

UNIVERSIDAD NACIONAL DE TUCUMÁN

FACE y T

Departamento de Electricidad, Electrónica y Computación

Laboratorio de Técnicas Digitales



TRABAJO DE GRADUACIÓN, INFORME FINAL

DISEÑO DE UN ROBOT OMNIDIRECCIONAL PARA EL TRANSPORTE DE CARGAS PEQUEÑAS

Autores:

MANSILLA, Gonzalo Nicolás	DNI:40215659
PONCE, Luis Ariel	DNI:40088805

Carrera: Ingeniería Electrónica

Tutor:

TEODOVICH SOSA, Lorgio José

Fecha de presentación: 08/2024

FICHA DE PRESENTACIÓN DE PROYECTOS **FINALES**

Nombre del proyecto: “Diseño de un robot omnidireccional para el transporte de cargas pequeñas”

Apellidos y nombres de los alumnos:

- Mansilla, Gonzalo Nicolas
- Ponce, Luis Ariel

DNI:

- 40215659
- 40088805

Objetivos: Diseñar un prototipo de vehículo autónomo para el transporte de cargas dentro de la industria

Especificaciones:

- Vehículo omnidireccional con ruedas mecanum impulsado por 4 motores brushed y baterías
- Sistema de posicionamiento y localización
- Lector de QR
- comunicación remota por medio de bluetooth
- manejo a distancia
- capacidad de moverse de manera autónoma

Conceptos técnicos involucrados:

- Sistema de control
- Programación
- Diseño de PCB

Duración: 1 año

Lugar de Realización: Laboratorio de técnicas digitales

Responsable/ Tutor: Ing. Teodovich, Lorgio

Firma:

Fecha:

Email: lteodovich@herrera.unt.edu.ar

Catedra: Electrónica 4

TABLA DE CONTENIDO

1. PROPÓSITOS
2. DESCRIPCIÓN GENERAL
3. MODOS DE FUNCIONAMIENTO
4. SENSORES, MÓDULOS Y PARTES INTERNAS DEL ROBOT
5. MODOS DE COMUNICACIÓN
6. MODELADO Y ANÁLISIS DEL ROBOT
7. SISTEMAS DE CONTROL
8. CIRCUITOS ELÉCTRICOS DEL ROBOT
9. PROGRAMACIÓN DE FIRMWARE
10. ANÁLISIS DE MERCADO
11. INCONVENIENTES, DESAFÍOS Y SOLUCIONES
12. FUTURAS MEJORAS
13. CONCLUSIONES
14. ANEXO I
15. ANEXO II
16. ANEXO III
17. BIBLIOGRAFIA

1. PROPÓSITOS

Necesidades a cumplir

El objetivo del proyecto es diseñar un pequeño robot destinado para el transporte de carga para el mercado argentino. El robot debe ser compacto y con capacidad de mover paquetes livianos de un punto a otro.

Uno de los propósitos del robot es que sea fácil de construir y de mantener en el mercado argentino, un mercado con políticas cambiantes que podrían afectar el abastecimiento de suministros electrónicos. El robot tiene un diseño modular que utiliza principalmente placas comerciales ampliamente disponibles en el mercado, especialmente aquellas relacionadas con el ecosistema "Arduino". Esto facilita el reemplazo y la resolución de problemas en caso de fallas. Este diseño permite que los módulos dentro del robot posean un funcionamiento estilo "plug and play", es decir, que dichas placas solo es necesario conectarlas al robot para que empiecen a funcionar sin necesidad de un paso previo, haciendo que la mano de obra en la construcción y en el mantenimiento del robot no sea complicada.

Un objetivo secundario de este robot, es ayudar a la industrialización del país, ofreciendo un producto local con un servicio técnico ofrecido por ingenieros argentinos. El avance de la industria 4.0 y la robotización que se está dando en las industrias alrededor del mundo atenta contra la competitividad de las industrias argentinas, empresas como Amazon o Ikea ya poseen robotizadas sus líneas de producción. Las empresas que ofrecen este tipo de robotización suelen trabajar con robots de empresas extranjeras que ante un problema de mantenimiento estos robots son desechados o requieren reparaciones costosas ya que es necesario traer el personal de servicio técnico desde el país de origen del robot. En cambio, al ser este robot diseñado y ensamblado a nivel local por especialistas del país por lo que ante un inconveniente se tendrá a disposición a los especialistas que conocerán a detalle todo lo relacionado al funcionamiento del robot.

Motivaciones

La robótica está dando pasos cada vez más agigantados en el mundo actual, acompañada de la inteligencia artificial y los constantes avances en diversos campos tecnológicos. Para todo ello es necesario contar con diferentes herramientas tecnológicas, las cuales son:

Electrónica, comunicaciones, programación y manejo de datos.

En la tesis, una de las cosas que se pensó a la hora de elegir el proyecto era englobar todos los conocimientos adquiridos a lo largo de la carrera de Ingeniería Electrónica y el área de la robótica es una ciencia que aglutina varias ramas tecnológicas relacionadas con esta.

El hacer un robot desde cero también es una motivación. Poder incorporar todas las tecnologías lleva tiempo, el modelar sistemas, ensayarlos, mejorarlos e implementarlos.

El desafío de diseñar un robot que pueda ser una solución importante para una fábrica o empresa nos motiva a pensar todo lo que se puede alcanzar. Este robot puede ser una solución sumamente

rentable para la empresa, ya que puede ser modificado a medida para que sea eficiente en su área de trabajo, de poco mantenimiento y moderada supervisión.

Todo esto permite poder mirar con mucha visión al futuro tecnológico y lo que se puede alcanzar y aprender de todas las nuevas tecnologías.

A medida que los años pasan, nuevos conocimientos tendrán que ser adquiridos para estar actualizados, a la vanguardia y listos para poder afrontar nuevos desafíos como ingenieros.

Hoy en día una de las cosas más importantes es el manejo de datos. Este proyecto maneja una cantidad importante de datos y eso permite poder analizar, procesar datos y tomar acción.

Innovaciones

En el mercado tecnológico, se habla de dos tipos de innovaciones:

- Innovación tecnológica incremental: es aquella en la que se aplican una serie de pequeñas mejoras o actualizaciones en productos, servicios, procesos o métodos que ya existen en la empresa. Es la forma de innovación tecnológica más común que se puede encontrar.
Por lo general, los avances en el mercado tecnológicos son de este tipo y se enfocan en mejorar la eficiencia de desarrollo, la productividad y la diferenciación competitiva de un producto o servicio ya existente. Cabe señalar que este tipo de innovación no genera nuevos mercados, pues los clientes/usuarios ya conocen el producto o servicio
- Innovación tecnológica disruptiva: con este tipo de innovación se crean nuevos productos o servicios que transforman el mercado y generan una red de valor, lo cual ayuda a la empresa a liderar el sector en el cual incursiona.

En este caso se puede decir que este proyecto incursiona en la innovación disruptiva, ya que en el mercado argentino es muy difícil encontrar robots de fabricación argentina, lo cual podría abrir un nuevo nodo de mercado que impulse a otros a adherirse y competir. Además, por su bajo costo y practicidad, se vuelve un producto atractivo para empresas o fábricas que buscan robotizar sus líneas de producción y volverse más competitivas. Si bien en el mercado extranjero este tipo de robots tienen un costo muy alto y son más avanzados tecnológicamente, por lo que este robot estaría apuntado a ese sector que no puede afrontar con este tipo de gasto como puede ser la pequeña/mediana empresa o fábricas de países en vías de desarrollo.

Análisis de mercado preliminar

Existen empresas en Argentina que se dedican a ofrecer servicios de robótica a empresas, que trabajan con robots diseñados por empresas extranjeras como el robot MiR 500 (Link: <https://mobile-industrial-robots.com/products/robots/mir250>) por lo que presentan el problema de que, ante un desperfecto del robot, ellos deben repararlo por su cuenta, vender sus piezas o traer personal capacitado de otro país para solucionarlo lo que añade un gasto extra al robot. Además, este tipo de robot se deben importar lo que puede verse como una opción desalentadora

para las empresas argentinas debido a que el país tiene un historial con respecto a la “trabas” en la importación.

Aunque en este caso también se emplearon productos importados, estos cuentan con una alta demanda en el mercado, lo que asegura la existencia de múltiples proveedores que los distribuyen, garantizando la disponibilidad de stock en el futuro.

Público objetivo

El mercado de estos robots es atendido por aquellas empresas que buscan automatizar flujos de mercadería y aumentar su productividad. Estas empresas buscan automatizar sus depósitos, lo cual reduce la mano de obra humana y lograr una mayor flexibilidad y aumentar el rendimiento de su logística. Al mejorar la logística, estas empresas pueden atender a sus clientes sin interrupción, dado que los robots y equipos de mantenimiento automáticos llevan a cabo las operativas de un depósito con la mínima intervención de los operarios.

2. DESCRIPCIÓN GENERAL

Descripción general del robot

El robot es un pequeño vehículo con 4 ruedas mecanum (en el capítulo 6 se explicará en detalle este tipo de ruedas), para el transporte de cargas pequeñas con capacidad de ser conducido de manera remota o de manera autónoma mediante la carga de un mapa del entorno donde se va a desplazar el robot. Posee un escáner QR para leer códigos impresos en papel, sensores de localización como encoder, giroscopios, acelerómetro, magnetómetro y 2 sensores de ultrasonido.

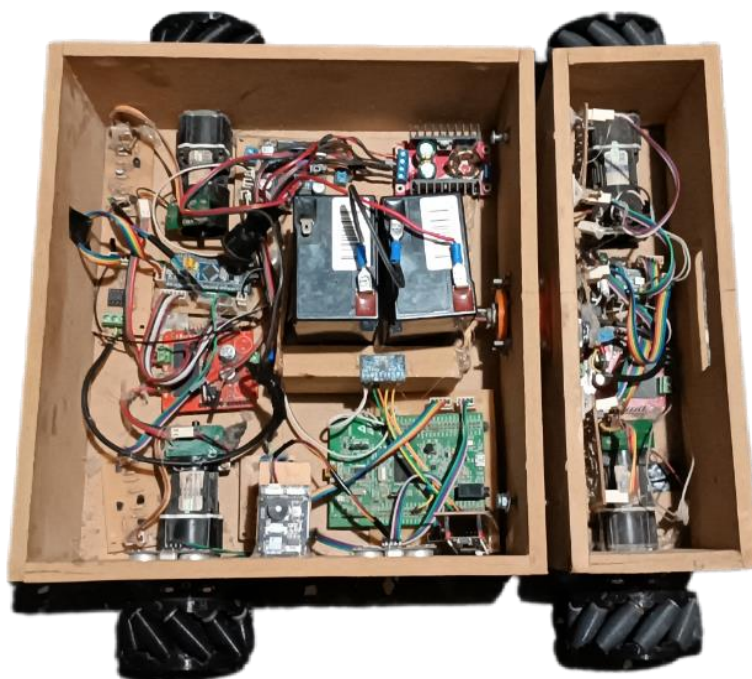


Figura 2.1 - Foto del Robot visto de arriba

El robot consta de dos estructuras rígidas unidos por un rodamiento que funciona como un amortiguador sencillo ante imperfecciones del terreno, haciendo que el robot asiente las cuatro ruedas y pueda circular en suelos que no sean parejos.

Contiene 2 placas Bluepill con el microcontrolador STM32F103C6T8 que funcionan como controlador de los motores del robot (2 motores por cada microcontrolador), donde mediante un sistema de comunicación por RS485 puede recibir parámetros para modificar la configuración de los lazos de control, modificar setpoints, cambiar el PWM del driver del motor, leer las cuentas del encoders, etc.

La placa principal, una Discovery board que posee un microcontrolador STM32F407VGT funciona como controlador principal del robot, realiza la configuración de los controladores de

los motores, controla la posición y orientación de robot, interactúa con los sensores del robot, además de correr la interfaz con el exterior. La comunicación con el usuario se realiza por medio de Bluetooth utilizando un módulo Bluetooth-uart para ese caso, recibiendo y enviando los caracteres de modo similar al funcionamiento de una consola de texto.

El robot posee 3 modos de funcionamiento seleccionables por el usuario:

- **Modo de control a distancia o “control remoto”:** Durante este modo, mediante la recepción de caracteres por medio de bluetooth, el robot realiza rutinas sencillas de desplazamiento hacia adelante, atrás, girar y desplazarse lateralmente. Es un modo de comunicación y funcionamiento sencillo ya que la trama de comunicación es muy corta, no hay control por parte del sistema de posición y orientación del robot por lo que es un modo útil para realizar pruebas, demostraciones o tareas sencillas.
- **Modo consola:** Al enviar el carácter “!” por medio de bluetooth al robot, se entra al modo consola del robot donde se tiene acceso a varios parámetros del robot, es básicamente tener un acceso a un control total del robot, desde leer los datos que envían los sensores como el ultrasonido o el giroscopio, indicar los coeficientes P, I y D de los controladores, tanto de los que se encuentran en los microcontroladores STM32F103 como los que están en el STM32F407. También se realizan identificaciones y captura de los distintos datos del robot, como por ejemplo, las cuentas de los encoder, valores del giroscopio y magnetómetro, resultados de los distintos algoritmos de fusión de sensores, desactivar o activar diferentes funciones del sistema de control, así como utilizar el sistema de posición y orientación para el desarrollo de actividades más complejas (por ejemplo, realizar un trabajo en conjunto entre el robot y una PC para desarrollar un sistema de control y manejo del robot más preciso o personalizado para una tarea específica).
- **Modo de auto navegación:** Este modo permite al robot desplazarse a través de su entorno, utilizando un mapa precargado en la memoria. Este mapa consta de estaciones y sus conexiones, con información sobre la distancia entre estaciones y la orientación de dicho tramo. El robot, en base a la estación donde se encuentra actualmente, su estación de destino y la orientación que posee actualmente, arma el trayecto más corto, sabiendo cuales son las estaciones intermedias que se encontraran en el camino, empezará a navegar automáticamente utilizando el giroscopio, magnetómetro y los sensores de ultrasonido para saber su orientación y no desviarse del camino. En cada estación se encuentra un código QR con el nombre de cada estación, por lo que al acercarse a una estación disminuye su velocidad para que pueda escanear el código de la estación con su escáner y de esta manera determinar que llegó a destino. La configuración inicial del modo de auto navegación, así como información sobre su ubicación y que está realizando actualmente se realiza por medio de la comunicación bluetooth desde el modo consola y el ingreso de un nuevo destino se realiza por el medio descrito anteriormente o colocando frente al escáner el código QR que representa la estación de destino.

Características

El robot cuenta con 4 motores faulhaber 2342L012CR que impulsan cada una de las ruedas, que es alimentado por 2 baterías de 6V 4.5AH de gel de electrolito absorbido de la marca Hi-Starx. La tensión de las baterías pasa primero por un regulador boost step-up, que asegura una salida constante de 12V para alimentar la etapa de potencia de los motores, garantizando un buen torque en las ruedas del robot sin importar la tensión de salida de las baterías. Esta tensión se conecta a la entrada del driver de puente H, encargado de gestionar el control y la interacción entre la etapa de potencia y la de control. Por otro lado, la etapa de control utiliza una alimentación de 5V, obtenida mediante un regulador buck step-down conectado a las baterías, que regula la tensión para alimentar los microcontroladores, módulos, sensores y otros componentes.

La etapa de control está compuesta por 2 microcontroladores STM32F103T8C6 que se encargan del control de los 4 motores del robot (2 por cada microcontrolador) y un microcontrolador SMT32F407VGT para el manejo general del robot que incluye el sistema de localización, interfaz de usuario, interacción con los sensores, etc. El robot cuenta con 2 sensores de ultrasonido, un escáner QR, 4 encoders, un módulo IMU que posee giroscopio, acelerómetro y magnetómetro, 3 módulos de comunicación RS485 y un módulo bluetooth-uart.

Con la comunicación bluetooth puede comunicarse de manera remota con el robot y hacer uso de los 3 modos de funcionamiento que posee el robot.

En la Figura 2.2 se muestra un diagrama completo de la estructura del robot:

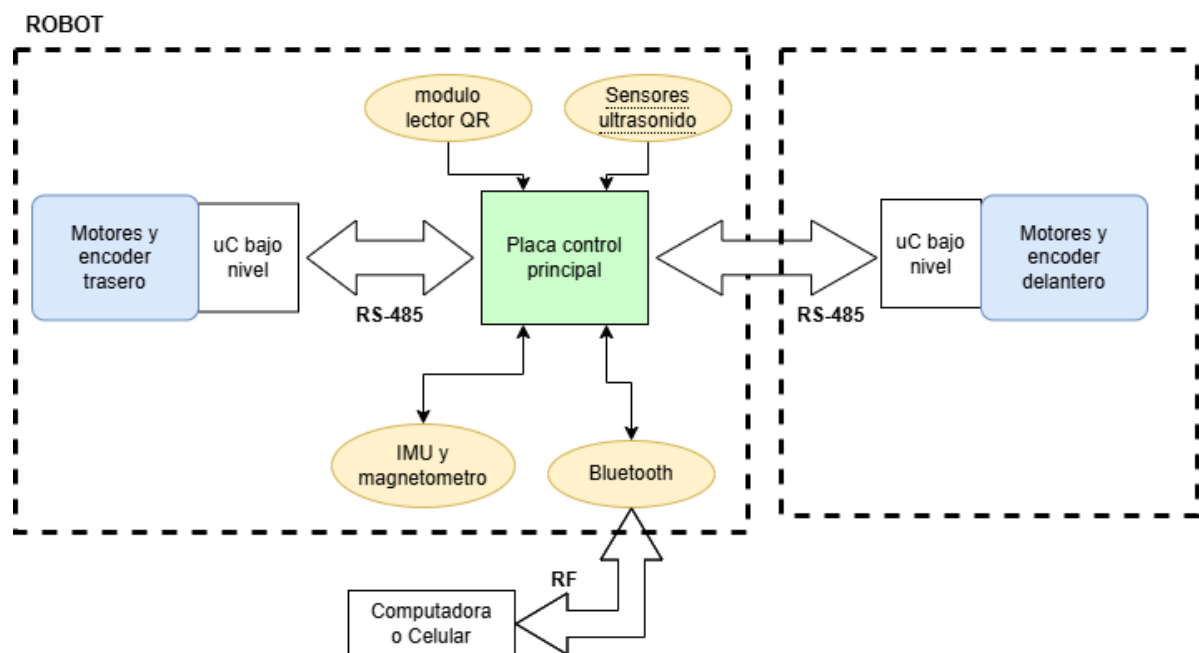


Figura 2.2 - Diagrama de la estructura del robot, indicando sus partes y cómo se relacionan

Algunas de las funciones que permite hacer el robot es:

- Auto Navegación, por medio de un mapa ya predefinido y con códigos QR.
- Controlador de posición y orientación del robot.
- Lectura de distancia por medio de ultrasonido.
- Manejar cada motor del robot de manera independiente o de forma conjunta.

- Realizar rutinas de identificación, donde se realiza una maniobra del robot y se obtienen datos de alguna de sus características a lo largo del tiempo.

Estas funciones se centrarán en detalle más adelante, en el capítulo correspondiente a los modos de funcionamiento.

Funcionamiento interno

Los microcontroladores STM32F103T8C6 se encargan del manejo de los motores por medio del driver de potencia, utilizando controladores PID, además se encargan de manejar los encoder para medir la velocidad de giro y la cantidad de vueltas que realizó la rueda. El microcontrolador posee 2 tipos de controladores para cada motor, un controlador de velocidad (el que se utiliza para el motor de auto navegación y para el sistema de control de posición y orientación del robot) y un controlador de posición del eje del motor. Posee un puerto de comunicación donde recibe los parámetros de funcionamiento de los motores como los valores de los coeficientes de los controladores, valores de setpoint, cambiar entre los tipos de lazo de control y también puede devolver valores como velocidad de giro o posición de las ruedas de los motores.

En resumen, los microcontroladores STM32F103T8C6 funcionan como controladores de motor que pueden ser operados por medio de un puerto de comunicación.

El microcontrolador STM32F407VGT se encarga de una etapa de más alto nivel que se encarga de configurar los controladores de los motores (en este caso son los microcontroladores STM32F103T8C6), obtener la posición de las 4 ruedas del robot, leer datos de los sensores y procesarlos para las diferentes rutinas que tiene programadas como la interfaz de consola, la de navegación, el control de posición y orientación. Por ejemplo, una de las tareas que realiza este microcontrolador es tomar los datos de los sensores relacionado con la localización del robot, y el encoder, mediante la comunicación RS485 con los microcontroladores STM32F103C6T8. Además, se puede medir la distancia a un objeto por medio de los sensores de ultrasonido HC-SR04. También puede medir la velocidad angular del robot utilizando el giroscopio, y junto con el campo magnético por medio del magnetómetro para luego combinar estos datos en un proceso llamado fusión de sensores, que consiste en unir datos provenientes de distintas fuentes pero que se relacionan con un mismo parámetro para obtener un resultado más fiable que si se hubiera tomado estos datos de forma separada. Un ejemplo de este proceso es la determinación de la orientación del robot, que puede lograrse utilizando diferentes métodos: usar el magnetómetro como brújula, el giroscopio integrando la velocidad angular, o los encoders a partir de la cantidad de giros de las ruedas. Sin embargo, cada uno de estos métodos presenta ciertas limitaciones o inconvenientes. El magnetómetro se ve influenciado por las corrientes eléctricas dentro y fuera del robot. El giroscopio posee un offset que varía con la temperatura por lo que ir integrando una constante puede ocasionar que, debido a la integración, nuestro error vaya creciendo a lo largo del tiempo. Las medidas en base a los encoders pueden dar errores y algunas de las ruedas pueden girar con deslizamiento, y los sensores de ultrasonido tienen un tiempo de respuesta alto. El fusionado de sensores permite juntar todas estas fuentes de datos para mitigar los problemas que tiene cada uno, hacer que la ventaja de un método se compense con la ventaja de otro, haciendo que el resultado final sea más confiable que utilizar cada uno de estos métodos

por separado. Gracias a esta técnica podemos tener mediciones confiables acerca de parámetros vitales para la navegación del robot.

Otra función del microcontrolador STM32F407VGT es la de implementar un sistema de control de posición y orientación mediante controladores PID, para controlar su desplazamiento respecto a un punto de referencia fijo, independiente de la posición del robot, permitiendo una mayor precisión y comodidad a la hora del manejo del mismo. Este sistema de control es el que se utiliza para el modo de auto navegación, para controlar el giro que debe realizar en cada estación para apuntar hacia la estación siguiente. Este sistema también se asegura que el robot no se desvíe de su camino (como, por ejemplo, detectar que se está desplazando lateralmente fuera de su recorrido), y también puede utilizar una pared como guía para controlar que posea la orientación correcta.

La comunicación entre los microcontroladores se realiza por medio de una red RS485 Maestro-Esclavo donde el maestro, que es el microcontrolador STM32F407VGT le solicita a sus esclavos, los STM32F103C8T6 que realicen ciertas funciones o devuelvan parámetros, como ser las cuentas de los encoders, etc. Esta comunicación se realiza por medio de los módulos RS485 utilizando el circuito integrado SN75176 que se encarga de transformar la interfaz RS485 a TTL y viceversa ya que dicho integrado puede funcionar como receptor o transmisor. La comunicación entre los microcontroladores STM32 del robot es half-duplex, es decir, que no es posible estar transmitiendo y recibiendo a la vez, solo se puede recibir o transmitir señal, siendo el motivo por lo que se decidió utilizar el sistema maestro-esclavo para la comunicación. En la Figura 2.3 se muestra un diagrama de la red de comunicación dentro del robot.

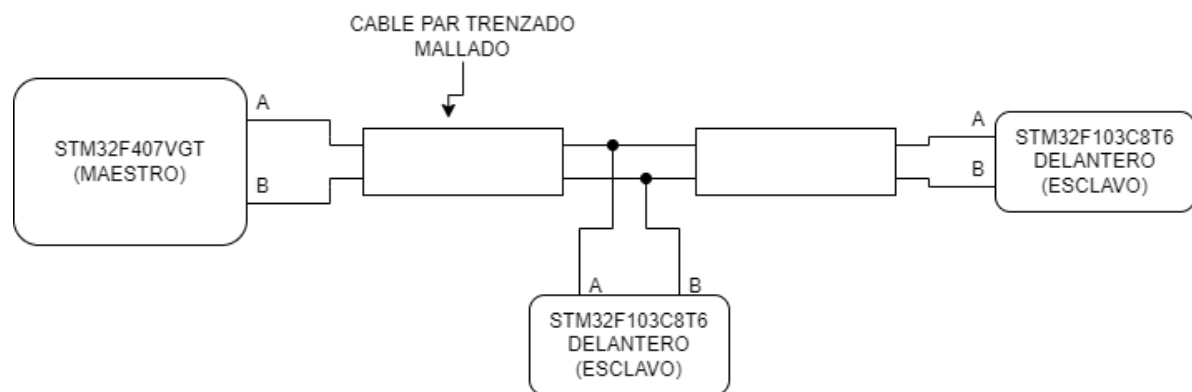


Figura 2.3 - Diagrama de conexión de la red de comunicación dentro del robot

Especificaciones

A continuación, se detalla una lista de las características físicas del robot

- Dimensiones: 39,3cm (largo) x 39,4cm (ancho) x 13,1 cm (alto)
- Dimensiones del chasis delantero: 9,7cm (largo) x 29,5cm(ancho) x 12 cm (alto)
- Dimensiones del chasis trasero: 28,2cm(largo) x 29,5 cm (ancho) x 12 cm (alto)
- Peso (sin carga): 5 kg

- Capacidad de carga: 5 kg
- Tracción: 4 motores a escobillas Faulhaber 2342L012CR
- Ruedas: 4 ruedas Mecanum u omnidireccional
- Batería: 2 baterías de gel electrolítico Hi Starx (6v 4,5AH)
- 4 Encoders ópticos de cuadratura
- 1 escáner QR (gm65)
- 1 giroscopio, magnetómetro y acelerómetro (MPU 9250)
- 2 sensores de ultrasonido HC-SR04
- Conexión inalámbrica Bluetooth
- 3 modos de funcionamiento: control remoto, consola y de auto navegación

3. MODOS DE FUNCIONAMIENTO

El robot cuenta con 3 modos de funcionamiento principales

- Modo de Manejo por Control Remoto
- Modo Consola
- Modo de Auto Navegación

Modo de Manejo por Control Remoto

En este el modo que se encuentra el robot al encenderse. Se trata de un modo de funcionamiento similar a la de los autos de radiocontrol. El robot recibe por medio de bluetooth un carácter que ejecutará una rutina básica de movimiento como ir hacia adelante, girar, desplazamiento lateral, etc. A continuación, se muestra una lista de los movimientos cargados.

Carácter	Acción
F	Moverse hacia adelante
B	Moverse hacia atrás
R	Moverse lateralmente hacia la derecha
L	Moverse lateralmente hacia la izquierda
S	Detenerse
G	Girar hacia la derecha
I	Girar hacia la izquierda

En caso que se reciba el carácter ‘!’ el robot saldrá de este modo para ingresar al modo consola.

Modo Consola

Durante este modo, el robot puede recibir comandos para ejecutar acciones específicas o ver los parámetros de funcionamiento del robot, como los coeficientes de los controladores del robot, ver las mediciones de los sensores, etc.

En el modo consola, el usuario puede acceder a diversos datos internos del robot como velocidad, mediciones del giroscopio, magnetómetro, ultrasonido, etc. Permitiendo que desde una computadora se pueda acceder a estos datos y realizar operaciones de control más complejos.

A continuación, se muestra una lista de comando cargados en el robot junto con una descripción de cada tipo argumento

Comandos	Descripción
reset	Resetea el sistema principal del robot (STM32F407VGT).
sProp [ID_MOTOR] [ID_CONT] [F]	Colocar el valor [F] en el coef. proporcional del controlador PID de [ID_CONT] del motor [ID_MOTOR].
sInte [ID_MOTOR] [ID_CONT] [F]	Colocar el valor [F] en el coef. integral del controlador PID de [ID_CONT] del motor [ID_MOTOR].
sDeri [ID_MOTOR] [ID_CONT] [F]	Colocar el valor [F] en el coef. derivativo del controlador PID de [ID_CONT] del motor [ID_MOTOR].
vProp [ID_MOTOR] [ID_CONT]	Visualizar el valor del coeficiente proporcional del coeficiente PID del [ID_CONT] del motor [ID_MOTOR].
vInte [ID_MOTOR] [ID_CONT]	Visualizar el valor del coeficiente proporcional del coeficiente PID del [ID_CONT] del motor [ID_MOTOR].
vDeri [ID_MOTOR] [ID_CONT]	Visualizar el valor del coeficiente proporcional del coeficiente PID del [ID_CONT] del motor [ID_MOTOR].
ContV [E]	Deshabilita o habilita el sistema de control de la velocidad.
Copyv [E]	Deshabilita o habilita el sistema de control de velocidad y posición.
Encod	Indica la cantidad de vuelta promedio que dieron los cuatros ruedas.
Encod [ID_MOTOR]	Indica la cantidad de vueltas que realizó la rueda acoplada al motor [ID_MOTOR].
Encod r	El valor de las vueltas de todas las ruedas se vuelve 0.
setSP [ID_CONT] [REF][F]	Coloca el valor [F] como setpoint en el controlador [ID_CONT]. Si [REF] es relativa entonces se sumará [F] al valor del setpoint actual, en cambio sí es absoluta entonces se colocara [F] como setpoint.
setSP [ID_CONT] [REF] [-ID_MOTOR] [F]	Coloca el valor [F] como setpoint en el controlador [ID_CONT] del motor

	[ID_MOTOR]. Si [REF] es relativa entonces se sumará [F] al valor del setpoint actual, en cambio sí es absoluta entonces se colocará [F] como setpoint.
Motor [ID_MOTOR]	Devuelve la velocidad del motor [ID_MOTOR] en rps.
scanQ	Se inicia el escaneo de un código QR. Se realizaron 5 intentos de escaneo.
scanQ [I]	Se inicia el escaneo de un código QR. Se realizará -I intentos de escaneo.
medUS	Devuelve las lecturas de los sensores de ultrasonido.
imuif a	Devuelve los valores medidos por el acelerómetro.
imuif g	Devuelve los valores medidos por el giroscopio.
imuif m	Devuelve los valores medidos por el magnetómetro.
imuif e	Devuelve las rotaciones de Euler, medidas por técnica de fusión de los sensores.
imuif o g [COOR]	Devuelve el valor de offset del giroscopio en la coordenada [COOR]
imuif o g [COOR] [F]	Coloca el valor [F] como valor de offset del giroscopio en la coordenada [COOR]
imuif o m [COOR]	Devuelve el valor de offset del magnetómetro en la coordenada [COOR]
imuif o m [COOR] [F]	Coloca el valor [F] como valor de offset del magnetómetro en la coordenada [COOR]
imuif o a [COOR]	Devuelve el valor de offset del acelerómetro en la coordenada [COOR]
imuif o a [COOR] [F]	Coloca el valor [F] como valor de offset del acelerómetro en la coordenada [COOR]
mover [ID_CONT] [DIR] [F]	Colocará el valor de [F] al controlador [ID_CONT] de tal forma para que el robot realice el movimiento [DIR].
auNav s [EST_O] [EST_M]	Inicia la rutina de auto navegación con el robot colocado en la estación [EST_O] apuntando a la estación [EST_M].
auNav d [EST_D]	Indica robot que debe dirigirse a la estación

	[EST_D].
auNav u	Indica cuál es la ubicación actual del robot.
auNav i	Indica el estado de la rutina de auto navegación.
robot e p	Inicia el controlador de posición y orientación del robot.
robot d	deshabilita el controlador de posición y orientación del robot.
robot s [THETA] [X] [Y]	Indica el valor de setpoint del controlador, indicando su orientación [THETA], su posición en x [X] y su posición en el eje y [Y]
robot u [THETA] [X] [Y]	Indica la ubicación actual del robot para el controlador de posición y orientación, indicando su orientación [THETA], su posición en x [X] y su posición en el eje y [Y]
robot v	Devuelve los datos de salida del controlador de posición y orientación, además de la velocidad del robot desde el punto de referencia anclado al robot y su velocidad respecto al punto de referencia fijo anclado en el entorno.
robot p [ID_CONT_R] [F]	Coloca el valor [F] como el valor del coeficiente proporcional del controlador [ID_CONT_R] del sistema de control de posición y orientación.
robot i [ID_CONT_R] [F]	Coloca el valor [F] como el valor del coeficiente integral del controlador [ID_CONT_R] del sistema de control de posición y orientación.
robot r [ID_CONT_R] [F]	coloca el valor [F] como el valor del coeficiente derivativo del controlador [ID_CONT_R] del sistema de control de posición y orientación.
robot c m [F]	Coloca como valor de saturación de los controladores el valor [F] expresado en rps.
robot c d [F]	Coloca como velocidad mínima de los motores [F], expresado en rps.
robot f s	Invierte el sentido de medición de ángulo dado por el imu.
robot f b	Devuelve el valor de offset para el ángulo medido por el imu.

robot f b [F]	Asigna el valor [F], expresado en grados, como valor de offset para el ángulo medido por el imu.
robot f i	Devuelve el ángulo medido por el imu.
robot f o	Devuelve el valor de offset para el ángulo medido por el ultrasonido.
robot f o [F]	Asigna el valor [F], expresado en grados, como valor de offset para el ángulo medido por el ultrasonido.
robot f u	devuelve el valor del ángulo medido por el ultrasonido.
robot f k	Activa o desactiva la técnica de fusión de sensores para el sistema de control de posición y orientación.
robot f l	Activa o desactiva el uso del ultrasonido en la técnica de fusión de sensores para el sistema de control de posición y orientación.
robot f d [F]	Coloca el valor [F] como el valor de offset de las mediciones de distancia de los sensores de ultrasonido.
ident [ID_CONT] [DIR] [F] [I]	Realiza un ensayo o identificación del sistema. Se colocará el valor [F] como setpoint a los controladores [ID_CONT] para que el robot realice la maniobra [DIR] y de guardarán en memoria [I] cantidad de datos de todos los sensores de localización del robot.
ident g	Devuelve los valores del giroscopio obtenidos durante el ensayo.
ident m	Devuelve los valores del magnetómetro obtenidos durante el ensayo.
ident a	Devuelve los valores del acelerómetro obtenidos durante el ensayo.
ident e	Devuelve los valores de las rotaciones de Euler obtenidas durante el ensayo.
ident r	Devuelve los valores de los encoders de las 4 ruedas obtenidos durante el ensayo.
ident n	Devuelve los valores de salida del contralor de posición y orientación, utilizando los valores de los encoder.
ident t	Devuelve los valores de salida de los

	controladores de posición y orientación, con datos obtenidos por la fusión de sensores.
ident k	Devuelve los valores de Theta obtenidos de los distintos sensores y métodos.
ident c g	Con los datos obtenidos, realiza la rutina de calibración del giroscopio y devuelve el valor de offset.
ident c m	Con los datos obtenidos, realiza la rutina de calibración del magnetómetro y devuelve el valor de offset.

Listado de argumentos:

Argumento	Descripción	Valores
[ID_MOTOR]	Identificador del motor interno del robot	dd: Motor delantero derecho di: Motor delantero izquierdo td: Motor trasero derecho ti: Motor trasero izquierdo
[ID_CONT]	Identificador del tipo de controlador del sistema de control del motor	P: Controlador de posición V: Controlador de velocidad
[I]	Valor numérico expresado como un número entero	Cualquier valor expresado como número entero Ej.: 1234
[F]	Valor numérico expresado en formato de número flotante o con punto decimal	Cualquier valor expresado como número con punto decimal. Ej.: -1.2345
[E]	Identificador para indicar habilitar o deshabilitar	e: Habilitar d: Deshabilitar
[DIR]	Identificador del tipo de movimiento del robot	ad: Adelante at: Atrás o reversa gi: Girar en sentido izquierdo gd: Girar en sentido derecho ld: Desplazarse lateralmente a la derecha li: Desplazarse lateralmente a la izquierda
[REF]	Indicar la referencia a tomar para aplicar el comando	a: Absoluta, desde el origen r: Relativa, desde su valor actual
[EST_O]	Indica el nombre de la	Cualquiera de las estaciones

	estación de origen	cargadas en el mapa
[EST_M]	Indica el nombre de la estación que está apuntando	Cualquiera de las estaciones cargadas en el mapa
[EST_D]	Indica el nombre de la estación de destino	Cualquiera de las estaciones cargadas en el mapa
[THETA]	Indica un ángulo expresado en grados	Valor numérico con punto decimal Ej.: 90
[X]	Indica distancia, expresado en metros	Valor numérico con punto decimal Ej.: 1.5
[Y]	Indica distancia, expresado en metros	Valor numérico con punto decimal Ej.: 1.5
[ID_CONT_R]	Identificador del controlador dentro del controlador de posición y orientación	t: Controlador de la orientación del robot x: Controlador de la posición en la coordenada X y: Controlador de la posición en la coordenada Y a: Todos los controladores dentro del sistema de control
[COOR]	Identificador de las coordenadas del IMU	x: Coordenada X y: Coordenada Y z: Coordenada Z

Modo de Auto Navegación

Durante este modo, el robot podrá navegar mediante un mapa ya configurado previamente. Este mapa contiene varias estaciones conectadas por caminos, preferentemente rectos, sin giros significativos a lo largo de su recorrido. El robot se encontrará parado en una de las estaciones, por medio de un comando por bluetooth, se indicará a qué estación debe dirigirse y tomará el camino más corto para ir a dicha estación. Una vez que haya llegado a la estación destino, entrará en modo espera, repitiendo el proceso anteriormente descrito. El robot tiene la capacidad de utilizar las paredes en su entorno como guía, posee 2 sensores de ultrasonido hacia el lado derecho que utiliza para medir la distancia a la pared y utilizarla para asegurarse que mantiene una distancia segura, de 30cm. El robot debe mantener una orientación paralela a la pared, razón por la cual cuenta con dos sensores de ultrasonido. Estos, junto con los métodos de orientación integrados, permiten medir la posición del robot en relación con la pared. De esta manera si se

diseña el mapa para que el trayecto pase de forma paralela a la pared, se puede utilizar esta forma de localización para darle mayor robustez al desplazamiento de robot y se posicione en forma absoluta en un lugar determinado. Si el trayecto no cuenta con una pared como guía, el robot ignorará las mediciones de los sensores de ultrasonido y utilizará únicamente los datos de orientación proporcionados por el magnetómetro, giroscopio y encoders.

Por medio del modo consola, se puede ingresar al modo de auto navegación indicando en qué estación se encuentra el robot y hacia qué estación está apuntando la parte móvil más pequeña del chasis.

4. SENSORES, MÓDULOS Y PARTES INTERNAS DEL ROBOT

Módulos del robot

El robot posee un diseño modular, es decir, que todas sus funcionalidades están separadas por partes para su fácil acceso y comprensión del sistema.

A continuación, se dará un listado de todas las partes del dispositivo con una breve descripción

Módulo RS485

Se encarga de realizar la comunicación entre la placa de control principal (Discovery board) y las placas esclavas (Bluepill), mediante la norma RS-485 de EIA (Electronics industries alliance) y TIA (Telecommunication industries associations). La norma especifica una comunicación con buena resistencia al ruido generado por interferencia electromagnética, por ello se usa a nivel industrial en redes de comunicación digital. Para implementar el módulo, se usó el chip IC SN76175, que es un transceiver de RS-485 half-duplex. La señal de los módulos USART de los microcontroladores se envía al SN76175 para gestionar la comunicación. Desde la perspectiva de los microcontroladores, solo se realiza una comunicación UART, pero con la ventaja de que el SN76175 transforma la señal al formato físico definido por la norma RS-485. En el capítulo 5 se detallará más a fondo el funcionamiento de este módulo y sus características.

Driver para motores DC

Es la etapa de potencia, que genera la tensión de PWM para manejar a los motores. Se utilizó el driver SparkFun Monster Motor Shield que admite una tensión máxima de 16V, corriente de salida de hasta 14A de forma continua, lo que encaja perfecto con los motores Faulhaber de 12V que mediante pruebas pudimos comprobar que consumen una corriente de hasta 2A por motor. Esta placa utiliza el IC VN12SP30 como puente H para los motores, posee protección contra sobrecorriente y sobretensión. Además, dispone de un sensor de corriente interno, que no es preciso, por lo que su propósito es más para verificar que el motor no esté consumiendo mucha corriente (situación que corresponde a un esfuerzo excesivo, que puede ser perjudicial para la vida útil del motor). A pesar de que este módulo supera con creces los requisitos de corriente para manejar 2 motores de 2A de corriente máxima, el motivo de elegir este módulo, es debido a que es muy usado en el entorno de Arduino para manejar grandes motores además de que se puede encontrar en páginas como Mercado Libre, por lo que de esta manera garantizamos de encontrar esta placa en un futuro en caso de necesitar para reparar el robot o construir uno nuevo.

Encoder

El encoder es un dispositivo electromecánico que codifica la posición o velocidad del eje del motor, obteniendo una señal digital. Existen diferentes tipos de encoders, para el caso del robot utilizamos encoder ópticos de cuadratura, que consiste un disco ranurado acoplado al eje del motor en medio de un diodo LED y 2 sensores ópticos, como se muestra en la Figura 4.1.

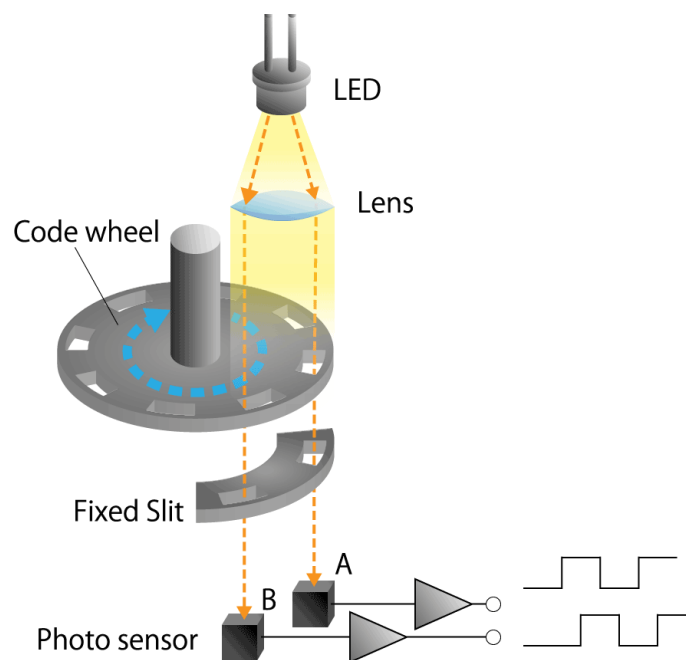


Figura 4.1 - Diagrama de funcionamiento de un encoder óptico de cuadratura

Este disco bloquea o deja pasar la luz dependiendo la posición del eje del motor, si existe una ranura en medio del led y el fotodiodo entonces dejará pasar la luz y si no hay ranura entonces la bloquea, generando a la salida una señal cuadrada en caso de que el eje está girando. La posición de los dos fotodiodos está colocada de tal manera que las señales de salida de ambos se encuentran con un desfase de 90° , de esta forma se logra determinar el sentido de giro del motor sabiendo que señal se encuentra delante de otra, como se muestra en la Figura 4.2.

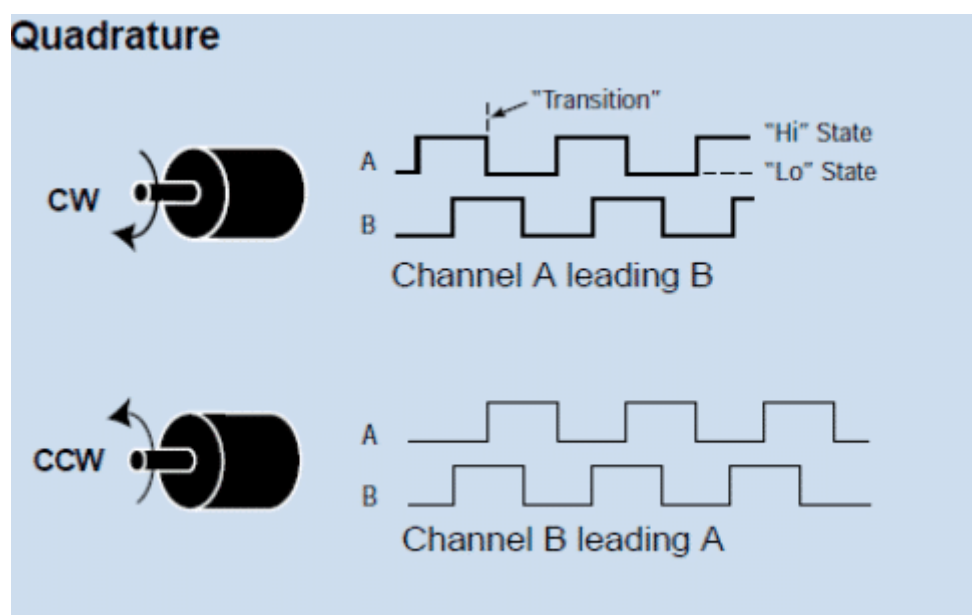


Figura 4.2 - Señal de salida del encoder según el sentido de giro

Los motores Faulhaber usados en el robot tiene acoplado un encoder óptico de cuadratura que consta solamente dos fototransistores con sus respectivos diodos LED por lo que era necesario desarrollar una placa acondicionadora de señal, que transforme la señal de salida analógica de los fototransistores, a una señal cuadrada. Dicha señal digital, es más inmune al ruido y a los “falsos pulsos” que se pudieran realizar los encoder. Para los leds del encoder, se colocó una resistencia limitadora de corriente, entre los 5v y el terminal positivo del encoder. Para acondicionar la señal de los fototransistores, se utilizaron transistores que funcionan como llave, conectando su base a la salida del encoder para lograr que la señal de salida se encuentre entre los 5v-0v.

Módulo GM65 QR

Es el módulo que se encarga de realizar el escaneo de los códigos QR, desarrollado por DFROBOT, posee un ASIC y un sensor CMOS para realizar el escaneo y decodificación de distintos tipos de códigos como el código QR o los códigos de barras que se suelen encontrar en los productos. Posee además una luz LED que sirve para iluminar en ambientes de poca luz y se comunica por medio de UART o USB. Se eligió este módulo por ser una solución económica y eficiente para escanear códigos QR, ya que incluye todos los elementos necesarios y puede ser utilizado con procesadores de bajo costo o capacidad limitada. Esto evita la necesidad de desarrollar un sistema más complejo, con cámara, iluminación LED y mayores recursos computacionales, como una Raspberry Pi o un microcontrolador dedicado exclusivamente a esta tarea, lo cual incrementaría los costos y complicaría la reparación o ensamblaje del robot.

Módulos de Ultrasonido

Se trata de los módulos de ultrasonido HC-SR04, utilizados para medir distancias y calcular la orientación del robot. Posee un parlante y un micrófono para ultrasonidos, cuando reciben el pulso en el pin “trigger”, el parlante emite un pulso de sonido de 40 KHz. aproximadamente. Este pulso rebota en el objeto que está al frente y volverá al módulo, el eco será escuchado por el micrófono. Midiendo el tiempo que tarda el pulso de sonido en ir y volver se podrá determinar a qué distancia está dicho objeto. Este módulo es uno de los más populares en el entorno de Arduino, lo que garantiza su disponibilidad futura. Además, su capacidad es ideal para este proyecto, ya que no se requiere medir grandes distancias.

Módulo bluetooth

Es el que se encarga de realizar la conexión remota entre el usuario y la Discovery board que consiste en recibir los datos por UART y enviarlos por Bluetooth y viceversa. Inicialmente se consideró utilizar el módulo HC-05, pero su tiempo de respuesta resulta lento para esta aplicación y, al manejar grandes volúmenes de datos, puede presentar pérdidas. Por ello, se optó por la Raspberry Pi Pico W como alternativa. A la Raspberry Pi Pico W se le cargó un código en MicroPython que emula el funcionamiento de un módulo HC-05. Este código permite almacenar los datos recibidos por el puerto UART y transmitirlos por Bluetooth, así como

recibir datos por Bluetooth y enviarlos al UART. Se optó por desarrollar el código en MicroPython debido a su sencillez y rapidez, gracias a su capa de abstracción de hardware. Además, ofrece excelente compatibilidad con la Raspberry Pi Pico W, una placa compacta y eficiente en términos de consumo energético, superando a otras opciones con comunicación inalámbrica como las placas ESP32.

Módulo de unidad de medidas inerciales

Este módulo sirve para detectar la orientación del robot, se utilizará el módulo GY-9250 que posee un sensor MPU 9250 que consta de un giroscopio, acelerómetro y magnetómetro. Este integrado posee comunicación I2C y SPI, para este trabajo en particular, se utilizará la comunicación SPI, con el objetivo de tener una comunicación lo más rápida posible. Se seleccionó este módulo porque combina un giroscopio y un magnetómetro en una sola placa, ambos esenciales para las tareas que se detallarán más adelante en este documento. Además, al ser ampliamente utilizado en el entorno Arduino, garantiza su disponibilidad para futuros reemplazos o proyectos.

Módulo de alimentación Step-down

Es el que se encarga de regular la tensión de la batería. La salida se ajusta, para obtener una tensión de 5V, la cual alimentará a la etapa lógica del robot. Para seleccionar este módulo, se eligió uno con una corriente de salida superior a 500 mA, que es el estimado del consumo máximo de la electrónica del robot. Además, cuenta con un sistema de regulación de voltaje tipo "switching" o "buck", que ofrece una mayor eficiencia en la reducción de voltaje, alcanzando un 92% de eficiencia en el módulo elegido.

Módulo de alimentación Step-up

Es el que se encarga de regular la tensión de la batería. La salida se ajusta, para obtener una tensión de 12V, la cual alimentará a la etapa de potencia del robot. El objetivo de este módulo es proporcionar una alimentación de 12V a los motores, garantizando una tensión adecuada para su funcionamiento, independientemente de la tensión de salida de las baterías. Esto mejora el desempeño del robot al desplazarse en su entorno. Por esta razón, se eligió una placa boost step-up, diseñada para manejar estos niveles de tensión y soportar la corriente de arranque de los motores, especialmente adecuada para su uso en robótica.

Motores del robot

Los motores faulhaber 2342L012CR fueron los elegidos para realizar todos los movimientos del robot. Estos son motores de corriente continua de tipo "brushed" o escobillas debido a que conmutación entre los distintos bobinados del motor se realiza por medios de un sistema mecánico que cambia la conexión entre los bobinados. Funciona a una tensión de 12V, posee un motorreductor de 1:64 que reduce la velocidad de salida haciendo que 64 vueltas del motor equivalgan a 1 vuelta de eje de salida a cambio de ganar un mayor torque en el motor, además contar con encoder para poder medir la posición y velocidad de giro del motor. Aunque existas mejores elecciones de motor como por ejemplo un motor "brushless" donde la conmutación de las bobinas se realiza de manera electrónica, obteniendo un motor que requiere menor mantenimiento y menor fricción interna debido a que no posee este sistema mecánico de

conminación, los motores del tipo “brushed” son los más comunes que podemos encontrar en el mercado argentino debido a que no requieren de mucha complejidad para operarlos además de ser el único tipo de motor que teníamos a nuestro alcance para hacer el proyecto.

Baterías de alimentación

Para alimentar el robot, se utilizaron dos baterías de gel electrolito de 6V y 4.5Ah de la marca Hi-Starx. La decisión de emplear dos baterías de 6V en lugar de una sola de 12V fue debido a la disponibilidad de componentes durante el desarrollo del proyecto, ya que únicamente se contaba con este tipo de baterías. Por este motivo, no se consideraron otras opciones como las baterías de ion de litio o LiPo, ya que suelen ser más costosas y requieren un equipamiento adicional para su manejo.

5. MODO DE COMUNICACIÓN

Comunicación interna del robot

El robot posee un sistema de comunicación interna para el envío de datos entre la central (stm32f4discovery) y los esclavos (Bluepill). En la Figura 5.1 se muestra un diagrama de la estructura del sistema de comunicación entre los microcontroladores dentro del robot.

ESTRUCTURA DEL SISTEMA DE COMUNICACIÓN

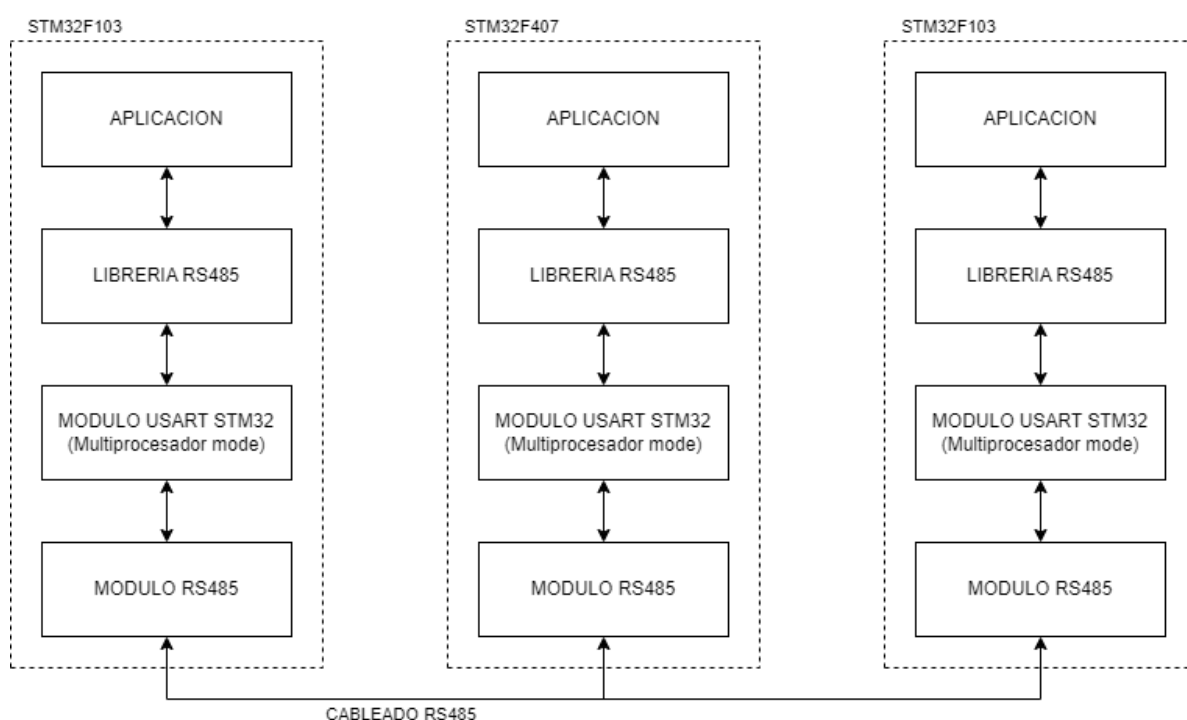


Figura 5.1 - Estructura del sistema de comunicación entre los microcontroladores dentro del robot, donde se puede observar las distintas capas del sistema.

RS485

RS-485 es un estándar de comunicación serial, que se utiliza comúnmente en entornos industriales y comerciales para la transmisión de datos. Diseñado por la Asociación de Estándares Electrónicos (Electronic Industries Alliance, EIA) en la década de 1980, RS-485 ha demostrado ser robusto y confiable, convirtiéndose en una elección popular para sistemas que requieren comunicaciones de larga distancia y resistencia al ruido.

Una de las características clave de RS-485 es su transmisión diferencial. Utiliza dos cables para cada señal (positiva y negativa) donde un cable transmite la señal y la otra transmite la misma señal, pero a la inversa, es decir, que cuando el cable positivo transmite un 1 lógico el cable negativo transmite un 0 lógico. Por lo tanto, para determinar si se está transmitiendo un 1 lógico

se analizará si la tensión en el cable positivo será mayor a la del cable negativo, lo que mejora la inmunidad al ruido y permite una transmisión más confiable a largas distancias.

Para aplicar este estándar, se diseñó una placa que servirá como un módulo transceiver (capaz de funcionar como transmisor como de receptor) que pasará la señal de niveles de 0 a 3,3V que proporcionan los pines de los microcontroladores a RS485 y viceversa. En el capítulo 8 se entrará más en detalle sobre el diseño de este módulo.

El motivo para elegir este protocolo de comunicación es debido a que posee una buena resistencia en ruido ya que el robot solo cuenta en su interior fuentes emisores de ruido, sino que al estar destinado a moverse dentro de fábricas donde es común encontrarse en motores o generadores que pueden producir grandes emisiones de ruido electromagnéticos que puedan atender a las líneas de comunicación del robot.

Además, este protocolo es fácil de implementar porque, no solo existen módulos para Arduino que realizan las conversiones a RS485, sino que también es posible encontrar circuitos integrados como el SN75176 o el MAX485 que cumplen la misma función y son fáciles de implementar en PCB ya que no requieren que las pistas de cobre cumplan con especificaciones de adaptación de línea como otros protocolos industriales como Ethernet o Bus CAN, solo es necesario colocar tres resistencias en la líneas de transmisión que sirven para polarizar dichas líneas y que la comunicación posea un mejor desempeño (Esta elección de resistencia se realiza en el capítulo 8).

Otros protocolos industriales como Ethernet o Bus CAN poseen mayores requisitos de adaptación de línea para permitir mejores velocidades de transmisión, característica que no necesitaremos ya que la velocidad de transmisión que se utilizó es de 115,2Kbit/s, mucho menor a los 12Mbit/s máximos que impone la norma RS485 por lo que sería una complicación innecesaria utilizar otros protocolos mas rápidos y complejos como Ethernet o Bus CAN. Por último, aunque Ethernet y Bus CAN permiten mayores velocidades de transmisión, la norma RS485 ofrece hasta 12 Mbit/s, muy por encima de los 115,2 kbit/s requeridos en este proyecto. Optar por esos protocolos más rápidos y complejos habría agregado costos y complicaciones innecesarias, tanto en la electrónica como en la disponibilidad futura de componentes.

Por lo tanto, aunque no se aproveche su capacidad para transmitir datos a largas distancias, el protocolo RS485 fue la opción más adecuada debido a su simplicidad, robustez y fácil accesibilidad.

Topología de la red:

RS-485 es comúnmente implementado como un sistema de bus, lo que significa que varios dispositivos pueden compartir la misma línea de transmisión. Esto facilita la conexión de múltiples dispositivos a un solo bus, lo que es especialmente útil en aplicaciones industriales. Este caso no es la excepción, donde todos los dispositivos están conectados al mismo bus bajo un sistema “maestro-esclavo”. La placa Discovery board será el maestro y las Bluepill los esclavos. Además, se diseñó un protocolo, donde el maestro envía paquetes a los esclavos y estos responden con la información solicitada. Para que la comunicación sea robusta, se agrega un código de verificación de errores a los paquetes.

Comunicación Multiprocesador

Los modelos USART de los microcontroladores STM32 poseen un modo de comunicación multiprocesador para ser usados en redes de comunicación maestro-esclavo, permitiendo una comunicación más eficiente desde el punto de vista de carga computacional.

La comunicación multiprocesador permite que cuando el maestro esté enviando un paquete de datos a uno de sus esclavos, los demás entraran en un modo de “mute” donde no recibirán los datos transmitidos por la línea de comunicación hasta que finalice dicha transmisión (de esta manera, los esclavos que no están participando en la comunicación, no perderán tiempo analizando los paquetes que no les pertenecen). Una vez que ya pasó el tiempo equivalente a la transmisión de un Byte y no se haya recibido ningún paquete nuevo, el módulo automáticamente saldrá del modo mute y volverá a su estado normal.

Este sistema tiene dos modos de funcionamiento:

- Idle Line Detection
- Address mask detection

Idle Line Detection:

Cuando el esclavo recibe los primeros paquetes datos y detecta que estos paquetes no son para él, por software ingresa al modo mute e ingresara automática al modo normal cuando detecte no se está enviando ningún paquete por el bus (Idle Line). En la Figura 5.2 se muestra cómo funciona este modo.

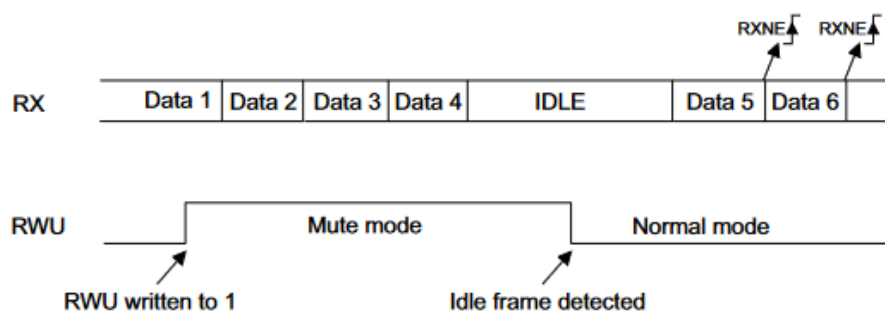


Figura 5.2 - Demostración del modo “Idle line detection”

Address mask detection:

Todos los esclavos tendrán una dirección dentro de la red. El esclavo estará en modo mute y pasará automáticamente al estado normal cuando reciba un paquete con el address del esclavo y volverá al estado mute cuando detecte que no se está enviando ningún paquete por el bus o cuando se haya enviado otro paquete con un address diferente. Para diferenciar un paquete con el address de un paquete con address el módulo analizará el MSB, si es 1 entonces el byte transmitido es de un address. Para funcionar en este modo es necesario configurar el módulo USART para enviar 9 bits donde el 9^{no} bit es el que se utilizará para indicar si el dato enviado se trata de un address o no, motivo por lo que cada dato recibido por uart ocupará 16 bit de memoria en vez de 8bit (tanto en el esclavo como en el maestro, sin embargo, este inconveniente es recompensado por más eficiencia por el lado de los esclavos). En la Figura 5.3 se muestra el funcionamiento de este modo.

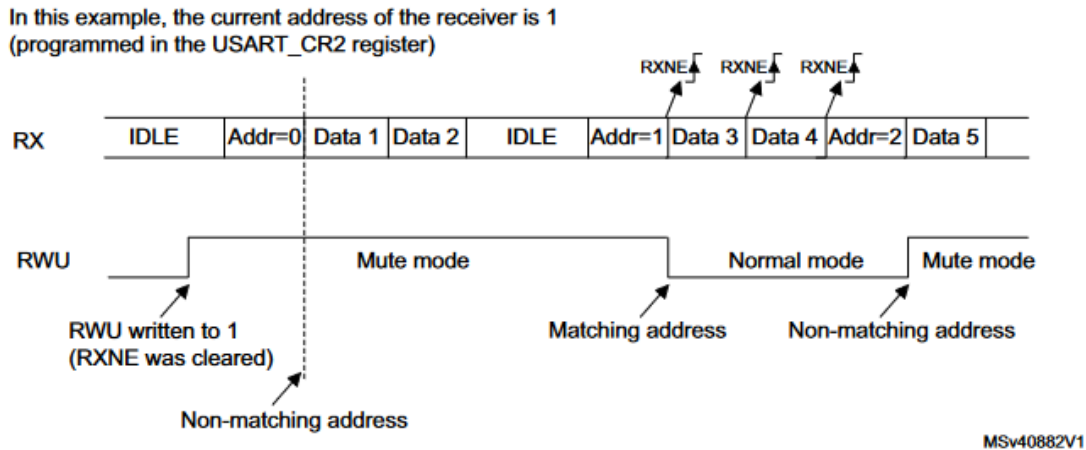


Figura 5.3 - Demostración del modo “Address mark detection”

Trama y control de datos

Se diseñó una trama de datos personalizada para la comunicación entre los microcontroladores, compatible con el modo multiprocesador de los módulos USART. Esta trama fue concebida para ser sencilla de implementar e interpretar por el programador, cumpliendo su propósito principal: transmitir órdenes y manejar datos numéricos de entrada y salida. Incluye un campo que especifica la tarea o servicio a ejecutar, junto con los datos necesarios para su operación o los resultados que se deben devolver. Además, incorpora una sección para la detección de errores en la transmisión y un marcador que señala el final de la trama.

Los paquetes de este sistema de comunicación están conformados de la siguiente manera

ADD	OPCODE	ARG	CRC	TAIL
1 byte	1 bytes	4 bytes	1 bytes	[0x0A, 0x0D] 2 bytes

- ADD: dirección del esclavo con el que se está haciendo la comunicación.
- OPCODE: Identificador de la tarea/servicio a realizar.
- ARG: argumento para la tarea/servicio a realizar.
- CRC: Verificación de redundancia cíclica de los bytes del add, opcode, arg.
- TAIL: Indica el fin de la transmisión, se utiliza con un salto de línea y retroceso.

El campo Address (ADD) tiene la función de identificar al esclavo específico con el que se establece la comunicación. El campo OPCODE indica el tipo de orden o servicio que debe ejecutar la Bluepill. Cada recurso o servicio disponible en la Bluepill está representado por un ID numérico, y cada tarea asociada a un recurso también cuenta con su propio ID numérico. Este campo tiene un tamaño de 1 byte, donde los 4 bits más significativos representan el ID del recurso a utilizar (por ejemplo: motores, encoders, controladores, etc.), y los 4 bits menos significativos corresponden al ID de los servicios de ese recurso (como configurar el PWM de

los motores, consultar el número de vueltas de una rueda o modificar los coeficientes de los controladores PID).

Algunas tareas como modificar los coeficientes de los controladores PID, necesitan de un parámetro de entrada para funcionar, por lo que se colocó un espacio de la trama de datos para esta función. El argumento posee un tamaño de 4 bytes permitiendo enviar cualquier tipo de dato. La transmisión UART manda tramas de 8bit, por lo que para enviar 16bit se divide dicho dato en dos partes de 8bit. En el caso de valores “float”, estos ocupan 32 bit en memoria por lo que se toma cada byte de la variable float, se envía el paquete y luego se lo vuelve a armar en destino, de esta manera el valor float no pierde precisión durante su trayecto.

Para garantizar la integridad de la comunicación, se emplea un código CRC que verifica que la trama de datos se haya transmitido correctamente. El CRC se calcula a partir del conjunto de datos que incluye el Address, el Opcode y los Argumentos. En el receptor, se recalcula el CRC utilizando los mismos datos, y si el resultado coincide con el valor recibido, se confirma que el paquete fue transmitido sin errores. Por último, está el TAIL de la transmisión para indicar que el paquete de datos ya terminó.

Comunicación robot-usuario.

El robot posee un modo de comunicación con el usuario por medio de Bluetooth, teniendo acceso a una interfaz de consola (Command line interfaz) para manejar y modificar distintos aspectos del robot. El sistema Bluetooth del robot está conformado por la Discovery que envía los datos por UART al módulo Bluetooth-serial.

La razón de utilizar una Raspberry pi pico w, en vez de utilizar los módulos Bluetooth que existen en el mercado, es por la rapidez, el tiempo de respuesta y su confiabilidad. Se realizaron pruebas con el módulo HC-05 bajo los modos de control remoto y se observó un tiempo de respuesta del orden del segundo (mediante un analizador lógico, se vio que la mayor parte del tiempo se perdía en el módulo HC-05). También se hizo una prueba de solicitudes de información de parámetros del robot y se observó que el módulo HC-05 puede presentar pérdida de datos cuando se realizan transmisiones de varios datos durante un periodo corto de tiempo. A la Raspberry pico w se le realizó un programa que emule el funcionamiento del módulo Bluetooth-serial, logrando reducir los tiempos de respuesta de forma considerable, además de que presenta mayor confiabilidad a la hora de transmitir los datos.

Volviendo a lo relacionado con la interfaz de usuario, cuando el robot se enciende, se mostrará el mensaje “robot iniciado”. Se ingresará automáticamente al “MODO DE CONTROL REMOTO” donde su comportamiento es similar a un auto de radio control (donde cada carácter hace que el robot realice un movimiento). Para ingresar al “CLI” (Interfaz de línea de comandos o Command Line Interface) se debe enviar el comando ‘!’ y se recibirá la frase “CONSOLA INICIADA” como señal de respuesta y el robot habrá pasado al “MODO CONSOLA”. Mediante el envío de otro comando, se podrá acceder al “MODO DE AUTO NAVEGACIÓN”. Dentro del “MODO DE AUTO NAVEGACIÓN”, se podrá seguir utilizando el “CLI”, pero solo se tiene acceso a los comandos relacionados a dicho modo.

Para volver al “MODO DE CONTROL REMOTO”, se debe entrar en el “MODO CONSOLA” y enviar el carácter “\$”.

En la Figura 5.4 se muestra un diagrama que muestra la transición entre los distintos modos de operación del robot.

TRANCISIÓN ENTRE LOS MODOS DE OPERACIÓN

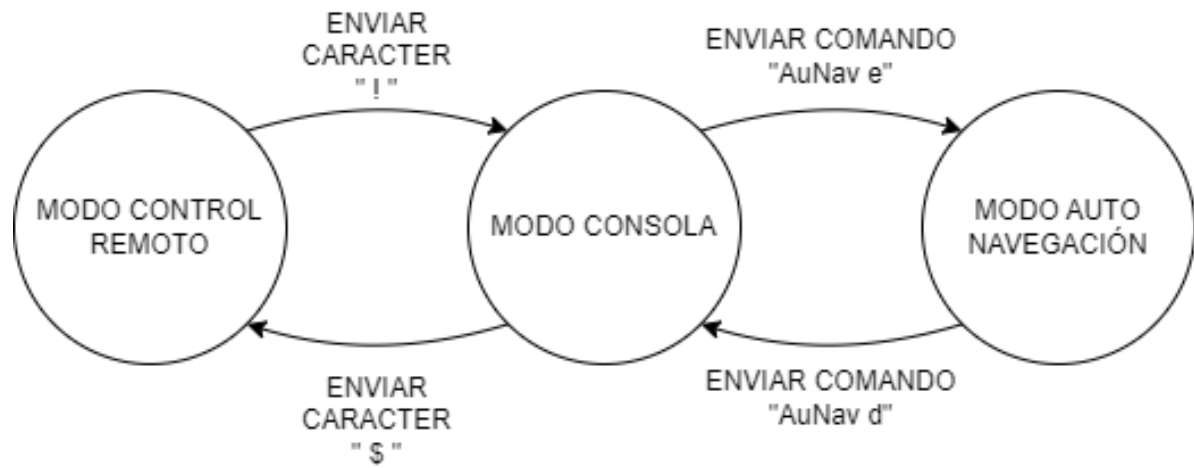
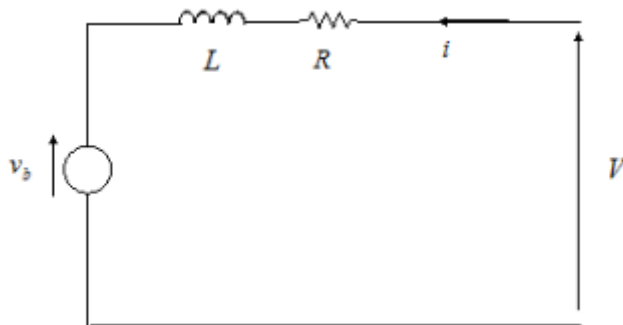


Figura 5.4 - Diagrama de estado de los modos de operación del robot

6. MODELADO Y ANÁLISIS DEL ROBOT

Análisis del motor

El motor DC o a escobillas se representa por medio del siguiente circuito:



Donde

R = Es la resistencia de la armadura de cobre del motor

L = Es la inductancia de la armadura de cobre del motor

V = Es la tensión aplicada al motor

v_b = Es la tensión contraelectromotriz producto de campo magnético del imán permanente del motor que induce en la armadura

v_b depende de la velocidad de giro del motor, se presenta de la siguiente manera:

$$v_b = K_v * \omega \quad (1)$$

donde:

K_v = Es la contratante contraelectromotriz

ω = Es la velocidad angular

Este motor ejerce un torque proporcional a la corriente del motor:

$$T = J * \alpha = K_t * i \quad (2)$$

donde:

K_t = Es la constante de torque

J = Es el momento de inercia en el eje del motor

α = Es la aceleración angular

Aplicando Ley de Kirchhoff de tensión de armadura del motor es

$$V(t) = v_b(t) + R * i(t) + L \frac{di}{dt}$$

En el dominio de Laplace en el circuito del motor se obtiene:

$$V = v_b + R * i + sL * i$$

$$V = v_b + (R + sL) * i \quad (3)$$

De las ecuaciones (1) y (2) las expresó en término de ω y reemplazo en (3).

$$\begin{aligned} v_b &= K_v * \Omega \\ i &= \frac{J}{K_t} * \alpha = \frac{J}{K_t} * s \cdot \Omega \\ V &= K_v * \Omega + (R + s \cdot L) * \frac{J}{K_t} * s \cdot \Omega \\ V &= \left(K_v + (R + s \cdot L) * \frac{J}{K_t} * s \right) \Omega \end{aligned}$$

Para el motor Faulhaber 2342012CR, L es despreciable frente a R , por lo que se elimina L en la ecuación.

$$V = \left(K_v + R * \frac{J}{K_t} * s \right) \Omega$$

Con esto se obtiene la función de transferencia del motor

$$\frac{\Omega}{V} = \frac{\frac{K_t}{J * R}}{s + \frac{K_v * K_t}{J * R}}$$

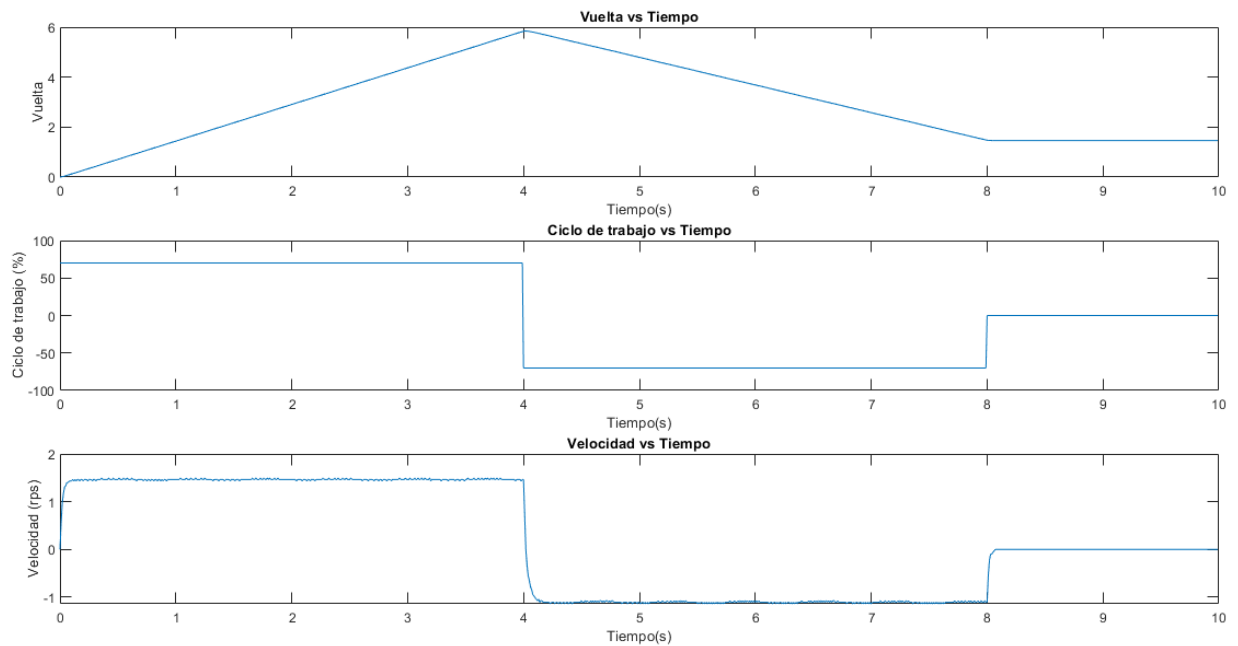
Ensayo del motor

Para realizar el ensayo, se excitó la armadura del motor con una señal PWM de 12V de amplitud y un ciclo de trabajo del 70%. La variable medida, es la velocidad del eje del motor, que se obtiene mediante un encoder. Para tomar estas medidas se decidió crear un programa para el STM32F103C6T8 que genera la señal PWM y decodifica las señales del encoder. La frecuencia de muestreo, es de 100 Hz (dicho valor se eligió en base a tiempo de respuesta del motor). Para asegurar que la comunicación por USART no pierda ningún dato, primero los datos de la medición se guardan en la memoria RAM y una vez terminada la medición (los datos son enviados a la PC a una velocidad más baja). Para realizar la medición, se desconectaron los terminales de TX y RX del módulo de RS485 y se conectó a los terminales RX y TX respectivamente de un conversor USB-Serial para facilitar la comunicación entre el microcontrolador STM32F103C6T8 y la computadora. El programa usado, no forma parte del funcionamiento normal y programa final del robot, se realizó y se empleó para la identificación de modelo del motor.

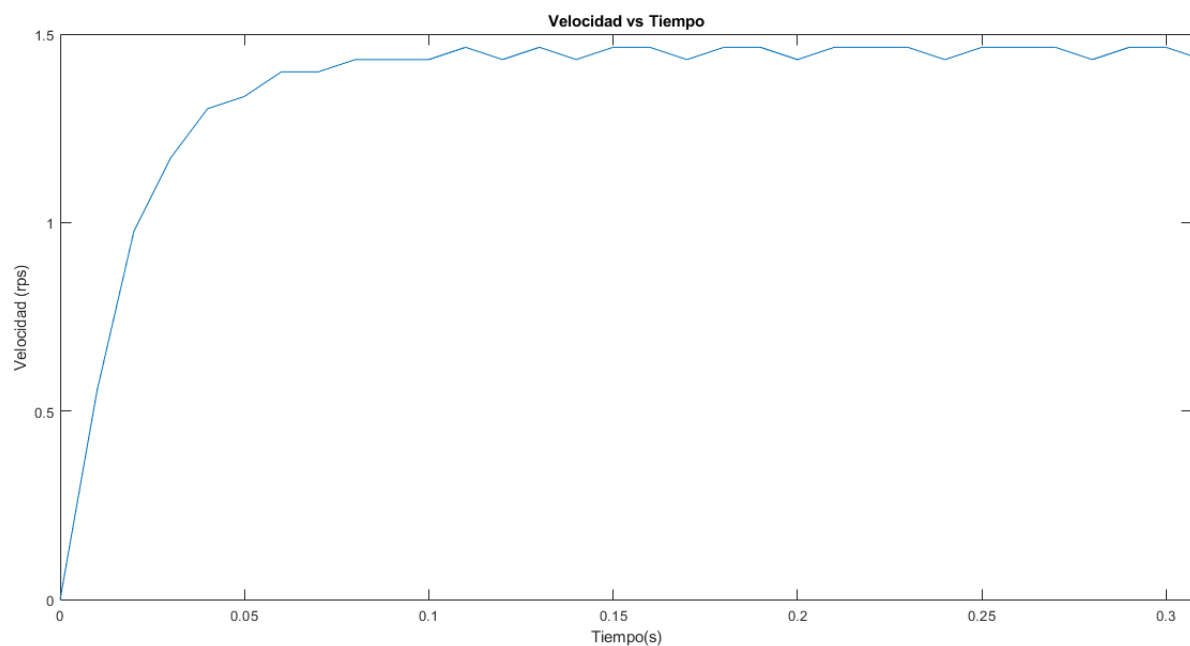
Para comunicarse con el módulo USB-Serial, se usó el programa Tera-Term para el envío, recepción y almacenamiento de los datos.

Los datos se envían en formato CSV (Valores separados por coma), para luego leerlos con un programa .m de Matlab y realizar las gráficas de los datos obtenidos.

Para el ensayo se envía una señal PWM con un ciclo de trabajo de 70% a 10kHz durante 4 segundos, luego se genera otra señal PWM con ciclo de trabajo 70% a 10kHz pero invirtiendo la polaridad, es decir aplicando tensión negativa, durante otros 4 segundos. Finalmente se apaga la señal de excitación (cero voltios), para ver cómo se detiene el motor. Se capturan los datos con el microcontrolador y se importaron en Matlab para analizarlos y obtener las siguientes gráficas.



Para corroborar que se hizo una buena elección en la frecuencia de muestreo, se observa con más detalle el tiempo de establecimiento de la señal al inicio cuando se produce el cambio de tensión.



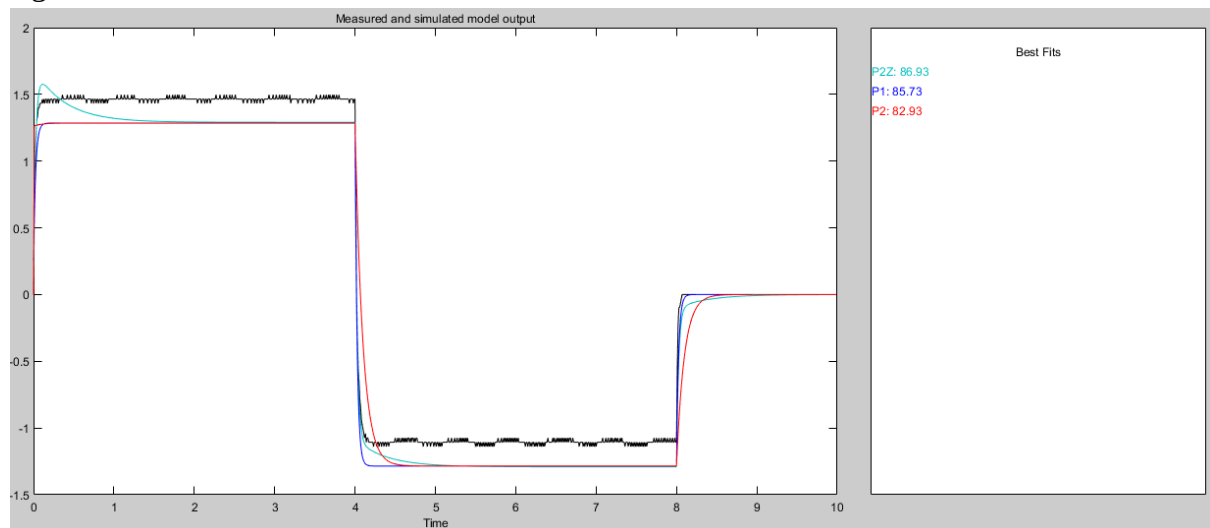
De la gráfica, se observa que el tiempo de establecimiento del sistema es de 0.1 segundos, 10 veces mayor que el tiempo de muestreo de 10 ms. El teorema de muestreo, pide que la frecuencia de muestreo sea por lo menos dos veces mayor a la máxima frecuencia del sistema. Por lo tanto, con 10 veces, se sobrepasa ese valor, asegurando que se capturan todas las dinámicas del sistema.

Algo importante que destacar es que la velocidad del motor, transcurrido el transitorio, varía según la polaridad de la tensión que excita el motor, siendo de ~ 1.5 rps en polarización directa y ~ 1 rps en polarización inversa. Esto es debido a que la orientación de las escobillas influye en el sentido de giro del motor, oponiendo más fricción de un sentido que de otro. Por este motivo se hizo el ensayo variando la polarización del motor, para obtener la función de transferencia que modele en promedio cada aspecto no lineal del motor.

Identificación del sistema

Para realizar la Identificación del sistema (que consiste en encontrar una función de transferencia que modele el comportamiento real del motor con el menor error), se usó el toolbox de Matlab “systemIdentification”. Para introducir los datos al toolbox, primero se debe tener en el “workspace” los arreglos que contienen la velocidad y la tensión de entrada del motor (procedimiento que se describió en la sección anterior).

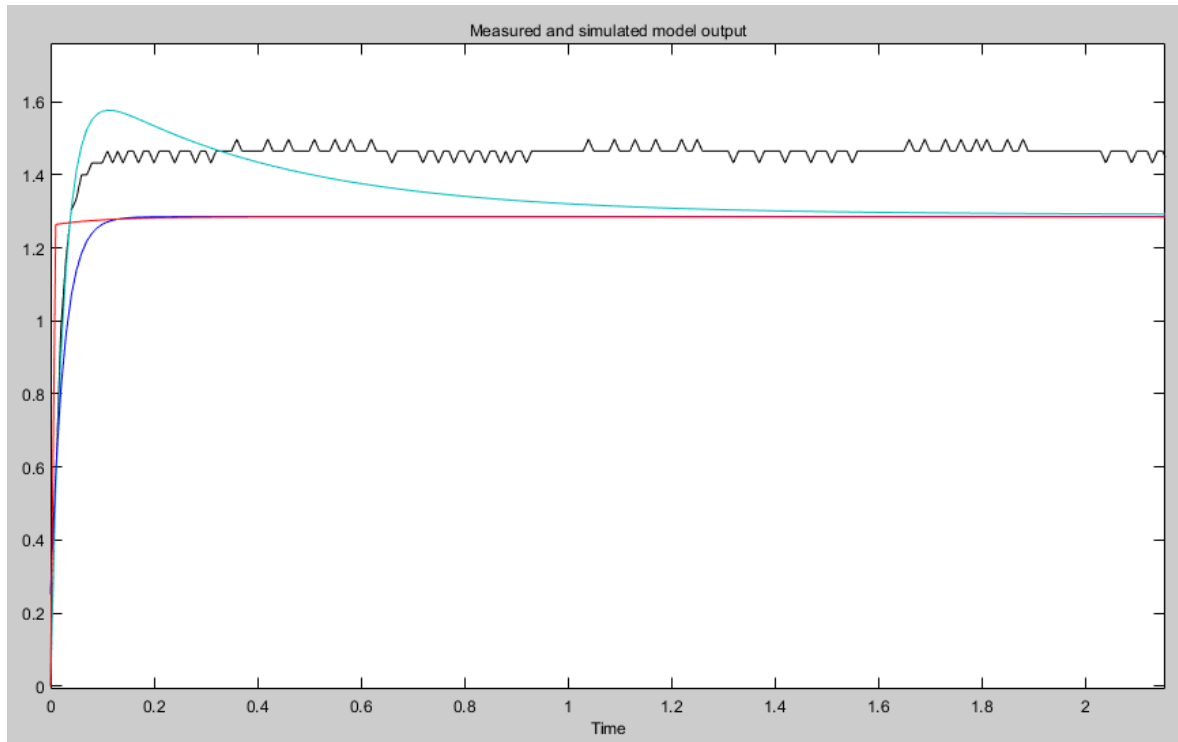
Para identificar el sistema se debe tener una estimación de la cantidad de polos y ceros que tiene el sistema. En base al modelo matemático del motor, se concluye que se puede usar perfectamente una función de transferencia de un solo polo. Sin embargo, también se probó identificar el sistema, usando como modelo matemático de 2 polos reales (en caso de que la inductancia del motor tenga una influencia significativa en el comportamiento del motor), otro modelo con 2 polos reales y un cero. De forma que se puedan comparar varios modelos y comprobar que las conclusiones del análisis anterior son correctas. El toolbox obtuvo el siguiente resultado:



Se observa a la izquierda la gráfica de la función de transferencia que encontró la herramienta aplicando la señal de entrada a los modelos. A la derecha se tiene una lista de las distintas funciones transferencia con los parámetros de polos y ceros que se ingresaron: siendo P1 para

el de 1 polo, P2 para el de 2 polos reales y P2Z para el de 2 polos reales y 1 cero y su respectivo puntaje que representa el nivel de ajuste del modelo con la respuesta real del sistema. A partir de esta gráfica se concluye que:

- Es importante el error cometido por los modelos en el escalón positivo, que no es simétrico al escalón negativo. El problema de la no linealidad del motor debido a aspectos mecánicos, no puede ser modelado con sistemas lineales.
- Aunque el modelo P2Z tiene un puntaje de ajuste más alto, presenta problemas en los transitorios, ya que genera un sobrepico en la respuesta al escalón de tensión, algo que no ocurre en el comportamiento real del motor.



Se concluye que P1 es la función transferencia elegida, teniendo en cuenta su comportamiento transitorio cualitativo similar (solo presenta un error en el valor de amplitud del escalón, pero la forma en general es la misma).

Data/model Info: P1
— □ ×

Model name:

P1

Color:

[0,0,1]

Process model with transfer function:

$$G(s) = \frac{K_p}{1 + T_{p1}s}$$

$K_p = 0.15301$
 $T_{p1} = 0.025644$

Modelo matemático del robot

Para comenzar con el modelo general del robot, primero se debe analizar que sucede con una rueda mecanum al comenzar a girar.

Las ruedas mecanum se destacan por poseer unos rodillos sobre la superficie de rodamiento de la rueda, estos rodillos tienen una orientación de 45° respecto al eje de la rueda y pueden girar libremente.



Suponiendo que una rueda mecanum se encuentra girando sin deslizamiento sobre una superficie a una velocidad V :

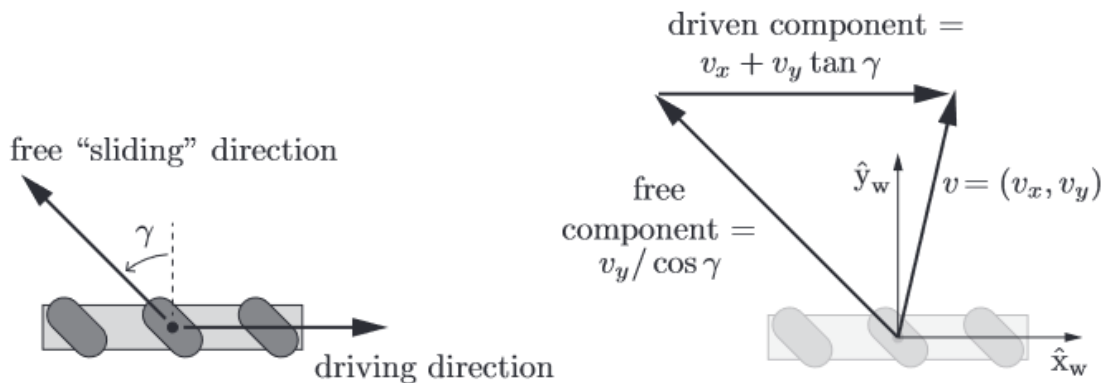
Como no hay deslizamiento V es igual a la suma de la velocidad de giro de la rueda $V_{rueda} = \omega_{rueda} * R_{rueda}$ y la velocidad de giro de los rodillos $V_{rodillos}$.

$$V = V_{rueda} + V_{rodillos}$$

Descomponiendo V en sus respectivas componentes de X e Y

$$V_x = V_{rueda} + V_{rodillos} * \sin(\gamma)$$

$$V_y = V_{rodillos} * \cos(\gamma)$$



En base a esto, se despeja V_{ruedas} para que quede en términos de la velocidad de movimiento

$$V_{rueda} = V_x - V_{rodillos} * \sin(\gamma); V_{rodillos} = V_y / \cos(\gamma)$$

$$V_{rueda} = V_x + V_y * \tan(\gamma) = \omega_{rueda} * R$$

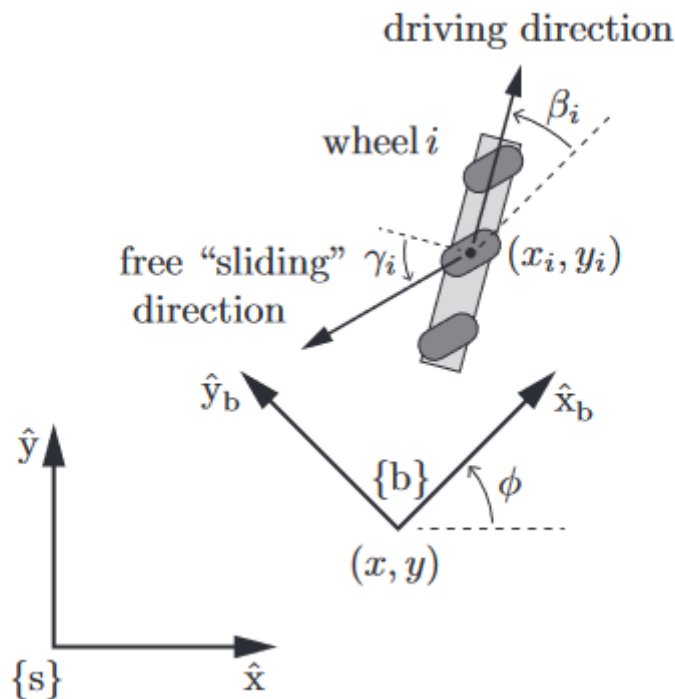
Se reescriben las fórmulas que representan el modelo del robot de forma matricial para obtener una representación matemática más clara y entendible

$$V_{rueda} = \omega_{rueda} * R = \begin{bmatrix} 1 & -\tan(\gamma) \end{bmatrix} * \begin{bmatrix} V_x \\ V_y \end{bmatrix}$$

$$\omega_{rueda} = \frac{1}{R} * \begin{bmatrix} 1 & -\tan(\gamma) \end{bmatrix} * \begin{bmatrix} V_x \\ V_y \end{bmatrix}$$

Considerando que la rueda se encuentra unida mecánicamente al robot, el centro de coordenadas de la rueda y la orientación de sus ejes X e Y puede no coincidir con las del robot. Por lo que se debe realizar la conversión hacia el sistema de coordenadas del robot.

Mediante la siguiente figura, se observa que la rueda posee una rotación β respecto al sistema de coordenadas del robot {b}.



Entonces las componentes V_x y V_y se deben pasar a {b} por lo que:

$$V_x = \cos(\beta) * V_{xb} + \sin(\beta) * V_{yb}$$

$$V_y = -\sin(\beta) * V_{xb} + \cos(\beta) * V_{yb}$$

Reemplazando estas ecuaciones en la ecuación de ω_{rueda} , se puede tener relaciones entre la velocidad de giro de la rueda con su desplazamiento visto desde el sistema de coordenadas del robot.

$$\omega_{rueda} = \frac{1}{R} * \begin{bmatrix} 1 & -\tan(\gamma) \end{bmatrix} * \begin{bmatrix} \cos(\beta) & \sin(\beta) \\ -\sin(\beta) & \cos(\beta) \end{bmatrix} * \begin{bmatrix} V_{xb} \\ V_{yb} \end{bmatrix}$$

Ahora que se tienen las ecuaciones que relacionan la velocidad de giro de la rueda y su velocidad desde el punto de vista del robot, se debe expresar la velocidad de giro del robot con las maniobras o movimientos del robot. El robot posee una movilidad “planar”, es decir, que es capaz de ir hacia adelante, atrás, izquierda, derecha y girar sobre su propio eje por lo que al hacer cada una de estas maniobras, cada una de las ruedas del robot debe poseer una cierta velocidad para realizar dicha acción.

Se analizará esto, teniendo en cuenta que el centro de la rueda se encuentra en el punto P:(X_i , Y_i) respecto al centro del robot. Si el robot se desplaza sobre el eje X o el eje Y , la velocidad de la rueda coincide con la del robot

$$V_{xb} = V_{xR}$$

$$V_{yb} = V_{yR}$$

Ahora teniendo en cuenta que el robot gira, la rueda poseerá una velocidad producto de esa rotación que viene dada por:

$$V_{xb} = -P * \sin(P_{angulo}) * \omega_R = -Y_i * \omega_R$$

$$V_{yb} = P * \cos(P_{angulo}) * \omega_R = X_i * \omega_R$$

Entonces la velocidad de las ruedas respecto al movimiento del robot quedaría expresada de esta forma:

$$V_{xb} = -Y_i * \omega_R + V_{xR}$$

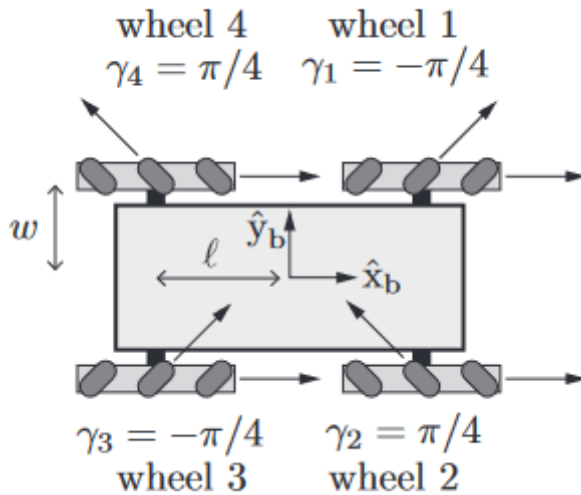
$$V_{yb} = X_i * \omega_R + V_{yR}$$

La fórmula completa es:

$$\omega_{rueda} = \frac{1}{R} * [1 \quad -\tan(\gamma)] \begin{bmatrix} \cos(\beta) & \sin(\beta) \\ -\sin(\beta) & \cos(\beta) \end{bmatrix} * \begin{bmatrix} -Y_i & 1 & 0 \\ X_i & 0 & 1 \end{bmatrix} * \begin{bmatrix} W_\theta \\ V_{xR} \\ V_{yR} \end{bmatrix}$$

Ahora se tiene la relación entre la velocidad de giro de las ruedas y el movimiento del robot para un sistema de coordenadas cuyo origen se encuentra en el centro del robot. Con esto realizado, al aplicarlo a las 4 ruedas del robot para obtener la fórmula matemática, se obtiene la relación entre las 4 ruedas del motor y su posición.

A continuación, se muestra un diagrama del robot con sus ruedas



se observa que $\beta = 0$ y existen 2 ruedas que poseen $\gamma = \pi/4$ y dos que $\gamma = -\pi/4$

$W_1 = W_{di} = W$ de la rueda delantera izquierda

$W_2 = W_{dd} = W$ de la rueda delantera derecha

$W_3 = W_{td} = W$ de la rueda trasera derecha

$W_4 = W_{ti} = W$ de la rueda trasera izquierda

por lo que aplicado las fórmulas a las ruedas del motor queda:

$$\omega_{rueda} = \frac{1}{R} * \begin{bmatrix} 1 & -\tan\left(\pm\frac{\pi}{4}\right) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \mp w & 1 & 0 \\ \pm l & 0 & 1 \end{bmatrix} * \begin{bmatrix} W_{\theta} \\ V_{xR} \\ V_{yR} \end{bmatrix}$$

Aplicado para cada rueda quedaría

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = \frac{1}{R} * \begin{bmatrix} -l-w & 1 & 1 \\ l+w & 1 & -1 \\ -l-w & 1 & -1 \\ l+w & 1 & 1 \end{bmatrix} * \begin{bmatrix} W_{\theta} \\ V_{xR} \\ V_{yR} \end{bmatrix}$$

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = \frac{1}{R} * H * \begin{bmatrix} W_{\theta} \\ V_{xR} \\ V_{yR} \end{bmatrix}$$

Con esto se puede despejar W_{θ} , V_{xR} y V_{yR} para dejarlos en función de la velocidad de movimiento de las ruedas y obtener una manera de medir la posición del robot utilizando los encoders de los motores y sabiendo su velocidad de giro.

$$\begin{bmatrix} W_{\theta} \\ V_{xR} \\ V_{yR} \end{bmatrix} = \frac{R}{4} * \begin{bmatrix} -\frac{1}{l+w} & \frac{1}{l+w} & -\frac{1}{l+w} & \frac{1}{l+w} \\ 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix}$$

Cuando el robot se mueve, puede ser difícil para la persona que lo controla, tener un punto de referencia claro. Ya que es más natural para una persona entender su ubicación en relación con su entorno, que entender la posición del robot en movimiento. Para resolver este problema, se transfiere el sistema de coordenadas del robot {R} a un sistema de coordenadas fijo {S} que no cambia.

$W_s = W_R$ porque la rotación es la misma sin importar el sistema de referencia

$$V_{xR} = V_{xS} * \cos(\theta) + V_{yS} * \sin(\theta)$$

$$V_{yR} = -V_{xS} * \sin(\theta) + V_{yS} * \cos(\theta)$$

escrito en manera matricial queda expresado de la siguiente manera

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = \frac{1}{R} * H * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{bmatrix} W_{\theta} \\ V_{xS} \\ V_{yS} \end{bmatrix}$$

ahora se necesita el pasaje de {R} a {S}

$$V_{xS} = V_{xR} * \cos(\theta) - V_{yR} * \sin(\theta)$$

$$V_{yS} = V_{xR} * \sin(\theta) + V_{yR} * \cos(\theta)$$

$$\begin{bmatrix} W_\theta \\ V_{xS} \\ V_{yS} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{bmatrix} W_\theta \\ V_{xR} \\ V_{yR} \end{bmatrix}$$

7. SISTEMA DE CONTROL

El robot posee 4 motores DC, que se pueden manejar en forma independiente para poder desplazarse en su entorno. A partir de la posición y orientación deseada del robot, se generan las señales de control de los motores (señales PWM). En la Figura 7.1 se muestra el esquema de control del robot, se aplicará un sistema de control en cascada, donde un sistema de control general, generará las referencias para el sistema de control individual de cada motor.

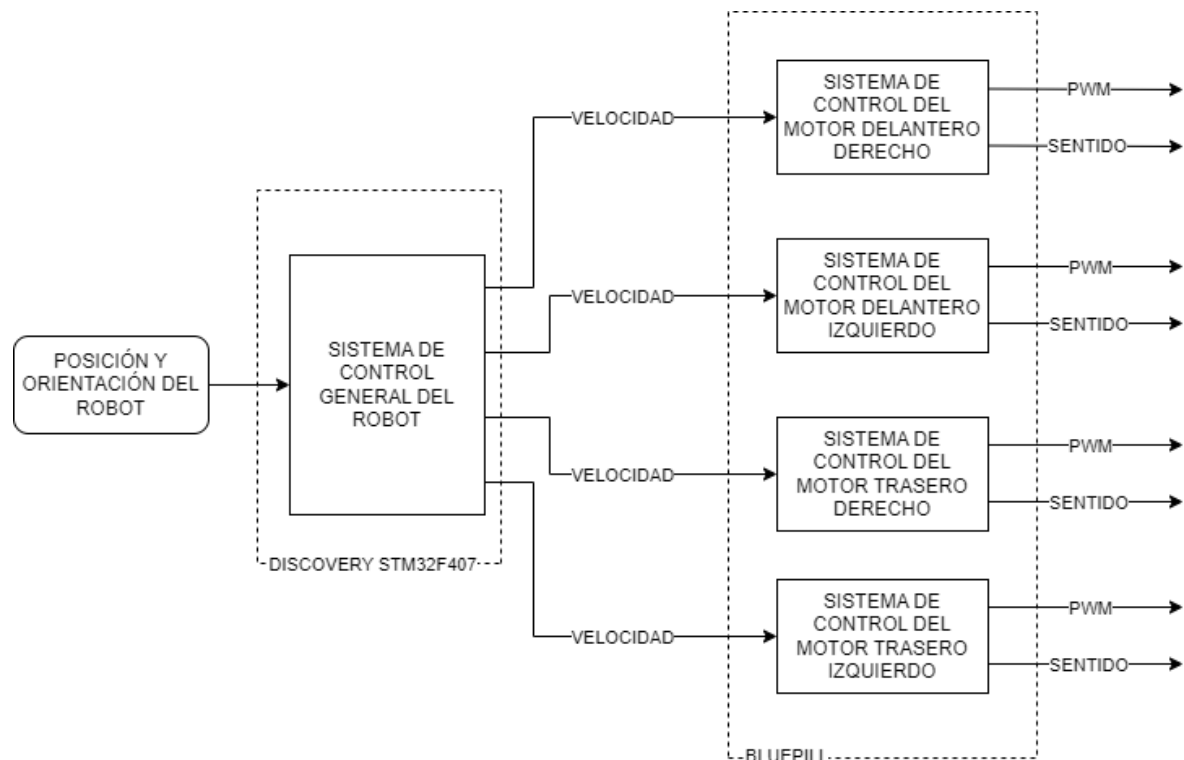


Figura 7.1 - Diagrama que muestra cómo están relacionados todos los sistemas de control del robot y desde qué placa se ejecuta cada sistema de control.

Las placas Bluepill que poseen el microcontrolador STM32F103, son las que se encargarán de realizar el control de los motores. Cada placa Bluepill, maneja dos motores a la vez. Mientras tanto la placa Discovery, se encargará de realizar el sistema de control general del robot, que es la que se encargará de manejar la posición y la orientación del robot.

Sistema de control del motor

Una vez terminado los ensayos y obtenida la función de transferencia del motor se procederá al diseño de un controlador en este caso se utilizará un controlador PI (proporcional para realizar la acción de control e integral para corregir cualquier error producido por los efectos de carga en el eje del motor cuando esté funcionando). Se agregó al controlador un bloque anti-windup, para que el término integral no afecte negativamente a la acción proporcional. La acción

integral, realiza una corrección ante los efectos externos (por ejemplo, al añadir más carga del motor). El efecto del bloque anti-windup se observa cuando el motor debe realizar un cambio brusco en su velocidad, como pasar de estar en reposo a alcanzar la velocidad máxima. En estos casos, el término integral del controlador puede acumular un error significativo, lo que, al acercarse al valor del setpoint, obliga al sistema a compensar dicho error acumulado, pudiendo generar sobrepicos en la respuesta. El algoritmo de anti-windup que utiliza el controlador, es el “Clamping” que consiste en detener la acumulacion del término integral cuando se detecta que la señal de entrada a la planta se encuentra cercano a su límite (en este caso sería cuando la tensión a aplicar al motor es cercana a los 12V). De esta manera se utilizan parámetros del controlador PI que dan una muy buena respuesta del sistema, sin producir sobrepicos y mejorando el tiempo de respuesta del motor.

En la Figura 7.2 se muestra el sistema de control que se armó en simulink para realizar la sintonización del controlador PI, y en la figura 7.3 se muestra una captura donde se muestran los coeficientes obtenidos.

Para la medición de la velocidad del motor se utilizará los encoder junto con el módulo de timer del microcontrolador STM32F103, dicho módulo tiene la opción de decodificar la señal en cuadratura que envía el encoder hacia el microcontrolador. Calculando la diferencia finita entre la cuenta actual y la cuenta anterior, se determina la velocidad del motor.

La señal de salida del controlador será la tensión que se debe aplicar en el motor, que luego el microcontrolador lo transformará a una señal PWM. El ciclo de trabajo de la señal PWM, será determinado por el valor de la tensión en el motor, además se determinará el sentido de giro motor en función del signo de la tensión a aplicar. En la Figura 7.4 se muestran las gráficas obtenidas en Simulink de la señal de entrada y salida del controlador.

Para la sintonización de los coeficientes del controlador se utilizará la herramienta Simulink utilizando la función de “autotune”.

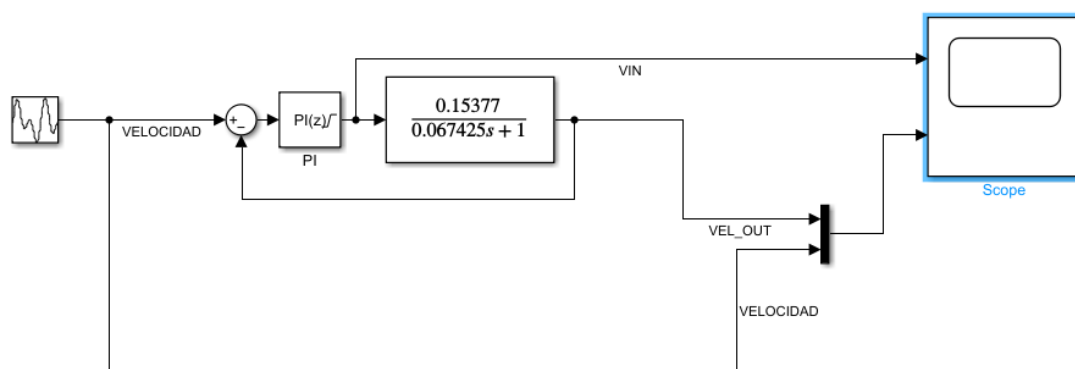


Figura 7.2 - Sistema de control de velocidad del motor, realizado en Simulink

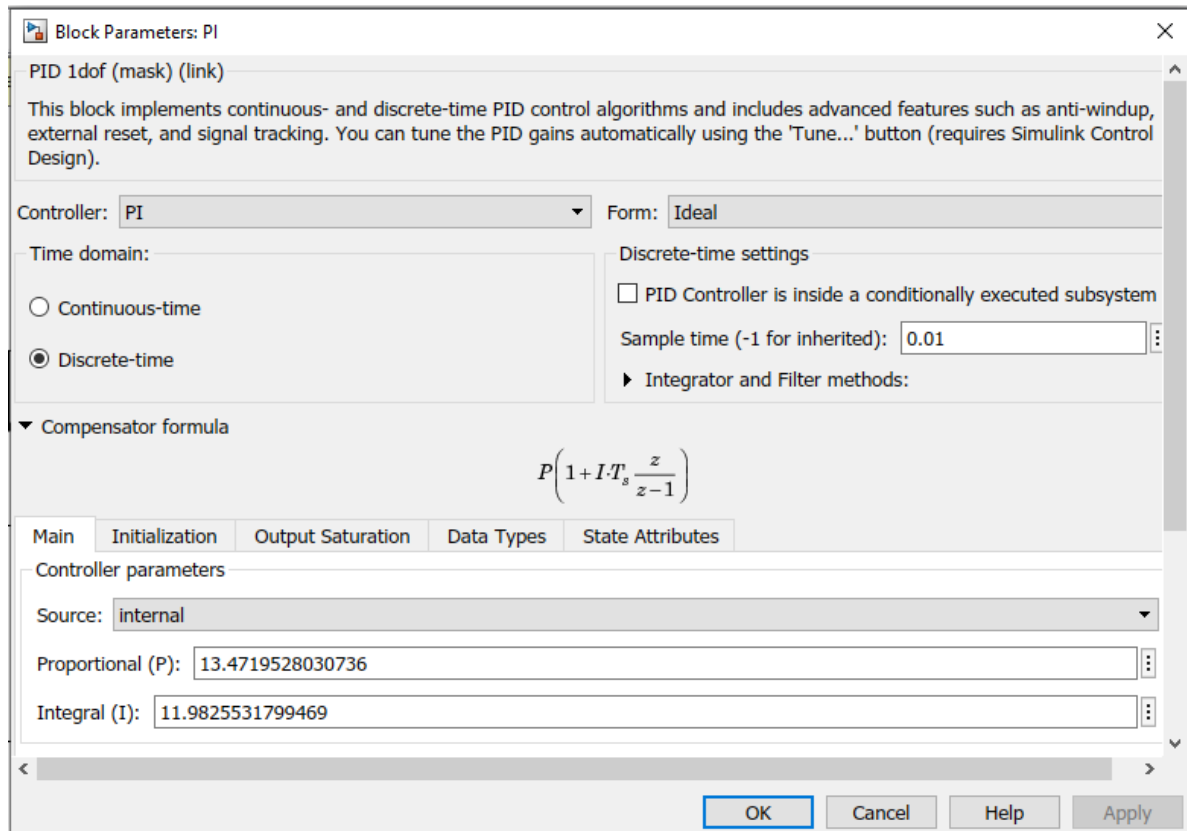


Figura 7.3 - Ventana del software Simulink donde se muestra la configuración del bloque de controlador PI

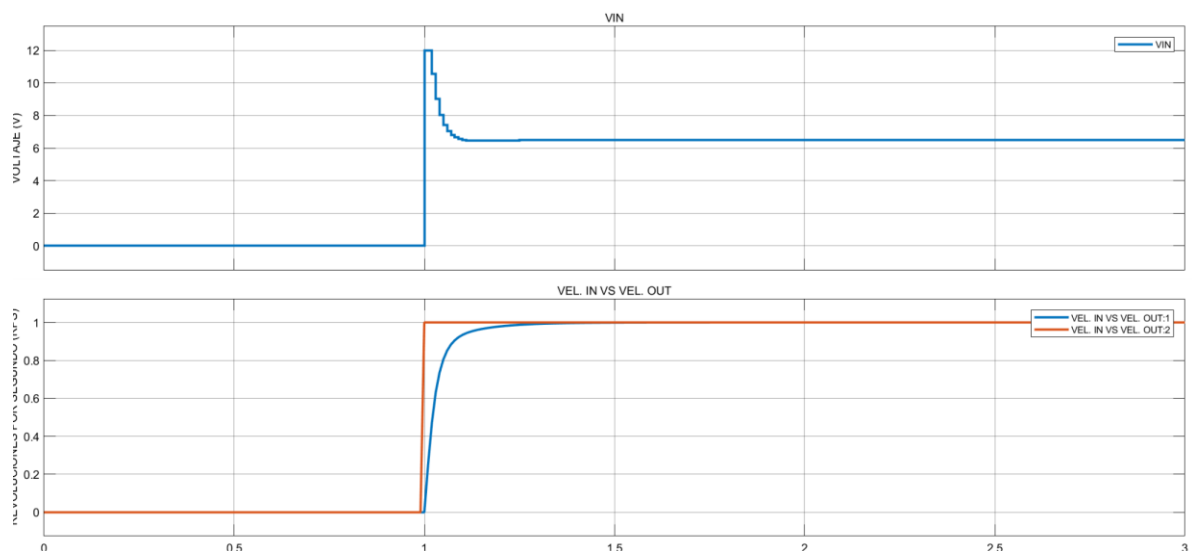


Figura 7.4 - Gráfica de las señales de entrada, salida y la tensión de alimentación de la planta

A partir del análisis del modelo del controlador simulado en Matlab y habiendo obtenido los parámetros de la respuesta deseada, se procedió a implementar en el sistema de control de motores programado en las placas Bluepill haciendo uso de las librerías descritas en el capítulo 9.

Al hacer las pruebas con los motores se puede observar un buen comportamiento de los mismos. Esto nos permite continuar al paso siguiente.

Sistema de control de posición del motor

Aunque el controlador desarrollado en esta sección no se usó en la versión final del control del robot, se usa en forma particular para un caso especial. Cuando el robot se configura en modo CONSOLA, se usa el controlador para posicionar el eje del motor en una posición determinada.

Para realizar el control de la posición del eje de uno de los motores, se realizó un control en cascada. El control en cascada combina un control de velocidad, seguido de un controlador PD (proporcional derivativo) que regula la posición. El objetivo de que el controlador de posición posea un término derivativo, es que la pendiente de la rampa generada por la acción de control del término proporcional no sea muy alta. Logrando que el cambio de velocidad sea suave. Un cambio de velocidad muy rápido, producirá picos de corriente muy elevados en el motor, reduciendo su vida útil. En la Figura 7.5 se muestra el sistema de control de posición del motor realizado en el programa Simulink

Teniendo en cuenta que el motor no posee la misma velocidad máxima girando en sentido horario que antihorario, se decidió saturar la señal de salida del controlador PD para asegurar que el motor tendrá un funcionamiento similar sin importar el sentido de giro. Además, se evita que el término integral del controlador PI, acumule errores muy grandes.

Para la determinación de los coeficientes del controlador se utilizó la herramienta Simulink de la misma forma que se usó en el caso anterior. En la Figura 7.6 se muestra una captura de la ventana del Simulink donde se muestra los coeficientes obtenidos y en la Figura 7.7 se muestra las gráficas obtenidas, midiendo la velocidad de salida del motor, la señal de entrada y salida de la posición del eje del motor y la tensión de alimentación del motor.

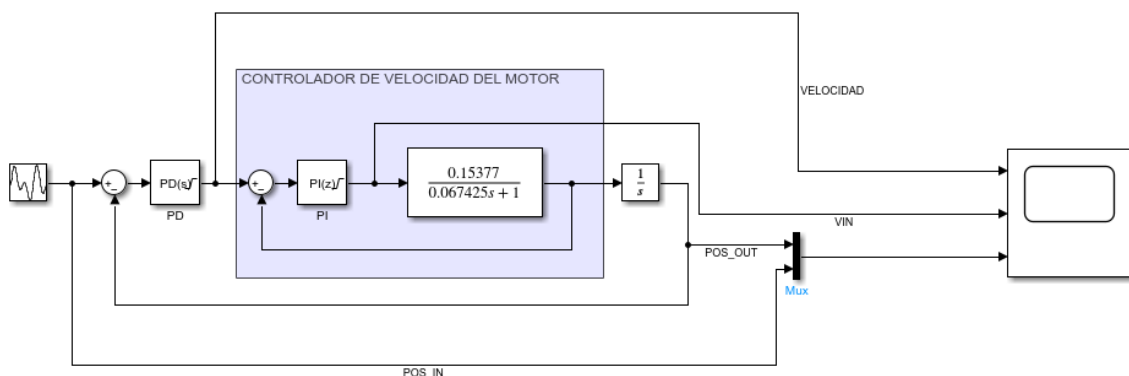


Figura 7.5 - Sistema de control de velocidad del motor, realizado en Simulink

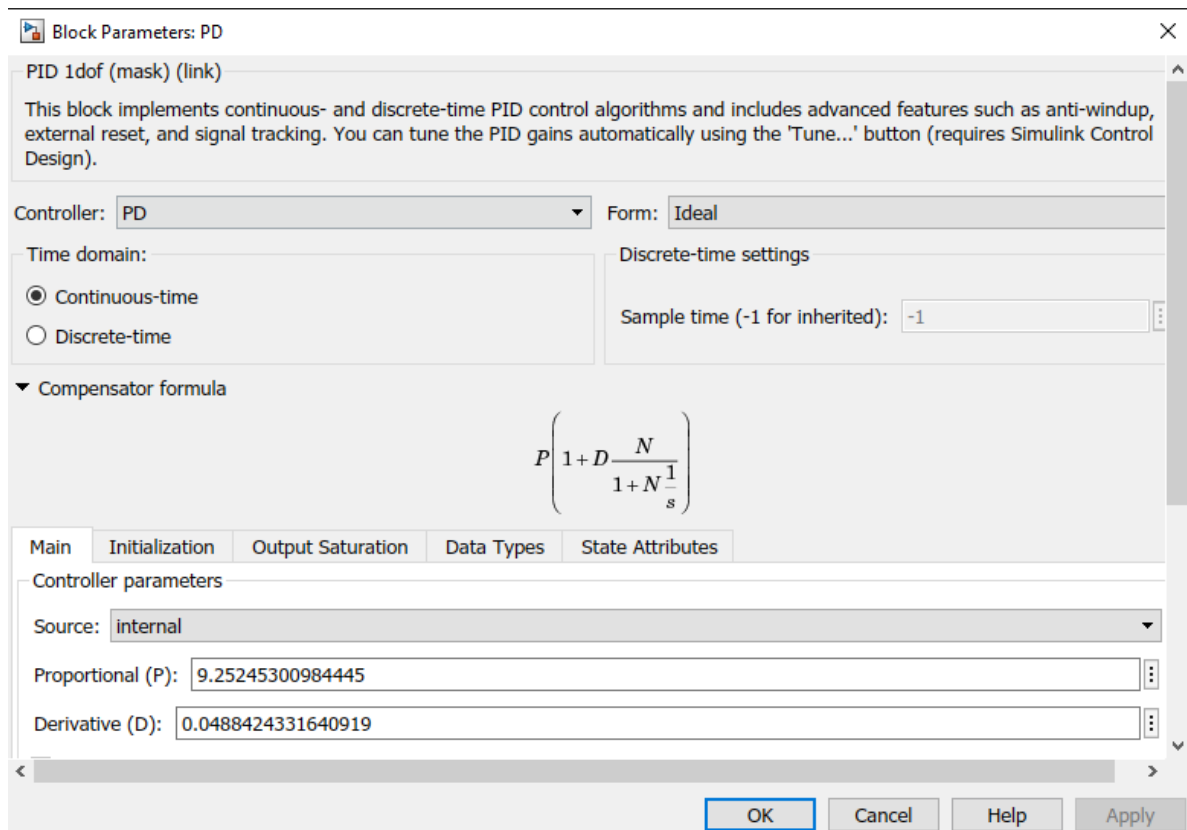


Figura 7.6 - Ventana del software Simulink donde se muestra la configuración del bloque de controlador PI

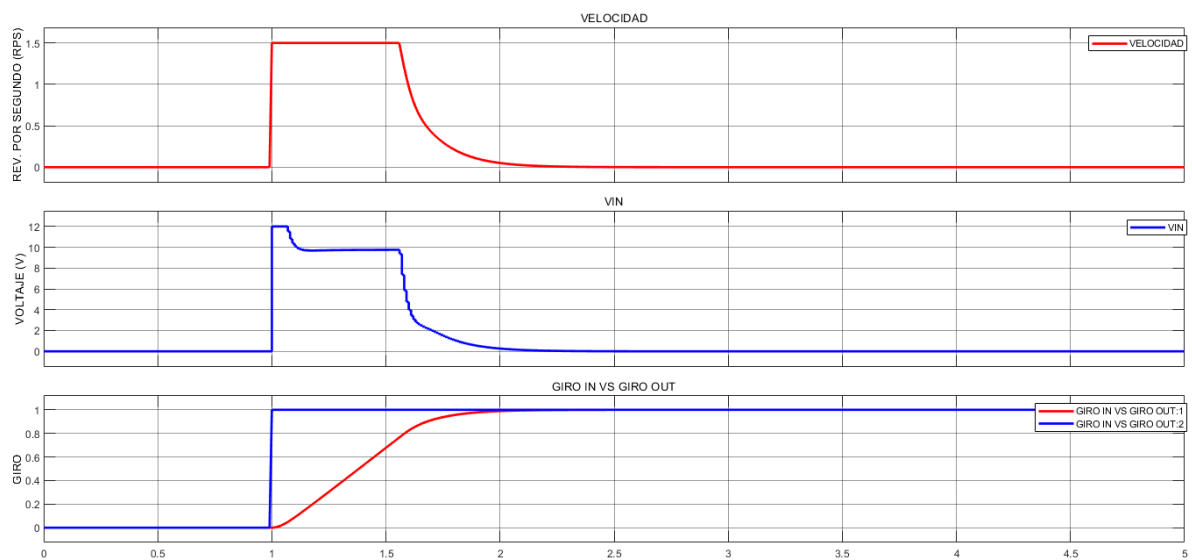


Figura 7.7 - Gráfica de las señales velocidad de salida del motor, la señal de entrada y salida de la posición del eje del motor y la tensión de alimentación del motor

Haciendo el análisis de la respuesta temporal del controlador, se obtuvo un comportamiento deseado. Los parámetros que cumplen con esto, han sido cargados en los sistemas de control

programados en las placas Bluepill, y a partir de ello se realizaron pruebas, verificando que el resultado obtenido es similar a lo obtenido en la simulación.

Este controlador se implementó inicialmente con la intención de desarrollar un sistema de control de posición del robot basado en él. Sin embargo, surgió un problema: el deslizamiento o resbalamiento de las ruedas sobre el suelo generaba una pérdida constante de precisión, lo que obligaba a reiniciar el sistema cada vez que esto ocurría. Por esta razón, se diseñó un sistema más complejo para gestionar la posición y orientación del robot, que se detalla a continuación.

Sistema de control general del robot

En el capítulo 6, donde se desarrolla el modelo matemático del robot, se obtuvo una relación entre las tres variables relacionadas con la velocidad de giro de robot y la velocidad de desplazamiento en el eje X e Y, y la velocidad de las 4 ruedas del robot. Con esto se realiza un controlador que maneja la orientación del robot y su posición en el eje X e Y.

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = H' * \begin{bmatrix} W_{\theta} \\ V_x \\ V_y \end{bmatrix}$$

$$\begin{bmatrix} W_{\theta} \\ V_x \\ V_y \end{bmatrix} = H'^{-1} * \begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix}$$

$$\begin{bmatrix} \theta(k) \\ X(k) \\ Y(k) \end{bmatrix} = \delta t * \begin{bmatrix} W_{\theta} \\ V_x \\ V_y \end{bmatrix} + \begin{bmatrix} \theta(k-1) \\ X(k-1) \\ Y(k-1) \end{bmatrix}$$

Con esto se puede desarrollar un conjunto de controladores PID, capaces de controlar la posición y orientación del robot. En la Figura 7.8 se muestra como está realizado el sistema de control de posición y orientación del robot.

Es importante tener en cuenta que no se posee una manera confiable de saber la posición real del robot, ya que la forma de obtener su posición es por medio de los encoders. Pero se debe considerar que este modelo funciona siempre y cuando las ruedas no se deslicen. Dicho deslizamiento se puede dar si el piso no está liso, o limpio, o no hay buen agarre de las ruedas al piso. Por este motivo se agregaron sensores, uno de ellos es el IMU, para verificar la orientación del robot. También se tiene un sensor ultrasónico para verificar la orientación y la posición del eje Y del robot (cuando es posible). Pero no se tiene un sensor fiable para poder verificar la posición del robot en su entorno (por ejemplo, un sensor LIDAR podría hacer este trabajo). Es decir que la posición en el eje X e Y pueden estar comprometidas cuando se está probando el robot en un escenario real, sin embargo, cuando se usa una simulación, estos problemas no existen.

Con esto en mente, se realizó el modelo completo en Simulink para las simulaciones y sintonizar los controladores teniendo en cuenta dos puntos importantes: primero la de limitar la velocidad de giro de la rueda a ± 1 rps y el segundo es tener en consideración en la simulación el “dead-zone” del motor. El “dead-zone”, es el rango de valores de entrada donde el motor no funcionará, esto ocurre en valores de entrada cercanos al cero, donde el motor a pesar de recibir tensión no puede pasar de estar en reposo a estar en movimiento debido a que no tiene la suficiente fuerza para moverse. Mediante pruebas se observó, que para velocidades de 0.1 rps el sistema de control del motor no funciona correctamente por culpa del “dead-zone” por lo que se decidió colocar un bloque que simulara este comportamiento en el Simulink.

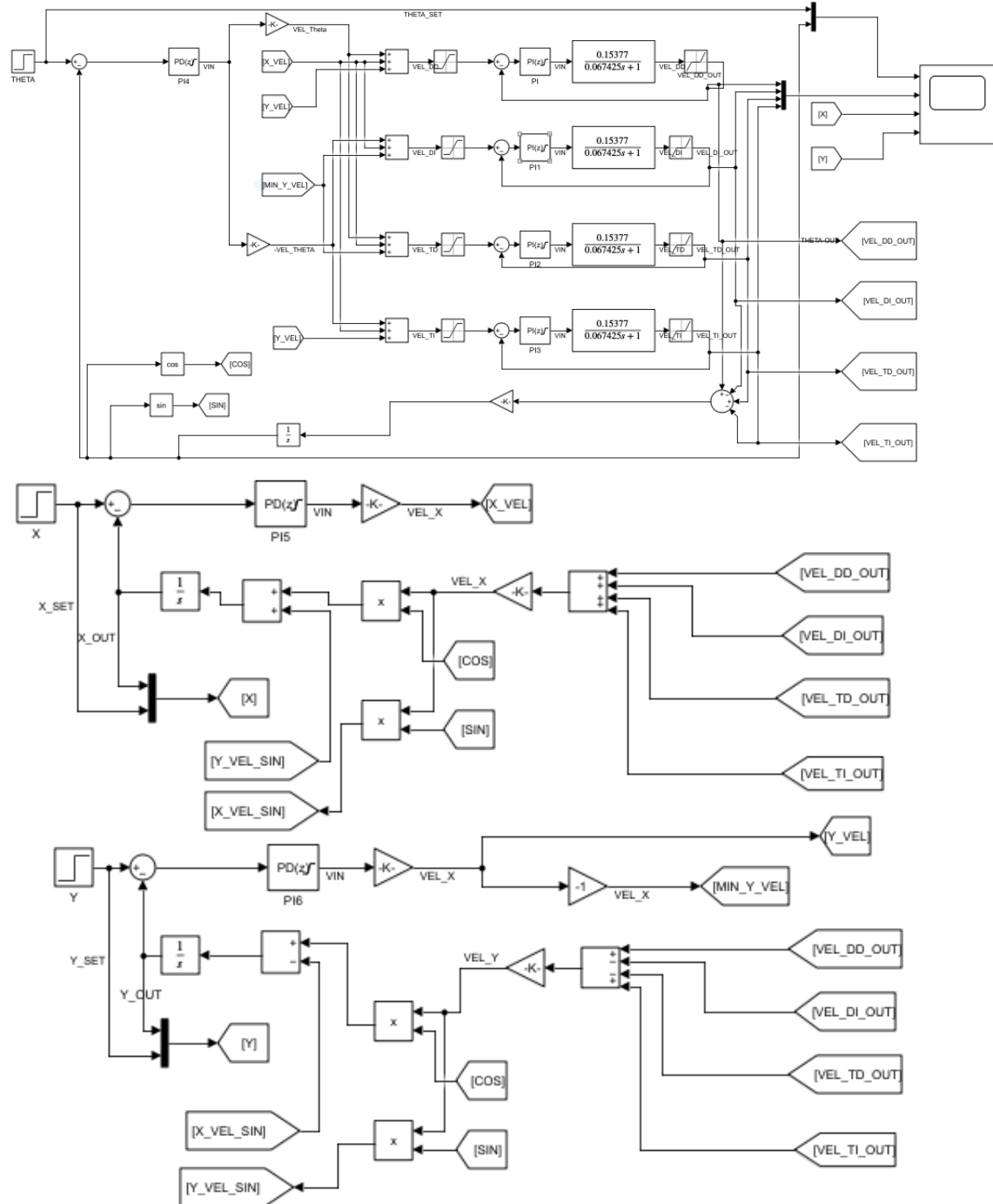


Figura 7.8 - Sistema de control de posición y orientación del robot, realizado en Simulink

Para este caso, donde existen varios controladores PID que se relacionan entre sí, es probable que la herramienta “Autotune” no funcione correctamente. Por lo que se procedió a determinar los coeficientes por prueba y error.

En la Figura 7.9 se muestran las gráficas de las entradas y salidas de las variables a controlar y de la señal de entrada a los controladores de velocidad de cada motor, luego de realizar la sintonización del robot.

Una vez obtenidos los primeros resultados, se realizaron las pruebas del controlador y se observó que, una vez que se llegaba al setpoint, los motores realizan pequeños movimientos bruscos. Analizando el sistema y las variables internas del código del controlador PID en el microcontrolador, se observó que el motor no llegaba exactamente al valor de setpoint, sino que existía un pequeño error que hacía que el término integral siguiera acumulando valor hasta el punto de realizar estos pequeños movimientos bruscos. El motivo del porque existía este pequeño error respecto al setpoint es debido al efecto de dead-zone del motor, dicho efecto se manifiesta de la siguiente manera: cuando el motor recibe una tensión de alimentación pequeña, genera una fuerza de torque que no es suficiente para vencer la fricción interna del motor haciendo que el motor no funcione. Como consecuencia del “dead-zone”, el motor presenta dificultades para girar a una velocidad pequeña y no permite que el motor llegue al valor de setpoint. Una reducción en los valores numéricos de los coeficientes integrales de los controladores PID hubiera logrado una reducción de este tipo de movimientos bruscos en el robot, pero a costa de perder tiempo de reacción en el motor. Para evitar que esto suceda, se decidió agregar una lógica extra al cálculo del término integral de los controladores. La misma consiste en que si el valor de salida de la planta es lo bastante cercana al valor de setpoint (es decir, cuando el error sea muy pequeño), este dejará de seguir acumulando error. Mediante ensayos se pudo determinar este valor de error para lograr que no se pierda precisión en el sistema, eliminar estos movimientos no deseados y conservar los tiempos de respuesta del motor.

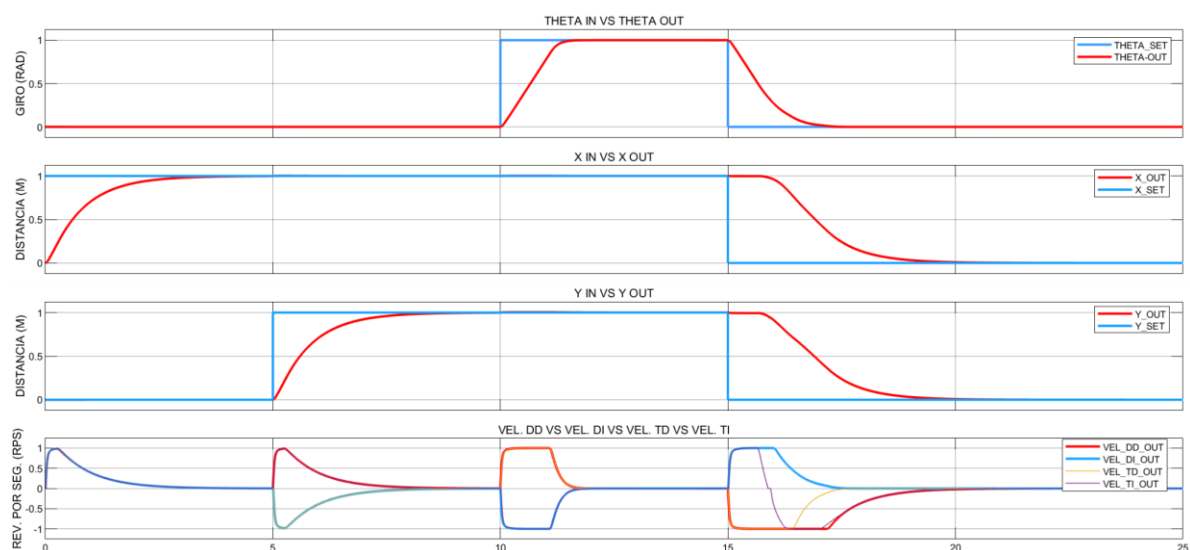


Figura 7.9 - Gráficas de las señales de entrada y salida de las variables de control y las señales de entrada para los controladores de velocidad de cada motor

8. CIRCUITOS ELÉCTRICOS DEL ROBOT

Circuito de las placas de adaptación de los encoders

Este circuito cumple la función de adaptar la señal captada por el sensor óptico y convertirla en una señal de 5V, lo más cuadrada posible para lograr una interpretación muy buena por parte del microcontrolador. En la Figura 8.1 se muestra el circuito eléctrico.

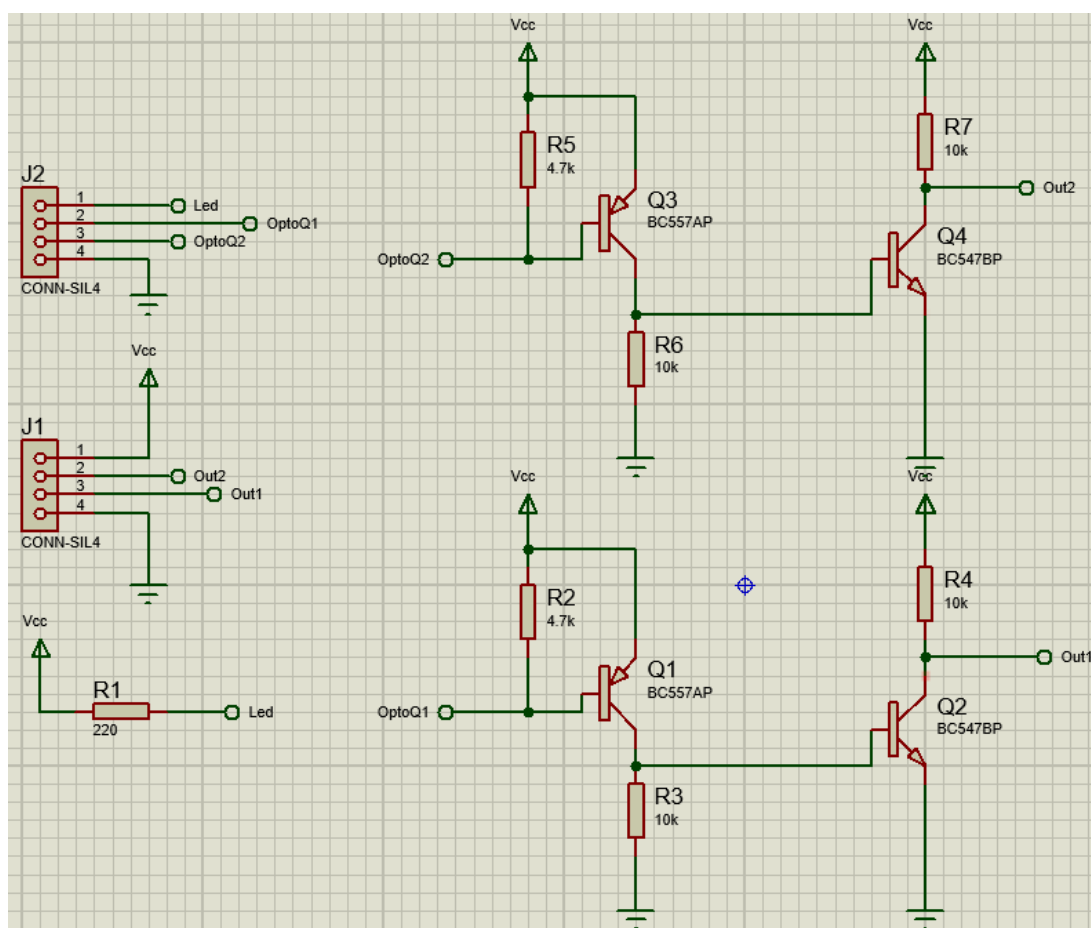


Figura 8.1 - Circuito de las placas de adaptaciones de los encoders

Los componentes de dicha placa son:

Transistores: dos BC547 y dos BC557.

Resistencias: una de 220 Ohms, dos de 4K7 y cuatro de 10K.

Los mismos han sido evaluados por ensayos en el laboratorio. Logrando así una señal óptima que ayude a calcular velocidades y posiciones del robot. En la Figura 8.2 se muestra una foto del circuito, armado en una protoboard.

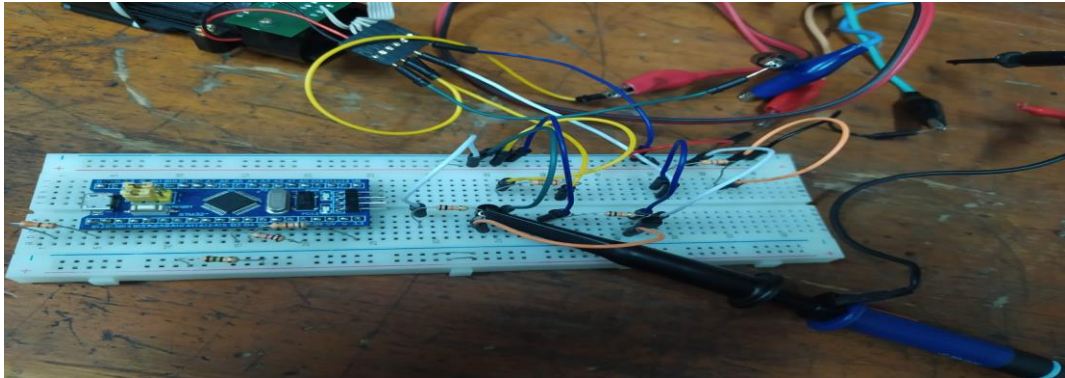


Figura 8.2 - Pruebas del circuito en una protoboard.

Se fue ensayando el circuito, cambiando las ganancias a través de la configuración de resistencias. En la Figura 8.3 se muestran los resultados obtenidos por el osciloscopio.

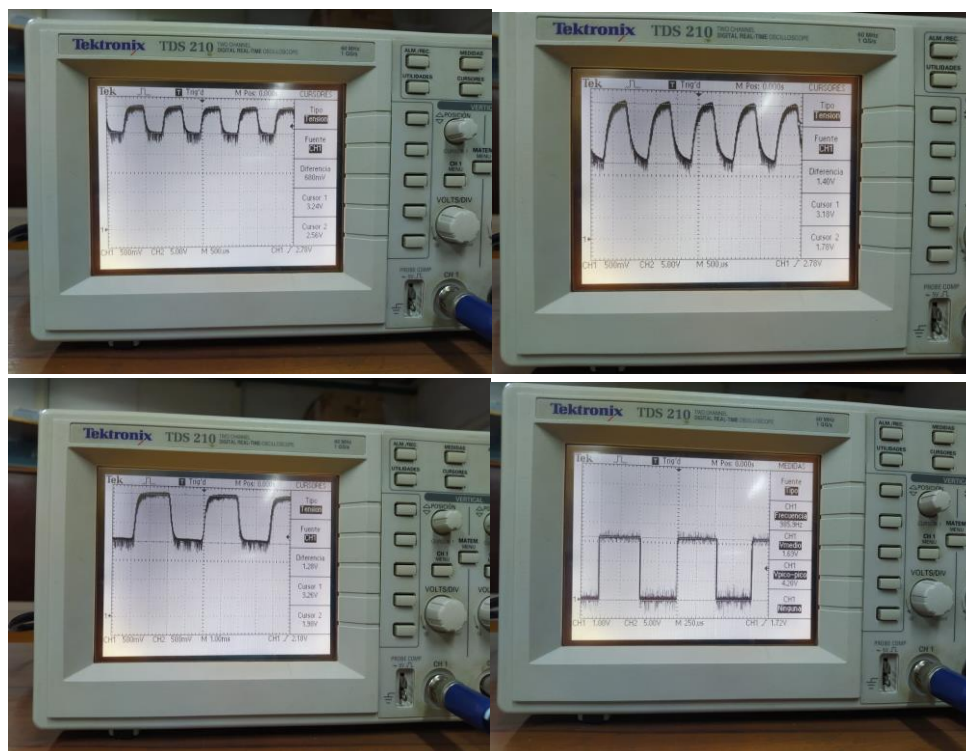


Figura 8.3 - Fotos de los resultados obtenidos en el osciloscopio al variar la ganancia del circuito

En la Figura 8.4 se tiene la placa terminada.

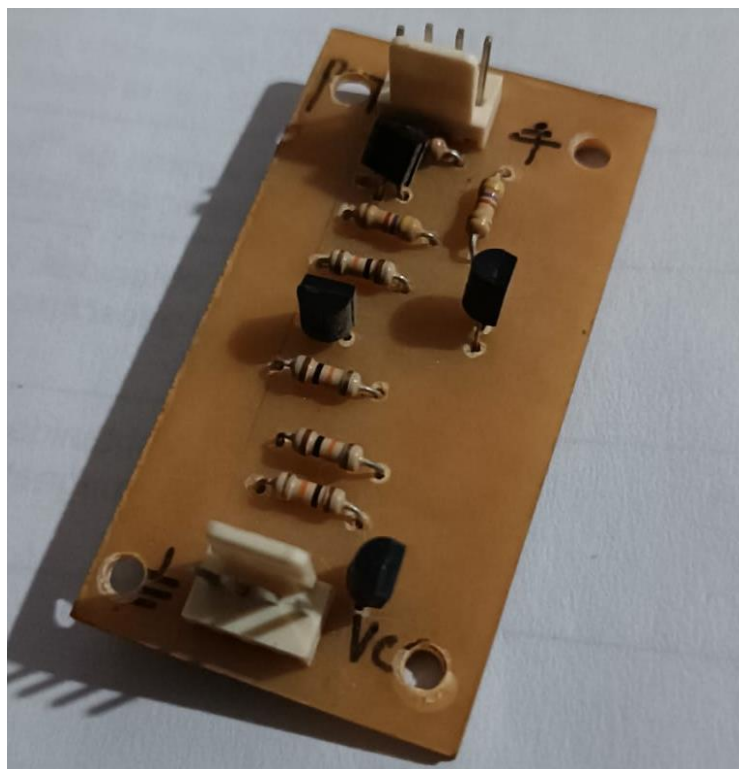


Figura 8.4 - Foto de la placa terminada

Circuito de Modulo de comunicación RS485

Se diseñó el circuito de comunicación de RS485 utilizando el IC SN75176. En la Figura 8.5 muestra el circuito del módulo para el maestro:

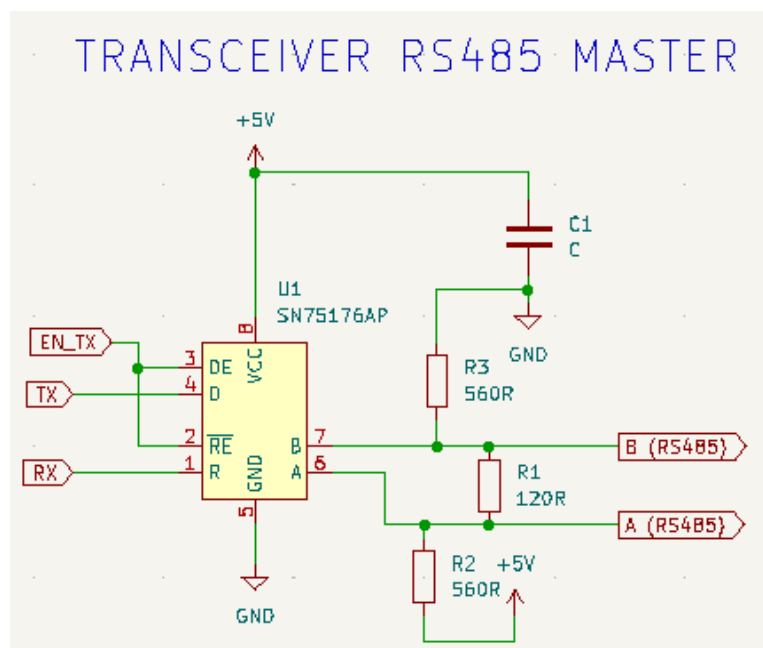


Figura 8.5 - Circuito eléctrico del módulo de comunicación RS485 para el maestro

El circuito es similar al circuito de aplicación que se encuentra en el datasheet del IC, con la diferencia que se agregó las resistencias R1, R2 y R3. La resistencia R1 es la carga para la señal diferencial de las terminales A y B. Esta señal diferencial, se transmite por un par trenzado con malla metálica conectada a tierra. La impedancia característica de la red de transmisión es aproximadamente de 120 Ohms. Las resistencias R2 y R3 son las llamadas resistencias de “Failsafe” cuya función es asegurar el correcto funcionamiento del sistema ante alguna falla como puede ser:

- Si todos los transceiver están funcionando como receptor.
- Si se produce una falla en los cables y se desconectan de la red.
- Si se produce un corto circuito.

La función de estas resistencias es la de forzar un estado lógico en la red, en este caso es un 1 lógico, ante una falla de estas características y evitar que los receptores del sistema capturen “datos espurios”. Los valores de dichas resistencias son tomadas en base al análisis del documento “The RS485 design guide” de Texas Instrument. En la Figura 8.6 se muestra el circuito de parte del esclavo.

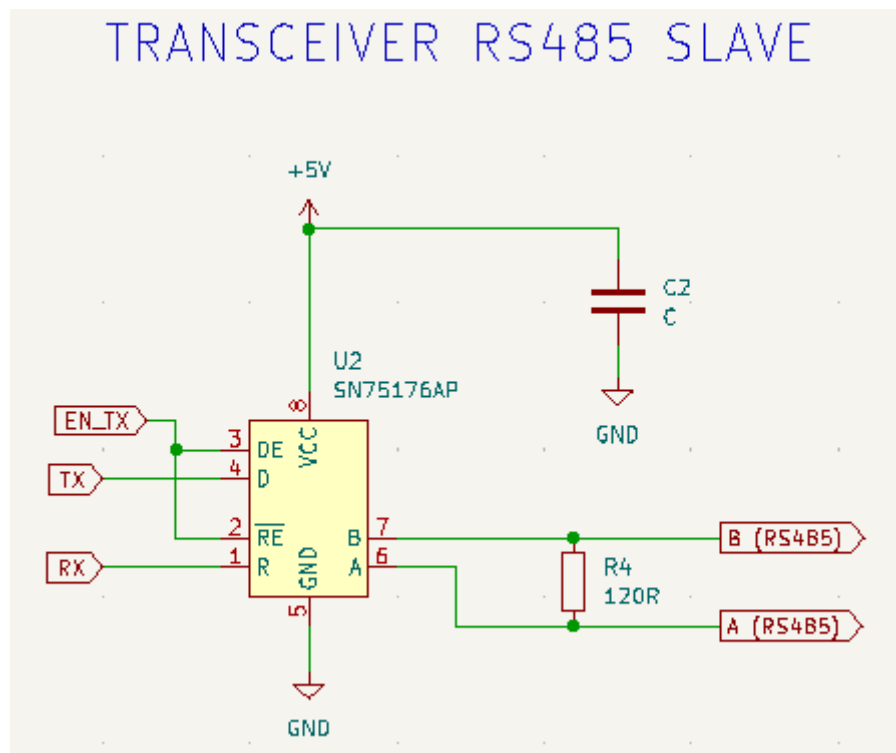


Figura 8.6 - Circuito eléctrico del módulo de comunicación RS485 para el esclavo

El circuito del esclavo, es similar al circuito del maestro, aunque la resistencia R4 es el terminador de la red (ver en la figura 8.7).

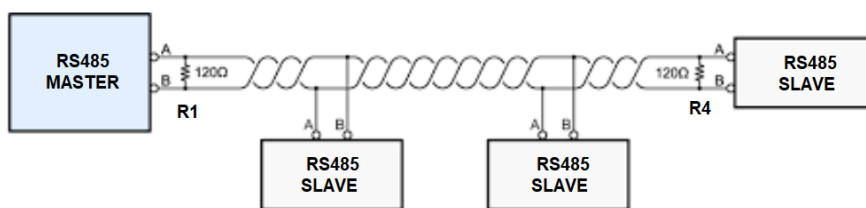


Figura 8.7 - Esquema general de la red 485.

La resistencia R4, solo debe estar presente en el último esclavo de la red.

9. PROGRAMACIÓN DEL FIRMWARE

Librerías

El desarrollo del firmware del dispositivo está conformado por varias librerías realizadas para manejar cada módulo o parte del robot utilizando el lenguaje C. Estas librerías están realizadas bajo un paradigma similar o a medio camino de la programación orientada a objeto, ya que este paradigma no es posible en el lenguaje C. Todas las librerías fueron creadas desde cero, a excepción de la librería destinada a manejar el módulo IMU que se partió de una librería destinada a otro microcontrolador, con el objetivo de lograr un código sencillo con las funcionalidades requeridas para hacer el proyecto y lograr una compatibilidad entre las librerías que están relacionadas entre sí como puede ser la librería de auto navegación con la de utilizar el escáner de códigos QR.

Todas las librerías realizadas para el firmware del robot poseen la siguiente estructura:

Struct

Todas las librerías poseen un Struct que es un conjunto de variables relacionados al módulo del robot que desean controlar como por ejemplo datos de recibido por comunicación, pwm del motor, etc.

Función init

Esta función se carga de inicializar las variables contenidas en el struct y hacer los pasos iniciales referidos al control de los módulos del robot como configurar los Timer.

Función handler

Esta función es la que se ejecuta al suceder un evento, interrupción o cada cierto intervalo de tiempo. Se encarga de atender a los respectivos módulos, actualizar sus variables, realizar una cierta tarea de control necesaria, etc. De esta forma, al necesitar utilizar alguna variable del struct en otra parte del código, solo será necesario acceder a ella directamente, sin preocuparse por el funcionamiento interno de la librería, ya que dichas variables son actualizadas periódicamente por la función handler.

Otras funciones

En esta categoría se engloban a las funciones referidas al cambio de configuración del dispositivo.

Para facilitar la tarea de programación, se utiliza la herramienta STM32CUBE para realizar la configuración de los módulos del microcontrolador. Esta herramienta posee una interfaz gráfica para realizar la configuración de cada uno de los módulos del microcontrolador para luego generar un código C que realiza dicha configuración.

También se hace uso de la Librería HAL (High abstraction layer) del microcontrolador para interactuar con las funcionalidades de sus módulos. Esta librería realiza una abstracción de hardware, que consiste en abstraer el trabajar bit a bit cuando se utiliza las funciones del microcontrolador, de esta manera haciendo un código más fácil de escribir y de leer. Si bien esta librería posee la desventaja de que no se logra un código muy eficiente, pero gracias a la potencia de los microcontroladores actuales, la eficiencia que se pierde no es relevante para la mayoría de aplicaciones, haciendo que se convierta en un estándar en la industria de desarrollo de firmware, reduciendo tiempo en el desarrollo y mantenimiento.

Algunas librerías como la de los encoder y de los motores poseen interacción de los pines que, según como se conecten físicamente al módulo pueden cambiar el sentido o la polaridad de la conexión, por ejemplo, que los pines horario y antihorario del STM32 se conecten al pin antihorario y horario del driver haciendo que el motor gire en sentido contrario al que se indica en el código. Para evitar confusiones con respecto al código y la conexión de los pines se agregó a la librería la opción de alternar estos pines, haciendo de esta manera que si hay un error en la conexión o por problema de la orientación del motor me conviene que el sentido positivo u horario sea girando en cierto sentido, se pueda modificar directamente por software, facilitando el concepto de armado modular del robot. El código, tanto de las librerías como el código principal de cada microcontrolador se puede encontrar en el anexo que acompaña a este documento, siendo el anexo I el que corresponde al código y librerías utilizadas en las placas Bluepill y en el anexo II el que corresponde al código de la placa Discovery board.

A continuación, se entrará más en detalle sobre la estructura de cada librería.

Librería para los Motores (motor.h)

Esta librería es la que se comunica con el driver de los motores. Esta librería debe generar una señal PWM en uno de los pines del microcontrolador además de indicar al puente H la polaridad de los terminales del motor mediante 2 pines del mismo, el pin horario y antihorario.

La señal de PWM se genera por medio del Timer del microcontrolador y para los pines horario y antihorario se utiliza el GPIO.

No es una librería complicada, solo enmascara la interacción con el Timer y el GPIO del microcontrolador para que del lado del programador sea más fácil entender que es lo que se está haciendo.

Librería para los Encoders (encoder.h)

Esta librería se encargará de interpretar las señales que provienen del encoder para determinar el sentido de rotación y la cantidad de giro que realizó la rueda. Para esto se utiliza el timer del STM32 que posee la funcionalidad de funcionar como un contador que dependiendo del estado de la señal aumenta o disminuye una cuenta.

El encoder que utilizamos está formado por un encoder óptico de cuadratura que consiste en un disco ranurado acoplado al eje del motor en medio de un diodo LED y 2 fotodiodos, como se muestra en la Figura 4.1. Este disco bloquea o deja pasar la luz dependiendo la posición del eje del motor, si existe una ranura en medio del led y el fotodiodo entonces dejará pasar la luz y si

no hay ranura entonces la bloqueara, generando a la salida una señal cuadrada en caso de que el eje esté girando. La posición de los dos fotodiodos está colocada de tal manera que las señales de salida de ambos se encuentran con un desfase de 90° , de esta manera se logra una mejor precisión con respecto a la posición del eje del motor y se puede determinar el sentido de giro del motor sabiendo que señal se encuentra adelante de otra, como se muestra en la Figura 4.2.

El encoder que se utilizó son los que vienen de fábrica en los motores Faulhaber, a excepción de uno debido a que se dañó, se tuvo que reemplazar con uno realizado con una impresora 3D, de 12 ranuras.

Las señales del encoder posee 4 estados posibles (00, 01, 11, 10), cada vez que la señal cambia de estado, el timer aumenta o disminuye una unidad dependiendo a qué estado cambie. Por cada ranura la señal pasa por 4 estados por lo que, en caso de que el contador aumente, esta suma 4 cuentas por cada ranura, por lo que por cada vuelta del motor, el timer realiza $12 \times 4 = 64$ cuentas. Teniendo en cuenta que el motor se encuentra acoplado a un motor reductor 1:64, una vuelta en la rueda equivale a $64 \times 64 = 3072$ cuentas del timer haciendo que cada cuenta del timer equivalga a un desplazamiento de 0.117 grados en la rueda

Librería del controlador PID (pid.h)

Si bien esta librería no maneja de manera directa ninguno de los módulos del robot, son bastante usados los controladores PID en varias de las librerías del firmware por lo que se realizó una librería aparte para este.

Es una librería sencilla que consta de Struct donde se encuentran las variables de importancia para el controlador como los coeficientes o el cálculo de cada término. Este controlador posee la función de saturar con la posibilidad de indicar cuales son los límites inferior y superior, y de aplicar una técnica de anti-windup llamado “clamping” que consiste en detener la integración del término integral en caso de que se detecte que la señal se satura.

En la Figura 9.1 se muestra el diagrama de flujo correspondiente al controlador PID implementado en esta librería. Se puede observar que la función de "clamping" está basada en una sentencia condicional que verifica si la suma de los tres términos (proporcional, derivativo e integral) calculados de manera convencional excede los límites de la señal de control. Si la suma supera dicho límite, el término integral se restablece a cero en ese momento, de lo contrario, no se realiza ningún cambio y el cálculo de la señal de control continúa normalmente.

Notar que esta es una de las pocas librerías que se encuentran tanto en las Bluepill y la Discovery board. Debido a que utiliza las librerías HAL, estas no son las mismas para ambas placas por lo que es necesario modificar el archivo .h para cada placa colocando “#include stm32f1xx_hal.h” para Bluepill y “#include stm32f4xx_hal.h” para Discovery.

Para facilitar la comunicación entre los dispositivos bajo esta modalidad se crearon dos librerías más, una llamada “conversion_array.h” que consiste en empaquetar y desempaquetar las variables float, int32_t, int16_t, etc. en los 4 bytes de argumento que posee este modo de comunicación y otra llamada “ordenes.h” donde se encuentra el valor de cada opcode y su función en el código.

Código principal de las placas Bluepill

Hasta aquí se describen las librerías que se encuentran implementadas para el funcionamiento de los módulos del robot, por lo que falta describir el programa principal del mismo.

Para empezar, se inicializan todas las variables y estructuras relacionados a los objetos o módulos a utilizar, colocando los coeficientes de los controladores en “0” para asegurarse que un inicio o reinicio de las Bluepill, sin antes haberse comunicado por la Discovery correctamente, no accionara los motores que maneja.

Durante el loop del código, las Bluepill utilizan el systick timer para medir la base de tiempo de las tareas que debe realizar. Este intercambio o ejecución de las tareas las realiza por polling y son 2 las tareas que debe atender, una es la del sistema de control de ambos motores controlando su velocidad o posición, obteniendo el valor de los encoder, etc. La otra tarea es la de controlar que exista una comunicación con la Discovery, en ciertas ocasiones cuando los motores están funcionando pueden introducir ruido en la línea de comunicación. El ruido hace que la trama se interrumpa, por lo que el objetivo de este control es detectar cuando se interrumpe una trama de comunicación, para vaciar los buffers de entrada y evitar que datos de la antigua trama no estén presentes cuando esté llegando una nueva trama.

A continuación, se describirán las librerías que se encuentran en la Discovery board

Librería para los módulos de ultrasonido (hc-sr04.h)

Esta librería es para utilizar los módulos HC-SR04 un medidor de distancia por ultrasonido.

Para iniciar una medición se debe enviar un pulso de 10us al pin Trigger del módulo, se emitirá una onda sonora que luego rebotará en un objeto y volverá al módulo. El tiempo que demora en realizarse este eco se podrá observar por el pin ECHO, donde el tiempo en que este pin está en alto será igual al tiempo de eco de la onda sonora. Para determinar la distancia del recorrido de la onda sonora habrá que multiplicar la velocidad del sonido por el tiempo de eco, esta distancia será igual al doble de la distancia del módulo al objeto. En la Figura 9.2 se muestra una foto del módulo, junto con un diagrama explicando su funcionamiento.

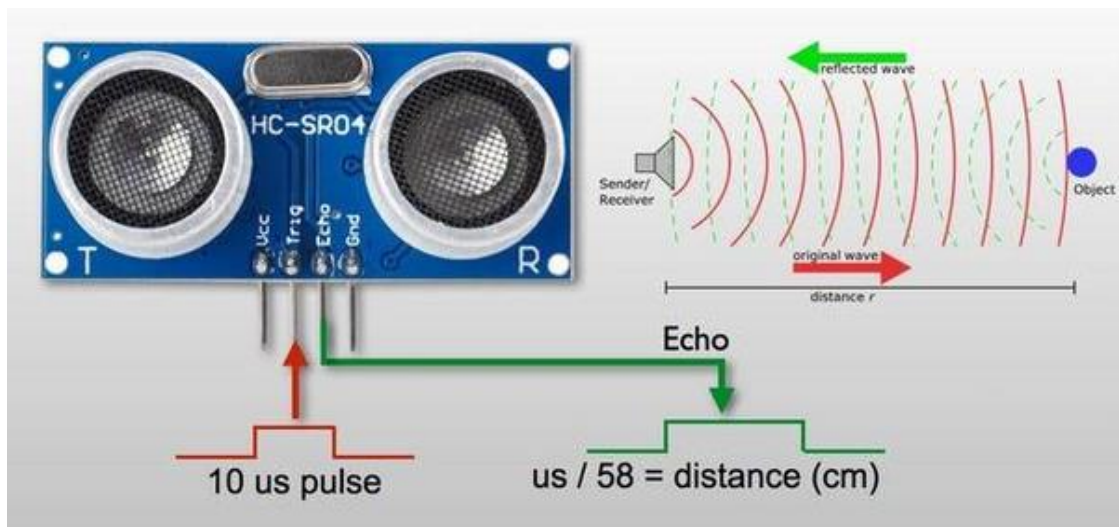


Figura 9.2 - Foto del módulo de ultrasonido HC-SR04 y un diagrama explicando su funcionamiento

El módulo estará constantemente midiendo ya que es utilizado para la localización del robot, por lo que para automatizar la generación de pedidos de medición se genera una señal PWM con un pulso o tiempo de “ON” de 10us en el pin de Trigger, mediante la prueba de medición a diferentes distancias se observó que el mayor tiempo que esta el pin Echo en alto es de ~70 ms por lo que se puso la señal de PWM a una frecuencia de 10Hz. Para la medición el tiempo de Echo se usó la función de input capture del timer de STM32 que consiste en hacer funcionar el contador en modo “free running” y cuándo se produce un flanco en el pin del canal (se puede configurar para capturar el flanco ascendente o descendente) realiza una captura del valor del timer y lo guarda en el registro asociado a dicho canal. En este caso se configura el timer para que cuando se realice una captura, se produzca una interrupción para cambiar la polaridad del flanco de captura y almacenar en memoria al valor del registro. Al momento de la captura del flanco descendente, se realiza la resta del registro y de la captura anterior para determinar el tiempo entre flancos y se determina la distancia que midió el módulo. Mediante pruebas se determinó que, a distancias mayores a 40 cm, el módulo ya no mide distancias de manera precisa por lo que se considera que la medición no es válida y no se puede utilizar.

Librería del módulo GM65

El módulo gm65 es un lector de códigos QR, de barras, etc. Posee dos puertos seriales de comunicación, por uart y por usb, junto con un botón para realizar la lectura de código. Para este caso se utilizará la comunicación por uart. El módulo posee distintas configuraciones acerca de los códigos a escanear, luz, tipos de códigos, formato de los datos de salida, etc. Con la configuración por default del módulo ya cubre con las necesidades que necesitamos para el proyecto por lo que no es necesario entrar en detalle acerca de este tema. Estas configuraciones se pueden realizar por medio de la transmisión serial o por escaneo de ciertos código QR que se encuentran dentro del manual del módulo por lo que se tendrá especial cuidado sobre este

último debido a que es posible que haga escanear estos códigos durante el funcionamiento del robot y produzcas fallas a propósito.

El módulo posee su propia forma de comunicación cuya trama varía según la acción a realizar además de utilizar CRC con su código ya incluido en el manual para verificar la integridad de los datos recibidos.

Trama de datos para leer registros

Max 256 bytes/time for zone bit reading.

Send Command Format:

Numbers of bytes	2bytes	1byte	1byte	2bytes	1byte	2bytes
Name	Head	Types	Lens	Address	Datas	CRC
Details	0x7E00	07H	01H	xxH	xxH	xxH

PS: Address : 0x0000~0x00FF (2 bytes) , address to start reading zone bit

Datas : 0x00~0xFF (1 byte) , Numbers of zone bit for Sequential read, 0x00= 256bytes

Receive Command Format:

Numbers of bytes	2bytes	1byte	1byte	1byte	2bytes
Name	Head	Types	Lens	Datas	CRC
Details	0x0200	00H	xxH	xxH	xxH

PS: Types: 0x00= read succeed

Lens : numbers of upload bytes; 0x00= 256bytes

Datas : 0x00~0xFF means read data.

If CRC wrong, no return data

If format or data wrong, no return data

Trama de datos para escribir registros

Send Command Format:

Numbers of bytes	2bytes	1byte	1byte	2bytes	1byte	2bytes
Name	Head	Types	Lens	Address	Datas	CRC
Detail	0x7E00	08H	01H	xxH	xxH	xxH

PS: Lens: 0x00~0xFF (1 byte) , means numbers of bytes of this datas, times of continuous writing.

0x00 means 256bytes

Address : 0x0000~0xFFFF (2 bytes) Start location of write

Datas : 0x00~0xFF (1~256 bytes) , dates wrote in zone bit

Receive Command Format:

Numbers of bytes	2bytes	1byte	1byte	1byte	2bytes
Name	Head	Types	Lens	Datas	CRC
Detail	0x0200	00H	01H	00H	0x3331

PS: Types: 0x00 write succeed

If CRC wrong, no return data

If format or data wrong, no return data

Trama de datos al escanear un código

Numbers of bytes	2 bytes	1 byte	1 byte	x bytes
Name	head	len	CODE ID	data
detail	0x0300	Y	Z	data

Para hacer esta librería, se tuvo que tener en cuenta ciertas consideraciones respecto al funcionamiento del módulo

- Una vez enviado el comando de escanear, el módulo escaneará por un cierto tiempo y luego se apagará automáticamente sin dar aviso de esto. Por lo que es necesario encontrar una forma de saber cuándo el módulo está escaneando, no encontró nada, o se desea que siga escaneando.
- Detectar si se escanea el código QR que habilita el cambio de configuración, y evitar que alguien interfiera en el funcionamiento del robot.

Para esto, se utiliza un timer para indicar cuando se acabó el tiempo de espera para cada etapa del proceso. Además, al terminar el escaneo de un código QR, se analiza que no sea el código de habilitar configuración por escaneo de códigos QR, en caso de que sea dicho código se inicia el proceso de enviar el código para deshabilitar la configuración por medio de códigos QR.

Para esto se desarrolló una máquina de estado para tener control de cada etapa del proceso de escaneo del módulo. En la Figura 9.2 se muestran la máquina de estado, junto con las transiciones entre cada estado del sistema.

teniendo como argumento el struct relacionado a la consola. Luego se crea un arreglo de este último donde se encuentran todos los comandos utilizados para el robot y sus respectivas funciones. Para facilitar la lectura del código se decidió colocar las funciones de los comandos en una librería llamada `<cli_handler.h>`.

Con la función de cargar carácter se lee uno a uno los caracteres que recibió el módulo uart, cuando se carga el carácter ‘!’ se habilita la consola y comienza el guardado de los caracteres hasta que se reciba el carácter ‘\$’. Cuando se detecta que se recibió el carácter salto de línea se interpreta que se ingresó un comando para que cuando se ejecute la función handler se realice la interpretación del comando y su ejecución.

Librería del IMU (imu.h)

Para el diseño de esta librería se partió de una librería de Arduino que se modificó su código fuente para que sea compatible con la librería HAL del STM32. Para el uso del IC MPU 9250, se debe realizar una serie de pasos previos antes de comenzar a tomar las medidas de los sensores, es por esto que se tomó como archivo de partida una librería de Arduino porque este tipo de pasos ya que encuentran realizados y contiene los “#define” referido a todas las direcciones de memoria del IMU. A continuación, se detalla las modificaciones que se realizaron al código de Arduino.

- Se cambiaron las funciones referidas al módulo SPI de Arduino, por las del STM32
- Se eliminaron las etapas de control del código que se encarga de verificar que se está comunicando con MPU 9250. Esto debido a que en el mercado se encuentran diferentes versiones del IC cuyo ID numérico no es igual al ID que se informa en el datasheet por lo que, para evitar problema en la línea de suministro para nuevos robot o reparaciones futuras, se eliminó esta etapa de control.
- Se realizaron arreglos de bug en el código referidos a la etapa del magnetómetro.

Un detalle a aclarar respecto al MPU 9250, es que dentro de la misma pastilla del IC se encuentra el MPU 9250 con el acelerómetro y el giroscopio dentro del mismo bloque, una interfaz de comunicación I2C auxiliar y un magnetómetro AK 8680. Esta interfaz auxiliar sirve para enmascarar varios sensores a través de un mismo dispositivo. En la Figura 9.3 se muestra de una manera gráfica cómo está constituido el circuito integrado del MPU 9250. Por lo que para el caso de obtener las medidas del magnetómetro se debe realizar unos pasos anteriores.

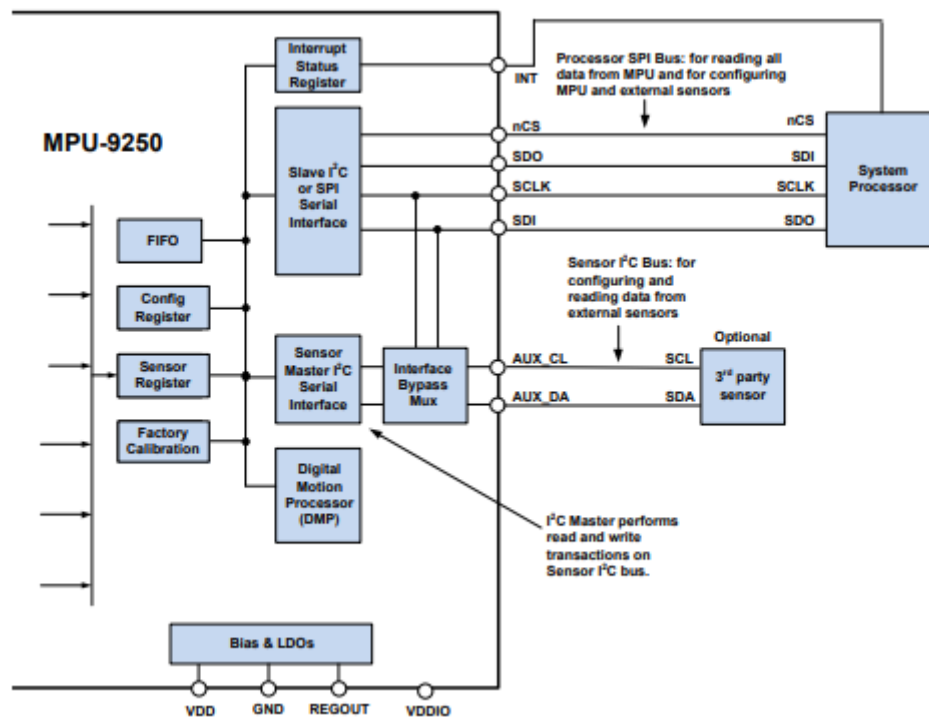


Figura 9.3 - Esquema donde se muestra como es la conexión interna IC MPU 9250

Una vez realizada la adaptación o “port” de la librería de arduino al STM32, empieza la programación relacionada a la etapa de captura de datos, filtrado y fusionado de sensores para obtener los datos relacionados a la orientación del robot.

En la etapa de “Init” se encarga de realizar la función de inicialización del MPU 9250, realiza una calibración del giroscopio y del magnetómetro y activa un timer para que realice una interrupción cada 5 ms.

La calibración del giroscopio consta de tomar varias medidas del giroscopio mientras el robot está quieto para luego realizar un promedio y obtener el offset del mismo. Para el magnetómetro existen dos tipos de distorsiones a corregir, Hard Iron y Soft Iron. Si se hace girar el magnetómetro sobre su eje z, y se capturan las medidas en el eje X e Y y se las grafica observando que estas deberían estar formando un círculo con centro en el origen debido a la rotación que existe del vector generado por el campo magnético de la tierra. Sin embargo se ve que se forma una elipse con su centro desplazado del origen, el desplazamiento del centro de la figura es lo que se llama “Hard Iron Distortion” y es debido a los campos eléctricos generados por las corrientes eléctricas de los dispositivos que se encuentran a su alrededor mientras que la deformación de la figura es lo que se llama “Soft Iron Distortion” y es debido a las piezas ferromagnéticas que se encuentran cerca del sensor que producen una distorsión de las lecturas del campo magnético ya que interfieren en la lectura del campo magnético de la tierra. El algoritmo de orientación necesita de saber cómo es la rotación del campo magnético terrestre por lo que la compensación del Hard Iron es crucial para el buen funcionamiento del sistema, por lo que se realizó el estudio de los valores de offset por el Hard Iron y la librería los utiliza para realizar la corrección de las medidas.

La distorsión por Soft Iron afecta la capacidad del dispositivo para determinar el norte magnético terrestre, ya que no solo altera las magnitudes medidas, sino que también puede provocar rotaciones en los ejes X e Y, lo que genera errores al indicar la ubicación del norte magnético. Sin embargo, al analizar la influencia de estos factores en el funcionamiento del robot, se concluye que no es crucial para el robot determinar la ubicación exacta del norte magnético, sino más bien poder registrar correctamente las rotaciones del robot en relación con algún elemento de referencia visible en su entorno.

El proceso de calibración del magnetómetro consiste únicamente en corregir la distorsión causada por Hard Iron mediante la captura de datos del magnetómetro mientras este gira sobre su propio eje. Posteriormente, se calcula la media de los datos obtenidos en cada eje del sensor para determinar el offset. A través de diversas pruebas de funcionamiento, se observó que este tipo de calibración es suficiente para obtener un rendimiento aceptable del sensor en la medición de la orientación del robot.

En base a los análisis anteriores, haciendo un análisis rápido durante el funcionamiento normal del robot, se llegaron a las siguientes conclusiones.

- El giroscopio no posee buen funcionamiento a bajas frecuencias debido al offset del mismo
- Las vibraciones en el robot pueden hacer que las lecturas del acelerómetro posean un ruido considerable en altas frecuencias
- Las corrientes de los motores pueden generar ruido en alta frecuencia en el magnetómetro debido a que estas corrientes están en función a la señal de control PWM.

Se observa que el giroscopio funciona bien en altas frecuencias, pero no en bajas, en cambio el acelerómetro y el magnetómetro funciona bien en bajas frecuencias, pero no en altas, por lo que se debe buscar una forma de unir las mejores cualidades de cada sensor y obtener resultados más robustos. Este proceso es lo que se llama fusión de sensores o “Sensor Fusión” que consiste en unir los datos de varios tipos de sensores con el objetivo de obtener mediciones más robustas y precisas.

Se analiza el caso de rotaciones en el eje Z, también llamado Yaw. Para este caso solo se usarán las medidas del giroscopio y del magnetómetro, debido a que las medidas de la gravedad del acelerómetro suelen tener una dirección similar a la del eje Z, y no es posible distinguir bien la rotación de ese vector por lo que se descarta. Para el acondicionamiento del sensor, se utilizará lo que se conoce como filtro complementario, que consiste en aplicarles a la medición del ángulo del magnetómetro un filtro pasa-bajo y a la del giroscopio un filtro pasa-alto y luego sumarlos. Estos filtros poseen la particularidad de que tienen la misma frecuencia de corte por lo que el resultado final tendrá las componentes en bajas frecuencias dada por el magnetómetro y las componentes en altas frecuencias dadas por el giroscopio.

A continuación, se detalla el proceso matemático para llegar a la fórmula de aplicación del dispositivo

$$FPB : \frac{1}{1 + \alpha * s} \quad FPA : \frac{\alpha s}{1 + \alpha s} \quad \alpha = \frac{1}{2 * \pi * f_c}$$

$$\theta_m = \arctan\left(\frac{m_y}{m_x}\right) \quad \theta_g = \frac{1}{s} * w_z$$

$$\theta_{sf} = \frac{1}{1 + \alpha * s} * \theta_m + \frac{\alpha s}{1 + \alpha s} * \frac{1}{s} * w_z$$

$$\theta_{sf} = \frac{1}{1 + \alpha * s} * \theta_m + \frac{\alpha}{1 + \alpha s} * w_z$$

$$\theta_{sf} * (1 + \alpha s) = \theta_m + \alpha * w_z$$

Pasando a tiempo discreto

$$\theta_{sf}(n) + \frac{\alpha}{T_s} (\theta_{sf}(n) - \theta_{sf}(n-1)) = \theta_m(n) + \alpha * w_z(n)$$

$$\frac{\alpha + T_s}{T_s} \theta_{sf}(n) = \frac{\alpha}{T_s} \theta_{sf}(n-1) + \theta_m(n) + \alpha * w_z(n)$$

$$\theta_{sf}(n) = \frac{\alpha}{T_s + \alpha} \theta_{sf}(n-1) + \frac{T_s}{T_s + \alpha} \theta_m(n) + \frac{T_s \alpha}{T_s + \alpha} * w_z(n)$$

$$\theta_{sf}(n) = \frac{\alpha}{T_s + \alpha} (\theta_{sf}(n-1) + T_s * w_z(n)) + \frac{T_s}{T_s + \alpha} \theta_m(n)$$

$$\text{Para } T_s = 0.01 \text{ } Fc = 1.4Hz = 0.11373$$

$$\theta_{sf}(n) = 0.92 (\theta_{sf}(n-1) + T_s * w_z(n)) + 0.8 \arctan \left(\frac{m_y}{m_x} \right)$$

Librería del controlador de posición y orientación del robot (controlador_robot.h)

Para el desarrollo de la librería del controlador de posición y orientación del robot se utiliza el modelo matemático obtenido del análisis general del robot. Este modelo era una ecuación matricial entre las velocidades de giro de cada una de las ruedas con la velocidad sobre cada uno de los grados de libertad del robot (X, Y y θ). Para mantener esta relación matricial dentro del código, se utilizó la librería CMSIS-DSP desarrollada por ARM para funcionar de manera eficiente en microcontroladores Cortex-M. Posee diferentes funciones para operaciones matriciales, vectoriales, filtros digitales. Se utilizó dicha librería para las operaciones matriciales más complejas mientras que para operaciones más simples se utilizó el método convencional. La estructura de control es simple, se trata de un controlador P utilizando la librería PID explicada anteriormente, que toma como valor de set la posición y orientación del robot, cuyo resultado se dirige a la planta del sistema

$$q = \begin{bmatrix} \theta \\ X \\ Y \end{bmatrix}$$

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = H * P \left(\begin{bmatrix} \theta_{set} \\ X_{set} \\ Y_{set} \end{bmatrix} - \begin{bmatrix} \theta_{out} \\ X_{out} \\ Y_{out} \end{bmatrix} \right)$$

$$\begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} = H * (P(q_{set} - q_{out}))$$

El reto en este sistema es encontrar los valores de θ , X e Y de salida del robot. Se vio que a partir de W de cada rueda es posible determinar la velocidad del robot, por lo que se podría integrar y obtener su posición, pero existe un problema con esto, ¿qué es? ¿Qué pasa si una de las ruedas empieza a deslizarse? En este caso los valores de velocidad serán erróneos y al integrarlas, se van a acumular un error a lo largo del tiempo, haciendo que se produzca un desfase entre lo que dice el controlador y la realidad. Es por esto que se debe utilizar una técnica de fusión de sensores entre lo que dice los encoder de las ruedas, los sensores de ultrasonido, el imu, etc. Para realizar el fusionado se utiliza el filtro de Kalman, que se trata de un algoritmo para identificar el estado oculto (estado no medible) de un sistema lineal al cual está sometido un ruido gaussiano o blanco aditivo. El estado del robot como θ , X o Y es un estado no medible ya que no se tiene un sensor que mida directamente este parámetro, se debe inferir por medio de otros sensores (ultrasonido, IMU, encoder). Según publicaciones científicas, los ruidos de estos sensores son cuasi-Gaussiano, por lo que se puede realizar una aproximación a un tipo de ruido Gaussiano para este caso.

Se tiene un sistema descrito de la siguiente manera:

$$X_k = A * X_{k-1} + B * U_k + Q \quad Z_k = C * X_k + R$$

donde:

X_k : es el estado del sistema en el instante k

X_{k-1} : es el estado del sistema en el instante $k-1$

A : es la matriz de transición del sistema

U_k : es la entrada de control del sistema en el instante k

B : es la matriz de control

Q : es el ruido del proceso, generalmente relacionado las imperfecciones de la fórmula del sistema k

Z_k : es la medición en el instante k

C : es la matriz de observación k

R : es el ruido de la medición

El filtro de Kalman consta de dos etapas, una de predicción que consiste en obtener el estado actual del sistema en base a un estado anterior y la de actualización o de corrección que consiste en corregir el estado actual que obtuvimos anteriormente por medio de un parámetro llamado

ganancia de Kalman (K) que el propio filtro escoge automáticamente en base a los errores del sistema.

$$X_{k|k} = X_{k|k-1} + K_k(Z_k - C * X_{k|k-1})$$

donde:

$X_{k|k-1}$: es el estado del sistema en el instante K, obtenido mediante el estado del sistema en el instante k-1

$X_{k|k}$: es el estado del sistema en el instante K, luego de realizar la corrección

K_k : es la ganancia de Kalman en el instante K

Otro parámetro a tener en cuenta para el filtro de Kalman, es la matriz de covarianza del error de estimación (P) que refleja la incertidumbre del sistema. Si bien este parámetro, al igual que K, el propio algoritmo lo irá determinando automáticamente, es necesario definir un valor P inicial al igual que un estado inicial del sistema.

A continuación, se detalla el paso a paso que debe realizar el algoritmo.

1. Primero se comienza la etapa de predicción

$$X_{k|k-1} = A * X_{k-1|k-1} + B * U_k$$

$$P_{k|k-1} = A * P_{k-1|k-1} * A^T + Q$$

2. Ahora comienza la etapa de actualización

$$K_k = P_{k|k-1} * C^T (C * P_{k|k-1} * C^T + R)^{-1}$$

$$X_{k|k} = X_{k|k-1} + K_k(Z_k - C * X_{k|k-1})$$

$$P_{k|k} = (I - K_k * C) P_{k|k-1}$$

Con esto en mente ya se puede empezar a aplicar el filtro en el controlador final del robot. Como cada estado del robot necesita de un tratamiento diferente debido a que cada uno hace uso de sensores diferentes, se eligió aplicar el filtro de manera separada en el código para obtener mayor claridad en su lectura.

A continuación, se muestra el análisis realizado para θ

$$\theta_{k|k-1} = \theta_{k-1|k-1} + T_s * \frac{R}{4(l+w)} * \begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix}$$

$$P_{k|k-1} = P_{k-1|k-1} + Q$$

$$K_k = \frac{P_{k|k-1}}{P_{k|k-1} + R}$$

$$\theta_{med} = \theta_{imu} \quad \text{Si US no esta activado}$$

$$\begin{aligned}\theta_{med} &= \frac{\theta_{imu} + \theta_{us}}{2} \quad \text{Si US esta activado} \\ \theta_{k|k} &= \theta_{k|k-1} + K_k(\theta_{med} - \theta_{k|k-1}) \\ P_{k|k} &= (1 - K_k) * P_{k|k-1}\end{aligned}$$

Ahora para el caso de Y, se implementó un filtro complementario para facilitar el cálculo matemático dentro del micro. Para este caso se tiene al sensor de ultrasonido que funciona bien a bajas frecuencias, pero a alta frecuencia no tiene buena respuesta ya que el sensor es lento para reaccionar a cambios rápidos en la distancia debido a su manera de funcionar pudiendo demorar 60ms en devolver una lectura, y se tienen las medidas del encoder, que por el deslizamiento de las ruedas, puede considerarse como un offset u ruido a frecuencias bajas en su lectura.

$$\begin{aligned}V_y &= \frac{R}{4} * \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} W_{dd} \\ W_{di} \\ W_{td} \\ W_{ti} \end{bmatrix} \\ D_{us} &= \frac{D_{udt} + D_{udd}}{2}\end{aligned}$$

$Y_{us} = D_{us} - D_{pared}$ El robot circulara al lado de una pared, por lo que debe estar a una distancia D_{pared}

$$Y_k = 0.92(Y_{k-1} + T_s V_y) + Y_{us}$$

10. ANÁLISIS DE MERCADO

Para realizar el análisis de costos del robot se tomará como medida la moneda dólar debido a la variabilidad del valor del peso argentino. A continuación, se detallan los gastos en materiales en el armado del robot.

Cotización	Dólar Blue
Fuente:	https://dolarhoy.com/
Precio de venta	\$1,300

COMPONENTE	UNIDAD	CANTIDAD	PRECIO POR UNIDAD (USD)	PRECIO POR UNIDAD (ARS)	PRECIO TOTAL (ARS)
DISCOVERY BOARD STM32F407G-DISC1	UN.	1	\$21.16	\$23,910.80	\$23,910.80
Bluepill	UN.	1	\$12.86	\$14,531.80	\$14,531.80
MONSTER MOTOR SHIELD	UN.	2	\$48.84	\$55,189.20	\$110,378.40
HC-SR04	UN.	2	\$2.32	\$2,621.60	\$5,243.20
GM65	UN.	1	\$45.99	\$51,968.70	\$51,968.70
MPU 9250	UN.	1	\$33.60	\$37,968.00	\$37,968.00
PLACAS DE COBRE PERTINAX 10X10	UN.	5	\$1.88	\$2,124.40	\$10,622.00
MÓDULO ENCODER ARDUINO	UN.	2	\$3.57	\$4,034.10	\$8,068.20
TRANSISTORES TBJ	UN.	16	\$0.25	\$282.50	\$4,520.00
RESISTENCIAS VARIAS	UN.	46	\$0.14	\$158.20	\$7,277.20
CAPACITORES VARIOS	UN.	3	\$0.16	\$180.80	\$542.40
CABLE PLANO MULTICOLOR DUPONT	METROS	1	\$11.85	\$13,390.50	\$13,390.50
CABLE DE AUDIO STEREO 1MM	METROS	1	\$0.65	\$734.50	\$734.50

CABLE MALLADO BIPOLAR	METROS	1	\$2.08	\$2,350.40	\$2,350.40
CONECTORES PALAS	UN.	4	\$0.12	\$135.60	\$542.40
PUNTERA TUBULAR 1 MM	UN.	17	\$0.02	\$22.60	\$384.20
REGULADOR STEP-DOWN	UN.	1	\$3.41	\$3,853.30	\$3,853.30
REGULADOR STEP-UP	UN.	1	\$4.92	\$5,559.60	\$5,559.60
CONECTORES MOLEX 4 TERMINALES	UN.	19	\$0.26	\$293.80	\$5,582.20
CONECTORES MOLEX 5 TERMINALES	UN.	7	\$0.33	\$372.90	\$2,610.30
CONECTORES MOLEX 2 TERMINALES	UN.	10	\$0.28	\$316.40	\$3,164.00
RASPBERRY PI PICO W	UN.	1	\$11.16	\$12,610.80	\$12,610.80
ZÓCALOS DIP 8 PINES	UN.	3	\$0.42	\$474.60	\$1,423.80
TIRA DE PINES HEMBRAS	UN.	9	\$0.94	\$1,062.20	\$9,559.80
MADERA MDF 9MM 30X40	UN.	5	\$2.81	\$3,175.30	\$15,876.50
RUEDAS MECANUM	UN.	4	\$8.91	\$10,068.30	\$40,273.20
TORNILLO PARA MADERA	UN.	18	\$0.02	\$22.60	\$406.80

Para determinar de gastos en mano de obra el mismo monto que gastos materiales.
y tomando ganancia como un 50% del costo de producción

Precio	En Materiales
En ARS	\$452,530.00
En USD	\$348.10
	En mano de obra

En ARS	\$452,530.00
En USD	\$348.10
	Ganancia
% de ganancia	50%
En ARS	\$226,265.00
En USD	\$174.05
	Total
En ARS	\$678,795.00
En USD	\$522.15

Con el precio de venta estimado analizaremos las competencias o alternativas que existen en el mercado actualmente.

El robot posee dos características que se deben destacar a la hora de compararlos con robot de otras compañías, una es que el modo de auto navegación funciona como una rutina de aplicación destinada para la aplicación final del robot, es decir, es una solución ya prefabricada. La otra característica es el Modo consola, que permite al programador utilizar al robot como base para su punto de partida para el desarrollo de un proyecto para una aplicación específica. Es decir, permite al usuario fabricar sus propias rutinas de aplicación. Con esto en mente se armará una tabla de comparación con robots de similares características

Además, es importante considerar que muchos de estos robots no se comercializan de forma individual, sino como parte de un servicio integral que incluye la instalación y automatización de la fábrica, lo que dificulta estimar el precio de cada robot de manera aislada.

A continuación, se muestra una tabla de robot para el transporte de carga de ~50kg aproximadamente.

MODELO	AIUMU NAVIRA	XBOT	CIOT	SHARKO 10	AMB-JS
Fabricante	INDAMA	Guangzhou Daxaing	Kedo Technology	MOVIGO	SEER ROBOTICS
País de origen	ARGENTINA	CHINA	CHINA	PAÍSES BAJOS	CHINA
Precio estimado	-	650-750 USD	1900-2300 USD	-	1500-1800 USD
Sistema de guía	SLAM	SEG. DE LÍNEA	SLAM	SLAM + QR	SLAM
Tipo de	DIFERENCIAL	DIFERENCIAL	DIFERENCIAL	DIFERENCIAL	DIFERENCIAL

tracción					
Programable	-	-	SI		
Funciones extras	-	MÓDULO DE CARGA INALÁMBRICA; APP DE CONTROL; WIRELESS RS485	MÓDULO DE CARGA INALÁMBRICA; APP Y WEB CONTROL	ESCÁNER DE CÓDIGOS DE BARRAS Y QR, DETECCIÓN DE OBJETOS; TRANSPORTE DE PALET;	ESTACIÓN DE CARGA; NAVEGACIÓN AUTÓNOMA

Con esta comparación se observa que el precio estimado se encuentra entre valores de precios de robot no tan avanzados como un seguidor de línea, y más avanzados como los robots con SLAM por medios de módulos LIDAR. Es importante señalar que la capacidad de carga de estos robots es superior a la del robot desarrollado en este proyecto. Por lo tanto, si se considera su comercialización, será necesario realizar un análisis estructural para diseñar un chasis más robusto y evaluar si se requiere actualizar componentes como los motores, las baterías o la etapa de potencia, lo que podría incrementar el costo del robot.

Argentina es un país que busca la industrialización, siendo la meta a alcanzar de varios dirigentes políticos, por lo que el desarrollo de productos y servicios destinados al negocio B2B (Business to Business) es una oportunidad de negocios.

La robótica junto con la inteligencia artificial es una de las áreas que más avances presentaron estos últimos años, permitiendo el desarrollo de productos más económicos y funcionales para la población, de lo que argentina no debe quedarse atrás si busca potenciar su industria. En argentina existen grandes empresas o startup en el área de inteligencia artificial como es “Globant” pero no existen empresas en robóticas que estén en el mismo nivel que las anteriores siendo esta última un aporte importante para el desarrollo del país.

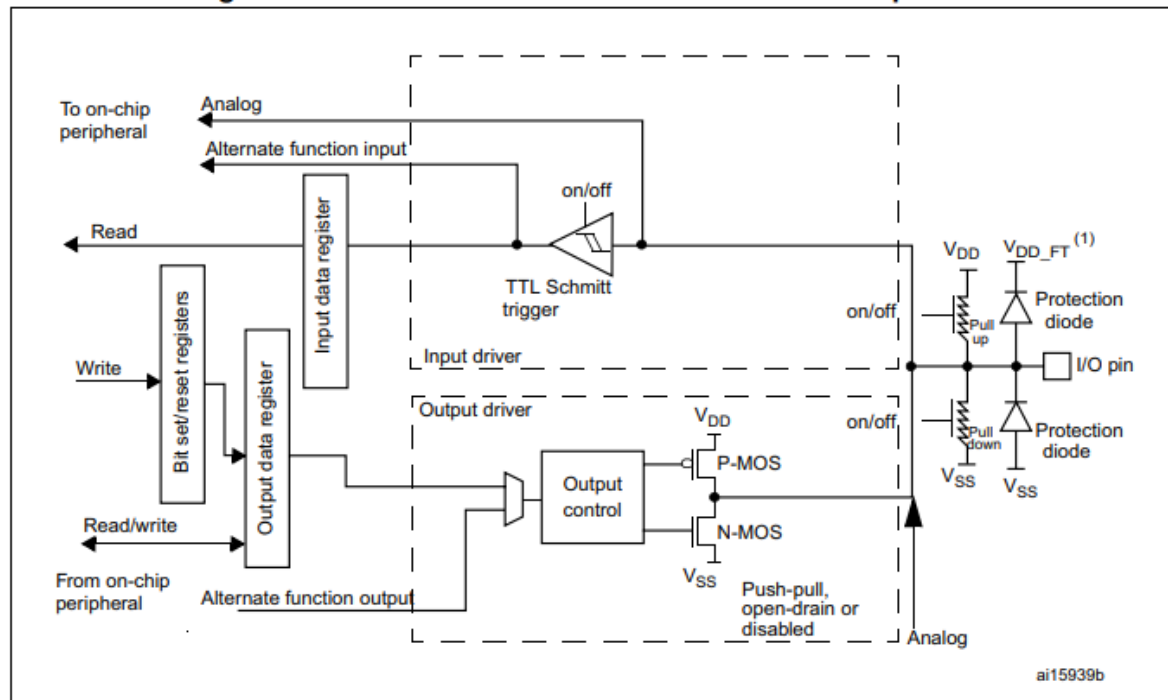
11. INCONVENIENTES, DESAFÍOS Y SOLUCIONES

A lo largo del proyecto del robot nos encontramos con varios inconvenientes y desafíos, a continuación, iremos detallando los diversos problemas que encontramos con el desarrollo del robot.

Variabilidad de los drivers de motores

El driver utilizado para los motores es el SparkFun Monster Motor Shield utiliza el Circuito Integrado VN12SP30 que es controlador de motor por puente H con protección contra sobrevoltage, subvoltage, cortocircuito, contra sobrecalentamiento y sensor de corriente. Sin embargo, no posee una buena compatibilidad con lógica de 3V3 que es el nivel de voltaje que maneja los STM32F103T8C6, siendo que la tensión de entrada mínima para que el driver detectase un “1” lógico es de 3.25V siendo en teoría compatible con la tensión salida del microcontrolador, pero el problema es que muchas placas SparkFun Monster Motor Shield tuvieron un problema en la línea de suministro de su proceso de fabricación, que recibieron lotes de VN12SP30 con alguna deficiencia haciendo que estos no funcionasen correctamente. La deficiencia más común es referida a la impedancia de entrada de los pines lógicos del integrado que al ser de un valor bajo, de aproximadamente 500Ω , hacia un efecto de carga en los pines del microcontrolador haciendo que estos permanecieran en “0” lógico. Durante el ensamblaje del robot, una de las placas presentó este problema, por lo que se decidió utilizar la función *open-drain* en los pines del microcontrolador, junto con una resistencia de pull-up externa.

La función open-drain consiste en dejar el transistor PMOS en corte o alta impedancia, haciendo que el módulo GPIO solo pueda accionar el transistor NMOS, haciendo que este funcione como una llave donde el “0” lógico representa la llave abierta y el “1” lógico representa la llave cerrada.

Figure 25. Basic structure of a five-volt tolerant I/O port bit

1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD} .

Figura 11.1 - Estructura básica de los pines de entrada/salida con 5V tolerant, obtenida del datasheet del microcontrolador STM32F103T8C6.

Algunos pines del microcontrolador STM32F103T8C6 son “Five-Volt Tolerant”, es decir, que a pesar de que el microcontrolador posee lógica de 3V3, estos pines soportan tensiones de 5V. Entonces, con la función open-drain activada, se puede colocar una resistencia de pull-up conectada a los 5V teniendo una salida con lógica de 5V, como se muestra en la Figura 11.1. Aunque, debido al problema de la impedancia de entrada, el valor de esta resistencia debe variar entre en placa y placa ya que si se coloca una resistencia de pull-up pequeña, funcionara bien en la placa con el integrado defectuoso, pero en una placa con el integrado funcionando correctamente, la resistencia pequeña haría que circulase mucha corriente por el transistor NMOS del microcontrolador, pudiendo hacer que este se queme. Por este motivo se eligió que solo una de las placas tuviese esta resistencia de pull-up pequeña.

Documentación GM65

El módulo de scanner QR GM65 posee una documentación incompleta, en lo referido a trama de datos que envía al leer un código QR ya que no se detalla este aspecto en el manual de usuario del mismo, a pesar de que se menciona los códigos de configuración que modifican el formato de esta trama, no se especifica cual es la trama de datos por default que posee el módulo. El inconveniente se resolvió de forma sencilla utilizando un analizador lógico, que permitió examinar la trama y corregir el problema.

Otro detalle que no se habla en la documentación es lo referido al mapa de registros del módulo, ya que existen funciones que son esenciales para evitar que el módulo sea configurado por parte de terceros ajenos a los usuarios del robot, la de dar inicio al escaneo y se restaurar la configuración por default. La de comenzar con el escaneo se especifica en el manual cuando detalla el funcionamiento del “command mode” que escribiendo un 1 en el registro 0x0002 da comienzo con el proceso de escaneo. Sin embargo, en el mapa de registro se menciona que todo el registro 0x0002 es de uso reservado y no se menciona esta función. Para encontrar el registro y el bit donde se realiza la restauración de la configuración por default fue más complicado, para eso se usó el analizador lógico para visualizar cómo era la trama que devuelve el módulo cuando se escanea varios los códigos QR de configuración, además se hizo un programa para el microcontrolador para que, por medio de un conversor USB-TTL, pudiera ver qué valores poseen los registros del módulo y además escribir sus valores desde la PC. Con esto se pudo determinar que al escribir en un bit del registro 0x0003 que aparece como reservado en el manual de usuario se tuvo acceso a esta función.

También se observó que, al escanear un código de configuración, el módulo devuelve su contenido por UART por lo que con esto se pudo diseñar una librería que detectase cuando se escanea el código que habilita la configuración por código QR para que inmediatamente el microcontrolador enviará la señal de restablecer la configuración por default y deshabilitar la configuración por códigos QR.

Encoder para uno de los motores

Uno de los motores vino con el encoder de fábrica quemado debido al mal uso, por lo que se tuvo que diseñar uno nuevo. Para esto se utilizaron 2 módulos de encoders que son para Arduino que tenían el optoacoplador, además de la etapa que pasa la señal analógica a digital, pero esta etapa posee el inconveniente que no está diseñada para motores tan rápidos, cuya señal analógica es de tan alta frecuencia que no entregaba correctamente la señal digital. Para solucionar esto, se tuvo que desoldar unos de los capacitores de la placa y utilizar la etapa analógica-digital que se usaron para los demás encoder.

Imitaciones de STM32F103T8C6

Un problema dentro del entorno de STM32F103T8C6 es que en el mercado se encuentran varias imitaciones de este microcontrolador que, en teoría funcionan exactamente igual en varios aspectos, al querer utilizar las herramientas de software oficiales de ST, estos detectas estas copias y rechazan la conexión para tareas como el debugger del programa, por lo que se volvió un inconveniente el escribir y grabar programas en estas placas. Por suerte, este microcontrolador posee un bootloader grabado de fábrica, los bootloader son programas que permiten grabar datos en la memoria FLASH del microcontrolador, que es el área de memoria donde se guardan los programas, además existe el programa ST Programmer que permite enviar un programa compilado al bootloader por medio del puerto serial. De esta manera se puede cargar el programa para que funcione en el microcontrolador, sin necesidad de utilizar las herramientas habituales.

Modelos matemáticos del robot

Cuando se desarrolló el modelo completo del robot, se supuso que las ruedas no resbalaban o deslizan al rodar, por lo que el modelo solo funciona si se respeta esta condición, por lo que se debe cumplir la condición

$$W_{dd} + W_{di} = W_{td} + W_{ti}$$

Donde

W_{dd} = Velocidad de giro de la rueda delantera derecha

W_{di} = Velocidad de giro de la rueda delantera izquierda

W_{td} = Velocidad de giro de la rueda trasera derecha

W_{ti} = Velocidad de giro de la rueda trasera izquierda

El problema de esto es que hay que asegurarse de que los 4 motores deben tener la misma respuesta del tiempo, un objetivo imposible de alcanzar ya que eso significaba que los 4 motores son exactamente idénticos, algo muy difícil de lograr. Además, mediante los ensayos que se les realizó a los motores se observó que el tiempo de asentamiento cambia según el sentido de giro haciendo que exista más variabilidad en maniobras donde las ruedas giran en sentido contrario. Este problema se manifiesta de manera evidente cuando el robot pasa de estar en reposo a moverse, y se acentúa durante maniobras como el desplazamiento lateral, donde los motores operan en sentidos opuestos, lo que genera una mayor variabilidad en los tiempos de respuesta. Esto provocaba que las mediciones de velocidad y posición basadas en los datos de los encoders fueran poco precisas, especialmente durante maniobras de giro y desplazamiento lateral. Para resolver este problema, se utilizó el IMU para todas las tareas relacionadas con el giro y la orientación del robot, mientras que los sensores de ultrasonido se emplearon para los desplazamientos laterales. Sin embargo, los movimientos hacia adelante y atrás se dejaron sin un sensor que los verificara, ya que estas maniobras generan desplazamientos mínimos, lo que hace que puedan tolerarse ciertas imprecisiones. Una mejor opción sería utilizar un sensor LIDAR para corroborar tanto desplazamientos laterales como los de ir hacia adelante o atrás, pero por una cuestión de fondos no fue posible acceder a este sensor.

Por este motivo, la rutina de auto-navegación incluye las siguientes consideraciones:

- El robot sólo puede moverse hacia atrás y adelante, no puede moverse lateralmente.
- Los giros que debe realizar el robot siempre serán aquellos que dejen al robot con sus sensores de ultrasonido apuntando a la pared cuando sea posible ya que de esta manera puede controlar si se está desviando del camino y corregirlo si es necesario.
- El robot enciende el escáner QR medio metro antes de llegar a la estación y hasta 1 metro después de la estación, de esta manera se asegura que capture el código QR de la estación por si existe un error en la distancia que estuvo midiendo.
- Al llegar a cada estación, el robot reinicia la medición de la distancia en el momento que escanea el código QR de la estación, de esta manera evita la acumulación de error ya que los movimientos de adelante y atrás son los únicos que no tenemos manera de corroborarse con un sensor extra.

Con estas últimas dos consideraciones se puede ver que se utiliza el Scanner QR como sensor para corregir distancias en el eje X del robot que es el eje donde se mueve el robot cuando hace los movimientos hacia adelante y atrás.

Módulo Bluetooth

Como se mencionó anteriormente, inicialmente se consideró utilizar el módulo HC-05 Bluetooth - Uart, sin embargo al observar que el robot tenía un tiempo de respuesta considerablemente más alto que si usábamos un conversor USB-TTL convencional para enviar los comandos, utilizamos un analizador lógico para analizar la comunicación uart y observamos que existían un tiempo de demora de aproximadamente de 1 segundo entre que se enviaba el dato por Bluetooth y se enviaba el dato por uart además de existir pérdida de datos si se envían varios datos a la vez por lo que se debió buscar una alternativa de comunicación inalámbrica.

Para esto, se decidió intentar utilizar la placa Raspberry pi pico W que posee comunicación Bluetooth y Wifi, un microcontrolador RP 2040 con doble núcleo ARM Cortex M0 que podría servir para tener una comunicación más rápido y confiable al tener un microcontrolador con cierta potencia. Para realizar la prueba se decidió realizar el programa con Micro Python, que es un port o adaptación de Python para microcontroladores, ya que este lenguaje fue pensado para realizar programas de una forma más rápidos ya que de manera nativa enmascara las partes de bajo nivel de manera que todos los módulos en Python se comportan como si fueran objetos en este entorno. Además, en microcontroladores de doble núcleo, Micro Python hace que en un núcleo se encargue de manejo a bajo nivel de los módulos del microcontrolador, como este caso sería la comunicación bluetooth, y en el otro núcleo ejecuta el código Python, logrando obtener buenos resultados en muchas situaciones. MicroPython utiliza la versión de Bluetooth de baja energía (Bluetooth Low Energy), esta es diferente de la versión 2.0 del Bluetooth convencional que emplea el módulo HC-05. Por esta razón, algunas herramientas de software previamente utilizadas para comunicarse con el HC-05, como la función Bluetooth de Windows 10 o 'Bluetooth Serial Tool', no eran compatibles. Esto obligó a recurrir a otras herramientas, como 'Bleak' o 'Bluetooth Serial Terminal' para Android, para lograr la comunicación.

Este código es muy sencillo, consiste en configurar el Bluetooth para que funcione como un dispositivo serial (Serial Port Profile) y hacer que todos los datos obtenidos por Bluetooth se envíen por comunicación uart por medio de los pines del microcontrolador y todos los datos recibidos por uart se envíen por bluetooth.

Una vez realizado el código y cargado en el microcontrolador se pudo comprobar que el tiempo de demora entre Bluetooth y uart se redujo increíblemente además de reducir considerablemente la pérdida de datos durante la comunicación, pudiendo mandar grandes cantidades de datos sin ocasionar pérdidas.

Con estos resultados, se decidió por cambiar el módulo HC-05 y utilizar la Raspberry pi pico W

12. FUTURAS MEJORAS

La “Navegación Autónoma” es de gran interés debido a los avances en robótica que el área automotriz, donde con los últimos avances de inteligencia artificial, se lograron desarrollar vehículos capaces de conducirse sin intervención humana. La navegación autónoma se refiere a la capacidad de un sistema, ya sea un vehículo, dron o robot, para desplazarse y tomar decisiones de manera independiente sin intervención humana directa. Este enfoque revoluciona diversos campos, desde el transporte hasta la exploración espacial y la logística. Utilizando sensores avanzados, sistemas de percepción, algoritmos de procesamiento de datos y técnicas de aprendizaje automático, la navegación autónoma permite a los dispositivos analizar su entorno, tomar decisiones en tiempo real y adaptarse a situaciones cambiantes. Este avance tecnológico no solo promete mejorar la eficiencia y seguridad, sino que también abre nuevas posibilidades en sectores que van desde la conducción autónoma hasta la automatización de procesos industriales y la exploración de entornos inhóspitos.

El robot que desarrollamos comprende una introducción a este tipo de área ya que, si bien no es capaz de detectar los objetos que hay a su alrededor, es capaz de desarrollar un camino, orientarse y guiarse automáticamente. Además, con el modo consola es posible obtener datos internos del robot y fusionarlo con datos externos en una computadora o dispositivos externos siendo posible en un futuro acoplarlo con sensores más novedosos incluso puedes desarrollar técnicas de navegación autónoma más complejas.

A continuación, se detalla cuáles son las mejoras que se puede realizar al robot.

- ***Agregar una computadora a bordo del robot:***

Técnicas de Navegación autónoma más novedosas hacen uso de la Inteligencias artificial o sensores como una Cámara de profundidad que pueden ser muy complejos para un microcontrolador, por lo que, colocar una computadora a bordo como una Raspberry Pi se convertiría en un fin necesario para lograr un sistema de navegación más autónomas, robusta e inteligente.

- ***Agregar sensores de Lidar:***

Los sensores Lidar se convirtieron en un estándar en muchas áreas de la robótica debido a que puede generar un mapa 2D, en algunas modelos incluso un mapa 3D, permitiendo utilizar algoritmo de SLAM (Simultaneous localization and mapping) lo que permiten generar mapas del entorno del robot y realizar el pathfinding o búsqueda de caminos, evadir obstáculos, etc.

- ***Mejorar la autonomía energética:***

Una de las cosas a recalcar sobre el robot es sobre su autonomía energética, es decir, la duración de la materia cuando el robot está moviéndose. Una mejora que se puede realizar al robot sin agregar elementos extras sería diseñar el código de los microcontroladores del robot pensados que generen un menor consumo o diseñar controladores para los motores pensando en la eficiencia energética, sin embargo, teniendo en cuenta que cada motor del robot consume $\sim 1A$,

en comparación de los 200mA que consume toda la electrónica del robot, se concluye que el consumo de $\sim 4A$ de los 4 motores es lo que más impacta en el rendimiento energético. Por lo que una solución más conveniente es utilizar baterías más mayor capacidad de carga como las de ion de litio y diseñarle al robot una estación de carga inalámbrica, es decir, diseñar un módulo de carga inalámbrica al robot y hacer una estación donde el robot pueda dirigirse ahí cuando tenga poca batería y cargarla de manera totalmente automática. De esta forma, sí que quisiera utilizar el robot como sistema de transporte de carga, se harán 2 grupos de esta manera un grupo estaría transportando cargas mientras que otro estaría en la estación de carga y cuando al primer grupo se quede sin batería harían el intercambio.

- ***Accesorios para el robot:***

Otro accesorio útil para el robot es adicionar accesorios al lomo del robot para mejorar sus capacidades como puede ser agregar una balanza para medir el peso que lleva encima y dar mensajes de advertencias cuando se sobrecarga al robot con mucho peso. Otro aditamento que se puede hacer es agregar un brazo robótico como el que tiene el robot KUKA YOUNBOT, aunque perdería la capacidad de transportar cargas, este brazo robótico le permitirá realizar diferentes tareas.

- ***Mejoras en el soporte de peso:***

Una mejora sustancial para el robot es mejorar la estructura del chasis mediante un análisis mecánico del mismo, para lograr aumentar el soporte de peso del mismo y aumentar el mercado disponible para el robot, llegando al punto de diseñar varias versiones según el soporte de peso que debe llevar. Aumentar el soporte de peso del robot es un aspecto muy importante dentro del punto de vista comercial ya que este tipo de robot son muy utilizados en industrias que tienen que llevar piezas pesadas desde dos puntos de la fábrica como puede ser la industria automotriz.

- ***Cambio a motores brushless***

En los motores Brushed o de escobillas, como los que usamos para este robot, posee unas escobillas metálicas que, según la posición del eje del motor, irá haciendo contacto con las distintas bobinas del motor haciendo que la corriente eléctrica del bobinado varía, es decir, realiza el “switching” de la corriente por medio de un mecanismo enteramente mecánico. En cambio, los motores brushless realizan este switching de manera electrónica, permitiendo mejores aspectos a los motores brushed como menos desgaste mecánico debido a la escobilla, menor ruido y mejor rendimiento, ya se vio en el análisis del mercado que muchos de los motores utilizados en los robots son motores brushless debido a su mejor rendimiento energético y menor desgaste mecánico.

- ***Añadir funciones de ciberseguridad al robot***

Es necesario contar con metodologías de ciberseguridad en la industria, automatismo y robótica para proteger sistemas críticos, prevenir ataques y asegurar la integridad y confiabilidad de las operaciones. Por este motivo, es necesario para futuras mejoras para el robot agregar funciones de criptográfica y sistemas de seguridad en la comunicación usuario-robot para evitar que terceros envíen comandos no deseados que produzcan daños en el entorno de trabajo o puedan obtener información sensible acerca del establecimiento.

- ***Realizar un programa con interfaz gráfica para la configuración del robot***

Para simplificar la configuración, mantenimiento y visualización del robot por parte de los operarios, es necesario desarrollar un programa con interfaz gráfica que facilite esta tarea. La comunicación actual, a través de línea de comandos, es poco interactiva y puede desmotivar a las empresas a incorporar el robot en su línea de producción, ya que podría considerarse una herramienta difícil de operar y encontrar operarios capacitados para su uso.

- ***Cargado de mapa para la rutina de auto navegación***

Incorporar la funcionalidad de cargar mapas de manera más sencilla al robot es esencial para agilizar el proceso de instalación en una fábrica, sin la necesidad de cargar el programa completo nuevamente en su memoria. Esta función debe ir acompañada del programa descrito en el apartado anterior, junto con las medidas de ciberseguridad necesarias para garantizar que solo personal autorizado pueda acceder o manipular este tipo de información dentro del robot. Una funcionalidad muy útil sería incorporar un creador de mapas interactivo, que permita diseñar el entorno de funcionamiento del robot o importar datos desde un plano. Con la ayuda de una inteligencia artificial, el sistema podría generar automáticamente el mapa por donde circulará el robot, de manera que, con solo presionar un botón, se envíen los datos necesarios para su funcionamiento.

- ***Diseñar una estación de carga:***

Un complemento útil para el ecosistema del robot en su entorno de trabajo sería diseñar una estación de carga que permita la recarga automática de sus baterías. Esto se podría implementar mediante una modificación en la rutina de auto navegación, configurando la estación de carga como un destino al que el robot se dirija automáticamente cuando detecte un nivel bajo de batería.

13. CONCLUSIONES

El diseño del robot requirió una revisión integral de conceptos relacionados con física, mecánica, matemáticas y electrónica para garantizar una comprensión completa del sistema. La robótica es un área multidisciplinaria que engloba la mecánica (para la estructura física del robot), la electrónica (para la interacción con los accionadores y sensores), la matemática (para el sistema de control) y la programación (para la planificación de los movimientos). Para el desarrollo del proyecto, se aplicaron los conocimientos adquiridos en la carrera de ingeniería, que sentaron las bases en estas disciplinas y que son fundamentales a la hora de encarar un proyecto de robótica.

Uno de los primeros problemas en el diseño del robot, fue construir la estructura de soporte. La mayoría de los diseños de robot con ruedas mecanum, poseen una estructura de una pieza en forma de plataforma o prisma, este diseño posee la desventaja de que, en suelos con algún tipo de bache, una de las ruedas puede perder el contacto con el suelo haciendo que pierda tracción además de introducir error al sistema de los encoder. Por lo que se decidió idear algún diseño que tuviera un sistema de amortiguación simple para esta etapa del prototipo ya que en los espacios de prueba el suelo no está bien alisado. Con esto en mente se llegó al modelo que se puede observar en el capítulo 2, donde el robot consta de dos secciones, una sección posee las dos ruedas delanteras y la otra tiene las 2 ruedas traseras. Dichas secciones están unidas por un rodamiento, de forma que una de las piezas gira con respecto a la otra, por lo que cuando una de las ruedas se encuentra con un bache, este rodamiento permite que esta rueda se adapte al suelo e impedir el problema de contacto que poseía el diseño anterior. Sin embargo, cuando el robot se desplaza en forma lateral, el punto de unión, donde se encuentra el rodamiento, permite un pivotaje haciendo que el robot tome una posición de arco (es decir, la pieza delantera realizaba un pequeño giro respecto a la pieza trasera debido al torque de los motores). Este arco, hace que cambie la orientación y posición de las ruedas delanteras y por lo tanto afecta el movimiento en sí. Para solucionar esto, se agregó dos tornillos pasantes colocados en la pieza trasera para que hicieran de tope al momento de realizar la maniobra y corregir este inconveniente.

También se investigó acerca de conceptos avanzados de procesamiento de señales, como es la fusión de sensores, un tema muy utilizado en robótica ya que permite “juntar” varias medidas de distintos sensores sobre una misma magnitud. de esta forma, se obtiene las mejores virtudes de cada uno, logrando una medida mucho más confiable y precisa. Otro tema que se investigó fue acerca la mecánica del robot, el cómo obtener un modelo matemático-físico que relacione la posición y orientación del robot con la velocidad del robot y viceversa. Revisando en libros académicos y publicaciones científicas como es el funcionamiento de las ruedas mecanum, la relación entre su velocidad de giro y su desplazamiento hasta obtener el modelo que relaciona la velocidad de giro de cada una de las ruedas con la posición, orientación y velocidad del robot mediante una ecuación matricial sencilla.

Se pudo experimentar las ventajas y comodidad que presenta el diseño modular, no solo en el circuito general del robot, sino también en la etapa de programación. Permitiendo dividir el proyecto en partes más pequeñas, facilitando la tarea de diseño, armado y testeo de cada una de

las placas del robot teniendo un flujo de diseño más ordenado y cómodo de trabajar. Permitiendo encontrar fallas de manera más rápida. Esto también se vio en la parte de programación del proyecto mediante el desarrollo de librerías que se ocupan de cada tarea por separado, como puede ser la interfaz de consola, controladores PID, manejo de los sensores, comunicación entre los microcontroladores del robot, etc. Esto permitió programar y realizar correcciones de errores de cada etapa por separado y no tener que revisar el código de todo el robot para encontrar donde se produjo el problema. Para situaciones en que las librerías deben comunicarse entre sí, se crearon los struct que contiene información de las variables que representan el estado y características de dicha tarea que debe realizar la librería, siendo este elemento algo similar a un objeto en el paradigma de programación POO (Programación orientada a objetos), pudiendo experimentar las ventajas y comodidades que ofrece este tipo de estilo de programación a la hora de diseñar un programa tan complejo como fue en este caso, realizando una abstracción que permite reducir el tiempo de programación y facilitar la lectura del código.

La etapa de auto navegación fue una de las partes más desafiantes durante el diseño del robot ya que se debía buscar una alternativa al sistema de navegación “seguidor de línea” que se utiliza en distintas industrias. Las ruedas mecanum permiten una mayor movilidad que las ruedas convencionales, permitiendo desplazarse de manera lateral y girar con facilidad sin necesidad de realizar muchas maniobras por lo que esta característica se tuvo en cuenta en el desarrollo del algoritmo de auto navegación. Se pensó primero en una forma de que el robot pueda detectar cuando llegó a destino, para esto se eligió el uso de códigos QR. Los lectores de código QR están teniendo un auge en su uso industrial por su facilidad de leer e instalar, además de existir en el mercado local lectores QR con comunicación UART, cumpliendo con la filosofía de diseño modular del robot. Para el sistema de guía se tiene el modelo de posición y orientación del robot para controlar su orientación, además se eligió utilizar un anillo de sensores de ultrasonido que sirva como detección de obstáculos y además para captar el entorno del robot, aunque por un problema en el presupuesto del robot solo se llegó a instalar 2 sensores ultrasonidos del lado donde se encontraba el lector QR. El motivo de esto es que para asegurar que el robot estará en una orientación y posición deseable con respecto a la pared u objeto donde se encontraba el código QR para que el lector pudiera escanear correctamente. Además, se observa que muchos de estos robots suelen andar en entornos donde posee un elemento como una pared al lado de su zona de conducción, por lo que se agregó valor a los sensores de ultrasonido escribiendo una función extra en el código para detectar cuando los sensores de ultrasonidos se encontraron con una pared y, en caso de encontrar una, utilizarla como una guía similar a la que cumple la línea amarilla en el suelo en el sistema “seguidor de línea”. Para asegurar que los sensores siempre están apuntando al lado donde está la pared, se agregó la información sobre si cierto trayecto posee pared y para qué lado estaría, en la memoria del microcontrolador principal, para que en momento de tener que realizar un giro, el robot elija el sentido y grados de giro para que los sensores de ultrasonidos apunten hacia una pared. De esta manera, se logró diseñar un sistema de navegación robusto, con capacidades similares a un “seguidor de línea” pero con una instalación y puesta en marcha más sencilla y económica.

La conducción autónoma está en camino de convertirse en uno de los pilares tecnológicos para el desarrollo industrial y tecnológico de la Argentina por lo que es necesario realizar este tipo de desarrollo para que existan personas capacitadas en esta área en el país.

14. ANEXO I

A continuación, se coloca el código principal del microcontrolador STM32F103C8T6

```

/* USER CODE BEGIN Header */
/**
 * *****
 * @file      : main.c
 * @brief     : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2022 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"
/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "encoder.h"
#include "motor.h"
#include "stdio.h"
#include "rs485.h"
#include "pid.h"
#include "ctrl_pyv.h"
#include "ordenes.h"
#include "conversion_array.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 */
// #define TRASERO
#ifndef TRASERO
#define DIR_UC 0
#define MODE_GPIO GPIO_MODE_OUTPUT_PP
#else
#define DIR_UC 1
#define MODE_GPIO GPIO_MODE_OUTPUT_OD
#endif
/* USER CODE END PM */
/* Private variables -----*/
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

```

```

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_tx;
/* USER CODE BEGIN PV */
struct rs485 rs;
struct rs485 *comm = &rs;
struct encoder encoder_1, encoder_2;
struct encoder *enc_1 = &encoder_1;
struct encoder *enc_2 = &encoder_2;
struct encoder *aux_enc;
struct motor motor_1, motor_2;
struct motor *mot_1 = &motor_1;
struct motor *mot_2 = &motor_2;
struct motor *aux_mot;
struct pid pid_pos_1;
struct pid pid_vel_1;
struct pid pid_pos_2;
struct pid pid_vel_2;
struct ctrl_pyv controlador_1;
struct ctrl_pyv controlador_2;
#ifdef EXPERIMENTO
//
//int8_t set_point[7]={50,70,20,-50,-70,20,0};
//int8_t pwm=0;
//uint8_t indice=0;
//uint8_t acum=0;
//int16_t cuenta;
//uint8_t l;
//char texto[20];
//
//endif
/* USER CODE END PV */
/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
/* 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 */
/* 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_DMA_Init();
MX_TIM2_Init();
MX_TIM3_Init();
MX_TIM4_Init();
MX_USART1_UART_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
    asignar_pin_motor(mot_1, dir3_Pin, dir3_GPIO_Port, dir2_Pin,
dir2_GPIO_Port);
    iniciar_motor(mot_1, HORARIO, &htim1, TIM_CHANNEL_4);
    asignar_pin_motor(mot_2, dir1_Pin, dir1_GPIO_Port, dir4_Pin,
dir4_GPIO_Port);
    iniciar_motor(mot_2, HORARIO, &htim1, TIM_CHANNEL_1);
    encoder_init_vel_gpio(enc_1, enc1_p1_GPIO_Port, enc1_p1_Pin,
enc1_p2_GPIO_Port, enc1_p2_Pin);
    encoder_init_tim_vel(enc_1, &htim4, TIM_CHANNEL_2, TIM_CHANNEL_1);
    encoder_init_sentido(enc_1, NORMAL, NORMAL);
    encoder_init(enc_1, &htim3);
    encoder_init_vel_gpio(enc_2, enc2_p1_GPIO_Port, enc2_p1_Pin,
enc2_p2_GPIO_Port, enc2_p2_Pin);
    encoder_init_tim_vel(enc_2, &htim4, TIM_CHANNEL_4, TIM_CHANNEL_3);
    encoder_init_sentido(enc_2, NORMAL, NORMAL);
    encoder_init(enc_2, &htim2);
    rs485_assign_pint(comm, tx_en_GPIO_Port, tx_en_Pin);
    rs485_init(comm, &huart1, ESCLAVO);
    pid_init(&pid_pos_1, 10, DESACTIVADO);
    pid_init(&pid_pos_2, 10, DESACTIVADO);
    pid_init(&pid_vel_1, 10, ACTIVADO);
    pid_init(&pid_vel_2, 10, ACTIVADO);
    controlador_1.pid_p = &pid_pos_1;
    controlador_1.pid_v = &pid_vel_1;
    controlador_2.pid_p = &pid_pos_2;
    controlador_2.pid_v = &pid_vel_2;
    pid_vel_1.coef_prop=2;
    pid_vel_1.coef_inte=0.1;
    pid_vel_2.coef_prop=2;
    pid_vel_2.coef_inte=0.1;
    ctrl_pyv_init(&controlador_1, 10, enc_1);
    ctrl_pyv_init(&controlador_2, 10, enc_2);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
    HAL_NVIC_EnableIRQ(TIM2_IRQn);
    HAL_NVIC_EnableIRQ(TIM3_IRQn);
    HAL_NVIC_EnableIRQ(TIM4_IRQn);
    uint16_t aux_16bit;
    int32_t aux_32bit;
    float aux_float;
    uint8_t buff_tx[5];
    uint32_t aux32;
    uint32_t tick = HAL_GetTick();
    uint32_t tick_ctrl = tick;
    uint32_t tick_ctrl_ant=tick_ctrl;
    uint8_t hab_controlador=1;

```

```

//      uint32_t tick10_motor=0;
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
    while (1) {
        if (rs.new_data == 1) {
            rs.new_data = 0;
            tick = HAL_GetTick();
            tick10_motor=0;
            if ((rs.data[0] & 0xF0) == 0x10) { //MOTOR_FUNCION
                switch (rs.data[0] & 0x0F) {
                    case 0: //cambiar velocidad de motor 1
                        cambiar_pwm(mot_1, rs.data[1]);
                        break;
                    case 1: //cambiar sentido de motor 1
                        cambiar_sentido(mot_1, rs.data[1]);
                        break;
                    case 2: //alternar pines de motor 1
                        alternar_pin_sentido(mot_1);
                    case 3: //cambiar velocidad de motor 2
                        //HAL_GPIO_TogglePin(test2_GPIO_Port,
test2_Pin);

                        cambiar_pwm(mot_2, rs.data[1]);
                        break;
                    case 4: //cambiar sentido de
motor 2

                        cambiar_sentido(mot_2, rs.data[1]);
                        break;
                    case 5: //alternar pines de
motor 2

                        alternar_pin_sentido(mot_2);
                        break;
                    case 6: //alternar motores
                        aux_mot = mot_1;
                        mot_1 = mot_2;
                        mot_2 = mot_1;
                        break;
                }
                buff_tx[0] = rs.data[0];
                rs485_transmit(comm, buff_tx, 1);
            } else if ((rs.data[0] & 0xF0) == ENCODER) {
                switch (rs.data[0] & 0x0F) {
                    case 0: //ver cuentas en
encoder 1

                        aux_16bit = encoder_ver_cuentas(enc_1);
                        //aux_16bit = 0x1234;
                        buff_tx[0] = rs.data[0];
                        buff_tx[1] = (uint8_t) (aux_16bit & 0x00FF);
                        buff_tx[2] = (uint8_t) ((aux_16bit & 0xFF00) >>
8);

                        rs485_transmit(comm, buff_tx, 3);
                        break;
                    case 1: //ver velocidad en
encoder 1

                        aux_float = enc_1->velocidad_w;
                        //aux_float = 1.234;
                        buff_tx[0] = rs.data[0];
                        buff_tx[1] = (uint8_t) ('0') + (uint8_t)
(aux_float);

```

```

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[2] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[3] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[4] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    rs485_transmit(comm, buff_tx, 5);
    break;
case 2: //ver giro en encoder
1
    aux_32bit = enc_1->giro_completo;
    //aux_32bit = 0x12345678;
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) (aux_32bit & 0xFF);
    buff_tx[2] = (uint8_t) ((aux_32bit & 0xFF00) >>
8);

    buff_tx[3] = (uint8_t) ((aux_32bit & 0xFF0000) >>
16);

    buff_tx[4] = (uint8_t) ((aux_32bit & 0xFF000000)
>> 24);

    rs485_transmit(comm, buff_tx, 5);
    break;
case 3: //alternar cuentas en encoder 1
    encoder_cambiar_sentido(enc_1);
    break;
case 4: //alternar velocidad en encoder 1
    encoder_cambiar_sentido_vel(enc_1);
    break;
case 5: //ver cuentas en encoder 2
    aux_16bit = encoder_ver_cuentas(enc_2);
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) (aux_16bit & 0x00FF);
    buff_tx[2] = (uint8_t) ((aux_16bit & 0xFF00) >>
8);

    rs485_transmit(comm, buff_tx, 3);
    break;
case 6: //ver velocidad en encoder 2
    aux_float = enc_2->velocidad_w;
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[2] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[3] = (uint8_t) ('0') + (uint8_t)
(aux_float);

    aux_float = aux_float - (uint8_t) (aux_float);
    aux_float = aux_float * 10;
    buff_tx[4] = (uint8_t) ('0') + (uint8_t)
(aux_float);

```

```

        rs485_transmit(comm, buff_tx, 5);
        break;
case 7: //ver giro en encoder 2
    aux_32bit = enc_2->giro_completo;
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) (aux_32bit & 0xFF);
    buff_tx[2] = (uint8_t) ((aux_32bit & 0xFF00) >>
8);

    buff_tx[3] = (uint8_t) ((aux_32bit & 0xFF0000) >>
16);

    buff_tx[4] = (uint8_t) ((aux_32bit & 0xFF000000)
>> 24);

    rs485_transmit(comm, buff_tx, 5);
    break;
case 8: //alternar cuentas en encoder2
    encoder_cambiar_sentido(enc_2);
    break;
case 9: //alternar velocidad en encoder 2
    encoder_cambiar_sentido_vel(enc_2);
    break;
case 10:
    aux_enc = enc_1;
    enc_1 = enc_2;
    enc_2 = enc_1;
    break;
case 11: //dar posicion del encoder 1
    aux_32bit = enc_1->giro_completo;
    aux_32bit = aux_32bit * CUENTAS_POR_VUELTA;
    aux_16bit = encoder_ver_cuentas(enc_1);
    aux_32bit = aux_32bit + (int32_t) (aux_16bit);
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) (aux_32bit & 0xFF);
    buff_tx[2] = (uint8_t) ((aux_32bit & 0xFF00) >>
8);

    buff_tx[3] = (uint8_t) ((aux_32bit & 0xFF0000) >>
16);

    buff_tx[4] = (uint8_t) ((aux_32bit & 0xFF000000)
>> 24);

    rs485_transmit(comm, buff_tx, 5);
    //l=sprintf(texto,"%d,%d,%d\n", (int)enc_1-
>giro_completo, (int) aux_16bit, (int)aux_32bit);
    //HAL_UART_Transmit_IT(&huart2, (uint8_t*)texto,
1);

    break;
case 12: //dar posicion del encoder 2
    aux_32bit = enc_2->giro_completo;
    aux_32bit = aux_32bit * CUENTAS_POR_VUELTA;
    aux_16bit = encoder_ver_cuentas(enc_2);
    aux_32bit = aux_32bit + (int32_t) (aux_16bit);
    buff_tx[0] = rs.data[0];
    buff_tx[1] = (uint8_t) (aux_32bit & 0xFF);
    buff_tx[2] = (uint8_t) ((aux_32bit & 0xFF00) >>
8);

    buff_tx[3] = (uint8_t) ((aux_32bit & 0xFF0000) >>
16);

    buff_tx[4] = (uint8_t) ((aux_32bit & 0xFF000000)
>> 24);

    rs485_transmit(comm, buff_tx, 5);
    break;

```

```

24);

    case 13: //Tiempo entre cuentas de encoder 1
        aux32 = encoder_ver_tiempo_entre_cuentas(enc_1);
        //aux_16bit = 0x1234;
        buff_tx[0] = rs.data[0];
        buff_tx[1] = (uint8_t) (aux32 & 0xFF);
        buff_tx[2] = (uint8_t) ((aux32 & 0xFF00) >> 8);
        buff_tx[3] = (uint8_t) ((aux32 & 0xFF0000) >> 16);
        buff_tx[4] = (uint8_t) ((aux32 & 0xFF000000) >> 24);

        rs485_transmit(comm, buff_tx, 5);
        break;

    case 14: //Tiempo entre cuentas de encoder 2
        aux32 = encoder_ver_tiempo_entre_cuentas(enc_2);
        //aux_16bit = 0x1234;
        buff_tx[0] = rs.data[0];
        buff_tx[1] = (uint8_t) (aux32 & 0xFF);
        buff_tx[2] = (uint8_t) ((aux32 & 0xFF00) >> 8);
        buff_tx[3] = (uint8_t) ((aux32 & 0xFF0000) >> 16);
        buff_tx[4] = (uint8_t) ((aux32 & 0xFF000000) >> 24);

        rs485_transmit(comm, buff_tx, 5);
        break;

    case 15: //RESET_CUENTAS
        __ENCODER_RESET_CUENTAS(enc_1);
        __ENCODER_RESET_CUENTAS(enc_2);
        buff_tx[0]=rs.data[0];
        rs485_transmit(comm, buff_tx, 5);
    }
} else if ((rs.data[0] & 0xF0) == CONTROL_PYV) {
    switch (rs.data[0]) {
        case SET_PROPORCIONAL_POSICION_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.pid_p->coef_prop = aux_float;
            rs485_transmit(comm, rs.data, 5);
            break;

        case SET_DERIVATIVO_POSICION_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.pid_p->coef_der = aux_float;
            rs485_transmit(comm, rs.data, 5);
            break;

        case SET_PROPORCIONAL_VELOCIDAD_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.pid_v->coef_prop = aux_float;
            rs485_transmit(comm, rs.data, 5);
            break;

        case SET_INTEGRAL_VELOCIDAD_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.pid_v->coef_inte = aux_float;
            rs485_transmit(comm, rs.data, 5);
            break;

        case SET_SETPOINT_POSICION_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.set_point = aux_float;
            rs485_transmit(comm, rs.data, 5);
            break;

        case SET_SETPOINT_VELOCIDAD_1:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.vel_set_point = aux_float;
            rs485_transmit(comm, rs.data, 5);
    }
}

```

```

        break;
    case AUMENTAR_SETPOINT_POSICION_1:
        aux_float = array_to_float(rs.data + 1);
        controlador_1.set_point += aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_PROPORCIONAL_POSICION_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.pid_p->coef_prop = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_DERIVATIVO_POSICION_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.pid_p->coef_der = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_PROPORCIONAL_VELOCIDAD_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.pid_v->coef_prop = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_INTEGRAL_VELOCIDAD_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.pid_v->coef_inte = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_SETPOINT_POSICION_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.set_point = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case SET_SETPOINT_VELOCIDAD_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.vel_set_point = aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case AUMENTAR_SETPOINT_POSICION_2:
        aux_float = array_to_float(rs.data + 1);
        controlador_2.set_point += aux_float;
        rs485_transmit(comm, rs.data, 5);
        break;
    case CONTROLAR_POSICION:
        controlador_1.solo_vel = DESACTIVADO;
        controlador_2.solo_vel = DESACTIVADO;
        rs485_transmit(comm, rs.data, 5);
        break;
    case CONTROLAR_VELOCIDAD:
        controlador_1.solo_vel = ACTIVADO;
        controlador_2.solo_vel = ACTIVADO;
        rs485_transmit(comm, rs.data, 5);
        break;
    }
} else if ((rs.data[0] & 0xF0) == CONTROL_PARAM1) {
    switch (rs.data[0]) {
    case VER_PROPORCIONAL_POSICION_1:
        aux_float = controlador_1.pid_p->coef_prop;
        float_to_array(aux_float, buff_tx + 1);
        rs485_transmit(comm, buff_tx, 5);
        break;

```

```

case VER_DERIVATIVO_POSICION_1:
    aux_float = controlador_1.pid_p->coef_der;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_PROPORCIONAL_VELOCIDAD_1:
    aux_float = controlador_1.pid_v->coef_prop;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_INTEGRAL_VELOCIDAD_1:
    aux_float = controlador_1.pid_v->coef_inte;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_SETPOINT_POSICION_1:
    aux_float = controlador_1.set_point;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_SETPOINT_VELOCIDAD_1:
    aux_float = controlador_1.vel_set_point;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_PROPORCIONAL_POSICION_2:
    aux_float = controlador_2.pid_p->coef_prop;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_DERIVATIVO_POSICION_2:
    aux_float = controlador_2.pid_p->coef_der;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_PROPORCIONAL_VELOCIDAD_2:
    aux_float = controlador_2.pid_v->coef_prop;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_INTEGRAL_VELOCIDAD_2:
    aux_float = controlador_2.pid_v->coef_inte;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_SETPOINT_POSICION_2:
    aux_float = controlador_2.set_point;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_SETPOINT_VELOCIDAD_2:
    aux_float = controlador_2.vel_set_point;
    float_to_array(aux_float, buff_tx + 1);
    rs485_transmit(comm, buff_tx, 5);
    break;
case VER_MODO_DEL_CONTROLADOR:
    buff_tx[1] = controlador_1.solo_vel;
    buff_tx[2] = controlador_2.solo_vel;
    rs485_transmit(comm, buff_tx, 5);

```

```

        break;
    }
} else if ((rs.data[0] & 0xF0) == CONTROL_PARAM2) {
    switch (rs.data[0]) {
        case VER_VELOCIDAD_1:
            aux_float = controlador_1.salida_vel;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_NUMERO_DE_VUELTAS_1:
            aux_float = controlador_1.salida_pos;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_TENSION_1:
            aux_float = controlador_1.tension;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_PWM_1:
            aux_float = controlador_1.tension * (100.0 /
12.0);

            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_VELOCIDAD_2:
            aux_float = controlador_2.salida_vel;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_NUMERO_DE_VUELTAS_2:
            aux_float = controlador_2.salida_pos;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_TENSION_2:
            aux_float = controlador_2.tension;
            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case VER_PWM_2:
            aux_float = controlador_2.tension * (100.0 /
12.0);

            float_to_array(aux_float, buff_tx + 1);
            rs485_transmit(comm, buff_tx, 5);
            break;
        case SET_MIN_VEL:
            aux_float = array_to_float(rs.data + 1);
            controlador_1.min_vel_set=aux_float;
            controlador_2.min_vel_set=aux_float;
            rs485_transmit(comm, buff_tx, 5);
            break;
    }
    hab_controlador=1;
}
}
else {
    if ((comm->uart->ErrorCode != 0)
        | (comm->uart->RxState == HAL_UART_STATE_READY)

```

```

        | (HAL_GetTick() > tick + 5)) {
        tick = HAL_GetTick();
        //aux32_t=comm->uart->ErrorCode;
        aux32 = comm->uart->Instance->CR1;
        comm->uart->ErrorCode = 0;
        HAL_NVIC_DisableIRQ(USART1_IRQn);
        MX_USART1_UART_Init();
        rs485_init(comm, &huart1, ESCLAVO);
        HAL_NVIC_EnableIRQ(USART1_IRQn);
    }
}
if (HAL_GetTick() >= tick_ctrl + 10) {
    if (hab_controlador==1) {
        ctrl_pyv_update(&controlador_1);
        ctrl_pyv_update(&controlador_2);
        cambiar_pwm(mot_1,
(int8_t)(controlador_1.tension*(100.0/12.0)));
        cambiar_pwm(mot_2,
(int8_t)(controlador_2.tension*(100.0/12.0)));
    }
    tick_ctrl_ant = HAL_GetTick();
    if(10<(tick_ctrl_ant-tick_ctrl)){
        HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
    }
    encoder_handler_tim_encoder(enc_2);
    encoder_handler_tim_encoder(enc_1);
    tick_ctrl=tick_ctrl_ant;
    //
    tick10_motor++;
    //
    if(tick10_motor>=10){
    //
        if(controlador_1.solo_vel==ACTIVADO){
    //
            controlador_1.vel_set_point=0.0;
    //
            controlador_2.vel_set_point=0.0;
    //
        }
    //
        else(controlador_1.)
    //
    }
}
Conf_Flancos();

//#endif
//#ifdef EXPERIMENTO
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
//#endif
}
/* USER CODE END 3 */
}
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    /** Initializes the RCC Oscillators according to the specified parameters
     * in the RCC_OscInitTypeDef structure.
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;

```

```

RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
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_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}
/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{
    /* USER CODE BEGIN TIM1_Init 0 */
    /* USER CODE END TIM1_Init 0 */
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
    /* USER CODE BEGIN TIM1_Init 1 */
    /* USER CODE END TIM1_Init 1 */
    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 39;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 99;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)

```

```

{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMode_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);
}
/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{
    /* USER CODE BEGIN TIM2_Init 0 */
    /* USER CODE END TIM2_Init 0 */
    TIM_Encoder_InitTypeDef sConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    /* USER CODE BEGIN TIM2_Init 1 */
    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 0;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 65535;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_OC_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
    sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;

```

```

sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
sConfig.IC1Filter = 0;
sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
sConfig.IC2Filter = 0;
if (HAL_TIM_Encoder_Init(&htim2, &sConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_TIMING;
sConfigOC.Pulse = 3072;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.Pulse = 62464;
if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */
/* USER CODE END TIM2_Init 2 */
}
/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{
    /* USER CODE BEGIN TIM3_Init 0 */
    /* USER CODE END TIM3_Init 0 */
    TIM_Encoder_InitTypeDef sConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    /* USER CODE BEGIN TIM3_Init 1 */
    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_OC_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
    sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;

```

```

sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
sConfig.IC1Filter = 0;
sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
sConfig.IC2Filter = 0;
if (HAL_TIM_Encoder_Init(&htim3, &sConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_TIMING;
sConfigOC.Pulse = 3072;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.Pulse = 62464;
if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
/* USER CODE END TIM3_Init 2 */
}
/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{
    /* USER CODE BEGIN TIM4_Init 0 */
    /* USER CODE END TIM4_Init 0 */
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};
    /* USER CODE BEGIN TIM4_Init 1 */
    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 71;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 65535;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)

```

```

{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim4) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 0;
if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */
/* USER CODE END TIM4_Init 2 */
}
/**
 * @brief USART1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART1_UART_Init(void)
{
    /* USER CODE BEGIN USART1_Init 0 */
    /* USER CODE END USART1_Init 0 */
    /* USER CODE BEGIN USART1_Init 1 */
    /* USER CODE END USART1_Init 1 */
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_9B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_MultiProcessor_Init(&huart1, 0, UART_WAKEUPMETHOD_ADDRESSMARK) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

/* USER CODE BEGIN USART1_Init 2 */
if (HAL_MultiProcessor_Init(&huart1, DIR_UC, UART_WAKEUPMETHOD_ADDRESSMARK) !=
HAL_OK)
{
    Error_Handler();
}
/* USER CODE END USART1_Init 2 */
}
/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{
    /* DMA controller clock enable */
    __HAL_RCC_DMA1_CLK_ENABLE();
    /* DMA interrupt init */
    /* DMA1_Channel4_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, test_Pin|test2_Pin|test3_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, tx_en_Pin|dir1_Pin|dir2_Pin|dir3_Pin
                        |dir4_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pins : test_Pin test2_Pin test3_Pin */
    GPIO_InitStructure.Pin = test_Pin|test2_Pin|test3_Pin;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
    GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
    /*Configure GPIO pins : tx_en_Pin dir1_Pin dir2_Pin dir3_Pin
                        dir4_Pin */
    GPIO_InitStructure.Pin = tx_en_Pin|dir1_Pin|dir2_Pin|dir3_Pin
                        |dir4_Pin;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
    GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);
    /* USER CODE BEGIN MX_GPIO_Init_2 */
    GPIO_InitStructure.Pin = dir1_Pin|dir2_Pin|dir3_Pin
                        |dir4_Pin;
    GPIO_InitStructure.Mode = MODE_GPIO;
    GPIO_InitStructure.Pull = GPIO_NOPULL;

```

```

    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if ((htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3)
        | (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)) {
//        encoder_handler_tim_encoder(enc_2);
    } else {
//        encoder_handler_tim_encoder(enc_1);
    }
//    encoder_handler_tim_vel(enc_1);
//    encoder_handler_tim_vel(enc_2);
//    HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
}
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim) {
//    if (htim->Instance == TIM2) {
//        encoder_handler_tim_encoder(enc_1);
////        HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
//    } else if (htim->Instance == TIM3) {
//        encoder_handler_tim_encoder(enc_2);
////        HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
//    }
//    HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM4) {
        encoder_handler_tim_vel_update(enc_1);
        encoder_handler_tim_vel_update(enc_2);
    }
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
    rs485_uart_tx_handle(comm);
//    HAL_GPIO_TogglePin(test_GPIO_Port, test_Pin);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
//    huart->gState=HAL_UART_STATE_READY;
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    HAL_NVIC_DisableIRQ(USART1_IRQn);
    rs485_uart_rx_handle(comm);
    rs485_receive(comm);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
}
void Conf_Flancos(void) {
//    encoder_configurar_flancos(enc_1);
//    encoder_configurar_flancos(enc_2);
}
/* 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 */

```

A continuación, se muestra el código de la librería de los motores

```

#include "stm32f1xx_hal.h"
#define HORARIO 0
#define ANTIHORARIO 1
#define MAX_VEL 99
#define MIN_VEL 3
/**
 * @brief Estructura asociada a los elementos que se utilizan para manejar el
motor
 */
struct motor{
    uint32_t channel;
    uint8_t sentido;
    uint16_t pin_horario;
    GPIO_TypeDef* port_horario;
    uint16_t pin_antihorario;
    GPIO_TypeDef* port_antihorario;
    uint8_t pwm;
    TIM_HandleTypeDef *htim;
};
void asignar_pin_motor (struct motor *mimotor, uint16_t pin_horario, GPIO_TypeDef*
port_horario, uint16_t pin_antihorario,
                        GPIO_TypeDef* port_antihorario );
void iniciar_motor(struct motor *mimotor, uint8_t sentido, TIM_HandleTypeDef *
htim, uint32_t channel);
void cambiar_sentido(struct motor *mimotor, uint8_t sentido);
void cambiar_pwm(struct motor *mimotor, int8_t pwm);
void alternar_pin_sentido(struct motor *mimotor);

/**
 * @brief Funcion para iniciar los motores.
 * @param mimotor: struct con la informacion de los elementos ocupados por el
motor.
 *          Sentido: Indico el sentido de giro del motor
 *          htim: TIM HANDLE asociado al motor
 *          channel: Canal de Timer a utilizar
 * @note Inicia el pwm destinado al motor

```

```

* @retval nada
*/
void iniciar_motor(struct motor *mimotor, uint8_t sentido, TIM_HandleTypeDef *
htim, uint32_t channel){
    mimotor->htim = htim;
    mimotor->channel = channel;
    cambiar_sentido(mimotor, sentido);
    HAL_TIM_PWM_Start(mimotor->htim, mimotor->channel);
    mimotor->pwm = __HAL_TIM_GET_COMPARE(htim,channel);
}
/**
* @brief Funcion para indicar cuales son los pines conectado al driver del
motor.
* @param mimotor: struct con la informacion de los elementos ocupados por el
motor.
*          pin_horario: pin de la pata conectada al pin horario del driver.
*          port_horario: puerto de la pata conectada al pin horario del driver.
*          pin_antihorario: pin de la pata conectada al pin antihorario del driver.
*          port_antihorario: puerto de la pata conectada al pin antihorario del
driver.
* @note Almacena la informacion de los pines en la struct motor
* @retval nada
*/
void asignar_pin_motor (struct motor *mimotor, uint16_t pin_horario, GPIO_TypeDef*
port_horario, uint16_t pin_antihorario,
GPIO_TypeDef* port_antihorario ){
    mimotor->pin_horario = pin_horario;
    mimotor->pin_antihorario = pin_antihorario;
    mimotor->port_antihorario = port_antihorario;
    mimotor->port_horario = port_horario;
}
/**
* @brief Funcion para alternar el sentido del motor.
* @param mimotor: struct con la informacion de los elementos ocupados por el
motor.
*          Sentido: Indico el sentido de giro del motor
* @note Inicia el pwm destinado al motor
* @retval nada
*/
void cambiar_sentido(struct motor *mimotor, uint8_t sentido){
    if(sentido == HORARIO){
        HAL_GPIO_WritePin(mimotor->port_horario, mimotor->pin_horario, SET);
        HAL_GPIO_WritePin(mimotor->port_antihorario, mimotor->
pin_antihorario, RESET);
        mimotor->sentido= HORARIO;
    }
    else{
        HAL_GPIO_WritePin(mimotor->port_horario, mimotor->pin_horario,
RESET);
        HAL_GPIO_WritePin(mimotor->port_antihorario, mimotor->
pin_antihorario, SET);
        mimotor->sentido=ANTIHORARIO;
    }
    mimotor->pwm = - mimotor->pwm;
}
/**
* @brief Funcion para cambiar el ciclo de trabajo del pwm.
* @param mimotor: struct con la informacion de los elementos ocupados por el
motor.

```

```

*          pwm: valor del pwm a colocar
* @note   Si el pwm es negativo, cambiar el sentido actual que posee el motor
* @retval nada
*/
void cambiar_pwm(struct motor *mimotor, int8_t pwm){
    uint8_t sentido;
    if(pwm < 0){
        cambiar_sentido(mimotor, ANTIHORARIO);
        pwm=-pwm;
        sentido=ANTIHORARIO;
    }
    else{
        cambiar_sentido(mimotor, HORARIO);
        sentido=HORARIO;
    }
    if(pwm > MAX_VEL){
        pwm=MAX_VEL;
    }
    if(pwm<MIN_VEL){
        pwm=0;
    }
    __HAL_TIM_SET_COMPARE(mimotor->htim, mimotor->channel, (uint8_t) pwm);
    mimotor->pwm=pwm;
    mimotor->sentido=sentido;
}
/**
 * @brief Funcion para alternar la info. de los pines conectado al driver del
motor.
 * @param mimotor: struct con la informacion de los elementos ocupados por el
motor.
 *          pin_horario: pin de la pata conectada al pin horario del driver.
 *          port_horario: puerto de la pata conectada al pin horario del driver.
 *          pin_antihorario: pin de la pata conectada al pin antihorario del driver.
 *          port_antihorario: puerto de la pata conectada al pin antihorario del
driver.
 * @note   cambia la info de los pines antihorario y horario
 * @retval nada
*/
void alternar_pin_sentido(struct motor *mimotor){
    uint16_t pin_aux = mimotor->pin_horario;
    GPIO_TypeDef* port_aux = mimotor->port_horario;
    mimotor->pin_horario = mimotor->pin_antihorario;
    mimotor->port_horario = mimotor->port_antihorario;
    mimotor->pin_antihorario = pin_aux;
    mimotor->port_antihorario = port_aux;
    cambiar_sentido(mimotor, mimotor->sentido);
    return;
}

```

A continuación, se muestra la librería del encoder.

```

#include "stm32f1xx_hal.h"
#define ALTERNAR_VEL      1
#define ALTERNAR_GIRO    1
#define CUENTA_A_GRADOS  0.117//=360/(64*12*4)
//#define GRADOS_A_VEL_GIRO 1/0.01
//#define CUENTA_A_VEL_GIRO CUENTA_A_GRADOS * GRADOS_A_VEL_GIRO
#define RADIO_RUEDA      0.07
#define USEG_A_SEG       0.000001

```

```

#define GRADOS_A_RADIANES      3.1416
#define CUENTAS_POR_VUELTA     (64*12*4) // = 3072
#define ALTERNADO              -1
#define INVERTIDO              -1
#define NORMAL                  1
#define MAX_UPDATE              10//Maxima de actualizaciones que tiene el
tim_vel para considerar velocidad=0
#define MAX_CAPTURA_TIEMPO    8
struct encoder{
    TIM_HandleTypeDef *encoder;
    TIM_HandleTypeDef *tim_vel;
    uint32_t          channel_1;
    uint32_t          channel_2;
    GPIO_TypeDef*channel_1_port;
    uint16_t          channel_1_pin;
    GPIO_TypeDef*channel_2_port;
    uint16_t          channel_2_pin;
    uint16_t          flanco_anterior;
    uint8_t           tim_vel_update;
    uint8_t           state;
    int32_t           giro_completo;
    //int32_t          giro_anterior;
    //uint16_t         cuentas_anteriores;
    int8_t            sentido;
    int8_t            sentido_vel;
    uint32_t          tiempo_entre_cuenta[MAX_CAPTURA_TIEMPO];
    float             velocidad_w;
    float             velocidad;
    uint8_t           n_de_cap;
    float             vuelta;
    //uint8_t          sentido;
};
void encoder_init_tim_vel(volatile struct encoder* en, TIM_HandleTypeDef* tim_vel,
uint32_t channel_1, uint32_t channel_2 );
void encoder_init_vel_gpio(volatile struct encoder* en, GPIO_TypeDef* ch1_port,
uint16_t ch1_pin, GPIO_TypeDef* ch2_port,
                               uint16_t ch2_pin);
void encoder_init(volatile struct encoder* en, TIM_HandleTypeDef* tim_encoder);
void encoder_init_sentido(volatile struct encoder* en, int8_t sentido, int8_t
sentido_vel);
uint8_t encoder_handler_tim_vel(volatile struct encoder* en);
void encoder_handler_tim_vel_update(volatile struct encoder* en);
void encoder_handler_tim_encoder(volatile struct encoder* en);
uint32_t restar_flancos(uint16_t flanco_1, uint16_t flanco_2, uint8_t n_de_giro);
void ver_estado_y_ajustar_flancos(volatile struct encoder* en);
int16_t encoder_ver_cuentas(volatile struct encoder* en);
void encoder_cambiar_sentido(volatile struct encoder* en );
void encoder_cambiar_sentido_vel(volatile struct encoder* en );
void encoder_configurar_flancos(volatile struct encoder* en);
uint32_t encoder_ver_tiempo_entre_cuentas(volatile struct encoder* en);
int32_t encoder_ver_cuentas_totales(volatile struct encoder* en);
float encoder_ver_vueltas_totales(volatile struct encoder* en);
#define __ENCODER_VER_CUENTAS(en)      __HAL_TIM_GET_COUNTER(en->encoder);
#define __ENCODER_VER_CUENTAS_TOTALES(en)  en->giro_completo*CUENTAS_POR_VUELTA
+ __ENCODER_VER_CUENTAS(en)
#define __ENCODER_VER_VUELTAS(en)
    (float)(__ENCODER_VER_CUENTAS_TOTALES(en))/CUENTAS_POR_VUELTA
#define __ENCODER_RESET_CUENTAS(en) __HAL_TIM_SET_COUNTER(en->encoder, 0); en-
>giro_completo = 0; en->vuelta=0;

```

```

/**
 * @brief Funcion para almacenar info del tim que funcionara como IC en encoder.
 * @param en: struct encoder.
 *          htim: TIM_HANDLE asociado al IC del encoder
 *          channel_1: Canal 1 de Timer a utilizar
 *          channel_2: Canal 2 de Timer a utilizar
 * @note
 * @retval nada
 */
void encoder_init_tim_vel(volatile struct encoder* en, TIM_HandleTypeDef* tim_vel,
uint32_t channel_1, uint32_t channel_2 ){
    en->tim_vel = tim_vel;
    en->channel_1 = channel_1;
    en->channel_2 = channel_2;
//    __HAL_TIM_CLEAR_FLAG(tim_vel, TIM_IT_CC1);
//    __HAL_TIM_CLEAR_FLAG(tim_vel, TIM_IT_CC2);
//    __HAL_TIM_CLEAR_FLAG(tim_vel, TIM_IT_CC3);
//    __HAL_TIM_CLEAR_FLAG(tim_vel, TIM_IT_CC4);
//    HAL_TIM_IC_Start_IT(tim_vel, channel_1);
//    HAL_TIM_IC_Start_IT(tim_vel, channel_2);
}
/**
 * @brief Funcion para almacenar info de los pines del tim que funcionara como IC
en encoder.
 * @param en: struct encoder.
 *          ch1_pin: pin Canal 1 de Timer a utilizar
 *          ch2_pin: pin Canal 2 de Timer a utilizar
 *          ch1_port: port Canal 1 de Timer a utilizar
 *          ch2_port: port Canal 2 de Timer a utilizar
 * @note
 * @retval nada
 */
void encoder_init_vel_gpio(volatile struct encoder* en, GPIO_TypeDef* ch1_port,
uint16_t ch1_pin, GPIO_TypeDef* ch2_port,
uint16_t ch2_pin){
    en->channel_1_port=ch1_port;
    en->channel_1_pin=ch1_pin;
    en->channel_2_port = ch2_port;
    en->channel_2_pin = ch2_pin;
    //ver_estado_y_ajustar_flancos(en);
}
/**
 * @brief Funcion para iniciar el encoder.
 * @param en: struct encoder.
 *          tim_encoder: TIM_HANDLE del timer que funcionara como encoder.
 * @note
 * @retval nada
 */
void encoder_init(volatile struct encoder* en, TIM_HandleTypeDef* tim_encoder){
    en->encoder= tim_encoder;
    en->giro_completo = 0;
    //en->giro_anterior = 0;
    //en->cuentas_anteriores = 0;
    en->velocidad = 0;
    en->velocidad_w = 0;
    en->tiempo_entre_cuenta[0]=0;
    en->tiempo_entre_cuenta[1]=0;
    en->tiempo_entre_cuenta[2]=0;
    en->tiempo_entre_cuenta[3]=0;
}

```

```

        en->vuelta=0;
        __HAL_TIM_SET_COMPARE(en->encoder, TIM_CHANNEL_3,( 0xFFFF &
CUENTAS_POR_VUELTA));
        __HAL_TIM_SET_COMPARE(en->encoder, TIM_CHANNEL_4,(0xFFFF - ( 0xFFFF &
CUENTAS_POR_VUELTA) ));
        HAL_TIM_Encoder_Start(en->encoder, TIM_CHANNEL_ALL);
    }
    /**
     * @brief handler que se ejecuta cuando el TIM ENCODER LLEGA al valor de un giro
     de rueda.
     * @param en: struct encoder.
     * @note
     * @retval nada
     */
    void encoder_handler_tim_encoder(volatile struct encoder* en){
        int16_t cnt= __HAL_TIM_GET_COUNTER(en->encoder);
        if(cnt >= CUENTAS_POR_VUELTA){
            cnt=cnt-CUENTAS_POR_VUELTA;
            __HAL_TIM_SET_COUNTER(en->encoder, cnt);
            en->giro_completo= en->giro_completo + 1;
            en->vuelta+=1.0;
        }
        if(cnt <= -CUENTAS_POR_VUELTA){
            cnt=cnt+CUENTAS_POR_VUELTA;
            __HAL_TIM_SET_COUNTER(en->encoder, cnt);
            en->giro_completo= en->giro_completo - 1;
            en->vuelta-=1.0;
        }
    }
    /**
     * @brief FUNCION para indicar el "sentido" de como se conector el encoder al uc.
     * @param en: struct encoder.
     * @note
     * @retval nada
     */
    void encoder_init_sentido(volatile struct encoder* en, int8_t sentido, int8_t
sentido_vel){
        en->sentido = sentido;
        en->sentido_vel = sentido_vel;
    }
    /**
     * @brief funcion para restar dos flancos captados por TIMER IC.
     * @param flanco_1: valor del flanco 1.
     *         flanco_2: valor del flanco 2.
     *         n_de_giro: numero de update que existen entre los dos flancos
     * @note la operacion que se realiza es flanco1-flanco2
     * @retval resultado de la resta
     */
    uint32_t restar_flancos(uint16_t flanco_1, uint16_t flanco_2, uint8_t n_de_giro){
        uint32_t resultado;
        if(flanco_1< flanco_2){
            resultado = (uint32_t)(flanco_1) + (0x10000 - (uint32_t)(flanco_2));
            if(n_de_giro >=1){
                n_de_giro--;
            }
        }
        else{
            resultado = (uint32_t)(flanco_1)- (uint32_t)(flanco_2);
        }
    }

```

```

        if(n_de_giro !=0){
            resultado = n_de_giro * 0x10000 + resultado;
        }
        return resultado;
    }
    /**
     * @brief funcion que devuelve el numero de cuentas del encoder.
     * @param en: struct encoder
     * @note
     * @retval cuenta ajustado con el sentido indicado
     */
    int16_t encoder_ver_cuentas(volatile struct encoder* en){
        int16_t cuenta;
        uint16_t cuenta_u;
        cuenta_u=__HAL_TIM_GET_COUNTER(en->encoder);
        cuenta =(int16_t)(cuenta_u);
        if(en->sentido==INVERTIDO){
            cuenta = -cuenta;
        }
        return cuenta;
    }
    /**
     * @brief funcion para cambiar el sentido de las cuentas.
     * @param en: struct encoder
     * @note
     * @retval nada
     */
    void encoder_cambiar_sentido(volatile struct encoder* en ){
        if(en->sentido == NORMAL){
            en->sentido = INVERTIDO;
        }
        else{
            en->sentido = NORMAL;
        }
    }
    /**
     * @brief funcion para cambiar el sentido de las tomas de velocidad del timer IC.
     * @param en: struct encoder
     * @note
     * @retval nada
     */
    void encoder_cambiar_sentido_vel(volatile struct encoder* en ){
        if(en->sentido_vel == NORMAL){
            en->sentido_vel = INVERTIDO;
        }
        else{
            en->sentido_vel = NORMAL;
        }
    }
    /**
     * @brief funcion para realizar una resta en valor absoluto.
     * @param x:numero a restar
     *         y:numero a restar
     * @note realiza al operacion |x-y|
     * @retval resultado de la resta
     */
    uint8_t resta_abs(uint32_t x, uint32_t y){
        if(x>=y){
            return (x-y);
        }
    }

```

```

    }
    else{
        return(y-x);
    }
}
/**
 * @brief funcion para detectar cuando hubo un problema en la captura de unos de
los tiempos.
 * @param en: struct encoder
 * @note
 * @retval nada
 */
uint8_t detectar_tiempo_fuera_de_lugar(volatile uint32_t* array, uint32_t*
array_resultado){
    uint8_t n=0;
    uint32_t array_ordenado[MAX_CAPTURA_TIEMPO];
    uint8_t aux;
    //armo el arreglo con los n primeros valores validos
    for(int i=0;i<MAX_CAPTURA_TIEMPO;i++){
        if(array[i]>100){
            array_ordenado[i]=*(array+i);
            n++;
        }
        else{
            break;
        }
    }
    if(n<4){
        return 0;
    }
    //lo ordeno de manera ascendente
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (array_ordenado[j] < array_ordenado[i]) {
                aux = array_ordenado[i];
                array_ordenado[i] = array_ordenado[j];
                array_ordenado[j] = aux;
            }
        }
    }
    uint32_t mediana;
    if(n%2==0){
        mediana=(array_ordenado[(n-1)/2])/2 + (array_ordenado[(n+1)/2])/2;
    }
    else{
        mediana=array_ordenado[n/2];
    }
    uint32_t umbral = mediana<<1;
    uint8_t indice_atipico[MAX_CAPTURA_TIEMPO];
    uint8_t j=0;
    for(int i=0;i<n;i++){
        if(resta_abs(array_ordenado[i],mediana)>umbral){
            indice_atipico[j]=i;
            j++;
        }
    }
    uint8_t i_inicio = 0;
    uint8_t i_final=0;
    if (j>0) {

```

```

        for(int i=0;i<j;i++){
            if((indice_atipico[j]-i_inicio)>=3){
                i_final=i_inicio + 3;
                break;
            }
            i_inicio=indice_atipico[j];
        }
        if(i_final>=MAX_CAPTURA_TIEMPO){
            return 0;
        }
    }
    else{
        i_final=3;
    }
    for(int i=0;i<4;i++){
        *(array_resultado+i)=array_ordenado[i_inicio+i];
    }
    return 1;
}
/**
 * @brief funcion para realizar un promedio de las diferentes capturas del timer
IC para determinar el tiempo entre cuenta.
 * @param en: struct encoder
 * @note
 * @retval nada
 */
uint32_t encoder_ver_tiempo_entre_cuentas(volatile struct encoder* en){
    return 0;
}
int32_t encoder_ver_cuentas_totales(volatile struct encoder* en){
    int32_t aux32=en->giro_completo*CUENTAS_POR_VUELTA +
(int32_t)encoder_ver_cuentas(en);
    return aux32;
}
float encoder_ver_vueltas_totales(volatile struct encoder* en){
    int16_t c=__HAL_TIM_GET_COUNTER(en->encoder);
    float v=c*1.0;
    float b=en->vuelta;
    v=v/(CUENTAS_POR_VUELTA);
    v=b+v;
    return v;
}

```

A continuación, se muestra la librería del RS485

```

//#ifndef __STM32F1xx_HAL_H
//#include "stm32f1xx_hal.h"
//#endif
//#ifndef __STM32F4xx_HAL_H
    #include "stm32f1xx_hal.h"
//#endif
#define MAX_LENGTH          9
struct rs485{
    uint8_t status;
    uint8_t master;
    uint8_t send_data;
    uint16_t tx_buffer[MAX_LENGTH];
    //uint16_t tx_size;
    uint16_t rx_buffer[MAX_LENGTH];
}

```

```

    //uint16_t rx_size;
    uint8_t new_data;
    uint8_t data[MAX_LENGTH - 4];
    //uint8_t len_data;
    UART_HandleTypeDef* uart;
    GPIO_TypeDef* transmitir_port;
    uint16_t transmitir_pin;
};

#define MAESTRO 0
#define ESCLAVO 1
#define MASTER MAESTRO
#define SLAVE ESCLAVO
#define TAIL1 0x0D
#define TAIL2 0x0A
//Error_code
#define PAQ_OK 0
#define PAQ_ER_TAIL 1
#define PAQ_ER_HEAD 2
#define PAQ_ER_CRC 3
void rs485_init(struct rs485* my_rs485, UART_HandleTypeDef* huart, uint8_t modo);
void rs485_assign_pint(struct rs485* my_rs485, GPIO_TypeDef* tx_port, uint16_t tx_pin);
void rs485_transmit(struct rs485* my_rs485, uint8_t* data, uint8_t size);
void rs485_transmit_master(struct rs485* my_rs485, uint8_t* data, uint8_t size, uint8_t dir);
void rs485_uart_tx_handle( struct rs485* my_rs485);
void rs485_uart_rx_handle( struct rs485* my_rs485);
uint8_t rs485_analizer_paq( struct rs485* my_rs485);
uint8_t rs485_crc(uint8_t val, uint8_t crc8_v);
void rs485_receive(struct rs485* my_rs485);

/**
 * @brief funcion para iniciar la comunicacion .
 * @param my_rs485: struct rs485
 *          huart: UART HANDLE a usar para la comunicacion rs485
 *          modo: indica si trabajara como esclavo o como maestro
 * @note
 * @retval nada
 */
void rs485_init(struct rs485* my_rs485, UART_HandleTypeDef* huart, uint8_t modo){
    my_rs485->uart= huart;
    my_rs485->master = modo;
    rs485_receive(my_rs485);
}

/**
 * @brief funcion para activar la recepcion de datos.
 * @param my_rs485: struct rs485
 * @note
 * @retval nada
 */
void rs485_receive(struct rs485* my_rs485){
    HAL_UART_Receive_IT(my_rs485->uart, (uint8_t*) my_rs485->rx_buffer, MAX_LENGTH);
    __HAL_UART_ENABLE_IT(my_rs485->uart, UART_IT_RXNE);
}

/**
 * @brief funcion para realizar una transmicion de datos en modo esclavo.
 * @param my_rs485: struct rs485
 *          data: puntero del array de datos a enviar

```

```

*          size: tamaño del array
* @note
* @retval nada
*/
void rs485_transmit(struct rs485* my_rs485, uint8_t* data, uint8_t size){
    //my_rs485->tx_size=(uint16_t) size;
    my_rs485->tx_buffer[0]=0x0; //Porque no tienen direccion
    //my_rs485->tx_buffer[1]=HEAD2;
    if(size>MAX_LENGTH-4){ //Si el tamaño de la cadena a mandar es mayor a
MAX_LENGTH menos
        //HEAD1 CRC TAIL1 TAIL2
        size = MAX_LENGTH-4;
    }
    uint8_t len_tx=0;
    while(len_tx<size){
        my_rs485->tx_buffer[len_tx + 1]=*(data + len_tx);
        len_tx++;
        //data++;
        //size--;
    }
    len_tx = size;
    while (len_tx< MAX_LENGTH){
        my_rs485->tx_buffer[len_tx + 1] = 0;
        len_tx++;
    }
    uint8_t crc=0;
    uint8_t i=0;
    while(i<MAX_LENGTH-3){
        crc = rs485_crc((uint8_t)my_rs485->tx_buffer[i], crc);
        i++;
    }
    my_rs485->tx_buffer[MAX_LENGTH - 3] = (uint16_t)crc;
    my_rs485->tx_buffer[MAX_LENGTH - 2] = TAIL1;
    my_rs485->tx_buffer[MAX_LENGTH - 1] = TAIL2;
    //my_rs485->tx_size = size + 4;
    HAL_GPIO_WritePin(my_rs485->transmitir_port, my_rs485->transmitir_pin, SET);
    HAL_StatusTypeDef st;
    st=HAL_UART_Transmit_DMA(my_rs485->uart, (uint8_t*)my_rs485->tx_buffer,
MAX_LENGTH);
    while(st!= HAL_OK){
        st=HAL_UART_Transmit(my_rs485->uart, (uint8_t*)my_rs485->tx_buffer,
MAX_LENGTH, HAL_MAX_DELAY);
    }
    //HAL_GPIO_WritePin(my_rs485->transmitir_port, my_rs485->transmitir_pin,
RESET);
}
/**
* @brief funcion para realizar una transmision de datos en modo maestro.
* @param my_rs485: struct rs485
*          data: puntero del array de datos a enviar
*          size: tamaño del array
*          dir: direccion del uc a comunicar
* @note
* @retval nada
*/
void rs485_transmit_master(struct rs485* my_rs485, uint8_t* data, uint8_t size,
uint8_t dir){
    //my_rs485->tx_size=(uint16_t) size;
    my_rs485->tx_buffer[0]=0x100 + (uint16_t) dir;

```

```

    //my_rs485->tx_buffer[1]=HEAD1;
    //my_rs485->tx_buffer[2]=HEAD2;;
    if(size>MAX_LENGTH-4){ //Si el tamaño de la cadena a mandar es mayor a
MAX_LENGTH menos
        //DIR HEAD1 CRC TAIL1 TAIL2
        size = MAX_LENGTH-4;
    }
    uint8_t len_tx=0;
    while(len_tx<size){
        my_rs485->tx_buffer[len_tx+1]=*(data + len_tx);
        len_tx++;
        //data++;
        //size--;
    }
    len_tx = size+1;
    while (len_tx< MAX_LENGTH-3){
        my_rs485->tx_buffer[len_tx] = 0;
        len_tx++;
    }
    uint8_t crc=0;
    uint8_t i=0;
    while(i<MAX_LENGTH-3){
        crc = rs485_crc((uint8_t)my_rs485->tx_buffer[i], crc);
        i++;
    }
    my_rs485->tx_buffer[MAX_LENGTH - 3] = (uint16_t)crc;
    my_rs485->tx_buffer[MAX_LENGTH - 2] = TAIL1;
    my_rs485->tx_buffer[MAX_LENGTH - 1] = TAIL2;
    //my_rs485->tx_size = size + 5;
    HAL_GPIO_WritePin(my_rs485->transmitir_port, my_rs485->transmitir_pin, SET);
    HAL_UART_Transmit_IT(my_rs485->uart, (uint8_t*)my_rs485->tx_buffer,
MAX_LENGTH);
}
/**
 * @brief funcion para realizar el crc de la comunicacion.
 * @param val:valor a aplicar el crc
 *        crc8_v: crc acumulado
 * @note
 * @retval crc acumulado
 */
uint8_t rs485_crc(uint8_t val, uint8_t crc8_v){
    uint8_t temp = 0;
    crc8_v ^= val;
    if (crc8_v & 0x01)
        temp ^= 0x5E;
    if (crc8_v & 0x02)
        temp ^= 0xBC;
    if (crc8_v & 0x04)
        temp ^= 0x61;
    if (crc8_v & 0x08)
        temp ^= 0xC2;
    if (crc8_v & 0x10)
        temp ^= 0x9D;
    if (crc8_v & 0x20)
        temp ^= 0x23;
    if (crc8_v & 0x40)
        temp ^= 0x46;
    if (crc8_v & 0x80)
        temp ^= 0x8C;

```

```

        crc8_v = temp;
        return (crc8_v);
    }
    /**
     * @brief funcion para analizar el paquetes de datos recibidos.
     * @param my_rs485: struct rs485
     * @note
     * @retval estado del paquete
     */
    uint8_t rs485_analyzer_paq( struct rs485* my_rs485){
        //Verificar Tail
        if(my_rs485->rx_buffer[MAX_LENGTH-1]!= TAIL2){
            return PAQ_ER_TAIL;
        }
        if(my_rs485->rx_buffer[MAX_LENGTH-2]!= TAIL1){
            return PAQ_ER_TAIL;
        }
        //Verificar CRC
        uint8_t crc=0;
        uint8_t i=0;//Pg hay dir + HEAD
        uint8_t i_crc=MAX_LENGTH-3;//
        while(i<i_crc){
            crc = rs485_crc((uint8_t)my_rs485->rx_buffer[i], crc);
            i++;
        }
        if((uint8_t) my_rs485->rx_buffer[i_crc]!=crc){
            return PAQ_ER_CRC;
        }
        //Armar el paquete de datos en 8bit
        if(my_rs485->master == SLAVE){
            i=1;
        }
        else{
            i=0;
        }
        uint8_t i_d=0;
        //my_rs485->len_data=MAX_LENGTH-4;//pg hay dir + crc+ tail + tail
        while(i<i_crc){
            my_rs485->data[i_d] = (uint8_t) my_rs485->rx_buffer[i];
            i++;
            i_d++;
        }
        my_rs485->new_data = 1;
        HAL_UART_Receive_IT(my_rs485->uart, (uint8_t*) my_rs485->rx_buffer,
MAX_LENGTH);
        return PAQ_OK;
    }
    /**
     * @brief funcion handler cuando se acaba una recepcion de datos.
     * @param my_rs485: struct rs485
     * @note
     * @retval nada
     */
    void rs485_uart_rx_handle(struct rs485* my_rs485){
        uint8_t e;
        e= rs485_analyzer_paq(my_rs485);
        if(e!= PAQ_OK){
            my_rs485->status=e;
        }
    }

```

```

        HAL_UART_Receive_IT(my_rs485->uart, (uint8_t*) my_rs485->rx_buffer,
MAX_LENGTH);
        __HAL_UART_ENABLE_IT(my_rs485->uart, UART_IT_RXNE);
    }
    /**
     * @brief funcion handler cuando se acaba una transmision de datos.
     * @param my_rs485: struct rs485
     * @note
     * @retval nada
     */
    void rs485_uart_tx_handle( struct rs485* my_rs485){
        HAL_GPIO_WritePin(my_rs485->transmitir_port, my_rs485->transmitir_pin,
RESET);
        HAL_UART_Receive_IT(my_rs485->uart, (uint8_t*)my_rs485->rx_buffer,
MAX_LENGTH);
    }
    /**
     * @brief funcion asignar el pin para switchear entre tx/rx.
     * @param my_rs485: struct rs485
     * @note
     * @retval nada
     */
    void rs485_assign_pint(struct rs485* my_rs485, GPIO_TypeDef* tx_port, uint16_t
tx_pin){
        my_rs485->transmitir_port = tx_port;
        my_rs485->transmitir_pin = tx_pin;
        HAL_GPIO_WritePin(tx_port, tx_pin, RESET);
    }
}

```

A continuación, se muestra la librería del controlador PID

```

/*
 * pid.h
 *
 * Created on: 27 abr. 2023
 * Author: Gonzalo Mansilla
 */
#include "stm32f1xx_hal.h"
#ifndef INC_PID_H_
#define INC_PID_H_
#define ACTIVADO 1
#define DESACTIVADO 0
#define PWM_MAX 100
#define LIMITE_SUP 12
#define LIMITE_INF -12
/*brief: estructura de datos que reúne los elementos relacionados con el PID
 *
 */
struct pid{
    float set_point;
    float salida_a_planta;
    float salida_sistema;
    float coef_prop;
    float coef_inte;
    float coef_der;
    float termino_intergrativo_acum;
    float error_anterior;
    uint8_t windup;
    uint8_t habilitacion;
}

```

```

    uint8_t saturar;
    int ts_ms;
    float error;
    float ter_prop;
    float ter_der;
    float ter_inte;
    float salida_anterior;
    float limite_inf;
    float limite_sup;
};

float pid_handler(struct pid *my_pid, float salida);
void pid_init(struct pid *my_pid, int ts, uint8_t windup);
#define __PID_SETPoint(pid,x)    pid.set_point=x
#define __PID_COEFICIENTE_PROPORCIONAL(pid,a)    pid.coef_prop=a
#define __PID_COEFICIENTE_INTEGRAL(pid,b)        pid.coef_inte=a
#define __PID_COEFICIENTE_DERIVATICO(pid,a)      pid.coef_der=a
//#endif /* INC_PID_H */

/*
 * pid.c
 *
 * Created on: 27 abr. 2023
 * Author: Gonzalo Mansilla
 */
#include <pid.h>
#include "stm32f1xx_hal.h"
/* @brief: Funcion para inicializar variables del PID
 * @param: my_pid: estructura del pid
 *          ts: tiempo de muestreo en milisegundos
 *          windup: habilitacion del windup
 * @ret: nada
 */
void pid_init(struct pid *my_pid, int ts, uint8_t windup){
    my_pid->ts_ms=ts;
    my_pid->windup=windup;
    my_pid->habilitacion=ACTIVADO;
    my_pid->coef_der=0;
    my_pid->coef_inte=0;
    my_pid->coef_prop=0;
    my_pid->error_anterior=0;
    my_pid->termino_intergrativo_acum=0;
    my_pid->set_point=0;
    my_pid->saturar=ACTIVADO;
    my_pid->limite_inf=LIMITE_INF;
    my_pid->limite_sup=LIMITE_SUP;
}
/* @brief: Funcion para inicializar variables del PID
 * @note: Debe ejecutarse de manera periodica en intervalo del ts indicado en la funcion init
 * @param: my_pid: estructura del pid
 *          salida: salida del sensor de medicion
 * @ret: Señal de control de la planta
 */
float pid_handler(struct pid *my_pid, float salida){
    if(my_pid->habilitacion== DESACTIVADO){
        my_pid->error_anterior=0;
        my_pid->termino_intergrativo_acum=0;
        my_pid->salida_sistema=salida;
        my_pid->salida_a_planta=salida;
    }
}

```

```

        return my_pid->set_point;
    }
    my_pid->salida_sistema=salida;
    my_pid->error= my_pid->set_point-salida;
    my_pid->ter_prop = my_pid->error * my_pid->coef_prop;
    my_pid->ter_der = my_pid->coef_der*(my_pid->error-my_pid->error_anterior);
    //float ter_inte = 0;
    my_pid->salida_anterior=my_pid->salida_a_planta;
    my_pid->ter_inte= my_pid->coef_inte*my_pid->error + my_pid-
>termino_intergrativo_acum;
    my_pid->salida_a_planta=my_pid->ter_prop + my_pid->ter_der + my_pid-
>ter_inte;
    if (my_pid->saturar==ACTIVADO) {
        if(my_pid->salida_a_planta>my_pid->limite_sup){
            my_pid->ter_inte=my_pid->termino_intergrativo_acum;
            my_pid->salida_a_planta=my_pid->ter_prop+my_pid-
>ter_der+my_pid->ter_inte;
            if(my_pid->salida_a_planta>my_pid->limite_sup){
                my_pid->salida_a_planta=my_pid->limite_sup;
                //if(my_pid->windup==ACTIVADO) my_pid-
>termino_intergrativo_acum=0;
            }
            //
        }
        else if(my_pid->salida_a_planta<my_pid->limite_inf){
            my_pid->ter_inte=my_pid->termino_intergrativo_acum;
            my_pid->salida_a_planta=my_pid->ter_prop+my_pid-
>ter_der+my_pid->ter_inte;
            if ((my_pid->salida_a_planta<my_pid->limite_inf)) {
                my_pid->salida_a_planta=my_pid->limite_inf;
                //
            }
            //
            if(my_pid->windup==ACTIVADO) my_pid-
>termino_intergrativo_acum=0;
        }
    }
    //->salida_a_planta=u;
    my_pid->error_anterior=my_pid->error;
    my_pid->termino_intergrativo_acum = my_pid->ter_inte;
    return my_pid->salida_a_planta;
}

```

A continuación, se muestra el código de controlador de velocidad y posición.

```

/*
 * ctrl_pyv.h
 *
 * Created on: 8 jun. 2023
 * Author: Gonzalo Mansilla
 */
#include "pid.h"
#include "encoder.h"
#ifndef INC_CTRL_PYV_H_
#define INC_CTRL_PYV_H_
#endif /* INC_CTRL_PYV_H_ */
#define MIN_E_INPUT_P 0.1
#define MIN_E_INPUT_V 0.01
struct ctrl_pyv{
    struct pid* pid_v;

```

```

    struct pid* pid_p;
    volatile struct encoder* enc;
    float ts_ms;
    float set_point;
    int32_t cuenta;
    int32_t cuenta_ant;
    int32_t delta_cuenta;
    float salida_pos;
    float salida_pos_ant;
    float delta_salida;
    float salida_vel;
    float vel;
    float tension;
    uint8_t solo_vel;
    float vel_set_point;
    // float delta_v_min;
    // float delta_p_min;
    float min_vel;
    float min_vel_set;
};

void ctrl_pyv_init( struct ctrl_pyv* ctrl, float ts_ms, struct encoder* enc);
float ctrl_pyv_update( struct ctrl_pyv* ctrl);
void ctrl_pyv_reset_pos( struct ctrl_pyv* ctrl);
//float ctrl_pyv_get_vel( struct ctrl_pyv* ctrl);
#define __ctrl_pyv_set_prop_posicion(ctrl_pyv,x)      ctrl_pyv->pid_p->coef_prop=x
#define __ctrl_pyv_set_der_posicion(ctrl_pyv,x)      ctrl_pyv->pid_p->
>coef_der=x
#define __ctrl_pyv_set_prop_velocidad(ctrl_pyv,x)    ctrl_pyv->pid_v->coef_prop=x
#define __ctrl_pyv_set_inte_velocidad(ctrl_pyv,x)    ctrl_pyv->pid_v->coef_inte=x
#define __ctrl_pyv_set_point(ctrl_pyv,x)            ctrl_pyv->set_point=x
#define __ctrl_pyv_set_encoder(ctrl_pyv,x)          ctrl_pyv->enc=x
#define __error_v_mayor_que_delta(ctrl_pyv,e)        e>ctrl_pyv->delta_v_min || e<-
ctrl_pyv->delta_v_min
#define __error_p_mayor_que_delta(ctrl_pyv,e)        e>ctrl_pyv->delta_p_min || e<-
ctrl_pyv->delta_p_min
#define __vel_mayor_a_vel_min(ctrl_pyv)              ctrl_pyv->
>vel>ctrl_pyv->min_vel || ctrl_pyv->vel<=-ctrl_pyv->min_vel

/*
 * ctrl_pyv.c
 *
 * Created on: 8 jun. 2023
 * Author: Gonzalo Mansilla
 */
#include "pid.h"
#include "encoder.h"
#include "ctrl_pyv.h"
#include "stm32f1xx_hal.h"
/**
 * @brief funcion de inicializacion.
 * @param ctrl: Struct del sistema de control de los motores
 *          ts_ms: tiempo de muestreo en milisegundos
 *          enc: Struct del encoder asociado al motor
 * @note
 * @retval nada
 */
void ctrl_pyv_init( struct ctrl_pyv* ctrl, float ts_ms, struct encoder* enc){
    ctrl->enc=enc;
    ctrl->ts_ms=ts_ms;

```

```

    __HAL_TIM_SET_COUNTER(enc->encoder, 0);
    enc->giro_completo = 0;
    ctrl->cuenta=0;
    ctrl->salida_pos=0;
    ctrl->salida_pos_ant=0;
    ctrl->set_point=0;
    ctrl->salida_vel=0;
    ctrl->vel=0;
    pid_init((ctrl->pid_p), ts_ms, DESACTIVADO);
    pid_init((ctrl->pid_v), ts_ms, ACTIVADO);
    ctrl->pid_p->coef_prop=20.0;
    ctrl->pid_p->coef_der=0.0f;
    ctrl->pid_p->coef_inte=0.00;
    ctrl->pid_v->coef_prop=1.0;
    ctrl->pid_v->coef_der=0.0;
    ctrl->pid_v->coef_inte=0.08;
    ctrl->solo_vel=DESACTIVADO;
    ctrl->vel_set_point=0.0;
    ctrl->min_vel_set=0.1;
    ctrl->min_vel=0.1;
    return;
}
/* @brief:   Funcion que realiza la actualizacion y las acciones de control del
sistema
* @note:     Debe ejecutarse de manera periodica en intervalo del ts indicado en
la funcion init
* @param:    ctrl: struct del sistema de control
* @ret:      Señal de control de la planta
*/
float ctrl_pyv_update( struct ctrl_pyv* ctrl){
    //ctrl->cuenta=encoder_ver_cuentas_totales(ctrl->enc);
    ctrl->salida_pos=encoder_ver_vueltas_totales(ctrl->enc);//((float)(ctrl-
>cuenta))/CUENTAS_POR_VUELTA;
    if(ctrl->salida_pos<ctrl->salida_pos_ant){
        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
    }
    ctrl->delta_salida=ctrl->salida_pos-ctrl->salida_pos_ant;
    ctrl->delta_cuenta=ctrl->cuenta- ctrl->cuenta_ant;
    ctrl->salida_vel=ctrl->delta_salida*(1000.0/ctrl->ts_ms);
    ctrl->pid_p->coef_inte=0.0;
    ctrl->pid_p->set_point=ctrl->set_point;
    // float e_p=ctrl->set_point-ctrl->salida_pos;
    //ctrl->pid_p->windup=DESACTIVADO;
    if(ctrl->solo_vel==DESACTIVADO){
        ctrl->vel=pid_handler((ctrl->pid_p), ctrl->salida_pos);//Controlador
PD de posicion
    }
    else{
        ctrl->vel=ctrl->vel_set_point;
    }
    if(__vel_mayor_a_vel_min(ctrl)) {
        ctrl->pid_v->set_point=ctrl->vel;
        ctrl->min_vel=ctrl->min_vel_set;
    }
    else {
        ctrl->pid_v->set_point=0;
        ctrl->pid_v->termino_intergrativo_acum=0;
        ctrl->tension=0;
        ctrl->salida_pos_ant=ctrl->salida_pos;
    }
}

```

```

        ctrl->min_vel=ctrl->min_vel_set + 0.05; //smit trigger
        return 0;
    }
    ctrl->pid_v->coef_der=0.0;
    //ctrl->pid_v->windup=DESACTIVADO;
    ctrl->tension=pid_handler((ctrl->pid_v), ctrl->salida_vel);
    ctrl->salida_pos_ant=ctrl->salida_pos;
    return ctrl->tension;
}
/* @brief:   Funcion resetear las variables de control del sistema de posicion
 * @param:   ctrl: struct del sistema de control
 */
void ctrl_pyv_reset_pos( struct ctrl_pyv* ctrl){
    __HAL_TIM_SET_COUNTER(ctrl->enc->encoder, 0);
    ctrl->enc->giro_completo = 0;
}

```

A continuación, se muestra el código del conversor de array.

```

/*
 * conversion_array.c
 *
 * Created on: 10 jun. 2023
 * Author: Gonzalo Mansilla
 */
#include "conversion_array.h"
void int16_to_array(int16_t in, uint8_t* out){
    *out=(uint8_t)(in&0x0F);
    *(out+1)=(uint8_t)((in&0xF0)>>8);
}
void uint16_to_array(uint16_t in, uint8_t* out){
    *out=(uint8_t)(in&0x0F);
    *(out+1)=(uint8_t)((in&0xF0)>>8);
}
void int32_to_array(int32_t in, uint8_t* out){
    *out=(uint8_t)(in&0x0F);
    *(out+1)=(uint8_t)((in&0xF0)>>8);
    *(out+2)=(uint8_t)((in&0xF00)>>16);
    *(out+3)=(uint8_t)((in&0xF000)>>24);
}
void uint32_to_array(uint32_t in, uint8_t* out){
    *out=(uint8_t)(in&0x0F);
    *(out+1)=(uint8_t)((in&0xF0)>>8);
    *(out+2)=(uint8_t)((in&0xF00)>>16);
    *(out+3)=(uint8_t)((in&0xF000)>>24);
}
void float_to_array(float in, uint8_t* out){
    uint32_t* p;
    uint8_t* pc;
    p=(uint32_t*)&in;
    pc=(uint8_t*)(p);
    *out=(*p);
    *(out+1)=(*pc+1);
    *(out+2)=(*pc+2);
    *(out+3)=(*pc+3);
}
int16_t array_to_int16(uint8_t* in){
    uint16_t o=(uint16_t)(*(in))+ ((uint16_t)(*(in+1))<<8);
    return((int16_t)(o));
}

```

```

}
uint16_t array_to_uint16(uint8_t* in){
    uint16_t o=(uint16_t)(*(in))+ ((uint16_t)(*(in+1))<<8);
    return o;
}
int32_t array_to_int32(uint8_t* in){
    uint32_t o=((uint32_t)(*(in))&0xFF)+ (((uint32_t)(*(in+1))<<8)&0xFF00)+
    (((uint32_t)(*(in+1))<<16)&0xFF0000) +
    (((uint32_t)(*(in+3))<<24)&0xFF000000);
    return ((int32_t)(o));
}
uint32_t array_to_uint32(uint8_t* in){
    uint32_t o=((uint32_t)(*(in))&0xFF)+ (((uint32_t)(*(in+1))<<8)&0xFF00)+
    (((uint32_t)(*(in+1))<<16)&0xFF0000) +
    (((uint32_t)(*(in+3))<<24)&0xFF000000);
    return o;
}
float array_to_float(uint8_t* in){
    float out;
    uint32_t* o=(uint32_t*)&out;
    uint8_t* pc=(uint8_t*)o;
    *pc=*in;
    *(pc+1)=*(in+1);
    *(pc+2)=*(in+2);
    *(pc+3)=*(in+3);
    return out;
}

```

A continuación, los símbolos utilizados para la comunicación entre los microcontroladores.

```

/*
 * ordenes.h
 *
 * Created on: Nov 24, 2022
 * Author: Gonzalo Mansilla
 */
#ifndef INC_ORDENES_H_
#define INC_ORDENES_H_
#endif /* INC_ORDENES_H_ */
//A CONTINUACION SE MUESTRAN LOS GRUPOS DE ORDENES DISPONIBLES PARA BLUEPILL
#define MOTOR 0x10
#define ENCODER 0x00
#define CONTROL_PYV 0x20
#define CONTROL_PARAM1 0x30
#define CONTROL_PARAM2 0x40
//A CONTINUACION SE MUESTRAN LAS ORDENES PARA LOS MOTORES
#define CAMBIAR_VELOCIDAD_MOTOR_1 MOTOR|0x00
#define CAMBIAR_SENTIDO_MOTOR_1 MOTOR|0x01
#define ALTERNAR_PINES_MOTOR_1 MOTOR|0x02
#define CAMBIAR_VELOCIDAD_MOTOR_2 MOTOR|0x03
#define CAMBIAR_SENTIDO_MOTOR_2 MOTOR|0x04
#define ALTERNAR_PINES_MOTOR_3 MOTOR|0x05
#define ALTERNAR_MOTORES MOTOR|0x06
#define HORARIO 0
#define ANTIHORARIO 1
//A CONTINUACION SE MUESTRAN LAS ORDENES PARA LOS ENCODER
#define VER_CUENTAS_ENCODER_1 ENCODER|0x00
#define VER_VELOCIDAD_ENCODER_1 ENCODER|0x01
#define VER_CANTIDAD_DE_GIRO_ENCODER_1 ENCODER|0x02

```

```

#define ALTERNAR_SENTIDO_CUENTA_ENCODER_1          ENCODER|0x03
#define ALTERNAR_SENTIDO_VELOCIDAD_ENCODER_1      ENCODER|0x04
#define VER_CUENTAS_ENCODER_2                      ENCODER|0x05
#define VER_VELOCIDAD_ENCODER_2                   ENCODER|0x06
#define VER_CANTIDAD_DE_GIRO_ENCODER_2            ENCODER|0x07
#define ALTERNAR_SENTIDO_CUENTA_ENCODER_2          ENCODER|0x08
#define ALTERNAR_SENTIDO_VELOCIDAD_ENCODER_2      ENCODER|0x09
#define ALTERNAR_ENCODERS                          ENCODER|0x0a
#define VER_POSICION_ENCODER_1                     ENCODER|0x0b
#define VER_POSICION_ENCODER_2                     ENCODER|0x0c
#define VER_TIEMPO_ENCODER_1                       ENCODER|0x0d
#define VER_TIEMPO_ENCODER_2                       ENCODER|0x0e
#define RESET_CUENTAS                              ENCODER|0x0f
//A CONTINUACION SE MUESTRAN LAS ORDENES PARA EL CONTROLADOR DE POSICION Y
VELOCIDAD
#define SET_PROPORCIONAL_POSICION_1                CONTROL_PYV|0X00
#define SET_DERIVATIVO_POSICION_1                  CONTROL_PYV|0X01
#define SET_PROPORCIONAL_VELOCIDAD_1               CONTROL_PYV|0X02
#define SET_INTEGRAL_VELOCIDAD_1                   CONTROL_PYV|0X03
#define SET_SETPOINT_POSICION_1                    CONTROL_PYV|0X04
#define SET_SETPOINT_VELOCIDAD_1                   CONTROL_PYV|0X05
#define SET_PROPORCIONAL_POSICION_2                CONTROL_PYV|0X06
#define SET_DERIVATIVO_POSICION_2                  CONTROL_PYV|0X07
#define SET_PROPORCIONAL_VELOCIDAD_2               CONTROL_PYV|0X08
#define SET_INTEGRAL_VELOCIDAD_2                   CONTROL_PYV|0X09
#define SET_SETPOINT_POSICION_2                    CONTROL_PYV|0X0A
#define SET_SETPOINT_VELOCIDAD_2                   CONTROL_PYV|0X0B
#define CONTROLAR_VELOCIDAD                        CONTROL_PYV|0X0C
#define CONTROLAR_POSICION                         CONTROL_PYV|0X0D
#define AUMENTAR_SETPOINT_POSICION_1               CONTROL_PYV|0X0E
#define AUMENTAR_SETPOINT_POSICION_2               CONTROL_PYV|0X0F
//A CONTINUACION SE PONE ORDENES PARA VER LOS PARAMETROS DEL CONTROLADOR
//LOS PARAMETROS RELACIONADO A LOS CONTROLADORES PD Y PI SE ENCUENTRAN
//AGRUPADOR EN CONTROL_PARAM1, MIENTRAS QUE LOS PARAMETROS RELACIONADOS
//A LOS ELEMENTOS FISICOS DEL ROBOT ESTAN EN CONTROL_PARAM2
#define VER_PROPORCIONAL_POSICION_1                CONTROL_PARAM1|0X00
#define VER_DERIVATIVO_POSICION_1                  CONTROL_PARAM1|0X01
#define VER_PROPORCIONAL_VELOCIDAD_1               CONTROL_PARAM1|0X02
#define VER_INTEGRAL_VELOCIDAD_1                   CONTROL_PARAM1|0X03
#define VER_SETPOINT_POSICION_1                    CONTROL_PARAM1|0X04
#define VER_SETPOINT_VELOCIDAD_1                   CONTROL_PARAM1|0X05
#define VER_PROPORCIONAL_POSICION_2                CONTROL_PARAM1|0X06
#define VER_DERIVATIVO_POSICION_2                  CONTROL_PARAM1|0X07
#define VER_PROPORCIONAL_VELOCIDAD_2               CONTROL_PARAM1|0X08
#define VER_INTEGRAL_VELOCIDAD_2                   CONTROL_PARAM1|0X09
#define VER_SETPOINT_POSICION_2                    CONTROL_PARAM1|0X0A
#define VER_SETPOINT_VELOCIDAD_2                   CONTROL_PARAM1|0X0B
#define VER_MODO_DEL_CONTROLADOR                   CONTROL_PARAM1|0X0C
#define VER_VELOCIDAD_1                            CONTROL_PARAM2|0X00
#define VER_NUMERO_DE_VUELTAS_1                     CONTROL_PARAM2|0X01
#define VER_TENSION_1                              CONTROL_PARAM2|0X02
#define VER_PWM_1                                   CONTROL_PARAM2|0X03
#define VER_VELOCIDAD_2                            CONTROL_PARAM2|0X04
#define VER_NUMERO_DE_VUELTAS_2                     CONTROL_PARAM2|0X05

```

```

#define VER_TENSION_2
CONTROL_PARAM2|0X06
#define VER_PWM_2
CONTROL_PARAM2|0X07
#define SET_MIN_VEL
CONTROL_PARAM2|0x08
#define SET_
#define MODO_POSICION 0
#define MODO_VELOCIDAD 1
//IDENTIFICADORES DE CADA MOTOR
#define MOTOR_DD 1
#define MOTOR_DI 2
#define MOTOR_TD 3
#define MOTOR_TI 4
//IDENTIFICADOR/DIRECCION DEL UC QUE ESTA MANEJANDO DICHO MOTOR
#define UC_MOTOR_DD 1
#define UC_MOTOR_DI 1
#define UC_MOTOR_TD 0
#define UC_MOTOR_TI 0
#define DIR_MOTOR_DD 1
#define DIR_MOTOR_DI 1
#define DIR_MOTOR_TD 0
#define DIR_MOTOR_TI 0
//NUMERO DEL MOTOR CON EL QUE ESTA IDENTIFICADO EN EL UC
#define N_MOTOR_DD 1
#define N_MOTOR_DI 2
#define N_MOTOR_TD 1
#define N_MOTOR_TI 2
//COMANDOS REFERIDOS A LA POSICION DE CADA MOTOR
#define CAMBIAR_SENTIDO_MOTOR_DD CAMBIAR_SENTIDO_MOTOR_1
#define CAMBIAR_SENTIDO_MOTOR_DI CAMBIAR_SENTIDO_MOTOR_2
#define CAMBIAR_SENTIDO_MOTOR_TD CAMBIAR_SENTIDO_MOTOR_1
#define CAMBIAR_SENTIDO_MOTOR_TI CAMBIAR_SENTIDO_MOTOR_2
//
#define CAMBIAR_VELOCIDAD_MOTOR_DD CAMBIAR_VELOCIDAD_MOTOR_1
#define CAMBIAR_VELOCIDAD_MOTOR_DI CAMBIAR_VELOCIDAD_MOTOR_2
#define CAMBIAR_VELOCIDAD_MOTOR_TD CAMBIAR_VELOCIDAD_MOTOR_1
#define CAMBIAR_VELOCIDAD_MOTOR_TI CAMBIAR_VELOCIDAD_MOTOR_2
#define PWM_RC_BT 20

```

15. ANEXO II

A continuación, se muestra el código principal del microcontrolador SMT32F407VGT

```

/* USER CODE BEGIN Header */
/**
 * *****
 * @file           : main.c
 * @brief          : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2022 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"
/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "ordenes.h"
#include "stdio.h"
#include "comandos_bt.h"
#include "conversion_array.h"
// #include "controlador_robot.h"
// #include "rs485.h"
// #include "com_bp.h"
#include "cli.h"
#include "ident_ctes.h"
#define MAX_DATA 9
#define RAMPA_PWM 5
// #define MODO_CAPTURA 10
/* 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 -----*/
SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;

```

```

TIM_HandleTypeDef htim9;
UART_HandleTypeDef huart5;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
DMA_HandleTypeDef hdma_uart5_tx;
DMA_HandleTypeDef hdma_usart3_tx;
/* USER CODE BEGIN PV */
struct rs485 comm;
struct rs485* rs = &comm;
struct com_bp bp;
struct auto_nav nav;
recorrido r;
struct cli consola;
cont_robot cont;
uint8_t buff[MAX_DATA];
uint8_t sentido;
uint8_t acum=0;
uint8_t set=0;
#define MAX_SET      7
float velocidad[4];
char comando;
uint8_t dir=0;
struct gm65 qr;
struct hcsr04 us;
IMU imu;
//
//#define MAX_BUFFER      256
float r_dd[MAX_BUFFER],r_di[MAX_BUFFER],r_td[MAX_BUFFER],r_ti[MAX_BUFFER];
float ax[MAX_BUFFER],ay[MAX_BUFFER],az[MAX_BUFFER];
float gx[MAX_BUFFER],gy[MAX_BUFFER],gz[MAX_BUFFER];
float mx[MAX_BUFFER],my[MAX_BUFFER],mz[MAX_BUFFER];
float pi[MAX_BUFFER],ro[MAX_BUFFER],ya[MAX_BUFFER];
float theta_enc[MAX_BUFFER], x_enc[MAX_BUFFER], y_enc[MAX_BUFFER];
float theta_cont[MAX_BUFFER], x_cont[MAX_BUFFER], y_cont[MAX_BUFFER];
float theta_us[MAX_BUFFER],theta_imu[MAX_BUFFER];
int espera=0;
//int n_total, n, n_datos;
/* USER CODE END PV */
/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_UART5_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM7_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM9_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code -----*/
/* USER CODE BEGIN 0 */
void enviar_y_recibir(struct rs485* rs, uint8_t orden, uint8_t dir){
    buff[0]=orden;
    rs485_transmit_master(rs, buff, MAX_DATA, dir);
}

```

```

__HAL_TIM_SET_COUNTER(&htim7,0);
HAL_TIM_Base_Start(&htim7);
uint16_t count;
while(rs->new_data==0){
    count = __HAL_TIM_GET_COUNTER(&htim7);
    if(count == 10000){
        __HAL_TIM_SET_COUNTER(&htim7,0);
        rs485_abort(rs);
        rs485_transmit_master(rs, buff, MAX_DATA, dir);
        rs485_receive(rs);
    }
    if(rs->uart->RxState==HAL_UART_STATE_READY){
        rs485_receive(rs);
    }
}
HAL_TIM_Base_Stop(&htim7);
}

void comando_bt(char comando){
    if(cli_cargar_caracter(&consola, comando)==CHAR_CARGADO) return;
    switch(comando){
        case ADELANTE_bt:
            velocidad[MOTOR_DD]=VEL_MAX;
            velocidad[MOTOR_DI]=VEL_MAX;
            velocidad[MOTOR_TD]=VEL_MAX;
            velocidad[MOTOR_TI]=VEL_MAX;
            break;
        case ATRAS_bt:
            velocidad[MOTOR_DD]=-VEL_MAX;
            velocidad[MOTOR_DI]=-VEL_MAX;
            velocidad[MOTOR_TD]=-VEL_MAX;
            velocidad[MOTOR_TI]=-VEL_MAX;
            break;
        case IR_A_DERECHA:
            velocidad[MOTOR_DD]=VEL_MAX;
            velocidad[MOTOR_DI]=-VEL_MAX;
            velocidad[MOTOR_TD]=-VEL_MAX;
            velocidad[MOTOR_TI]=VEL_MAX;
            break;
        case IR_A_IZQUIERDA:
            velocidad[MOTOR_DD]=-VEL_MAX;
            velocidad[MOTOR_DI]=VEL_MAX;
            velocidad[MOTOR_TD]=VEL_MAX;
            velocidad[MOTOR_TI]=-VEL_MAX;
            break;
        case GIRAR_A_DERECHA:
            velocidad[MOTOR_DD]=VEL_MAX;
            velocidad[MOTOR_DI]=-VEL_MAX;
            velocidad[MOTOR_TD]=VEL_MAX;
            velocidad[MOTOR_TI]=-VEL_MAX;
            break;
        case GIRAR_A_IZQUIERDA:
            velocidad[MOTOR_DD]=-VEL_MAX;
            velocidad[MOTOR_DI]=VEL_MAX;
            velocidad[MOTOR_TD]=-VEL_MAX;
            velocidad[MOTOR_TI]=VEL_MAX;
            break;
        case PARAR:
            velocidad[MOTOR_DD]=0;
            velocidad[MOTOR_DI]=0;

```

```

        velocidad[MOTOR_TD]=0;
        velocidad[MOTOR_TI]=0;
        break;
    }
}

void cambiar_sentido(uint8_t motor){
    switch(motor){
        case MOTOR_DI:
            com_bp_enviar(&bp, CAMBIAR_SENTIDO_MOTOR_DI, UC_MOTOR_DI);
            rs->new_data=0;
            break;
        case MOTOR_DD:
            com_bp_enviar(&bp, CAMBIAR_SENTIDO_MOTOR_DD, UC_MOTOR_DD);
            rs->new_data=0;
            break;
        // case MOTOR_TD:
        //     com_bp_enviar(&bp, CAMBIAR_SENTIDO_MOTOR_TD, UC_MOTOR_TD);
        //     rs->new_data=0;
        //     break;
        // case MOTOR_TI:
        //     com_bp_enviar(&bp, CAMBIAR_SENTIDO_MOTOR_TI, UC_MOTOR_TI);
        //     rs->new_data=0;
        //     break;
    }
}

void cambiar_vel(uint8_t motor, float vel){
    float_to_array(vel, bp.buff+1);
    switch(motor){
        case MOTOR_DI:
            com_bp_enviar(&bp, SET_SETPOINT_VELOCIDAD_2, UC_MOTOR_DI);
            rs->new_data=0;
            break;
        case MOTOR_DD:
            com_bp_enviar(&bp, SET_SETPOINT_VELOCIDAD_1, UC_MOTOR_DD);
            rs->new_data=0;
            break;
        case MOTOR_TD:
            com_bp_enviar(&bp, SET_SETPOINT_VELOCIDAD_1, UC_MOTOR_TD);
            rs->new_data=0;
            break;
        case MOTOR_TI:
            com_bp_enviar(&bp, SET_SETPOINT_VELOCIDAD_2, UC_MOTOR_TI);
            rs->new_data=0;
            break;
    }
}

void actualizar_vel(){
    for(int i=1; i<5;i++){
        cambiar_vel(i, velocidad[i]);
    }
}

#define VERIFICAR(s,c)    s=c; if(s==COMANDO_ERROR) return COMANDO_ERROR
#define VERIFICAR_COMM(s,c)    s=c; if(s!=COMANDO_OK) return COMANDO_ERROR_COMM
#define ENVIAR_TEXTO(longitud)    HAL_UART_Transmit_DMA(com->uart, (uint8_t*) com->texto, longitud);
int ident_tomar_datos(struct cli *com){
    // int status;
    // status=com_bp_ver_vuelta(com->bp, MOTOR_DD, r_dd+com->n);
    // status=com_bp_ver_vuelta(com->bp, MOTOR_DI, r_di+com->n);

```

```

// status=com_bp_ver_vuelta(com->bp, MOTOR_TD, r_td+com->n);
// status=com_bp_ver_vuelta(com->bp, MOTOR_TI, r_ti+com->n);
r_dd[com->n]=com->robot->cuentas_output[INDEX_MOTOR_DD];
r_di[com->n]=com->robot->cuentas_output[INDEX_MOTOR_DI];
r_td[com->n]=com->robot->cuentas_output[INDEX_MOTOR_TD];
r_ti[com->n]=com->robot->cuentas_output[INDEX_MOTOR_TI];
theta_enc[com->n]=com->robot->posicion_output_enc[INDEX_THETA];
y_enc[com->n]=com->robot->posicion_output_enc[INDEX_Y];
x_enc[com->n]=com->robot->posicion_output_enc[INDEX_X];
theta_cont[com->n]=com->robot->posicion_output[INDEX_THETA];
y_cont[com->n]=com->robot->posicion_output[INDEX_Y];
x_cont[com->n]=com->robot->posicion_output[INDEX_X];
theta_us[com->n]=com->robot->angulo_us;
theta_imu[com->n]=com->robot->angulo_imu;
ax[com->n]=com->robot->imu->a_f[0];
ay[com->n]=com->robot->imu->a_f[1];
az[com->n]=com->robot->imu->a_f[2];
gx[com->n]=com->robot->imu->g_f[0];
gy[com->n]=com->robot->imu->g_f[1];
gz[com->n]=com->robot->imu->g_f[2];
mx[com->n]=com->robot->imu->m_f[0];
my[com->n]=com->robot->imu->m_f[1];
mz[com->n]=com->robot->imu->m_f[2];
pi[com->n]=com->robot->imu->pitch;
ro[com->n]=com->robot->imu->roll;
ya[com->n]=com->robot->imu->yaw;
int l=0;
// l=sprintf(com->texto,"%f,%f,%f,%f\n",r_dd[com->n],r_di[com->n],r_td[com->n],r_ti[com->n]);
com->n++;
if(com->n>=com->n_total){
    l=sprintf(com->texto+l,"\nTermino la captura de datos\n\n");
    com_bp_move(com->bp, com->iden_cont, com->iden_dir, 0);
    com->identificacion=0;
}
ENVIAR_TEXTO(1);
return COMANDO_OK;
}

int ident_dar_datos(struct cli *com){
    int l2;
    if(com->n_datos>=com->n_total){
        l2=sprintf(com->texto,"Se termino el envio de datos\n\n");
        ENVIAR_TEXTO(l2);
        com->iden_datos=0;
    }
    switch(com->iden_datos){
        case DATOS_GYRO:
            l2=sprintf(com->texto,"%f,%f,%f\n", gx[com->n_datos],gy[com->n_datos],gz[com->n_datos]);
            ENVIAR_TEXTO(l2);
            break;
        case DATOS_MAG:
            l2=sprintf(com->texto,"%f,%f,%f\n", mx[com->n_datos],my[com->n_datos],mz[com->n_datos]);
            ENVIAR_TEXTO(l2);
            break;
        case DATOS_ACCEL:
            l2=sprintf(com->texto,"%f,%f,%f\n", ax[com->n_datos],ay[com->n_datos],az[com->n_datos]);

```

```

        ENVIAR_TEXTO(l2);
        break;
    case DATOS_IMU:
        l2=sprintf(com->texto,"%0.3f,%0.3f,%0.3f\n", pi[com->n_datos],ro[com-
>n_datos],ya[com->n_datos]);
        ENVIAR_TEXTO(l2);
        break;
    case DATOS_ENC:
        l2=sprintf(com->texto,"%0.3f,%0.3f,%0.3f,%0.3f\n", r_dd[com-
>n_datos],r_di[com->n_datos],r_td[com->n_datos],r_ti[com->n_datos]);
        ENVIAR_TEXTO(l2);
        break;
    case DATOS_CONT_ENC://posicionamiento por encoder
        l2=sprintf(com->texto,"%0.3f,%0.3f,%0.3f\n", theta_enc[com-
>n_datos],x_enc[com->n_datos],y_enc[com->n_datos]);
        ENVIAR_TEXTO(l2);
        break;
    case DATOS_CONT://posicionamiento por encoder
        l2=sprintf(com->texto,"%0.3f,%0.3f,%0.3f\n", theta_cont[com-
>n_datos],x_cont[com->n_datos],y_cont[com->n_datos]);
        ENVIAR_TEXTO(l2);
        break;
    case DATOS_KAL_THETA://posicionamiento por encoder
        l2=sprintf(com->texto,"%0.3f,%0.3f,%0.3f,%0.3f\n", theta_cont[com-
>n_datos],theta_enc[com->n_datos],theta_imu[com->n_datos],theta_us[com->n_datos]);
        ENVIAR_TEXTO(l2);
        break;
    default:
        return COMANDO_ERROR;
        break;
}
com->n_datos++;
return COMANDO_ERROR;
}
int valor_mayor(float* a,int largo){
    int i_out=0;
    for(int i=0;i<largo;i++){
        if(*(a+i_out)<*(a+i)){
            i_out=i;
        }
    }
    return i_out;
}
int valor_menor(float* a,int largo){
    int i_out=0;
    for(int i=0;i<largo;i++){
        if(*(a+i_out)>*(a+i)){
            i_out=i;
        }
    }
    return i_out;
}
int ident_cal(struct cli* com){
    float offset[3];
    if(com->iden_datos==POSPROC_GYRO){
        offset[0]=0.0;
        offset[1]=0.0;
        offset[2]=0.0;
        for(int i=0;i<com->n_total;i++){

```

```

        offset[0]+=(gx[i]/(com->n_total*1.0));
        offset[1]+=(gy[i]/(com->n_total*1.0));
        offset[2]+=(gz[i]/(com->n_total*1.0));
    }
    offset[0]=-com->robot->imu->g_offset[0];
    offset[1]=-com->robot->imu->g_offset[1];
    offset[2]=-com->robot->imu->g_offset[2];
    int l=sprintf(com->texto,"offset en gyro\n gx=%.3f gy=%.3f
gz=%.3f\n>", offset[0], offset[1], offset[2]);
    ENVIAR_TEXTO(1);
}
else if(com->iden_datos==POSPROC_MAG){
    float x_h,x_l,y_h,y_l,z_h,z_l;
    int largo=com->n_total;
    x_h=mx[valor_mayor(mx, largo)];//-com->robot->imu->m_offset[0];
    x_l=mx[valor_menor(mx, largo)];//-com->robot->imu->m_offset[0];
    y_h=my[valor_mayor(my, largo)];//-com->robot->imu->m_offset[1];
    y_l=my[valor_menor(my, largo)];//-com->robot->imu->m_offset[1];
    z_h=mz[valor_mayor(mz, largo)];//-com->robot->imu->m_offset[2];
    z_l=mz[valor_menor(mz, largo)];//-com->robot->imu->m_offset[2];
    offset[0]=(x_h+x_l)/2.0-com->robot->imu->m_offset[0];
    offset[1]=(y_h+y_l)/2.0-com->robot->imu->m_offset[1];
    offset[2]=(z_h+z_l)/2.0-com->robot->imu->m_offset[2];
    int l2=sprintf(com->texto,"offset en mag\n mx=%.3f my=%.3f
mz=%.3f\n>", offset[0], offset[1], offset[2]);
    ENVIAR_TEXTO(12);
}
com->iden_datos=SIN_TAREA;
return COMANDO_OK;
}
#define INICIO_CAL 0
#define GIRO_GD 1
#define CAMBIO_SENTIDO 2
#define GIRO_GI 3
#define FIN_CAL 4
#define PARAR_CAL 5
#define TIEMPO_GIRANDO 10.0
#define REP (TIEMPO_GIRANDO/TS)
//int estado_cal=PARAR_CAL;
int i=0;
float offset_gd[3], offset_gi[3];
float mx_h,mx_l,my_h,my_l,mz_h,mz_l;
void calibracion(struct cli* com){
    float magx,magy,magz;
    int l;
    // float x_h,x_l,y_h,y_l,z_h,z_l;
    switch(com->estado_cal){
        case PARAR_CAL:
            return;
            break;
        case INICIO_CAL:
            com_bp_move(&bp, CONT_VELOCIDAD, GIRAR_DER, 1);
            mx_h=com->robot->imu->m_f[0];
            mx_l=mx_h;
            my_h=com->robot->imu->m_f[1];
            my_l=my_h;
            mz_h=com->robot->imu->m_f[2];
            mz_l=mz_h;
            com->estado_cal=GIRO_GD;

```

```

        i=0;
        break;
    case GIRO_GD:
        magx=com->robot->imu->m_f[0];
        magy=com->robot->imu->m_f[1];
        magz=com->robot->imu->m_f[2];
        if(mx_h<magx) mx_h=magx;
        if(my_h<magy) my_h=magy;
        if(mz_h<magz) mz_h=magz;
        if(mx_l>magx) mx_l=magx;
        if(my_l>magy) my_l=magy;
        if(mz_l>magz) mz_l=magz;
        i++;
        if(i>=REP){
            com->estado_cal=CAMBIO_SENTIDO;
            i=0;
        }
        com_bp_moverse(&bp, CONT_VELOCIDAD, GIRAR_DER, 1);
        break;
    case CAMBIO_SENTIDO:
        i++;
        if(i>=(1/TS)){
            offset_gd[0]=(mx_h+mx_l)/2.0 - com->robot->imu->m_offset[0];
            offset_gd[1]=(my_h+my_l)/2.0 - com->robot->imu->m_offset[1];
            offset_gd[2]=(mz_h+mz_l)/2.0 - com->robot->imu->m_offset[2];
            com->estado_cal=GIRO_GI;
            i=0;
            mx_h=com->robot->imu->m_f[0];
            mx_l=mx_h;
            my_h=com->robot->imu->m_f[1];
            my_l=my_h;
            mz_h=com->robot->imu->m_f[2];
            mz_l=mz_h;
            com_bp_moverse(&bp, CONT_VELOCIDAD, GIRAR_IZQ, 1);
        }
        else com_bp_moverse(&bp, CONT_VELOCIDAD, GIRAR_IZQ, 0);
        break;
    case GIRO_GI:
        magx=com->robot->imu->m_f[0];
        magy=com->robot->imu->m_f[1];
        magz=com->robot->imu->m_f[2];
        if(mx_h<magx) mx_h=magx;
        if(my_h<magy) my_h=magy;
        if(mz_h<magz) mz_h=magz;
        if(mx_l>magx) mx_l=magx;
        if(my_l>magy) my_l=magy;
        if(mz_l>magz) mz_l=magz;
        i++;
        if(i>=REP){
            com->estado_cal=FIN_CAL;
            i=0;
        }
        com_bp_moverse(&bp, CONT_VELOCIDAD, GIRAR_IZQ, 1);
        break;
    case FIN_CAL:
        offset_gi[0]=(mx_h+mx_l)/2.0 - com->robot->imu->m_offset[0];
        offset_gi[1]=(my_h+my_l)/2.0 - com->robot->imu->m_offset[1];
        offset_gi[2]=(mz_h+mz_l)/2.0 - com->robot->imu->m_offset[2];
        com->estado_cal=PARAR_CAL;

```

```

        i=0;
        mx_h=com->robot->imu->m_f[0];
        mx_l=mx_h;
        my_h=com->robot->imu->m_f[1];
        my_l=my_h;
        mz_h=com->robot->imu->m_f[2];
        mz_l=mz_h;
        com_bp_moverse(&bp, CONT_VELOCIDAD, GIRAR_IZQ, 0);
        l=sprintf(com->texto, "La calibracion ha terminado\nLos valores
antiguos de offset so:\nmx=%.3f my=%.3f mz=%.3f\n", com->robot->imu-
>m_offset[0], com->robot->imu->m_offset[1], com->robot->imu->m_offset[2]);
        com->robot->imu->m_offset[0]=-(offset_gd[0]+offset_gi[0])/2.0;
        com->robot->imu->m_offset[1]=-(offset_gd[1]+offset_gi[1])/2.0;
        com->robot->imu->m_offset[2]=-(offset_gd[2]+offset_gi[2])/2.0;
        l+=sprintf(com->texto+l, "Los valores nuevos de offset son:\nmx=%.3f
my=%.3f mz=%.3f\n", com->robot->imu->m_offset[0], com->robot->imu->m_offset[1], com-
>robot->imu->m_offset[2]);
        ENVIAR_TEXTO(1);
        com->calibracion=0;
        break;
    }
    return;
}

void qr_escaneo(struct cli* com){
    // int j=0;
    // while(com->uart->gState==HAL_UART_STATE_BUSY_TX || com->uart-
>gState==HAL_UART_STATE_BUSY){
    //     j++;
    //     if(j==0) break;
    // }
    // j=0;
    if(com->uart->gState==HAL_UART_STATE_BUSY_TX || com->uart-
>gState==HAL_UART_STATE_BUSY){return;}
    int l;
    if(SCAN_EXITOSO(com->qr)){
        if(SCAN_QR_EXITOSO(com->qr)){
            l=sprintf(com->texto, "Se escaneo un codigo QR, contiene:\n");
            int l2=gm65_extraer_scan(com->qr, &(com->texto[l]));
            l=l+l2;
            com->texto[l]='\n';
            com->texto[l+1]='>';
            l+=2;
            ENVIAR_TEXTO(1);
        }
        else{
            l=sprintf(com->texto, "No se escaneo un codigo QR\n\n");
            ENVIAR_TEXTO(1);
        }
        RESET_SCAN(com->qr);
        com->scaneando_qr=DESACTIVADO;
    }
    else if(com->scaneando_qr==ACTIVADO && com->qr->lectura_qr==FALLO_QR){
        l=sprintf(com->texto, "No se pudo scanear\n\n");
        ENVIAR_TEXTO(1);
        com->scaneando_qr=DESACTIVADO;
    }
}

/* USER CODE END 0 */
/**

```

```

* @brief The application entry point.
* @retval int
*/
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* 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_DMA_Init();
    MX_UART5_Init();
    MX_USART3_UART_Init();
    MX_TIM7_Init();
    MX_USART2_UART_Init();
    MX_TIM9_Init();
    MX_TIM1_Init();
    MX_TIM2_Init();
    MX_SPI1_Init();
    MX_SPI2_Init();
    MX_TIM6_Init();
    /* USER CODE BEGIN 2 */
    rs485_assign_pint(rs, rs485_pin_GPIO_Port, rs485_pin_Pin);
    rs485_init(rs, &huart5, MASTER);
    com_bp_init(&bp, rs, &htim7);
    cli_init(&consola, &huart3, &bp);
    auto_nav_init(&nav, &qr, &r);
    controlador_robot_init(&cont, &bp);
    imu_init(&imu, &hspi2, GPIOC, GPIO_PIN_4, 0.010);
    nav.bp=&bp;
    nav.uart=consola.uart;
    cont.imu=&imu;
    cont.us=&us;
    HAL_UART_Receive_IT(&huart3, (uint8_t*)&comando, 1);
    com_bp_enviar(&bp, CONTROLAR_VELOCIDAD, UC_MOTOR_TD);
    com_bp_enviar(&bp, CONTROLAR_VELOCIDAD, UC_MOTOR_DD);
    com_bp_reset_encoder(&bp);
    gm65_init(&qr, &huart2, &htim9);
    ultrasonic_init(&htim1, &htim2, TIM_CHANNEL_1, TIM_CHANNEL_4);
    consola.qr=&qr;
    consola.us=&us;
    consola.nav=&nav;
    consola.robot=&cont;
    nav.robot=&cont;
    HAL_TIM_Base_Start_IT(&htim6);
    // int16_t giro1_a, giro2_a;
    // uint32_t t1,t2;
    // int giro1,giro2;
    uint32_t tick;
    //uint16_t count;
    // pwm=0;

```

```

// pwm_ant = 0;
tick = HAL_GetTick();
// int16_t cuenta1,cuenta2;
// int32_t pos1,pos2;
// char texto[128];
// uint8_t l=0;
// l=sprintf(texto,"pwm_set,pwm,pos_motor1,pos_motor2, t1, t2\n");
// HAL_UART_Transmit_DMA(&huart3, (uint8_t*)texto, l);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
tick = HAL_GetTick();
int ts_ms=1000*TS;
// int tick_imu;
// int tick_f;
// float v;
while (1)
{
    if(HAL_GetTick()>=tick + ts_ms){
        tick = HAL_GetTick();
        HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
//        imu_update(&imu);
//        tick_imu=HAL_GetTick();
        if(consola.enable==ACTIVADO){
            controlador_robot_update(&cont);
            if(consola.identificacion==1){
                com_bp_moverse(&bp, consola.iden_cont,
consola.iden_dir, consola.iden_sp);
                ident_tomar_datos(&consola);
            }
            else{
//                com_bp_ver_vuelta(&bp, MOTOR_TI, &v);
                if((consola.iden_datos&MASK_DATOS)==DAR_DATOS){
                    espera++;
                    if(espera>=(0.5/TS)){
                        espera=0;
                        ident_dar_datos(&consola);
                    }
                }
                else if((consola.iden_datos&MASK_DATOS)==POSPROC){
                    ident_cal(&consola);
                }
                else if(consola.calibracion==1){
//                    estado_cal=INICIO_CAL;
                    calibracion(&consola);
                }
            }
        }
        if(consola.scaneando_qr==ACTIVADO) qr_escaneo(&consola);
        auto_nav_handle(&nav);
//        com_bp_test(&bp);
    }
    else{
        com_bp_enviar(&bp, CONTROLAR_VELOCIDAD, UC_MOTOR_TD);
        com_bp_enviar(&bp, CONTROLAR_VELOCIDAD, UC_MOTOR_DD);
        actualizar_vel();
    }
    tick_f=HAL_GetTick();
}
cli_handler(&consola);

```

```

//
//      if(huart3.RxState!=HAL_UART_STATE_READY){
//          HAL_UART_Receive_IT(&huart3, (uint8_t*)&comando, 1);
//      }
//  }
//  /* USER CODE END WHILE */
//  /* USER CODE BEGIN 3 */
//  /* 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
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 8;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    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_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    {
        Error_Handler();
    }
}
/**
 * @brief SPI1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI1_Init(void)
{
    /* USER CODE BEGIN SPI1_Init 0 */
    /* USER CODE END SPI1_Init 0 */
    /* USER CODE BEGIN SPI1_Init 1 */

```

```

/* USER CODE END SPI1_Init 1 */
/* SPI1 parameter configuration*/
hspi1.Instance = SPI1;
hspi1.Init.Mode = SPI_MODE_MASTER;
hspi1.Init.Direction = SPI_DIRECTION_2LINES;
hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
hspi1.Init.NSS = SPI_NSS_SOFT;
hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
hspi1.Init.CRCPolynomial = 10;
if (HAL_SPI_Init(&hspi1) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN SPI1_Init 2 */
/* USER CODE END SPI1_Init 2 */
}
/**
 * @brief SPI2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI2_Init(void)
{
    /* USER CODE BEGIN SPI2_Init 0 */
    /* USER CODE END SPI2_Init 0 */
    /* USER CODE BEGIN SPI2_Init 1 */
    /* USER CODE END SPI2_Init 1 */
    /* SPI2 parameter configuration*/
    hspi2.Instance = SPI2;
    hspi2.Init.Mode = SPI_MODE_MASTER;
    hspi2.Init.Direction = SPI_DIRECTION_2LINES;
    hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
    hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;
    hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
    hspi2.Init.NSS = SPI_NSS_SOFT;
    hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
    hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
    hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
    hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
    hspi2.Init.CRCPolynomial = 10;
    if (HAL_SPI_Init(&hspi2) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN SPI2_Init 2 */
    /* USER CODE END SPI2_Init 2 */
}
/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{

```

```

/* USER CODE BEGIN TIM1_Init 0 */
/* USER CODE END TIM1_Init 0 */
TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_IC_InitTypeDef sConfigIC = {0};
/* USER CODE BEGIN TIM1_Init 1 */
/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = 167;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 65535;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 0;
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
/* USER CODE END TIM1_Init 2 */
}
/**
 * @brief TIM2 Initialization Function
 * @param None

```

```

* @retval None
*/
static void MX_TIM2_Init(void)
{
    /* USER CODE BEGIN TIM2_Init 0 */
    /* USER CODE END TIM2_Init 0 */
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    /* USER CODE BEGIN TIM2_Init 1 */
    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 84-1;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 100000-1;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = 10;
    sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM2_Init 2 */
    /* USER CODE END TIM2_Init 2 */
    HAL_TIM_MspPostInit(&htim2);
}

/**
 * @brief TIM6 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM6_Init(void)
{

```

```

/* USER CODE BEGIN TIM6_Init 0 */
/* USER CODE END TIM6_Init 0 */
TIM_MasterConfigTypeDef sMasterConfig = {0};
/* USER CODE BEGIN TIM6_Init 1 */
/* USER CODE END TIM6_Init 1 */
htim6.Instance = TIM6;
htim6.Init.Prescaler = 83;
htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
htim6.Init.Period = 10000;
htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM6_Init 2 */
/* USER CODE END TIM6_Init 2 */
}
/**
 * @brief TIM7 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM7_Init(void)
{
    /* USER CODE BEGIN TIM7_Init 0 */
    /* USER CODE END TIM7_Init 0 */
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    /* USER CODE BEGIN TIM7_Init 1 */
    /* USER CODE END TIM7_Init 1 */
    htim7.Instance = TIM7;
    htim7.Init.Prescaler = 83;
    htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim7.Init.Period = 65535;
    htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM7_Init 2 */
    /* USER CODE END TIM7_Init 2 */
}
/**
 * @brief TIM9 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM9_Init(void)

```

```

{
    /* USER CODE BEGIN TIM9_Init 0 */
    /* USER CODE END TIM9_Init 0 */
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    /* USER CODE BEGIN TIM9_Init 1 */
    /* USER CODE END TIM9_Init 1 */
    htim9.Instance = TIM9;
    htim9.Init.Prescaler = 16799;
    htim9.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim9.Init.Period = 4000-1;
    htim9.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim9.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim9) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim9, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM9_Init 2 */
    /* USER CODE END TIM9_Init 2 */
}
/**
 * @brief UART5 Initialization Function
 * @param None
 * @retval None
 */
static void MX_UART5_Init(void)
{
    /* USER CODE BEGIN UART5_Init 0 */
    /* USER CODE END UART5_Init 0 */
    /* USER CODE BEGIN UART5_Init 1 */
    /* USER CODE END UART5_Init 1 */
    huart5.Instance = UART5;
    huart5.Init.BaudRate = 115200;
    huart5.Init.WordLength = UART_WORDLENGTH_9B;
    huart5.Init.StopBits = UART_STOPBITS_1;
    huart5.Init.Parity = UART_PARITY_NONE;
    huart5.Init.Mode = UART_MODE_TX_RX;
    huart5.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart5.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_MultiProcessor_Init(&huart5, 0, UART_WAKEUPMETHOD_IDLELINE) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN UART5_Init 2 */
    /* USER CODE END UART5_Init 2 */
}
/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{
    /* USER CODE BEGIN USART2_Init 0 */
    /* USER CODE END USART2_Init 0 */

```

```

/* USER CODE BEGIN USART2_Init 1 */
/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 9600;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */
/* USER CODE END USART2_Init 2 */
}
/**
 * @brief USART3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART3_UART_Init(void)
{
    /* USER CODE BEGIN USART3_Init 0 */
    /* USER CODE END USART3_Init 0 */
    /* USER CODE BEGIN USART3_Init 1 */
    /* USER CODE END USART3_Init 1 */
    huart3.Instance = USART3;
    huart3.Init.BaudRate = 9600;
    huart3.Init.WordLength = UART_WORDLENGTH_8B;
    huart3.Init.StopBits = UART_STOPBITS_1;
    huart3.Init.Parity = UART_PARITY_NONE;
    huart3.Init.Mode = UART_MODE_TX_RX;
    huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart3.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart3) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART3_Init 2 */
    /* USER CODE END USART3_Init 2 */
}
/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{
    /* DMA controller clock enable */
    __HAL_RCC_DMA1_CLK_ENABLE();
    /* DMA interrupt init */
    /* DMA1_Stream3_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream3_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream3_IRQn);
    /* DMA1_Stream7_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream7_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream7_IRQn);
}
/**

```

```

* @brief GPIO Initialization Function
* @param None
* @retval None
*/
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(CS_I2C_SPI_GPIO_Port, CS_I2C_SPI_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port, OTG_FS_PowerSwitchOn_Pin,
GPIO_PIN_SET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
|Audio_RST_Pin|rs485_pin_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pin : CS_I2C_SPI_Pin */
    GPIO_InitStruct.Pin = CS_I2C_SPI_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(CS_I2C_SPI_GPIO_Port, &GPIO_InitStruct);
    /*Configure GPIO pins : OTG_FS_PowerSwitchOn_Pin PC4 */
    GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin|GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    /*Configure GPIO pin : PDM_OUT_Pin */
    GPIO_InitStruct.Pin = PDM_OUT_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
    HAL_GPIO_Init(PDM_OUT_GPIO_Port, &GPIO_InitStruct);
    /*Configure GPIO pin : B1_Pin */
    GPIO_InitStruct.Pin = B1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
    /*Configure GPIO pin : I2S3_WS_Pin */
    GPIO_InitStruct.Pin = I2S3_WS_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
    HAL_GPIO_Init(I2S3_WS_GPIO_Port, &GPIO_InitStruct);

```

```

/*Configure GPIO pin : BOOT1_Pin */
GPIO_InitStruct.Pin = BOOT1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pin : CLK_IN_Pin */
GPIO_InitStruct.Pin = CLK_IN_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
HAL_GPIO_Init(CLK_IN_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pin : PB12 */
GPIO_InitStruct.Pin = GPIO_PIN_12;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
                        Audio_RST_Pin rs485_pin_Pin */
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                    |Audio_RST_Pin|rs485_pin_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
/*Configure GPIO pins : I2S3_MCK_Pin I2S3_SCK_Pin */
GPIO_InitStruct.Pin = I2S3_MCK_Pin|I2S3_SCK_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
/*Configure GPIO pin : VBUS_FS_Pin */
GPIO_InitStruct.Pin = VBUS_FS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(VBUS_FS_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pins : OTG_FS_ID_Pin OTG_FS_DM_Pin OTG_FS_DP_Pin */
GPIO_InitStruct.Pin = OTG_FS_ID_Pin|OTG_FS_DM_Pin|OTG_FS_DP_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF10_OTG_FS;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
/*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pins : Audio_SCL_Pin Audio_SDA_Pin */
GPIO_InitStruct.Pin = Audio_SCL_Pin|Audio_SDA_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/*Configure GPIO pin : MEMS_INT2_Pin */
GPIO_InitStruct.Pin = MEMS_INT2_Pin;

```

```

GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart){
    if(huart->Instance==UART5){
        rs485_uart_rx_handle(rs);
    }
    else if(huart->Instance==USART3){
        comando_bt(comando);
        HAL_UART_Receive_IT(&huart3, (uint8_t*)&comando, 1);
    }
    else if(huart->Instance==USART2){
        gm65_uart_RX_handler(&qr);//, &huart2, &htim9);
    }
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef* huart){
    if(huart->Instance==UART5){
        rs485_uart_tx_handle(rs);
    }
    else if(huart->Instance==USART2){
        gm65_uart_TX_handler(&qr);//, &huart2, &htim9);
    }
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef* huart){
    //HAL_UART_Receive_it(&huart3, (uint8_t*)&comando, 1);
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim){
    if(htim->Instance==TIM9){
        gm65_Tim_handler(&qr);//, &huart2, &htim9);
    }
    else if(htim->Instance==TIM2){
        ultrasonic_tim_handler(&us, &htim1);
    }
    else if(htim->Instance==TIM6){
//        int t=__HAL_TIM_GET_COUNTER(htim);
//        HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, SET);
//        imu_update(&imu);
//        HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, RESET);
//        int t2=__HAL_TIM_GET_COUNTER(htim);
//        int t3=__HAL_TIM_GET_COUNTER(htim);
    }
}
/* 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 */

```

A continuación, se muestra el código de la librería del GM65.

```

/*
 * gm65.h
 *
 * Created on: 16 dic. 2021
 * Author: Gonzalo Mansilla
 */
#ifndef INC_GM65_H_
#define INC_GM65_H_
#include "stm32f4xx_hal.h"
#define QR_START 0x00 //QR configurado listo para
operar
#define QR_SETUP_OFF 0x10 //--Estado inicial, se manda conf
para deshabilitar introducir configuracion por qr
#define QR_SETUP_OFF_T 0x11 //Se envia la configuracion de
rechazar configuracion por QR
#define QR_SETUP_OFF_R 0x12 //Se espera la confirmacion del qr del
comando enviado
#define QR_SETUP_NO_LIGHT_T 0x13
#define QR_SETUP_NO_LIGHT_R 0x14
#define QR_IDLE 0x20
#define QR_SCAN 0x30
#define QR_SCAN_T 0x31
#define QR_SCAN_RC 0x32
#define QR_SCAN_RH 0x33
#define QR_SCAN_RD 0x34
#define QR_SCAN_SETUP_OFF 0x40
#define QR_SCAN_SETUP_OFF_T 0x41
#define QR_SCAN_SETUP_OFF_R 0x42
#define QR_SCAN_NO_LIGHT_T 0x43
#define QR_SCAN_NO_LIGHT_R 0x44
#define QR_ABORT 0x50
#define QR_RECEPTION_UNKNOWN 0x60
#define QR_LENGTH_COMMAND 9
#define QR_LENGTH_RESPOND_COMMAND 7
#define QR_LENGTH_HEAD 3
#define QR_COMMAND_WAIT 5000
#define QR_SCAN_WAIT 55000
#define REINTENTOS_MAXIMOS-1//se escanea hasta detectar el codigo qr;

```

```

#define SIN_REINTENTOS      0
#define REINTENTAR          1
#define IDLE_QR              0
#define FALLO_QR            1
#define LECTURA_OK_QR      2
#define SCANEANDO_QR        3
struct gm65{
    uint8_t qrBuffer[256];
    uint8_t qrLengthDecode;
    uint8_t qrStatus;
    uint8_t qrDataReady;
    uint8_t reintentos;
    uint8_t reintentar;
    int reintentos_max;
    int lectura_qr;
    TIM_HandleTypeDef *htim;
    UART_HandleTypeDef *huart;
};
void gm65_init( struct gm65 *myQR, UART_HandleTypeDef *huart, TIM_HandleTypeDef
*htim); //prepara el modulo para trabajar con el dato
void gm65_uart_TX_handler(struct gm65 *myQR);//, UART_HandleTypeDef *huart,
TIM_HandleTypeDef *htim);
void gm65_uart_RX_handler(struct gm65 *myQR);//, UART_HandleTypeDef *huart,
TIM_HandleTypeDef *htim);
void gm65_Tim_handler(struct gm65 *myQR);//, UART_HandleTypeDef *huart,
TIM_HandleTypeDef *htim);
void gm65_scan(struct gm65 *myQR);//, UART_HandleTypeDef *huart);
void gm65_abort(struct gm65 *myQR);//, UART_HandleTypeDef *huart,
TIM_HandleTypeDef *htim);
int gm65_extraer_scan(struct gm65 *myQR, char* string);
uint8_t qrRespuestaValida(uint8_t *respuesta);
uint8_t qrRespSetupOnCode(uint8_t *code);
uint16_t gm65_crc(uint8_t* ptr, unsigned int len);
#define REINTENTOS_SCAN(gm65, intentos)      gm65->reintentos_max=intentos;
#define SCAN_EXITOSO(gm65)                    gm65->qrDataReady==1
#define RESET_SCAN(gm65)                     gm65->qrDataReady=0
#define EXTRAER_SCAN(gm65, string, largo)    string=(char)(gm65-
>qrBuffer+1);largo=gm65->qrLengthDecode-2;gm65->qrDataReady=0
#define SCAN_QR_EXITOSO(gm65)                 (SCAN_EXITOSO(gm65))&& (gm65-
>qrBuffer[0]==0x51) && (gm65->qrBuffer[gm65->qrLengthDecode-1]==0x0D)
#define SCANEANDO(gm65)                      ((gm65-
>qrStatus)&0xF0)==QR_SCAN
#endif /* INC_GM65_H_ */
/*
 * gm65.c
 *
 * Created on: 16 dic. 2021
 * Author: Gonzalo Mansilla
 */
#include "gm65.h"
#include "stm32f4xx_hal.h"
/*
#define QR_START              0
#define QR_SETUP_OFF         1
#define QR_SETUP_OFF_T       11
#define QR_SETUP_OFF_R       12
#define QR_IDLE               2
#define QR_SCAN               3
#define QR_SCAN_T             31

```

```

#define QR_SCAN_RC 32
#define QR_SCAN_RH 33
#define QR_SCAN_RD 34
#define QR_SCAN_SETUP_OFF 4
#define QR_SCAN_SETUP_OFF_T 41
#define QR_SCAN_SETUP_OFF_R 42
#define QR_ABORT 5
#define QR_LENGTH_COMMAND 9
#define QR_LENGTH_RESPOND_COMMAND 7
#define QR_LENGTH_HEAD 3
#define QR_COMMAND_WAIT 5000
#define QR_SCAN_WAIT 55000
struct gm65{
    uint8_t qrBuffer[256];
    uint8_t qrLengthDecode;
    uint8_t qrStatus;
    uint8_t qrDataReady;
};
*/
uint8_t qrSetupOFF[9]={ 0x7E, 0,
                        0x08,
                        0x01,
                        0x00, 0x03,
                        0x03,
                        0x11, 0xA9};

uint8_t qrLight_OFF[9]={ 0x7E, 0,
                        0x08,
                        0x01,
                        0x00, 0x00,
                        0x51,
                        0x3E, 0x4D};

uint8_t qrWriteOK[]={ 0x02, 0x00,
                      0x00,
                      0x01,
                      0x00,
                      0x33, 0x31};

uint8_t qrScan[]={ 0x7E, 0x00,
                   0x08,
                   0x01,
                   0x00, 0x02,
                   0x01,
                   0xAB, 0xCD};

uint8_t qrSetupOnRespondCode[]={0x51, 0x80, 0x52, 0x30, 0x33, 0x30, 0x32, 0x30,
0x30, 0x2E};
uint8_t buffAux[11];
uint8_t dataLeft;
uint8_t scanPending=0;
uint8_t SetupOnAnalyze = 0;
//CRC-CCITT (XModem)
uint16_t gm65_crc(uint8_t* ptr, unsigned int len)
{
    unsigned int crc = 0;
    while(len-- != 0)
    {
        for(unsigned char i = 0x80; i != 0; i >>= 1)
        {
            crc <= 1;
            if((crc&0x10000) !=0)
            crc ^= 0x11021;
        }
    }
}

```

```

    if((*ptr&i) != 0)
    crc ^= 0x1021;
    }
    ptr++;
    }
    return crc;
}
uint8_t qrRespondValid(uint8_t *respond){ //Funcion para verificar la confirmacion
de las configuraciones del qr
    uint8_t flag = 0;
    for(int i=0; i<7; i++){
        if(*(respond+i) != *(qrWriteOK+i)){
            flag = 1;
            break;
        }
    }
    return flag;
}
uint8_t qrRespSetupOnCode(uint8_t *code){
    uint8_t flag = 0;
    for(int i=0; i<10; i++){
        if(*(code+i) != *(qrSetupOnRespondCode+i)){
            flag = 1;
            break;
        }
    }
    return flag;
}
}
void gm65_init(struct gm65 *myQR, UART_HandleTypeDef *huart, TIM_HandleTypeDef
*htim){
    myQR->htim=htim;
    myQR->huart=huart;
    __HAL_TIM_CLEAR_FLAG(myQR->htim, TIM_FLAG_UPDATE);
    __HAL_TIM_CLEAR_IT(myQR->htim, TIM_IT_UPDATE);
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    __HAL_TIM_SET_COUNTER(myQR->htim, 0);
    myQR->qrStatus = QR_SETUP_OFF_T;
    myQR->qrDataReady=0;
    myQR->qrLengthDecode=0;
    myQR->reintentar= REINTENTAR;
    myQR->reintentos=0;
    myQR->reintentos_max=REINTENTOS_MAXIMOS;
}
void gm65_uart_TX_handler(struct gm65 *myQR){//, UART_HandleTypeDef *myQR->huart,
TIM_HandleTypeDef *myQR->htim){
    HAL_TIM_Base_Stop_IT(myQR->htim);
    switch (myQR->qrStatus){
    case QR_SETUP_OFF_T:
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_RESPOND_COMMAND);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
        __HAL_TIM_SET_COUNTER(myQR->htim, 0);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SETUP_OFF_R;
        break;
    case QR_SETUP_NO_LIGHT_T:
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_RESPOND_COMMAND);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);

```

```

        __HAL_TIM_SET_COUNTER(myQR->htim, 0);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SETUP_NO_LIGHT_R;
        break;
    case QR_SCAN_T:
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_RESPOND_COMMAND);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
        __HAL_TIM_SET_COUNTER(myQR->htim, 0);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SCAN_RC;
        break;
    case QR_SCAN_SETUP_OFF_T:
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_RESPOND_COMMAND);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
        __HAL_TIM_SET_COUNTER(myQR->htim, 0);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SCAN_SETUP_OFF_R;
        break;
    case QR_SCAN_NO_LIGHT_T:
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_RESPOND_COMMAND);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
        __HAL_TIM_SET_COUNTER(myQR->htim, 0);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SCAN_NO_LIGHT_R;
        break;
    default:
        break;
}
}

void gm65_uart_RX_handler(struct gm65 *myQR){//, UART_HandleTypeDef *myQR->huart,
TIM_HandleTypeDef *myQR->htim){
    HAL_TIM_Base_Stop_IT(myQR->htim);
    __HAL_TIM_SET_COUNTER(myQR->htim,0);
    switch (myQR->qrStatus){
    case QR_SETUP_OFF_R:
        if(qrRespondValid(myQR->qrBuffer) == 1){
            HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF,
QR_LENGTH_COMMAND);
            myQR->qrStatus = QR_SETUP_OFF_T;
        }
        else{
            myQR->qrStatus = QR_START;
            HAL_UART_Receive_IT(myQR->huart, buffAux, QR_LENGTH_HEAD);
            dataLeft=0;
            HAL_UART_Transmit_IT(myQR->huart, qrLight_OFF,
QR_LENGTH_COMMAND);
            myQR->qrStatus = QR_SETUP_NO_LIGHT_T;
        }
        break;
    case QR_SETUP_NO_LIGHT_R:
        if(qrRespondValid(myQR->qrBuffer) == 1){
            HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF,
QR_LENGTH_COMMAND);
            myQR->qrStatus = QR_SETUP_OFF_T;
        }
        else{

```

```

        myQR->qrStatus = QR_START;
        HAL_UART_Receive_IT(myQR->huart, buffAux, QR_LENGTH_HEAD);
        dataLeft=0;
    }
    break;
case QR_SCAN_RC:
    if(qrRespondValid(myQR->qrBuffer) == 1){
        HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
        myQR->qrStatus = QR_SCAN_T;
    }
    else{
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer,
QR_LENGTH_HEAD);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_SCAN_WAIT);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SCAN_RH;
    }
    break;
case QR_SCAN_RH:
    if((myQR->qrBuffer[0]=0x03) && (myQR->qrBuffer[1] == 0x00)){
        myQR->qrLengthDecode = myQR->qrBuffer[2];
        HAL_UART_Receive_IT(myQR->huart, myQR->qrBuffer, myQR-
>qrLengthDecode);
        __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_SCAN_WAIT);
        HAL_TIM_Base_Start_IT(myQR->htim);
        myQR->qrStatus = QR_SCAN_RD;
    }
    else{
        HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
        myQR->qrStatus = QR_SCAN_T;
    }
    break;
case QR_SCAN_RD:
    if((myQR->qrLengthDecode == 0x0B) && (qrRespSetupOnCode(myQR-
>qrBuffer)==0)){//SI SE DETECTO QUE SE ESCANEO UN CODE DE CONFIGURACION
        HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF,
QR_LENGTH_COMMAND);
        myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    }
    else{
        myQR->qrDataReady=1;
        myQR->lectura_qr=LECTURA_OK_QR;
        myQR->qrStatus = QR_START;
    }
    break;
case QR_SCAN_SETUP_OFF_R:
    if(qrRespondValid(myQR->qrBuffer) == 1){
        HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF,
QR_LENGTH_COMMAND);
        myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    }
    else{
        // HAL_UART_Transmit_IT(myQR->huart, qrLight_OFF,
QR_LENGTH_COMMAND);
        // myQR->qrStatus = QR_SCAN_NO_LIGHT_T;
        myQR->qrStatus = QR_SCAN_T;
        HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
    }
    break;

```

```

    case QR_SCAN_NO_LIGHT_R:
        if(qrRespondValid(myQR->qrBuffer) == 1){
            HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF,
QR_LENGTH_COMMAND);
            myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
        }
        else{
            HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
            myQR->qrStatus = QR_SCAN_T;
        }
        break;
    case QR_START:
        if((buffAux[0]==0x03) && (buffAux[1]==0x00)){
            dataLeft = buffAux[2];
            if(dataLeft <= 11){
                HAL_UART_Receive_IT(myQR->huart, buffAux, dataLeft);
                __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
                HAL_TIM_Base_Start_IT(myQR->htim);
                if(dataLeft == 0x0B) SetupOnAnalyze=1;
                dataLeft=0;
            }
            else{
                dataLeft=dataLeft - 11;
                HAL_UART_Receive_IT(myQR->huart, buffAux, dataLeft);
                __HAL_TIM_SET_AUTORELOAD(myQR->htim, QR_COMMAND_WAIT);
                HAL_TIM_Base_Start_IT(myQR->htim);
            }
            myQR->qrStatus= QR_RECEPTION_UNKNOWN;
            break;
        }
    case QR_RECEPTION_UNKNOWN:
        if(dataLeft ==0){
            if((SetupOnAnalyze == 1) && (qrRespSetupOnCode(buffAux)==0)){
                gm65_init(myQR, myQR->huart, myQR->htim);
                SetupOnAnalyze = 0;
            }
        }
        myQR->qrStatus=QR_START;
        if(scanPending == 1) {
            scanPending = 0;
            gm65_scan(myQR);//, myQR->huart);
        }
    }
    default:
        break;
}

}

void gm65_Tim_handler(struct gm65 *myQR){//, UART_HandleTypeDef *myQR->huart,
TIM_HandleTypeDef *myQR->htim){
    HAL_UART_AbortReceive_IT(myQR->huart);
    HAL_TIM_Base_Stop_IT(myQR->htim);
    __HAL_TIM_SET_COUNTER(myQR->htim, 0);
    if ((myQR->qrStatus&0xf0)==QR_SCAN) {
        if((myQR->reintentar==REINTENTAR)){
            if((myQR->reintentos_max>=0)&&(myQR->reintentos>=myQR-
>reintentos_max)){
                myQR->reintentos=0;
                myQR->lectura_qr=FALLO_QR;
                //myQR->qrStatus=QR_START;
                gm65_abort(myQR);//, myQR->huart, myQR->htim);
            }
        }
    }
}

```

```

        }
        else{
            myQR->reintentos++;
        }
    }
    else{
        //myQR->qrStatus=QR_START;
        gm65_abort(myQR);//, myQR->huart, myQR->htim);
    }
}
else{
    myQR->reintentos=0;
}
switch (myQR->qrStatus){
case QR_SETUP_OFF_R:
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SETUP_OFF_T;
    break;
case QR_SETUP_NO_LIGHT_R:
    HAL_UART_Transmit_IT(myQR->huart, qrLight_OFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SETUP_NO_LIGHT_T;
    break;
case QR_SCAN_RC:
    // HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
    // myQR->qrStatus = QR_SCAN_T;
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    break;
case QR_SCAN_RH:
    // HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
    // myQR->qrStatus = QR_SCAN_T;
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    break;
case QR_SCAN_RD:
    // HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
    // myQR->qrStatus = QR_SCAN_T;
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    break;
case QR_SCAN_SETUP_OFF_R:
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    myQR->qrStatus = QR_SCAN_SETUP_OFF_T;
    break;
default:
    break;
}
}
void gm65_scan(struct gm65 *myQR){//, UART_HandleTypeDef *myQR->huart){
    if(myQR->qrStatus == QR_START){
        HAL_UART_AbortReceive_IT(myQR->huart);
        myQR->qrStatus=QR_SCAN_T;
        HAL_UART_Transmit_IT(myQR->huart, qrScan, QR_LENGTH_COMMAND);
        myQR->qrDataReady=0;
        myQR->lectura_qr=SCANEANDO_QR;
    }
    else if(myQR->qrStatus == QR_RECEPTION_UNKNOWN) scanPending = 1;
}

```

```

void gm65_abort(struct gm65 *myQR){//, UART_HandleTypeDef *myQR->huart,
TIM_HandleTypeDef *myQR->htim){
    HAL_UART_Abort_IT(myQR->huart);
    HAL_TIM_Base_Stop_IT(myQR->htim);
    __HAL_TIM_CLEAR_FLAG(myQR->htim, TIM_FLAG_UPDATE);
    __HAL_TIM_CLEAR_IT(myQR->htim, TIM_IT_UPDATE);
    HAL_UART_Transmit_IT(myQR->huart, qrSetupOFF, QR_LENGTH_COMMAND);
    __HAL_TIM_SET_COUNTER(myQR->htim, 0);
    myQR->qrStatus = QR_SETUP_OFF_T;
    myQR->reintentos_max=0;
    myQR->qrDataReady=0;
    myQR->lectura_qr=FALLO_QR;
}
int gm65_extraer_scan(struct gm65 *myQR, char* string){
    if(myQR->qrLengthDecode<3) return -1;
    char* c=(char*)(myQR->qrBuffer+1);
    for(int i=0; i<(myQR->qrLengthDecode-2); i++){
        *(string+i)=*(c+i);
    }
    return myQR->qrLengthDecode-2;
}

```

A continuación, se muestra el código de la librería del HC-SR04

```

/*
 * hc-sr04.h
 *
 * Created on: 7 ene. 2022
 * Author: Gonzalo Mansilla
 */
#ifndef INC_HC_SR04_H_
#define INC_HC_SR04_H_
#include "stm32f4xx_hal.h"
#define NUMBER_OF_SENSOR 2
#define FACTOR_CONV 0.017
#define DELANTERO 0
#define TRASERO 1
#define DISTANCIA_ENTRE_SENSORES 12
struct hcsr04{
    float distance[NUMBER_OF_SENSOR];
    uint16_t risingEdge[NUMBER_OF_SENSOR];
    uint16_t fallingEdge[NUMBER_OF_SENSOR];
    uint8_t edgeDetect;
    uint8_t dataReady[NUMBER_OF_SENSOR];
    uint16_t difference[NUMBER_OF_SENSOR];
};
void ultrasonic_init(TIM_HandleTypeDef* htim, TIM_HandleTypeDef* htim_pwm,
uint32_t channel_pwm_1, uint32_t channel_pwm_2);
void ultrasonic_tim_handler(struct hcsr04* myUS, TIM_HandleTypeDef* htim);
uint16_t tiempo_entre_flanco(uint16_t flanco_ascendente, uint16_t
flanco_descendente);
#endif /* INC_HC_SR04_H_ */

/*
 * hc-sr04.c
 *
 * Created on: 7 ene. 2022
 * Author: Gonzalo Mansilla
 */

```

```

#include "hc-sr04.h"
/*
 * El sensor de ultrasonido se conecta un pwm para cada trig y que el echo a un
 * canal
 * configurado a capturar flanco ascendente y a otro canal configurado a capturar
 * flanco
 * descendente, es decir, que el pin echo se conecta a 2 pines del uc
 * */
void ultrasonic_init(TIM_HandleTypeDef* htim, TIM_HandleTypeDef* htim_pwm,
uint32_t channel_pwm_1, uint32_t channel_pwm_2){
    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
TIM_INPUTCHANNELPOLARITY_RISING);
    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_2,
TIM_INPUTCHANNELPOLARITY_FALLING);
    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_3,
TIM_INPUTCHANNELPOLARITY_RISING);
    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4,
TIM_INPUTCHANNELPOLARITY_FALLING);
    HAL_TIM_IC_Start(htim, TIM_CHANNEL_1);
    HAL_TIM_IC_Start(htim, TIM_CHANNEL_2);
    HAL_TIM_IC_Start(htim, TIM_CHANNEL_3);
    HAL_TIM_IC_Start(htim, TIM_CHANNEL_4);
    __HAL_TIM_CLEAR_FLAG(htim,TIM_FLAG_CC1);
    __HAL_TIM_CLEAR_FLAG(htim,TIM_FLAG_CC2);
    __HAL_TIM_CLEAR_FLAG(htim,TIM_FLAG_CC3);
    __HAL_TIM_CLEAR_FLAG(htim,TIM_FLAG_CC4);
    // __HAL_TIM_CLEAR_IT(htim_pwm,TIM_IT_UPDATE);
    HAL_TIM_Base_Start_IT(htim_pwm);
    HAL_TIM_PWM_Start(htim_pwm, channel_pwm_1);
    HAL_TIM_PWM_Start(htim_pwm, channel_pwm_2);
}
//void ultrasonic_tim_handler(struct hcsr04* myUS, TIM_HandleTypeDef* htim){
//
//    if (myUS->edgeDetect == 0) //primer flanco del sensor
//    {
//        myUS->risingEdge[0] = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
//        myUS->edgeDetect = 1;
//        __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
TIM_INPUTCHANNELPOLARITY_FALLING); //Cambio de polaridad del IC
//        return;
//    }
//    else if(myUS->edgeDetect==1) //flag_flanco1 = 1
//    {
//        myUS->fallingEdge[0] = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);
//        HAL_TIM_IC_Stop_IT(htim, TIM_CHANNEL_1);
//
//        if (myUS->fallingEdge[0] > myUS->risingEdge[0])
//        {
//            myUS->difference[0] = myUS->fallingEdge[0] - myUS-
>risingEdge[0];
//        }
//        else
//        {
//            myUS->difference[0] = (0xFFFF - myUS->risingEdge[0]) + myUS-
>fallingEdge[0];
//        }
//
//        myUS->dataReady[0] = 1;
//    }
//}

```

```

//      myUS->edgeDetect = 2;
//      myUS->distance[0] = ((float)myUS->difference[0]) * FACTOR_CONV;
//      __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
TIM_INPUTCHANNELPOLARITY_RISING); //Cambio de polaridad del IC
//      HAL_TIM_IC_Start_IT(htim, TIM_CHANNEL_2);
//      return;
//  }
//  else if(myUS->edgeDetect==2){
//      myUS->risingEdge[1] = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
//      myUS->edgeDetect = 3;
//      __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_2,
TIM_INPUTCHANNELPOLARITY_FALLING); //Cambio de polaridad del IC
//      return;
//  }
//
//  else if(myUS->edgeDetect==3) //flag_flanco1 = 1
//  {
//      myUS->fallingEdge[1] = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_2);
//      HAL_TIM_IC_Stop_IT(htim, TIM_CHANNEL_2);
//
//      if (myUS->fallingEdge[1] > myUS->risingEdge[1])
//      {
//          myUS->difference[1] = myUS->fallingEdge[1] - myUS-
>risingEdge[1];
//      }
//      else
//      {
//          myUS->difference[1] = (0xFFFF - myUS->risingEdge[1]) + myUS-
>fallingEdge[1];
//      }
//
//      myUS->dataReady[1] = 1;
//      myUS->edgeDetect = 0;
//      myUS->distance[1] = ((float)myUS->difference[1]) * FACTOR_CONV;
//      __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_2,
TIM_INPUTCHANNELPOLARITY_RISING); //Cambio de polaridad del IC
//      HAL_TIM_IC_Start_IT(htim, TIM_CHANNEL_1);
//      return;
//  }
//}
//
/*
* En la interrupcion del timer del pwm, producto de la actualizacion del timer, se
debe colocar
* este handler para obtener las captura de los flancos y calcular la distancia
* */
void ultrasonic_tim_handler(struct hcsr04* myUS, TIM_HandleTypeDef* htim){
    if(htim->Instance->SR && 0x1E == 0x1E){
        myUS->risingEdge[0] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1);
        myUS->fallingEdge[0] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_2);
        myUS->distance[0] = tiempo_entre_flanco(myUS->risingEdge[0], myUS-
>fallingEdge[0]) * FACTOR_CONV;
        myUS->dataReady[0] = 1;
        myUS->risingEdge[1] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_3);
        myUS->fallingEdge[1] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_4);
        myUS->distance[1] = tiempo_entre_flanco(myUS->risingEdge[1], myUS-
>fallingEdge[1]) * FACTOR_CONV;
        myUS->dataReady[1] = 1;
    }
}

```

```

    }
    else{
        myUS->distance[0]=0;
        myUS->dataReady[0] = 1;
        myUS->distance[1]=0;
        myUS->dataReady[1] = 1;
    }

//
//      if((__HAL_TIM_GET_FLAG(htim, TIM_CHANNEL_3))&& (__HAL_TIM_GET_FLAG(htim,
TIM_CHANNEL_4))){
//
//          myUS->risingEdge[1] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_3);
//          myUS->fallingEdge[1] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_4);
//          myUS->distance[1] = tiempo_entre_flanco(myUS->risingEdge[0], myUS-
>fallingEdge[0]) * FACTOR_CONV;
//          myUS->dataReady[1] = 1;
//      }
//      else{
//          myUS->distance[1]=0;
//          myUS->dataReady[1] = 1;
//      }
//      __HAL_TIM_CLEAR_FLAG(htim, TIM_CHANNEL_1);
//      __HAL_TIM_CLEAR_FLAG(htim, TIM_CHANNEL_2);
//      __HAL_TIM_CLEAR_FLAG(htim, TIM_CHANNEL_3);
//      __HAL_TIM_CLEAR_FLAG(htim, TIM_CHANNEL_4);
//      return;
}
uint16_t tiempo_entre_flanco(uint16_t flanco_ascendente, uint16_t
flanco_descendente){
    if(flanco_ascendente > flanco_descendente){
        return (flanco_ascendente - flanco_descendente);
    }
    else{
        return (flanco_descendente + 0xffff - flanco_ascendente);
    }
}

```

A continuación, se muestra el código de la librería de la comunicación con los demás microcontroladores.

```

/*
 * com_bp.h
 *
 * Created on: 17 ago. 2023
 * Author: Gonzalo Mansilla
 */
#ifndef INC_COM_BP_H_
#define INC_COM_BP_H_
#include "stm32f4xx_hal.h"
#include "rs485.h"
#include "ordenes.h"
#include "conversion_array.h"
#define CONT_VELOCIDAD 1
#define CONT_POSICION 2
#define PROPORCIONAL 0
#define INTEGRAL 1
#define DERIVATIVO 2
#define BP_ERROR 0
#define BP_OK 1

```

```

#define INTENTOS_MAX      5
#define ABSOLUTO           0
#define RELATIVA          1
#define ADELANTE          0
#define ATRAS             1
#define LATERAL_IZQ       2
#define LATERAL_DER       3
#define GIRAR_IZQ         4
#define GIRAR_DER         5
struct com_bp{
    TIM_HandleTypeDef *tim_base;
    struct rs485 *rs;
    uint8_t buff[9];
};
void com_bp_init(struct com_bp* bp, struct rs485* rs, TIM_HandleTypeDef *tim);
int com_bp_enviar( struct com_bp* bp, uint8_t orden, uint8_t dir);
int com_bp_set_coef(struct com_bp* bp, uint8_t coef, uint8_t motor,uint8_t cont,
float coef_f);
int com_bp_ver_coef(struct com_bp* bp, uint8_t coef, uint8_t motor,uint8_t cont,
float* coef_f);
int com_bp_test(struct com_bp* bp);
int com_bp_set_setpoint(struct com_bp* bp, uint8_t cont, uint8_t ref, float f);
int com_bp_set_setpoint_a_motor(struct com_bp* bp, uint8_t cont, uint8_t
ref,uint8_t motor, float f);
int com_bp_set_coef_todos(struct com_bp* bp, uint8_t coef, uint8_t cont, float f);
int com_bp_ver_vuelta(struct com_bp* bp, uint8_t motor, float *vuelta);
int com_bp_ver_vuelta_prom(struct com_bp* bp, float *vuelta);
int com_bp_ver_velocidad(struct com_bp* bp, uint8_t motor, float *vel);
int com_bp_ver_float(struct com_bp* bp, uint8_t orden, uint8_t dir, float* f_out);
int com_bp_moverse(struct com_bp*bp,uint8_t controlador, uint8_t direccion, float
velocidad);
int com_bp_reset_encoder(struct com_bp*bp);
int com_bp_set_controlador(struct com_bp* bp, uint8_t controlador);
int com_bp_ver_pwm_todos(struct com_bp* bp, int* pwm);
int com_bp_ver_pwm_de_motor(struct com_bp* bp, uint8_t motor, int* pwm);
#define BP_VER_FLOAT(BP,ORDEN,DIR,F_OUT)      com_bp_enviar(BP, ORDEN, DIR);
F_OUT=array_to_float(bp->rs->data+1)
#endif /* INC_COM_BP_H */
/*
 * com_bp.c
 *
 * Created on: 17 ago. 2023
 * Author: Gonzalo Mansilla
 */
#include "com_bp.h"
#define MAX_DATA      9
void com_bp_init(struct com_bp *bp, struct rs485 *rs, TIM_HandleTypeDef *tim) {
    bp->tim_base = tim;
    bp->rs = rs;
    com_bp_set_coef_todos(bp, PROPORCIONAL, CONT_POSICION, 9.25245);
    com_bp_set_coef_todos(bp, DERIVATIVO, CONT_POSICION, 0.45152);
    com_bp_set_coef_todos(bp, PROPORCIONAL, CONT_VELOCIDAD,
13.47195);//13.47195);
    com_bp_set_coef_todos(bp, INTEGRAL, CONT_VELOCIDAD, 1.614277);//1.614277);
}
int com_bp_enviar(struct com_bp *bp, uint8_t orden, uint8_t dir) {
    bp->buff[0] = orden;
    // if(orden==(CONTROLAR_POSICION)){
    //     bp->buff[0]=orden;

```

```

// }
rs485_transmit_master(bp->rs, bp->buff, MAX_DATA, dir);
__HAL_TIM_SET_COUNTER(bp->tim_base, 0);
HAL_TIM_Base_Start(bp->tim_base);
uint16_t count;
int intentos=0;
while (bp->rs->new_data == 0) {
    count = __HAL_TIM_GET_COUNTER(bp->tim_base);
    if (count >= 5000 ) {
        __HAL_TIM_SET_COUNTER(bp->tim_base, 0);
        rs485_abort(bp->rs);
        rs485_transmit_master(bp->rs, bp->buff, MAX_DATA, dir);
        rs485_receive(bp->rs);
        intentos++;
        if(intentos>INTENTOS_MAX){
            if(dir==1){
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, RESET);
            }
            else{
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, RESET);
            }
            HAL_TIM_Base_Stop(bp->tim_base);
            return BP_ERROR;
        }
    }
    if (bp->rs->uart->RxState == HAL_UART_STATE_READY) {
        rs485_receive(bp->rs);
    }
}
HAL_TIM_Base_Stop(bp->tim_base);
bp->rs->new_data=0;
if(dir==1){
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, SET);
}
else{
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, SET);
}
return BP_OK;
}

int com_bp_set_coef(struct com_bp *bp, uint8_t coef, uint8_t motor,
    uint8_t cont, float coef_f) {
    uint8_t orden;
    uint8_t dir_uc;
    if (motor == MOTOR_DD || motor == MOTOR_DI) {
        dir_uc = DIR_MOTOR_DD;
    } else {
        dir_uc = DIR_MOTOR_TD;
    }
    float_to_array(coef_f, bp->buff + 1);
    if (motor == MOTOR_DD || motor == MOTOR_TD) {
        if (cont == CONT_VELOCIDAD) {
            if (coef == PROPORCIONAL) {
                orden = SET_PROPORCIONAL_VELOCIDAD_1;
            } else {
                orden = SET_INTEGRAL_VELOCIDAD_1;
            }
        } else {
            if (coef == PROPORCIONAL) {
                orden = SET_PROPORCIONAL_POSICION_1;
            }
        }
    }
}

```

```

        } else {
            orden = SET_DERIVATIVO_POSICION_1;
        }
    }
} else {
    if (cont == CONT_VELOCIDAD) {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_VELOCIDAD_2;
        } else {
            orden = SET_INTEGRAL_VELOCIDAD_2;
        }
    } else {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_POSICION_2;
        } else {
            orden = SET_DERIVATIVO_POSICION_2;
        }
    }
}
return com_bp_enviar(bp, orden, dir_uc);
}

int com_bp_ver_coef(struct com_bp* bp, uint8_t coef, uint8_t motor, uint8_t cont,
float* coef_f){
    uint8_t orden;
    uint8_t dir_uc;
    if (motor == MOTOR_DD || motor == MOTOR_DI) {
        dir_uc = DIR_MOTOR_DD;
    } else {
        dir_uc = DIR_MOTOR_TD;
    }
    if (motor == MOTOR_DD || motor == MOTOR_TD) {
        if (cont == CONT_VELOCIDAD) {
            if (coef == PROPORCIONAL) {
                orden = VER_PROPORCIONAL_VELOCIDAD_1;
            } else {
                orden = VER_INTEGRAL_VELOCIDAD_1;
            }
        } else {
            if (coef == PROPORCIONAL) {
                orden = VER_PROPORCIONAL_POSICION_1;
            } else {
                orden = VER_DERIVATIVO_POSICION_1;
            }
        }
    }
} else {
    if (cont == CONT_VELOCIDAD) {
        if (coef == PROPORCIONAL) {
            orden = VER_PROPORCIONAL_VELOCIDAD_2;
        } else {
            orden = VER_INTEGRAL_VELOCIDAD_2;
        }
    } else {
        if (coef == PROPORCIONAL) {
            orden = VER_PROPORCIONAL_POSICION_2;
        } else {
            orden = VER_DERIVATIVO_POSICION_2;
        }
    }
}
}

```

```

    int status=com_bp_enviar(bp, orden, dir_uc);
    *coef_f=array_to_float(bp->rs->data+1);
    return status;
}
int com_bp_test(struct com_bp* bp){
    int test=com_bp_enviar(bp, VER_CUENTAS_ENCODER_1, 0);
    test= test | (com_bp_enviar(bp, VER_CUENTAS_ENCODER_1, 1)<<1);
    return test;
}
int com_bp_set_setpoint(struct com_bp* bp, uint8_t cont, uint8_t ref, float f){
    float sp=0;
    int status=BP_OK;
    if(ref==RELATIVA && cont==CONT_POSICION){
        status=com_bp_enviar(bp, VER_SETPOINT_POSICION_1, 0);
        if(status==BP_ERROR) return BP_ERROR;
        sp=array_to_float(bp->rs->data+1);
        sp+=f;
    }
    else sp=f;
    float_to_array(sp, bp->buff+1);
    if(cont==CONT_POSICION){
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_1, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_2, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_1, 1);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_2, 1);
        if(status==BP_ERROR) return BP_ERROR;
    }
    else{
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_1, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_2, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_1, 1);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_2, 1);
        if(status==BP_ERROR) return BP_ERROR;
    }
    return status;
}
int com_bp_set_coef_todos(struct com_bp* bp, uint8_t coef, uint8_t cont, float f){
    int status=com_bp_set_coef(bp, coef, MOTOR_DD, cont, f);
    status&=com_bp_set_coef(bp, coef, MOTOR_DI, cont, f);
    status&=com_bp_set_coef(bp, coef, MOTOR_TD, cont, f);
    status&=com_bp_set_coef(bp, coef, MOTOR_TI, cont, f);
    return status;
}
int com_bp_ver_vuelta(struct com_bp* bp, uint8_t motor, float *vuelta){
    int orden;
    int dir_uc;
    switch(motor){
        case MOTOR_DD:
            orden=VER_NUMERO_DE_VUELTAS_1;
            dir_uc=DIR_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden=VER_NUMERO_DE_VUELTAS_2;

```

```

        dir_uc=DIR_MOTOR_DI;
        break;
    case MOTOR_TD:
        orden=VER_NUMERO_DE_VUELTAS_1;
        dir_uc=DIR_MOTOR_TD;
        break;
    case MOTOR_TI:
        orden=VER_NUMERO_DE_VUELTAS_2;
        dir_uc=DIR_MOTOR_TI;
        break;
}
int status=com_bp_enviar(bp, orden, dir_uc);
*vuelta=array_to_float(bp->rs->data+1);
return status;
}
int com_bp_ver_vuelta_prom(struct com_bp* bp, float *vuelta){
    int status=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_1, 0);
    *vuelta=array_to_float(bp->rs->data+1);
    status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_2, 0);
    *vuelta+=array_to_float(bp->rs->data+1);
    status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_1, 1);
    *vuelta+=array_to_float(bp->rs->data+1);
    status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_2, 1);
    *vuelta+=array_to_float(bp->rs->data+1);
    *vuelta/=4;
    return status;
}
int com_bp_set_setpoint_a_motor(struct com_bp* bp, uint8_t cont, uint8_t
ref,uint8_t motor, float f){
    float sp=0;
    int status=BP_OK;
    if(ref==RELATIVA && cont==CONT_POSICION){
        int orden_p;
        int dir_uc_p;
        switch(motor){
            case MOTOR_DD:
                orden_p=VER_SETPOINT_POSICION_1;
                dir_uc_p=DIR_MOTOR_DD;
                break;
            case MOTOR_DI:
                orden_p=VER_SETPOINT_POSICION_2;
                dir_uc_p=DIR_MOTOR_DI;
                break;
            case MOTOR_TD:
                orden_p=VER_SETPOINT_POSICION_1;
                dir_uc_p=DIR_MOTOR_TD;
                break;
            case MOTOR_TI:
                orden_p=VER_SETPOINT_POSICION_2;
                dir_uc_p=DIR_MOTOR_TI;
                break;
        }
        status=com_bp_enviar(bp, orden_p,dir_uc_p );
        if(status==BP_ERROR) return BP_ERROR;
        sp=array_to_float(bp->rs->data+1);
        sp+=f;
    }
    else sp=f;
    float_to_array(sp, bp->buff+1);
}

```

```

    int orden;
    int dir_uc;
    if (cont==CONT_POSICION) {
        switch(motor){
            case MOTOR_DD:
                orden=SET_SETPOINT_POSICION_1;
                dir_uc=DIR_MOTOR_DD;
                break;
            case MOTOR_DI:
                orden=SET_SETPOINT_POSICION_2;
                dir_uc=DIR_MOTOR_DI;
                break;
            case MOTOR_TD:
                orden=SET_SETPOINT_POSICION_1;
                dir_uc=DIR_MOTOR_TD;
                break;
            case MOTOR_TI:
                orden=SET_SETPOINT_POSICION_2;
                dir_uc=DIR_MOTOR_TI;
                break;
        }
    }
    else{
        switch(motor){
            case MOTOR_DD:
                orden=SET_SETPOINT_VELOCIDAD_1;
                dir_uc=DIR_MOTOR_DD;
                break;
            case MOTOR_DI:
                orden=SET_SETPOINT_VELOCIDAD_2;
                dir_uc=DIR_MOTOR_DI;
                break;
            case MOTOR_TD:
                orden=SET_SETPOINT_VELOCIDAD_1;
                dir_uc=DIR_MOTOR_TD;
                break;
            case MOTOR_TI:
                orden=SET_SETPOINT_VELOCIDAD_2;
                dir_uc=DIR_MOTOR_TI;
                break;
        }
    }
    status=com_bp_enviar(bp, orden, dir_uc);
    return status;
}

int com_bp_ver_float(struct com_bp* bp, uint8_t orden, uint8_t dir, float* f_out){
    int status=com_bp_enviar(bp, orden, dir);
    *f_out=array_to_float(bp->rs->data+1);
    return status;
}

int com_bp_ver_velocidad(struct com_bp* bp, uint8_t motor, float* vel){
    uint8_t orden_p, dir_uc_p;
    switch(motor){
        case MOTOR_DD:
            orden_p=VER_VELOCIDAD_1;
            dir_uc_p=DIR_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden_p=VER_VELOCIDAD_2;

```

```

        dir_uc_p=DIR_MOTOR_DI;
        break;
    case MOTOR_TD:
        orden_p=VER_VELOCIDAD_1;
        dir_uc_p=DIR_MOTOR_TD;
        break;
    case MOTOR_TI:
        orden_p=VER_VELOCIDAD_2;
        dir_uc_p=DIR_MOTOR_TI;
        break;
    }
    uint8_t status=BP_VER_FLOAT(bp,orden_p,dir_uc_p,*vel);
    return status;
}
int com_bp_movearse(struct com_bp*bp,uint8_t controlador, uint8_t direccion, float
sp){
    float vel[4];
    switch(direccion){
    case ADELANTE:
        vel[0]=sp;
        vel[1]=sp;
        vel[2]=sp;
        vel[3]=sp;
        break;
    case ATRAS:
        vel[0]=-sp;
        vel[1]=-sp;
        vel[2]=-sp;
        vel[3]=-sp;
        break;
    case LATERAL_DER:
        vel[0]=sp;
        vel[1]=-sp;
        vel[2]=-sp;
        vel[3]=sp;
        break;
    case LATERAL_IZQ:
        vel[0]=-sp;
        vel[1]=sp;
        vel[2]=sp;
        vel[3]=-sp;
        break;
    case GIRAR_DER:
        vel[0]=-sp;
        vel[1]=sp;
        vel[2]=-sp;
        vel[3]=sp;
        break;
    case GIRAR_IZQ:
        vel[0]=sp;
        vel[1]=-sp;
        vel[2]=sp;
        vel[3]=-sp;
        break;
    }
    int status=BP_OK;
    // status&=com_bp_set_controlador(bp, controlador);
    for(int i=1;i<=MOTOR_TI;i++){

```

```

        status&=com_bp_set_setpoint_a_motor(bp, controlador, ABSOLUTO, i,
vel[i-1]);
    }
    return status;
}

int com_bp_reset_encoder(struct com_bp*bp){
    int status=BP_OK;
    status&=com_bp_enviar(bp, RESET_CUENTAS, 0);
    status&=com_bp_enviar(bp, RESET_CUENTAS, 1);
    return status;
}

int com_bp_set_controlador(struct com_bp* bp, uint8_t controlador){
    if(controlador==(CONT_POSICION) || controlador==(CONTROLAR_POSICION))
controlador=(CONTROLAR_POSICION);
    else if(controlador==(CONT_VELOCIDAD) || controlador==(
CONTROLAR_VELOCIDAD)) controlador=CONTROLAR_VELOCIDAD;
    else return BP_ERROR;
    int status=com_bp_enviar(bp, controlador, 0);
    status&=com_bp_enviar(bp, controlador, 1);
    return status;
}

int com_bp_ver_pwm_de_motor(struct com_bp* bp, uint8_t motor, int* pwm){
    int status=BP_OK;
    float p;
    // status&=com_bp_ver_float(bp, VER_PWM_1, UC_MOTOR_DD, &p);
    int orden;
    int dir;
    switch(motor){
        case MOTOR_DD:
            orden=VER_PWM_1;
            dir=UC_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden=VER_PWM_2;
            dir=UC_MOTOR_DI;
            break;
        case MOTOR_TD:
            orden=VER_PWM_1;
            dir=UC_MOTOR_TD;
            break;
        case MOTOR_TI:
            orden=VER_PWM_2;
            dir=UC_MOTOR_TI;
            break;
    }
    status=com_bp_ver_float(bp, orden, dir, &p);
    *pwm=(int)p;
    return status;
}

int com_bp_ver_pwm_todos(struct com_bp* bp, int* pwm){
    int status=BP_OK;
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_DD, pwm);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_DI, pwm+1);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_TD, pwm+2);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_TI, pwm+3);
    return status;
}
/*
* com_bp.c

```

```

*
* Created on: 17 ago. 2023
* Author: Gonzalo Mansilla
*/
#include "com_bp.h"
#define MAX_DATA 9
void com_bp_init(struct com_bp *bp, struct rs485 *rs, TIM_HandleTypeDef *tim) {
    bp->tim_base = tim;
    bp->rs = rs;
    com_bp_set_coef_todos(bp, PROPORCIONAL, CONT_POSICION, 9.25245);
    com_bp_set_coef_todos(bp, DERIVATIVO, CONT_POSICION, 0.45152);
    com_bp_set_coef_todos(bp, PROPORCIONAL, CONT_VELOCIDAD,
13.47195); //13.47195);
    com_bp_set_coef_todos(bp, INTEGRAL, CONT_VELOCIDAD, 1.614277); //1.614277);
}
int com_bp_enviar(struct com_bp *bp, uint8_t orden, uint8_t dir) {
    bp->buff[0] = orden;
    // if(orden==(CONTROLAR_POSICION)){
    //     bp->buff[0]=orden;
    // }
    rs485_transmit_master(bp->rs, bp->buff, MAX_DATA, dir);
    __HAL_TIM_SET_COUNTER(bp->tim_base, 0);
    HAL_TIM_Base_Start(bp->tim_base);
    uint16_t count;
    int intentos=0;
    while (bp->rs->new_data == 0) {
        count = __HAL_TIM_GET_COUNTER(bp->tim_base);
        if (count >= 5000 ) {
            __HAL_TIM_SET_COUNTER(bp->tim_base, 0);
            rs485_abort(bp->rs);
            rs485_transmit_master(bp->rs, bp->buff, MAX_DATA, dir);
            rs485_receive(bp->rs);
            intentos++;
            if(intentos>INTENTOS_MAX){
                if(dir==1){
                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, RESET);
                }
                else{
                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, RESET);
                }
                HAL_TIM_Base_Stop(bp->tim_base);
                return BP_ERROR;
            }
        }
        if (bp->rs->uart->RxState == HAL_UART_STATE_READY) {
            rs485_receive(bp->rs);
        }
    }
    HAL_TIM_Base_Stop(bp->tim_base);
    bp->rs->new_data=0;
    if(dir==1){
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, SET);
    }
    else{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, SET);
    }
    return BP_OK;
}
int com_bp_set_coef(struct com_bp *bp, uint8_t coef, uint8_t motor,

```

```

        uint8_t cont, float coef_f) {
uint8_t orden;
uint8_t dir_uc;
if (motor == MOTOR_DD || motor == MOTOR_DI) {
    dir_uc = DIR_MOTOR_DD;
} else {
    dir_uc = DIR_MOTOR_TD;
}
float_to_array(coef_f, bp->buff + 1);
if (motor == MOTOR_DD || motor == MOTOR_TD) {
    if (cont == CONT_VELOCIDAD) {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_VELOCIDAD_1;
        } else {
            orden = SET_INTEGRAL_VELOCIDAD_1;
        }
    } else {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_POSICION_1;
        } else {
            orden = SET_DERIVATIVO_POSICION_1;
        }
    }
} else {
    if (cont == CONT_VELOCIDAD) {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_VELOCIDAD_2;
        } else {
            orden = SET_INTEGRAL_VELOCIDAD_2;
        }
    } else {
        if (coef == PROPORCIONAL) {
            orden = SET_PROPORCIONAL_POSICION_2;
        } else {
            orden = SET_DERIVATIVO_POSICION_2;
        }
    }
}
return com_bp_enviar(bp, orden, dir_uc);
}

int com_bp_ver_coef(struct com_bp* bp, uint8_t coef, uint8_t motor, uint8_t cont,
float* coef_f){
    uint8_t orden;
    uint8_t dir_uc;
    if (motor == MOTOR_DD || motor == MOTOR_DI) {
        dir_uc = DIR_MOTOR_DD;
    } else {
        dir_uc = DIR_MOTOR_TD;
    }
    if (motor == MOTOR_DD || motor == MOTOR_TD) {
        if (cont == CONT_VELOCIDAD) {
            if (coef == PROPORCIONAL) {
                orden = VER_PROPORCIONAL_VELOCIDAD_1;
            } else {
                orden = VER_INTEGRAL_VELOCIDAD_1;
            }
        } else {
            if (coef == PROPORCIONAL) {
                orden = VER_PROPORCIONAL_POSICION_1;
            }
        }
    }
}

```

```

        } else {
            orden = VER_DERIVATIVO_POSICION_1;
        }
    }
} else {
    if (cont == CONT_VELOCIDAD) {
        if (coef == PROPORCIONAL) {
            orden = VER_PROPORCIONAL_VELOCIDAD_2;
        } else {
            orden = VER_INTEGRAL_VELOCIDAD_2;
        }
    } else {
        if (coef == PROPORCIONAL) {
            orden = VER_PROPORCIONAL_POSICION_2;
        } else {
            orden = VER_DERIVATIVO_POSICION_2;
        }
    }
}
int status=com_bp_enviar(bp, orden, dir_uc);
*coef_f=array_to_float(bp->rs->data+1);
return status;
}

int com_bp_test(struct com_bp* bp){
    int test=com_bp_enviar(bp, VER_CUENTAS_ENCODER_1, 0);
    test= test | (com_bp_enviar(bp, VER_CUENTAS_ENCODER_1, 1)<<1);
    return test;
}

int com_bp_set_setpoint(struct com_bp* bp, uint8_t cont, uint8_t ref, float f){
    float sp=0;
    int status=BP_OK;
    if(ref==RELATIVA && cont==CONT_POSICION){
        status=com_bp_enviar(bp, VER_SETPOINT_POSICION_1, 0);
        if(status==BP_ERROR) return BP_ERROR;
        sp=array_to_float(bp->rs->data+1);
        sp+=f;
    }
    else sp=f;
    float_to_array(sp, bp->buff+1);
    if(cont==CONT_POSICION){
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_1, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_2, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_1, 1);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_POSICION_2, 1);
        if(status==BP_ERROR) return BP_ERROR;
    }
    else{
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_1, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_2, 0);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_1, 1);
        // if(status==BP_ERROR) return BP_ERROR;
        status&=com_bp_enviar(bp, SET_SETPOINT_VELOCIDAD_2, 1);
        if(status==BP_ERROR) return BP_ERROR;
    }
}

```

```

        return status;
    }
    int com_bp_set_coef_todos(struct com_bp* bp, uint8_t coef, uint8_t cont, float f){
        int status=com_bp_set_coef(bp, coef, MOTOR_DD, cont, f);
        status&=com_bp_set_coef(bp, coef, MOTOR_DI, cont, f);
        status&=com_bp_set_coef(bp, coef, MOTOR_TD, cont, f);
        status&=com_bp_set_coef(bp, coef, MOTOR_TI, cont, f);
        return status;
    }
    int com_bp_ver_vuelta(struct com_bp* bp, uint8_t motor, float *vuelta){
        int orden;
        int dir_uc;
        switch(motor){
            case MOTOR_DD:
                orden=VER_NUMERO_DE_VUELTAS_1;
                dir_uc=DIR_MOTOR_DD;
                break;
            case MOTOR_DI:
                orden=VER_NUMERO_DE_VUELTAS_2;
                dir_uc=DIR_MOTOR_DI;
                break;
            case MOTOR_TD:
                orden=VER_NUMERO_DE_VUELTAS_1;
                dir_uc=DIR_MOTOR_TD;
                break;
            case MOTOR_TI:
                orden=VER_NUMERO_DE_VUELTAS_2;
                dir_uc=DIR_MOTOR_TI;
                break;
        }
        int status=com_bp_enviar(bp, orden, dir_uc);
        *vuelta=array_to_float(bp->rs->data+1);
        return status;
    }
    int com_bp_ver_vuelta_prom(struct com_bp* bp, float *vuelta){
        int status=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_1, 0);
        *vuelta=array_to_float(bp->rs->data+1);
        status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_2, 0);
        *vuelta+=array_to_float(bp->rs->data+1);
        status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_1, 1);
        *vuelta+=array_to_float(bp->rs->data+1);
        status&=com_bp_enviar(bp, VER_NUMERO_DE_VUELTAS_2, 1);
        *vuelta+=array_to_float(bp->rs->data+1);
        *vuelta/=4;
        return status;
    }
    int com_bp_set_setpoint_a_motor(struct com_bp* bp, uint8_t cont, uint8_t
ref,uint8_t motor, float f){
        float sp=0;
        int status=BP_OK;
        if(ref==RELATIVA && cont==CONT_POSICION){
            int orden_p;
            int dir_uc_p;
            switch(motor){
                case MOTOR_DD:
                    orden_p=VER_SETPOINT_POSICION_1;
                    dir_uc_p=DIR_MOTOR_DD;
                    break;
                case MOTOR_DI:

```

```

        orden_p=VER_SETPOINT_POSICION_2;
        dir_uc_p=DIR_MOTOR_DI;
        break;
    case MOTOR_TD:
        orden_p=VER_SETPOINT_POSICION_1;
        dir_uc_p=DIR_MOTOR_TD;
        break;
    case MOTOR_TI:
        orden_p=VER_SETPOINT_POSICION_2;
        dir_uc_p=DIR_MOTOR_TI;
        break;
    }
    status=com_bp_enviar(bp, orden_p,dir_uc_p );
    if(status==BP_ERROR) return BP_ERROR;
    sp=array_to_float(bp->rs->data+1);
    sp+=f;
}
else sp=f;
float_to_array(sp, bp->buff+1);
int orden;
int dir_uc;
if (cont==CONT_POSICION) {
    switch(motor){
        case MOTOR_DD:
            orden=SET_SETPOINT_POSICION_1;
            dir_uc=DIR_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden=SET_SETPOINT_POSICION_2;
            dir_uc=DIR_MOTOR_DI;
            break;
        case MOTOR_TD:
            orden=SET_SETPOINT_POSICION_1;
            dir_uc=DIR_MOTOR_TD;
            break;
        case MOTOR_TI:
            orden=SET_SETPOINT_POSICION_2;
            dir_uc=DIR_MOTOR_TI;
            break;
    }
}
else{
    switch(motor){
        case MOTOR_DD:
            orden=SET_SETPOINT_VELOCIDAD_1;
            dir_uc=DIR_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden=SET_SETPOINT_VELOCIDAD_2;
            dir_uc=DIR_MOTOR_DI;
            break;
        case MOTOR_TD:
            orden=SET_SETPOINT_VELOCIDAD_1;
            dir_uc=DIR_MOTOR_TD;
            break;
        case MOTOR_TI:
            orden=SET_SETPOINT_VELOCIDAD_2;
            dir_uc=DIR_MOTOR_TI;
            break;
    }
}

```

```

    }
}
status=com_bp_enviar(bp, orden, dir_uc);
return status;
}
int com_bp_ver_float(struct com_bp* bp, uint8_t orden, uint8_t dir, float* f_out){
    int status=com_bp_enviar(bp, orden, dir);
    *f_out=array_to_float(bp->rs->data+1);
    return status;
}
int com_bp_ver_velocidad(struct com_bp* bp, uint8_t motor, float* vel){
    uint8_t orden_p, dir_uc_p;
    switch(motor){
        case MOTOR_DD:
            orden_p=VER_VELOCIDAD_1;
            dir_uc_p=DIR_MOTOR_DD;
            break;
        case MOTOR_DI:
            orden_p=VER_VELOCIDAD_2;
            dir_uc_p=DIR_MOTOR_DI;
            break;
        case MOTOR_TD:
            orden_p=VER_VELOCIDAD_1;
            dir_uc_p=DIR_MOTOR_TD;
            break;
        case MOTOR_TI:
            orden_p=VER_VELOCIDAD_2;
            dir_uc_p=DIR_MOTOR_TI;
            break;
    }
    uint8_t status=BP_VER_FLOAT(bp,orden_p,dir_uc_p,*vel);
    return status;
}
int com_bp_moverse(struct com_bp*bp,uint8_t controlador, uint8_t direccion, float
sp){
    float vel[4];
    switch(direccion){
        case ADELANTE:
            vel[0]=sp;
            vel[1]=sp;
            vel[2]=sp;
            vel[3]=sp;
            break;
        case ATRAS:
            vel[0]=-sp;
            vel[1]=-sp;
            vel[2]=-sp;
            vel[3]=-sp;
            break;
        case LATERAL_DER:
            vel[0]=sp;
            vel[1]=-sp;
            vel[2]=-sp;
            vel[3]=sp;
            break;
        case LATERAL_IZQ:
            vel[0]=-sp;
            vel[1]=sp;
            vel[2]=sp;

```

```

        vel[3]=-sp;
        break;
    case GIRAR_DER:
        vel[0]=-sp;
        vel[1]=sp;
        vel[2]=-sp;
        vel[3]=sp;
        break;
    case GIRAR_IZQ:
        vel[0]=sp;
        vel[1]=-sp;
        vel[2]=sp;
        vel[3]=-sp;
        break;
    }
    int status=BP_OK;
    // status&=com_bp_set_controlador(bp, controlador);
    for(int i=1;i<=MOTOR_TI;i++){
        status&=com_bp_set_setpoint_a_motor(bp, controlador, ABSOLUTO, i,
        vel[i-1]);
    }
    return status;
}
int com_bp_reset_encoder(struct com_bp*bp){
    int status=BP_OK;
    status&=com_bp_enviar(bp, RESET_CUENTAS, 0);
    status&=com_bp_enviar(bp, RESET_CUENTAS, 1);
    return status;
}
int com_bp_set_controlador(struct com_bp* bp, uint8_t controlador){
    if(controlador==(CONT_POSICION) || controlador==(CONTROLAR_POSICION))
        controlador=(CONTROLAR_POSICION);
    else if(controlador==(CONT_VELOCIDAD) || controlador==(
        CONTROLAR_VELOCIDAD)) controlador=CONTROLAR_VELOCIDAD;
    else return BP_ERROR;
    int status=com_bp_enviar(bp, controlador, 0);
    status&=com_bp_enviar(bp, controlador, 1);
    return status;
}
int com_bp_ver_pwm_de_motor(struct com_bp* bp, uint8_t motor, int* pwm){
    int status=BP_OK;
    float p;
    // status&=com_bp_ver_float(bp, VER_PWM_1, UC_MOTOR_DD, &p);
    int orden;
    int dir;
    switch(motor){
    case MOTOR_DD:
        orden=VER_PWM_1;
        dir=UC_MOTOR_DD;
        break;
    case MOTOR_DI:
        orden=VER_PWM_2;
        dir=UC_MOTOR_DI;
        break;
    case MOTOR_TD:
        orden=VER_PWM_1;
        dir=UC_MOTOR_TD;
        break;
    case MOTOR_TI:

```

```

        orden=VER_PWM_2;
        dir=UC_MOTOR_TI;
        break;
    }
    status=com_bp_ver_float(bp, orden, dir, &p);
    *pwm=(int)p;
    return status;
}

int com_bp_ver_pwm_todos(struct com_bp* bp, int* pwm){
    int status=BP_OK;
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_DD, pwm);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_DI, pwm+1);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_TD, pwm+2);
    status&=com_bp_ver_pwm_de_motor(bp, MOTOR_TI, pwm+3);
    return status;
}

```

A continuación, se muestra el código de la librería de la interfaz de consola.

```

/*
 * cli.h
 *
 * Created on: 17 ago. 2023
 * Author: Gonzalo Mansilla
 */
#ifndef INC_CLI_H_
#define INC_CLI_H_
#include "stm32f4xx_hal.h"
#include "ordenes.h"
#include "com_bp.h"
#include "gm65.h"
#include "hc-sr04.h"
#include "auto_nav.h"
#include "controlador_robot.h"
// #include "cli_handler.h"
#define MAX_COMMAND_LENGTH 40
#define CHAR_CARGADO 1
#define CHAR_NO_CARGADO 0
#define ACTIVADO 1
#define DESACTIVADO 0
#define LISTO 1
#define NO_LISTO 0
#define COMANDO_OK BP_OK
#define COMANDO_OK_NO_RESPOND 11
#define COMANDO_ERROR 10
#define NO_COMANDOS 3
#define COMANDO_NO_ENCONTRADO 3
#define COMANDO_ERROR_COMM BP_ERROR
#define COMANDO_VACIO 12
#define COMANDO_DESHABILITADO 13
#define INDEX_FIN_COMANDO 5
struct cli{
    UART_HandleTypeDef *uart;
    char c[MAX_COMMAND_LENGTH];
    uint8_t index;
    uint8_t enable;
    uint8_t mensaje_bienvenida;
    uint8_t comandoListo;
    char texto[300];
}

```

```

//Elementos para el funcionamiento del robot
struct com_bp *bp;
struct gm65 *qr;
struct hcsr04 *us;
struct auto_nav *nav;
int identificacion;
int n;
int n_total;
int n_datos;
int iden_datos;
float iden_sp;
int iden_dir;
int iden_cont;
int calibracion;
int estado_cal;
int scaneando_qr;
cont_robot* robot;
};

void cli_init(struct cli *com, UART_HandleTypeDef *uart, struct com_bp *bp);
int cli_cargar_caracter(struct cli *com, char carac);
int cli_handler(struct cli *com);
#define ENVIAR_MENSAJE(mensaje) int longitud=sprintf(com->texto,mensaje);
HAL_UART_Transmit_DMA(com->uart, (uint8_t*)com->texto, longitud);
#endif /* INC_CLI_H */
/*
* cli.c
*
* Created on: 17 ago. 2023
* Author: Gonzalo Mansilla
*/
#include <cli_handler.h>
#include "cli.h"
#include "stdio.h"
#include "stddef.h"
#include "stdlib.h"
#include "string.h"
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof(arr[0]))
typedef struct shellComm{
    const char command[5];
    int (*handler)(struct cli *com);
}shell;
static const shell comandos[]={
    {"sProp",sProp_handler},
    {"sInte",sInte_handler},
    {"sDeri",sDeri_handler},
    {"vProp",vProp_handler},
    {"vInte",vInte_handler},
    {"vDeri",vDeri_handler},
    {"Copyv",Copyv_handler},
    {"ContV",ContV_handler},
    //{"ContM",ContM_handler},
    {"Encod",Encod_handler},
    {"Motor",Motor_handler},
    {"setSP",setSP_handler},
    {"scanQ",scanQ_handler},
    {"medUS",medUS_handler},
    {"mover",mover_handler},
    {"reset", reset_handler},
    {"auNav",auNav_handler},

```

```

        {"ident", ident_handler},
        {"robot", robot_handler},
        {"imuif", imu_handler},
        {"calmg", calmg_handler}
    };
    const size_t num_comandos=ARRAY_SIZE(comandos);
    const char texto[16]="robot iniciado\n\n";
    void cli_init(struct cli *com, UART_HandleTypeDef *uart, struct com_bp *bp){
        com->enable=DESACTIVADO;
        com->index=0;
        com->uart=uart;
        com->bp=bp;
        com->identificacion=0;
        com->scaneando_qr=DESACTIVADO;
        HAL_UART_Transmit_DMA(uart, (uint8_t*)texto, 16);
    }
    int cli_cargar_caracter(struct cli *com, char carac){
        if(carac=='!'){
            com->enable=ACTIVADO;
            com->index=0;
            com->mensaje_bienvenida=1;
            return CHAR_CARGADO;
        }
        if(com->enable==DESACTIVADO){
            return CHAR_NO_CARGADO;
        }
        if(carac=='$'){
            com->enable=DESACTIVADO;
            return CHAR_CARGADO;
        }
        com->c[com->index]=carac;
        if(carac=='\n'){
            // com->enable=DESACTIVADO;
            com->comandoListo=LISTO;
            com->index=0;
            return CHAR_CARGADO;
        }
        com->index++;
        if(com->index==MAX_COMMAND_LENGTH){
            com->comandoListo=LISTO;
            // com->enable=DESACTIVADO;
            com->index=0;
        }
        return CHAR_CARGADO;
    }
    int match_comando(char* comando_in, const char* comando_list){
        for(int i=0; i<5;i++){
            if(*(comando_in + i) != *(comando_list+i)) return 0;
        }
        return 1;
    }
    int cli_handler(struct cli *com){
        if(com->mensaje_bienvenida==1){
            com->mensaje_bienvenida=0;
            ENVIAR_MENSAJE("CONSOLA INCIADA\n\n>")
        }
        if(com->comandoListo==NO_LISTO){
            return NO_COMANDOS;
        }
    }

```

```

char comm_rec[5];
int status=COMANDO_NO_ENCONTRADO;
for(int i=0; i<5;i++){
    if(com->c[i]=='\n'){
        status=COMANDO_VACIO;
        com->comandoListo=NO_LISTO;
        ENVIAR_MENSAJE("\n>")
        return status;
    }
}
comm_rec[0]=com->c[0];
comm_rec[1]=com->c[1];
comm_rec[2]=com->c[2];
comm_rec[3]=com->c[3];
comm_rec[4]=com->c[4];
for(const shell *comm= comandos; comm<&comandos[num_comandos]; comm++){
    if(match_comando(comm_rec, comm-> command)){
        // if((!match_comando(comm_rec, "auNav")) &&(com->nav-
// >enable==AUTO_NAV_ACTIVADO) ){
//         status=COMANDO_DESHABILITADO;
//         break;
//     }
    status=comm->handler(com);
    break;
}
}
if(status==COMANDO_ERROR){
    int l=sprintf(com->texto,"El comando se utilizo
incorrectamente\n\n>");
    HAL_UART_Transmit_DMA(com->uart, (uint8_t*)com->texto, l);
}
else if(status==COMANDO_NO_ENCONTRADO){
    int l=sprintf(com->texto,"Se ingreso un comando desconocido\n\n>");
    HAL_UART_Transmit_DMA(com->uart, (uint8_t*)com->texto, l);
}
else if(status==COMANDO_ERROR_COMM){
    int l=sprintf(com->texto,"No se pudo comunicar con uno de los UCs de
los motores\n\n>");
    HAL_UART_Transmit_DMA(com->uart, (uint8_t*)com->texto, l);
}
else if(status==COMANDO_OK){
    ENVIAR_MENSAJE("Comando ejecutado correctamente\n\n>")
}
else if(status==COMANDO_DESHABILITADO){
    ENVIAR_MENSAJE("Este comando fue deshabilitado por una tarea
superior\n\n>")
}
com->comandoListo=NO_LISTO;
return status;
}

```

A continuación, se muestra el código del controlador del robot.

```

/*
* controlador_pos.h
*
* Created on: 26 nov 2023
* Author: Gonzalo Mansilla
*/

```

```

#ifndef INC_CONTROLADOR_ROBOT_H_
#define INC_CONTROLADOR_ROBOT_H_
#include "com_bp.h"
#include "pid.h"
#include "arm_math.h"
#include "arm_const_structs.h"
#include "imu.h"
#include "hc-sr04.h"
#define INDEX_THETA      0
#define INDEX_X          1
#define INDEX_Y          2
#define DELTA_ADMISIBLE_THETA 0.01
#define DELTA_ADMISIBLE_X 0.1
#define DELTA_ADMISIBLE_Y 0.1
#define INDEX_MOTOR_DD 0
#define INDEX_MOTOR_DI 1
#define INDEX_MOTOR_TD 2
#define INDEX_MOTOR_TI 3
#define CONTROLADOR_ERROR BP_ERROR
#define CONTROLADOR_OK BP_OK
#define TS_MS 100
#define L_ROBOT 0.13
#define W_ROBOT 0.18
#define LPLUSW_ROBOT (L_ROBOT+W_ROBOT)
#define R_WHEEL 0.04
#define INV_R_WHEEL (1/R_WHEEL)
#define CONTROLAR_ROBOT_FRAME 0
#define VEL_MAX 1
typedef struct{
// float x_setpoint;
// float y_setpoint;
// float theta_setpoint;
// float x_pos;
// float y_pos;
// float theta_pos;
int enable;
float u_input[4];
arm_matrix_instance_f32 u_input_instance;
float u_output[4];
arm_matrix_instance_f32 u_output_instance;
float cuentas_output[4],cuentas_ant_output[4];
arm_matrix_instance_f32 cuentas_output_instance,cuentas_ant_output_instance;
float setpoint[3];
arm_matrix_instance_f32 setpoint_instance;
float posicion_output[3];
arm_matrix_instance_f32 posicion_output_instance;
float posicion_output_enc[3];
arm_matrix_instance_f32 posicion_output_enc_instance;
float posicion_input[3];
arm_matrix_instance_f32 posicion_input_instance;
float setpoint_vel[3];
arm_matrix_instance_f32 setpoint_vel_instance;
// float output[3];
// arm_matrix_instance_f32 output_instance;
float vel_robot_output[3];
arm_matrix_instance_f32 vel_robot_output_instance;
float vel_fixed_output[3];
arm_matrix_instance_f32 vel_fixed_output_instance;
float vel_robot_input[3];

```

```

arm_matrix_instance_f32 vel_robot_input_instance;
float vel_fixed_input[3];
arm_matrix_instance_f32 vel_fixed_input_instance;
struct pid pid[3];
struct com_bp* bp;
int controlar;
float coef_pos[3];
float coef_vel[3];
float coef_wheel;
float coef_d_wheel;
float vel_max;
float dead_zone;
IMU* imu;
float angulo_bias_imu;
float angulo_imu;
float angulo_sentido;
struct hcsr04* us;
int us_hab;
int us_med_valid;
float angulo_bias_us;
float angulo_us;
float dist_max_us;
float dist_us;
float dist_us_bias;
float p_k_us;
float k_us;
float q_us;
float r_us;
int kalman;
// float pos_out_ant_k[3];
float p_k_k_m1;
float p_k;
float k;
float q;
float r;
}cont_robot;
int controlador_robot_init(cont_robot* cont, struct com_bp* bp);
int controlador_robot_update(cont_robot* cont);
int controlador_robot_enable(cont_robot* cont, int controlador);
int controlador_robot_disable(cont_robot* cont);
int controlador_robot_setpoint(cont_robot* cont, float theta, float x, float y);
int controlador_robot_kalman(cont_robot* cont);
float pasar_rango_menos2pi_2pi(float x);
float pasar_rango_0_2pi(float x);
#endif /* INC_CONTROLADOR_ROBOT_H_ */

/*
 * controlador_robot.c
 *
 * Created on: 26 nov 2023
 * Author: Gonzalo Mansilla
 */
#include "controlador_robot.h"
#include "ordenes.h"
#include "main.h"
#define VERIFICAR(status,c) status=c; if(status==BP_ERROR) return
CONTROLADOR_ERROR
static float robot_frame_to_vel_wheel[4][3]={ {-LPLUSW_ROBOT,1,1},

```

```

1},
LPLUSW_ROBOT,1,-1},

    {LPLUSW_ROBOT,1,1},
};
static arm_matrix_instance_f32 robot_frame_to_vel_wheel_instance;
static float fixed_frame_to_robot_frame[3][3]={    {1,0,0},

    {0,1,0},

    {0,0,1},
};
static arm_matrix_instance_f32 fixed_frame_to_robot_frame_instance;
static float robot_frame[3];
static arm_matrix_instance_f32 robot_frame_instance;
float vel_u_a_vel_dist=2.0*3.14159*R_WHEEL/4.0;
float vel_u_a_vel_theta=(2.0*3.14159*R_WHEEL/4.0)/LPLUSW_ROBOT;
float abs_f(float x){
    if(x<0) x*=-1;
    return x;
}
float pasar_rango_0_2pi(float x){
    while(x>2*3.14159){
        x-=2*3.14159;
    }
    while(x<0){
        x+=2*3.14159;
    }
    return x;
}
float pasar_rango_menos2pi_2pi(float x){
    while(x>2*3.14159){
        x-=2*3.14159;
    }
    while(x<-2*3.14159){
        x+=2*3.14159;
    }
    return x;
}
float pasar_rango_menos2pi_0(float x){
    while(x>0){
        x-=2*3.14159;
    }
    while(x<-2*3.14159){
        x+=2*3.14159;
    }
    return x;
}
void pasar_ang_imu_a_sistema(cont_robot* cont){
    cont->angulo_imu=cont->imu->yaw - cont->angulo_bias_imu;
    cont->angulo_imu*=cont->angulo_sentido;
    // float imu_rango_0_2pi=pasar_rango_0_2pi(cont->angulo_imu);
    // float theta_robot_rango=pasar_rango_0_2pi(cont->posicion_output[INDEX_THETA]);
    // float a=imu_rango_0_2pi-theta_robot_rango;
    // float b=2*3.1415-a; //angulo complementarios
    // if(a<b){

```

```

//          cont->angulo_imu=pasar_rango_menos2pi_2pi(cont-
>posicion_output[INDEX_THETA]+ a);
//      }
//      else{
//          cont->angulo_imu=pasar_rango_menos2pi_2pi(cont-
>posicion_output[INDEX_THETA]-b);
//      }
}
#define TS_US0.1
#define N_USO_US    3
static int n_datos_leidos=0;
static int datos_valid=1;
int obtener_ang_us(cont_robot* cont){
    if((cont->us->dataReady[DELANTERO]==0 || cont->us->dataReady[TRASERO]==0) &&
datos_valid==0){
        cont->angulo_us=0.0;
        cont->dist_us=0.0;
        return CONTROLADOR_ERROR;
    }
    if(cont->us->distance[0]>cont->dist_max_us || cont->us->distance[1]>cont-
>dist_max_us){
        cont->angulo_us=0;
        cont->dist_us=0;
        return CONTROLADOR_ERROR;
    }
    n_datos_leidos++;
    if(cont->us->dataReady[DELANTERO]==1 || cont->us->dataReady[TRASERO]==1){
        datos_valid=1;
        cont->us->dataReady[0]=0;
        cont->us->dataReady[1]=0;
        n_datos_leidos=0;
    }
    if(n_datos_leidos>N_USO_US){
        datos_valid=0;
        n_datos_leidos=0;
    }
    float delta=cont->us->distance[TRASERO]-cont->us->distance[DELANTERO];
    float ang=atan2f(delta,DISTANCIA_ENTRE_SENSORES);
    cont->angulo_us=ang-cont->angulo_bias_us;
    cont->dist_us=0.01*(cont->us-
>distance[DELANTERO])*arm_cos_f32(pasar_rango_0_2pi(cont-
>posicion_output[INDEX_THETA])) - cont->dist_us_bias;
    //    cont->dist_us=0.01*(cont->us->distance[0]+cont->us->distance[1])/2 - cont-
>dist_us_bias;
    //    cont->dist_us*=0.01;
    return CONTROLADOR_OK;
}
int controlador_robot_init(cont_robot* cont, struct com_bp* bp){
    //    cont->x_setpoint=0;
    //    cont->x_pos=0;
    //    cont->y_setpoint=0;
    //    cont->y_pos=0;
    //    cont->theta_pos=0;
    //    cont->theta_setpoint=0;
    cont->setpoint[INDEX_THETA]=0;
    cont->setpoint[INDEX_X]=0;
    cont->setpoint[INDEX_Y]=0;
    arm_mat_init_f32(&cont->setpoint_instance, 3, 1, cont->setpoint);
    pid_init(&cont->pid[0], TS_MS, ACTIVADO);
}

```

```

    pid_init(&cont->pid[1], TS_MS, ACTIVADO);
    pid_init(&cont->pid[2], TS_MS, ACTIVADO);
    // pid_init(&cont->pid[3], TS_MS, ACTIVADO);
    cont->pid[0].coef_prop=1;
    cont->pid[1].coef_prop=1;
    cont->pid[2].coef_prop=1;
    // cont->pid[3].coef_prop=1;
    cont->pid[0].coef_der=0;
    cont->pid[0].coef_inte=0;
    cont->pid[1].coef_der=0;
    cont->pid[1].coef_inte=0;
    cont->pid[2].coef_der=0;
    cont->pid[2].coef_inte=0;
    // cont->pid[0].limite_inf=-0.2;
    // cont->pid[0].limite_sup=0.2;
    // cont->pid[1].limite_inf=-0.2;
    // cont->pid[1].limite_sup=0.2;
    // cont->pid[2].limite_inf=-0.2;
    // cont->pid[2].limite_sup=0.2;
    // cont->pid[3].limite_inf=-0.2;
    // cont->pid[3].limite_sup=0.2;
    cont->pid[0].saturar=DESACTIVADO;
    cont->pid[1].saturar=DESACTIVADO;
    cont->pid[2].saturar=DESACTIVADO;
    cont->pid[0].windup=ACTIVADO;
    cont->pid[1].windup=ACTIVADO;
    cont->pid[2].windup=ACTIVADO;
    // cont->pid[3]->coef_der=0;
    // cont->pid[3]->coef_inte=0;
    cont->coef_pos[0]=5;
    cont->coef_pos[1]=5;
    cont->coef_pos[2]=5;
    cont->coef_vel[0]=1;
    cont->coef_vel[1]=1;
    cont->coef_vel[2]=1;
    cont->coef_wheel=0;
    cont->coef_d_wheel=0;
    cont->cuentas_ant_output[0]=0;
    cont->cuentas_ant_output[1]=0;
    cont->cuentas_ant_output[2]=0;
    cont->cuentas_ant_output[3]=0;
    arm_mat_init_f32(&cont->cuentas_ant_output_instance, 4, 1, cont-
>cuentas_ant_output);
    arm_mat_init_f32(&cont->cuentas_output_instance, 4, 1, cont-
>cuentas_output);
    arm_mat_init_f32(&cont->u_input_instance, 4, 1, cont->u_input);
    arm_mat_init_f32(&cont->u_output_instance, 4, 1, cont->u_output);
    arm_mat_init_f32(&cont->vel_robot_output_instance, 3, 1, cont-
>vel_robot_output);
    arm_mat_init_f32(&robot_frame_to_vel_wheel_instance, 4, 3,
&robot_frame_to_vel_wheel[0][0]);
    arm_mat_init_f32(&fixed_frame_to_robot_frame_instance, 3, 3,
&fixed_frame_to_robot_frame[0][0]);
    arm_mat_init_f32(&cont->vel_robot_input_instance, 3, 1, cont-
>vel_robot_input);
    arm_mat_init_f32(&cont->vel_fixed_input_instance, 3, 1, cont-
>vel_fixed_input);
    arm_mat_init_f32(&cont->vel_robot_output_instance, 3, 1, cont-
>vel_robot_output);

```

```

    arm_mat_init_f32(&cont->vel_fixed_output_instance,3,1,cont-
>vel_fixed_output);
    arm_mat_init_f32(&cont->setpoint_vel_instance, 3, 1, cont->setpoint_vel);
    arm_mat_init_f32(&cont->posicion_input_instance, 3, 1, cont-
>posicion_input);
    arm_mat_init_f32(&cont->posicion_output_instance, 3, 1, cont-
>posicion_output);
    arm_mat_init_f32(&cont->posicion_output_enc_instance, 3, 1, cont-
>posicion_output_enc);
    arm_mat_init_f32(&robot_frame_instance, 3, 1, robot_frame);
    cont->angulo_imu=0;
    cont->vel_max=1.0;
    cont->dead_zone=0.1;
    cont->angulo_sentido=1;
    cont->angulo_bias_us=0;
    cont->dist_max_us=50.0;
    cont->p_k=0.001;
    cont->q=0.001;//ruido del proceso
    cont->r=0.01;//ruido de medicion
    cont->kalman=ACTIVADO;
    cont->us_hab=DESACTIVADO;
    cont->dist_us_bias=0.350;
    cont->q_us=0.01;
    cont->r=0.01;
    cont->bp=bp;
    return CONTROLADOR_OK;
}
int controlador_robot_enable(cont_robot* cont, int controlador){
    int status;
    VERIFICAR(status,com_bp_set_controlador(cont->bp, CONTROLAR_VELOCIDAD));
    VERIFICAR(status,com_bp_reset_encoder(cont->bp));
    cont->cuentas_ant_output[0]=0;
    cont->cuentas_ant_output[1]=0;
    cont->cuentas_ant_output[2]=0;
    cont->cuentas_ant_output[3]=0;
    cont->posicion_output[0]=0;
    cont->posicion_output[1]=0;
    cont->posicion_output[2]=0;
    cont->setpoint[INDEX_THETA]=0;
    cont->setpoint[INDEX_X]=0;
    cont->setpoint[INDEX_X]=0;
    cont->angulo_imu=0;
    cont->angulo_bias_imu=cont->imu->yaw;
    cont->dist_us_bias=0.30;
    if(controlador==(CONT_POSICION) || controlador==(CONTROLAR_POSICION)) {
        controlador=(CONTROLAR_POSICION);
        cont->pid[0].coef_prop=cont->coef_pos[0];
        cont->pid[1].coef_prop=cont->coef_pos[1];
        cont->pid[2].coef_prop=cont->coef_pos[2];
    }
    else if(controlador==(CONT_VELOCIDAD) || controlador==(
CONTROLAR_VELOCIDAD)) {
        controlador=CONTROLAR_VELOCIDAD;
        cont->pid[0].coef_prop=cont->coef_vel[0];
        cont->pid[1].coef_prop=cont->coef_vel[1];
        cont->pid[2].coef_prop=cont->coef_vel[2];
    }
    else if(controlador==CONTROLAR_ROBOT_FRAME){
        cont->pid[0].coef_prop=cont->coef_vel[0];

```

```

        cont->pid[1].coef_prop=cont->coef_vel[1];
        cont->pid[2].coef_prop=cont->coef_vel[2];
    }
    else return BP_ERROR;
    cont->controlar=controlador;
    cont->enable=ACTIVADO;
    return status;
}

int controlador_robot_disable(cont_robot* cont){
    cont->enable=DESACTIVADO;
    int b=com_bp_moverse(cont->bp, CONTROLAR_VELOCIDAD, ADELANTE, 0);
    return b;
}

int controlador_robot_setpoint(cont_robot* cont, float theta, float x, float y){
    // cont->setpoint[INDEX_THETA]=theta;
    cont->setpoint[INDEX_X]=x;
    cont->setpoint[INDEX_Y]=y;
    cont->setpoint[INDEX_THETA]=theta;
    // float theta_rango=pasar_rango_0_2pi(theta);
    // float theta_robot_rango=pasar_rango_0_2pi(cont-
    >posicion_output[INDEX_THETA]);
    // float a=theta_rango-theta_robot_rango;
    // float b=2*3.1415-a; //angulo complementarios
    // if(a<b){
    //     cont->setpoint[INDEX_THETA]=pasar_rango_menos2pi_2pi(cont-
    >posicion_output[INDEX_THETA]+ a);
    // }
    // else{
    //     cont->setpoint[INDEX_THETA]=pasar_rango_menos2pi_2pi(cont-
    >posicion_output[INDEX_THETA]-b);
    // }
    // VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
    ABSOLUTO, MOTOR_DD, 0));
    // VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
    ABSOLUTO, MOTOR_DI, 0));
    // VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
    ABSOLUTO, MOTOR_TD, 0));
    // VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
    ABSOLUTO, MOTOR_TI, 0));
    return CONTROLADOR_OK;
}

int controlador_robot_kalman(cont_robot* cont){
    if(cont->kalman==ACTIVADO){
        cont->kalman=DESACTIVADO;
        return CONTROLADOR_OK;
    }
    cont->kalman=ACTIVADO;
    // cont->pos_out_ant_k[INDEX_THETA]=cont->posicion_output[INDEX_THETA];
    // cont->pos_out_ant_k[INDEX_X]=cont->posicion_output[INDEX_X];
    // cont->pos_out_ant_k[INDEX_Y]=cont->posicion_output[INDEX_Y];
    return CONTROLADOR_OK;
}

static float state_us[2]={0.0,0.0};
static float state_imu[2]={0.0,0.0};
const float B_high[]={0.00146,0.00292,0.00146};
const float B_low[]={0.94597,0.89195,0.94598};
const float A_r[]={1,1.88903,0.89487};
int controlador_robot_update(cont_robot* cont){
    int status;

```

```

    VERIFICAR(status, com_bp_ver_vuelta(cont->bp, MOTOR_DD, &cont-
>cuentas_output[INDEX_MOTOR_DD]));
    VERIFICAR(status, com_bp_ver_vuelta(cont->bp, MOTOR_DI, &cont-
>cuentas_output[INDEX_MOTOR_DI]));
    VERIFICAR(status, com_bp_ver_vuelta(cont->bp, MOTOR_TD, &cont-
>cuentas_output[INDEX_MOTOR_TD]));
    VERIFICAR(status, com_bp_ver_vuelta(cont->bp, MOTOR_TI, &cont-
>cuentas_output[INDEX_MOTOR_TI]));
    arm_mat_sub_f32(&cont->cuentas_output_instance, &cont-
>cuentas_ant_output_instance, &cont->u_output_instance);
    arm_mat_scale_f32(&cont->u_output_instance, 1.0/TS, &cont-
>u_output_instance);
    cont->cuentas_ant_output[0]=cont->cuentas_output[0];
    cont->cuentas_ant_output[1]=cont->cuentas_output[1];
    cont->cuentas_ant_output[2]=cont->cuentas_output[2];
    cont->cuentas_ant_output[3]=cont->cuentas_output[3];
    cont->vel_robot_output[INDEX_X]=(cont->u_output[INDEX_MOTOR_DD]+cont-
>u_output[INDEX_MOTOR_DI]+cont->u_output[INDEX_MOTOR_TD]+
cont-
>u_output[INDEX_MOTOR_TI]);
    cont->vel_robot_output[INDEX_X]*=(vel_u_a_vel_dist);
    cont->vel_robot_output[INDEX_Y]=(cont->u_output[INDEX_MOTOR_DD]-cont-
>u_output[INDEX_MOTOR_DI]+cont->u_output[INDEX_MOTOR_TI]-
cont-
>u_output[INDEX_MOTOR_TD]);
    cont->vel_robot_output[INDEX_Y]*=(vel_u_a_vel_dist);
    cont->vel_robot_output[INDEX_THETA]=(cont->u_output[INDEX_MOTOR_TI]-cont-
>u_output[INDEX_MOTOR_DD]+cont->u_output[INDEX_MOTOR_DI]-
cont-
>u_output[INDEX_MOTOR_TD]);
    cont->vel_robot_output[INDEX_THETA]*=(vel_u_a_vel_theta);
    float theta_en_rango=pasar_rango_0_2pi(cont->posicion_output[INDEX_THETA]);
    float vx_cos,vx_sin,vy_cos,vy_sin;
    vx_cos=cont->vel_robot_output[INDEX_X]*arm_cos_f32(theta_en_rango);
    vx_sin=cont->vel_robot_output[INDEX_X]*arm_sin_f32(theta_en_rango);
    vy_cos=cont->vel_robot_output[INDEX_Y]*arm_cos_f32(theta_en_rango);
    vy_sin=cont->vel_robot_output[INDEX_Y]*arm_sin_f32(theta_en_rango);
    cont->vel_fixed_output[INDEX_X]=vx_cos-vy_sin;
    cont->vel_fixed_output[INDEX_Y]=vx_sin + vy_cos;
    cont->vel_fixed_output[INDEX_THETA]=cont->vel_robot_output[INDEX_THETA];
    cont->posicion_output_enc[INDEX_X] = cont->posicion_output[INDEX_X] + cont-
>vel_fixed_output[INDEX_X]*TS;
    cont->posicion_output_enc[INDEX_Y] = cont->posicion_output[INDEX_Y] + cont-
>vel_fixed_output[INDEX_Y]*TS;
    cont->posicion_output_enc[INDEX_THETA]= cont->posicion_output[INDEX_THETA] +
cont->vel_robot_output[INDEX_THETA]*TS;
    // cont->posicion_output_enc[INDEX_THETA]=pasar_rango_menos2pi_2pi(cont-
>posicion_output_enc[INDEX_THETA]);
    pasar_ang_imu_a_sistema(cont);
    cont->us_med_valid=obtener_ang_us(cont);
    if(cont->kalman==ACTIVADO){
    // float ang_k_k_m1=cont->posicion_output[INDEX_THETA] + cont-
>vel_robot_output[INDEX_THETA]*TS;
    cont->p_k_k_m1=cont->p_k+cont->q;
    cont->k=cont->p_k_k_m1/(cont->p_k_k_m1+cont->r);
    float ang_medido;
    if(cont->us_hab==ACTIVADO && cont->us_med_valid==CONTROLADOR_OK &&
fabsf(cont->angulo_us - cont->angulo_imu)<=0.3){
    // ang_medido=cont->angulo_imu + cont->angulo_us;

```

```

//          ang_medido/=2.0;
//          filtro pasa bajo para us
float x_us=cont->angulo_us;
float y_us = B_low[0]*x_us + state_us[0];
state_us[0]=B_low[1]*x_us - y_us*A_r[1] + state_us[1];
state_us[1]=B_low[2]*x_us - y_us*A_r[2];
float x_imu=cont->angulo_imu;
float y_imu = B_high[0]*x_imu + state_imu[0];
state_imu[0]=B_high[1]*x_imu - y_imu*A_r[1] + state_imu[1];
state_imu[1]=B_high[2]*x_imu - y_imu*A_r[2];
ang_medido=y_imu+y_us;
cont->posicion_output[INDEX_Y]=0.92*cont-
>posicion_output_enc[INDEX_Y] - 0.08*cont->dist_us;
    }
    else{
        ang_medido=cont->angulo_imu;
        cont->posicion_output[INDEX_Y]=cont-
>posicion_output_enc[INDEX_Y];
    }
    cont->posicion_output[INDEX_THETA]=cont-
>posicion_output_enc[INDEX_THETA]+cont->k*(ang_medido-cont-
>posicion_output_enc[INDEX_THETA]);
    cont->p_k=(1-cont->k)*cont->p_k_k_m1;
}
else{
    cont->posicion_output[INDEX_THETA]=cont-
>posicion_output_enc[INDEX_THETA];
    cont->posicion_output[INDEX_Y]=cont->posicion_output_enc[INDEX_Y];
}
cont->posicion_output[INDEX_X]=cont->posicion_output_enc[INDEX_X];
if(cont->enable==DESACTIVADO) {
//          com_bp_moverse(cont->bp, CONT_VELOCIDAD, ADELANTE, 0);
return CONTROLADOR_OK;
}
theta_en_rango=pasar_rango_0_2pi(cont->posicion_output[INDEX_THETA]);
cont->pid[0].set_point=cont->setpoint[0];
cont->pid[1].set_point=cont->setpoint[1];
cont->pid[2].set_point=cont->setpoint[2];
fixed_frame_to_robot_frame[1][1]=arm_cos_f32(theta_en_rango);
fixed_frame_to_robot_frame[1][2]=arm_sin_f32(theta_en_rango);
fixed_frame_to_robot_frame[2][1]=-fixed_frame_to_robot_frame[1][2];
fixed_frame_to_robot_frame[2][2]=fixed_frame_to_robot_frame[1][1];

//
//          float dif=cont->setpoint[INDEX_THETA]-cont->posicion_output[INDEX_THETA];
//          if(dif>3.14159){
//              cont->posicion_output[INDEX_THETA]-=2*3.14159;
//          }
//          else if(dif<-3.14159){
//              cont->posicion_output[INDEX_THETA]+=2*3.14159;
//          }
//          if(cont->controlar==(CONTROLAR_POSICION)){
//              if((fabsf(cont->posicion_output[INDEX_THETA] - cont-
>setpoint[INDEX_THETA])>=DELTA_ADMISIBLE_THETA)
//                  || (fabsf(cont->posicion_output[INDEX_X] - cont-
>setpoint[INDEX_X])>=DELTA_ADMISIBLE_X)
//                  || (fabsf(cont->posicion_output[INDEX_Y] - cont-
>setpoint[INDEX_Y])>=DELTA_ADMISIBLE_Y)){
//                  cont->posicion_input[0]=pid_handler(&cont->pid[0], cont-
>posicion_output[0]);

```

```

        cont->posicion_input[1]=pid_handler(&cont->pid[1], cont-
>posicion_output[1]);
        cont->posicion_input[2]=pid_handler(&cont->pid[2], cont-
>posicion_output[2]);
        arm_mat_mult_f32(&fixed_frame_to_robot_frame_instance, &cont-
>posicion_input_instance, &robot_frame_instance);
        arm_mat_mult_f32(&robot_frame_to_vel_wheel_instance,
&robot_frame_instance, &cont->u_input_instance);
    }
    else{
        cont->u_input[0]=0.0;
        cont->u_input[1]=0.0;
        cont->u_input[2]=0.0;
        cont->u_input[3]=0.0;
    }
}
else if (cont->controlar==(CONTROLAR_VELOCIDAD)){
    cont->vel_fixed_input[0]=pid_handler(&cont->pid[0], cont-
>vel_fixed_output[0]);
    cont->vel_fixed_input[1]=pid_handler(&cont->pid[1], cont-
>vel_fixed_output[1]);
    cont->vel_fixed_input[2]=pid_handler(&cont->pid[2], cont-
>vel_fixed_output[2]);
    arm_mat_mult_f32(&fixed_frame_to_robot_frame_instance, &cont-
>vel_fixed_input_instance, &robot_frame_instance);
    arm_mat_mult_f32(&robot_frame_to_vel_wheel_instance,
&robot_frame_instance, &cont->u_input_instance);
}
else{
    cont->vel_robot_input[0]=pid_handler(&cont->pid[0], cont-
>vel_robot_output[0]);
    cont->vel_robot_input[1]=pid_handler(&cont->pid[1], cont-
>vel_robot_output[1]);
    cont->vel_robot_input[2]=pid_handler(&cont->pid[2], cont-
>vel_robot_output[2]);
    arm_mat_mult_f32(&robot_frame_to_vel_wheel_instance, &cont-
>vel_robot_input_instance, &cont->u_input_instance);
}
arm_mat_scale_f32(&cont->u_input_instance, 1.0/(R_WHEEL*2*3.14159), &cont-
>u_input_instance);
float porcen;
for(int i=0;i<4;i++){
    if(cont->u_input[i]>cont->vel_max){
//        cont->u_input[i]=1.0;
        porcen=cont->vel_max/cont->u_input[i];
        cont->u_input[0]*=porcen;
        cont->u_input[1]*=porcen;
        cont->u_input[2]*=porcen;
        cont->u_input[3]*=porcen;
    }
    else if(cont->u_input[i]<-cont->vel_max){
//        cont->u_input[i]=-1.0;
        porcen=-cont->vel_max/cont->u_input[i];
        cont->u_input[0]*=porcen;
        cont->u_input[1]*=porcen;
        cont->u_input[2]*=porcen;
        cont->u_input[3]*=porcen;
    }
}
}

```

```

    float e_wheel=cont->u_output[INDEX_MOTOR_DD] + cont-
>u_output[INDEX_MOTOR_DI] -
                                cont->u_output[INDEX_MOTOR_TD] - cont-
>u_output[INDEX_MOTOR_TI];
    if(e_wheel<0) e_wheel*=-1;
    cont->u_input[INDEX_MOTOR_DD] -= cont->coef_wheel*e_wheel;
    cont->u_input[INDEX_MOTOR_DI] -= cont->coef_wheel*e_wheel;
    cont->u_input[INDEX_MOTOR_TD] -= cont->coef_wheel*e_wheel;
    cont->u_input[INDEX_MOTOR_TI] -= cont->coef_wheel*e_wheel;
    float s_wheel_i=0;
    float s_wheel_o=0;
    float d_wheel_i_o=s_wheel_i-s_wheel_o;
    cont->u_input[INDEX_MOTOR_DD] -= cont->coef_d_wheel*d_wheel_i_o;
    cont->u_input[INDEX_MOTOR_DI] -= cont->coef_d_wheel*d_wheel_i_o;
    cont->u_input[INDEX_MOTOR_TD] -= cont->coef_d_wheel*d_wheel_i_o;
    cont->u_input[INDEX_MOTOR_TI] -= cont->coef_d_wheel*d_wheel_i_o;
    for(int i=0;i<4;i++){
        if(cont->u_input[i]<cont->dead_zone && cont->u_input[i]>-cont-
>dead_zone){
            cont->u_input[i]=0;
        }
    }
    if (cont->u_input[0]>0) {
        for(int i=0; i<4; i++){
            s_wheel_i+=abs_f(cont->u_input[i]);
            s_wheel_o+=abs_f(cont->u_output[i]);
        }
    }
    for(int i=0;i<4;i++){
        if(cont->u_input[i]==0){
            if(i==INDEX_MOTOR_DD){
                cont->u_input[INDEX_MOTOR_DI]=cont-
>u_input[INDEX_MOTOR_TD]+ cont->u_input[INDEX_MOTOR_TI];
            }
            if(i==INDEX_MOTOR_DI){
                cont->u_input[INDEX_MOTOR_DD]=cont-
>u_input[INDEX_MOTOR_TD]+ cont->u_input[INDEX_MOTOR_TI];
            }
            if(i==INDEX_MOTOR_TD){
                cont->u_input[INDEX_MOTOR_DI]=cont-
>u_input[INDEX_MOTOR_DD]+ cont->u_input[INDEX_MOTOR_TI];
            }
            if(i==INDEX_MOTOR_TI){
                cont->u_input[INDEX_MOTOR_TD]=cont-
>u_input[INDEX_MOTOR_DD]+ cont->u_input[INDEX_MOTOR_DI];
            }
        }
    }
    VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
ABSOLUTO, MOTOR_DD, cont->u_input[INDEX_MOTOR_DD]));
    VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
ABSOLUTO, MOTOR_DI, cont->u_input[INDEX_MOTOR_DI]));
    VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
ABSOLUTO, MOTOR_TD, cont->u_input[INDEX_MOTOR_TD]));
    VERIFICAR(status, com_bp_set_setpoint_a_motor(cont->bp, CONT_VELOCIDAD,
ABSOLUTO, MOTOR_TI, cont->u_input[INDEX_MOTOR_TI]));
    //
    //     cont->cuentas_ant_output[0]=cont->cuentas_output[0];
    //     cont->cuentas_ant_output[1]=cont->cuentas_output[1];

```

```
//     cont->cuentas_ant_output[2]=cont->cuentas_output[2];
//     cont->cuentas_ant_output[3]=cont->cuentas_output[3];
return CONTROLADOR_OK;
}
```

A continuación, se muestra el código de la librería de pathfinding.

```
#include"destino.h"
#include"string.h"
#define MISMO_DESTINO_Y_ORIGEN        NUMERO_DE_ESTACIONES+1
#define TRAYECTO_OK                   NUMERO_DE_ESTACIONES+2
#define TRAYECTO_ERROR                NUMERO_DE_ESTACIONES+3
#define NO_HAY_RECORRIDO               NUMERO_DE_ESTACIONES+4
#define NO_EXISTE_ESTACION             NUMERO_DE_ESTACIONES+5
#define EXISTE_ESTACION                NUMERO_DE_ESTACIONES+7
typedef struct {
    int recorrido[NUMERO_DE_ESTACIONES];
    int estaciones_restantes;
    int estacion_de_partida;
    int estacion_actual;
    int estacion_siguiente;
    int estacion_anterior;
    int indice_recorrido;
    int destino;
    float orientacion;
    float giro_a_realizar;
}recorrido;
void pf_init(recorrido* r, int estacion_actual, float orientacion);
int pf_definir_recorrido(recorrido* r, int partida, int destino);
int pf_siguiente_estacion(recorrido* r);
int string_a_id_estacion(char* est);
int id_a_string_estacion(int id, char*est);
float orientacion_de_tramo_pared(recorrido* r);
#define PRECESOR(origen, destino)      precesor_mapa[origen][destino]
#define DISTANCIA(origen, destino)     distancia_mapa[origen][destino]
#define SET_ESTACION_ACTUAL(recorrido, estacion)  recorrido->
estacion_actual=estacion
#define ORIENTACION_DE_TRAMO(origen, destino)
orientacion_mapa[origen][destino]
#define UBICACION_DE_PARED(origen, destino)
ubicacion_de_pared_en_mapa[origen][destino]
/*
* pathfinding.c
*
* Created on: 11 sep. 2023
* Author: Gonzalo Mansilla
*/
#include"pathfinding.h"
void pf_init(recorrido* r, int estacion_actual, float orientacion){
    r->estacion_actual=estacion_actual;
    r->indice_recorrido=0;
    r->estaciones_restantes=0;
    r->orientacion=orientacion;
    r->giro_a_realizar=0;
//     r->estacion_actual=estacion_actual;
return;
}
int pf_definir_recorrido(recorrido* r, int partida,int destino){
    if(partida==destino){
```

```

        return MISMO_DESTINO_Y_ORIGEN;
    }
    if(destino<0 || partida<0 || partida>=NUMERO_DE_ESTACIONES ||
destino>=NUMERO_DE_ESTACIONES) return TRAYECTO_ERROR;
    r->estacion_actual=partida;
    r->destino=destino;
    r->indice_recorrido=0;
    r->recorrido[r->indice_recorrido]=destino;
    r->estaciones_restantes=1;
    r->estacion_de_partida=r->estacion_actual;
    while(r->recorrido[r->indice_recorrido]!=r->estacion_actual){
        r->recorrido[r->indice_recorrido+1]=PRECESOR(r->estacion_actual,r-
>recorrido[r->indice_recorrido]);
        if(r->recorrido[r->indice_recorrido+1]==r->estacion_actual) break;
        r->estaciones_restantes++;
        r->indice_recorrido++;
        if(r->indice_recorrido>NUMERO_DE_ESTACIONES){
            return TRAYECTO_ERROR;
        }
    }
    return TRAYECTO_OK;
}
int pf_siguiete_estacion(recorrido* r){
    if(r->estaciones_restantes==0 || r->indice_recorrido<0){ return
NO_HAY_RECORRIDO;}
    r->estacion_siguiete=r->recorrido[r->indice_recorrido];
    r->indice_recorrido--;
    r->estaciones_restantes--;
    r->giro_a_realizar= ORIENTACION_DE_TRAMO(r->estacion_actual,r-
>estacion_siguiete);
    r->giro_a_realizar-= r->orientacion;
    if(r->giro_a_realizar>360) r->giro_a_realizar-=360;
    else if(r->giro_a_realizar<=-360) r->giro_a_realizar+=360;
    return r->estacion_siguiete;
}
int string_a_id_estacion(char* est){
    for(int i=0; i<NUMERO_DE_ESTACIONES;i++){
        if(!strcmp(est, mapa[i].nombre)){
            return mapa[i].id;
        }
    }
    return NO_EXISTE_ESTACION;
}
int id_a_string_estacion(int id, char*est){
    for(int i=0; i<NUMERO_DE_ESTACIONES;i++){
        if(id==mapa[i].id){
            strcpy(est,mapa[i].nombre);
            return EXISTE_ESTACION;
        }
    }
    return NO_EXISTE_ESTACION;
}
float orientacion_de_tramo_pared(recorrido* r){
    float orientacion=ORIENTACION_DE_TRAMO(r->estacion_actual,r-
>estacion_siguiete);
    if(UBICACION_DE_PARED(r->estacion_actual,r->estacion_siguiete)==IZQUIERDA){
        orientacion+=180;
    }
    return orientacion;
}

```

}

16. ANEXO III

- Placa de control principal:



STM32F4DISCOVERY

Data brief

Discovery kit with STM32F407VG MCU



Picture is not contractual.

Product status link

STM32F4DISCOVERY

Features

- STM32F407VGT6 microcontroller featuring 32-bit Arm® Cortex®-M4 with FPU core, 1-Mbyte Flash memory and 192-Kbyte RAM in an LQFP100 package
- USB OTG FS
- ST MEMS 3-axis accelerometer
- ST-MEMS audio sensor omni-directional digital microphone
- Audio DAC with integrated class D speaker driver
- User and reset push-buttons
- Eight LEDs:
 - LD1 (red/green) for USB communication
 - LD2 (red) for 3.3 V power on
 - Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
 - Two USB OTG LEDs, LD7 (green) V_{BUS} and LD8 (red) over-current
- Board connectors:
 - USB with Micro-AB
 - Stereo headphone output jack
 - 2.54 mm pitch extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing
- Flexible power-supply options: ST-LINK, USB V_{BUS} , or external sources
- External application power supply: 3 V and 5 V
- Comprehensive free software including a variety of examples, part of STM32CubeF4 MCU Package, or STSW-STM32068 for using legacy standard libraries
- On-board ST-LINK/V2-A debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE

Description

The STM32F4DISCOVERY Discovery kit leverages the capabilities of the STM32F407 high-performance microcontrollers, to allow users to develop audio applications easily. It includes an ST-LINK/V2-A embedded debug tool, one ST-MEMS digital accelerometer, one digital microphone, one audio DAC with integrated class D speaker driver, LEDs, push-buttons, and a USB OTG Micro-AB connector.

Specialized add-on boards can be connected by means of the extension header connectors.



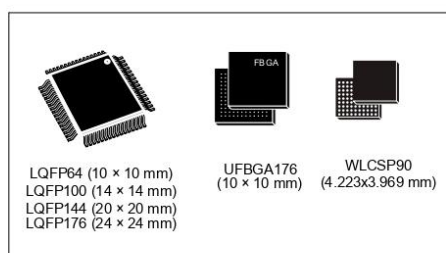
STM32F405xx STM32F407xx

Arm® Cortex®-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM,
USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera

Datasheet - production data

Features

- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
 - Up to 1 Mbyte of Flash memory
 - Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
 - 512 bytes of OTP memory
 - Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR and NAND memories
- LCD parallel interface, 8080/6800 modes
- Clock, reset and supply management
 - 1.8 V to 3.6 V application supply and I/Os
 - POR, PDR, PVD and BOR
 - 4-to-26 MHz crystal oscillator
 - Internal 16 MHz factory-trimmed RC (1% accuracy)
 - 32 kHz oscillator for RTC with calibration
 - Internal 32 kHz RC with calibration
- Low-power operation
 - Sleep, Stop and Standby modes
 - V_{BAT} supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS A/D converters: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 168 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
- Debug mode
 - Serial wire debug (SWD) & JTAG interfaces
 - Cortex-M4 Embedded Trace Macrocell™
- Up to 140 I/O ports with interrupt capability
 - Up to 136 fast I/Os up to 84 MHz
 - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
 - Up to 3 × I²C interfaces (SMBus/PMBus)
 - Up to 4 USARTs/2 UARTs (10.5 Mbit/s, ISO 7816 interface, LIN, IrDA, modem control)
 - Up to 3 SPIs (42 Mbits/s), 2 with muxed full-duplex I²S to achieve audio class accuracy via internal audio PLL or external clock
 - 2 × CAN interfaces (2.0B Active)
 - SDIO interface
- Advanced connectivity
 - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
 - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
 - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII



- **Placas de bajo nivel:**



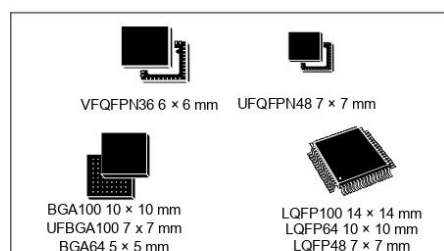
STM32F103x8 STM32F103xB

Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces

Datasheet - production data

Features

- Arm® 32-bit Cortex®-M3 CPU core
 - 72 MHz maximum frequency, 1.25 DMIPS / MHz (Dhrystone 2.1) performance at 0 wait state memory access
 - Single-cycle multiplication and hardware division
- Memories
 - 64 or 128 Kbytes of Flash memory
 - 20 Kbytes of SRAM
- Clock, reset and supply management
 - 2.0 to 3.6 V application supply and I/Os
 - POR, PDR, and programmable voltage detector (PVD)
 - 4 to 16 MHz crystal oscillator
 - Internal 8 MHz factory-trimmed RC
 - Internal 40 kHz RC
 - PLL for CPU clock
 - 32 kHz oscillator for RTC with calibration
- Low-power
 - Sleep, Stop and Standby modes
 - V_{BAT} supply for RTC and backup registers
- 2x 12-bit, 1 µs A/D converters (up to 16 channels)
 - Conversion range: 0 to 3.6 V
 - Dual-sample and hold capability
 - Temperature sensor
- DMA
 - 7-channel DMA controller
 - Peripherals supported: timers, ADC, SPIs, I²Cs and USARTs
- Up to 80 fast I/O ports
 - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant



- Debug mode
 - Serial wire debug (SWD) and JTAG interfaces
- Seven timers
 - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
 - 16-bit, motor control PWM timer with dead-time generation and emergency stop
 - Two watchdog timers (independent and window)
 - SysTick timer 24-bit downcounter
- Up to nine communication interfaces
 - Up to two I²C interfaces (SMBus/PMBus®)
 - Up to three USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
 - Up to two SPIs (18 Mbit/s)
 - CAN interface (2.0B Active)
 - USB 2.0 full-speed interface
- CRC calculation unit, 96-bit unique ID
- Packages are ECOPACK®

Table 1. Device summary

Reference	Part number
STM32F103x8	STM32F103C8, STM32F103R8 STM32F103V8, STM32F103T8
STM32F103xB	STM32F103RB STM32F103VB, STM32F103CB, STM32F103TB

- Comunicación en red RS485:

Product
FolderOrder
NowTechnical
DocumentsTools &
SoftwareSupport &
Community**SN75176A**

SLLS100B –JUNE 1984–REVISED JANUARY 2015

SN75176A Differential Bus Transceiver

1 Features

- Bidirectional Transceiver
- Meets or Exceeds the Requirements of ANSI Standards EIA/TIA-422-B and ITU Recommendations V.11
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Driver and Receiver Outputs
- Individual Driver and Receiver Enables
- Wide Positive and Negative Input/Output Bus Voltage Ranges
- Driver Output Capability ± 60 mA Max
- Thermal-Shutdown Protection
- Driver Positive-Current Limiting and Negative-Current Limiting
- Receiver Input Impedance 12 k Ω Min
- Receiver Input Sensitivity ± 200 mV
- Receiver Input Hysteresis 50 mV Typ
- Operates From Single 5-V Supply
- Lower Power Requirements

2 Applications

- Low Speed RS485 communication (5 Mbps or less)
- For 10 Mbps, use SN75176B

3 Description

The SN75176A differential bus transceiver is a monolithic integrated circuit designed for bidirectional data communication on multipoint bus-transmission lines. It is designed for balanced transmission lines and meets ANSI Standard EIA/TIA-422-B and ITU Recommendation V.11.

The SN75176A combines a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be externally connected together to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus whenever the driver is disabled or $V_{CC} = 0$. These ports feature wide positive and negative common-mode voltage ranges making the device suitable for party-line applications.

The driver is designed to handle loads up to 60 mA of sink or source current. The driver features positive- and negative-current limiting and thermal shutdown for protection from line fault conditions. Thermal shutdown is designed to occur at a junction temperature of approximately 150°C. The receiver features a minimum input impedance of 12 k Ω , an input sensitivity of ± 200 mV, and a typical input hysteresis of 50 mV.

The SN75176A can be used in transmission-line applications employing the SN75172 and SN75174 quadruple differential line drivers and SN75173 and SN75175 quadruple differential line receivers.

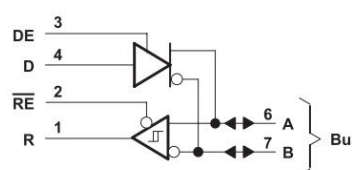
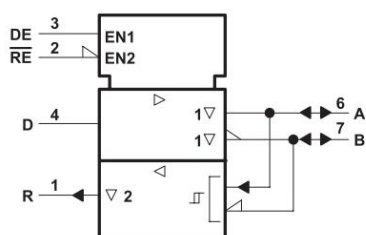
The SN75176A is characterized for operation from 0°C to 70°C.

Device Information⁽¹⁾

PART NUMBER	PACKAGE (PIN)	BODY SIZE (NOM)
SN75176A	SOIC (8)	4.90 mm $\times$ 3.91 mm
	PDIP (8)	9.81 mm $\times$ 6.35 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

Simplified Schematics



An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

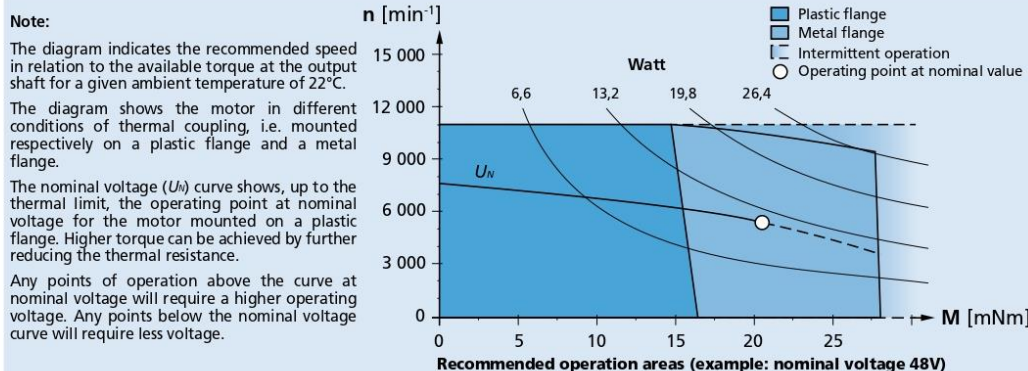
- **Motores:**
DC-Micromotors
 Graphite Commutation

 20,5 mNm
 21,7 W

Series 2342 ... CR

Values at 22°C and nominal voltage	2342 S	006 CR	012 CR	018 CR	024 CR	036 CR	048 CR	
Nominal voltage	U_N	6	12	18	24	36	48	V
Terminal resistance	R	0,362	1,93	4,14	7,14	15,9	31,2	Ω
Rotor inductance	L	13,1	69,1	142	264	569	1 130	μH
Efficiency, max.	η_{max}	75	77	78	79	79	80	%
No-load current, typ.	I_0	0,157	0,0694	0,0479	0,0351	0,024	0,0163	A
No-load speed	n_0	8 310	7 530	7 990	7 870	8 110	7 690	min^{-1}
Stall torque	M_H	95,6	86,1	87,7	93,2	92,9	89	mNm
Rotor inertia	J	5,6	5,7	6,2	5,8	6,5	6	gcm^2
Friction torque	M_R	0,98	1	0,99	0,99	0,99	0,95	mNm
Torque constant	k_M	6,36	14,6	21	28,6	42	59,1	mNm/A
Speed constant	k_n	1 500	653	455	334	228	162	min^{-1}/V
Slope of n-M curve	$\Delta n/\Delta M$	85,5	86,2	89,8	83,3	86,1	85,3	$\text{min}^{-1}/\text{mNm}$
Thermal resistance:								
– winding to housing	$R_{\text{th}1}$	5,6						K/W
– housing to ambient (external plastic flange)	$R_{\text{th}2p}$	15						K/W
– housing to ambient (external metal flange)	$R_{\text{th}2m}$	2,1						K/W
Thermal time constant:								
– winding	τ_{w1}	12						s
– housing (external plastic flange)	τ_{w2p}	580						s
– housing (external metal flange)	τ_{w2m}	78						s
Operating temperature range:								
– motor		-30 ... +100						°C
– winding, max. permissible		+125						°C
Shaft bearings		ball bearings, preloaded						
Shaft diameter		3						mm
Radial shaft load max.:								
– dynamic at 3 000 min^{-1} (3 mm from bearing)		20						N
Axial shaft load max.:								
– dynamic at 3 000 min^{-1}		2						N
– static (shaft unsupported)		20						N
– static (shaft supported)		900						N
Shaft play, max.:								
– radial		0,015						mm
– axial		0						mm
Speed up to	n_{max}	11 000						min^{-1}
Number of pole pairs		1						
Mass		88						g
Housing material		steel, nickel plated						
Magnet material		NdFeB						
Rated values for continuous operation								
Rated torque	M_N	16,6	19,8	19,6	20,5	20,2	20,5	mNm
Rated current (thermal limit)	I_N	3	1,6	1,11	0,846	0,57	0,409	A
Rated speed	n_N	6 700	5 270	5 690	5 660	5 850	5 390	min^{-1}

Note: Rated values are calculated with nominal voltage and at a 22°C ambient temperature. The $R_{\text{th}2}$ value has been reduced by 50%.



For notes on technical data and lifetime performance refer to "Technical Information".

Edition 2023 Nov. 22

© DR. FRITZ FAULHABER GMBH & CO. KG
 Specifications subject to change without notice.
www.faulhaber.com

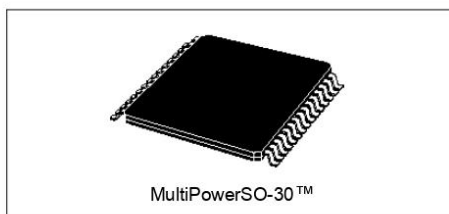
- **Driver motor DC:**



VNH2SP30-E

Automotive fully integrated H-bridge motor driver

Datasheet - production data



Features

Type	$R_{DS(on)}$	I_{out}	V_{CCmax}
VNH2SP30-E	19 mΩ max (per leg)	30 A	41 V

- AEC-Q100 qualified
- 5 V logic level compatible inputs
- Undervoltage and overvoltage shutdown
- Overvoltage clamp
- Thermal shutdown
- Cross-conduction protection
- Linear current limiter
- Very low standby power consumption
- PWM operation up to 20 kHz
- Protection against loss of ground and loss of V_{CC}
- Current sense output proportional to motor current
- Package: ECOPACK®



Description

The VNH2SP30-E is a full bridge motor driver intended for a wide range of automotive applications. The device incorporates a dual monolithic high side driver and two low side switches. The high side driver switch is designed

using STMicroelectronics well known and proven proprietary VIPower™ M0 technology which permits efficient integration on the same die of a true power MOSFET with intelligent signal/protection circuitry.

The low side switches are vertical MOSFETs manufactured using STMicroelectronics proprietary EHD (STripFET™) process. The three die are assembled in a MultiPowerSO-30 package on electrically isolated leadframes. This package, specifically designed for the harsh automotive environments, offers improved thermal performance thanks to exposed die pads. Moreover, its fully symmetrical mechanical design allows superior manufacturability at board level. The input signals IN_A and IN_B can directly interface with the microcontroller to select the motor direction and brake condition. The $DIAG_A/EN_A$ or $DIAG_B/EN_B$, when connected to an external pull-up resistor, enable one leg of the bridge. They also provide a feedback digital diagnostic signal. The normal operating condition is explained in the truth table. The motor current can be monitored with the CS pin by delivering a current proportional to its value. The speed of the motor can be controlled in all possible conditions by the PWM up to 20 kHz. In all cases, a low level state on the PWM pin will turn off both the LS_A and LS_B switches. When PWM rises to a high level, LS_A or LS_B turn on again depending on the input pin state.

Table 1. Device summary

Package	Order code
	Tape and reel
MultiPowerSO-30	VNH2SP30TR-E

- **Sensor de ultrasonido:**



Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

- Sensor inercial (IMU):

	MPU-9250 Product Specification	Document Number: PS-MPU-9250A-01 Revision: 1.1 Release Date: 06/20/2016
---	---------------------------------------	---

1.2 Purpose and Scope

This document provides a description, specifications, and design related information on the MPU-9250 MotionTracking device. The device is housed in a small 3x3x1mm QFN package.

Specifications are subject to change without notice. Final specifications will be updated based upon characterization of production silicon. For references to register map and descriptions of individual registers, please refer to the MPU-9250 Register Map and Register Descriptions document.

1.3 Product Overview

MPU-9250 is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8963 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Hence, the MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor™ (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU-6515. With its dedicated I²C sensor bus, the MPU-9250 directly provides complete 9-axis MotionFusion™ output. The MPU-9250 MotionTracking device, with its 9-axis integration, on-chip MotionFusion™, and run-time calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices, guaranteeing optimal motion performance for consumers. MPU-9250 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I²C port.

MPU-9250 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ± 250 , ± 500 , ± 1000 , and $\pm 2000^\circ/\text{sec}$ (dps), a user-programmable accelerometer full-scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$, and a magnetometer full-scale range of $\pm 4800\mu\text{T}$.

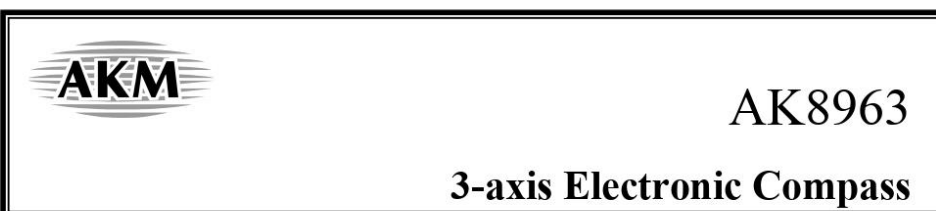
Other industry-leading features include programmable digital filters, a precision clock with 1% drift from -40°C to 85°C , an embedded temperature sensor, and programmable interrupts. The device features I²C and SPI serial interfaces, a VDD operating range of 2.4V to 3.6V, and a separate digital IO supply, VDDIO from 1.71V to VDD.

Communication with all registers of the device is performed using either I²C at 400kHz or SPI at 1MHz. For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz.

By leveraging its patented and volume-proven CMOS-MEMS fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the package size down to a footprint and thickness of 3x3x1mm, to provide a very small yet high performance low cost package. The device provides high robustness by supporting 10,000g shock reliability.

1.4 Applications

- Location based services, points of interest, and dead reckoning
- Handset and portable gaming
- Motion-based game controllers
- 3D remote controls for Internet connected DTVs and set top boxes, 3D mice
- Wearable sensors for health, fitness and sports



1. Features

A 3-axis electronic compass IC with high sensitive Hall sensor technology.
Best adapted to pedestrian city navigation use for cell phone and other portable appliance.

Functions:

- 3-axis magnetometer device suitable for compass application
- Built-in A to D Converter for magnetometer data out
- 14-/16-bit selectable data out for each 3 axis magnetic components
 - Sensitivity: 0.6 μ T/LSB typ. (14-bit)
 - 0.15 μ T/LSB typ. (16-bit)
- Serial interface
 - I²C bus interface.
 - Standard mode and Fast mode compliant with Philips I²C specification Ver.2.1
 - 4-wire SPI
- Operation modes:
 - Power-down, Single measurement, Continuous measurement, External trigger measurement, Self test and Fuse ROM access.
- DRDY function for measurement data ready
- Magnetic sensor overflow monitor function
- Built-in oscillator for internal clock source
- Power on Reset circuit
- Self test function with built-in internal magnetic source

Operating temperatures:

- -30°C to +85°C

Operating supply voltage:

- Analog power supply +2.4V to +3.6V
- Digital Interface supply +1.65V to analog power supply voltage.

Current consumption:

- Power-down: 3 μ A typ.
- Measurement:
 - Average power consumption at 8 Hz repetition rate: 280 μ A typ.

Package:

- | | | |
|---------|----------------------|--|
| AK8963C | 14-pin WL-CSP (BGA): | 1.6 mm $\times$ 1.6 mm $\times$ 0.5 mm (typ.) |
| AK8963N | 16-pin QFN package: | 3.0 mm $\times$ 3.0 mm $\times$ 0.75 mm (typ.) |

1. Introduction of Module

1.1 Introduction

MG65 Bar code reader module is a high performance scanner, can read 1D bar code easily and read 2D bar code with high speed. It also wins high scan speed for linear code, even for bar code on paper or screen.

MG65 bar code reader module is an advanced bar code decoding algorithm which developed on image recognition algorithm, can easily and accurately read bar code, simplify secondary development.

MG65 works stable in dark and large temperature range.



1.2 Technical Specification

Default scan mode		Continuous scan	
Read code time for once		3s	Parameter: 0.1-25.5s; step-size: 0.1s;0 means no time limited
Reading interval		1S	Parameter: 0.1-25.5s; step-size: 0.1s;0 means no time limited
Output		GBK	GBK、UNICODE、BIG5
Interface		USB	USB 、UART、USB VCom
Interface (TTL-232)	Serial Baud Rate	9600	adjustable, details at 2.1
	Verification	N	
	Data bit	8	
	Stop bit	1	
	CTSRTS	No	
serial mode	Read code time for once	5s	Parameter: 0.1-25.5s; step-size: 0.1s;0 means no time limited

17. BIBLIOGRAFÍA

- Kevin M. Lynch and Frank C. Park, “Modern Robotics: Mechanics, Planning, and Control”, Cambridge University Press, 2017.
- Åström, Karl J., and Tore Hägglund, “Advanced PID Control”, International Society of Automation, 2006.
- Taheri, Hamid, Bing Qiao, and Nurallah Ghaeminezhad, "Kinematic model of a four mecanum wheeled mobile robot.", International journal of computer applications, Vol.113, Nro 3, pp 6-9, March 2015.
- Katsuhiko Ogata, “Ingeniería de control moderna, Quinta edición”, Prentice Hall, 2013.
- Katsuhiko Ogata, “Sistemas de control en tiempo discreto, Segunda edición”, Prentice Hall & IBD, 2013.
- Bishop Gary and Welch Greg, “An introduction to the kalman filter”, UNC-Chapel Hill, TR 95-041, July 2006.
- Ariffin, N. H., Arsad, N., & Bais, B, “Low cost MEMS gyroscope and accelerometer implementation without Kalman Filter for angle estimation”, 2016 International Conference on Advances in Electrical, Electronic and Systems Engineering (ICAEEES), pp. 77-82. November 2016
- Kugelstadt, Thomas, “The RS485 Design Guide”, Texas Instrument, 2011.
- Kugelstadt, Thomas, “RS485 failsafe biasing: Old versus new transceiver”, Texas Instrument, 2013.
- Talat Ozyagcilar, “Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors”, Freescale semiconductor, Appl. Note AN4248, January 2012.
- ST Electronics, “Discovery kit with STM32F407VG MCU”, October 2020.(ANEXO I)
- ST Electronics, “STM32F407VG Datasheet”, August 2020.(ANEXO II)
- ST Electronics, “STM32F103C8T6 Datasheet”, March 2022.(ANEXO III)
- Texas Instrument, “SNx5176B Differential Bus Transceivers Datasheet”, December 2021.(ANEXO IV)
- Faulhaber, “DC - Micromotor Serie 2342...CR Datasheet”, November 2022.(ANEXO V)
- ST Electronics, “VNH2SP30-E Datasheet”, October 2018.(ANEXO VI)
- ElecFreak, “Ultrasonic Ranging Module HC - SR04”,(ANEXO VII)
- TDK InvenSense “MPU-9250 Product Specification”, June 2016.(ANEXO VIII)
- Asahi Kasei, “AK8983 Datasheet”, October 2013.(ANEXO IX)
- Hangzhou Grow Technology Co., Ltd, “GM65 Bar Code Reader Module User Manual”, September 2016.(ANEXO X)
- Raspberry Pi Ltd, “Raspberry Pi Pico Datasheet”, June 2023.(ANEXO XI)