UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

PROGRAMACIÓN, DISEÑO Y MONTAJE DE UN ROBOT MÓVIL CONTROLADO POR LIDAR Y COMANDOS DE VOZ

TRABAJO FINAL DEL

Máster Universitario en Ingeniería Mecatrónica

AUTOR:

Álvaro Serna Pérez

TUTOR:

Juan José Serrano Martín

Fecha: septiembre de 2021

RESUMEN

Este proyecto se compone del diseño mecánico de piezas para su impresión en 3D mediante SolidWorks y del diseño de software de un carro robotizado que, a través del uso de motores CC con encoder en las ruedas, es capaz de desplazarse hasta el punto deseado por el usuario mediante ordenes de voz. En el trayecto al punto de destino el robot es capaz de evitar objetos tanto fijos como móviles que se encuentren a 360 grados del robot gracias un sensor LIDAR implantado mediante un sensor de distancia laser de tiempo de vuelo (ToF) y un motor paso a paso. El programa es ejecutado desde un microcontrolador de ST y el programa de reconocimiento de voz en un pc externo y la comunicación serie de ambos se hace de forma transparente mediante un módulo radio.

SUMMARY

The project is composed by the mechanical design of pieces for a 3D printer using SolidWorks and software design of a robotic car, which movement is given through wheeled CC motors with encoder, that can move to a desired point giving voice commands by the user. On the way to the destination point the robot can avoid static and moving objects around the robot thanks to a LIDAR sensor made by a time-of-flight laser distance sensor (ToF) and a stepper motor. The embedded program is executed from a ST microcontroller and the voice recognition from an external pc where la serial communication between both is done transparently by a radio module.

INDICE

1 Introducción 5
2 Software 10
2.1 STM32CubeIDE 10
2.2 Termite 11
2.3 Pycharm 12
2.4 SolidWorks 12
2.5 Microsoft Office 12
2.6 Logic 12
3 Reconocimiento de voz 13
3.1 Librerías de Python: 13
3.2 Funciones del script: 14
3.3 Ajustes previos: 15
3.4 Explicación del código: 15
4 Selección de componentes 17
4.1 Polea síncrona dentada aluminio 18
4.2 Correa síncrona dentada 19
4.3 Rodamiento 20
5 Diseño de componentes mecánicos 21
5.1 Cubierta: 21
5.2 Pieza Sensor 22
5.3 Radio Holder 23
5.4 Soporte motor 23
5.5 Cubre polea 24
6 Componentes electrónicos 25
6.1 Convertidor DC-DC XL6009E1 25
6.2 Driver L298N para motores CC 25
6.3 Controlador del motor paso a paso DRV8825 26
6.4 Motor CC + Encoder 28
6.5 Sensor de distancia de tiempo de vuelo (ToF) VL53L1X 31
6.5.1 Funcionamiento 31
6.6 Motor paso a paso. 37
6.7 Modulo radio Telit LE50 – 868 38

6.8 Microcontrolador ST Núcleo-L476RG 40
6.9 Batería de Litio-Polimero (LiPo) 11V 41
6.10 Batería de ion-litio 5V-2A 42
7 Distribución de los elementos en el carro 43
7.1 Pinout Microcontrolador 45
8 Ensayo cálculo de pulsos por vuelta del encoder 46
9 Configuración de timers del microcontrolador 49
9.1 Ajuste de parámetros del reloj de tiempo real RTC 49
9.2 PWM del motor paso a paso 51
9.3 PWM motores ruedas 53
9.4 Timer de encoders 54
9.5 Timer interrupciones odometría 55
10 Odometría 55
10.1 Interrupciones 56
10.2 Cálculo de las velocidades de las ruedas 57
10.3 Modelado cinemático directo 58
10.3.1 Velocidades lineales de las ruedas: 59
10.3.2 Velocidad lineal instantánea: 59
10.3.3 Velocidad angular del vehículo: 59
10.3.4 Actualización de la posición 60
11 Calibración de los motores 62
12 Funciones del código 66
13 Máquina de estados 72
13.1 Flujogramas 73
14 Montaje y resultados 74
15 Conclusiones y vías futuras 80
15.1 Problemas encontrados 81
15.2 Vías futuras 82
16 Presupuesto 84
16.1 Costes materiales 84
16.2 Gastos personales 84
16.3 Coste de licencias 85
16.4 Presupuesto total 85

17 Pliego de condiciones 86
17.1 Definición 86
17.2 Condiciones técnicas 86
17.3 Condiciones económicas 86
18 Bibliografía 87
19 Anexos 88
19.1 Código de Python 88
19.2 Código de STM32 91
19.3 Planos 111
19.4 Datasheets 111

1 INTRODUCCIÓN

Este proyecto es fruto de la idea de adaptar una herramienta novedosa y práctica como es la interacción mediante comandos de voz a un campo que empieza a tener una cabida y frecuencia elevada en la industria, los robotización autónoma.

Con el objetivo de realizar todas las partes de un proyecto como este se ha hecho uso de conocimientos ya adquiridos en la carrera universitaria de Ingeniería Mecánica como el desarrollo en 3D de algunas piezas de chasis, de prácticas aprendidas durante este máster de Ingeniería mecatrónica como la selección de los componentes electrónicos, el control de los motores empleados, el uso del sensor de distancia y las comunicaciones necesarias para cumplir el objetivo del proyecto. Sin olvidarse del reto de aprender en tiempo reducido un nuevo lenguaje de programación con el objetivo de implantar un sistema de reconocimiento de voz lo más robusto posible y que pueda asemejarse a una aplicación real en el mundo de la producción industrial. El hecho de aprender y juntar diferentes campos de la ingeniería supone un bonito reto que es llevado a cabo por la ambición personal de reinventarse e integrarse en nuevas tecnologías desde un campo clásico como la mecánica.

Ilustración 1. Ejemplo de robot AGV.

Analizando la industria ya existen robots autónomos en varios sectores cuya utilidad es transportar objetos de un punto a otro. Estos robots funcionan de una forma previamente programada, es decir, un técnico o ingeniero establece los puntos a los que quiere ir y por donde debe pasar y lo recorre de forma repetitiva hasta el siguiente cambio que se realiza. No hay ningún sistema de movimiento autónomo en

el que se den instrucciones puntuales e improvisadas que pueden tener utilidad en servicios momentáneos y puede evitar desplazamientos largos y pesados o en zonas peligrosas del personal de la empresa.

Lo más parecido que se puede encontrar en la actualidad con aplicaciones industriales son robots AGVs (*Auto Guided Vehicles*), cuya presencia aumenta de forma notable con el paso de los años. Estos se pueden ver hoy en día en varios sectores como la automoción, alimentación, farmacia, aeronáutica o simple logística ofreciendo ventajas como la reducción en costes operativos del 20% (dependiendo del sector), trazabilidad y puntualidad del transporte del producto del 99,9 % y la capacidad de producción aumenta también notablemente en sectores como la automoción.

Las formas que tienen estos robots colaborativos de moverse y, más importante, de orientarse son varias, que se aplican según casos y aplicaciones;

- Navegación magnética: El robot sigue una cinta magnética enterrada en el suelo del lugar donde trabaja. El AGV utiliza una antena magnética para detectarla.

Ilustración 2. Esquema de Navegación magnética

- Navegación óptica: El AGV utiliza una cámara con la que, gracias a la visión artificial, lee las rutas pintadas en el suelo.

Ilustración 3. Esquema de navegación óptica

- Navegación láser: Se instalan reflectores en zonas estratégicas de la planta para garantizar que el AGV puede detectar siempre 3 de ellos a la vez en cualquier punto de la trayectoria.

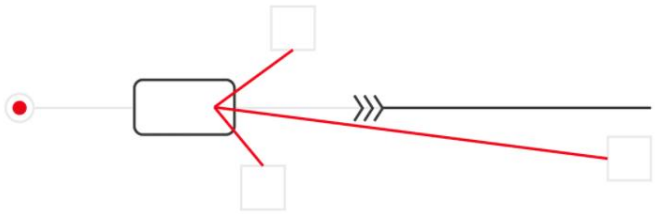


Ilustración 4. Esquema de navegación láser

- Navegación QR: Un sistema parecido a la navegación óptica, pero en este caso la cámara capta unos códigos QR situados en el suelo de la trayectoria del robot como si fuese una matriz y que al leerlos proporciona información sobre su posición y cuáles son las siguientes instrucciones que debe seguir.

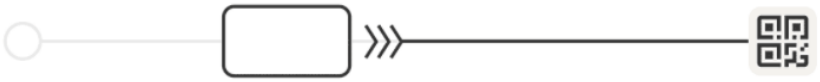


Ilustración 5. Esquema de navegación por lectura de códigos QR.

- Navegación natural: Este es el sistema que más se podría asimilar al empleado en el proyecto actual. El robot lleva un sensor láser que escanea el ambiente en el que se encuentra y mediante técnicas como SLAM crea un mapa virtual de los alrededores del robot y traza una trayectoria según sea óptimo.

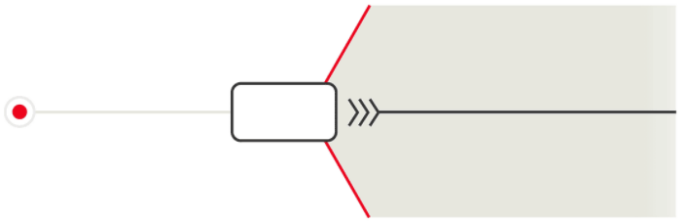


Ilustración 6. Esquema de navegación natural.

- Navegación dual: Combina 2 de los métodos descritos anteriormente, los cuales se alternan según sea más adecuado en ese momento.

En este caso el proyecto tendrá un sensor láser que escaneará el ambiente, pero no creará un mapa virtual debido a la complejidad del método para conseguirlo, sino que detectará los objetos alrededor del robot en 360 grados (como si de un sensor LiDAR de pulsos se tratase) y en función de donde encuentre el objeto más cercano y se esquivará haciendo un giro hacia el lado contrario donde se haya encontrado el objeto.

La trayectoria que recorre el robot es siempre una línea recta entre el punto de partida en el que se encuentra y el punto final. La trayectoria se calcula mediante trigonometría simple y en el momento el que encuentra un objeto y se desvía de la trayectoria para esquivarlo ha de calcularse una nueva trayectoria para llegar al punto final.

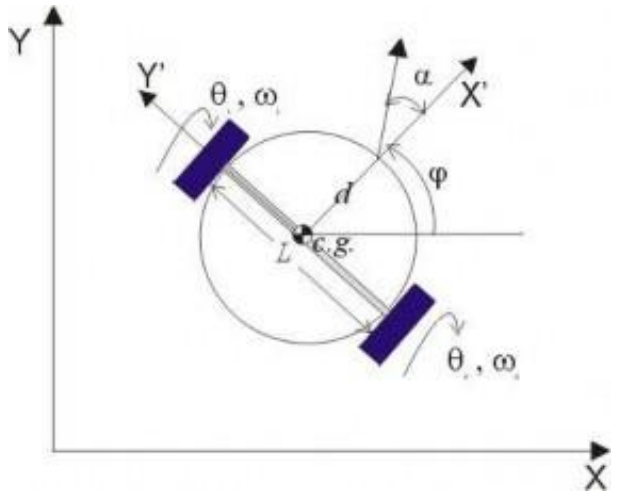


Ilustración 7. Representación cartesiana de odometría diferencial.

La posición del vehículo se obtiene mediante un sistema de odometría en unos ejes cartesianos X e Y, que se establece con una velocidad de actualización muy rápida con el objetivo de ajustar lo máximo posible la posición y no acumular demasiados errores. Este sistema de odometría no es ni mucho menos muy preciso, pero resulta válido para un proyecto en el que se cuenta con pocos recursos y es válido para realizar un prototipo en el que, en vías futuras, podría adaptarse y mudar a otro sistema de localización mucho más preciso y funcional.

Como ya se ha comentado, la interacción con el robot se hace a través de comandos de voz. Con conocimientos básicos de Python se puede escribir un script en el que utilizando el motor de conversión de voz a texto *"speech to text"* de Google en el que

se pueda despertar el script con un comando de llamada para después indicar en voz alta el punto al que se quiera mandar el robot.

Una vez que se sepa a que punto va a ir el robot es el mismo script de Python el que mande un comando por comunicación serie al microcontrolador. Esto no se puede hacer a través del clásico cable por lo que se hace a través de un módulo de radio.

El objetivo del proyecto es desarrollar todos los aspectos dichos desde cero. Obteniendo los resultados más precisos y útiles para una posible aplicación industrial.

2 SOFTWARE

Se entiende como software como el equipamiento de programas de cómputo que se emplean para realizar tareas en un sistema informático. Se han empleado diferentes programas para abarcar las diferentes áreas que componen. Es aquí donde se enumeran estos programas y se explican cuáles son los ajustes previos al uso si es que lo hay en cada uno de ellos.

2.1 STM32CubeIDE

Programa de desarrollo propio de la empresa STMicroelectronics cuyas funciones incluyen la configuración de pines GPIO, de timers y comunicaciones de la placa, la configuración del reloj de tiempo real RTC y editor de texto y compilador en lenguaje C (basado en eclipse) para la escritura del código del programa.

Ilustración 8. Logotipo STM32

Para trabajar con este programa se han seguido los siguientes pasos.

1. Se selecciona el microcontrolador con el que se trabaja de la lista.
2. Se selecciona la opción de generar un archivo .c y .h por cada periférico usado para mayor claridad.
3. Se comprueba la velocidad de funcionamiento del reloj del microcontrolador. En este caso se ha dejado como está, a 80 MHz. Esto es de gran importancia para la configuración de los timers más adelante.

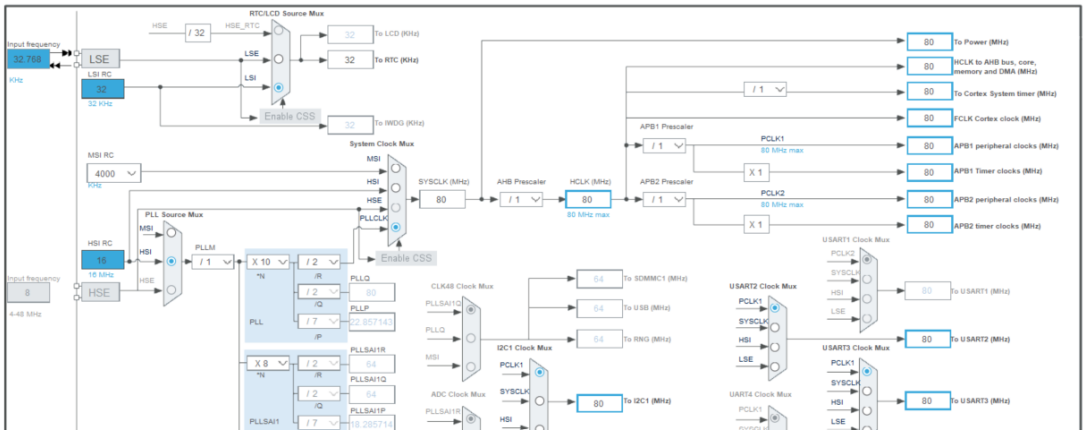


Ilustración 9. Muestra de los ajustes de reloj del proyecto

Para activar tanto las comunicaciones como los GPIO como los *timers* se hace desde el configurador de pines, donde se pueden ajustar con varios detalles.

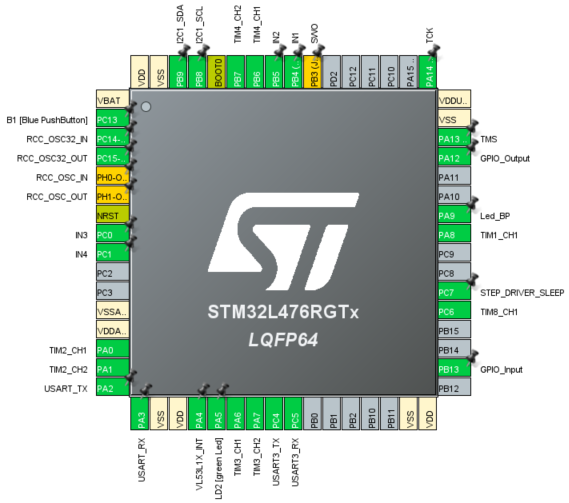


Ilustración 10. Muestra del Pinout del proyecto.

2.2 Termite

Hyperterminal RS232 utilizado para leer las comunicaciones con el microcontrolador. Tiene como ajustes previos tan solo son velocidad de *baudrate* (en este caso la comunicación USART trabaja con 115200 baudios) y el puerto COM donde se encuentre conectado el dispositivo con el que se quiere hacer la comunicación serie.

Ilustración 11. Logotipo termite

2.3 Pycharm

Es un IDE con depurador utilizado para desarrollar Python. Como ajustes previos simplemente instalar el compilador de Python e instalar las librerías necesarias.

Ilustración 12. Logotipo Pycharm

2.4 SolidWorks

Es un programa de diseño CAD y CAE con el que se desarrollan las piezas a imprimir en impresora 3D y sus respectivos planos.

Ilustración 13. Logotipo SolidWorks

2.5 Microsoft Office

El paquete office se ha utilizado para realizar tanto tablas, cálculos y documentación. Se han utilizado los programas Microsoft Word, Microsoft Excel y Microsoft PowerPoint.

2.6 Logic

Programa destinado para la visualización de señales digitales mediante el uso de un analizador lógico. Se ha utilizado en situaciones puntuales.

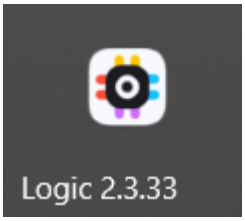


Ilustración 14. Logotipo Logic

3 RECONOCIMIENTO DE VOZ

A la hora de interactuar con el robot se han barajado diferentes opciones como pueden ser el control mediante un mando a distancia vía Wifi o Bluetooth pero con el objetivo de hacer un manejo del robot más sencillo y más natural se ha decantado por una interacción mediante comandos de voz, con un simple funcionamiento en el que se hace un llamamiento mediante una palabra clave "robot" y a continuación decir un comando que el programa reconocerá y asignará a un punto al que el robot se desplazará.

Este programa de reconocimiento de voz ha sido escrito en Python y el funcionamiento del script a grandes rasgos es que reconoce el punto al que se desea mover y envía, por comunicación serie, un comando al microcontrolador que este ha de reconocer y discriminar en función del comando recibido para que el robot siga unos pasos determinados.

El script de este reconocimiento de voz funciona de la siguiente forma:

3.1 Librerías de Python:

Para poder usar algunas funciones necesarias para el propósito buscado es necesario instalar algunas librerías de Python que se describen a continuación:

- *Speech_recognition:* Librería con la función de reconocimiento de voz de Google. Se ha utilizado esta librería porque, aunque tiene la desventaja de que necesita conexión a internet ya que envía el audio a los servidores de Google que son lo que hacen el paso de texto a audio, es sin ninguna duda el que mejor funciona tras probar otros servicios disponibles. Lo ideal sería utilizar un servicio que funcione offline pero no es posible recurrir a un servicio offline que funcione con una mínima calidad.
- *Serial:* Librería necesaria para establecer la comunicación serie con el microcontrolador, tanto para enviar como para recibir.
- *Time:* Librería con la utilidad de implementar esperas dentro del desarrollo del programa.
- *Pyttsx3:* Tiene la función de convertir un texto escrito a voz

3.2 Funciones del script:

Se han creado una serie de funciones para poder hacer la transformación de voz a texto óptima.

- *listen_init:* Su objetivo es hacer la escucha del comando inicial y hacer la transformación del audio recibido a texto. Cuando no encuentra ningún texto imprime por pantalla un mensaje informativo indicando que no se ha escuchado nada
- *listen_sentence:* Su objetivo es hacer la escucha del punto al que ha de moverse, por eso tiene un tiempo algo más largo que el de listen_init. También hace la transformación del audio recibido a texto. Cuando no encuentra ningún texto imprime por pantalla un mensaje informativo indicando que no se ha escuchado nada.
- *filter_word:* Función que se encarga de encontrar la palabra indicada en el parámetro de entrada dentro del texto previamente obtenido a partir del comando de voz. Si esta palabra se encuentra se envía un True
- *speak:* Transforma un texto escrito en audio hablado que sale por altavoces.

3.3 Ajustes previos:

Antes de empezar el bucle del script, hay que hacer unos ajustes previos tanto de las librerías como de los periféricos.

En primer lugar, hay que iniciar el motor de transformar el texto a voz, en el cual hay ajustar la velocidad de habla y el idioma en el que leerá el texto que se quiere escuchar (en este caso castellano).

En cuanto al transformador de voz a texto de Google hay que seleccionar el método de entrada (micrófono) y el idioma en el que está el audio que se ha escuchado (castellano).

Para la comunicación serie hay que definir el puerto *COM* en el que se encuentra el microcontrolador y algunos otros parámetros típicos de una comunicación serie como el Baudrate, el tamaño del bit que se envía, la paridad y la cantidad de bits finales.

3.4 Explicación del código:

En este punto tiene como objetivo explicar detalladamente el código escrito para esta interacción del usuario con el robot.

En primer lugar, se utiliza una función propia de la librería de reconocimiento de voz de Google llamada *adjust_for_ambient_noise* que, durante un segundo, escucha con el método de entrada seleccionado el sonido ambiente para poder filtrar ese ruido de lo que se reciba más adelante en las diferentes escuchas que se realicen. Esto es un paso opcional pero que mejora considerablemente la fiabilidad del motor de Google de escucha.

Tras este paso se entra en el bucle del programa, en el cual se comienza escuchando el sonido mediante la función *listen_init* para escuchar la palabra clave que encenderá el robot, en este caso la palabra "robot". Cuando se encuentra esta palabra clave continua con el siguiente paso. Si no encuentra la palabra robot se repetirá de forma indefinida este paso hasta que encuentre esta palabra clave.

Cuando ya se ha llamado al robot comienza la segunda escucha mediante la función *listen_sentence* con la cual escucha mediante el micrófono la frase que diga el usuario

y la pasa a texto mediante el motor de Google y busca, en esa frase, una serie de palabras clave que indican el punto al que se desea desplazar, estas palabras son:

- Punto A: A, 1, laminación….
- Punto B: B, 2, estampación …
- Punto C: C, 3, estampación …

Si encuentra una de estas palabras da un valor a la variable "destino" según el punto al que se vaya a enviar para, posteriormente, enviarlo por comunicación serie al microcontrolador mediante la función de la librería establecida para ello y una previa codificación en ASCII.

Una vez enviado el comando para que pueda empezar a trabajar el microcontrolador el programa de reconocimiento tan solo debe esperar a la respuesta del microcontrolador una vez que haya terminado el programa. Para esto simplemente se pone a escuchar todos los comandos que se envían por serie hasta el PC y cuando se reciba el comando "FIN". Entonces cierra la comunicación serie y vuelve a empezar el programa.

Es importante destacar que cuando se reciben datos por el canal serie, en este caso por USART, al igual que para enviar los datos hay que codificar el mensaje en este caso hay que decodificarlo mediante el uso de la extensión indicada a continuación.

```
mensaReci = TerminalSerial.read(8).decode('utf-8')
```

Ilustración 15. Ejemplo de decodificación de mensaje recibido por terminal serie

El código íntegro del script está adjuntado al final de la memoria en el apartado de anexos.

4 SELECCIÓN DE COMPONENTES

Antes de hacer el diseño de piezas 3D se hace la selección de componentes, puesto que es imposible encontrar componentes mecánicos que coincidan exactamente con las necesidades del proyecto, pero sí que se pueden diseñar las piezas a imprimir con las medidas exactas que se quieran sin demasiados problemas.

El chasis del robot está compuesto por 2 tablas de plástico rígido fijadas entre ellas mediante tornillería. Toda la tornillería se aísla mediante unos protectores de plástico para evitar falsos contactos entre los componentes que puedan provocar cortocircuitos.

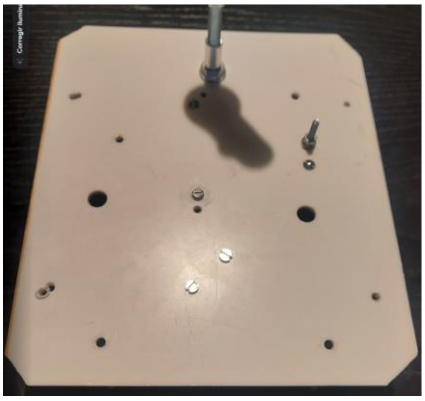


Ilustración 16. Chasis del robot.

En cuanto a la transmisión del movimiento se emplean 3 ruedas, 2 de ellas están acopladas mediante una pieza atornillada a motores de corriente continua. Estas son ruedas de goma con cámara de aire de 65 milímetros de diámetro y dibujos en la llanta para aumentar la tracción con el suelo.

Ilustración 17. Ruedas empleadas.

La tercera rueda es una rueda loca de metal con llanta de goma que simplemente sirve de apoyo para el carro y que no establece ninguna dirección ni movimiento.

Para la transmisión del movimiento desde el motor paso a paso a la pieza "cubierta sensor" donde se alberga el sensor ToF se hace mediante una correa de transmisión dentada (tras probar insatisfactoriamente una goma elástica que da problemas con la precisión) que ha de ser seleccionada junto con piezas convenientes.

4.1 Polea síncrona dentada aluminio

El movimiento se transmite en primer lugar desde el motor paso a paso. Para poder enganchar la correa se hace uso de una polea síncrona dentada. Esta ha de tener los requisitos de tener el mismo diámetro interno que el eje del motor paso a paso para poder encajarlo en él y que tenga un diámetro externo 3 veces más pequeño que la parte donde hace contacto la correa en la pieza "cubre sensor".

Tras aclarar esto, se selecciona la polea síncrona aluminio, PC relleno de vidrio con 22 dientes, de 2 mm de espaciado, 5mm de calibre del vendedor "RS Components"

Ilustración 18. Polea síncrona de RS

4.2 Correa síncrona dentada

Dado que se tiene el modelo 3D de la pieza que ha de soportar la correa y se ha seleccionado la rueda dentada que se encaja en el motor paso a paso se puede hacer un cálculo de la longitud que debe tener la correa dentada a seleccionar. Esto se hace apoyándose en un croquis de SolidWorks para poder hacer las mediciones de cada lado del croquis.

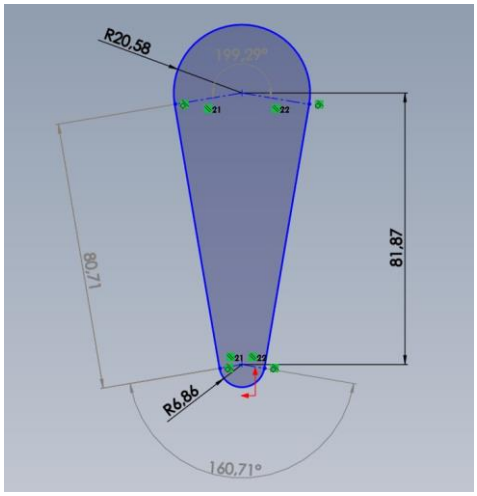


Ilustración 19. Croquis para cálculo de correa síncrona dentada.

Haciendo la suma de las cotas que se pueden ver en el croquis:

$$80.71 + 80.71 + 71.58 + 19.24 = 252.24\ mm$$

Con este dato y el requisito que tenga el mismo espaciado de dientes que la polea síncrona seleccionada se selecciona la correa síncrona RS PRO de $256.03\ mm$. La pequeña diferencia con el cálculo hecho significa que quedará un poco más holgada pero no supone un problema real en la transmisión del movimiento.

Ilustración 20. Correa síncrona dentada.

4.3 Rodamiento

Para facilitar el giro de la pieza "cubierta sensor" sobre la pieza "cubierta" se hace uso de un rodamiento. Este rodamiento debe tener como requisitos que el diámetro interior ha de ser el mismo que el diámetro exterior del saliente de la pieza "cubierta" y que el diámetro exterior del rodamiento sea el mismo que el diámetro interior de la pieza "cubierta sensor".

Ilustración 21. Rodamiento

Se ha seleccionado el rodamiento de bolas de ranura profunda RS PRO, de diámetro interior 20mm y de diámetro exterior 42mm.

5 DISEÑO DE COMPONENTES MECÁNICOS

A la hora de construir el coche robotizado se ha hecho uso del recurso de la impresión 3D por el cual se imprime en plástico las piezas que se han diseñado previamente en un software CAD, en este caso SolidWorks, y que se usan tanto como de soporte de los elementos electrónicos y motores como para formar parte del chasis del robot. El diseño de estas piezas ha sido realizado, de forma íntegra, por el autor del trabajo. Las piezas construidas son las que se describen a continuación.

5.1 Cubierta:

Esta cubierta tiene el objetivo de cubrir el microcontrolador, el motor y el DC-DC del robot. Sirve de soporte del rodamiento que hará el giro del sensor ToF y para fijar, en la parte trasera, el módulo de radio para la transmisión serie.

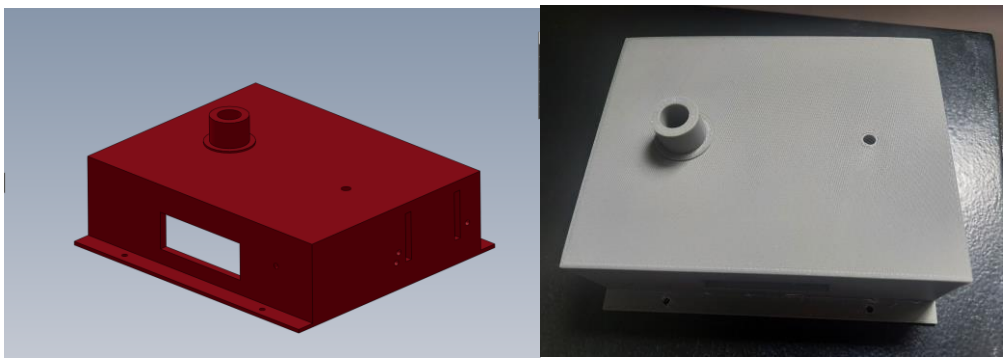


Ilustración 22. Pieza impresa. Cubierta del robot

5.2 Pieza Sensor

El sensor ha de realizar un giro de 360 grados para poder abarcar todo el campo que se encuentra alrededor del robot. Para hacer este giro el sensor ha de apoyarse en una pieza que sea capaz de albergar dentro de si el sensor VL53L1X con su módulo de resistencias de *pull-up* y los cables *dupont*. Esta pieza gira con el rodamiento, por lo que debe tener un espacio para poder insertarlo y tiene una franja dentada en la cual se ajusta la cinta de transmisión del movimiento proveniente del motor paso a paso.

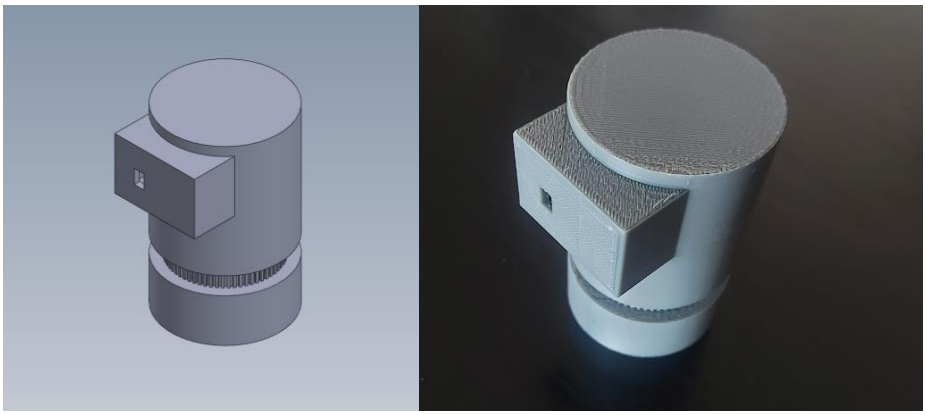


Ilustración 23. Pieza impresa. Pieza sensor.

5.3 Radio Holder

Esta tiene la función de mantener fijo el módulo de radio Telit a la pieza cubierta, de modo que no se suelten las conexiones y que se mantenga la antena fija en la misma posición para evitar posibles errores en la comunicación.

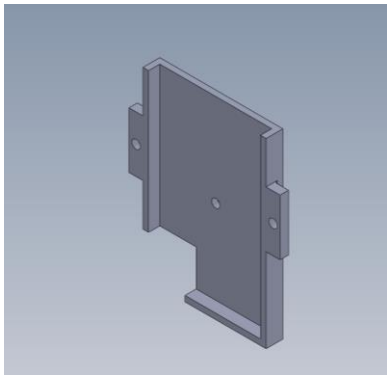


Ilustración 24.Pieza impresa. Radio Holder

5.4 Soporte motor

Esto es un pequeño soporte cuya única función es levantar el motor paso a paso para poder ajustar la altura de la rueda dentada que transmite el movimiento a la pieza donde se encuentra el sensor.

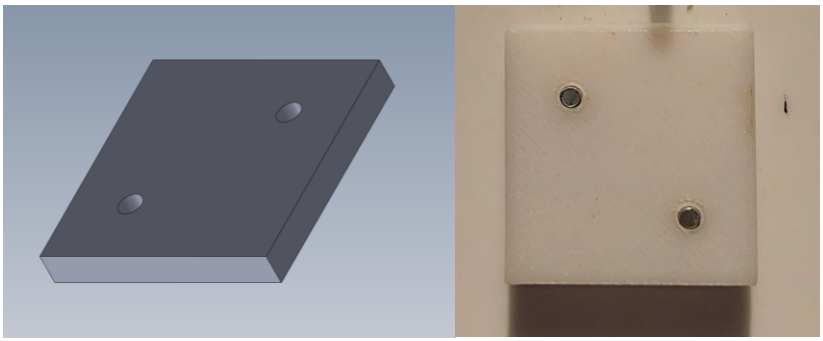


Ilustración 25. Pieza impresa. Soporte motor.

5.5 Cubre polea

Esta una pieza que se utiliza de forma estética con el objetivo de cubrir y ocultar la correa de transmisión.

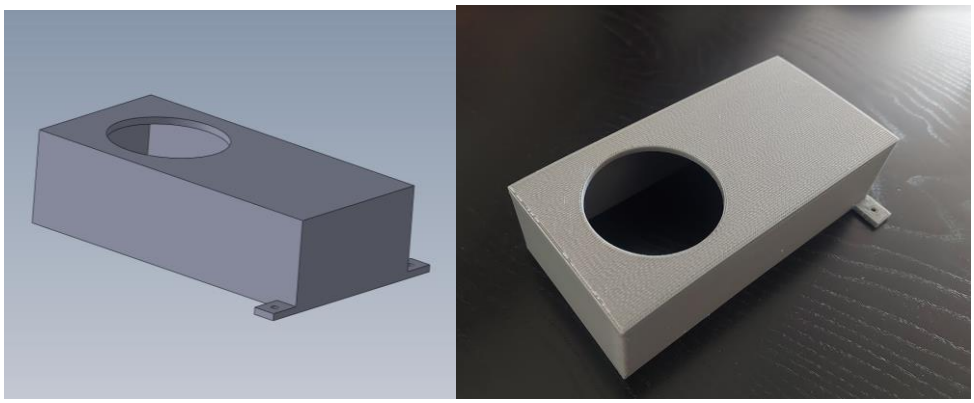


Ilustración 26. Pieza impresa. Cubre polea.

6 COMPONENTES ELECTRÓNICOS

Los movimientos del robot se logran gracias a los 2 motores de corriente continua y al motor paso a paso, pero, para el control de estos es necesario de algunos componentes extra que también se describirán en este apartado.

6.1 Convertidor DC-DC XL6009E1

Tanto los motores CC que se encuentran en las ruedas como el motor paso a paso del movimiento del sensor se alimentan con 12 V. La salida de tensión que proporciona el microcontrolador es de 5V como máximo por lo que se ha optado por este convertidor de tensión como solución a este problema. Es un elevador de tensión "Boost up" ajustable mediante potenciómetro con entrada de 5V y salida, en este caso, de 12 V.

Ilustración 27. Elevador de tensión DC-DC XL6009E1.

6.2 Driver L298N para motores CC

Este es un controlador capaz de controlar 2 motores de corriente continua. Se le aplica una tensión de entrada de 12V y tiene 6 pines como entradas de control de los motores (3 para cada motor) que son:

- ENA: Entrada de PWM del motor 1.
- IN1: Entrada del pin que indica la dirección 1 del motor 1.
- IN2: Entrada del pin que indica la dirección 2 del motor 1.
- IN3: Entrada del pin que indica la dirección 1 del motor 2.
- IN4: Entrada del pin que indica la dirección 2 del motor 2.
- ENA: Entrada de PWM del motor 2.

Ilustración 28. Controlador de los motores de corriente continua de las ruedas

6.3 Controlador del motor paso a paso DRV8825

Ilustración 29. Controlador DRV8825 del motor paso a paso

Este es un controlador necesario para poder trabajar con el motor paso a paso con el que cuenta el proyecto. Se alimenta a 5V y hay tiene varias entradas y pines para el control del motor.

- El pin DIR es el que determina la dirección de giro del motor, horario con 1 y antihorario con 0.
- En el pin STEP se aplica el PWM para mover el motor los pasos deseados en la aplicación.
- En el pin SLEEP y RESET se aplica tensión cuando se quiere que trabaje el motor. A este pin se le deja de dar tensión cuando no trabaja para ahorrar electricidad y evitar sobrecalentamientos del motor y del controlador

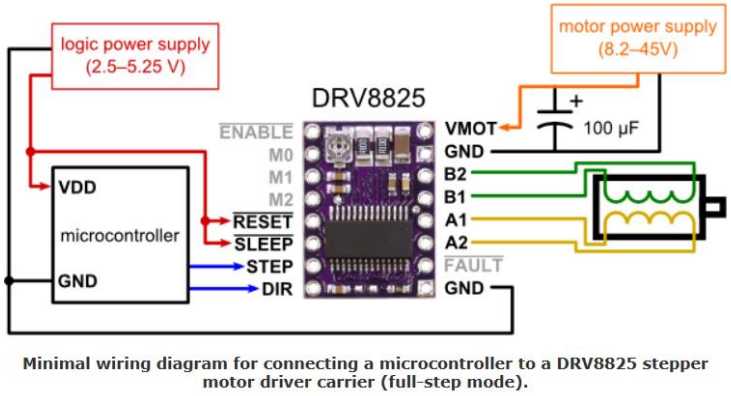


Ilustración 30. Pinout de las conexiones del controlador DRV8825.

El controlador tiene la posibilidad de variar la tensión de fase del motor a través de un potenciómetro.

Para este motor paso a paso, que tiene una corriente de 1.7 A por fase, es necesario limitar el controlador a una intensidad de 1.7 A aproximadamente.

$$I_{limite} = V_{ref} \cdot 2$$
$$1{,}7 = V_{ref} \cdot 2$$
$$V_{ref} = 0{,}85\ V$$

Según la hoja de especificaciones del controlador, para conseguir un paso completo del motor, que es lo más conveniente en el proyecto, hay que aplicar un 71% de la tensión límite del motor, por lo que la tensión que hay que ajustar en el controlador con el potenciómetro es:

MODE2	MODE1	MODE0	STEP MODE
0	0	0	Full step (2-phase excitation) with 71% current
0	0	1	1/2 step (1-2 phase excitation)
0	1	0	1/4 step (W1-2 phase excitation)
0	1	1	8 microsteps/step
1	0	0	16 microsteps/step
1	0	1	32 microsteps/step
1	1	0	32 microsteps/step
1	1	1	32 microsteps/step

Ilustración 31. Tabla de la hoja de especificaciones del modo de trabajo del controlador

$$V_{pot} = 0{,}85 \cdot 0{,}71 = 0{,}6\,V$$

En los pines de la derecha del controlador se conectan los bornes de las 2 bobinas del motor paso a paso y en la alimentación del motor a 12 V ha de ponerse un condensador previamente.

Con en fin de facilitar las conexiones del controlador del motor paso a paso se ha hecho un pequeño circuito en el que se incluye el condensador y los pines necesarios para conectar al microcontrolador.

Ilustración 32. Mini circuito adaptado para el controlador del motor paso a paso

6.4 Motor CC + Encoder

Ilustración 33. Motor cc con encoder empleado.

Para desplazar el robot se hace uso de motores de corriente continua con tensión nominal a 12 voltios en cada una de las ruedas. No se tienen datos de fabricante del motor (excepto velocidad máxima de 110rpm) por lo que para obtener sus características se hace referencia al trabajo indicado en bibliografía en el que, experimentalmente, se obtiene tan solo el siguiente dato:

- Pares de polos: 1

Un motor de corriente continua (motor CC o DC) transforma la energía eléctrica en energía mecánica gracias a la inducción magnética y se compone principalmente de 2 partes:

- Rotor: Es la parte del motor que sufre el movimiento. Este movimiento ocurre gracias a la polaridad que aparece en el devanado de cobre al circular la corriente eléctrica.
- Estator: Es la parte fija del motor y donde se encuentran los imanes permanentes que hacen que se atraiga y repulse el rotor para generar el movimiento.

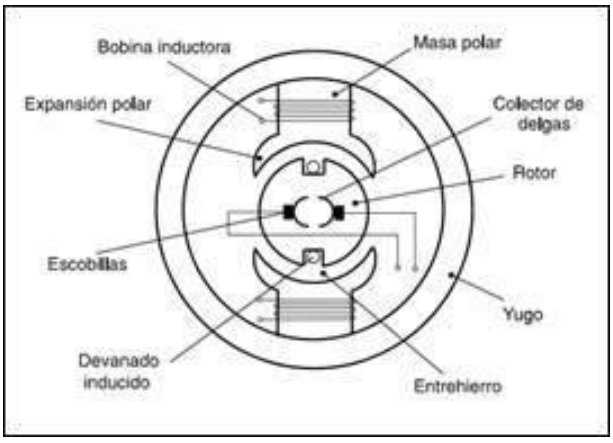


Ilustración 34. Estructura interna motor CC

El sentido de giro del motor es fácilmente controlable, pues invirtiendo el sentido de la corriente cambiará la polaridad del devanado del rotor y así cambia el sentido de giro.

En este caso, para controlar la velocidad de giro del motor se hará uso de la técnica de modulación de ancho de pulsos (PWM) por la cual, cuanto mayor sea el ciclo de

trabajo de la onda cuadrada a la que se somete el motor de corriente continua, mayor será la velocidad de giro del motor. Con un ciclo de trabajo máximo es como como conectar el motor a la tensión nominal directamente.

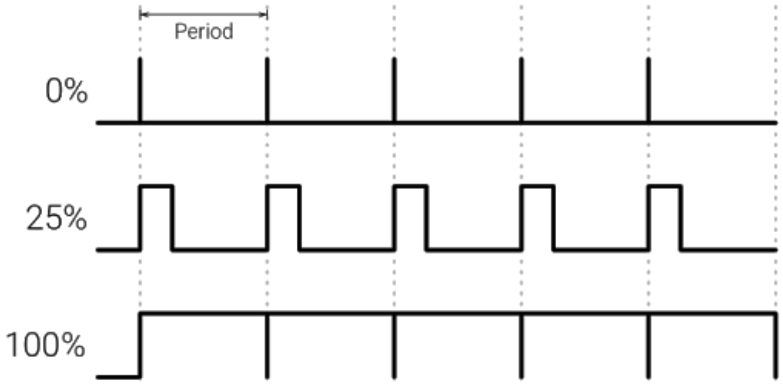


Ilustración 35. Ejemplo PWM

El motor usado viene incorporado con un encoder de cuadratura que a su vez usa 2 sensores de efecto hall para de medir el giro que ha realizado el motor en cualquier momento.

Cada uno de los sensores de efecto hall está conectado a un canal que se lleva hasta el microcontrolador y en cada uno de los canales se generan 2 ondas cuadradas que están desfasadas 90º y comparando las 2 ondas cuadradas aumenta o disminuye la cuenta del encoder.

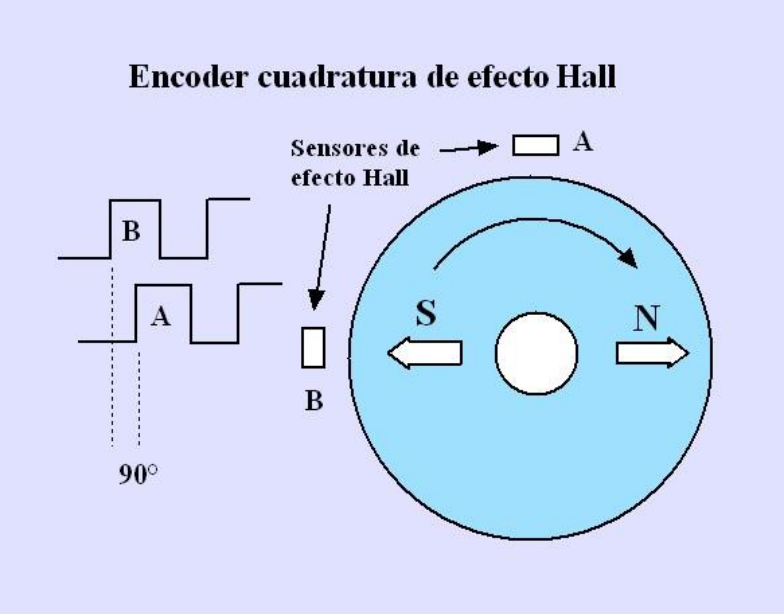


Ilustración 36. Estructura interna Encoder

Los encoders de cuadratura se utilizan para conocer la velocidad de giro del motor más adelante contando los pulsos que da en un tiempo determinado.

Estos motores también cuentan con una relación de transmisión reductora entre el giro del motor y el giro de la rueda, pero no se tienen ninguna información por parte del fabricante sobre esta relación. Para obtener estos datos se recurre a un ensayo en el que gracias a un ensayo con un sensor de efecto hall se calcula el número de pulsos que da el encoder en una vuelta de rueda.

La distribución de pines de estos motores es la siguiente:

Ilustración 37. Conexiones motor CC + encoder empleado

6.5 Sensor de distancia de tiempo de vuelo (ToF) VL53L1X

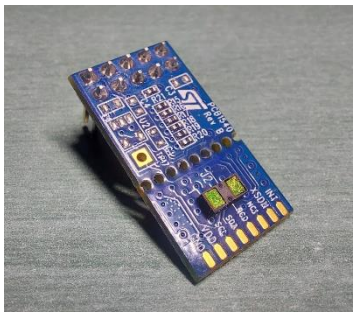


Ilustración 38. Sensor ToF VL53L1X

Para conseguir el sensor LIDAR deseado se ha optado por un sensor láser de tiempo de vuelo de la marca STMicroelectronics. En este apartado se explicará el funcionamiento de este tipo de sensores, sus conexiones y su aplicación para conseguir el sensor LiDAR deseado.

6.5.1 Funcionamiento

Estos son unos sensores que determinan la distancia de los cuerpos que se encuentran en la parte frontal del sensor (dentro del rango del sensor). Esta distancia se determina mediante la medición del tiempo que tarda el láser en salir del sensor, reflejar con el cuerpo y volver a la célula del sensor. Este tipo de sensores tienen varias utilidades como puede ser la de este proyecto, crear un sensor LiDAR u otros como los sistemas SLAM o algunas más simples como sensores de posición en móviles o en automóviles.

Este sensor seleccionado es la tercera generación de este tipo de sensores que fabrica la empresa STMicroelectronics. Es un sensor que puede detectar objetos a una distancia máxima teórica de 4000 mm con un campo de visión máximo de 27º con una precisión de 1 mm.

Con este sensor vienen varios modos de funcionamiento como el de *SIngleRangingMode o* el *MultiRangingMode* que permite realizar medidas de forma continua o medidas individuales.

Para la comunicación del sensor con el microcontrolador se hace uso de la interfaz I^2C (Inter-integrated circuit) de 400 kHz, que es bus serie.

Este sensor, en este caso, viene dentro de una placa denominada *Satellite* que viene con las conexiones necesarias para una fácil integración del dispositivo con un microcontrolador.

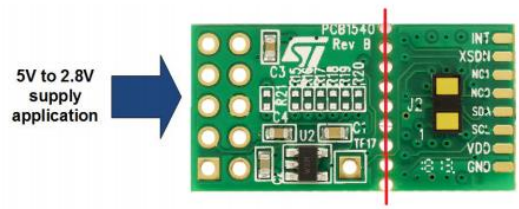


Ilustración 39. Sensor ToF VL53L1X

El modo de trabajo normal de este sensor es insertando el sensor en la placa de evaluación P-NUCLEO-53L1A1. Esta placa puede albergar 3 sensores de este tipo y funcionan de forma estática, es decir, la placa de expansión P-NUCLEO-53L1A1 se inserta encima del microcontrolador gracias a las conexiones de Arduino y tiene las funciones de medir las distancias que se encuentran delante de los 3 sensores y detectar los gestos que se realizan delante de ellos. Este modo de trabajo del sensor choca frontalmente con el objetivo del proyecto, pues el sensor LiDAR que se busca construir debe medir en 360 grados y para ello, este sensor ha de girar sobre sí mismo y así medir a todo su alrededor.

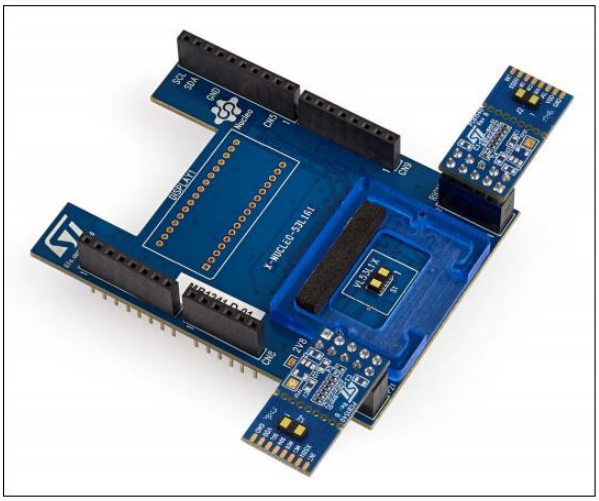


Ilustración 40. X-NUCLEO -53L1A1 expansion board

Con el fin de usar el sensor de independiente se realizan 3 acciones.

1. Conectar los pines del sensor directamente al microcontrolador:

   Conociendo la distribución de pines del sensor y conociendo las bases de las conexiones de comunicación I^2C se conectan los pines de SCL (clock), SDA (data), VDD (conexión a 3,3 Voltios), GND e INT (interrupción).

Figure 2. **Satellite schematic and list of materials**

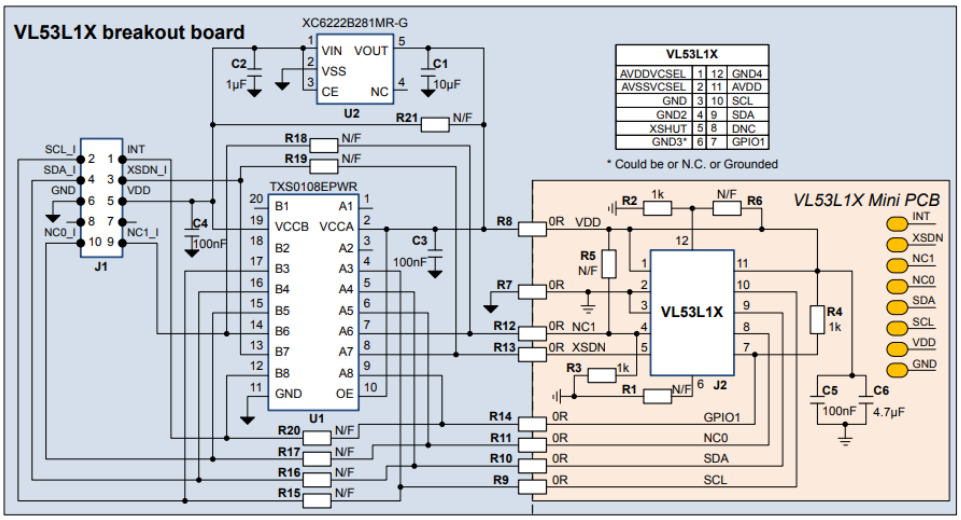


Ilustración 41. Esquema interno X-NUCLEO -53L1A1

2. Soldar resistencia de pull-up:

   Observando el esquema electrónico del P-NUCLEO-53L1A1 *board expansion* se puede ver que integra unas resistencias de pull-up que, al conectar el sensor directamente, no se encuentra y hay que poner. Las resistencias de pull-up son de 47k Ohmios y se sueldan a la salida del SCL y SDA.

3. Instalar librerías y API del sensor:

   El fabricante ofrece una API en su web en la que vienen los modos de funcionamiento del sensor y unos ejemplos para poder utilizarlo directamente. El problema viene a la hora de adaptar e instalar esto en el proyecto de STM32 en el que se está trabajando, para lo cual hay que hacer los siguientes pasos:

a. Se guardan e incluyen las librerías del sensor VL53L1X que vienen en la API dentro de la carpeta de Inc del Core.

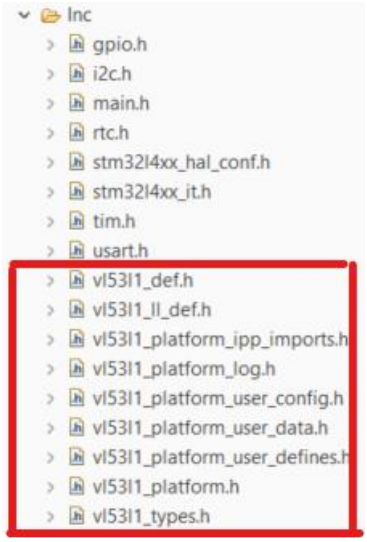


Ilustración 42. Librerias API VL53L1X

b. De la misma forma se incluyen en la subcarpeta del Core llamada Src (source) el archivo dels sensor VL53L1X.

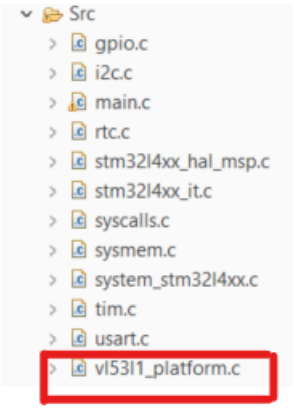


Ilustración 43. Librerias API VL53L1X

c. Dentro de la carpeta *"Drivers"* en el proyecto hay que incluir varios archivos. Lo primero, se crea la carpeta BSP con los archivos de la placa de expansión P-NUCLEO-53L1A1.

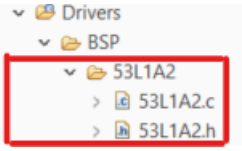


Ilustración 44. Librerias API VL53L1X

También dentro de la carpeta BSP se copian los archivos de la API correspondientes a los componentes.

Ilustración 45. Librerias API VL53L1X

d. Cambiar todos los *includes* referidos a la placa por la placa que se está utilizando, la placa L476RG, y en el main los *includes* del sensor y la placa de expansión.

```
/* USER CODE BEGIN Includes */
#include "vl53l1_api.h"
#include "53L1A2.h"
```

Ilustración 46. Includes en el código

e. Por último, la inicialización. Esta es una parte del código que se ha escrito y adaptado a dentro del *main* desde el archivo inicial. En estas líneas de código se establecen parámetros importantes como la dirección de la comunicación del I2C, leer los datos de identificación del modelo y se establecen los primeros estados de la compleja máquina de estados que tiene la API del sensor.

```
//printf("VL53L1X Examples...\n");
 Print_Usart2("VL53L1X Examples...\n");
   Dev->I2cHandle = &hi2c1;
   Dev->I2cDevAddr = 0x52;

   /* Allow to select the sensor to be used, multi-sensor is not managed in this example.
   Only when use use the Left ToF in interrupt mode,  solder the U7 on the X-Nucleo-53L1A1 board
   Only when use the Right ToF in interrupt mode, solder the U7 on the X-Nucleo-53L1A1 board
   See "Solder drop configurations" in the X-Nucleo-53L1A1 User Manual for more details */
   ToFSensor = 1; // Select ToFSensor: 0=Left, 1=Center, 2=Right
   status = XNUCLEO53L1A1_ResetId(ToFSensor, 0); // Reset ToF sensor
   HAL_Delay(2);
   status = XNUCLEO53L1A1_ResetId(ToFSensor, 1); // Reset ToF sensor
   HAL_Delay(2);

   VL53L1_RdByte(Dev, 0x010F, &byteData);
   sprintf(buff_vis,"VL53L1X Model_ID: %02X\n\r", byteData);
   Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;
   VL53L1_RdByte(Dev, 0x0110, &byteData);
   sprintf(buff_vis,"VL53L1X Module_Type: %02X\n\r", byteData);
   Print_Usart2(buff_vis);/*Print_Usart3(buff_vis);*/
   VL53L1_RdWord(Dev, 0x010F, &wordData);
   sprintf(buff_vis,"VL53L1X: %02X\n\r", wordData);
   Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;

   /*-----------------------------SI SE MIDE EN EL WHILE 1 -------------------------*/
    status = VL53L1_WaitDeviceBooted(Dev);
    status = VL53L1_DataInit(Dev);
    status = VL53L1_StaticInit(Dev);
    /* VL53L1_SetPresetMode function is mandatory to be called even if default PresetMode is the VL53L1_PRESETMODE_RANGING */
    status = VL53L1_SetPresetMode(Dev, VL53L1_PRESETMODE_RANGING);
    //    status = VL53L1_SetDistanceMode(Dev, VL53L1_DISTANCEMODE_SHORT);
    //    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(Dev, 60000);
    status = VL53L1_StartMeasurement(Dev);
    if(status){
      sprintf(buff_vis,"VL53L1_StartMeasurement failed: error = %d \n", status);
      Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;
      estado = 106;
/     while(1);
    }
```

Ilustración 47. Inicialización sensor

En el modo de funcionamiento en el que trabaja, como se ha explicado antes, es capaz de detectar varios objetos a diferentes distancias y así, para este trabajo, se utiliza la distancia menor donde se encuentre el objeto.

```
Objeto numero 1 a D=154 mm
Objeto numero 1 a D=497 mm
Objeto numero 1 a D=510 mm
Objeto numero 1 a D=521 mm
Objeto numero 1 a D=506 mm, Objeto numero 2 a D=2192 mm, Objeto numero 3 a D=3060 mm, Objeto numero 4 a D=3791 mm
Objeto numero 1 a D=514 mm
Objeto numero 1 a D=515 mm, Objeto numero 2 a D=2008 mm, Objeto numero 3 a D=3055 mm, Objeto numero 4 a D=3850 mm
Objeto numero 1 a D=521 mm
Objeto numero 1 a D=516 mm, Objeto numero 2 a D=1389 mm, Objeto numero 3 a D=4071 mm
Objeto numero 1 a D=374 mm
```

Ilustración 48. Ejemplo detección de varios objetos

Este es un ejemplo de que el sensor ToF puede detectar varios objetos en el modo de funcionamiento en el que se está trabajando.

Las conexiones de este sensor son la alimentación a 3.3 V, tierra, SDA, SCL y INT. El pin interrupción es un GPIO de salida (output) que funciona de interrupción en el que cada vez que emite señal empieza a hacer el código de medición.

6.6 Motor paso a paso.

Un motor paso a paso es un motor de corriente continua en el cual el giro del motor se divide en un número de pasos determinados (dependiendo de la construcción del motor). El motor paso a paso usado en el proyecto tiene 200 divisiones de 1, 8º.

Ilustración 49. Motor paso a paso empleado

El funcionamiento de este tipo de motores es similar al de un motor de corriente continua como los utilizados en las ruedas en este trabajo, pero en este caso se produce la magnetización de una serie de bobinas de forma consecutiva y contiguas, provocando un movimiento del rotor tan solo desde la bobina que está magnetizada

hasta la siguiente bobina que será magnetizada, de este modo se crea el efecto de un movimiento de paso a paso.

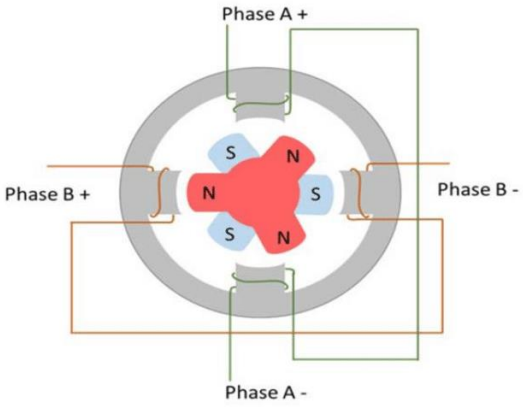


Ilustración 50. Esquema interno motor paso a paso

Estos motores además tienen la posibilidad de trabajar en modo paso completo (pasos de 1,8º) o medio paso (pasos de 0,9º) haciendo que las 2 bobinas se queden magnetizadas, quedándose el rotor a mitad de paso entre las 2 bobinas. Para el objetivo de este proyecto no interesa realizar medios pasos pues 200 series es más que suficiente para poder realizar las medidas necesarias con el sensor ToF.

6.7 Modulo radio Telit LE50 – 868

Ilustración 51. Telit LE50-868

Al tratarse de un robot móvil la problemática dada de la comunicación de este con el pc desde donde se dan las órdenes y monitoriza los datos obtenidos del sensor y el encoder del robot se soluciona con la comunicación serie a través de sistema de radio con 2 módulos, uno en el robot conectado por USART al microcontrolador y otro conectado al ordenador por USB. Estos módulos de radio funcionan en el modo "transparente" que hace comunicación limpia, de un modo identico al que haría una comunicación por cable.

Este modo de funcionamiento se establece desde los comandos propios del sistema del módulo de radio. Este ajuste se establece de la siguiente forma:

1. Se conecta el módulo mediante los cables dupont al adaptador serie FTDI FT232RL y este, a través de un cable USB se conecta al PC. Las conexiones del módulo son:
    - GND
    - VCC
    - TX
    - RX
2. Utilizando el programa termite como interfaz de la comunicación serie entre el módulo de radio y el PC se escriben en primer lugar el comando '+++' y después el comando 'AT/S' para ajustar los parámetros requeridos.
3. Existen una serie de parámetros, pero no hay necesidad de tocar todos ellos para hacer el uso deseado. El primero que se quiere modificar es el 220, que mediante el comando 'ATS220=1' se establece el modo de funcionamiento transparente, tal y como se ha explicado anteriormente.
4. A continuación, han de modificar los valores de los parámetros 252 y 256. Estos 2 parámetros pueden tener un valor cualquiera asegurándose que en el otro módulo de radio son los valores contrarios. En este caso en un módulo se han puesto los valores 252 = 2 y 256 = 5 y en el otro sensor se han puesto 252 = 5 y 256 = 2.
5. Por último, el parámetro que se modifica es el 255, que se pone con un valor de 129.
6. Tras ajustar los parámetros necesarios se envía el comando 'ATO' para salir del modo de edición de parámetros y ponerse en modo ejecución

Estos pasos se han hecho tanto en el módulo que se encuentra en el robot como en el que está en el PC. Tras esto ya se puede utilizar como vía de comunicación del robot y el PC mediante protocolo USART.

6.8 Microcontrolador ST Núcleo-L476RG

En un proyecto la parte más importante es el sistema embebido con el que se trabaja. Un sistema embebido es un sistema de computación en el que se carga un programa que se realizará de forma repetida (en bucle) y cuyos componentes se encuentra integrados en una placa base. Dentro de esta placa base se encuentra el microcontrolador (o microprocesador MPU), que es donde se hacen los procesos en tiempo real. El microcontrolador está compuesto por los siguientes elementos:

- CPU (Unidad central de procesamiento).
- ALU (unidad Aritmética Lógica): Es la encargada de las operaciones lógicas.
- Memoria RAM: Encargada de la lectura y escritura de datos.
- Memoria ROM: Memoria dedicada al almacenamiento. Aquí es donde se almacena el programa a ejecutar. La memoria FLASH es de este tipo.
- Periféricos: Son circuitos internos digitales cuya función la interacción con el mundo exterior. Son capaces de leer lecturas de sensores analógicos, realizar comunicación con terminales digitales (con puertos seriales realiza comunicaciones I"C, SPI, USART…) o convertir señales analógicas en señales digitales.

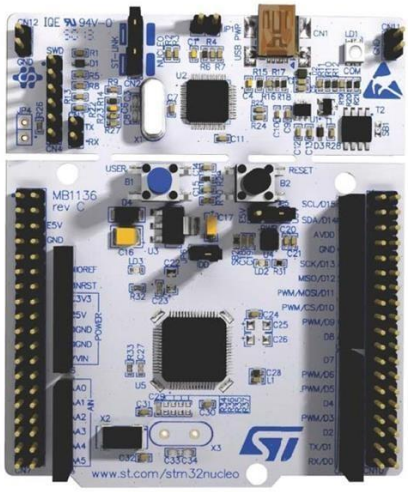


Ilustración 52. . Placa de STM32 núcleo LG76RG

Una vez que están todos los elementos electrónicos definidos queda seleccionar el microcontrolador. Se ha seleccionado el microcontrolador STM32 núcleo LG76RG por su compatibilidad con el sensor VL53L1X seleccionado. Al ser componentes del mismo fabricante tienen compatibilidad gracias a la API dada por el fabricante para este sensor.

Esta es una placa de desarrollo con un microcontrolador ARM Cortex-M4 de 32 bits a 80 MHz con conectores tanto Arduino (aunque en este proyecto no son necesarios) como ST Morpho, con 18 interfaces de comunicación serie (en el proyecto solo son necesarios 2. USART e I2C) y depurador/programador ST-LINK.

El pinout del microcontrolador en cuestión es el siguiente:

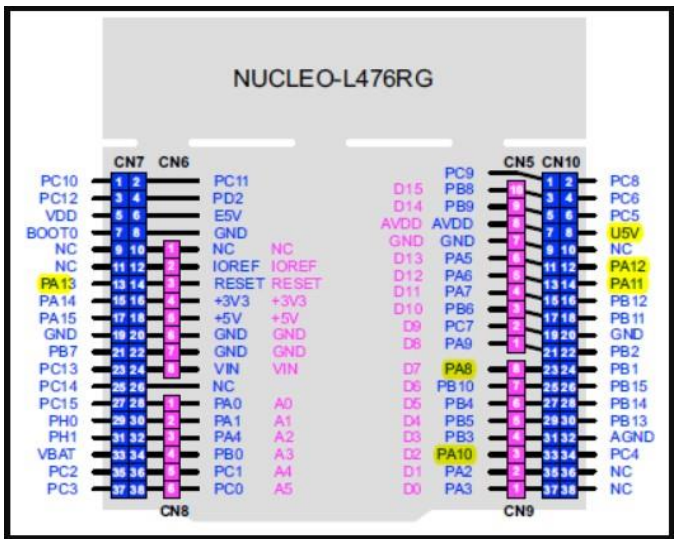


Ilustración 53. Pinout placa STML476RG

6.9 Batería de Litio-Polimero (LiPo) 11V

Esta es una batería recargable compuesta por 3 celdas de 3,7 Voltios cada una, lo que suman 11,1 Voltios totales. Tienen la capacidad de ofrecer una alta tasa de descarga, pudiendo variar entre 1 y 25 A.

Las baterías de Litio-Polimero tienen aplicación dentro del mundo de la robótica porque pueden ofrecer tasas de descarga superiores a 1 A y por su bajo peso en proporción a su capacidad.

Esta batería se instala con el único objetivo de alimentar los motores de corriente continua situados en las ruedas porque, en el caso de que se quisiesen alimentar desde las salidas de tensión que ofrece el propio microcontrolador, en el momento en el que se utilicen los motores de corriente continua y el motor paso a paso a la vez se producen bajadas de tensión en las que se ve afectada, de forma puntual, la

velocidad de giro de las ruedas, lo que provoca algunos problemas en el funcionamiento.

El positivo y el negativo que salen de esta batería se colocan directamente en el controlador L298N de los motores de corriente continua.

Ilustración 54. Batería Litio Polímero

6.10 Batería de ion-litio 5V-2A

Batería que alimenta el microcontrolador y lo que se alimenta desde el microcontrolador. Es una batería que da una salida de 2.1 A y 5V con una capacidad de 5.000 mAh.

7 DISTRIBUCIÓN DE LOS ELEMENTOS EN EL CARRO

Dado que se tratan de varios elementos electrónicos los que se van a utilizar se ha hecho un estudio de la disposición que han detener dentro de las 2 superficies que se disponen.

Este trabajo es beneficioso para algunos aspectos como obtener la máxima claridad en el entramado de cables que se genera al conectar tantos dispositivos, para que no se creen toques falsos entre componentes que puedan hacer cortocircuitos y para evitar las posibles interferencias en las señales que se envían tanto para comunicaciones como para el control de los motores.

En el piso superior del chasis de plástico, bajo la pieza "cubierta" se sitúa el convertidor DC-DC, el microcontrolador (que queda fijado al chasis con unos tornillos) y el motor paso a paso, el cual tiene pegado a un lateral su controlador DRV8825 y el condensador necesario para su funcionamiento. También en este espacio, pero incrustado en la pared de la pieza impresa en 3d "cubierta", se encuentra el módulo de radio Telit.

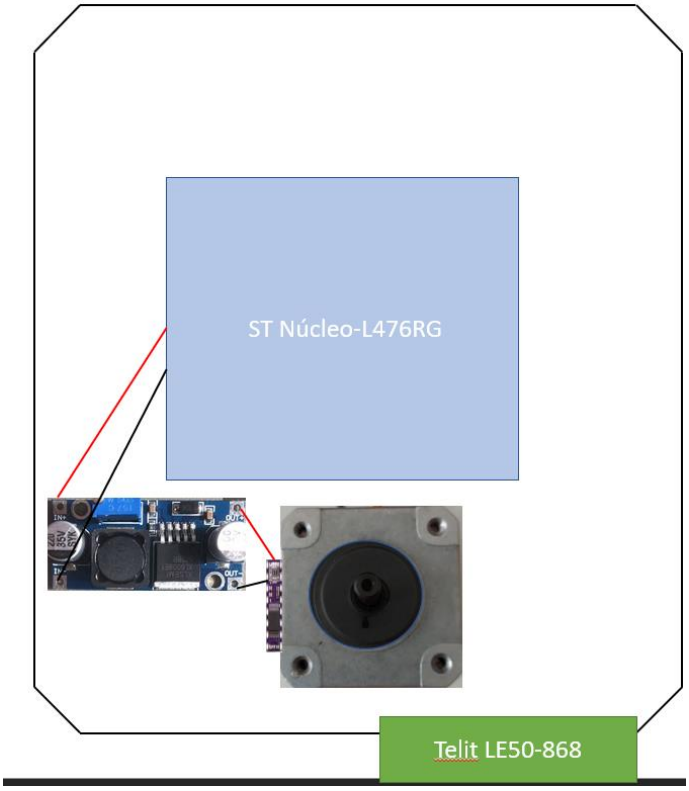


Ilustración 55. Distribución piso superior

Con esta disposición tanto el motor paso a paso como como el módulo de radio se alimentan desde el microcontrolador a 5V. Cabe destacar que el sensor ToF también se alimenta desde el microcontrolador a 3.3 V, pero este sale por el orificio que se encuentra en la pieza "cubierta sensor" para poder salir al exterior y hacer las medidas girando.

En la parte intermedia que se genera entre las 2 tablas de plástico rígido que componen el chasis del vehículo se colocan las 2 baterías que alimentan por separado los motores de corriente continua de las ruedas y el microcontrolador.

La batería de LiPo de 11,1 V se conecta al controlador de los motores de corriente continua que también se encuentra en este espacio. Así se aíslan las conexiones de la batería de LiPo (que tiene una tensión e intensidad muy elevada y puede quemar componentes electrónicos al mínimo contacto) y se hace el recorrido de cables lo más corto posible para las conexiones de los motores.

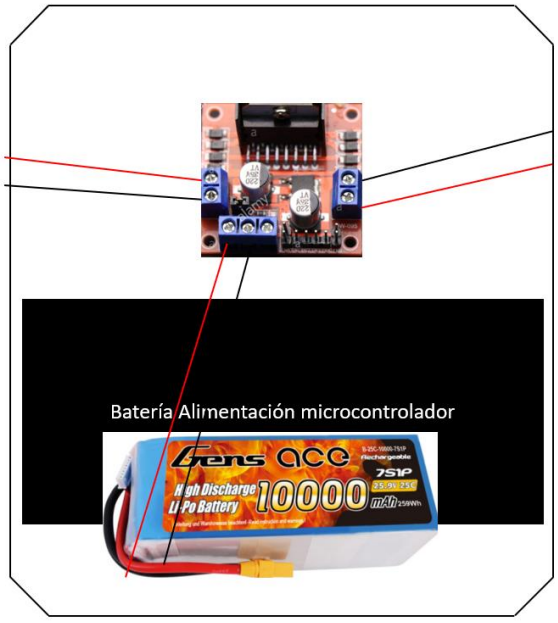


Ilustración 56. Distribución piso inferior

7.1 Pinout Microcontrolador

Para la claridad del diseño del sistema de cables se ha tenido en cuenta con riguroso cuidado la distribución de conexiones de los pines del microcontrolador. La distribución es la siguiente:

	Pinout		
	Motores ruedas	**PIN**	**Label**
Rueda izquierda	ENA	PA0	TIM2_CH1
	IN1	PB4	IN1
	IN2	PB5	IN2
Rueda derecha	IN3	PC0	IN3
	IN4	PC1	IN4
	ENB	PA1	TIM2_CH2
	Encoder rueda derecha (I)	**PIN**	**Label**
	Channel 1	PA6	TIM3_CH1
	Channel 2	PA7	TIM3_CH2
	Encoder rueda izquierda (N)	**PIN**	**Label**
	Channel 1	PB6	TIM4_CH1
	Channel 2	PB7	TIM4_CH2
	Sensor VL53L1X	**PIN**	**Label**
	SCL	SCL	
	SDA	SDA	
	IT	PA4	VL53L1_X_INT
	StepMotor	**PIN**	**Label**
	PWM_StepM	PA8	TIM1_CH1
	DIR_StepM	PA9	DIR_StepMotor
	SleepReset	PC7	
	Modulo de Radio	**PIN**	**Label**
	USART3_TxD	PC4	USART3_TxD
	USART3_RxD	PC5	USART3_RxD

Ilustración 57. Tabla con conexiones microcontrolador

8 ENSAYO CÁLCULO DE PULSOS POR VUELTA DEL ENCODER

Debido a que no se tiene ningún dato del motor en el aspecto del encoder más allá de las conexiones de este, es necesario hacer un ensayo para conocer esta información tan básica del encoder.

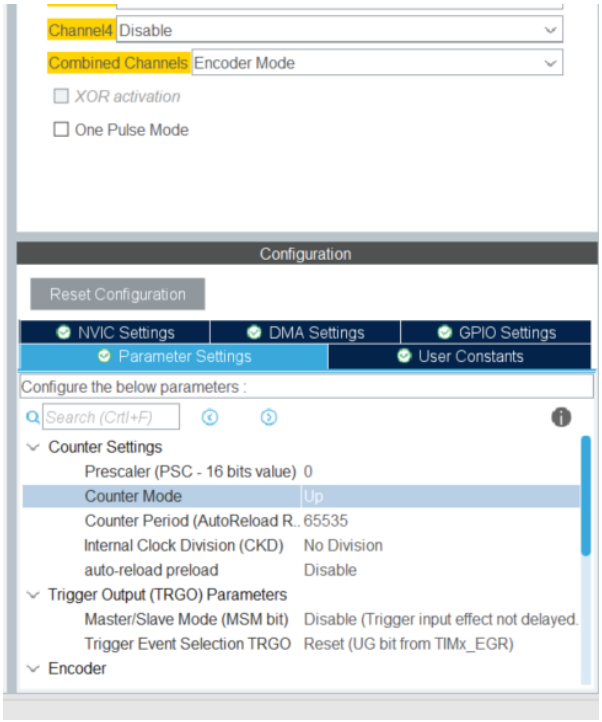


Ilustración 58. Ajustes del timer del encoder

Con los parámetros indicados en la imagen para el timer del encoder, el objetivo del ensayo es contar el número de pulsos que emite el encoder en una dirección en una vuelta de la salida mecánica del motor CC, es decir, una vuelta de la rueda.

Ilustración 59. Sensor de efecto hall

Para conseguir esto de una forma precisa se hace uso de un sensor de efecto hall, que detectará cuando pasa el imán que se ha situado en la rueda y con el cual se conocerá cuando se da una vuelta completa de esta.

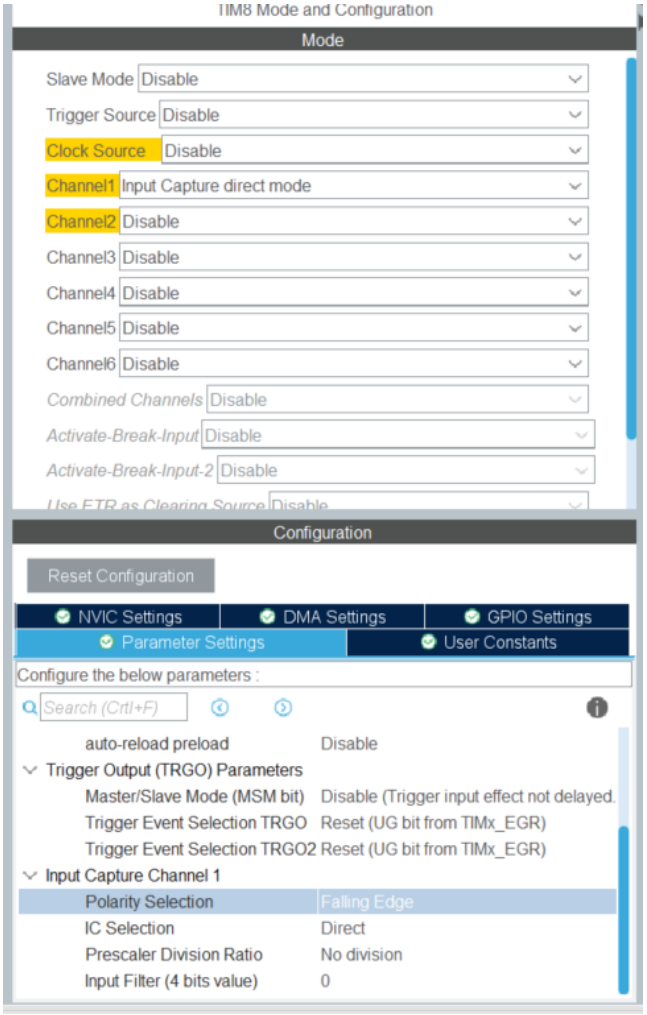


Ilustración 60. Ajustes del timer8 para el experimento

Se activa un *timer*, en este caso el *timer* 8, con interrupciones y el canal 1 en modo *input capture mode*, donde cada vez que el sensor de efecto hall (en flanco de bajada) detecte el imán, se active la interrupción, que guarde el valor que ha recogido el encoder y aumente el número de vueltas.

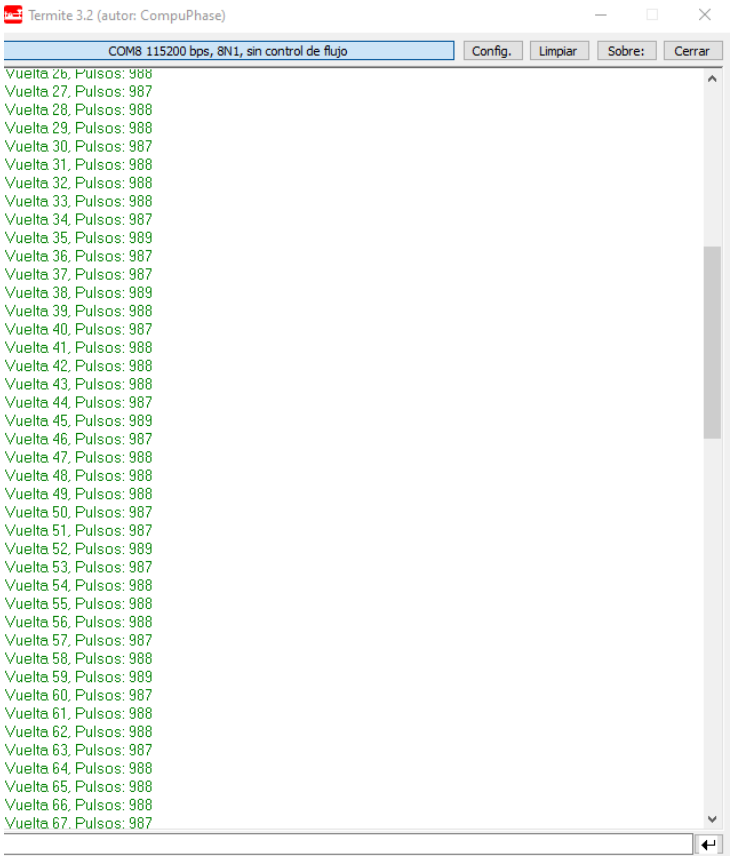


Ilustración 61. Muestra de termite de los pulsos en cada vuelta

Tras 100 vueltas de rueda y haciendo una media de los valores que se obtienen, se puede afirmar que por cada vuelta que da la rueda el encoder envía 988 pulsos. Esta información se trata de la siguiente forma.

Pulsos	Vuelta Rueda	Grados / Vuelta	Pulsos encoder / Grado	Grados /Pulsos Encoder	Grados (Rad)/Pulso Encoder
988	1	360	2,744444444	0,36437247	0,006356275

Con esta información podemos saber cuánto ha girado contando los pulsos que ha enviado cada encoder y poder así actualizar la posición del robot.

El ensayo se repite para el sensor de la otra rueda, en la que se obtienen resultados idénticos.

9 CONFIGURACIÓN DE TIMERS DEL MICROCONTROLADOR

En el funcionamiento del programa se hacen uso de varios temporizadores en diferentes modos de trabajo. Un timer es una herramienta del microcontrolador que sirve para contar, ya sea tiempo, pulsos o interrupciones. El microcontrolador cuenta con timers de 16 bits (son capaces de contar hasta 65.536) y tiene un solo timer de 32 bits (capaz de contar hasta 4.294.967.296). Las funciones que tienen los timers empleados son las siguientes:

- Reloj de tiempo real RTC.
- PWM del motor paso a paso.
- PWM de los motores de corriente continua.
- Encoders.
- Timer para las interrupciones.

9.1 Ajuste de parámetros del reloj de tiempo real RTC

Para poder calcular las velocidades de las ruedas se ha optado por la opción de mantener una cuenta del tiempo total que funciona el robot, de modo que se puede tener referencia del momento en el que se toma la velocidad de los motores de las ruedas. También se usará más adelante para la aplicación del control de velocidad sobre las ruedas.

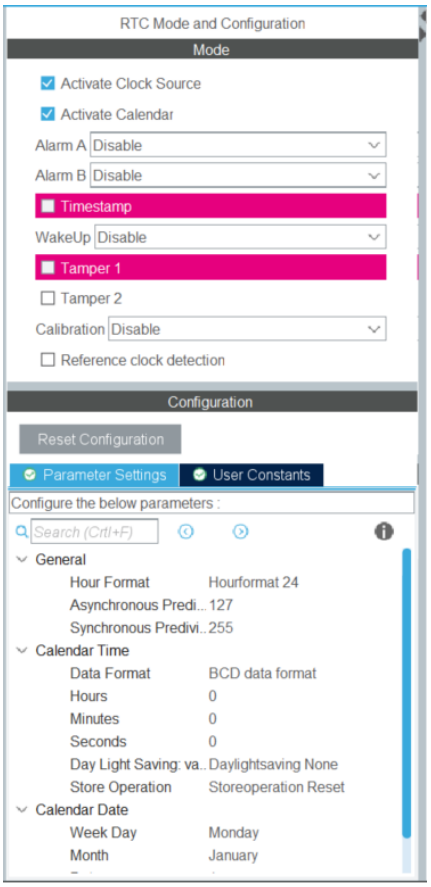


Ilustración 62. Parámetros RTC

En el código se escribe una función en la que se obtiene del registro *sTime* los valores de las horas, minutos y segundos que lleva en funcionamiento el programa.

La precisión de las medidas a la hora de medir velocidades tiene que ser bastante fina, por lo que el tiempo que se quiere obtener ha de ser en milisegundos.

Para obtener los milisegundos se recurre al registro *sTime.Subseconds* que devuelve los milisegundos en función en valor que se ha dado en el parámetro *Asynchronus Predivisor* por lo que hay que hacer un cambio de escala mediante una ecuación matemática y utilizando el registro de tiempo sTime.SecondFraction. Para obtener los milisegundos totales (que es lo que se usará en el cálculo de las velocidades) se hace el cambio de unidades y la suma total.

La función creada devuelve el valor de los milisegundos.

9.2 PWM del motor paso a paso

Como se ha explicado anteriormente el motor paso a paso se controla mediante una señal PWM gracias al controlador DRV8825.

Para poder trabajar de esta forma en primer lugar hay que crear el temporizador con el programa STM32CubeMX. Este será el temporizador TIM1 que se selecciona como "PWM Generator CH1" en el canal 1 del temporizador. Esto genera un pin en el microcontrolador por donde saldrá la onda cuadrada que se busca para el control de los motores.

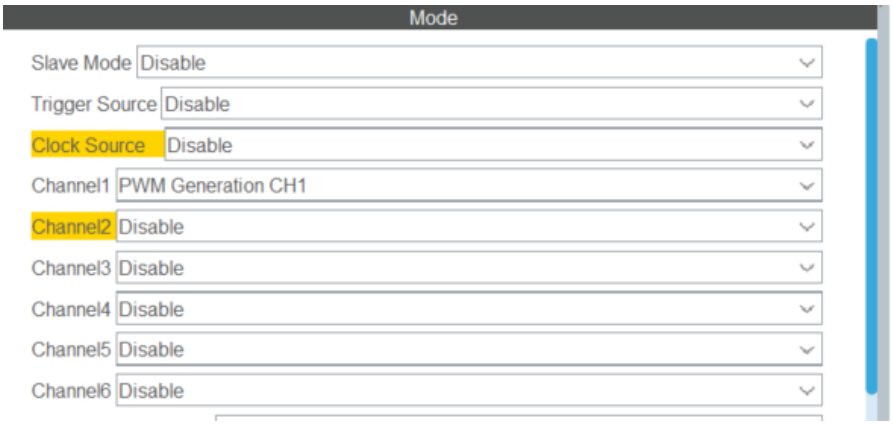


Ilustración 63. Ajustes timer PWM motor paso a paso

Es importante, del modo que se ha diseñado el código de control del step motor, activar las interrupciones de este timer del modo "*capture compare interrupt*" con el propósito de crear una interrupción cada pulso enviado del PWM, así se pueden contar los pulsos que se han enviado.

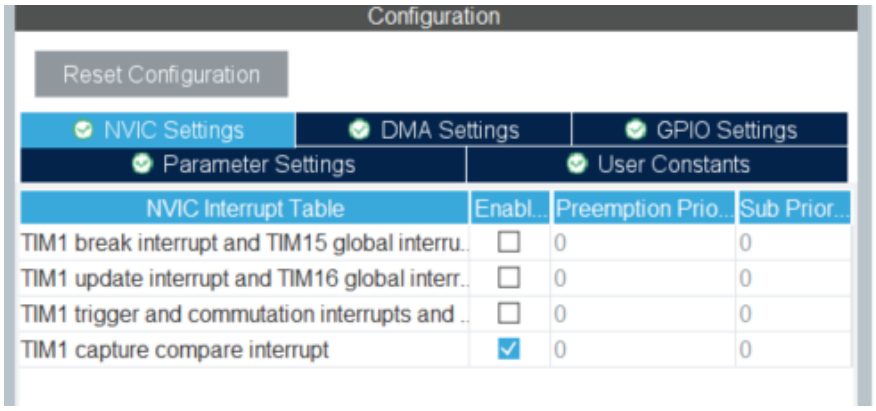


Ilustración 64. Ajustes timer PWM motor paso a paso

Una vez hecho esto se pueden ajustar los parámetros del prescaler.

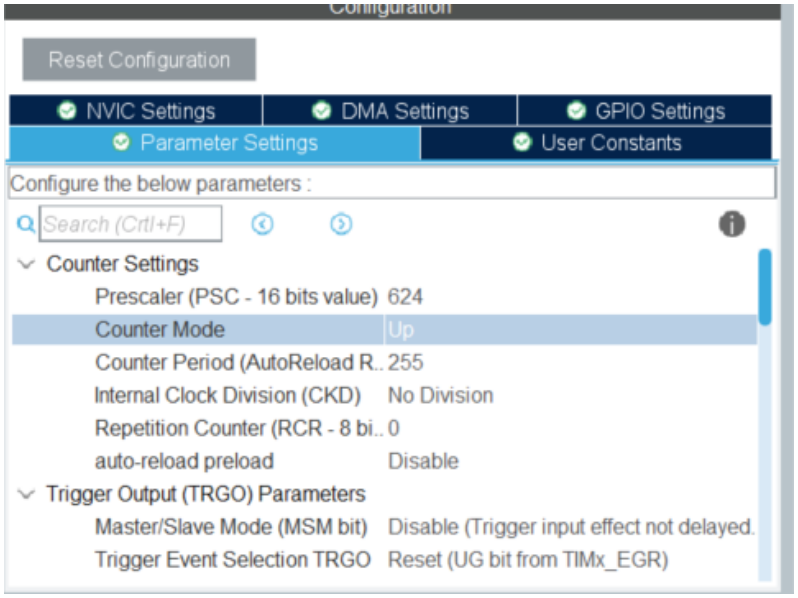


Ilustración 65. Ajustes PWM motor paso a paso

Después de una serie de pruebas se hace la comprobación de que el motor paso a paso seleccionado funciona a una frecuencia de 500 hz por lo que se ajustan los parámetros para esta frecuencia de funcionamiento utilizando la siguiente expresión:

$$f_{pwm} = \frac{f_{clock}}{(Prescaler + 1) \cdot (Autoreload + 1)}$$

Sabiendo que la velocidad establecida para funcionamiento del microcontrolador es de 80MHZ y dando un valor de autoreload de 255 (realmente es 256), para un PWM de 500 Hz se obtiene un valor para el prescaler de 624.

Con estos ajustes se puede crear una función que indicando los pulsos que se quieren dar mueve ese número de pasos en la dirección indicada.

El objetivo es poder controlar el motor enviando un valor de PWM con valor máximo el ajuste establecido de autorrecarga y mínimo cero.

9.3 PWM motores ruedas

Los motores de las ruedas, como se ha indicado previamente, tienen un control de velocidad mediante PWM. Este PWM se genera mediante el timer 2.

El valor que se da al preescaler del timer es 80 para obtener la siguiente velocidad de reloj.

$$Velocidad\ timer = \frac{80\ MHz}{80} = 1\ MHz$$

Tras hacer unas cuantas pruebas y con el objetivo de conseguir mayor precisión en la velocidad aplicada en las ruedas (en el apartado del control PID es un aspecto muy favorable) se ha dado un valor de AutoReload de 10.000, lo que significa que el valor de PWM aplicado en el motor será sobre este valor de AutoReload y que se trabaja con una onda cuadrada de 1 longitud de onda 10 milisegundos.

Ilustración 66. Ajustes parámetros del timer del PWM de las ruedas

Este mismo timer es el que se utiliza para generar las 2 ondas cuadradas para los 2 motores saliendo cada una por un canal de timer.

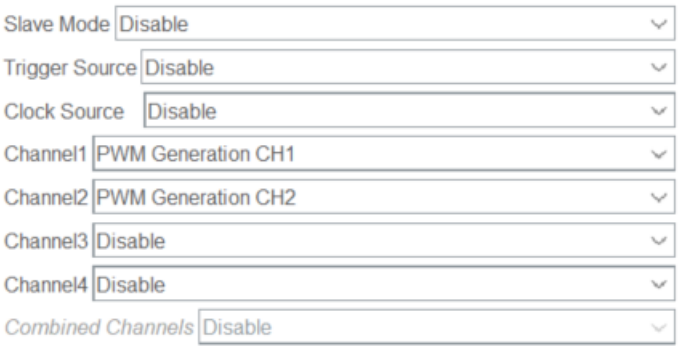


Ilustración 67. Canales de generaciónd de la señal PWM

9.4 Timer de encoders

Para obtener los datos del encoder se ha de configurar el timer de una forma determinada. Al trabajar con 2 encoders (uno en cada rueda) se han de ajustar los parámetros de 2 *timers* de forma idéntica. Lo que se explica a continuación se ha aplicado a los *timers* 3 y 4.

El ajuste es bastante sencillo. Se configura el encoder en modo Encoder Mode en la sección de combined channels. Esto hace que se activen 2 pines (uno para cada canal del encoder).

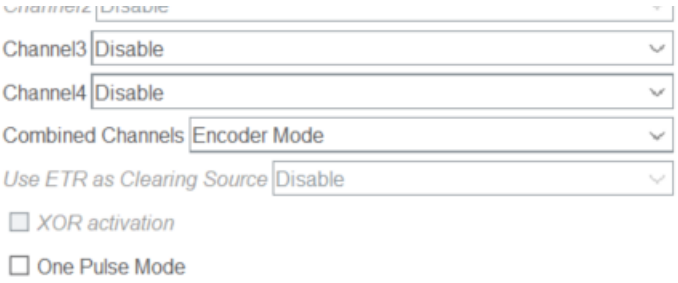


Ilustración 68.Ajuste encoder mode

En cuanto a los parámetos no se toca nada, simplemente se pone en el registro de autoreload el máximo posible (65535 pulsos) que pueda contar el timer de 16 bits.

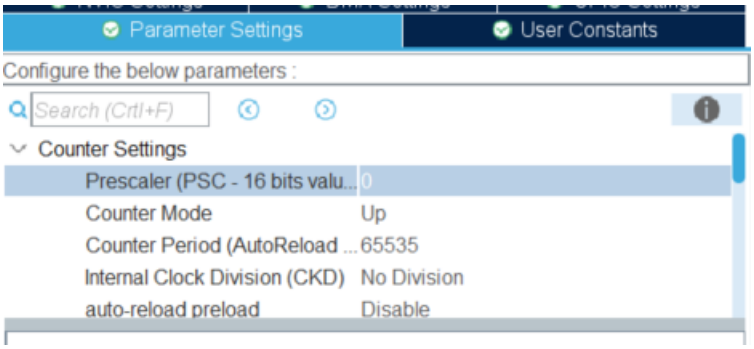


Ilustración 69. Ajustes parámetros del timer del encoder

9.5 Timer interrupciones odometría

Este timer queda explicado en el siguiente apartado donde se muestra la configuración y diseño de la odometría completa.

10 ODOMETRÍA

La odometría se conoce como el estudio que estima la posición de vehículos con ruedas mediante el uso de la rotación de las ruedas, que permite actualizar la posición del vehículo a lo largo de un tiempo de ejecución del movimiento.

Esto no sirve para determinar con precisión la posición del robot en largas distancias, pues se basa en la suposición de que las revoluciones de las ruedas del robot se transforman en desplazamiento lineal respecto al suelo. Es un método que puede dar posiciones bastante precisas en periodos cortos de tiempo pero que a lo largo del tiempo acumula errores y la posición obtenida puede ser bastante poco precisa.

A la hora de realizar este método han de realizarse una serie de pasos de carácter necesarios que se describen a continuación.

10.1 Interrupciones

A la hora de conocer la posición del robot mediante odometría, se ha de hacer los cálculos lo más frecuentemente posible y así tener en cuenta todos los posibles cambios que se realizan a lo largo de la trayectoria realizada.

Para conseguir esto se hace uso de las interrupciones del código, que se realizarán repetidamente con una frecuencia a determinar en el cube.MX.

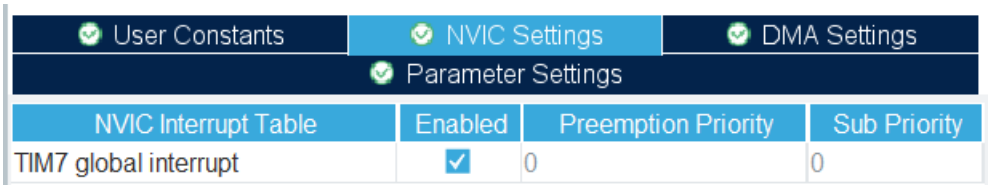


Ilustración 70. Ajuste timer interrupciones

Estas interrupciones ejecutarán un código simple que servirá para hacer los cálculos de las velocidades y la actualización de la posición del robot.

Para ejecutar esta interrupción con una frecuencia que pueda actualizar la posición lo más frecuente posible se hacen los cálculos para poder ejecutarla cada 25 milisegundos.

$$tiempo = \frac{Autoreload}{\frac{Velocidad\ del\ reloj}{Prescaler}}$$

$$\frac{50}{\frac{80.000.000\ Hz}{40.000}} = 0{,}025\ s = 25\ ms$$

INTERRUPCION ODOMETRÍA		
Vel. Microcontrolador	80	MHz
Vel. Microcontrolador (en Hz)	80000000	Hz
Prescaler	40000	
Reloj	2000	cont/s
Autoreload	50	
Tiempo	0,025	s
Tiempo	25	ms

Así con estos ajustes de los parámetros de *prescaler* y de *autoreload* se obtiene la ejecución de las interrupciones cada 25 milisegundos.

10.2 Cálculo de las velocidades de las ruedas

Para calcular las velocidades en las ruedas hay que basarse en 2 variables que son la cuenta de los encoders incorporados en las ruedas y el tiempo que se ha tardado en contar esos pulsos del encoder.

Para hacer esto, se crea dentro de cada interrupción la variable del valor que se obtiene el del encoder y la variable del encoder de la interrupción anterior. Realizando una resta se consigue de resultado el número de pulsos que ha enviado el encoder desde que se ha ejecutado la última interrupción.

Algo parecido se hace con el tiempo. Se obtiene el valor del tiempo en milisegundos en cada interrupción y se resta con el valor del tiempo en la interrupción anterior, obteniendo el tiempo en el que se miden los pulsos del encoder.

Para calcular las velocidades de cada rueda simplemente se aplica la siguiente ecuación

$$W_{rueda} = \frac{(Encoder_{actual} - Encoder_{anterior}) \cdot K_{pul-rad}}{(Tiempo_{actual} - Tiempo_{anterior})}$$

Donde el valor de $K_{pul\text{-}rad}$ es el factor de conversión de pulsos a radianes calculado previamente en el ensayo del apartado 4, que es 0,006356275 para convertir los pulsos del encoder en radianes.

De este modo se obtienen las velocidades de las ruedas en radianes por milisegundos (porque el tiempo que se utiliza está en milisegundos) cada 25 ms.

Pulsos	Vuelta Rueda	Grados / Vuelta	Pulsos encoder / Grado	Grados /Pulsos Encoder	Grados (Rad)/Pulso Encoder
988	1	360	2,744444444	0,36437247	0,006356275

Ilustración 71. Datos calculados encoder

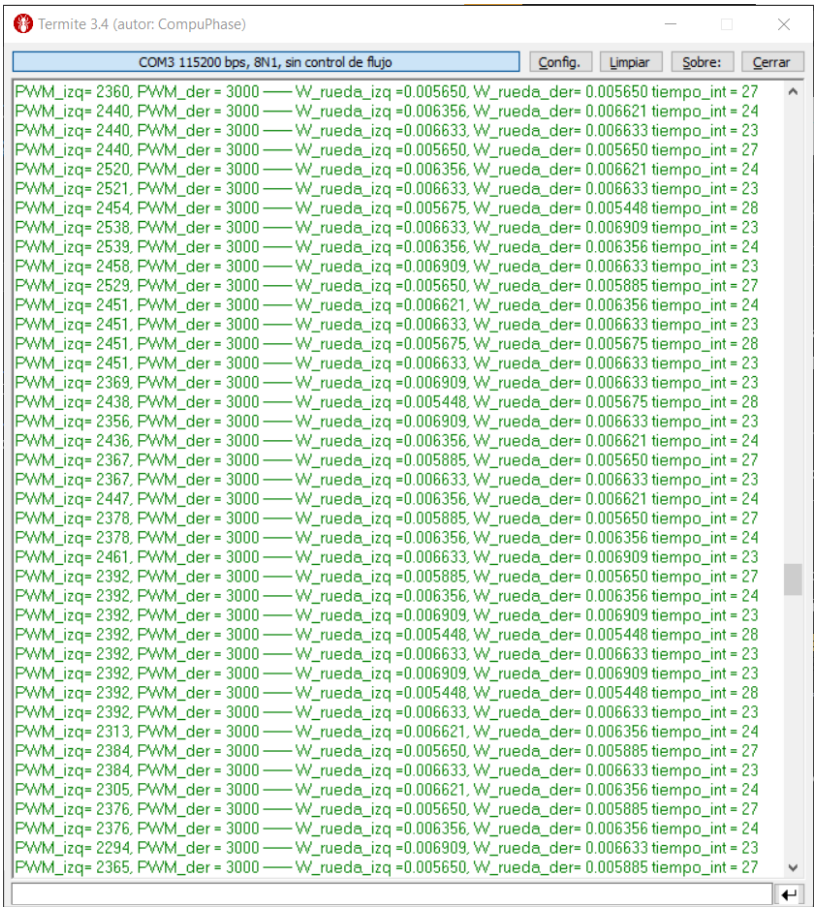


Ilustración 72. Muestra de velocidades angulares calculadas en las interrupciones en termite

10.3 Modelado cinemático directo

A partir de las velocidades angulares de cada una de las ruedas se puede hacer una transformación a la velocidad lineal de las ruedas y, por lo tanto, la velocidad lineal de un punto medio que se considera el centro del robot. Esta velocidad del vehículo se puede trasladar a unas posiciones X e Y dentro de un espacio bidimensional y aproximar un ángulo de giro del robot respecto a un punto inicial.

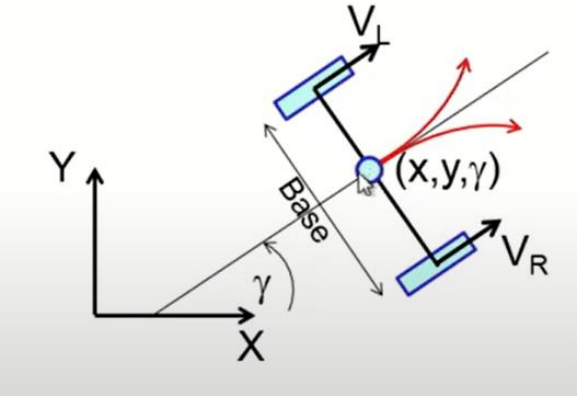


Ilustración 73. Esquema cartesiano de odometría

Se siguen unos pequeños pasos para poder actualizar la posición del robot que se componen de unas pequeñas ecuaciones cinemáticas que se describen a continuación.

10.3.1 Velocidades lineales de las ruedas:

Conociendo el radio de las ruedas que se utilizan en el robot, se puede hacer la conversión de la velocidad angular a la velocidad lineal de las ruedas siguiendo las siguientes ecuaciones:

$$Vel_{RuedDer} = R_{der} \cdot \omega_{der}$$
$$Vel_{RuedIzq} = R_{Izq} \cdot \omega_{Izq}$$

Las ruedas seleccionadas para este proyecto tienen un diámetro de 68 mm, lo que son 34mm de radio de cada rueda.

10.3.2 Velocidad lineal instantánea:

El centro de las ruedas se toma como punto de referencia del robot. Es el punto que actualizará las posiciones del robot y, por lo tanto, hay que calcular la velocidad lineal de este punto, que será la media (al ser el punto medio de las 2 ruedas) de las 2 velocidades lineal de las ruedas del vehículo.

$$Vel_{inst} = \frac{\left(Vel_{RuedDer} + Vel_{RuedIzq}\right)}{2}$$

10.3.3 Velocidad angular del vehículo:

A partir de las velocidades lineales de las ruedas se puede calcular la velocidad angular del vehículo respecto a si mismo, es decir, la velocidad de rotación que se

reproduce. En caso de que la velocidad de las ruedas sea la misma se producirá una velocidad angular nula. Esta velocidad angular aparece cuando hay una diferencia entre las velocidades lineales de las ruedas. La ecuación que representa esta velocidad angular se puede expresar de la siguiente forma:

$$\bar{\omega} = \frac{\left(Vel_{RuedDer} - Vel_{RuedIzq}\right)}{Base}$$

Entendiendo la base como la distancia que se encuentra entre las 2 ruedas del vehículo. En esta aplicación la base del vehículo es de 190 mm.

Una vez que tenemos la velocidad lineal del vehículo y la velocidad angular de rotación de este, se puede actualizar la localización en cuanto a coordenadas X, Y y γ

10.3.4 Actualización de la posición

Se hace mediante unas ecuaciones que cinemáticas, que son las siguientes:

$$\gamma' = \gamma + \bar{\omega} \cdot t$$
$$x' = Vel_{inst} \cdot \cos(\gamma) \cdot t$$
$$y' = Vel_{inst} \cdot \sin(\gamma) \cdot t$$

Por lo tanto, la actualización de las posiciones se produce en las interrupciones del timer 7 del microcontrolador que ocurre cada 25 ms.

Ilustración 74. Posiciones actualizadas odometría.

11 CALIBRACIÓN DE LOS MOTORES

En primer lugar, se conectan los 2 motores se conectan al controlador L298N ya descrito anteriormente y se conectan los pines al microcontrolador ST.

Aplicando desde el código un mismo valor de PWM a ambos motores se aprecia que no siguen la misma velocidad, por lo que hay que calibrarlos antes de poder trabajar con ellos.

Para hacer el movimiento del vehículo hacia adelante o para atrás el motor de la derecha ha de girar en sentido antihorario pero el de la izquierda ha de girar en sentido horario. Este hecho hace que no giren a la misma velocidad a un mismo PWM por razones mecánicas. Para solucionar esto se recurre a un método de control que se expone a continuación.

Con el objetivo final de que las 2 ruedas giren a la misma velocidad se tomará la rueda más lenta (en este caso es la rueda derecha) como referencia a la que tiene que alcanzar la rueda izquierda. La rueda derecha tendrá una velocidad (más o menos) fija porque se le asigna un valor de PWM que no variará en ningún momento del proceso de control.

Ahora, el problema es la rueda izquierda, que es la que será la salida del sistema en el método de control. Tras hacer varias pruebas se ha optado por la implantación de un control PID, pues se valora en este tipo de aplicaciones que el control no tenga una oscilación que gire el vehículo de su trayectoria inicial y que tenga un error nulo o, en su defecto, lo más pequeño posible para que no se desvía cuando sigue una trayectoria recta.

En este PID implementado la referencia es la velocidad angular de la rueda derecha, la salida del sistema es la velocidad de la rueda derecha y la acción de control que se aplica al sistema es el PWM de la rueda izquierda.

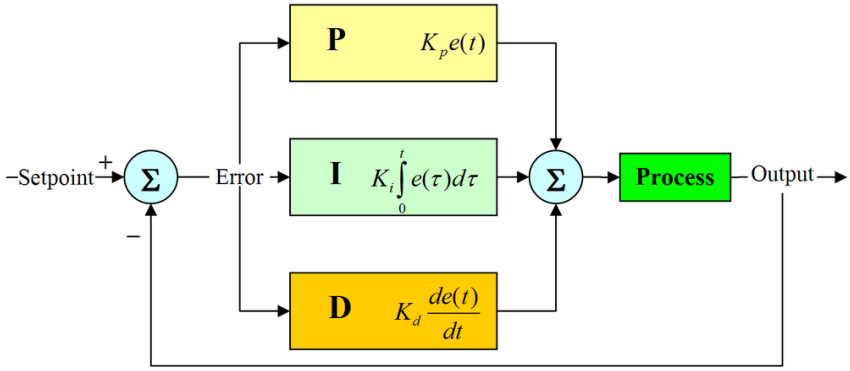


Ilustración 75. Esquema PID

Este diagrama de bloques se ha de implementar en el código del IDE de STM32 programado en lenguaje c. Para calcular las acciones proporcional, derivativa e integral del control se hace mediante el cálculo de las variables de error derivativa y suma integrativa, tal y como se muestra en la siguiente captura del código.

```
//////////////////////  Implatacion del PID para control de la rueda izquierda (esclava) en moviemiento recto //////////////////////////
  if(flag_correccion_pwm == 1){

    error_Vel = W_rueda_der - W_rueda_izq;                                  // Error de velocidades angulares de la rueda derecha y la rueda izquierda. Referencia - salida
    error_Vel_Deriv = (error_Vel - error_Vel_ant)/difer_tiempo;             // Error de velocidades derivadas respecto el tiempo. Necesario para la accion derivativa
    suma_integrativ += (error_Vel * difer_tiempo);                                  // Suma integrativa del error de velocidades. Necesario para la accion integral
    pwmLW = pwmLW + (int)(error_Vel *Kp + (error_Vel_Deriv/1000* Kd)+ (Ki * suma_integrativ));    // Suma de todas las acciones de control del PID
    LeftWheel(1,pwmLW);                                                             // Se aplica la accion de control obtenida

    error_Vel_ant = error_Vel;              // almacena el valor del ultimo error de velocidad para poder hacer la parte derivativa del PID

  }
```

Ilustración 76. Líneas de código PID

Una vez escrito el código del control PID queda ajustar sus constantes con el objetivo de que sea control sin una sobreoscilación importante, que tenga un tiempo de establecimiento lo más rápido posible y un error a la referencia lo más cercano a cero posible.

En primer lugar, se ajusta la acción proporcional K_p para obtener un tiempo de establecimiento rápido, pero sin que sature. Ahora se ajusta la contante derivativa K_d con el objetivo de eliminar la sobreoscilación que se encuentre y por último se fija un valor para la constante de integración K_I para que elimine el error a la referencia acumulado que pueda existir.

Tras periodo de pruebas se establecen los siguientes valores para las constantes de control:

$$K_p = 300000 \quad K_d = 10000 \quad K_I = 50$$

Con estas constantes se obtienen unos resultados de este estilo.

```
PWM_izq= 1675, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004929 error_Vel = 0.000000,
PWM_izq= 1675, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004929 error_Vel = 0.000000,
PWM_izq= 1657, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1657, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1657, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1678, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1678, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1678, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1679, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1679, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1679, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1680, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1680, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1680, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
PWM_izq= 1721, PWM_der = 2500 ——— W_rueda_izq =0.004849, W_rueda_der= 0.004980 error_Vel = 0.000131,
PWM_izq= 1721, PWM_der = 2500 ——— W_rueda_izq =0.004849, W_rueda_der= 0.004980 error_Vel = 0.000131,
PWM_izq= 1721, PWM_der = 2500 ——— W_rueda_izq =0.004849, W_rueda_der= 0.004980 error_Vel = 0.000131,
PWM_izq= 1743, PWM_der = 2500 ——— W_rueda_izq =0.005071, W_rueda_der= 0.005139 error_Vel = 0.000068,
PWM_izq= 1743, PWM_der = 2500 ——— W_rueda_izq =0.005071, W_rueda_der= 0.005139 error_Vel = 0.000068,
PWM_izq= 1743, PWM_der = 2500 ——— W_rueda_izq =0.005071, W_rueda_der= 0.005139 error_Vel = 0.000068,
PWM_izq= 1745, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004929 error_Vel = 0.000000,
PWM_izq= 1745, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004929 error_Vel = 0.000000,
PWM_izq= 1745, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004929 error_Vel = 0.000000,
PWM_izq= 1728, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.004980 error_Vel = -0.000066,
PWM_izq= 1728, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.004980 error_Vel = -0.000066,
PWM_izq= 1728, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.004980 error_Vel = -0.000066,
PWM_izq= 1711, PWM_der = 2500 ——— W_rueda_izq =0.004994, W_rueda_der= 0.004929 error_Vel = -0.000065,
PWM_izq= 1711, PWM_der = 2500 ——— W_rueda_izq =0.004994, W_rueda_der= 0.004929 error_Vel = -0.000065,
PWM_izq= 1711, PWM_der = 2500 ——— W_rueda_izq =0.004994, W_rueda_der= 0.004929 error_Vel = -0.000065,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005059, W_rueda_der= 0.004864 error_Vel = -0.000195,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005059, W_rueda_der= 0.004864 error_Vel = -0.000195,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005059, W_rueda_der= 0.004864 error_Vel = -0.000195,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.005046 error_Vel = 0.000000,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.005046 error_Vel = 0.000000,
PWM_izq= 1654, PWM_der = 2500 ——— W_rueda_izq =0.005046, W_rueda_der= 0.005046 error_Vel = 0.000000,
PWM_izq= 1636, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1636, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1636, PWM_der = 2500 ——— W_rueda_izq =0.004929, W_rueda_der= 0.004864 error_Vel = -0.000065,
PWM_izq= 1656, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004929 error_Vel = 0.000065,
PWM_izq= 1656, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004929 error_Vel = 0.000065,
PWM_izq= 1656, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004929 error_Vel = 0.000065,
PWM_izq= 1676, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1676, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1676, PWM_der = 2500 ——— W_rueda_izq =0.004915, W_rueda_der= 0.004980 error_Vel = 0.000066,
PWM_izq= 1677, PWM_der = 2500 ——— W_rueda_izq =0.004864, W_rueda_der= 0.004864 error_Vel = 0.000000,
```

Ilustración 77. Lecturas de referencias, errores y acciones de control del PID

Cabe destacar que al no disponer de una forma de dibujar los resultados en tiempo real se han ajustado los valores analizando el resultado de diferencia entre la referencia y la salida del sistema (Referencia – velocidad rueda izquierda).

Esto se puede aplicar para cualquier PWM que se aplique de referencia en la rueda derecha. La rueda izquierda alcanzará la velocidad de la de la derecha de modo que el robot sigue una trayectoria recta.

Tras comprobarlo en el robot, se puede ver que se acerca mucho al resultado esperado sin conseguir una oscilación cero, pero es más que suficiente para el objetivo propuesto en el proyecto.

Para conocer cuál es el PWM de la rueda izquierda en el que la velocidad es igual que la velocidad de la rueda derecha con el PWM establecido se hacen pruebas en las que se deja correr el PID a un PWM de la rueda derecha fijo y se obtiene un valor del PWM de la rueda izquierda aproximado.

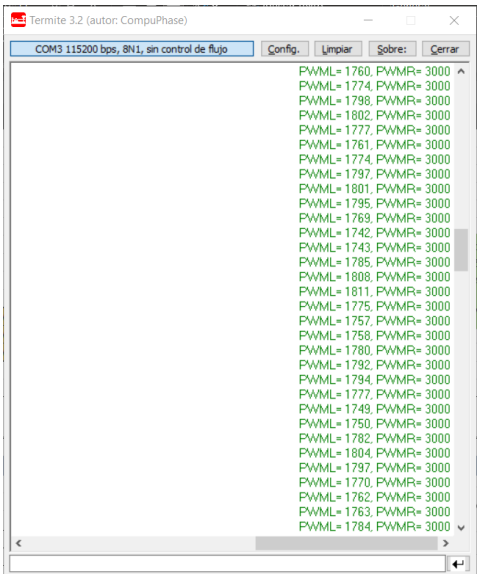


Ilustración 78. Cálculo equivalencia de Velocidades para un PWM

Esto se hace para 3 valores de PWM de la rueda derecha con el objetivo de, más adelante, saber cuáles son las velocidades a las que se tienen que poner las ruedas para hacer los giros en los que se esquivan los objetos que se encuentran con el sensor. Los valores obtenidos son los siguientes:

PWM Rueda derecha	PWM Rueda izquierda
4000	2990
3000	1790
5000	3300

12 FUNCIONES DEL CÓDIGO

En el desarrollo del código se han escrito una serie de funciones que sirven de ayuda tanto para controlar los motores y el sensor como para aplicar acciones predeterminadas que se repetirán a lo largo del proceso y sirven también para conseguir un código en la máquina de estados más limpio y claro.

Aquí se procede a explicar breves cuales son las funciones que se han escrito y su aplicación.

- *LeftWheel(direction, pwm):* Función que sirve para aplicar el PWM indicado a la rueda izquierda. Controla la dirección de giro del motor también aplicando tensión en los pines conectados al L298N.
  - Direction = 1 → movimiento adelante.
  - Direction = 2 → movimiento atrás.
  - Direction = 0 → motor parado.

- *RightWheel(direction, pwm):* Función que sirve para aplicar el PWM indicado a la rueda derecha. Controla la dirección de giro del motor también aplicando tensión en los pines conectados al L298N.
  - Direction = 1 → movimiento adelante.
  - Direction = 2 → movimiento atrás.
  - Direction = 0 → motor parado.

- *RotaRobot(sentido, Gamma_rot):* Función cuya utilidad es rotar el robot desde parado. Según el valor de la variable de entrada *sentido* el robot gira de la siguiente forma:
  - Sentido = 1 → Gira en sentido horario.
  - Sentido = 2 → Gira en sentido antihorario.

  Se le aplica un PWM fijo a cada rueda a través de la función *LeftWheel* y *RightWheel* de modo que el robot gira hasta que el ángulo Gamma en la que se encuentra el robot alcanza el indicado en la variable de entrada. Esto se

calcula mediante odometría. Cuando alcanza el ángulo requerido se para el robot.

- *AvoidObject():* Función que agrupa todo el algoritmo de evitar objetos. Trabaja con las variables de la distancia a la que se ha encontrado el objeto con el sensor ToF y con la zona en la que se ha encontrado el objeto. No tiene variables de entrada porque las variables de la distancia y la zona se han declarado como variables globales y se actualizan antes de aplicar esta función.

  Las zonas en las que se detecta el objeto se definen con el paso del motor paso a paso. Se realizan 10 pasos por vuelta por lo que la división queda como indica la figura a continuación, siendo los números en verde los correspondientes al movimiento horario y los rojos del movimiento antihorario del motor paso a paso.

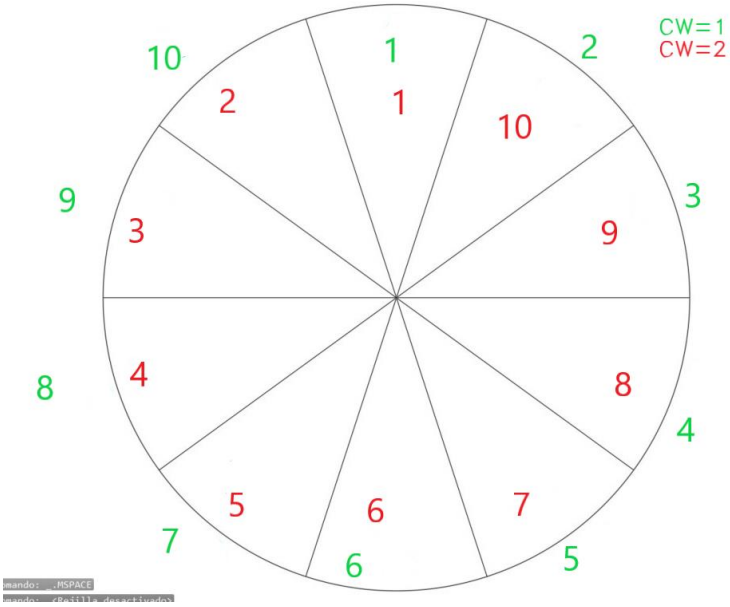


Ilustración 79. Esquema de zonas de detección del sensor

Una vez que se sabe dónde se ha encontrado el objeto se ha de modificar el pwm de las ruedas. Si el objeto se encuentra en la zona de la derecha del robot ha de girar a la izquierda y si se encuentra a la izquierda ha de girar a la derecha.

Si el objeto se encuentra en la parte frontal se compara la medida de las 2 zonas colindantes a la zona 1 y el robot girará hacia donde haya mayor distancia

Si el objeto se encuentra en la parte de atrás del robot (zona 6) se aplica un acelerón a las ruedas para que pueda alejarse rápidamente.
Todos estos cambios en el valor de los PWM de las ruedas se hacen en función de la distancia a la que se haya encontrado el objeto a esquivar, de modo que si el objeto está muy cerca el robot haga un giro muy brusco para evitarlo, pero si el objeto se encuentra muy lejos se hace un giro más suave porque hay tiempo para evitarlo.
Si el robot ha de girar a la derecha aumentará el valor del PWM de la rueda izquierda, como se acaba de explicar, en función de la distancia a la que se haya encontrado el objeto y así poder hacer el giro. Lo contrario pasa para el giro a la izquierda, donde se aumentará el PWM de la rueda izquierda.
Para calcular cuánto ha de modificarse el PWM se ha hecho uso de la herramienta de Excel de aproximación de mínimos cuadrados, en donde se ha asignado un porcentaje de aumento del valor de velocidad nominal del PWM de cada rueda según la distancia que haya dado la medición del sensor. En el código se aplica la función aproximada de mínimos cuadrados obtenida.

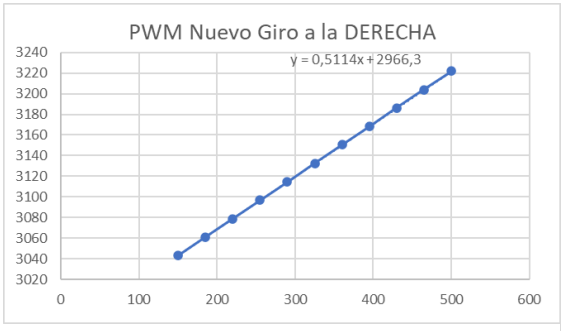
ZONAS 2 Y 10

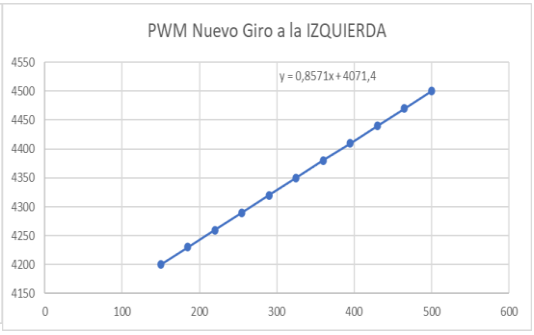
RUEDA IZQUIERDA

	distancia {mm}	PWM Nuevo
170%	150	3043
171%	185	3061
172%	220	3079
173%	255	3097
174%	290	3115
175%	325	3133
176%	360	3150
177%	395	3168
178%	430	3186
179%	465	3204
180%	500	3222

RUEDA DERECHA

	distancia {mm}	PWM Nuevo
140%	150	4200
141%	185	4230
142%	220	4260
143%	255	4290
144%	290	4320
145%	325	4350
146%	360	4380
147%	395	4410
148%	430	4440
149%	465	4470
150%	500	4500





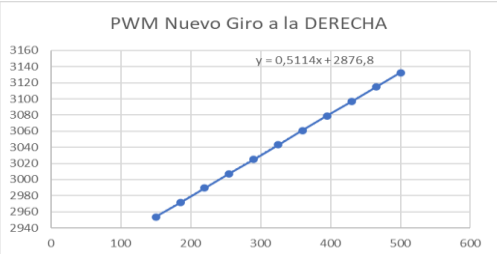
ZONAS 3, 4, 8 y 9

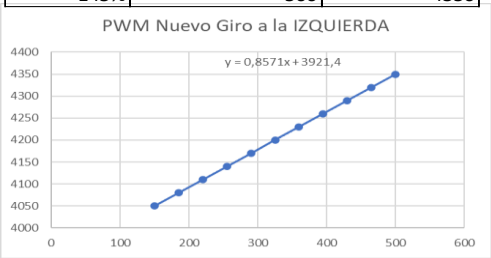
RUEDA IZQUIERDA

	distancia {mm}	PWM Nuevo
165%	150	2954
166%	185	2971
167%	220	2989
168%	255	3007
169%	290	3025
170%	325	3043
171%	360	3061
172%	395	3079
173%	430	3097
174%	465	3115
175%	500	3133

RUEDA DERECHA

	distancia {mm}	PWM Nuevo
135%	150	4050
136%	185	4080
137%	220	4110
138%	255	4140
139%	290	4170
140%	325	4200
141%	360	4230
142%	395	4260
143%	430	4290
144%	465	4320
145%	500	4350





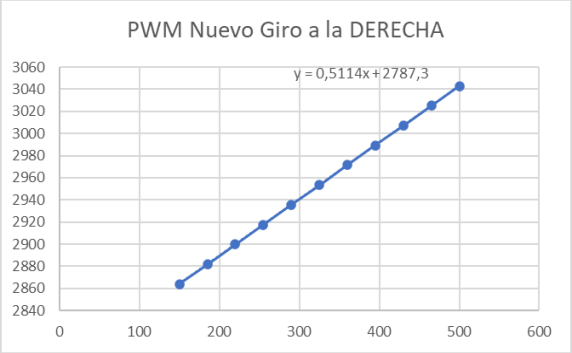
ZONAS 5 y 7

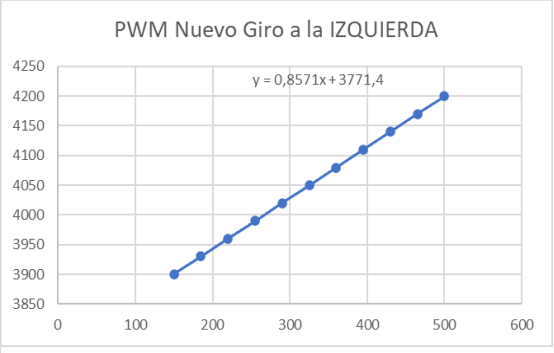
RUEDA IZQUIERDA

	distancia {mm}	PWM Nuevo
160%	150	2864
161%	185	2882
162%	220	2900
163%	255	2918
164%	290	2936
165%	325	2954
166%	360	2971
167%	395	2989
168%	430	3007
169%	465	3025
170%	500	3043

RUEDA DERECHA

	distancia {mm}	PWM Nuevo
130%	150	3900
131%	185	3930
132%	220	3960
133%	255	3990
134%	290	4020
135%	325	4050
136%	360	4080
137%	395	4110
138%	430	4140
139%	465	4170
140%	500	4200

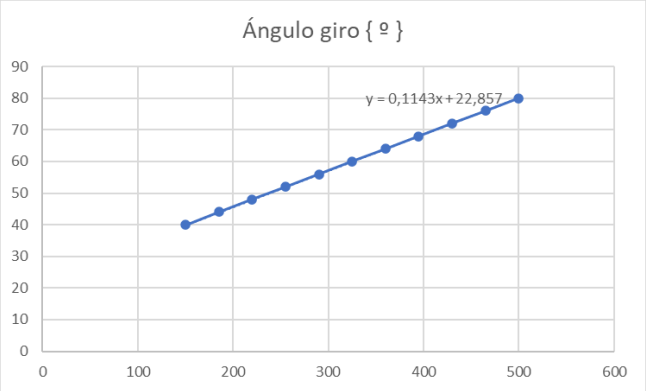




Un método parecido se aplica al ángulo de giro que tiene que alcanzar el robot para esquivar el objeto adecuadamente.

El ángulo de giro dicho será proporcional a la medida dada por el sensor a la que se encuentra el objeto. Esto se hace así para todas las zonas del barrido del sensor excepto para la zona 1, que siempre girará hasta los 72º.

Distancia {mm}	Ángulo giro { º }
150	40
185	44
220	48
255	52
290	56
325	60
360	64
395	68
430	72
465	76
500	80



Una vez establecidos los nuevos PWM de las ruedas para hacer la maniobra de esquive del objeto se aplican a las ruedas y el robot gira hasta que alcanza el ángulo también calculado Gamma_turn. Entonces se para el robot y rota gracias a la función RotaRobot para volver al ángulo original (90º, mirando al frente).

- *Step (pasos, Step_direction):* Función que controla el motor paso a paso. En primer lugar, se enciende el pin conectado al SLEEP del controlador DRV8825.
  La variable de entrada Step_direction, que marca el sentido de giro del motor, y tiene los siguientes posibles valores
  - Step_direction = 0 → Giro en sentido horario.
  - Step_direction = 1 → Giro en sentido antihorario.

  El motor gira el número de pasos que se ha indicado en la variable de entrada de *pasos* haciendo que cada paso que se de salte una interrupción que hace que aumente la cuenta.

- *ToFMeasure():* Función que utiliza el sensor ToF para realizar las medidas. Se ha ajustado para obtener, de los objetos que encuentre el sensor (que pueden ser varios debido al modo de funcionamiento), la medida del objeto

que se encuentre más cerca del robot. Esta medida es la que devuelve la función.

- *ResetConfigVL53L3x():* Función que sirve para resetear los valores iniciales del sensor ToF para que en el caso de que deje de medir por algún tipo de error, que se reinicie y pueda volver a medir.

13 MÁQUINA DE ESTADOS

ESTADO 0:

Estado inicial. Entra aquí en después de realizar las inicializaciones de todos los periféricos y de todas las variables. Espera a que se reciba una señal por USART3 desde el archivo de Python para poder empezar.

Si no llega nada no pasa nada. Vuelve a repetirse el estado 0

Si llega algún dato:

1. Se asigna el dato recibido a las coordenadas X, Y que tiene el punto de destino. Hay 3 puntos seleccionados.
2. Se pasa al estado 1.

ESTADO 1:

Hace el cálculo del ángulo que tiene que girar y la distancia que tiene que recorrer desde el punto en el que se encuentra. Lo almacena en las variables alfa e hipotenusa.

El robot gira el ángulo calculado en el estado 1 en el sentido necesario para hacer una trayectoria de línea recta con la función *RotaRobot*.

Cuando termina se para.

Pasa al estado 102.

ESTADO 2:

Comienza el movimiento de las ruedas con un arranque suave hasta la velocidad nominal. Cuando termina se alcanza la velocidad nominal se cambia la variable de la máquina de estados para cambiar al estado 3.

ESTADO 3:

El motor paso a paso comienza a hacer el movimiento que provoca la rotación del sensor ToF y se empiezan a hacer las mediciones

- Si detecta el sensor un objeto a una distancia inferior a 500 mm: pasa al estado 5.
- Si se llega al punto destino de forma normal: estado 4.

ESTADO 4:

(se pone el flag_correccion_pwm a 0 para que no se igualen las velocidades de las ruedas)

El robot ha llegado al punto destino por lo que se paran las ruedas y se vuelve a la posición recta (ángulo $\frac{\pi}{2}$) mediante la función *RotaRobot*. Se pasa al estado 6.

ESTADO 5:

El sensor ha encontrado un objeto. Se entra en la función *AvoidObject* en la cual se cambian los PWM de las ruedas en función de donde se haya encontrado el objeto para girar el robot hasta un ángulo gamma determinado. Cuando llega a ese ángulo gamma se para y se aplica la función *RotaRobot* para volver al ángulo recto en el que mira hacia adelante (en dirección Y).

Se pasa al estado 101 para volver a hacer los cálculos del ángulo que ha de girar para ir al punto destino.

ESTADO 6:

Se manda por comunicación serie través de la radio el comando al pc para poder volver a empezar el programa. Este comando 'Q' se envía 3 veces para asegurarse que es recibido de forma correcta por el PC (A veces coincide el fin del *timeout* de la recepción de datos con el envío por parte del microcontrolador).

Se pasa al estado 0.

13.1 Flujogramas

Los flujogramas del código escrito en Python y del programa del microcontrolador se adjuntan al final de la memoria para aportar mayor claridad. Estos han sido exportados desde Google Docs.

14 MONTAJE Y RESULTADOS

Como último paso se realiza el montaje de todos los componentes diseñados e impresos en 3D, los componentes electrónicos y los componentes seleccionados.

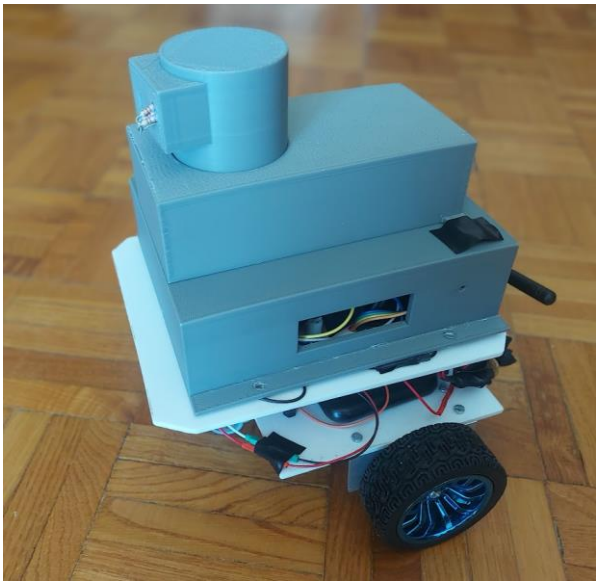


Ilustración 80. Montaje final del robot.

Cada uno de los aspectos que se han descrito a lo largo de esta memoria se han probado de forma de independiente. Desde el motor paso a paso, que se comprueba que da una vuelta completa con 200 pulsos, los motores de corriente continua con sus encoder o el sensor ToF. Este apartado comienza cuando se confirma que todo ha funcionado de correctamente.

El montaje se realiza siguiendo la disposición de elementos en el carro explicada en su respectivo apartado. Los elementos se unen entre ellos mediante tornillería y en el caso del sensor ToF se aplica una masa adhesiva "*Blue Tack*" con el objetivo de conseguir fijación dentro de su caja. Durante el montaje, debido a que los componentes electrónicos están muy cerca unos de otros se han optado por recubrir algunos de estos componentes con cinta aislante.

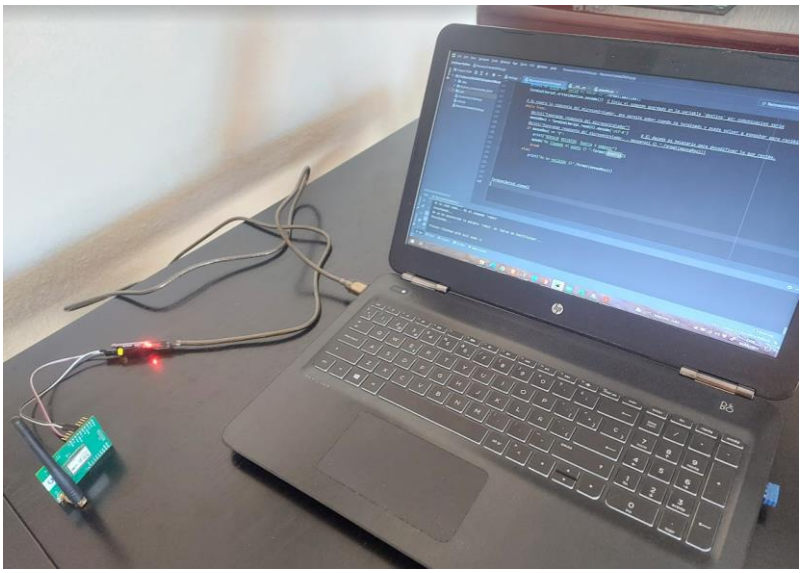


Ilustración 81. PC con módulo de radiofrecuencia en funcionamiento.

Tras montar con éxito el robot y comprobar que el funcionamiento de los componentes en el conjunto sigue siendo correcto al igual que cuando se comprobó de forma independiente se procede a hacer las pruebas del programa.

Para probar el robot se habilita un espacio (lo más espacioso posible dentro del domicilio del autor) donde se colocan unas marcas medidas que son los puntos a los que se debe dirigir el robot según se le mande. El punto '0' es el punto donde parte el robot y el resto de los puntos son a los que se dirige según las órdenes dadas.

Ilustración 82. Espacio de pruebas del proyecto.

Después de una primera prueba del conjunto se observa que el movimiento recto funciona de forma correcta gracias al PID implantado, el movimiento del sensor gracias al motor paso a paso es correcto también, pero a la hora de esquivar un objeto detectado en la trayectoria no se esquiva de una forma muy efectiva porque al girar hacia la derecha el giro es muy agresivo y si tiene que hacer el giro a la derecha es demasiado débil y tarda demasiado en alcanzar el ángulo deseado.

Es normal obtener estos resultados en una primera prueba, por lo que se opta por reajustar las ecuaciones en las que se establecen los PWM de giro de las ruedas para que sea más efectivo el giro.

También se puede ver que el sensor, al estar en continuo movimiento de rotación da algunas medidas nulas, pero se consigue hacer funcionar el robot a pesar de este inconveniente.

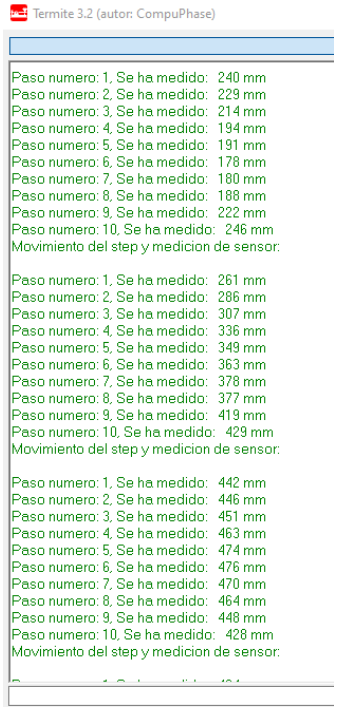


Ilustración 83. Mediciones del sensor montado en carro. 10 mediciones por vuelta

Una vez que funcionan de forma correcta las maniobras de esquivar se hace la prueba de la interfaz de voz, comprobando que se envía desde el microcontrolador el comando "Q" para hacer saber al PC que ha terminado el proceso y que se pueda volver a empezar el proceso.

Se hacen las correspondientes pruebas del funcionamiento en 3 puntos diferentes obteniendo resultados aproximados al punto de destino.

Aquí se muestran los datos recibidos en termite en el funcionamiento del robot en el recorrido al punto 3 establecido:

```
---------------------- INCIO PROGRAMA ----------------------
3
Estado inical X= 0.000000, Y=0.000000 Gamma=1.570796
Se mueve un angulo de -19.653826 y una distancia de 1486.606934
Gira en sentido antihorario
Esta en el punto X= 1.132466, Y=-7.113525 Gamma=1.930260
Comienza el movimiento adelante...
Termina arranque suave.
Medidas son: 8888, 3235, 8888, 1543, 925, 885, 1140, 1128, 1348, 1330
Esta en el punto X= -111.059296, Y=283.549805, Gamma=1.936115
Medidas son: 1630, 3172, 8888, 1321, 1346, 1305, 8888, 1048, 1007, 1768
Esta en el punto X= -186.172501, Y=481.335022, Gamma=1.933773
Medidas son: 73, 1624, 8888, 1665, 1091, 1056, 979, 1470, 1282, 8888
Esta en el punto X= -262.573303, Y=683.985107, Gamma=1.932602
Medidas son: 1447, 2803, 1426, 2820, 1196, 3265, 8888, 1309, 1349, 1885
Esta en el punto X= -338.008209, Y=884.741516, Gamma=1.932602
Medidas son: 2035, 1408, 8888, 1831, 1443, 1406, 1140, 1745, 8888, 1140
Esta en el punto X= -410.774933, Y=1078.062500, Gamma=1.931431
Medidas son: 968, 1494, 5527, 1076, 8888, 1818, 2581, 1473, 1570, 2012
Esta en el punto X= -484.622009, Y=1274.541626, Gamma=1.932602
Medidas son: 1795, 1197, 8888, 1970, 1603, 1626, 1140, 1970, 5790, 1019
Esta en el punto X= -558.336060, Y=1470.595825, Gamma=1.926748
Ha llegado al objetivo X= -558.336060, Y=1470.595825, Gamma =1.926748
Rota para volver al frente un angulo de Gamma =20.461596
Gira en sentido horario
Ya ha girado. esta en X= -565.946228, Y=1489.881348, Gamma =1.538013
QQQ
```

Considerando que el punto 3 tiene las coordenadas X = -500, Y = 1400 tiene bastante buena precisión. En el eje X se va 65 milímetros a la izquierda y en el eje Y 89 mm hacia arriba. Estos valores teóricos son los calculados con la odometría, pero en la realidad es algo diferente pero no algo muy determinante.

Debido a la precisión de la odometría es imposible conseguir resultados más precisos que los que se obtienen. Hay que destacar que al ser un sistema que con errores acumulados cuanta más distancia recorra el robot más impreciso será y en el caso

que detecte un objeto, al hacer giros y rotaciones la diferencia entre la posición calculada y la posición real es mayor.

La rutina de funcionamiento es la siguiente:

El archivo de Python está escuchando continuamente, se puede dejar de fondo todo el tiempo deseado y cuando escucha la palabra "robot" se activa el resto del programa.

El PC pregunta a qué punto se quiere ir y se contesta pudiendo decir uno de los varios comandos disponibles para cada punto. Ejemplo punto 1: punto A, punto 1, estación laminación (para una posible aplicación industrial). El PC informa que el robot va a ir a ese punto y comienza a moverse hacia ese punto. Cuando el robot llega el PC informe mediante voz que ya ha llegado y se puede volver a dar otra orden.

15 CONCLUSIONES Y VÍAS FUTURAS

Llegados a este punto se da por terminado el proyecto. Esto quiere decir que se han conseguido los resultados que se esperaban satisfactoriamente.

En primer lugar, hay que indicar que se han aplicado los conocimientos adquiridos en diferentes asignaturas del Máster de Ingeniería Mecatrónica cursado en aspectos como el control automático de motores eléctricos, de implementación de este control en sistemas embebidos a través de una máquina de estados y el diseño de piezas mediante software CAD/CAM, destacando la ayuda del tutor del proyecto en aspectos más allá del conocimiento actual y en una comprensión mucho más profunda de la asignatura del procedimiento de trabajo con microcontroladores de la marca *STMicroelectronics*.

El aprendizaje de unas nociones de software de reconocimiento de voz e implementación de este para la comunicación serie ha sido más que satisfactorio con un resultado muy efectivo para considerarse un primer contacto.

Se ha comprendido las limitaciones de la odometría como sistema de localización del vehículo en el entorno y se ajustado y probado de la forma más precisa para hacer de estas limitaciones un leve inconveniente.

Sin ninguna duda el proceso más largo, tedioso y problemático ha sido la adaptación del sensor ToF empleado a esta aplicación, puesto que no está fabricado para su conexión directamente a la placa sino a través del módulo P-NUCLEO-53L1A1 de la propia marca. Tampoco está ideado en el sentido de la función que desempeña en este proyecto, pues la comunicación I2C que utiliza el sensor con el microcontrolador está pensada para sistemas en los que no hay mucha longitud de cable y sin movimientos, por lo que en un sistema rotativo como que se ha hecho ha dado muchos problemas de interferencia de las señales CLK y SDA del I2C.

La importancia de la organización en el montaje físico es clave tanto en el sentido de la rapidez del montaje gracias a la organización previa del *layout* de los elementos como en la precaución de contactos entre elementos (a lo largo del montaje se han sufrido cortocircuitos y electrónica quemada en algunas ocasiones siendo necesario la adquisición de material nuevo).

15.1 Problemas encontrados

Uno de los problemas más importantes que se han encontrado ha sido la adaptación del sensor a la placa por su complejidad a la hora de instalar la API de forma correcta.

A la hora del funcionamiento del sensor en la aplicación real ha ocurrido un error que no se ha conseguido solucionar de forma total. Al encontrarse en continuo movimiento de rotación el sensor realiza algunas medidas nulas. Esto no pasa cuando el sensor trabaja estáticamente. Se ha utilizado un analizador lógico para revisar las señales digitales que envía la comunicación I2C del sensor con el microcontrolador.

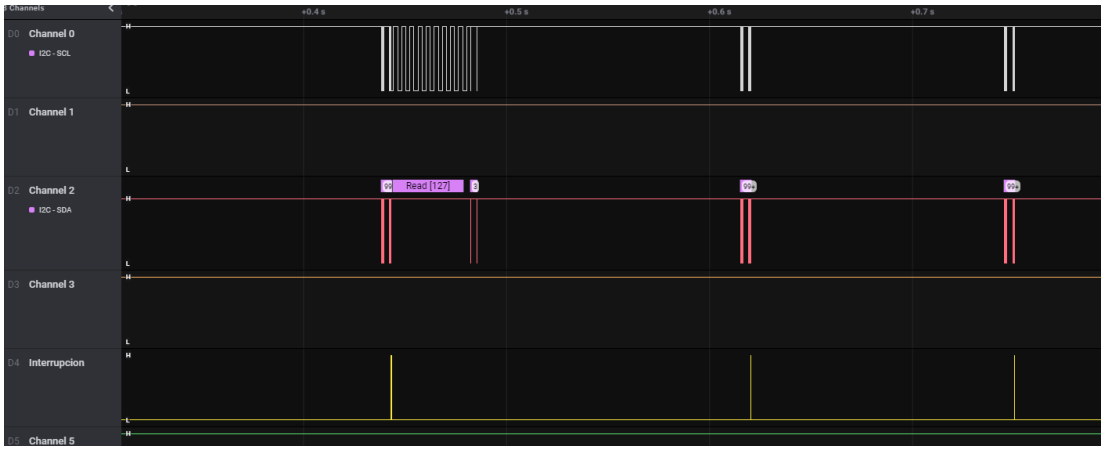


Ilustración 84. Muestra de la comunicación I2C defectuosa

Analizando la señal se puede ver que a la izquierda funciona de forma correcta y a la derecha tanto la señal del *SCL* como la señal del *SDA* quedan distorsionadas. Esto se debe a que se crean interferencias entre señales debido al giro del sensor y de los cables Dupont que los conectan. Se ha probado a utilizar un cable apantallado pero el resultado es el mismo.

Para corregirlo se ha creado una función que reinicia la configuración inicial del sensor y hace que vuelva a medir. Esta función se activa cuando aparecen 2 mediciones seguidas nulas (en la función está programado para que cuando se haga una medición nula se envíe un 8888mm). Se aprecia mejoría, pero no se consigue un resultado 100% fiable debido a que vuelve a ocurrir este error.

15.2 Vías futuras

Por último, se quiere analizar los aspectos a mejorar del proyecto que son conocidos por el autor y la razón por la que no se ha podido hacer estas mejoras.

1. Reconocimiento de voz online: El motor de reconocimiento de voz empleado es el creado por Google ya que es el que mejores resultados ha proporcionado tras hacer una serie de pruebas con otros servicios, pero el gran inconveniente que tiene es que este motor envía la información recogida a los servidores de Google vía internet y es allí donde se convierte a texto. En un ambiente industrial en el que no tiene por qué haber conexión a internet dentro de una planta esto resulta un problema. Como posibles soluciones se puede aplicar un motor de reconocimiento de voz offline (que también se ha probado) pero no está tan perfeccionado ni funciona de forma efectiva. La otra posible solución es proporcionar conexión a internet de forma independiente al PC mediante una tarjeta de telefonía móvil.

2. Sistema de localización: La odometría como sistema de localización se puede considerar aceptable en trayectorias no muy largas y sin muchos giros, pero ni mucho menos perfecta.

   Hay muchos otros sistemas que son mucho mejores a la hora de localizar el vehículo en el mapa como son el SLAM, que genera un mapa virtual donde se localiza y genera trayectorias más precisas y actualizadas, sistemas de localización por wifi en las que a través de 3 antenas se localiza de forma muy precisa la posición del robot y otros sistemas aún más depurados.

   Estas mejoras son muy susceptibles de poder realizarse, pero no ha sido posible por su complejo diseño e implementación, por el aumento de presupuesto, falta de recursos y de tiempo. Sin duda sufriría una mejora importante en su funcionamiento.

3. Timer de 32 bits: Los timers de las ruedas dan un pulso por cada ranura recorrida del encoder. Estos timers tienen una resolución de 16 bits por lo que

son capaces de contar hasta 65536. Esto, transportado a medidas reales son cerca de 13.5 metros, lo cual es suficiente para un proyecto como este y para las pruebas que se hacen pero en aplicaciones industriales estos timers deberían ser de 32 bits que son capaces de contar hasta 4294967296 pulsos, lo que son 887699 metros El microcontrolador seleccionado tan solo tiene un timer de 32 bits por lo que habría que buscar otro microcontrolador que tenga 2 timers de 32 bits o conseguir la forma de encadenar 2 timers de 16 bits para conseguir el segundo timer de 32 bits

16 PRESUPUESTO

El presupuesto de este proyecto se hace de forma aproximada y contando que el único trabajador participante ha sido el autor de este proyecto, Álvaro Serna Pérez. El presupuesto se ha desglosado en 2 partes que son el coste de los materiales y el pago de las horas de trabajo del trabajador.

16.1 Costes materiales

Es la suma de todos los componentes adquiridos para el montaje del robot. Ninguno de los costes a continuación tiene incluidos los gastos de envío.

Componentes	PRECIO
Correa dentada Rs	2,53 €
Polea síncrona RS	11,02 €
Rodamiento RS	3,49 €
Microcontrolador RS	16,44 €
Motores CC + Ruedas	30,99 €
Bateria 1	18,29 €
Cables dupont	5,00 €
Estaño	2,00 €
Termoretractiles	1,50 €
Sensor ToF VL53L3X	17,48 €
Convertidor DC-DC XL6009E1	3,13 €
Driver L298N	2,68 €
Cable USB	3,00 €
Driver DRV8825	3,09 €
Tack Azul (pegamento)	0,90 €
Tornillos	2,50 €
Tuercas	1,00 €
Módulo FT232RL	3,60 €
TOTAL	128,64 €

16.2 Gastos personales

En cuanto al coste de gastos de personal, con un único trabajador, se basa en el XIX convenio colectivo de ingenierías y estudios técnicos, con un salario de 20 €/h. Se han recogido las actividades que el ingeniero ha realizado a lo largo del tiempo de diseño y montaje del proyecto.

Actividad	Horas	Costo por horas	Coste
Diseño de reconocimiento de voz	35	20	700,00 €
Diseño de piezas	12	20	240,00 €
Selección de componentes	2	20	40,00 €
Ajuste de componentes mecánicos	7	20	140,00 €
Adaptación del sensor ToF	30	20	600,00 €
Layout del carro	1	20	20,00 €
Soldar	2	20	40,00 €
Odometría	15	20	300,00 €
Control motores CC	15	20	300,00 €
Programación código	100	20	2.000,00 €
Montajes y reajustes	15	20	300,00 €
TOTAL SIN IVA	234	20	4.680,00 €
TOTAL CON IVA (21%)			5.662,80 €

16.3 Coste de licencias

En el departamento de ingeniería se hace uso de software cuyas licencias entra dentro de los costes de ingeniería en proyectos de esta índole. En este caso no hay mucho software de pago pues el principalmente usado es de licencia libre.

Software	Precio
SolidWorks	1.010,00 €
Microsoft Office	99,00 €
TOTAL	1.109,00 €

16.4 Presupuesto total

Una vez definidos los costes de los aspectos del proyecto se muestran de forma conjunta para comprender el coste total.

Concepto	Precio
Componentes	128,64 €
Personal	5.662,80 €
Software	1.109,00 €
TOTAL	6.900,44 €

17 PLIEGO DE CONDICIONES

17.1 Definición

El objetivo de este apartado es definir cuáles son las condiciones técnicas establecidas en la programación, diseño y montaje del robot controlado por comandos de voz y LiDAR.

17.2 Condiciones técnicas

Los elementos electrónicos con una función determinante en el proyecto son complementados con su documentación técnica en este documento que será anexada al final de este con el fin de clarificar las pertinentes características de cualquiera de los ámbitos de los elementos electrónicos.

Las piezas diseñadas en 3 dimensiones han sido impresas por el departamento de ingeniería de la Universidad Politécnica De Valencia, por lo que se asegura el cumplimiento de las normativas de seguridad y calidad para este tipo de trabajos.

En cuanto al hardware empleado para el diseño computacional hay que aclarar que no es el mínimo requerido para su desarrollo, pero es el que se ha empleado.

- Procesador Intel Core i7-9750H.
- Memoria RAM 16 GB.
- Almacenamiento interno libre de 500 GB.
- Tarjeta Gráfica Nvidia GTX 1650.

De la misma forma, el software empleado es el siguiente:

- Windows 10 Profesional.
- Paquete Microsoft Office 2019
- SolidWorks 2019
- STM32 CubeIDE versión 1.7.0
- Logic 2.3.33
- Pycharm 2020.3.4

17.3 Condiciones económicas

Tal y como se ha indicado en el apartado del cálculo del presupuesto este es un aspecto sin una gran relevancia puesto que el objetivo del proyecto no es el de ofrecer un servicio o producto con el que obtener un beneficio económico, sino que es una muestra de conocimientos y habilidades para un Trabajo de Fin de Máster. La

elaboración de un presupuesto aproximado es un aspecto más del conocimiento de un ingeniero y como tal se ha tratado.

18 BIBLIOGRAFÍA

Introducción AGV: https://www.astimobilerobotics.com/es/tecnologia-agv/navegacion-slam#header

Rmovil – Cinemática – Diferencial. Polimedia UPV: https://media.upv.es/#/portal/video/3889d460-8181-11e6-8f97-b7415bf6c110

Control con DRV8805: http://carlini.es/manejar-un-motor-stepper-con-un-driver-drv8825-y-arduino/

Documentación asignatura Sistemas Embebidos del Máster Ingeniería Mecatrónica UPV.

Documentación asignatura Diseño Electrónico Avanzado Ingeniería Mecatrónica UPV.

Documentación asignatura Control Automático Ingeniería Mecatrónica UPV.

19 ANEXOS

19.1 Código de Python

```
import re
import pyttsx3
import speech_recognition as sr
import time
import serial

# Ajustes del reconocimiento de voz
engine = pyttsx3.init()  # inicia el motor de habla
engine.setProperty("rate", 120)
engine.setProperty("voice", "spanish")
r = sr.Recognizer()
mic = sr.Microphone()

# Ajustes de la comunicacion serie
COM = "COM11"                                   # CAMBIAR SEGUN DONDE
ESTÉ CONECTADO EL MÓDULO DE RADIO
TerminalSerial = serial.Serial(COM, baudrate=115200, bytesize=8,
parity='N', stopbits=1, timeout=2)
DATO = 0

comandos_Punto_0 = ["0", "inical"]
comandos_Punto_A =["A", "1", "laminación"]
comandos_Punto_B = ["B", "2", "corte"]
comandos_Punto_C = ["C", "3"]


def listen_sentence():
    try:
        with mic as source:
            print("Escuchando...")
            audio = r.listen(source, timeout=None,
phrase_time_limit=6)
            text = r.recognize_google(audio, language="es-ES")
            return text
    except:
        print("No he oido nada... Da la instruccion ")

def listen_init():
    try:
        with mic as source:
            # r.adjust_for_ambient_noise(source, duration = 1)  #
listen for 1 second to calibrate the energy threshold for ambient
noise levels
            print("Escuchando...")
            audio = r.listen(source, timeout=None,
phrase_time_limit=3)
            text = r.recognize_google(audio, language="es-ES")
            return text
    except:
        print("No he oido nada... Di el comando 'robot' ")
```

```
def speak(sentence):
    engine.say(sentence)
    engine.runAndWait()


def filter_word(text, command):
    try:
        if command in text.split():
            print("Palabra encontrada ")
            return True
        else:
            print("No se ha encontrado la palabra '{}' en
'{}'...".format(command, text))
    except:
        pass
        # print("No se ha escuchado ningun texto")


if __name__ == "__main__":
    with mic as source:
        r.adjust_for_ambient_noise(source,
                                   duration=1)  # listen for 1
second to calibrate the energy threshold for ambient noise levels

    while True:
        text = None
        while True:
            text = listen_init()
            if filter_word(text, "robot"):
                break

        speak("Dime, qué quieres que haga?")

        destino = 0                                # Se resetea el
valor de destino para el siguiente bucle
        while destino == 0:
            try:
                respuesta = listen_sentence()
                print(respuesta)

                ############## CASO 0 #################
                ########### Ir al punto 0 #############
                for palabra in comandos_Punto_0:
                    if palabra in respuesta.split():
                        destino = '0'
                        memoria = palabra
                        speak("De acuerdo, Voy al punto inicial")

                ############## CASO 1 #################
                ########### Ir al punto 1 #############
                for palabra in comandos_Punto_A:
                    if palabra in respuesta.split():
                        destino = '1'
                        memoria = palabra
```

```
                        speak("De acuerdo, Voy al punto A")

                ############## CASO 2 #################
                ########## Ir al punto 2 #############
                for palabra in comandos_Punto_B:
                    if palabra in respuesta.split():
                        destino = '2'
                        memoria = palabra
                        speak("De acuerdo, Voy al punto B")
                ############## CASO 3 #################
                ########## Ir al punto 3 #############
                for palabra in comandos_Punto_C:
                    if palabra in respuesta.split():
                        destino = '3'
                        memoria = palabra
                        speak("De acuerdo, Voy al punto C")



            except:
                speak("No te he entendido, repite el comando para
empezar")

        print("Se envía por serie el valor {}".format(destino))
        TerminalSerial.write(destino.encode())  # Envia el comando
guardado en la variable 'destino' por comunicacion serie

        # Se espera la respuesta del microcontrolador, que permite
saber cuando ha terminado y puede volver a escuchar para recibir
nuevos comandos
        while True:
            #print("Esperando respuesta del microcontrolador")
            mensaReci = TerminalSerial.read(8).decode('utf-8')
# El decode es necesario para decodificar lo que recibe.
            #print("Esperando respuesta del microcontrolador...
mensareci {} ".format(mensaReci))
            if mensaReci == "Q":
                print("MENSAJE RECIBIDO. Vuelta a empezar")
                speak("Ha llegado al punto {} ".format(memoria))
                break
            else:
                print("Se ha recibido {}".format(mensaReci))




TerminalSerial.close()
```

19.2 Código de STM32

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "rtc.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "vl53l1_api.h"
#include "53L1A2.h"
#include "math.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
/*-----------------Step Motor-----------------------------*/
uint8_t step=0,state=0,cuenta=0, Sent;

uint8_t fin;
/*--------------------------------------------------------*/

uint8_t estado;         //Variable de la maquina de estados
uint8_t flag_correccion_pwm = 0;

uint32_t encoder_distance = 900000;

/*------------------Motores ruedas--------------------------*/
uint16_t  pwmRW = 3000, pwmLW_slow = 1790 , pwmRW_slow= 3000, pwmLW_boost = 1770, pwmRW_boost
= 3000, PWMLW_turn, PWMRW_turn;
```

```
int  Kp = 300000, Kd= 10000, Ki = 50, error_Vel_ant =0, pwmLW = 1500;
uint16_t encoder_izq = 0 , encoder_der =0, encoder_izq_ant = 5000, encoder_der_ant = 5000;
uint32_t  msecTOT, msecTOT_ant = 0, difer_tiempo;
int difer_encoder_der, difer_encoder_izq;
uint8_t  Base = 190, Sentido_rotacion, Radio_rueda = 34;        //En milimetros
#define Pi 3.141592654
float  W_rueda_izq, W_rueda_der, diferenia, V_rd, V_ri, V_inst, W_rot, Gamma = Pi/2, X = 0 ,
Y = 0, K_pul_rad = 0.006356275, Gamma_turn, error_Vel, error_Vel_Deriv, suma_integrativ;

/*-------------------------------------------------------------*/

/*-----------------Calculos de trayectoria--------------------*/
float hipotenusa, alfa;
int Sent_Rota;


/*-------------------------------------------------------------*/

/*-----------------Sensor-----------------------------*/
uint32_t measure[200], min_distance = 500, Dist_obj;

uint8_t CW = 2, flag_measure =0, Zone_Object, ContReset =0;


I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;
VL53L1_Dev_t                    dev;
VL53L1_DEV                      Dev = &dev;
int status;
volatile int IntCount;
#define isInterrupt 1 /* If isInterrupt = 1 then device working in interrupt mode, else
device working in polling mode */
/*-------------------------------------------------------------*/

int flag = 0, i;

/*--------------Comunicacion microcontrolador - archivo python ------------------*/
char var1;
uint8_t buffer_mensa_received[48];
float X_dest, Y_dest;           //Son las coordenadas destino que se establecen segun el
comando que se reciba desde el python



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */


/*-----------------------------Step Motor-------------------------------------*/
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
{
        if(htim->Instance == TIM1) // Si la fuente de la interrupcion viene del temporizador
1
        {
                cuenta++;
                                                                        // incrementa el
contador en cada interrupcion
                if(cuenta == step)
                                                                // Si el contador de ha
aumentado en el numero de pasos deseado
                {
                        HAL_TIM_PWM_Stop_IT(&htim1,TIM_CHANNEL_1);      // Para el PWM
```

```
                        cuenta=0;
                                                                        // resetea el
contador
                        state=0;
                                                                // Para salir del bucle while
dentro de la funcion de paso

                }
        }
}
/*-----------------------------------------------------------------------------*/

void LeftWheel(uint8_t direction, uint16_t pwm){
        if (direction == 2){    // Movimiento hacia atrás
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
        }
        else if (direction == 1){       //Movimiento hacia adelante
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        }
        else if (direction == 0){       // Parado
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
        }
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, pwm);
}

void RightWheel(uint8_t direction, uint16_t pwm){
        if (direction == 2){            // Movimiento hacia atrás
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET);
        }
        else if (direction == 1){               //Movimiento hacia adelante
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        }
        else if (direction == 0){               //Parado
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET);
        }
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_2, pwm);
}

void RotaRobot (uint8_t sentido, float Gamma_rot){
        Gamma_rot = Gamma_rot * Pi/180;                 // Se introduce el angulo en la
funcion en grados y aqui se pasa a radianes
        float Gamma_init = Gamma;               // Se almacema el ángulo inicial del robot
(en radianes)

        //sentido 1 giro horario------------ sentido 2 giro antihorario
        switch( sentido){
        case 1:                         // Giro sentido horario
                Print_Usart2("\n");
                Print_Usart3("Gira en sentido horario\n");
                //Rueda izquierda adelante
                LeftWheel(1, 1500);
                //Rueda derecha atrás
                RightWheel(2,1600);
                break;

        case 2:         // Giro sentido antihorario
                Print_Usart3("Gira en sentido antihorario \n");
                // Rueda derecha adelante
                RightWheel(1, 1600);
                //Rueda izquierda atras
                LeftWheel(2, 1500);
        }
```

```
        // Se aplica el pwm a las ruedas


        while(fabs(Gamma - Gamma_init) <= Gamma_rot){}          //Espera a que se termine de
hacer la rotación hasta el ángulo que se desea
        //Cuando llega aq ese ángulo se para el motor
        LeftWheel(0,pwmLW_slow);
        RightWheel(0,pwmRW_slow);

}

void AvoidObj(){
        uint16_t buff_Avoid[220];
        float Gamma_init;
//      Zone_Object;    // Es la zona del circulo en la que se ha encontrado el objeto. Va de
cero a 9
//      Dist_obj;               // Es la distancia a la que se ha encontrado el objeto.
Tambien se puede saber en measure[Zone Object];
//      Sent    ;               // El sentido en el que ha girado el stepper

        switch (Sent){          // Para saber si el giro se ha hecho en sentido horario o
antihorario

        case 1:                         // Gira en sentido horario
                switch(Zone_Object){
                case 0: // EL objeto se ha encontrado en la zona 1 (En la parte frontal)
                        // Se reduce la velocidad a la velocidad lenta
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        // Se compara distancia zona 10 con zona 2. Girará en dirección de la
que más lejos esté el objeto detectado
                        if(measure[9] >= measure[1]){
        //Gira a la izquierda
                                PWMLW_turn = pwmRW_slow;
                                PWMRW_turn = 0.8571* Dist_obj + 3771.4;
        // Funcion que se saca del excel
                                Sentido_rotacion = 1;
                        // Después tiene que rotar a la derecha para volver al ángulo inicial
                        }else {
                        // Gira a la derecha
                                PWMRW_turn = pwmRW_slow ;       // Funcion que se saca del
excel
                                PWMLW_turn = 0.5114 * Dist_obj +2250.3;
                                Sentido_rotacion = 2;           // Después tiene que rotar a
la izquierda para volver al ángulo inicial
                        }
                        Gamma_turn = 72/* En grados */* Pi/180;         //angulo en radianes
que tiene que girar el robot
                        break;

                case 1: // El objeto se encuentra en la zona 2
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj +4071.4;
        // Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 2:// El objeto se encuentra en la zona 3
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
```

```
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3921.4;// Funcion que se saca del
excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 3:// El objeto se encuentra en la zona 4
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3921.4;// Funcion que se saca del
excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 4:// El objeto se encuentra en la zona 5
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3771.4;// Funcion que se saca del
excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 5:// El objeto se encuentra en la zona 6
                        // Aumenta la velocidad del robot
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_boost);
                        LeftWheel(1, pwmLW_boost);
                        Gamma_turn = Gamma;
                        HAL_Delay(1000);
                        break;

                case 6:// El objeto se encuentra en la zona 7
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2787.3;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 7:// El objeto se encuentra en la zona 8
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);
```

```
                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2876.8;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 8:// El objeto se encuentra en la zona 9
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2876.8;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 9:// El objeto se encuentra en la zona 10
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2966.3;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;
                }
                //Fin del switch de Zone_object
                ////////////////////////////////

                //Se aplica la velocidad de giro a las ruedas
                sprintf(buff_Avoid,"Velocidad lenta de funcionamiento\n");
                Print_Usart3(buff_Avoid);Print_Usart2(buff_Avoid);
                Gamma_turn = Gamma_turn *Pi/180;
                                                                                // Se pasa a
radianes
                Gamma_init = Gamma;
                RightWheel(1, PWMRW_turn);
                LeftWheel(1, PWMLW_turn);
                sprintf(buff_Avoid,"Esperando a que cumpla Gamma =%f \n", Gamma_turn);
                Print_Usart3(buff_Avoid);Print_Usart2(buff_Avoid);
                while(fabs(Gamma - Gamma_init) <= Gamma_turn){}
                                                                        //Espera a que
elngulo de giro sea el que se desea
//              while(Gamma < Gamma_turn){}

        //Espera a que el ángulo de giro sea el que se desea
                //Avanza recto durante 2.5 segundos

                Print_Usart3("Avanza recto durante 2.5 segundos a PWMR=3000 \n");
                flag_correccion_pwm = 1;
                RightWheel(1,3000);
                HAL_Delay(2500);
                flag_correccion_pwm = 0;
                RightWheel(0, pwmRW_slow);
                LeftWheel(0, pwmLW_slow);
                HAL_Delay(1000);
```

```
                sprintf(buff_Avoid,"Ya ha hecho 1 segundo. Se para en X= %f, Y=%f, Gamma=%f
\n", X, Y, Gamma);
                Print_Usart3(buff_Avoid);
                //Cuando llega a este punto se para y gira el robot en el sentido contrario
al que ha girado al principio.
                Print_Usart3("Rota para volver a 90 grados (Mirar adelante)");
                // Rota para volver a gamma Pi (Mirar al frente)
                Gamma_turn = Gamma - (Pi/2);
                if (Gamma_turn > 0){ Sent_Rota = 1;}
                else {Sent_Rota = 2;}
                RotaRobot(Sent_Rota, fabs(Gamma_turn * (180/Pi)));
                HAL_Delay(500);
                break;          //Sale del switch de Sent

        case 0:                                                         // Hace giro en
sentido antihorario
                switch(Zone_Object){
                case 0: // EL objeto se ha encontrado en la zona 1 (En la parte frontal)
                        // Se reduce la velocidad a la velocidad lenta
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        // Se compara distancia zona 10 con zona 2. Girará en dirección de la
que más lejos esté el objeto detectado
                        if(measure[9] >= measure[1]){                                   //
Gira a la derecha
                                PWMLW_turn = 0.5114 * Dist_obj + 2250.3;        // Funcion
que se saca del excel
                                PWMRW_turn = pwmRW_slow;
                                Sentido_rotacion = 1;           // Después tiene que rotar a
la derecha para volver al ángulo inicial
                        }else {
                        // Gira a la izquierda
                                PWMRW_turn = 0.8571 * Dist_obj +3771.4; // Funcion que se
saca del excel
                                PWMLW_turn = pwmLW_slow;
                                Sentido_rotacion = 2;           // Después tiene que rotar a
la izquierda para volver al ángulo inicial
                        }
                        Gamma_turn = 72 * Pi/180;               //angulo que tiene que girar
el robot
                        break;

                case 1: // El objeto se encuentra en la zona 2
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2966.3;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 2:// El objeto se encuentra en la zona 3
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2876.8;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
```

```
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 3:// El objeto se encuentra en la zona 4
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2876.8;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 4:// El objeto se encuentra en la zona 5
                        // Se reduce la velocidad a la velocidad lenta. Girará a la derecha
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMRW_turn = pwmRW_slow;
                        PWMLW_turn = 0.5114 * Dist_obj + 2787.3;                        //
Funcion que se saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 2;           // Después tiene que rotar a la
izquierda para volver al ángulo inicial
                        break;

                case 5:// El objeto se encuentra en la zona 6
                        // Aumenta la velocidad del robot
                        // Se reduce la velocidad a la velocidad lenta.
                        RightWheel(1, pwmRW_boost);
                        LeftWheel(1, pwmLW_boost);
                        Gamma_turn = Gamma;
                        HAL_Delay(1000);

                case 6:// El objeto se encuentra en la zona 7
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3771.4;        // Funcion que se
saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 7:// El objeto se encuentra en la zona 8
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3921.4;        // Funcion que se
saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;
```

```
                case 8:// El objeto se encuentra en la zona 9
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 3921.4;        // Funcion que se
saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;

                case 9:// El objeto se encuentra en la zona 10
                        // Se reduce la velocidad a la velocidad lenta. Girará a la izquierda
                        RightWheel(1, pwmRW_slow);
                        LeftWheel(1, pwmLW_slow);
                        HAL_Delay(500);

                        PWMLW_turn = pwmLW_slow;
                        PWMRW_turn = 0.8571 * Dist_obj + 4071.4;        // Funcion que se
saca del excel
                        Gamma_turn = 0.1143* Dist_obj + 22.857;
                        Sentido_rotacion = 1;           // Después tiene que rotar a la
derecha para volver al ángulo inicial
                        break;
                }
                //Fin del switch de Zone_object
                ////////////////////////////////

                //Se aplica la velocidad de giro a las ruedas
                Gamma_turn = Gamma_turn *Pi/180;        // Se pasa a radianes
                Gamma_init = Gamma;
                RightWheel(1, PWMRW_turn);
                LeftWheel(1, PWMLW_turn);
                while(fabs(Gamma - Gamma_init) <= Gamma_turn){}         //Espera a que
elngulo de giro sea el que se desea
//              while(Gamma < Gamma_turn){}             //Espera a que el ángulo de giro sea
el que se desea                 //ESTO HAY QUE PENSAR COMO EST?N LOS ANGULOS PARA QUE ESTO
SE CUMPLA

                Print_Usart3("Avanza recto durante 2.5 segundos a PWMR=3000 \n");
                flag_correccion_pwm = 1;
                RightWheel(1,3000);
                HAL_Delay(2500);
                flag_correccion_pwm = 0;
                RightWheel(0, pwmRW_slow);
                LeftWheel(0, pwmLW_slow);
                HAL_Delay(1000);
                sprintf(buff_Avoid,"Ya ha hecho 2.5 segundos. Se para en X= %f, Y=%f,
Gamma=%f \n", X, Y, Gamma);
                Print_Usart3(buff_Avoid);
                //Cuando llega a este punto se para y gira el robot en el sentido contrario
al que ha girado al principio.
                Print_Usart3("Rota para volver a 90 grados (Mirar adelante)");
                // Rota para volver a gamma Pi (Mirar al frente)
                Gamma_turn = Gamma - (Pi/2);
                if (Gamma_turn > 0){ Sent_Rota = 1;}
                else {Sent_Rota = 2;}
                RotaRobot(Sent_Rota, fabs(Gamma_turn * (180/Pi)));

//              Gamma_turn = Gamma_turn * 180/Pi;
//              RotaRobot(Gamma_turn, Sentido_rotacion);

                HAL_Delay(500);
                break;          //Sale del switch de CW
```

```
                }                       //////// Fin del switch del CW //////////////
}

void Step(uint16_t pasos ,uint8_t Step_direction)  // Funcion de motor paso a paso.
Necesitará 2 parametros. Pasos y direccion
{
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,SET);//Pin del driver del stepmotor para el sleep,
se activa al empezar y se desactiva al finalizar para que no consuma electricidad
        step=pasos;
        if(Step_direction==0)
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_9,GPIO_PIN_RESET);     // Si la variable
direction es cero, se pone a cero el pin
        else
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_9,GPIO_PIN_SET);       // Si es un valor
distinto de cero, la logica del pin DIR es 1
        state=1;
        HAL_TIM_PWM_Start_IT(&htim1,TIM_CHANNEL_1);     // comienza el PWM
        while(1) // EL motor paso a paso tardará un tiempo en moverse a la posicion deseada
        {                                                       //      Se crea bucle infinito para
evitar que el programa entre nuevamente en la funcion paso
                if(state==0)                                    // Cuando la posicion del
motor es la deseada la variable de estada será cero y saldrá del ciclo
                        break;
                HAL_Delay(1);
        }
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,RESET);      //Pin del driver del stepmotor para
el sleep, se activa al empezar y se desactiva al finalizar para que no consuma electricidad

}
void Arranque_Suave(){
        for ( pwmRW =1700 ;pwmRW < 4000 ;pwmRW += 575){
                  RightWheel(1,pwmRW);
                  HAL_Delay(300);
          }

}

int ToFMeasure(void){
          uint8_t buff_vis[220];
          VL53L1_MultiRangingData_t MultiRangingData;
          VL53L1_MultiRangingData_t *pMultiRangingData = &MultiRangingData;
          uint8_t NewDataReady = 0;
          uint16_t Object[50];
          uint16_t no_of_object_found , j, min = 8888;


              status = VL53L1_GetMeasurementDataReady(Dev, &NewDataReady);
//            HAL_Delay(1);
              if((!status)&&(NewDataReady!=0)){
                status = VL53L1_GetMultiRangingData(Dev, pMultiRangingData);
                no_of_object_found=pMultiRangingData->NumberOfObjectsFound;
                if(no_of_object_found>5){no_of_object_found=5;}
//              sprintf(buff_vis,"Cuenta=%5d, ", pMultiRangingData->StreamCount);
//              Print_Usart2(buff_vis);
//              sprintf(buff_vis,"#Objs=%1d ", no_of_object_found);
//              Print_Usart2(buff_vis);
//              Print_Usart3(buff_vis);
                for(j=0;j<no_of_object_found;j++){
//                if(j!=0)printf("\n                      ");
                Object[j] = pMultiRangingData->RangeData[j].RangeMilliMeter;
//              sprintf(buff_vis,"Objeto numero %d a D=%d mm",j+1 ,Object[j]);
//              Print_Usart2(buff_vis);
//              Print_Usart3(buff_vis);
                if(Object[j] < min){min = Object[j];}
                if(j!=(no_of_object_found - 1)){
//                      sprintf(buff_vis,", ");
//                      Print_Usart2(buff_vis);
//                      Print_Usart3(buff_vis);
```

```
                }

//                sprintf(buff_vis,"status=%d, D=%5dmm, Signal=%2.2f Mcps, Ambient=%2.2f
Mcps",
//                        pMultiRangingData->RangeData[j].RangeStatus,
//                        pMultiRangingData->RangeData[j].RangeMilliMeter,
//                        pMultiRangingData->RangeData[j].SignalRateRtnMegaCps/65536.0,
//                        pMultiRangingData->RangeData[j].AmbientRateRtnMegaCps/65536.0);
//                Print_Usart2(buff_vis);
                }
//              Print_Usart2("\n");
//              Print_Usart3("\n");
//              if (status==0){
                  status = VL53L1_ClearInterruptAndStartMeasurement(Dev);
//              }
              }
              if (min == 8888){ContReset++;}
              else {ContReset = 0;}
              if (ContReset >= 3){
                  ResetConfigVL53L1x();         //Se aplica el reset del sensor
                  ContReset = 0;
              }
          return min;

}

void ResetConfigVL53L1x(){
    uint8_t ToFSensor = 1, buff_vis[220]; // Select ToFSensor: 0=Left, 1=Center, 2=Right
         XNUCLEO53L1A1_Init();
        I2C_HandleTypeDef hi2c1;
        UART_HandleTypeDef huart2;
        VL53L1_Dev_t                   dev;
        VL53L1_DEV                     Dev = &dev;
        int status;
        volatile int IntCount;
        #define isInterrupt 1 /* If isInterrupt = 1 then device working in interrupt mode,
else device working in polling mode */

//         Print_Usart2("VL53L1X Examples...\n");
            Dev->I2cHandle = &hi2c1;
            Dev->I2cDevAddr = 0x52;


    status = XNUCLEO53L1A1_ResetId(ToFSensor, 0); // Reset ToF sensor
    HAL_Delay(2);
    status = XNUCLEO53L1A1_ResetId(ToFSensor, 1); // Reset ToF sensor
    HAL_Delay(2);

          status = VL53L1_WaitDeviceBooted(Dev);
          status = VL53L1_DataInit(Dev);
          status = VL53L1_StaticInit(Dev);
          /* VL53L1_SetPresetMode function is mandatory to be called even if default
PresetMode is the VL53L1_PRESETMODE_RANGING */
          status = VL53L1_SetPresetMode(Dev, VL53L1_PRESETMODE_RANGING);
          //    status = VL53L1_SetDistanceMode(Dev, VL53L1_DISTANCEMODE_SHORT);
          //    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(Dev, 60000);
          status = VL53L1_StartMeasurement(Dev);
          if(status){
                sprintf(buff_vis,"VL53L1_StartMeasurement failed: error = %d \n", status);
                Print_Usart3(buff_vis);/*Print_Usart3(buff_vis)*/;
                estado = 106;
//          while(1);
          }

}

/*-------------------------------------------------------------------------*/
```

```
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)

PUTCHAR_PROTOTYPE
{
  HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, 0xFFFF);
  return ch;
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin==VL53L1X_INT_Pin)
  {
    IntCount++;
  }
}

void Print_Usart2(uint8_t *texto);
void Print_Usart3(uint8_t *texto);
void RangingLoop(void); /* */



/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
          uint8_t byteData;
          uint16_t wordData;


          uint8_t ToFSensor = 1; // 0=Left, 1=Center(default), 2=Right
          uint8_t buff_vis[112];
          uint8_t buff_2[220];
          uint8_t buff[220];
          //Variables de la posicion del robot


          estado = 100;                                                  // Declara el estado
inicial en el que empieza la maquina de estados


  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_TIM1_Init();
  MX_TIM3_Init();
  MX_TIM7_Init();
  MX_RTC_Init();
  MX_TIM4_Init();
  MX_TIM8_Init();
  MX_TIM2_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */
  XNUCLEO53L1A1_Init();
  HAL_TIM_Base_Start_IT(&htim7);

  /*------------------------- INICIA PWM DE MOTORES DE LAS RUEDAS ------------------------
*/
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
  /*--------------------------------------------------------------------------------------
*/


  //------------------STEP MOTOR-----------------------------//
  __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1, 128);
  //---------------------------------------------------------//

  //------------------INICIALIZACION ENCODER-----------------------------//
  HAL_TIM_Encoder_Start (&htim3,TIM_CHANNEL_1);
  HAL_TIM_Encoder_Start (&htim3,TIM_CHANNEL_2);
  HAL_TIM_Encoder_Start (&htim4,TIM_CHANNEL_1);
  HAL_TIM_Encoder_Start (&htim4,TIM_CHANNEL_2);
  //---------------------------------------------------------//

  HAL_TIM_IC_Start_IT(&htim8, TIM_CHANNEL_1);



  //printf("VL53L1X Examples...\n");
   Print_Usart2("VL53L1X Examples...\n");
     Dev->I2cHandle = &hi2c1;
     Dev->I2cDevAddr = 0x52;

     /* Allow to select the sensor to be used, multi-sensor is not managed in this example.
     Only when use use the Left ToF in interrupt mode,  solder the U7 on the X-Nucleo-53L1A1
board
     Only when use the Right ToF in interrupt mode, solder the U7 on the X-Nucleo-53L1A1
board
     See "Solder drop configurations" in the X-Nucleo-53L1A1 User Manual for more details */
     ToFSensor = 1; // Select ToFSensor: 0=Left, 1=Center, 2=Right
     status = XNUCLEO53L1A1_ResetId(ToFSensor, 0); // Reset ToF sensor
     HAL_Delay(2);
     status = XNUCLEO53L1A1_ResetId(ToFSensor, 1); // Reset ToF sensor
     HAL_Delay(2);

     VL53L1_RdByte(Dev, 0x010F, &byteData);
     sprintf(buff_vis,"VL53L1X Model_ID: %02X\n\r", byteData);
     Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;
     VL53L1_RdByte(Dev, 0x0110, &byteData);
     sprintf(buff_vis,"VL53L1X Module_Type: %02X\n\r", byteData);
     Print_Usart2(buff_vis);/*Print_Usart3(buff_vis);*/
     VL53L1_RdWord(Dev, 0x010F, &wordData);
     sprintf(buff_vis,"VL53L1X: %02X\n\r", wordData);
     Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;
//     RangingLoop();
```

```
    /*----------------------------SI SE MIDE EN EL WHILE 1 -------------------------*/

          status = VL53L1_WaitDeviceBooted(Dev);
          status = VL53L1_DataInit(Dev);
          status = VL53L1_StaticInit(Dev);
          /* VL53L1_SetPresetMode function is mandatory to be called even if default
PresetMode is the VL53L1_PRESETMODE_RANGING */
          status = VL53L1_SetPresetMode(Dev, VL53L1_PRESETMODE_RANGING);
          //    status = VL53L1_SetDistanceMode(Dev, VL53L1_DISTANCEMODE_SHORT);
          //    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(Dev, 60000);
          status = VL53L1_StartMeasurement(Dev);
          if(status){
                sprintf(buff_vis,"VL53L1_StartMeasurement failed: error = %d \n", status);
                Print_Usart2(buff_vis);/*Print_Usart3(buff_vis)*/;
                estado = 106;
//          while(1);
          }

          HAL_Delay(4000);

          sprintf(buff,"----------------------- INCIO PROGRAMA ------------------------
\n");
          Print_Usart3(buff);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          switch (estado){

          /*---------------------------------- MAQUINA DE ESTADOS --------------------------
-----------*/

          case 100:                                                    // Estado donde
espera a recibir la señal del pc para empezar el proceso

          //Comunicacion del archivo python con el microcontrolador
          while (HAL_UART_Receive (&huart3, buffer_mensa_received, 1, 500) != HAL_OK){
                // Tiene que esperar a que se reciba algo. No se hace nada mientras no se
reciba
          }

                var1 = buffer_mensa_received[0] - 0x30;
                            // Necesario para pasar de ASCII a decimal
                sprintf(buff,"Estado inical X= %f, Y=%f Gamma=%f \n",X,Y, Gamma);
                Print_Usart3(buff);/*Print_Usart2(buff);*/
//              HAL_Delay(3000);
                switch(var1){
                case 0:
                        X_dest = 0;
                        Y_dest = 0;
                        estado = 101;
                        break;
                case 1:
                        X_dest = -400;
                        Y_dest = 2500;
                        estado = 101;
                        break;

                case 2:
                        X_dest = 600;
                        Y_dest = 2100;
                        estado = 101;
                        break;
                case 3:
```

```
                        X_dest = -500;
                        Y_dest = 1400;
                        estado = 101;
                        break;

        }

        break;


        case 101:

                                                    // Segun la orden que ha recibido del
pc, hace el pitagoras para calcular el ángulo que ha de rotar y la hipotenusa que ha de
recorrer
                hipotenusa = sqrt(pow(X_dest-X, 2) + pow(Y_dest-Y, 2));
                alfa =  atan((X_dest-X)/(Y_dest-Y));

                            // Si es negativo tiene que girar a la izquierda, si es
positivo gira a la derecha.
                if (alfa > 0){ Sent_Rota = 1;}
                else {Sent_Rota = 2;}
                alfa = alfa * 180/Pi;

                                            // Se pasa a grados
                sprintf(buff,"Se mueve un angulo de %f y una distancia de %f \n", alfa,
hipotenusa);
                Print_Usart3(buff);
                RotaRobot(Sent_Rota,fabs(alfa));

                                        // Gira el sentido y el ángulo calculados.
                HAL_Delay(1000);
                sprintf(buff,"Esta en el punto X= %f, Y=%f Gamma=%f \n",X,Y, Gamma);
                Print_Usart3(buff);Print_Usart2(buff);
                estado = 102;

                                                    // Cambia de estado en el que
comienza el movimiento hacia adelante
                break;

        case 102:

                                                        // ESTADO AVANZAR ROBOT
ADELANTE
                Print_Usart3("Comienza el movimiento adelante...\n");
                flag_correccion_pwm = 1;

                                            // Para activar el PID
                //Hace un arranque suave hasta llegar al PWM deseado
                Arranque_Suave();
                Print_Usart3("Termina arranque suave.\n");

////////////////////////////////////////////////////////////////////////////////////////
/
                ///// La rueda izquierda se mueve con la interrupcion, que ajusta el PWM
mediante el PID /////

////////////////////////////////////////////////////////////////////////////////////////
/

                estado = 103;

                                                    // Cambia a siguiente estado
                Step(60,1);
                break;

        case 103:
```

```
                                                                    // GIRO DE MOTOR STEP Y
MEDICION DE SENSOR Y ENCODER
                    sprintf(buff,"Movimiento del step y medicion de sensor: \n \n");
                    Print_Usart3(buff);
                    switch (Sent){
                    case 1:
                            Sent = 0;
                            break;
                    case 0:
                            Sent = 1;
                            break;
                    }

                    for(int k =0; k<10;k++){
                           Step(60,Sent);

                                                                // Hace que de 3 vueltas el motor y
una vuelta el sensor con 10  mediciones
                           measure[k]= ToFMeasure();
                           if (measure[k] >= 130){
                                   if (measure[k] < min_distance && flag_measure == 0){

                  // Se encuentra un objeto a una distancia menor a la determinada
                                                flag_measure = 1;

                                                        // Flag para evitar que cuente 2
objetos cercanos en vez de 1
                                                Dist_obj = measure[k];
                                                Zone_Object = k;
                                                /* ---------------- IMPRIME POR PANTALLA
POSICION DEL OBJETO Y LA DISTANCIA A LA QUE SE ENCUENTRA ----------------------------------
*/
                                                sprintf(buff,"OBJETO DETECTADO EN LA ZONA %d
a %dmm con CW=%d \n\n\n",Zone_Object+1, Dist_obj, Sent);
                                                Print_Usart3(buff);
                                                ////////////////////////////////  Prueba-- Se
para el robot para ver donde ha detectado el objeto  ////////////////////////////////////
                                                flag_correccion_pwm = 0;

//////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
                                                estado = 105;
                                           }
                           }


                   if (Y >= Y_dest){

                                                                // En caso de que haya llegado al
punto requerido
                                    estado = 104;
                                    flag_measure = 1;

                                    }

                   }
                   sprintf(buff,"Medidas son: %d, %d, %d, %d, %d, %d, %d, %d, %d, %d
\n",measure[0], measure[1], measure[2], measure[3],measure[4], measure[5], measure[6],
measure[7],measure[8], measure[9]);
                   Print_Usart3(buff);
                   sprintf(buff,"Esta en el punto X= %f, Y=%f, Gamma=%f \n",X,Y, Gamma);
                   Print_Usart3(buff);
                   break;

           case 104:                                                    // ROBOT HA LLEGADO
AL PUNTO OBJETIVO
                   flag_correccion_pwm =0;
```

```
                LeftWheel(0,0);

                                                // Se para el carrito
                RightWheel(0,0);

                                                // Se para el carrito
                sprintf(buff,"Ha llegado al objetivo X= %f, Y=%f, Gamma =%f \n",X,Y,
Gamma);
                Print_Usart3(buff);Print_Usart2(buff);
                Gamma_turn = (Gamma - Pi/2) *(180/Pi);

                sprintf(buff,"Rota para volver al frente un angulo de Gamma =%f \n",
Gamma_turn);
                Print_Usart3(buff);
                HAL_Delay(1000);
                if(Gamma_turn < 0){RotaRobot(2, fabs(Gamma_turn));}
                else {RotaRobot(1,fabs( Gamma_turn));}
                HAL_Delay(500);

                sprintf(buff,"Ya ha girado. esta en X= %f, Y=%f, Gamma =%f \n",X,Y, Gamma);
                Print_Usart3(buff);Print_Usart2(buff);

                // Manda comando de que ha finalizado el programa
                HAL_Delay(3000);
                for(i = 0; i<3; i++){
                        sprintf(buff,"Q");
                        Print_Usart3(buff);
                        HAL_Delay(2000);
                }

                estado = 100;
                break;

        case 105:                                                       // ENTRA EN MODO
EVITAR OBSTACULO
                flag_correccion_pwm =0;
                HAL_Delay(500);
                sprintf(buff,"SE ESQUIVA EL OBJETO: \n");
                Print_Usart3(buff);Print_Usart2(buff);
                AvoidObj();
                HAL_Delay(500);
                //------------//
                LeftWheel(0,0);

                                        // Se para el carrito
                RightWheel(0,0);
                sprintf(buff,"FIN DE MANIOBRA DE ESQUIVAR, X= %f, Y=%f, Gamma = %f \n", X,
Y, Gamma);
                HAL_Delay(4000);
                Print_Usart3(buff); Print_Usart2(buff);
                flag_measure = 0;                       //Resetea el flag de detectar objetos
otra vez
                estado = 101;

                break;

        case 106:
                sprintf(buff,"ERROR EN EL SENSOR TOF \n \n");
                Print_Usart3(buff);
                while(1){};


        }               // Fin del switch
```

```
//
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }     //FIN DEL WHILE(1)
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim7){
        uint8_t timebuffer[220];
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_12);                  //Comprobar cuando entra en
la interrupcion
        encoder_der = __HAL_TIM_GetCounter (&htim3) + 5000;
                                                                //Encoder rueda
derecha
```

```
        encoder_izq= __HAL_TIM_GetCounter (&htim4) + 5000;
                                                                    // Encoder rueda
izquierda.------- Se pone +5000 para poder hacer giros al iniciar el movimiento
        msecTOT = RTC_MilisecTOT();
        difer_encoder_izq = encoder_izq - encoder_izq_ant;
        difer_encoder_der = encoder_der - encoder_der_ant;
        difer_tiempo = msecTOT - msecTOT_ant;
        W_rueda_izq = (((float)difer_encoder_izq) * K_pul_rad) / difer_tiempo;
        W_rueda_der = (((float)difer_encoder_der) * K_pul_rad) / difer_tiempo;
        difereniia = W_rueda_izq - W_rueda_der;
        //Velocidades lineales
        V_rd = (float) Radio_rueda * W_rueda_der;               // En mm por milisegundo
        V_ri = (float) Radio_rueda * W_rueda_izq;               // En mm por milisegundo
        V_inst = (V_rd+V_ri)/2;
        // Velocidad de rotacion
        W_rot = (V_rd-V_ri)/Base;
        //Actualizacion de la posicion
        Gamma = Gamma + W_rot * (float)difer_tiempo;
        if(Gamma > 2*Pi){Gamma =0;}
        if(Gamma <0){ Gamma = 2*Pi;}
        X = X + V_inst * (float)cos(Gamma)*(float)difer_tiempo;
        Y = Y + V_inst * (float)sin(Gamma)*(float)difer_tiempo;
        ////////////////////  Implatancion del PID para control de la rueda izquierda
(esclava) en moviemiento recto ////////////////////////
        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
        if(flag_correccion_pwm == 1){
          error_Vel = W_rueda_der - W_rueda_izq;
                                // Error de velocidades angulares de la rueda derecha y la
rueda izquierda. Referencia - salida
          error_Vel_Deriv = (error_Vel - error_Vel_ant)/difer_tiempo;
                // Error de velocidades derivadas respecto el tiempo. Necesario para la
accion derivativa
          suma_integrativ += (error_Vel * difer_tiempo);
                                                          // Suma integrativa del error de
velocidades. Necesario para la accion integral
          pwmLW = pwmLW + (int)(error_Vel *Kp + (error_Vel_Deriv/1000* Kd)+ (Ki *
suma_integrativ));      // Suma de todas las acciones de control del PID
          LeftWheel(1,pwmLW);
                                                                                    //
Se aplica la accion de control obtenida
          msecTOT_ant = msecTOT;
          error_Vel_ant = error_Vel;                            // almacena el valor del
ultimo error de velocidad para poder hacer la parte derivativa del PID

        }
          encoder_izq_ant = encoder_izq;
          encoder_der_ant = encoder_der;

}

////////// INTERRUPCION PARA ENSAYO DE CONTAR PULSOS DE ENCODER POR VUELTA DE RUEDA
//////////////////////
//void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim8){
//      uint8_t timebuffer[220];
////                                    encoder_der = __HAL_TIM_GetCounter (&htim3);
//                                      encoder_izq= __HAL_TIM_GetCounter (&htim4);
//                                      difer_encoder_izq = encoder_izq - encoder_izq_ant;
//
//                                sprintf(timebuffer, "Vuelta %d, ---Pulsos: %d \n", i,
difer_encoder_izq);
//                                flag = 1;
//                                Print_Usart2(timebuffer);
////                                encoder_der_ant= encoder_der  ;
//                                encoder_izq_ant= encoder_izq  ;
//                                i++;
//
////                             sprintf((char*)timebuffer,"Sensor encontrado \n");
```

```
////                            Print_Usart2(timebuffer);
//
//}

void Print_Usart2(uint8_t *texto)
{
                if(HAL_UART_Transmit(&huart2, texto, (uint16_t) strlen(texto), 100)!= HAL_OK)
                {
                        Error_Handler2();
                }
                //HAL_Delay (50); // si se usa en CALLBACK el Delay cuelga el micro.
                return;
}
void Print_Usart3(uint8_t *texto)
{
                if(HAL_UART_Transmit(&huart3, texto, (uint16_t) strlen(texto), 100)!= HAL_OK)
                {
                        Error_Handler3();
                }
                //HAL_Delay (50); // si se usa en CALLBACK el Delay cuelga el micro.
                return;
}

void Error_Handler2(void)
{
  /* USER CODE BEGIN Error_Handler */
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
void Error_Handler3(void)
{
  /* USER CODE BEGIN Error_Handler */
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
int RTC_MilisecTOT(void)
        {
                uint8_t showtime[32];
                uint8_t showdate[32];
                RTC_TimeTypeDef sTime;
                RTC_DateTypeDef sDate;
                /* Get the RTC current Time */
                HAL_RTC_GetTime(&hrtc, &sTime, FORMAT_BIN);
                /* Get the RTC current Date */
                HAL_RTC_GetDate(&hrtc, &sDate, FORMAT_BIN);
                /* Display time Format : hh:mm:ss */
                uint32_t msec = 1000 * (sTime.SecondFraction - sTime.SubSeconds) /
(sTime.SecondFraction + 1);
                uint32_t msecTOT = ((((sTime.Hours)*60 + sTime.Minutes)*60 + sTime.Seconds
)*1000) + msec;
//              sprintf((char*)showtime,"TIME: %0.2d:%0.2d:%0.2d:%0.4d,-- %0.4d ms -- %0.6d
ms TOT \n",sTime.Hours,
//              sTime.Minutes, sTime.Seconds, sTime.SubSeconds, msec, msecTOT);
//              Print_Usart2((char*)showtime);
                return msecTOT;
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
```

```
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

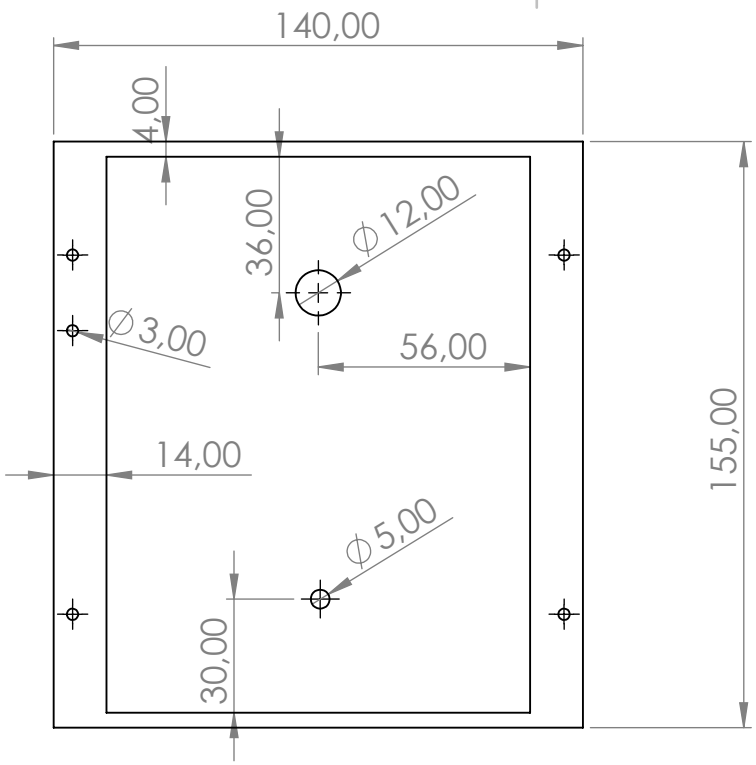
19.3 Planos

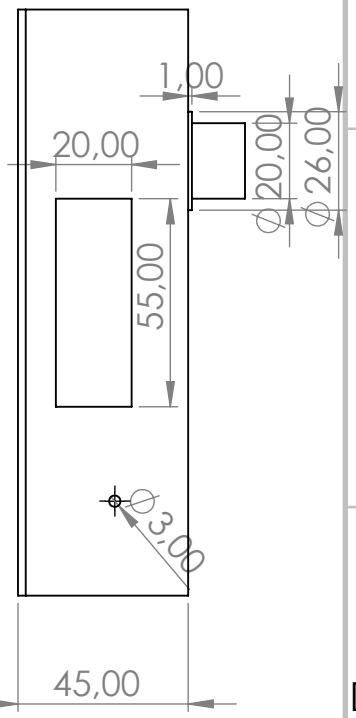
Los planos se adjuntan al final de la memoria porque son exportados desde SolidWorks y de esta forma no pierden calidad de imagen.

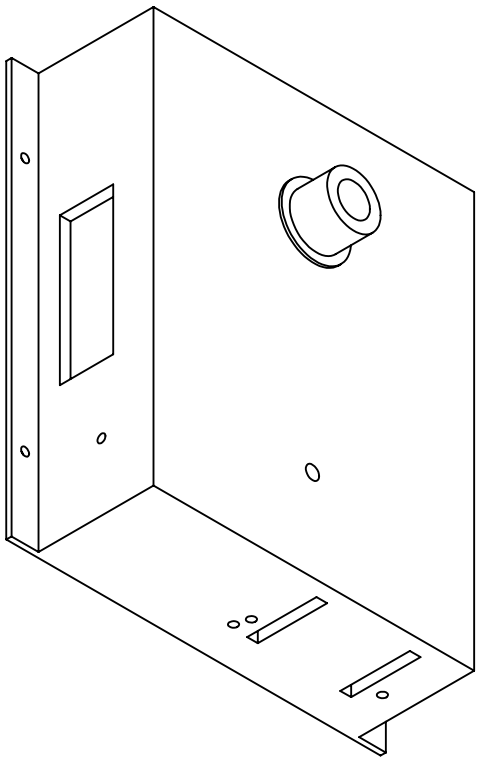
19.4 Datasheets

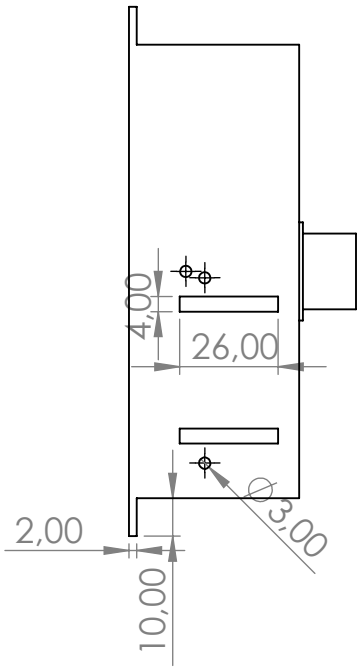
Estos documentos son adjuntados al final de la memoria puesto que no son realizados por el autor del proyecto, sino que son descargados desde las webs de los fabricantes de los componentes utilizados.

PLANOS

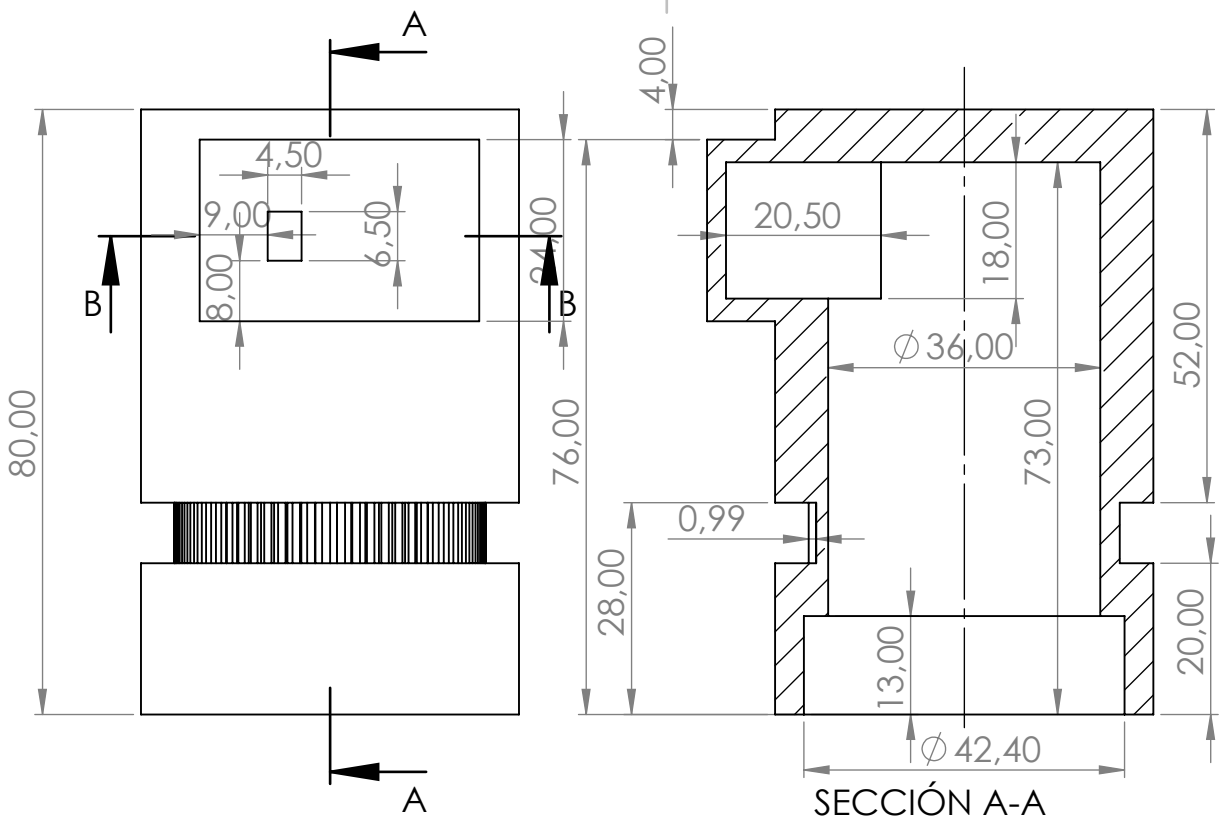




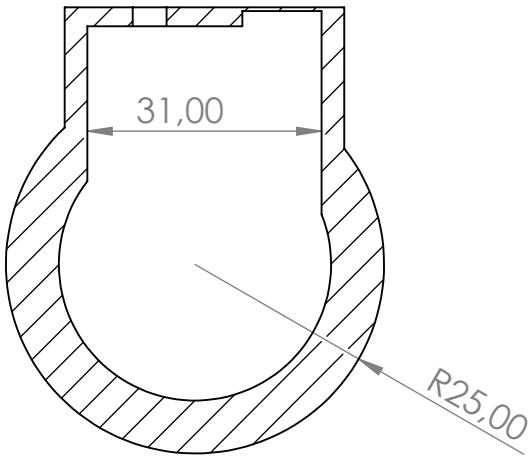




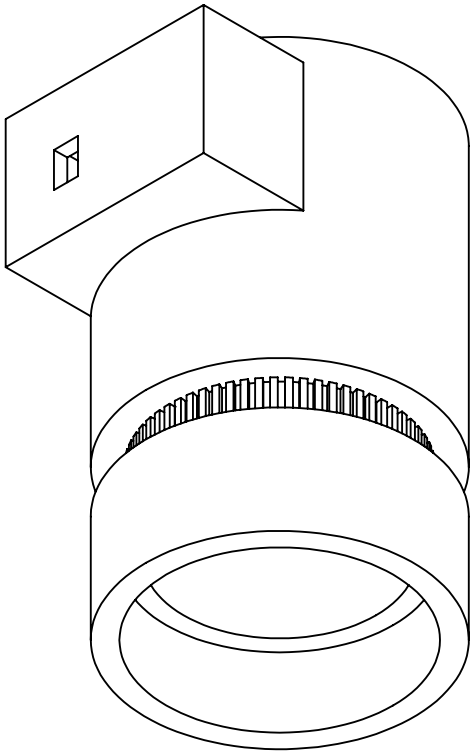
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: XXX
DISEÑADO POR Álvaro	UNIVERSITAT POLITÈCNICA DE VALÈNCIA Escuela Técnica Superior de Ingeniería del Diseño
FECHA 29/08/21	TÍTULO: Cubierta
Trabajo Fin De Máster Máster en Ingeniería Mecánica	Autor Álvaro Serna Pérez ESCALA:1:2 HOJA 1 DE 1
	A4



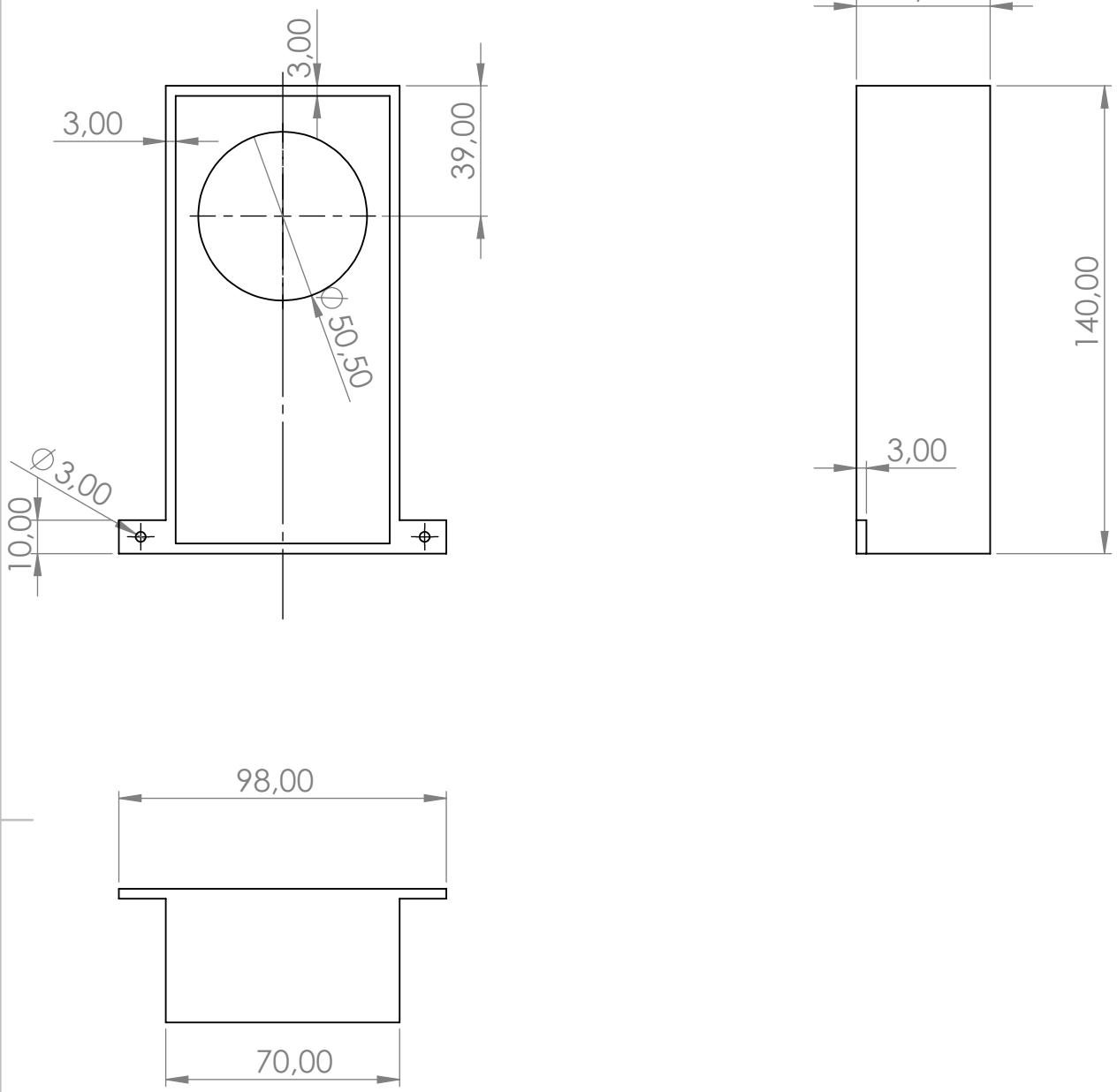
SECCIÓN A-A



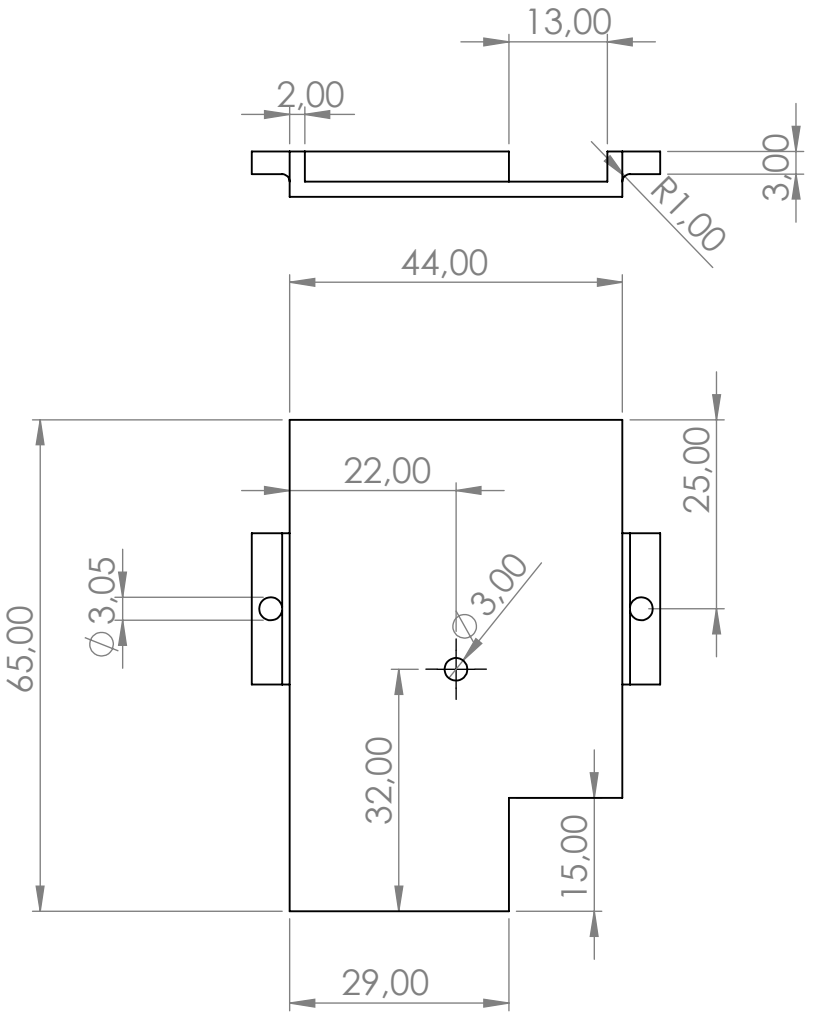
SECCIÓN B-B

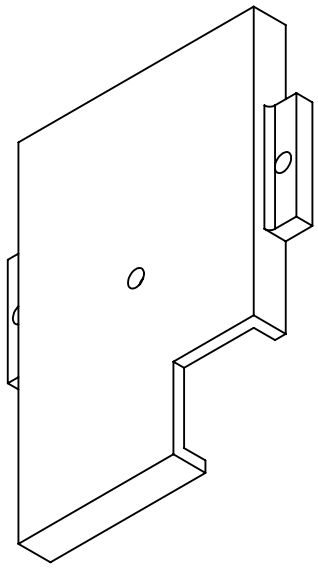


SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: XXX	UNIVERSITAT POLITÈCNICA DE VALÈNCIA Escuela Técnica Superior de Ingeniería del Diseño
DISEÑADO POR Álvaro		TÍTULO: CubiertaSensor
FECHA 29/08/21		
Trabajo Fin De Máster	Autor Álvaro Serna Pérez	A4
Máster en Ingeniería Mecánica	ESCALA:1:1	HOJA 1 DE 1

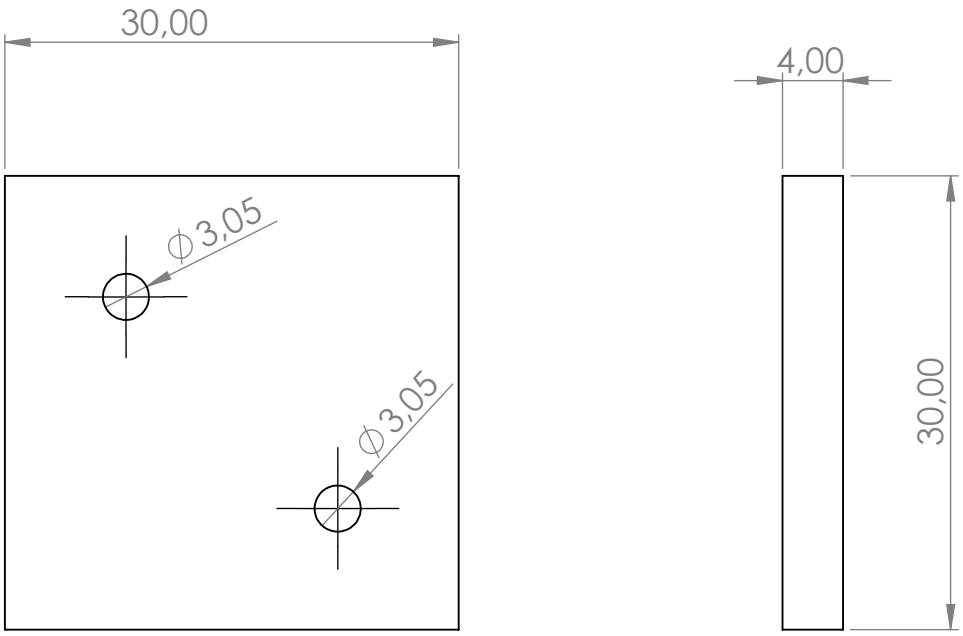


SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: XXX
DISEÑADO POR: Álvaro	UNIVERSITAT POLITÈCNICA DE VALÈNCIA Escuela Técnica Superior de Ingeniería del Diseño
FECHA: 29/08/21	TÍTULO: Cubre-Polea
Trabajo Fin De Máster Máster en Ingeniería Mecánica	Autor: Álvaro Serna Pérez ESCALA:1:2 HOJA 1 DE 1
	A4





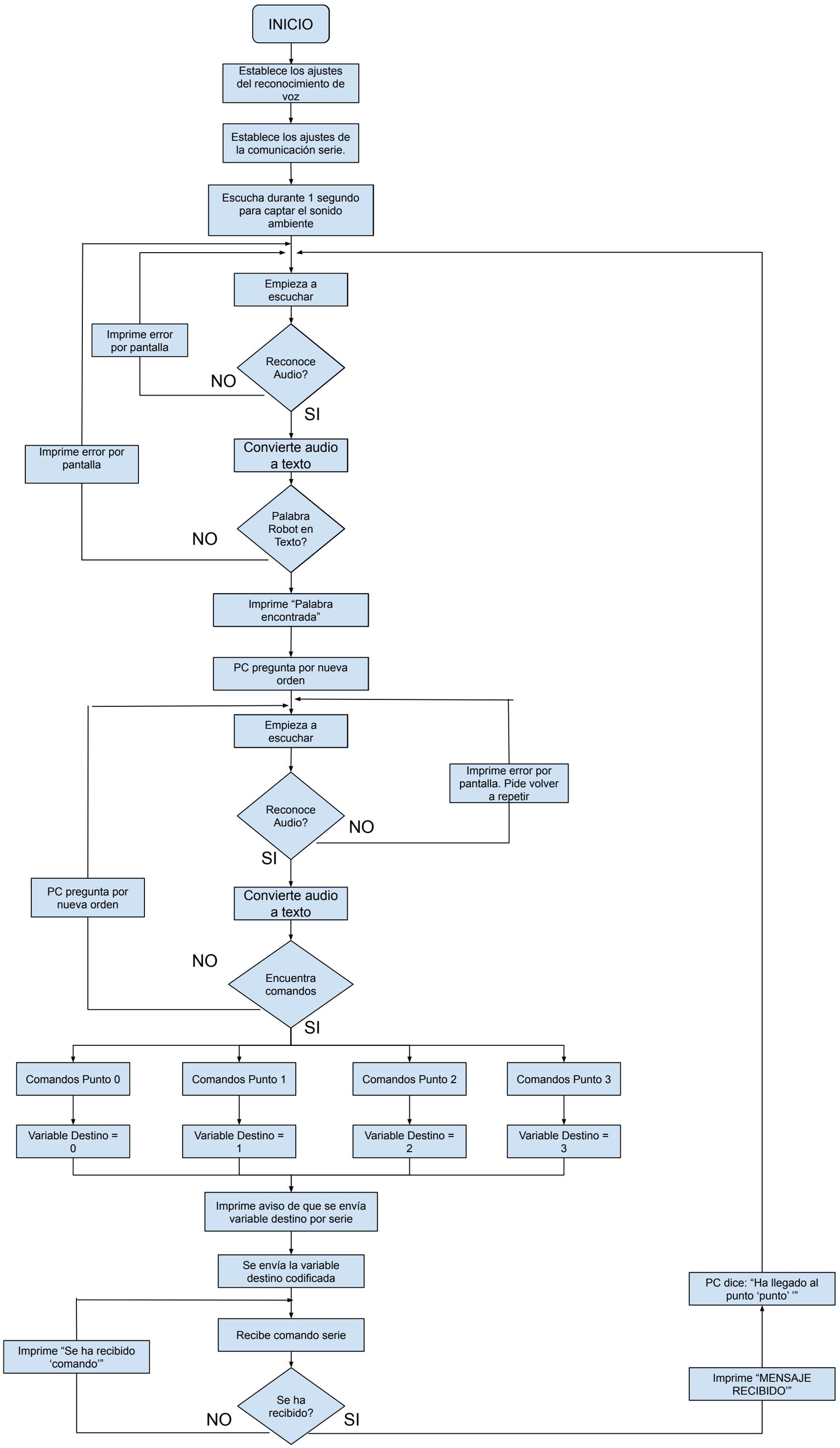
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: XXX	UNIVERSITAT POLITÈCNICA DE VALÈNCIA Escuela Técnica Superior de Ingeniería del Diseño
DISEÑADO POR Álvaro	TÍTULO: Radio Holder	
FECHA 29/08/21		
Trabajo Fin De Máster	Autor Álvaro Serna Pérez	A4
Máster en Ingeniería Mecánica	ESCALA:1:1	HOJA 1 DE 1



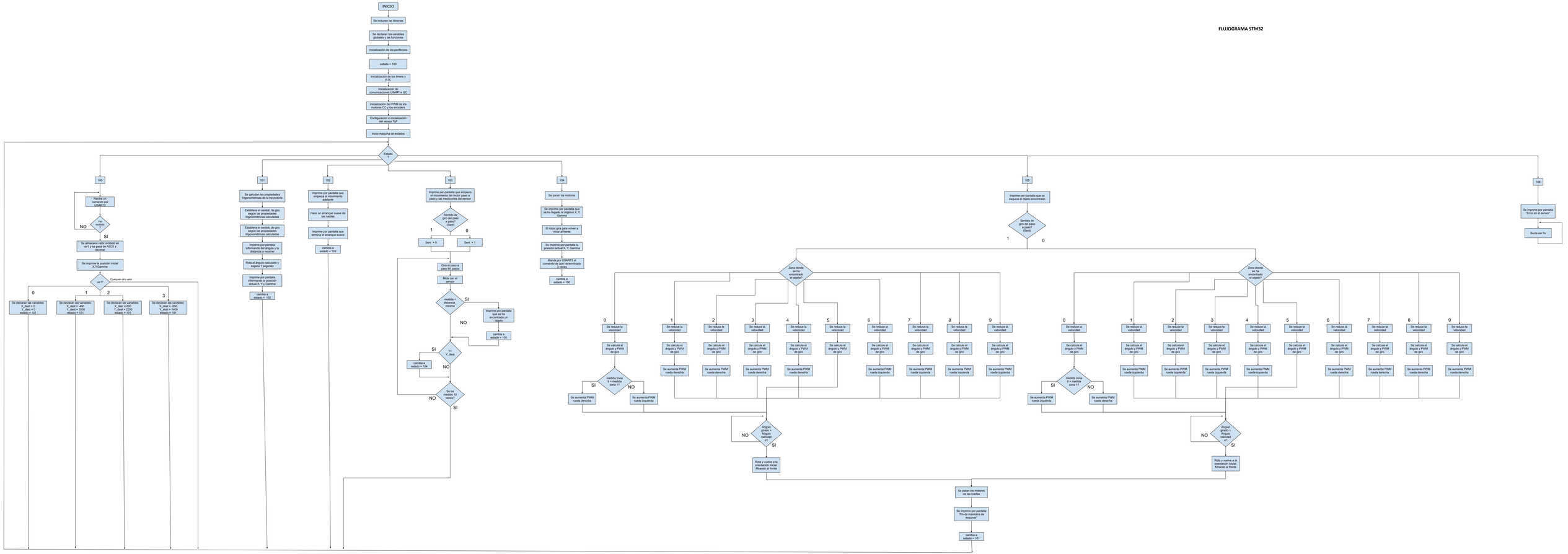
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: XXX	UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Escuela Técnica Superior de Ingeniería del Diseño
DISEÑADO POR Álvaro		TÍTULO:	
FECHA 29/08/21		Soporte motor	
Trabajo Fin De Máster		Autor Álvaro Serna Pérez	A4
Máster en Ingeniería Mecánica		ESCALA:2:1	HOJA 1 DE 1

FLUJOGRAMAS

FLUJOGRAMA PYTHON



FLUJOGRAMA STM32

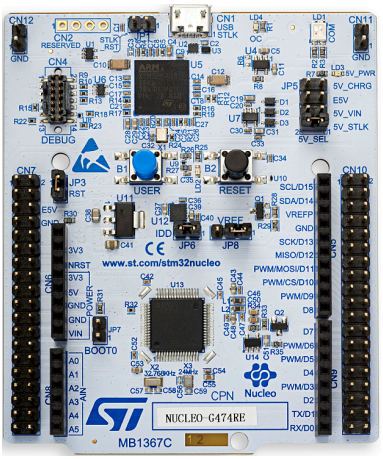


DATASHEETS

NUCLEO-XXXXRX NUCLEO-XXXXRX-P

Data brief

STM32 Nucleo-64 boards



NUCLEO-G474RE example. Boards with different references show different layouts. Picture is not contractual.

Product status link
NUCLEO-XXXXRX
NUCLEO-F030R8, NUCLEO-F070RB, NUCLEO-F072RB, NUCLEO-F091RC, NUCLEO-F103RB, NUCLEO-F302R8, NUCLEO-F303RE, NUCLEO-F334R8, NUCLEO-F401RE, NUCLEO-F410RB, NUCLEO-F411RE, NUCLEO-F446RE, NUCLEO-G070RB, NUCLEO-G071RB, NUCLEO-G0B1RE, NUCLEO-G431RB, NUCLEO-G474RE, NUCLEO-G491RE, NUCLEO-L010RB, NUCLEO-L053R8, NUCLEO-L073RZ, NUCLEO-L152RE, NUCLEO-L452RE, NUCLEO-L476RG.
NUCLEO-XXXXRX-P
NUCLEO-L412RB-P, NUCLEO-L433RC-P, NUCLEO-L452RE-P.

Features

- Common features
  - STM32 microcontroller in LQFP64 package
  - 1 user LED shared with ARDUINO®
  - 1 user and 1 reset push-buttons
  - 32.768 kHz crystal oscillator
  - Board connectors:
    - ARDUINO® Uno V3 expansion connector
    - ST morpho extension pin headers for full access to all STM32 I/Os
  - Flexible power-supply options: ST-LINK, USB V_{BUS}, or external sources
  - On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port and debug port
  - Comprehensive free software libraries and examples available with the STM32Cube MCU Package
  - Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE
- Board-specific features
  - External SMPS to generate V_{core} logic supply
  - 24 MHz HSE
  - Board connectors:
    - External SMPS experimentation dedicated connector
    - Micro-AB or Mini-AB USB connector for the ST-LINK
    - MIPI® debug connector
  - Arm® Mbed Enabled™ compliant

Description

The STM32 Nucleo-64 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the external SMPS significantly reduces power consumption in Run mode.

The ARDUINO® Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

The STM32 Nucleo-64 board does not require any separate probe as it integrates the ST-LINK debugger/programmer.

The STM32 Nucleo-64 board comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package.

1 Ordering information

To order an STM32 Nucleo-64 board, refer to Table 1. For a detailed description of each board, refer to its user manual on the product web page. Additional information is available from the datasheet and reference manual of the target STM32.

Table 1. List of available products

Order code	Board reference	User manual	Target STM32	Differentiating features
NUCLEO-F030R8	MB1136	UM1724	STM32F030R8T6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F070RB			STM32F070RBT6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F072RB			STM32F072RBT6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F091RC			STM32F091RCT6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F103RB			STM32F103RBT6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F302R8			STM32F302R8T6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F303RE			STM32F303RET6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F334R8			STM32F334R8T6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F401RE			STM32F401RET6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F410RB			STM32F410RBT6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F411RE			STM32F411RET6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-F446RE			STM32F446RET6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-G070RB	MB1360	UM2324	STM32G070RBT6	• ST-LINK/V2-1 on Micro-AB USB connector
NUCLEO-G071RB			STM32G071RBT6	• ST-LINK/V2-1 on Micro-AB USB connector

Order code	Board reference	User manual	Target STM32	Differentiating features
NUCLEO-G0B1RE	MB1360	UM2324	STM32G0B1RET6	• ST-LINK/V2-1 on Micro-AB USB connector
NUCLEO-G431RB	MB1367	UM2505	STM32G431RBT6U	• STLINK-V3E on Micro-AB USB connector • 24 MHz HSE • MIPI® debug connector
NUCLEO-G474RE			STM32G474RET6U	• STLINK-V3E on Micro-AB USB connector • 24 MHz HSE • MIPI® debug connector
NUCLEO-G491RE			STM32G491RET6U	• STLINK-V3E on Micro-AB USB connector • 24 MHz HSE • MIPI® debug connector
NUCLEO-L010RB	MB1136	UM1724	STM32L010RBT6	• ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-L053R8			STM32L053R8T6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-L073RZ			STM32L073RZT6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-L152RE			STM32L152RET6	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-L412RB-P	MB1319	UM2206	STM32L412RBT6PU	• ST-LINK/V2-1 on Micro-AB USB connector • External SMPS
NUCLEO-L433RC-P			STM32L433RCT6PU	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Micro-AB USB connector • External SMPS
NUCLEO-L452RE	MB1136	UM1724	STM32L452RET6U	• ST-LINK/V2-1 on Mini-AB USB connector
NUCLEO-L452RE-P	MB1319	UM2206	STM32L452RET6PU	• ST-LINK/V2-1 on Micro-AB USB connector • External SMPS
NUCLEO-L476RG	MB1136	UM1724	STM32L476RGT6U	• Arm® Mbed Enabled™ • ST-LINK/V2-1 on Mini-AB USB connector

1.1 Product marking

The sticker located on the top or bottom side of the PCB board shows the information about product identification such as board reference, revision, and serial number.

The first identification line has the following format: “MBxxxx-Variant-yzz”, where “MBxxxx” is the board reference, “Variant” (optional) identifies the mounting variant when several exist, "y" is the PCB revision and "zz" is the assembly revision: for example B01.

The second identification line is the board serial number used for traceability.

Evaluation tools marked as “ES” or “E” are not yet qualified and therefore not ready to be used as reference design or in production. Any consequences deriving from such usage will not be at ST charge. In no event, ST will be liable for any customer usage of these engineering sample tools as reference designs or in production.

“E” or “ES” marking examples of location:

- On the targeted STM32 that is soldered on the board (For an illustration of STM32 marking, refer to the STM32 datasheet “Package information” paragraph at the *www.st.com* website).
- Next to the evaluation tool ordering part number that is stuck or silk-screen printed on the board.

Some boards feature a specific STM32 device version, which allows the operation of any bundled commercial stack/library available. This STM32 device shows a "U" marking option at the end of the standard part number and is not available for sales.

In order to use the same commercial stack in his application, a developer may need to purchase a part number specific to this stack/library. The price of those part numbers includes the stack/library royalties.

1.2 Codification

The meaning of the codification is explained in Table 2.

Table 2. Codification explanation

NUCLEO-XXYYRT NUCLEO-XXYYRT-P	Description	Example: NUCLEO-L452RE
XX	MCU series in STM32 Arm Cortex MCUs	STM32L4 Series
YY	MCU product line in the series	STM32L452
R	STM32 package pin count	64 pins
T	STM32 Flash memory size: • 8 for 64 Kbytes • B for 128 Kbytes • C for 256 Kbytes • E for 512 Kbytes • G for 1 Mbyte • Z for 192 Kbytes	512 Kbytes
-P	STM32 has external SMPS function	No SMPS

2 Development environment

2.1 System requirements

- Windows® OS (7, 8 and 10), Linux® 64-bit, or macOS®
- USB Type-A or USB Type-C® to Micro-B cable, or USB Type-A or USB Type-C® to Mini-B cable (depending on the board reference)

Note: macOS® is a trademark of Apple Inc. registered in the U.S. and other countries.
All other trademarks are the property of their respective owners.

2.2 Development toolchains

- IAR Systems - IAR Embedded Workbench®(1)
- Keil® - MDK-ARM(1)
- STMicroelectronics - STM32CubeIDE
- Arm® - Mbed Studio(2) (3)

1. *On Windows® only.*
2. *Arm and Mbed are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and or elsewhere.*
3. *Refer to the os.mbed.com website and to the "Ordering information" section to determine which order codes are supported.*

2.3 Demonstration software

The demonstration software, included in the STM32Cube MCU Package corresponding to the on-board microcontroller, is preloaded in the STM32 Flash memory for easy demonstration of the device peripherals in standalone mode. The latest versions of the demonstration source code and associated documentation can be downloaded from *www.st.com*.

Revision history

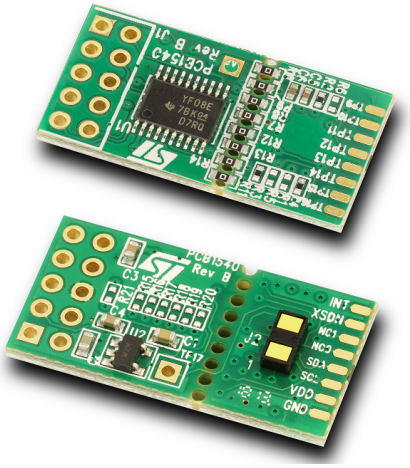
Table 3. Document revision history

Date	Version	Changes
10-Feb-2014	1	Initial release.
13-Feb-2014	2	Added *Table 1: Device summary* and updated *Table 2: Ordering information*.
11-Apr-2014	3	Extended the applicability to NUCLEO-F302R8. Updated *Table 1: Device summary* and *Table 2: Ordering information*.
26-May-2014	4	Extended the applicability to NUCLEO-L053R8, NUCLEO-F072RB, NUCLEO-F334R8 and NUCLEO-F411RE. Updated *Table 1* and *Table 2*.
9-Sep-2014	5	Extended the applicability to NUCLEO-F091RC and NUCLEO-F303RE. Updated *Features*. Updated *Table 1: Device summary* and *Table 2: Ordering information*.
16-Dec-2014	6	Extended the applicability to NUCLEO-F070RB, NUCLEO-L073RZ and NUCLEO-L476RG. Updated *Table 1: Device summary* and *Table 2: Ordering information*.
8-Jul-2015	7	Extended the applicability to NUCLEO-F410RB, NUCLEO-F446RE. Updated *Table 1: Device summary* and *Table 2: Ordering information*.
29-Nov-2016	8	Extended the applicability to NUCLEO-L452RE. Updated *Table 1: Device summary* and *Table 2: Ordering information*. Added *Table 3: Codification explanation*.
16-Nov-2017	9	Extended document scope to the NUCLEO-L452RE-P and NUCLEO-L433RC-P boards: • Updated *Features* • Updated *Table 1: Device summary*, *Table 2: Ordering information* and *Table 3: Codification explanation* • Updated *System requirement*, *Development toolchains* and *Demonstration software*
15-Dec-2017	10	Updated *Features*, *Description* and *System requirement*. Extended document scope to the NUCLEO-L010RB board: updated *Table 1: Device summary* and *Table 2: Ordering information*.
24-Aug-2018	11	Extended document scope to the NUCLEO-L412RB-P board: updated *Table 1: Device summary* and *Table 2: Ordering information*.
22-Oct-2018	12	Extended document scope to the NUCLEO-G070RB and NUCLEO-G071RB boards: • Updated *Table 1: Device summary* and *Table 2: Ordering information* • Added NUCLEO-GXXXRX top view on the cover page
8-Apr-2019	13	Revised the entire document to accommodate to multiple feature combinations: • Reorganized *Features* • Updated *Description* • Added *Ordering information* and *Development environment* • Updated *Table 1. List of available products* and *Table 2. Codification explanation* Extended document scope to the NUCLEO-G431RB and NUCLEO-G474RE boards.
25-Oct-2020	14	Extended document scope to the NUCLEO-G0B1RE and NUCLEO-G491RE: updated List of available products.

VL53L1X-SATEL

Data brief

Breakout boards based on the VL53L1X long distance ranging Time-of-Flight sensor



Features

- Two breakout boards, integrating:
  - VL53L1X long distance ranging Time-of-Flight (ToF) sensor
  - Regulator: 5 to 2.8 V range input voltage (output voltage: 2.8 V)
  - VL53L1X signal interface level shifter
- True distance measurement independent of target size and reflectance
- Divisible board enabling use as mini PCB breakout board, easy to integrate in customer device

Description

The VL53L1X-SATEL breakout boards can be used for easy integration into customer devices.

Thanks to the voltage regulator and level shifters, the VL53L1X breakout boards can be used in any application with a 2.8 V to 5 V supply.

The PCB section supporting the VL53L1X module is perforated so that developers can break off the mini PCB for use in a 2.8 V supply application using flying leads. This makes it easier to integrate the VL53L1X-SATEL breakout boards into development and evaluation devices due to their small form factor.

Product status link
VL53L1X-SATEL

1 VL53L1X overview

The VL53L1X is the latest product based on ST's patented FlightSense™ technology. This is a ground-breaking technology allowing absolute distance to be measured independent of target reflectance. Instead of estimating the distance by measuring the amount of light reflected back from an object (which is significantly influenced by color and surface), the VL53L1X precisely measures the time the light takes to travel to the nearest object and reflect back to the sensor (Time-of-Flight).

Combining an IR emitter and a range sensor, the VL53L1X is easy to integrate and saves OEMs long and costly optical and mechanical design optimizations.

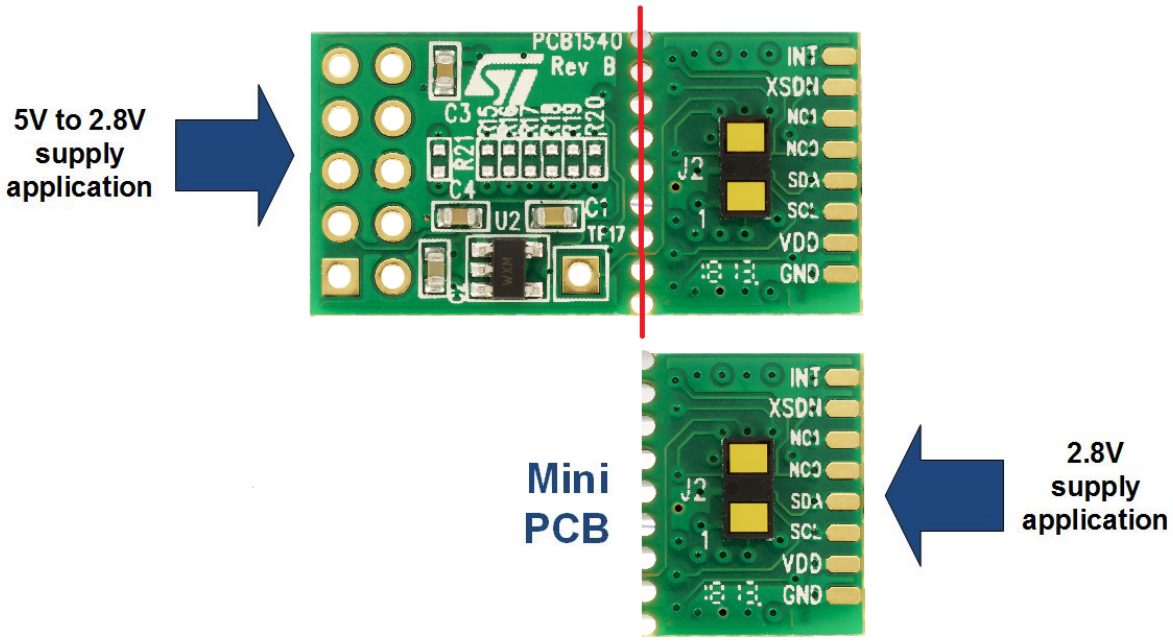
The module is designed for low-power operations. Ranging measurements can be automatically performed at user defined intervals. Multiple threshold and interrupt schemes are supported to minimize host operations.

Note: The VL53L1X is delivered with a liner, to prevent potential foreign material to penetrate inside the module holes during the assembly process. The liner must be removed at the latest possible step during final assembly and before module calibration.

2 Breakable board

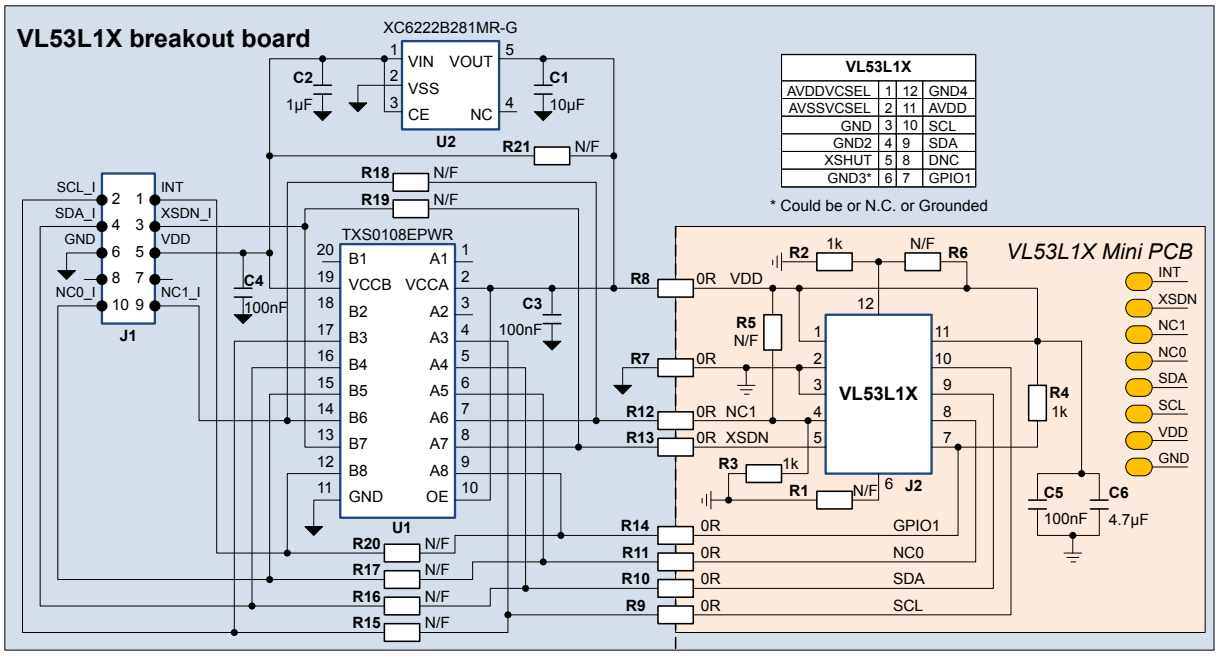
For 2.8 V supply applications, the breakout boards can be broken along the red dotted line as shown in the figure below, to use the “mini PCB”. This set up is easier to integrate into a customer device due to its small form factor.

Figure 1. Breakout board layout



3 Schematic and list of materials

Figure 2. **Satellite schematic and list of materials**



4 Ordering information

Table 1. **Ordering information**

Order code	Description
VL53L1X-SATEL	Two VL53L1X breakout boards

Revision history

Table 2. Document revision history

Date	Version	Changes
03-May-2018	1	Initial release
20-Feb-2019	2	Updated cover image and Figure 1. Breakout board layout.