aseguramos que el semaforo no se encuentra ↩
     creado, y en ese caso lo creamos
  {
    mutex = xSemaphoreCreateMutex();

    if ( ( mutex ) != NULL ) // Ponemos el semáforo como disponible
      xSemaphoreGive( ( mutex ) );
  }

  //COLAS-------------------------------------------
  Medidas_msg_queue = xQueueCreate(Medidas_msg_queue_len, sizeof(String));
  Serial.println("Declarado Semaforo y colas");

  //INICIALIZAR TAREAS------------------------------
```

```
  //Tarea Movimiento y lectura LIDAR
  xTaskCreatePinnedToCore(lidarTask,
                          "lidarTask",
                          LIDAR_TASK_MEM_SIZE,
                          NULL,
                          LIDAR_TASK_PRIORITY,
                          NULL,
                          app_cpu1);
  Serial.println("Declarado Tarea Movimiento y lectura LIDAR");

  //Tarea Movimiento del robot
  xTaskCreatePinnedToCore(movimientoTask,
                          "movimientoTask",
                          MOVIMIENTO_TASK_MEM_SIZE,
                          NULL,
                          MOVIMIENTO_TASK_PRIORITY,
                          NULL,
                          app_cpu1);
  Serial.println("Declarado Tarea Movimiento del robot");
  //Tarea enviar por MQTT
  xTaskCreatePinnedToCore(sendMQTTTask,
                          "sendMQTTTask",
                          SEND_MQTT_TASK_MEM_SIZE,
                          NULL,
                          SEND_MQTT_TASK_PRIORITY,
                          NULL,
                          app_cpu1);
  Serial.println("Declarado Tarea enviar por MQTT");

  //Tarea recibir de MQTT
  xTaskCreatePinnedToCore(receiveMQTTTask,
                          "receiveMQTTTask",
                          RECEIVE_MQTT_TASK_SIZE,
                          NULL,
                          RECEIVE_MQTT_TASK_PRIORITY,
                          NULL,
                          pro_cpu0);//app_cpu1
  Serial.println("Declarado Tarea recibir de MQTT");

  Serial.printf("Termina setup en %lu ms\n\n", millis());

  //Eliminar tarea Setup y Loop, para asegurar que solo hay una tarea ↩
     corriendo
  vTaskDelete(NULL);

}

//                                                    LOOP
//↩
     ---------------------------------------------------------------------------

void loop() {}


//↩
     ---------------------------------------------------------------------------

//                                                    TAREAS
```

```
//↩
-----------------------------------------------------------------------------------

//------------------------------lidarTask-------------------------------------
void lidarTask(void *pvParameters) {

  bool retornoOK=false;
  setupLIDAR();
  while (true) {

    if(Accion==Cancelar){
      if(not retornoOK){
        //En caso de haber alcanzado previamente los 0 grados no lanzo de ↩
            nuevo la función
        retornoLIDAR();
        retornoOK=true;
      }
    }
    else{
      retornoOK=false; //Reactivamos el movimiento del LIDAR

      //Tomo medida
      scanLIDAR = controlLIDAR();

      if (xSemaphoreTake(mutex, 0) == pdTRUE) {
        //Se toma el mutex bloquea la sección evitando que se modifique el↩
             valor de poseRobot

        String mensaje_medidas;
        StaticJsonDocument<192> Medidas;//Construyo mensaje medidas
        //Publico en cm y radianes
        Medidas["x"] =poseRobot.x*100;
        Medidas["y"] =poseRobot.y*100;
        Medidas["theta"] =poseRobot.theta;
        Medidas["d_obs"] = scanLIDAR.distancia;
        Medidas["alfa_obs"] = scanLIDAR.angulo*(Pi/180);
        serializeJsonPretty(Medidas, mensaje_medidas); //Serializo para ↩
            mandar a la cola
        Serial.println(mensaje_medidas);
        //Añado a la cola si queda memoria en ella
        if (xQueueSend(Medidas_msg_queue,(void *)&mensaje_medidas, ↩
            Medidas_msg_queue_len) != pdTRUE) {
             Serial.println("Cola Medidas Medidas_msg_queue llena");
        }
        xSemaphoreGive(mutex);
      }
    }
     vTaskDelay(LIDAR_TASK_MS / portTICK_PERIOD_MS);
  }

}

//------------------------------movimientoTask--------------------------------
void movimientoTask(void *pvParameters) {

  setupMotores();
  while (true) {
    ControlMotor(Accion, scanLIDAR.angulo, scanLIDAR.distancia);
```

```
    //Calculo posición
    if (xSemaphoreTake(mutex, 0) == pdTRUE) {
      //Se toma el mutex bloquea la sección para evitar modificar el valor↩
          de poseRobot
      poseRobot=estimar_pos();
      xSemaphoreGive(mutex);
    }

    vTaskDelay(MOVIMIENTO_TASK_MS / portTICK_PERIOD_MS);
  }
}

//------------------------------sendMQTTTask-------------------------------------
void sendMQTTTask(void *pvParameters) {

  String msg_medidas;
  while (true) {
    //Comprobamos la conexión
    if (!mqtt_client.connected()) {
        conecta_mqtt();
    }
    //Si hay datos en la cola
    if (xQueueReceive(Medidas_msg_queue, (String *)&msg_medidas, 0) == ↩
       pdTRUE)
    {
      mqtt_client.publish(tPub_medidaObs,msg_medidas.c_str());
      Serial.printf("Publicando a [%s] \n", tPub_medidaObs);
    }

    vTaskDelay(SEND_MQTT_TASK_MS / portTICK_PERIOD_MS);
  }
}
//------------------------------receiveMQTTTask----------------------------------
void receiveMQTTTask(void *pvParameters) {

  while (true) {
    //Comprobamos la conexión
    if (!mqtt_client.connected()) {
      conecta_mqtt();
    }
    mqtt_client.loop();
    // Permitir al cliente que procese los mensajes entrantes y mantenga ↩
       su conexión con el servidor.

    vTaskDelay(RECEIVE_MQTT_TASK_MS / portTICK_PERIOD_MS);
  }
}

//↩
   ------------------------------------------------------------------------------

//                                                          FUNCIONES
//↩
   ------------------------------------------------------------------------------

//------------------------------CONECTA WIFI-------------------------------------
void conecta_wifi() {
```

```
  Serial.printf("\nConectando a %s: \n", red_ssid);

  //Se establece la configuración para no usar DHCP
  WiFi.mode(WIFI_STA);
  //WiFi.config(WiFiIP,WiFiNet,WiFiGateWay);
  WiFi.begin(red_ssid, red_password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
    Serial.print(".");
  }

  String IP = WiFi.localIP().toString().c_str();
  Serial.printf("\nWiFi conectado, IP: %s\n", IP.c_str());
}


//--------------------CONECTA MQTT--------------------
void conecta_mqtt() {

  while (!mqtt_client.connected()) {

    Serial.print("Conectando con MQTT...");
    String ID_PLACA= "ESP32-";
    ID_PLACA += String(random(0xffff), HEX);

    if (mqtt_client.connect(ID_PLACA.c_str(),mqtt_user,mqtt_pass)){
      Serial.print("Conectado a broker: %");
      Serial.println(mqtt_server);
      subscribeMQTT();
    }else {
      Serial.print("Fallido, rc=");
      Serial.print(mqtt_client.state());
      Serial.println("Reintenando en 5 segundos");
      // Reintenta a los 5 segundos
      delay(5000);
    }
  }
}

//--------------------SUSCRIBE--------------------
void subscribeMQTT(){
  //Suscripciones
  mqtt_client.subscribe(tSub_accion);
  Serial.println("Cliente suscripto a los siguientes topics: ");
  Serial.println(tSub_accion);
}

//--------------------CALLBACK--------------------
void callback(char* topic, byte* payload, unsigned int length) {

  char *mensaje_subs = (char *)malloc(length+1); // Reservo memoria para ↩
     copia del mensaje
  strncpy(mensaje_subs, (char*)payload, length); // Copio el mensaje en ↩
     cadena de caracteres
  mensaje_subs[length]='\0'; // Caracter cero marca el final de la cadena

  Serial.printf("Mensaje recibido [%s] %s\n", topic, mensaje_subs);
```

```
  //Compruebo que es el topic adecuado, si ambas cadenas son iguales ↩
      strcmp retorna un 0
  if(strcmp(topic,tSub_accion)==0){
    //------------TOPIC ACCION DE MOVIMIENTO
    Accion = (char)payload[0];
  }else{
    //------------TOPIC DESCONOCIDO
    Serial.println("Error: Topic desconocido");
  }

  free(mensaje_subs);
}
```

C.0.3. Movimiento.h

```
/*
 * Se hace uso de 2 stepper NEMA 17HS16-2004S con controlador Pololu A4988
 * Ambos potenciometros son de 1.24V
 * Imax del motor es 2A -> Vref= Imax*8*Ra4988=2.8.0.1=1.44
 * Usamos entre un 80% y 90% por seguridad -> en nuestro caso 86% = 1.22
 * Nema17 cuenta con 2 fases: 1.1 Ohmios/fase, 2V/fase y 2.2 A/fase
 * Se hace uso de un modulo controlador de steppers para el a4988: PotL↩
    =1'24 y PotR=1'18
 * El driver posee 3 selectores para la resolución del paso:
 *MS1---MS2---MS3---Microstep Resolution---
 *Low   Low   Low   Full step
 *High  Low   Low   Half step
 *Low   High  Low   Quarter step
 *High  High  Low   Eighth step
 *High  High  High  Sixteenth step
 *-----------------------------------------
*/

// Guarda para evitar inclusión duplicada
#ifndef MOVIMIENTO_H
#define MOVIMIENTO_H

//-------------------------------
//         LIBRERIAS
//-------------------------------
#include <Arduino.h>

//-----------------------------------------
//           DATOS Y DECLARACIONES
//-----------------------------------------
const double Pi = 3.14159265358979323846264338327795;
#define PIN_GIRORANDOM              17 //Pin giro sentido aleatorio
//CONFIGURACIÓN MOVIMIENTO
#define STEPS_X_REV                 3200 //Esto es el número de pasos por ↩
    revolución - MODO Eighth step
#define NUMSTEPS                    1600 //Número de pasos que queremos que ↩
    realice
#define TURNSTEPS                   349  //Número de pasos para girar en ↩
    modo navegar (equivale a 11.25134328 grados)
#define STEP_SPEED                  2000 //Velocidad pasos/us //2000
```

```
//PINES
//MotorRIGHT
#define STEPPER_R_DIRPIN          27
#define STEPPER_R_STEPPIN         26
#define STEPPER_R_ENABLEPIN       25 //LOW enciende el motor, HIGH lo ↩
    apaga

//MotorLEFT
#define STEPPER_L_DIRPIN          5
#define STEPPER_L_STEPPIN         18
#define STEPPER_L_ENABLEPIN       19

//Parametros Constantes de Movimiento
const double distMin = 30; //En cm
const double LimAngulo = 60; //En grados
const int rangInf = 1861; //Aproximadamente 60 grados de giro
const int rangSup = 3722; //Aproximadamente 120 grados de giro
//Una vuelta entera de rueda son 3200 pasos que provoca un giro aprox de ↩
    51.5820895 grados

//CONSTANTES DE ACCIONES
const char Explora='E';          //Exploración
const char Avanzar='A';          //Avanzar con paso constante
const char Retroceder='R';    //Retroceder con paso constante
const char GiroIzq='I';          //Girar Izq con paso constante
const char GiroDrch='D';        //Girar Drchcon paso constante

//CONSTANTES CÁLCULO DE POSE
const double radRueda = 0.0336; //En m
const double distRuedas = 0.2245; //En m

//CONSTANTES DE CALIBRACIÓN
const int calib1_recto = 500; //Pasos para 1 correccion del mvto recto
const int calib2_recto = 1010; //Pasos para 2 correcion del mvto recto
const int calib_gd = 56;//Pasos para corregir el giro a la derecha
const int calib_gi = 67;//Pasos para corregir el giro a la izquierda
const double calib_dist = 0.985;//Factor de escala que se multiplica a la ↩
    medida

//ODOMETRIA
struct pose {
  double x;
  double y;
  double theta;
};

//FUNCIONES
void setupMotores();
void ControlMotor(char comando, double dbAnguloObs, double dbPosObs);
void arranque();
void avanzar();
void retroceder();
void giroDerecha(int giroRsteps);
void giroIzquierda(int giroLsteps);
void parar();
struct pose estimar_pos();
```

```
#endif // Fin de guarda
```

C.0.4. Movimiento.cpp

```
//-----------------------------
//          LIBRERIAS
//-----------------------------
#include "Movimiento.h"

//-------------------------------------------
//            DATOS Y DECLARACIONES
//-------------------------------------------
//Variables posición y orientación
pose pose_prev;
pose pose_act;

int steps_ML = 0; //Pasos que da la rueda izq
int steps_MR = 0; //Pasos que da la rueda drch

//-------------------------------------------------------------------
//                           FUNCIONES
//-------------------------------------------------------------------

//-------------------------setupMotores-----------------------------
void setupMotores(){

  pinMode(STEPPER_R_DIRPIN, OUTPUT);
  pinMode(STEPPER_R_STEPPIN, OUTPUT);
  pinMode(STEPPER_R_ENABLEPIN, OUTPUT);

  pinMode(STEPPER_L_DIRPIN, OUTPUT);
  pinMode(STEPPER_L_STEPPIN, OUTPUT);
  pinMode(STEPPER_L_ENABLEPIN, OUTPUT);

  arranque();

  pose_act.x=0;
  pose_act.y=0;
  pose_act.theta=0;
  pose_prev=pose_act;

}

//-------------------------ControlMotor-----------------------------
void ControlMotor(char comando, double dbAnguloObs, double dbPosObs){

  switch(comando){
    case Explora:
      Serial.printf("Exploración distancia: %fcm en un angulo de: %f ↩
          grados\n",dbPosObs,dbAnguloObs);

      //Si tengo un obstáculo en el camino (a <30cm y entre +-60 grados)
      if(dbPosObs<distMin and (dbAnguloObs<LimAngulo or dbAnguloObs>(360-↩
          LimAngulo)) ){

        //Movimiento de evasión
```

```
        Serial.println("Esquivar");
        retroceder();

        randomSeed(analogRead(PIN_GIRORANDOM)); //Mide un pin no usado y ↩
            la semilla tendrá como referencia el valor de ruido que lea
        int randomGiro = random(1,2);
        int randomStep = random(rangInf,rangSup); //Giro aleatorio, entre↩
             60 y 120 grados aproximadamente, en una dirección aleatoria


        switch(randomGiro){
           case(1): giroIzquierda(randomStep);break;
           case(2): giroDerecha(randomStep); break;
        }
      }else{ // obstáculo detrás o no hay, continúo recto
        avanzar();
      }
      break;
    case Avanzar:
      avanzar();
      break;
    case Retroceder:
      retroceder();
      break;
    case GiroIzq:
      giroIzquierda(TURNSTEPS); break;
    case GiroDrch:
      giroDerecha(TURNSTEPS); break;
    default:
      parar();
  }

}

//-------------------------------ODOMETRÍA-----------------------------------
struct pose estimar_pos(){

  //Cálculo en radianes y metros
  double dist=0.5*((double)(steps_MR+steps_ML)/(double)(STEPS_X_REV))*(2*↩
      Pi*radRueda);
  dist=dist*calib_dist;//Serial.printf("Dist %f\n",dist);
  double deltaTheta=(1/distRuedas)*((double)(steps_MR-steps_ML)/(double)(↩
      STEPS_X_REV))*(2*Pi*radRueda);
  //Serial.printf("DeltaTheta %f\n",deltaTheta);

  pose_act.x= pose_prev.x + dist*cos(pose_prev.theta + deltaTheta/2);
  //Serial.printf("XAct %f\n",pose_act.x);
  pose_act.y= pose_prev.y + dist*sin(pose_prev.theta + deltaTheta/2);
  //Serial.printf("YAct %f\n",pose_act.y);
  pose_act.theta= pose_prev.theta + deltaTheta;
  if(pose_act.theta>=2*Pi){
    pose_act.theta=pose_act.theta-(2*Pi);
  }
  else if(pose_act.theta<=-2*Pi){
    pose_act.theta=pose_act.theta+(2*Pi);
  }
  //Serial.printf("ThetaAct %f\n",pose_act.theta);
```

```
  pose_prev = pose_act; //Actualizamos pose anterior

  return pose_act;
}


//------------------------------------------------------------------
//                    FUNCIONES DE MOVIMIENTO
//------------------------------------------------------------------
void arranque(){
  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  digitalWrite(STEPPER_R_STEPPIN, HIGH);
  digitalWrite(STEPPER_L_STEPPIN, HIGH);
  digitalWrite(STEPPER_R_STEPPIN, LOW);
  digitalWrite(STEPPER_L_STEPPIN, LOW);
  delayMicroseconds(STEP_SPEED);

}

//--------------------AVANZAR-------------------------
void avanzar(){
  Serial.println("Avanzar");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  for(int i = 0; i < NUMSTEPS; i++)
  {
    if (i % calib1_recto == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);

    } else if (i % calib2_recto == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    } else {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
```

```
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }
  steps_ML = NUMSTEPS;
  steps_MR = NUMSTEPS;
}

//--------------------RETROCEDER-----------------------------------
void retroceder(){
  Serial.println("Retroceder");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, LOW);
  digitalWrite(STEPPER_L_DIRPIN, LOW);

  for(int i = 0; i < NUMSTEPS; i++)
  {
    if (i % calib1_recto == 0) {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);

    } else if (i % calib2_recto == 0) {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    } else {
      digitalWrite(STEPPER_R_DIRPIN, LOW);

      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }

  steps_ML = -NUMSTEPS;
  steps_MR = -NUMSTEPS;
}

//--------------------GIRAR A LA DERECHA---------------------------
```

```
void giroDerecha(int giroRsteps){
  Serial.printf("Girar %d pasos hacia la derecha", giroRsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
  digitalWrite(STEPPER_L_DIRPIN, HIGH);//Avanza

  for(int steps = 0; steps<giroRsteps; steps++)
  {
    if (steps % calib_gd == 0) {
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);

    } else {
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }

  steps_ML=giroRsteps;
  steps_MR=-giroRsteps;
}


//-------------------GIRAR A LA IZQUIERDA-------------------------------
void giroIzquierda(int giroLsteps){
  Serial.printf("Girar %d pasos hacia la izquierda", giroLsteps);
  Serial.println("Girar hacia la izquierda");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
  digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede

  for(int steps = 0; steps<giroLsteps; steps++)
  {
    if (steps % calib_gi == 0) {
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);

    } else {
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
```

```
    }
  }

  steps_ML=-giroLsteps;
  steps_MR=giroLsteps;
}

//--------------------PARAR-----------------------------
void parar(){
  Serial.println("Parada");
  digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
  digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
  steps_ML = 0;
  steps_MR = 0;
}
```

C.0.5. Lidar.h

```
/*
 * Se emplea un sensor VL53l0X que rotará usando un motor paso a paso
 * Conexiones:
 *  NARANJA  XSHUT 23
 *  AMARILLO SCL   22
 *  VERDE    SDA   21
 *  El motor será un stepper 28BYJ-48:
 *                          Modo Half step: 0.0879; Modo Full step 0.176
 *                         Modo Half step: 4096; Modo Full step 2048
 * Ángulo de paso 5,625 y una reductura interna de 1:63.68395
 * Es decir, un paso total de 0,087 por lo que son 4076 pasos por revoluci↩
    ón (en half step mode)
 * Tras varias pruebas se ha determinado que se necesita para nuestro ↩
    LIDAR 8192 con steps
 * El montaje incorpora otra reductora mecánica, cuya relación de ↩
    transmisión de engranajes es 4:1 -> 4*2048=8192
 * Máxima velocidad 15rpm
*/

// Guarda para evitar inclusión duplicada
#ifndef LIDAR_H
#define LIDAR_H
//-------------------------------
//         LIBRERIAS
//-------------------------------
#include <Wire.h>
#include <Stepper.h>
#include <VL53L0X.h>

//-----------------------------------------------
//            DATOS Y DECLARACIONES
//-----------------------------------------------
//PINES
//Vl53l0x
#define XSHUT 23

//MotorLIDAR
```

```
#define LIDAR_STEPPER_IN1              16
#define LIDAR_STEPPER_IN2              4
#define LIDAR_STEPPER_IN3              2
#define LIDAR_STEPPER_IN4              15

//CONFIGURACIÓN STEPPER
#define MOTOR_STEPS_REV                2048    // Esto es el número de ↩
    pasos por revolución del stepper
#define LIDAR_STEPS_REV                8192    // Esto es el número de ↩
    pasos por revolución del LIDAR
#define LIDAR_STEPS                    256     // Esto es el número de ↩
    pasos por revolución del stepper
#define LIDAR_SPEED                    15      //Velocidad en revs por ↩
    minuto (RPM)

//CONFIGURACIÓN VL53L0X
//Modos de lectura laser
//#define LONG_RANGE
//define HIGH_SPEED
#define HIGH_ACCURACY

//Constantes
const double calib_laserDiff30=30; //Offset de calibración en la medida
const double calib_laserEsc=0.985; //Factor proporcional de calibración en↩
     la medida

const double distanciaMAX=95; //Distancia máxima de escaneo

struct datosLIDAR {
  double angulo;
  double distancia;
};

//Funciones
void setupLIDAR();
struct datosLIDAR controlLIDAR();
void retornoLIDAR();

#endif // Fin de guarda
```

C.0.6. Lidar.cpp

```
//-----------------------------
//          LIBRERIAS
//-----------------------------
#include "Lidar.h"

//-----------------------------------------
//            DATOS Y DECLARACIONES
//-----------------------------------------
//OBJETOS
Stepper LIDAR_STEPPER(MOTOR_STEPS_REV, LIDAR_STEPPER_IN4,LIDAR_STEPPER_IN2↩
     , LIDAR_STEPPER_IN3,LIDAR_STEPPER_IN1 ); //Modo FullStep secuencia ↩
    IN4-IN2-IN3-IN1 ANTIHORARIO para dar un paso
VL53L0X SENSOR_MEDICION;//Define nuestro sensor
```

```
//VARIABLES GLOBALES
int currentsteps=0;

//------------------------------------------------------------------------
//                          FUNCIONES
//------------------------------------------------------------------------
//-------------------------setupLIDAR-------------------------------------
void setupLIDAR() {

  LIDAR_STEPPER.setSpeed(LIDAR_SPEED); //Establecemos velocidad giro del ↩
     LIDAR

  Wire.begin();
  pinMode(XSHUT, OUTPUT);
  digitalWrite(XSHUT, HIGH);
  SENSOR_MEDICION.setTimeout(500);
  do{
    SENSOR_MEDICION.init();
    if (!SENSOR_MEDICION.init()){
      Serial.println("Fallo al detectar e inicializar el sensor VL53L0X");
    }
  }while(!SENSOR_MEDICION.init());

  SENSOR_MEDICION.setSignalRateLimit(0.55);
  //Modos de medición del laser
  #if defined LONG_RANGE
    // lower the return signal rate limit (default is 0.25 MCPS)
    SENSOR_MEDICION.setSignalRateLimit(0.1);
    // increase laser pulse periods (defaults are 14 and 10 PCLKs)
    SENSOR_MEDICION.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
    SENSOR_MEDICION.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, ↩
       14);
  #endif
  #if defined HIGH_SPEED
    // reduce timing budget to 20 ms (default is about 33 ms)
    SENSOR_MEDICION.setMeasurementTimingBudget(20000);
  #elif defined HIGH_ACCURACY
    // increase timing budget to 200 ms
    SENSOR_MEDICION.setMeasurementTimingBudget(50000);
  #endif

}

//-------------------------controlLIDAR-----------------------------------
struct datosLIDAR controlLIDAR() {

  datosLIDAR lecturaLIDAR;

  LIDAR_STEPPER.step(LIDAR_STEPS);
  currentsteps=currentsteps+LIDAR_STEPS;
  if(currentsteps==LIDAR_STEPS_REV){
    currentsteps=0;
    Serial.println("Vuelta completa");
  }
  lecturaLIDAR.angulo = 360*(double)currentsteps/LIDAR_STEPS_REV;

  double distancia = ((SENSOR_MEDICION.readRangeSingleMillimeters()-↩
```

```
      calib_laserDiff30)*calib_laserEsc)/10; //Dato lectura del sensor en ↩
      cm
  if(distancia>distanciaMAX){
    lecturaLIDAR.distancia=distanciaMAX;
  }else{
    lecturaLIDAR.distancia=distancia;
  }

  Serial.printf("Distancia: %fcm y angulo: %f grados\n",lecturaLIDAR.↩
      distancia,lecturaLIDAR.angulo);
  return lecturaLIDAR;
}

void retornoLIDAR(){

  if(currentsteps!=0){
    Serial.println("Moviendo LIDAR a posicion inicial");
    while(currentsteps<LIDAR_STEPS_REV){
      LIDAR_STEPPER.step(1);
      currentsteps=currentsteps+1;
      Serial.printf("Angulo actual: %f grados\n", 360*(double)currentsteps↩
          /LIDAR_STEPS_REV);
    }
    currentsteps = 0;
    Serial.println("Posición inicial LIDAR alcanzada");
  }
  else{
    Serial.println("Posición inicial LIDAR ya alcanzada");
  }

}
```

APÉNDICE D

Programación del mapa

```
# ------------------------------
#         LIBRERIAS
# ------------------------------
import paho.mqtt.client as mqtt  # Para usar MQTT
import json  # Para leer el JSON
import numpy as np  # Mapa
import math  # Cálculos
import io  # Para E/S
from matplotlib import pyplot as plt # Dibujar
from datetime import date
from PIL import Image  # Manejar imagenes


# MQTT-------------
MQTT_ADDRESS = '----------'
MQTT_USER = '-------'
MQTT_PASSWORD = '-------'


clientID = "BuilderPI"

# Topics-----------
tpub_mapa = 'builder/mapa'
tsub_genMapa = 'control/genMapa'
tsub_medidaObs = 'robot/medidaObs'

# Posiciones iniciales en el mapa
genObsmapa = "Activar"

class Mapa():

    def __init__(self,tam,tamCelda):
        self.logOdd_map = np.zeros((tam,tam))
        self.centro=int(tam/2)
        self.tamCelda=tamCelda

        self.FOV = 15.2*np.pi/180 # Campo visión laser vl53l0x
        self.maxDist = 95.0
        self.distLaser=2.4
        self.distCentroLidar=1.7
```

```
        self.posLidar=[self.centro, self.centro]
        self.posObjeto = [[self.centro, self.centro],[self.centro, self.↩
            centro]]


        self.logOdd_libre = np.log(0.35/0.65)
        self.logOdd_Ocupado = np.log(0.65/0.35)

    def GenerarPosiciones(self, medida, genPosObj):

        # Se añade el offset del mapa
        posXLidar = self.centro-math.floor(medida["x"]/self.tamCelda)
        posYLidar = self.centro-math.floor(medida["y"]/self.tamCelda)
        self.posLidar = [posXLidar, posYLidar]
        print("posLidar en el mapa: ", self.posLidar)

        if(genPosObj):

            # Calculamos la posición del objeto en cm, recibo el dato en ↩
                cm
            Coef1AngMax=self.distCentroLidar+(medida["d_obs"]+self.↩
                distLaser)*math.cos(medida["alfa_obs"]+(self.FOV/2))
            Coef1AngMin=self.distCentroLidar+(medida["d_obs"]+self.↩
                distLaser)*math.cos(medida["alfa_obs"]-(self.FOV/2))

            Coef2AngMax=(medida["d_obs"]+self.distLaser)*math.sin(medida["↩
                alfa_obs"]+(self.FOV/2))
            Coef2AngMin=(medida["d_obs"]+self.distLaser)*math.sin(medida["↩
                alfa_obs"]-(self.FOV/2))

            posXObjeto_L = posXLidar-math.floor((math.cos(medida["theta"])↩
                *Coef1AngMax-math.sin(medida["theta"])*Coef2AngMax)/self.↩
                tamCelda)
            posXObjeto_R = posXLidar-math.floor((math.cos(medida["theta"])↩
                *Coef1AngMin-math.sin(medida["theta"])*Coef2AngMin)/self.↩
                tamCelda)

            posYObjeto_L = posYLidar-math.floor((math.sin(medida["theta"])↩
                *Coef1AngMax+math.cos(medida["theta"])*Coef2AngMax)/self.↩
                tamCelda)
            posYObjeto_R = posYLidar-math.floor((math.sin(medida["theta"])↩
                *Coef1AngMin+math.cos(medida["theta"])*Coef2AngMin)/self.↩
                tamCelda)
            self.posObjeto = [[posXObjeto_L, posYObjeto_L], [posXObjeto_R,↩
                 posYObjeto_R]]

            print("posObjeto en el mapa: ", self.posObjeto)


    def ActualizarMapa(self, distancia):
        #Algoritmo de Bresenham

        for haz in self.posObjeto:
            if (haz[0] > self.logOdd_map.shape[0] or haz[1] > self.↩
                logOdd_map.shape[1] and haz[0]>0 and haz[1]>0):
                continue
```

```
# Extremo inicial del algoritmo
x1, y1 = self.posLidar
x2, y2 = haz
dx = x2-x1
dy = y2-y1

# Estudiamos inclinacion
inclinacion = (abs(dy) > abs(dx))
if inclinacion:
    x1, y1 = y1, x1
    x2, y2 = y2, x2
    # print("inclinacion")

# Intercambio start-end
intercambio = False
if(x1 > x2):
    x1, x2 = x2, x1
    y1, y2 = y2, y1
    intercambio = True
    # print("intercambio")

# Recalculamos distancias
dx = x2-x1
dy = y2-y1

# Error
error = int(dx/2.0)
if(y2 > y1):
    IncY = 1
else:
    IncY = -1

# Trazado de lineas
y = y1
puntos = []
for x in range(x1, x2+1):
    if(inclinacion):
        pto = [y, x]
    else:
        pto = [x, y]

    puntos.append(pto)
    # Coordenadas en el mapa de la linea desde la pos del robot a obs
    error -= abs(dy)
    if error < 0:
        y += IncY
        error += dx

# Coordenadas en el mapa de la linea desde la pos del robot a obs
if intercambio:
    puntos.reverse()

# 2-Actualizar mapa
for coord in puntos:
    if(distancia == self.maxDist):
        # No hay obstáculo
```

```
                    self.logOdd_map[coord[0], coord[1]] += self.↩
                        logOdd_libre
                else:
                    # Hay obstáculo, compruebo si es la celda del ↩
                        obstaculo
                    if(coord == haz): #1 ,0
                        self.logOdd_map[coord[0], coord[1]] += self.↩
                            logOdd_Ocupado
                    else:
                        # Celda del camino al obstaculo
                        self.logOdd_map[coord[0], coord[1]] += self.↩
                            logOdd_libre

            # Almacenamos copia de la imagen
            plt.imsave('Mapa'+str(date.today())+'.png', 1.0 - 1./(1.+np.↩
                exp(self.logOdd_map)), cmap='Greys', vmin=0, vmax=1)


    def PublicarMapa(self,client,topic):
        # Leemos a imagen
        img = Image.open('Mapa'+str(date.today())+'.png')
        # Añadimos la posicion del robot (RGB rojo)
        img.putpixel((self.posLidar[1],self.posLidar[0]), (255, 0, 0))
        # BytesIO es un archivo "falso" guardado en memoria
        imgByteArr = io.BytesIO()
        # Se almacena la imagen en este espacio de memoria
        img.save(imgByteArr, format='PNG')
        # Se convierte la imagen en byte array
        byteArr = imgByteArr.getvalue()
        # Publicar
        result = client.publish(topic, byteArr)
        status = result[0]
        if status == 0:
            print("Publicado")
        else:
            print("Fallo al enviar le mensaje al topic")


def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print('Conectado al broker')
        client.subscribe(tsub_genMapa)
        client.subscribe(tsub_medidaObs)
        print("Suscrito al topic: ", tsub_genMapa, " y ", tsub_medidaObs)
    else:
        print("Conexión fallida")


def on_message(client, userdata, msg):
    global genObsmapa  # Se modifica la variable global

    # Evitamos posibles problemas obteniendo los datos recibidos por MQTT
    try:
        if(msg.topic == tsub_genMapa):
            genObsmapa = str(msg.payload.decode("utf-8"))
            print("  Generar   obs en el mapa?: %s" % genObsmapa)

        elif(msg.topic == tsub_medidaObs):
```

```
            listaObs = json.loads(msg.payload)
            print(listaObs)

            if(genObsmapa == "Activar"):
                mapa.GenerarPosiciones(listaObs, True)
                # Calculamos la pos del LIDAR=robot y del obstáculo leido
                if(mapa.posLidar[0] <= mapa.logOdd_map.shape[0] and mapa.↩
                    posLidar[1] <= mapa.logOdd_map.shape[1]):
                    mapa.ActualizarMapa(listaObs["d_obs"])  # Y cremos el ↩
                        mapa
                    mapa.PublicarMapa(client,tpub_mapa)  # Creamos imagen ↩
                        y publicamos
            else:
                mapa.GenerarPosiciones(listaObs, False)
                # Calculamos la pos solo del LIDAR=robot
                mapa.PublicarMapa(client,tpub_mapa)  # Creamos imagen y ↩
                    publicamos

    except:
        print("No JSON returned")

if __name__ == '__main__':
    mapa=Mapa(int(30),int(30))

    mqtt_client = mqtt.Client(clientID)
    mqtt_client.username_pw_set(MQTT_USER, MQTT_PASSWORD)  # Establecer ↩
        contraseña
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message
    mqtt_client.connect(MQTT_ADDRESS, 1883)

    mqtt_client.loop_forever()
```

APÉNDICE E

Código de la aplicación

E.0.1. Código de la interfaz

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="#014B8C">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="130dp"
            android:layout_height="50dp"
            android:layout_gravity="center"
            android:layout_marginLeft="20dp"
            android:foreground="@drawable/logouma" />

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="120dp"
            android:layout_height="50dp"
            android:layout_gravity="center"
            android:layout_marginLeft="100dp"
            android:foreground="@drawable/logoeii" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
```

```
            android:id="@+id/txtModo"
            android:layout_width="117dp"
            android:layout_height="wrap_content"
            android:text="@string/ModoActual"
            android:textColor="@color/black"
            android:textSize="17sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/txtModoResult"
            android:layout_width="101dp"
            android:layout_height="wrap_content"
            android:text="@string/CANCELAR"
            android:textColor="@color/black"
            android:textSize="17sp"
            android:textStyle="bold"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
        <TextView
            android:id="@+id/txtDibujar"
            android:layout_width="76dp"
            android:layout_height="wrap_content"
            android:text="@string/GenMapa"
            android:textColor="@color/black"
            android:textSize="17sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/txtDibujarResult"
            android:layout_width="38dp"
            android:layout_height="wrap_content"
            android:text="ON"
            android:textColor="@color/black"
            android:textSize="17sp"
            android:textStyle="bold"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">


        <TextView
            android:id="@+id/txtPosX"
```

```
        android:layout_width="23dp"
        android:layout_height="22sp"
        android:gravity="center"
        android:text="@string/PosX"
        android:textColor="@color/black"
        android:textSize="17sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txtPosXResult"
        android:layout_width="76dp"
        android:layout_height="wrap_content"
        android:text="0.00m"
        android:textColor="@color/black"
        android:textSize="17sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txtPosY"
        android:layout_width="24dp"
        android:layout_height="22sp"
        android:gravity="center"
        android:text="@string/PosY"
        android:textColor="@color/black"
        android:textSize="17sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txtPosYResult"
        android:layout_width="67dp"
        android:layout_height="wrap_content"
        android:text="0.00m"
        android:textColor="@color/black"
        android:textSize="17sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txtOri"
        android:layout_width="22dp"
        android:layout_height="22sp"
        android:gravity="center"
        android:text="@string/Ori"
        android:textColor="@color/black"
        android:textSize="17sp"
```

```
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/txtOriResult"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="0.00   "
            android:textColor="@color/black"
            android:textSize="17sp"
            android:textStyle="bold"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />


    </LinearLayout>

    <ImageView
        android:id="@+id/imageViewMapa"
        android:layout_width="220dp"
        android:layout_height="220dp"
        android:layout_gravity="center"
        android:layout_margin="10dp"
        android:foreground="@mipmap/ic_launcher" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="18dp"
        android:gravity="center"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:background="#014B8C"
            android:gravity="center"
            android:text="@string/EXPLORAR"
            android:textColor="@color/white"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:background="#014B8C"
            android:gravity="center"
            android:text="@string/NAVEGAR"
```

```
            android:textColor="@color/white"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:background="@color/purple_700"
            android:gravity="center"
            android:text="@string/ESCANEAR"
            android:textColor="@color/white"
            android:textStyle="bold" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:gravity="center"
        android:orientation="horizontal">

        <ToggleButton
            android:id="@+id/tgBtnExp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="@string/EXPLORAR"
            android:textIsSelectable="false"
            android:textOff="@string/DESACTIVADO"
            android:textOn="@string/ACTIVADO"
            app:rippleColor="@color/black" />

        <ToggleButton
            android:id="@+id/tgBtnNav"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:text="@string/NAVEGAR"
            android:textOff="@string/DESACTIVADO"
            android:textOn="@string/ACTIVADO"
            app:rippleColor="@color/black" />

        <ToggleButton
            android:id="@+id/tgBtnScan"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:button="@android:drawable/ic_menu_mapmode"
            android:buttonTint="@color/black"
            android:checked="true"
            android:text="@string/ESCANEAR"
            android:textOff="@string/DESACTIVADO"
```

```
            android:textOn="@string/ACTIVADO"
            app:rippleColor="@color/black" />

    </LinearLayout>

    <Button
        android:id="@+id/btnCan"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:backgroundTint="@color/purple_500"
        android:foregroundGravity="center_vertical|center|↩
            center_horizontal"
        android:text="@string/CANCELAR"
        android:textColor="@color/white"
        android:textSize="15sp"
        android:textStyle="bold"
        app:rippleColor="@color/black" />

    <ToggleButton
        android:id="@+id/btnA"
        android:layout_width="70dp"
        android:layout_height="60dp"
        android:layout_gravity="center"
        android:layout_marginTop="5dp"
        android:foreground="@drawable/flechaavanzar"
        android:foregroundTint="#ffff0000"
        android:foregroundTintMode="add"
        app:rippleColor="@color/black" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_marginTop="10dp"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btnI"
            android:layout_width="20dp"
            android:layout_height="match_parent"
            android:layout_marginLeft="70dp"
            android:layout_weight="1"
            android:foreground="@drawable/flechaizq"
            android:foregroundTint="#FF0000"
            android:foregroundTintMode="add"
            android:gravity="center"
            app:rippleColor="@color/black" />

        <ToggleButton
            android:id="@+id/btnP"
            android:layout_width="30dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:layout_marginLeft="20dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
```

```
            android:backgroundTint="@color/purple_500"
            android:foreground="@drawable/ic_toggle"
            android:gravity="center"
            android:textOff="@null"
            android:textOn="@null"
            app:rippleColor="@color/black" />

        <Button
            android:id="@+id/btnD"
            android:layout_width="20dp"
            android:layout_height="match_parent"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="70dp"
            android:layout_weight="1"
            android:foreground="@drawable/flechaderecha"
            android:foregroundTint="#FF0000"
            android:foregroundTintMode="add"
            android:gravity="center"
            app:rippleColor="@color/black" />

    </LinearLayout>

    <ToggleButton
        android:id="@+id/btnR"
        android:layout_width="70dp"
        android:layout_height="60dp"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:foreground="@drawable/flechaatras"
        android:foregroundTint="#FF0000"
        android:foregroundTintMode="add"
        app:rippleColor="@color/black" />

</LinearLayout>
```

E.0.2. Código de la actividad

```
package com.example.robotfg;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.content.pm.ActivityInfo;
import android.content.res.ColorStateList;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.media.Image;
import android.os.Build;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
```

```
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.*;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.text.DecimalFormat;
import java.util.logging.Logger;

public class MainActivity extends AppCompatActivity {

    ToggleButton Modo_Exp, Modo_Nav, Modo_Scan;
    Button Cancelar,Derecha,Izquierda;
    ToggleButton Avanzar, Pausa, Retroceder;
    TextView ModoAct, CrearMapaResult, XResult, YResult, OriResult;

    //MQTT
    public MqttAndroidClient client;
    public String mqtt_server="tcp://------";
    public String mqtt_username="----";
    public String mqtt_password="-----";

    //Topics
    public String tPub_accion="control/accion";
    public String tPub_representar="control/genMapa";

    public String tSub_mapa="builder/mapa";
    public String tSub_medidas="robot/medidaObs";


    @RequiresApi(api = Build.VERSION_CODES.M)
    @SuppressLint("ClickableViewAccessibility")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        connectaMQTT();

        //Elementos de pantalla
```

```
        Modo_Exp= findViewById(R.id.tgBtnExp);
        Modo_Nav= findViewById(R.id.tgBtnNav);
        Modo_Scan= findViewById(R.id.tgBtnScan);

        Cancelar= findViewById(R.id.btnCan);

        Avanzar= findViewById(R.id.btnA);
        Retroceder= findViewById(R.id.btnR);
        Derecha= findViewById(R.id.btnD);
        Izquierda= findViewById(R.id.btnI);

        Pausa= findViewById(R.id.btnP);

        ModoAct=findViewById(R.id.txtModoResult);
        CrearMapaResult=findViewById(R.id.txtDibujarResult);
        XResult=findViewById(R.id.txtPosXResult);
        YResult=findViewById(R.id.txtPosYResult);
        OriResult=findViewById(R.id.txtOriResult);

        Avanzar.setClickable(false);
        Retroceder.setClickable(false);
        Izquierda.setClickable(false);
        Derecha.setClickable(false);
        Pausa.setClickable(false);

        //CANCELAR----------------------
        Cancelar.setOnClickListener(view -> {

            Modo_Nav.setChecked(false);
            Modo_Exp.setChecked(false);
            Modo_Scan.setChecked(false);

            ModoAct.setText(getString(R.string.CANCELAR));
            CrearMapaResult.setText(getString(R.string.DESACTIVADO));

            Avanzar.setClickable(false);
            Retroceder.setClickable(false);
            Izquierda.setClickable(false);
            Derecha.setClickable(false);
            Pausa.setClickable(false);

            Pausa.setVisibility(View.VISIBLE);
            Avanzar.setVisibility(View.VISIBLE);
            Retroceder.setVisibility(View.VISIBLE);
            Izquierda.setVisibility(View.VISIBLE);
            Derecha.setVisibility(View.VISIBLE);

            Avanzar.setChecked(false);
            Retroceder.setChecked(false);
            Avanzar.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));
            Retroceder.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));

            publicaMQTT(tPub_accion,"C");
        });

        //MODO SCAN (Representar Mapa)----------------
```

```
        Modo_Scan.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if (isChecked) {
                publicaMQTT(tPub_representar,"Activar");
                //Indicamos el modo actual
                CrearMapaResult.setText(getString(R.string.ACTIVADO));

            }else{
                publicaMQTT(tPub_representar,"Desactivar");
                //Indicamos el modo actual
                CrearMapaResult.setText(getString(R.string.DESACTIVADO));
            }
        });

        //MODO EXPLORAR----------------
        Modo_Exp.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if (isChecked) {

                //Desactivamos modos e indicamos el modo actual
                Modo_Nav.setChecked(false);
                ModoAct.setText(getString(R.string.EXPLORAR));

                //Habilitamos pausa y deshabilitamos botonera
                Pausa.setVisibility(View.VISIBLE);
                Avanzar.setVisibility(View.INVISIBLE);
                Retroceder.setVisibility(View.INVISIBLE);
                Izquierda.setVisibility(View.INVISIBLE);
                Derecha.setVisibility(View.INVISIBLE);
                Avanzar.setChecked(false);
                Retroceder.setChecked(false);
                Avanzar.setForegroundTintList(ColorStateList.valueOf(0↩
                    xffff0000));
                Retroceder.setForegroundTintList(ColorStateList.valueOf(0↩
                    xffff0000));
                Pausa.setClickable(true);

            }else{
                Pausa.setChecked(false);
                Pausa.setClickable(false);
            }
        });

        //MODO NAVEGAR----------------
        Modo_Nav.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if (isChecked) {

                //Desactivamos modos e indicamos el modo actual
                Modo_Exp.setChecked(false);
                ModoAct.setText(getString(R.string.NAVEGAR));
                Avanzar.setClickable(true);
                Retroceder.setClickable(true);
                Izquierda.setClickable(true);
                Derecha.setClickable(true);

                //Habilitamos botones y  deshabilitamos pausa
                Pausa.setVisibility(View.INVISIBLE);
                Avanzar.setVisibility(View.VISIBLE);
                Retroceder.setVisibility(View.VISIBLE);
                Izquierda.setVisibility(View.VISIBLE);
```

```
            Derecha.setVisibility(View.VISIBLE);
        }else{
            Pausa.setChecked(false);
            Avanzar.setChecked(false);
            Retroceder.setChecked(false);
            Avanzar.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));
            Retroceder.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));
            Avanzar.setClickable(false);
            Retroceder.setClickable(false);
            Izquierda.setClickable(false);
            Derecha.setClickable(false);
        }
    });

    //BOTONERA----------------
    Avanzar.setOnCheckedChangeListener((buttonView, isChecked) -> {
        if (isChecked) {
            Retroceder.setChecked(false);
            Avanzar.setForegroundTintList(ColorStateList.valueOf(0↩
                xFFDABDBD));
            publicaMQTT(tPub_accion,"A");
        }else{
            if(!Retroceder.isChecked()){
                publicaMQTT(tPub_accion,"P");
            }
            Avanzar.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));
        }
    });

    Retroceder.setOnCheckedChangeListener((buttonView, isChecked) -> {
        if (isChecked) {
            Avanzar.setChecked(false);
            Retroceder.setForegroundTintList(ColorStateList.valueOf(0↩
                xFFDABDBD));
            publicaMQTT(tPub_accion,"R");

        }else{
            if(!Avanzar.isChecked()){
                publicaMQTT(tPub_accion,"P");
            }
            Retroceder.setForegroundTintList(ColorStateList.valueOf(0↩
                xffff0000));
        }
    });

    Izquierda.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if(Modo_Nav.isChecked()) {
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    publicaMQTT(tPub_accion, "I");
                    Avanzar.setChecked(false);
                    Retroceder.setChecked(false);
                    Izquierda.setForegroundTintList(ColorStateList.↩
                        valueOf(0xFFDABDBD));
```

```
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    publicaMQTT(tPub_accion, "P");//se deja de ↩
                        presionar
                    Izquierda.setForegroundTintList(ColorStateList.↩
                        valueOf(0xffff0000));
                }
            }
            return true;
        }
    });
    Derecha.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if(Modo_Nav.isChecked()) {
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    publicaMQTT(tPub_accion, "D");
                    Avanzar.setChecked(false);
                    Retroceder.setChecked(false);
                    Derecha.setForegroundTintList(ColorStateList.↩
                        valueOf(0xFFDABDBD));
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    publicaMQTT(tPub_accion, "P");//se deja de ↩
                        presionar
                    Derecha.setForegroundTintList(ColorStateList.↩
                        valueOf(0xffff0000));
                }
            }
            return true;
        }
    });

    Pausa.setOnCheckedChangeListener((buttonView, isChecked) -> {
        if (isChecked) {
            if(Modo_Exp.isChecked()){
                //Envía explorar
                publicaMQTT(tPub_accion,"E");
            }
        }else{
            //MANDAMOS PAUSA
            publicaMQTT(tPub_accion, "P");
        }
    });

    callbackMQTT();
}

//-----------------------MQTT-----------------------------
//CONECTAR
private void connectaMQTT(){
    String clientId = MqttClient.generateClientId();
    client = new MqttAndroidClient(this.getApplicationContext(),↩
        mqtt_server,clientId);
    MqttConnectOptions options = new MqttConnectOptions();
    options.setUserName(mqtt_username);
    options.setPassword(mqtt_password.toCharArray());
    options.setAutomaticReconnect(true);
```

```
        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Toast.makeText(MainActivity.this,"Conectado a MQTT", ↩
                        Toast.LENGTH_SHORT).show();
                    suscripcionMQTT();
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken, ↩
                    Throwable exception) {
                    Toast.makeText(MainActivity.this,"No ha sido posible ↩
                        conectarse a MQTT", Toast.LENGTH_LONG).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    //CALLBACK
    private void callbackMQTT(){
        //Declarar callback MQTT
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
                Toast.makeText(MainActivity.this,"Conexión al servidor ↩
                    MQTT perdida. Reconectando...);", Toast.LENGTH_SHORT).↩
                    show();
                // Reconexión automatica.
            }
            @SuppressLint({"DefaultLocale", "SetTextI18n"})
            @RequiresApi(api = Build.VERSION_CODES.M)
            @Override
            public void messageArrived(String topic, MqttMessage message) ↩
                throws JSONException {
                if(topic.equals(tSub_mapa)){
                    ImageView Mapa = findViewById(R.id.imageViewMapa);
                    byte[] mapabyte = message.getPayload();
                    Drawable imagemapa = new BitmapDrawable(getResources()↩
                        ,BitmapFactory.decodeByteArray(mapabyte,0,mapabyte↩
                        .length));
                    Mapa.setForeground(imagemapa);
                }
                else if(topic.equals(tSub_medidas)){
                    try{
                        JSONObject jsonPose = new JSONObject(new String(↩
                            message.getPayload()));
                        double x = Double.parseDouble(jsonPose.getString("↩
                            x"))/100;
                        double y= Double.parseDouble(jsonPose.getString("y↩
                            "))/100;
                        double theta = Math.toDegrees(Double.parseDouble(↩
                            jsonPose.getString("theta")));
                        XResult.setText(String.format ("%.2f", x) + "m");
                        YResult.setText(String.format ("%.2f", y) + "m");
```

```
                        OriResult.setText(String.format ("%.2f", theta) + "  ");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });
    }
    //SUSCRIPCION
    private void suscripcionMQTT(){
        try{
            client.subscribe(tSub_mapa,0);
            client.subscribe(tSub_medidas,0);
            Toast.makeText(MainActivity.this,"Subscrito", Toast.LENGTH_SHORT).show();
        }catch (MqttException e){
            e.printStackTrace();
        }
    }

    //PUBLICACION
    private void publicaMQTT(String topic, String mensaje){
        try {
            if(client.isConnected()){
                client.publish(topic, mensaje.getBytes(),1,false);
                Toast.makeText(this,"Publicando: " + mensaje + ", on topic: " +topic,Toast.LENGTH_SHORT).show();
            }
        } catch ( MqttException e) {
            e.printStackTrace();
        }
    }
}
```

APÉNDICE F

Calibración del movimiento

F.0.1. Código de la cabecera (Movimiento.h)

```
/*
 * Se hace uso de 2 stepper NEMA 17HS16-2004S con controlador Pololu A4988
 * Ambos potenciometros son de 1.24V
 * VMotor no debe caer por debajo de 10V
 * Imax del motor es 2A -> Vref= Imax*8*Ra4988=2.8.0.1=1.44
 * Usamos entre un 80% y 90% por seguridad -> en nuestro caso 86% = 1.22
 * Nema17 cuenta con 2 fases: 1.1 Ohmios/fase, 2V/fase y 2.2 A/fase
 * Se hace uso de un modulo controlador de steppers para el a4988
 * El driver posee 3 selectores para la resolución del paso:
 *MS1---MS2---MS3---Microstep Resolution---
 *Low   Low   Low   Full step
 *High  Low   Low   Half step
 *Low   High  Low   Quarter step
 *High  High  Low   Eighth step
 *High  High  High  Sixteenth step
 *-----------------------------------------
 * Una vuelta entera de rueda son 3200 pasos que provoca un giro aprox de ↩
    51.5820895 grados
*/

// Guarda para evitar inclusión duplicada
#ifndef MOVIMIENTO_H
#define MOVIMIENTO_H

//-------------------------------
//         LIBRERIAS
//-------------------------------
#include <Arduino.h>

//-----------------------------------------
//           DATOS Y DECLARACIONES
//-----------------------------------------
//CONFIGURACIÓN MOVIMIENTO
#define STEPS_X_REV                3200 //Esto es el número de pasos por ↩
    revolución - MODO Eighth step
#define STEP_SPEED                 2000 //Velocidad us/paso, se crea señanl↩
     pwm
```

```
//PINES
//MotorRIGHT
#define STEPPER_R_DIRPIN          27
#define STEPPER_R_STEPPIN         26
#define STEPPER_R_ENABLEPIN       25 //LOW enciende el motor, HIGH lo ↩
    apaga

//MotorLEFT
#define STEPPER_L_DIRPIN          5
#define STEPPER_L_STEPPIN         18
#define STEPPER_L_ENABLEPIN       19

//CALIBRACION
//#define ARRANQUE
//#define RECTO
//#define GIRO_1
#define GIRO_5

const double r=0.0336;
const double dr=0.2255;
const double Pi = 3.14159265358979323846264338327 95;
const int vueltas = 5;

//FUNCIONES
void rotaD();
void rotaI();
void avanzar();
void retroceder();
void giroDerecha(int giroRsteps);
void giroIzquierda(int giroLsteps);
void arranque();
#endif // Fin de guarda
```

F.0.2. Código fuente

```
//-----------------------------
//          LIBRERIAS
//-----------------------------
#include "Movimiento.h"

//-------------------------------------------
//            DATOS Y DECLARACIONES
//-------------------------------------------
bool derecha=true ;

//-----------------------------------------------------------------------
//                            FUNCIONES
//-----------------------------------------------------------------------

//-----------------------------setupMotores-------------------------------
void setup() {

  pinMode(STEPPER_R_DIRPIN, OUTPUT);
  pinMode(STEPPER_R_STEPPIN, OUTPUT);
  pinMode(STEPPER_R_ENABLEPIN, OUTPUT);
```

```
  pinMode(STEPPER_L_DIRPIN, OUTPUT);
  pinMode(STEPPER_L_STEPPIN, OUTPUT);
  pinMode(STEPPER_L_ENABLEPIN, OUTPUT);

#if defined ARRANQUE
  avanzar(800); //Se prueba a mover el robot un pequeño desplazamiento y ↩
      estudiar el efecto del arranque
#elif defined GIRO_1
  arranque();
  if (derecha) {
      for (int i = 1; i <= 33; i++){giroDerecha(337);}//Número de pasos ↩
         para girar en modo navegar (equivale a 11.25134328 grados)
    }else{
      for (int i = 1; i <= 33; i++){giroIzquierda(337);}//Número de pasos ↩
         para girar en modo navegar (equivale a 11.25134328 grados)
    }
    digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
    digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apag#do
#elif defined GIRO_5
  arranque();
  for (int i = 0; i <= vueltas; i++){
    if (derecha) {
      for (int i = 1; i <= 33; i++){giroDerecha(337);}//Número de pasos ↩
         para girar en modo navegar (equivale a 11.25134328 grados)
    }else{
      for (int i = 1; i <= 33; i++){giroIzquierda(337);}//Número de pasos ↩
         para girar en modo navegar (equivale a 11.25134328 grados)
    }
    digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
    digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
  }
#elif defined RECTO
   arranque();
   double metros = 1.015; //Hay que añadir un 1.5% de distancia respecto a↩
       lo que quieres andar
   int pasos_recto = round((metros * 2 * STEPS_X_REV) / (2 * 2 * Pi * r));
   avanzar(pasos_recto);
#endif

}

void loop() {
//NO APLICA
}

//-------------------------------------------------------------------
//                    FUNCIONES DE MOVIMIENTO
//-------------------------------------------------------------------
void arranque(){
  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  digitalWrite(STEPPER_R_STEPPIN, HIGH);
  digitalWrite(STEPPER_L_STEPPIN, HIGH);
  digitalWrite(STEPPER_R_STEPPIN, LOW);
```

```
  digitalWrite(STEPPER_L_STEPPIN, LOW);
  delayMicroseconds(STEP_SPEED);

}

//--------------------AVANZAR-----------------------------
void avanzar(int pasos) {
  Serial.println("Avanzar");

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH);
  digitalWrite(STEPPER_L_DIRPIN, HIGH);

  for (int i = 0; i < pasos; i++)
  {

    if (i % 500 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    } else if (i % 1010 == 0) {
      digitalWrite(STEPPER_R_DIRPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    } else {
      digitalWrite(STEPPER_R_DIRPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }

  digitalWrite(STEPPER_R_ENABLEPIN, HIGH);  //Motor apagado
  digitalWrite(STEPPER_L_ENABLEPIN, HIGH);  //Motor apagado
}


//--------------------GIRAR A LA DERECHA-----------------------------
void giroDerecha(int giroRsteps) {
  Serial.printf("Girar %d pasos hacia la derecha", giroRsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, LOW); //Retrocede
```

```
  digitalWrite(STEPPER_L_DIRPIN, HIGH);//Avanza

  for (int steps = 0; steps < giroRsteps; steps++)
  {
    if (steps % 56 == 0) {
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);

    } else {
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }
}


//--------------------GIRAR A LA IZQUIERDA-----------------------------
void giroIzquierda(int giroLsteps) {
  Serial.printf("Girar %d pasos hacia la izquierda", giroLsteps);

  digitalWrite(STEPPER_R_ENABLEPIN, LOW);  //Motor encendido
  digitalWrite(STEPPER_L_ENABLEPIN, LOW);  //Motor encendido

  digitalWrite(STEPPER_R_DIRPIN, HIGH); //Avanza
  digitalWrite(STEPPER_L_DIRPIN, LOW); //Retrocede

  for (int steps = 0; steps < giroLsteps; steps++)
  {
    if (steps % 67 == 0) {
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);

    } else {
      digitalWrite(STEPPER_R_STEPPIN, HIGH);
      digitalWrite(STEPPER_L_STEPPIN, HIGH);
      delayMicroseconds(STEP_SPEED);
      digitalWrite(STEPPER_R_STEPPIN, LOW);
      digitalWrite(STEPPER_L_STEPPIN, LOW);
      delayMicroseconds(STEP_SPEED);
    }
  }
}
```

APÉNDICE G

Calibración de la lectura del láser

```
#include <Wire.h>
#include <VL53L0X.h>

#define XSHUT 23

VL53L0X sensor;

//#define LONG_RANGE
//#define HIGH_SPEED
#define HIGH_ACCURACY

double minima;
double maxima;
double medida_acumulada;
int counter = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println("----------Iniciando----------");
  Wire.begin();
  pinMode(XSHUT, OUTPUT);
  digitalWrite(XSHUT, HIGH);
  sensor.init();

  sensor.setTimeout(500);
  if (!sensor.init())
  {
    Serial.println("Failed to detect and initialize sensor!");
    while (1) {}
  }
  //set the return signal rate limit (default is 0.25 MCPS)
  sensor.setSignalRateLimit(0.55);//055
  // set laser pulse periods (defaults are 14 and 10 PCLKs)
  //sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
  //sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);

#if defined LONG_RANGE
  // lower the return signal rate limit (default is 0.25 MCPS)
  sensor.setSignalRateLimit(0.1);
```

```
  // increase laser pulse periods (defaults are 14 and 10 PCLKs)
  sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
  sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
#endif

#if defined HIGH_SPEED
  // reduce timing budget to 20 ms (default is about 33 ms)
  sensor.setMeasurementTimingBudget(20000);
#elif defined HIGH_ACCURACY
  //  sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 17);
  //  sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 13);
  sensor.setMeasurementTimingBudget(50000);//200000
#endif
  //sensor.startContinuous();
  maxima = 0;
  minima = sensor.readRangeSingleMillimeters();
  medida_acumulada = 0;

  Serial.println("Fin setup");
}

void loop()
{
  double medida = (sensor.readRangeSingleMillimeters() - 30) * 0.985;

  //MEDIDA
  Serial.print(medida);
  Serial.print(",");
  counter++;

  //MÁXIMO
  if (medida > maxima) {
    maxima = medida;
  }
  Serial.print(maxima);
  Serial.print(",");

  //MEDIA
  medida_acumulada = medida_acumulada + medida;
  double media = medida_acumulada / counter;
  Serial.print(media);
  Serial.print(",");

  //MÍNIMO
  if (medida < minima) {
    minima = medida;
  }
  Serial.println(minima);

  delay(10);
}
```

APÉNDICE H

Cálculo posición del objeto

H.0.1. Cálculo posiciones del objeto

Para este calculo no se tendra en cuenta el campo de visión del láser, pero éste sería un incremento o decremento del valor de alfa. Se desarrolla la composición de transformadas homogéneas, que para este proyecto solo se utilizarán el desplazamiento y rotación en el eje Z.

```
syms theta distRobot distLidar alfa distLaser distObj;

%Robot repecto de origen
wTr=rotZ(theta)*desp([distRobot;0;0])
```

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & \text{distRobot}\cos(\theta) \\ \sin(\theta) & \cos(\theta) & 0 & \text{distRobot}\sin(\theta) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%CentroLidar respecto del robot
rT_lidar=desp([distLidar;0;0])
```

$$\begin{pmatrix} 1 & 0 & 0 & \text{distLidar} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%Laser respecto a Centro del Lidar
Lidar_T_Laser=rotZ(alfa)*desp([distLaser;0;0])
```

$$\begin{pmatrix} \cos(\text{alfa}) & -\sin(\text{alfa}) & 0 & \text{distLaser}\cos(\text{alfa}) \\ \sin(\text{alfa}) & \cos(\text{alfa}) & 0 & \text{distLaser}\sin(\text{alfa}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%Obj respecto al laser
Laser_T_Obj=desp([distObj;0;0])
```

$$\begin{pmatrix} 1 & 0 & 0 & \text{distObj} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%Objeto respeto del Robot
rT_Objeto=rT_lidar*Lidar_T_Laser*Laser_T_Obj
```

$$\begin{pmatrix} \cos(\text{alfa}) & -\sin(\text{alfa}) & 0 & \text{distLidar} + \text{distObj}\cos(\text{alfa}) + \text{distLaser}\cos(\text{alfa}) \\ \sin(\text{alfa}) & \cos(\text{alfa}) & 0 & \text{distObj}\sin(\text{alfa}) + \text{distLaser}\sin(\text{alfa}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%Objeto respecto al origen
wT_Objeto=wTr*rT_Objeto
```

$$\begin{pmatrix} \sigma_1 & -\sigma_5 - \sigma_4 & 0 & \cos(\theta)\ \sigma_2 + \text{distRobot}\cos(\theta) - \sin(\theta)\ \sigma_3 \\ \sigma_5 + \sigma_4 & \sigma_1 & 0 & \sin(\theta)\ \sigma_2 + \text{distRobot}\sin(\theta) + \cos(\theta)\ \sigma_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$$\sigma_1 = \cos(\text{alfa})\ \cos(\theta) - \sin(\text{alfa})\ \sin(\theta)$$

$$\sigma_2 = \text{distLidar} + \text{distObj}\cos(\text{alfa}) + \text{distLaser}\cos(\text{alfa})$$

$$\sigma_3 = \text{distObj}\sin(\text{alfa}) + \text{distLaser}\sin(\text{alfa})$$

$$\sigma_4 = \sin(\text{alfa})\ \cos(\theta)$$

$$\sigma_5 = \cos(\text{alfa})\ \sin(\theta)$$

```
%Posición del objeto respecto al origen
wT_Objeto(:,4)
```

$$\begin{pmatrix} \sigma_1 & -\sigma_5 - \sigma_4 & 0 & \cos(\theta)\ \sigma_2 + \text{distRobot}\cos(\theta) - \sin(\theta)\ \sigma_3 \\ \sigma_5 + \sigma_4 & \sigma_1 & 0 & \sin(\theta)\ \sigma_2 + \text{distRobot}\sin(\theta) + \cos(\theta)\ \sigma_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$\sigma_1 = \cos(\text{alfa})\ \cos(\theta) - \sin(\text{alfa})\ \sin(\theta)$

$\sigma_2 = \text{distLidar} + \text{distObj}\ \cos(\text{alfa}) + \text{distLaser}\ \cos(\text{alfa})$

$\sigma_3 = \text{distObj}\ \sin(\text{alfa}) + \text{distLaser}\ \sin(\text{alfa})$

$\sigma_4 = \sin(\text{alfa})\ \cos(\theta)$

$\sigma_5 = \cos(\text{alfa})\ \sin(\theta)$

```
%Posición del robot respecto al origen
wTr(:,4)
```

$$\begin{pmatrix} \text{distRobot}\cos(\theta) \\ \text{distRobot}\sin(\theta) \\ 0 \\ 1 \end{pmatrix}$$

```
%Posición del objeto respecto al origen menos - posición del robot ↩
    respecto al origen
PosFinal=simplify(wT_Objeto(:,4)-wTr(:,4))
```

$$\begin{pmatrix} \cos(\theta)\ (\text{distLidar} + \text{distObj}\cos(\text{alfa}) + \text{distLaser}\cos(\text{alfa})) - \sin(\text{alfa})\sin(\theta)\ (\text{distObj} + \text{distLaser}) \\ \sin(\theta)\ (\text{distLidar} + \text{distObj}\cos(\text{alfa}) + \text{distLaser}\cos(\text{alfa})) + \sin(\text{alfa})\cos(\theta)\ (\text{distObj} + \text{distLaser}) \\ 0 \\ 0 \end{pmatrix}$$

H.0.2. rotz.m

```
function RZ=rotZ(theta)

s=sin(theta); c=cos(theta);
if isnumeric(theta)
    if abs(c)<1e-10
        c=0;
    end
    if abs(s)<1e-10
        s=0;
    end
end

RZ=[c,-s,0,0;s,c,0,0;0,0,1,0;0,0,0,1];
```

H.0.3. desp.m

```
function D=desp(p)

D=[1,0,0,p(1);0,1,0,p(2);0,0,1,p(3);0,0,0,1];
```