# Trabajo Fin de Grado

Robot para prácticas de la asignatura Sistemas de Tiempo Real

Robot for practical training of the subject Real-Time Systems.

Autor/es

Javier Fernández Muñoz

Director/es

José Luis Villarroel Salcedo

Titulación del autor

Ingeniería electrónica y automática

Escuela de ingeniería y arquitectura (EINA)

2024

## Agradecimientos:

*A mis padres Alberto y Soledad, por no dejar que me rindiera.*

*A mi hermano Juan, por ser un ejemplo a seguir.*

*A Saúl, por ser el mejor aliado a lo largo de la carrera.*

*A mis abuelos Juan y Paulino, sin ellos no estaría escribiendo estas palabras.*

## Resumen:

¿Cuál es el límite? Es la pregunta que me he realizado estos meses mientras trabajaba en este proyecto.

En la asignatura de cuarto año del grado de ingeniería electrónica y automática, sistemas de tiempo real, nos enfocamos en sistemas de funcionamiento crítico. En su teoría, programación y práctica. El tiempo que abarca la asignatura sirve para poder entender el funcionamiento de este tipo de sistemas y mediante unos modestos robots de prácticas comprobar una aplicación sencilla de sus funciones.

En este trabajo se propone el análisis más profundo del robot, así como mejorar su control y utilidad, donde se aprovecha al máximo todas las posibilidades de las que dispone. ¿Se puede mejorar el funcionamiento de los sensores? Si las ruedas patinan, ¿Se puede compensar este deslizamiento? ¿Puede navegar de manera autónoma? ¿Cómo podemos mejorarlo?

Para ello, se parte de la base realizada en la asignatura donde se realiza una puesta a punto de todos los controladores o *drivers* del robot, para continuar con la ampliación de las aplicaciones más importantes.

Se propone un algoritmo nuevo y un controlador de posición para crear una aplicación demostrativa de navegación, así como la creación de una consola de manipulación nueva para facilitar el manejo y poder representar la odometría del robot en tiempo real.

Mediante el uso de aplicaciones como SYS/BIOS y UIA se realiza un análisis de tiempo real para verificar el cumplimiento de los requisitos de este tipo de sistemas.

El resultado final muestra una mejora general de la gran mayoría de los aspectos del robot planteados en la asignatura, todo ello para facilitar y ampliar su uso como herramienta de aprendizaje.

# Índice

## Contenido

# 1.Introducción:

## 1.1 Antecedentes

### 1.1.1 Asignatura Sistemas de Tiempo Real (STR)

El sistema ABS (*Antiblockiersystem*) de un automóvil se encarga de modificar la fuerza con la que se ejerce el frenado y conseguir que los neumáticos no deslicen. Es un tipo de sistema que debe funcionar con gran precisión dentro de un plazo de respuesta determinado. Este es un ejemplo de **sistema de tiempo real**.

*"A real-time system is one in which the correctness of the system depends not only on the logical result of computation, but also* ***on the time at which the results are generated****"*

(Stankovic, 1988).

Un sistema de tiempo real es un sistema informático diseñado para responder a eventos o entradas dentro de un intervalo de tiempo específico. La principal característica distintiva de un sistema de tiempo real es su necesidad de cumplir con restricciones temporales estrictas para garantizar su corrección y eficacia.

Siendo los ejemplos más significativos dentro del ámbito de la automoción, véase el antes mencionado ABS, también control de vuelo, posición… Gracias a la expansión de la electrónica cada vez son más demandados como su utilización en aparatos multimedia o medición de sensores o el caso de la **robótica** que engloba este trabajo.

La asignatura Sistema de tiempo real de cuarto año del grado de ingeniería electrónica y automática nos permite entender estos sistemas. A lo largo de esta asignatura estudiamos en profundidad los conceptos básicos y hacemos hincapié en el desarrollo de estos. Para ello, el contenido se reparte entre clases teóricas y prácticas o de laboratorio.

En las clases de contenido teórico se explica todo lo necesario para entender el funcionamiento de sistemas empotrados con requisitos de tiempo real:

La clave para el diseño y la implementación efectiva de sistemas de tiempo real radica en la capacidad de garantizar que las tareas críticas se completen dentro de los plazos requeridos, utilizando técnicas de programación, planificación y gestión de recursos específicas para cumplir con estas exigencias temporales.

los aspectos clave que debe presentar este tipo de sistemas para que funcionen correctamente son (Burns, A y Wellings, A. 2009):

**Planificación y gestión de tareas:** utilizan algoritmos de planificación específicos para asignar recursos de manera óptima y cumplir con los plazos requeridos. Algunos algoritmos comunes incluyen "*Rate Monotonic Scheduling"* (RMS) y *"Earliest Deadline First"* (EDF)

**Prioridades de Tareas:** Las tareas críticas tienen prioridades más altas para asegurar que se cumplan a tiempo.

**Gestión de Concurrencia y Sincronización:** Para asegurar la coherencia de los datos, se utilizan métodos de sincronización como son los semáforos o *mutex*. Es vital controlar los recursos que se comparten como la CPU o la memoria.

**Control de eventos y temporización:** Utilización de temporizadores para tener un control preciso del tiempo y poder programar las tareas y sus plazos. Las tareas requieren responder rápidamente a interrupciones y eventos externos para cumplir los plazos, lo que conlleva un manejo eficiente de las interrupciones.

**Garantía de tiempo:** para poder garantizar los plazos se realizan análisis exhaustivos donde las tareas críticas deben cumplir con estos plazos especificados.

Respecto al contenido práctico o de laboratorio, se dispone de unos robots móviles con los que se desarrolla un sistema de tiempo real. El contenido de las prácticas abarca desde el modelado de los componentes, diseño e implementación de reguladores y servos, programación del microcontrolador, hasta la puesta a punto y verificación de requisitos (Villarroel, José Luis y Abadía, David. 2023. 3).

En este caso se trabajará con un sistema con techo de prioridad, Es decir, con un protocolo de **herencia de prioridad**.

## 1.2 Objetivos

La idea de este trabajo final de grado es la de ampliar la configuración y aplicación del robot desarrollado en la asignatura. Englobando en cuatro puntos generales:

**Puesta a punto del sistema**: Mejorar todos los *drivers* que controlan los sensores del robot para poder conseguir nuevas funcionalidades y optimizar las existentes. La puesta a punto es la fase inicial y la más crucial de todas, ya que el resto de los objetivos siguientes dependen totalmente del funcionamiento de los *drivers* que se han modificado en esta fase.

**Desarrollo de aplicación demostrativa de navegación:** Conseguir que el robot pueda desenvolverse de manera autónoma en un entorno cerrado. El algoritmo de navegación será más depurado que el creado en las clases prácticas para seguimiento de muros. Además, se estudiarán los problemas recurrentes en movimiento y como se han solventado. Este objetivo es el más extenso ya que conlleva la gran mayoría de pruebas prácticas que se han realizado con el propio robot.

**Creación de nueva consola de telemanipulación:** Se diseña una nueva consola de telemanipulación con un diseño gráfico. Además, se representará la estimación de la posición del robot, es decir, **la odometría.** Junto a la posición, se representará una **nube de puntos** con los valores obtenidos de los sensores de ultrasonidos para crear un mapa de la zona.

De esta manera, se facilita y agiliza muchos de los procesos que se van a estudiar junto a la aplicación de navegación añadiendo herramientas extra en la consola.

**Análisis del tiempo real:** estudio de todos los aspectos clave de todo sistema de tiempo real ya explicados en el apartado 1.1.1 Asignatura Sistemas de tiempo real (STR).

Por último, se estudiará el uso de **nuevos sensores** que se podrían añadir al robot para mejorar su uso.

## 1.3 Estructura de la Memoria:

### 1.3.1 Diseño estructural

Con el siguiente diagrama (*figura 1*) se describe la organización seguida de los distintos objetivos mencionados en el apartado 1.2:

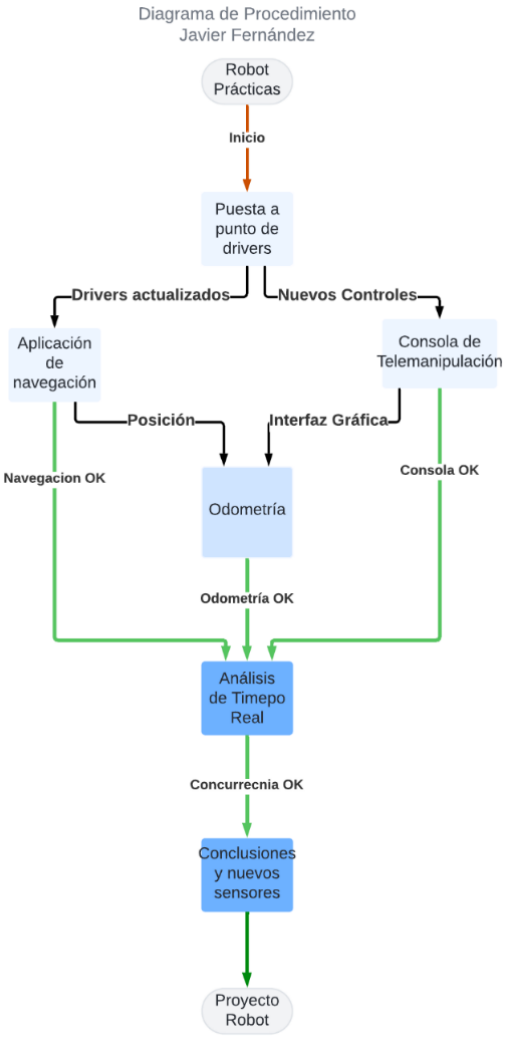



*Figura 1, Procedimiento.*

La estructura sigue un orden lógico de progresión. Para poder desarrollar la aplicación de navegación o la consola, primero es necesario **actualizar los *drivers***. Una vez el robot está preparado, se crea la aplicación de **navegación** y se diseña la nueva **consola de telemanipulación.**

Con la aplicación y la consola definidas, se calcula, **la odometría**. Una vez se hayan definido todos los objetivos referidos al uso del robot, se realiza el **análisis de tiempo real** del sistema y se realizan las conclusiones del trabajo.

### 1.3.2 Herramientas usadas

Para llevar a cabo las distintas fases del trabajo, se han usado diversos programas informáticos:

***Code composer studio (CCS):*** la tarjeta de desarrollo LAUNCHXL-F28377S que controla todos los procesos del robot, se programa mediante un ordenador externo con el programa de la empresa *Texas Instruments (TI).* Este programa es un entorno de desarrollo especifico de los controladores y procesadores de TI.

Incluye un compilador optimizado de C/C++ donde se crea el proyecto específico en un espacio de trabajo y así poder depurar y probar los programas compilados. De esta forma se controlan las distintas entradas y salidas de la placa, así como un entorno interactivo donde se pueden inspeccionar las distintas variables y sus tiempos de cómputo, indispensables para los sistemas de tiempo real.

También incluye los *drivers* necesarios para la propia placa, así como unos módulos especiales de SYS/BIOS y XDC *tools*.

**SYS/BIOS:** Es un módulo de tiempo real creado por TI, el cual está integrado en la herramienta de CCS. Es un software de código abierto que funciona con las plataformas de MSP430 de la misma marca. Entre sus principales características permite planificar según prioridades fijas y de herencia de prioridad. También dispone de herramientas de análisis de tiempo real como cálculo de tiempos de cómputo y gráficos de ejecución.

Gracias a SYS/BIOS podemos crear relojes que activen funciones sean esporádicas o periódicas. De esta manera, se definen interrupciones vía *software* o *hardware* por medio de rutinas con diferentes niveles de prioridad (hasta 16). También se definen tareas para las diferentes aplicaciones que se necesiten. Tanto las interrupciones como las tareas tienen determinada una prioridad. De esta manera una interrupción *hardware* es la más urgente, seguida de las tipo *software* y por último las tareas con los programas que se crean.

A lo largo de la memoria se explicarán las tareas creadas junto a su nivel de prioridad y sus interrupciones dependiendo del uso, así como de los servidores con herencia de prioridad.

Para poder sincronizar todas estas funciones, es necesario habilitar semáforos que controlan los pasos de unas a otras. (Villarroel, José Luis y Abadía, David. 2023. 4).

**UIA y XDC*tools*:** UIA (*Unified Instrumentation Architecture*), facilita el cálculo de tiempos de cómputo de las tareas y servidores del sistema de manera gráfica. Por otro lado, *XDCtools*, es una herramienta que facilita el uso de sistemas embebidos en *Code Composer Studio.*

***Pycharm:*** Es un entorno de desarrollo integrado utilizado para el lenguaje de programación *Python*. Al ser multiplataforma integra múltiples opciones y aplicaciones, además soporta desarrollo con *anaconda*. En este caso se ha utilizado este programa para realizar la consola de telemanipulación y la odometría del robot. Al comunicarnos con el

robot mediante un módulo de comunicación, podemos usar cualquier tipo de lenguaje para interpretar la información.

Antes de tener disponible la nueva versión de la consola para poder realizar pruebas de manera rápida, se utilizaba conjuntamente ***Matlab*** como herramienta matemática para poder interpretar los valores obtenidos, así como su representación y cálculo.

El módulo de comunicación XBee (apartado 2.2.1) requiere de un software de la empresa *Digi* para poder comprobar su funcionamiento. XCTU nos permite comprobar mediante una consola si el emisor y receptor del módulo están funcionando correctamente y si se pueden comunicar entre ellos.

# 2. Descripción del Robot

## 2.1 El Robot

El robot se desarrolló como una ayuda práctica para facilitar la compresión de los temas presentados en la asignatura. Con unas sesiones de laboratorio los alumnos pueden entender y aplicar la teoría presentada en el curso y a su vez trabajar con placas de desarrollo y su programación orientada a la robótica y control que está relacionado con otras asignaturas abordadas en el grado de ingeniería electrónica y automática (Sistemas electrónicos programables, robots autónomos, electrónica industrial…)

Durante el cuatrimestre, se realiza un trabajo que consiste en desarrollar una aplicación de seguimiento de muros de manera autónoma. El desarrollo de esta aplicación es sencillo debido a que el objeto principal de la tarea es la de realizar un análisis de la caracterización de tiempo real.

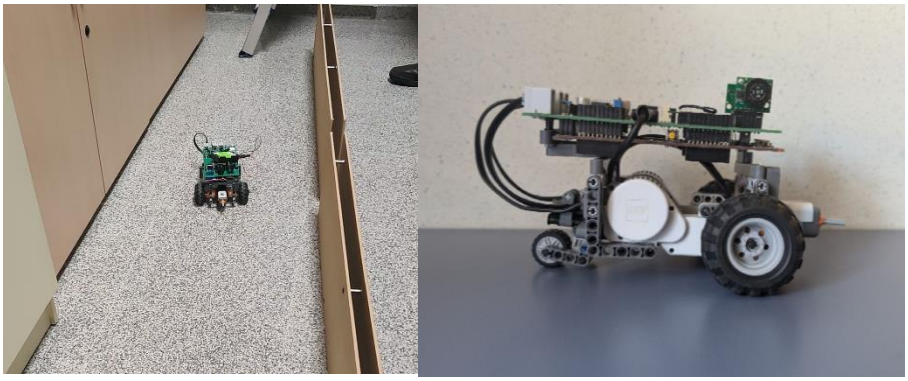


*Figura 2 Robot en las prácticas de la asignatura de STR.*

Por caracterización de tiempo real nos referimos al análisis de procesos y funcionamiento de un sistema que debe ejecutarse de manera rápida, para ello, el programa se dividirá en las distintas tareas que deba realizar el robot tales como medición del entorno mediante sensores, tarea de navegación, odometría, giroscopio, servos… cada una cumpliendo sus plazos de respuesta previamente establecidos por el usuario.

## 2.2 Componentes del Robot:

### 2.2.1 Hardware

El robot, está construido mediante piezas o bloques de la marca *LEGO Technic,* y sobre la estructura se han incorporado los distintos componentes que lo constituyen:

Placa de desarrollo LAUNCHXL-F28377S: esta placa es de la empresa Texas Instruments y pertenece al proyecto de *LaunchPad Development Kit* el cual es un set de desarrollo que abarca múltiples usos, ya que esta placa específicamente dispone de varios conversores AD (12 canales diferenciales de 16 *bits* o 24 canales de 12 *bits*), 24 canales PWM, 6 módulos de captura,3 *Timers* y 168 pines GPIO con puertos desde A hasta F.

Constituida por un procesador de 32 bits que trabaja a 200 MHz, memoria interna de 512K tipo *flash* y 132K de RAM. La placa requiere de 3.3 V de alimentación. (Texas Instruments 2015 [10]) (Texas Instruments 2015 [11])

La placa se programa mediante conexión de un cable *micro-usb* a un ordenador con el programa de desarrollo de la propia marca ***Code Composer Studio***.

Servomotores LEGO *Education* NXT 9842: Servo motores que incorporan un *encoder* para medir la velocidad con 1 grado de precisión. (LEGO 2006 [13])

LB1836M *Driver motor*: Es un controlador de motor diseñado para controlar motores de corriente continua (DC) en aplicaciones que requieren un control preciso de velocidad (PWM). El rango de tensión es de 2-7.5v. (ON Semiconductor 2013).

Sensor LEGO 9843 NXT: Sensor de presión que actúa a modo de ***bumper*** para el robot, se activa cuando se pulsa. (LEGO 2006 [12])

Sensores ultrasonidos LV-MaxSonar -EZ series: Módulos ultrasonidos pueden detectar objetos desde 15 centimetros hasta un rango de 6.45 metros (254 pulgadas). Operan con tensiones entre 2.5v- 5v. (MaxBotix, 2015)

Giroscopio L3G4200D: Es un giroscopio de tres ejes fabricado por *STMicroelectronics* que proporciona mediciones precisas de la velocidad angular en aplicaciones de detección de movimiento. Se comunica mediante el protocolo de comunicación I2C o SPI. (Digilent. 2011) (STMicroelectronics 2010)

Xbee S1: Módulos de comunicación inalámbrica mediante RF (radiofrecuencia). Funcionan bajo el estándar IEEE 802.15.4 de redes de área personal y baja potencia (100m en interiores). Además, se integran mediante una interfaz serie UART (*universal asynchronous receiver / transmitter*) que permite comunicar el robot con un ordenador externo. Funciona a 9600bps. (Digi International 2009).

Todo ello se alimenta directamente con 5V para ello, se dispone de un alimentador con alargador para poder conectarlo a la red, pero también se puede usar un adaptador de pilas de manera que cuando realizamos pruebas en movimiento, el robot pueda moverse sin ningún problema ni entorpecimiento. Además, el regulador de tensión que incorpora el

robot admite hasta tensiones de 7V5, límite de los motores lo cual podría usarse si requiere de más potencia para moverse.

Las unidades de trabajo usadas en el robot corresponden al **sistema internacional (SI).**

### 2.2.2 Software

De la asignatura de sistemas de tiempo real, se utiliza el proyecto base "ROBOT" del programa *Code Composer Studio*, donde están incluidos todos los *drivers,* librerías y el programa principal *main.*

También se dispone de la consola de telemanipulación utilizada en las clases prácticas.

# 3.Desarrollo:

## 3.1 *Drivers*: Actualización y puesta a punto

El proyecto "ROBOT" en *Code Composer Studio* dispone de varios *drivers* para facilitar el control de los distintos sensores que dispone, así como para gestionar las diferentes aplicaciones entre sí. En los referidos a este trabajo, se han modificado y extendido para mejorar su uso.

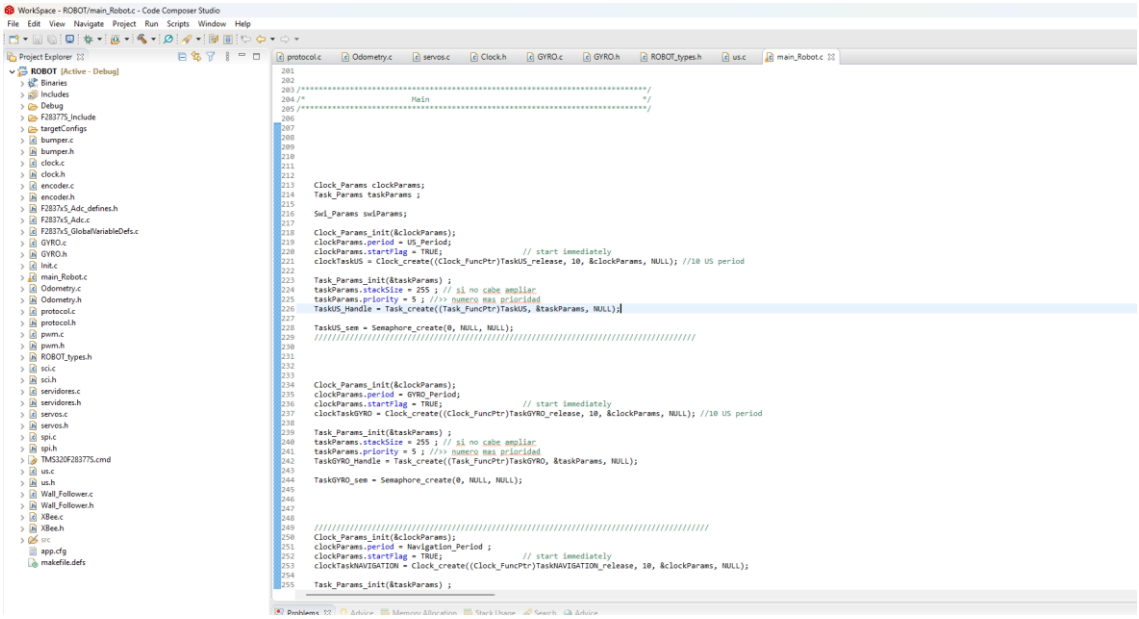



*Figura 3 Captura de Pantalla del proyecto ROBOT en CCS*

Dentro de estos *drivers* destacan *"US"* encargado de gestionar los ultrasonidos, *"GYRO"* que controla el giroscopio. "Servidores*"* gestiona las prioridades de las distintas tareas. Y por último *"Protocol"* y *"Odometry*" se han modificado para adaptar el envío de la odometría y recepción de órdenes a través del módulo XBee a un ordenador de control.

### 3.1.1 *Driver* "US":

Para poder controlar la navegación, el robot, este dispone de tres ultrasonidos situados en la parte superior delantera: frontal, derecho e izquierdo.

Estos ultrasonidos (*LV-MaxSonar -EZ series*) pueden detectar objetos desde 15 cm (Apartado 2.2.1). Entre los 15-50cm, aparece un posible error de ±5cm por la posible cancelación de la onda de retorno. Este efecto disminuye conforme el objetivo a medir está a una distancia superior de 50cm.

El *driver* gestiona desde la lectura de valor hasta las funciones para optimizar la medición.

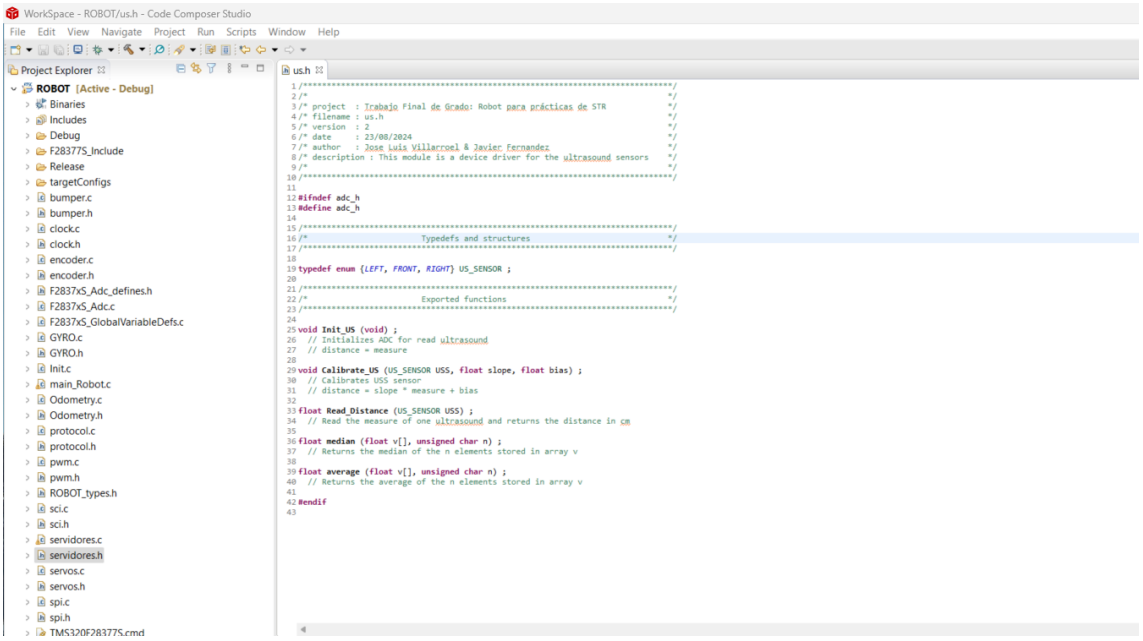



*Figura 4 Captura de pantalla del script US.h y sus funciones*

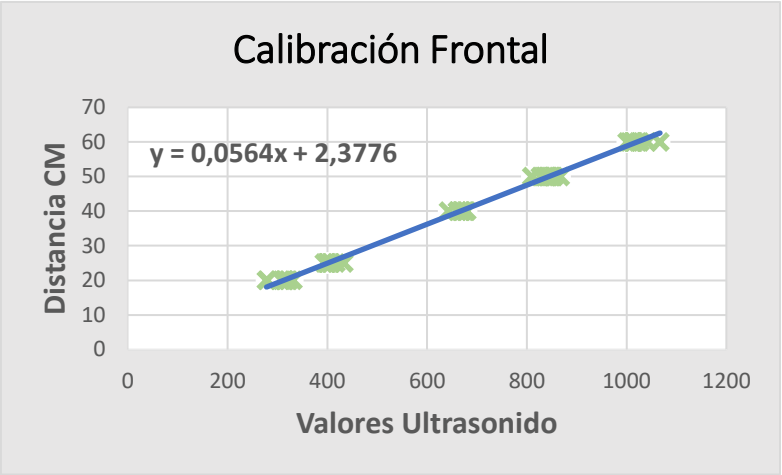
Primero, se inicializan las entradas a las que están conectados los correspondientes ultrasonidos. Con la función ***Read_Channel*** elegimos el ultrasonido que queramos medir (frontal, izquierdo o derecho) y leemos el valor que recibimos de los sensores.

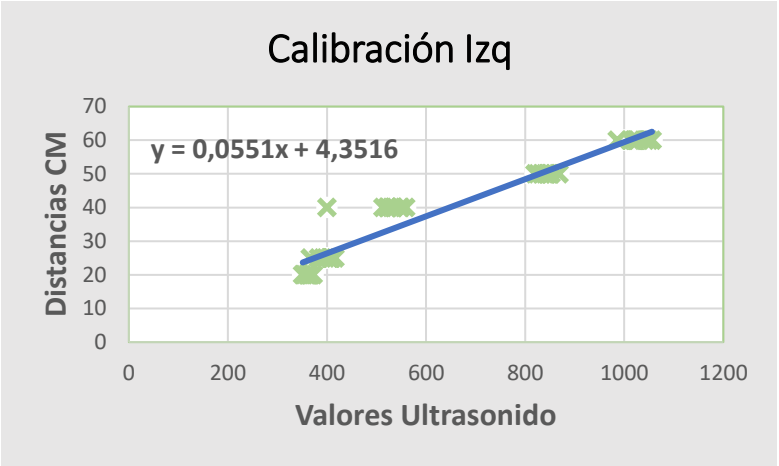
Para poder usar los valores obtenidos es necesario calibrarlos para poder obtener la medida en este caso en centímetros.

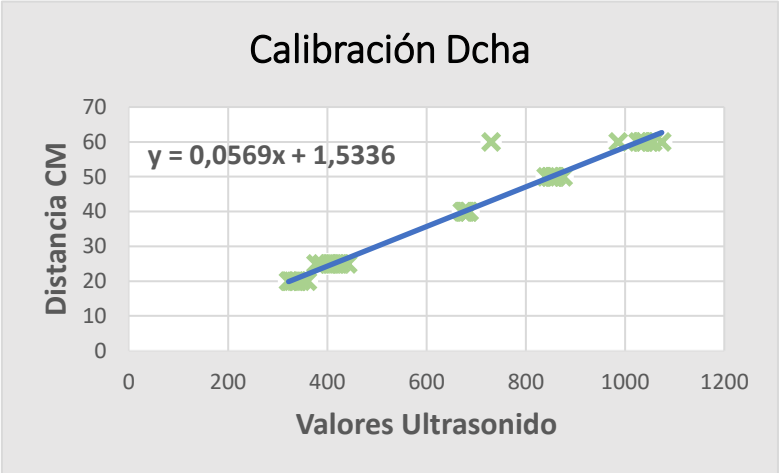
Una forma fácil y óptima de hacerlo es con una cinta métrica. Marcando distancias fijas de 20, 25, 40, 50 y 60 cm (rango habitual a usar en el robot) medimos cuantos $mV$ (valor del sensor) obtenemos a estas distancias realizando varias interacciones. Generamos una tabla de valores con la que podemos calcular una regresión lineal y obtenemos una pendiente y una ordenada para cada ultrasonido con la ecuación:

$$y = mx + n$$

Donde Y serán los cm calculados y X el valor obtenido del sensor ($mV$). De esta manera obtenemos los valores de calibración:



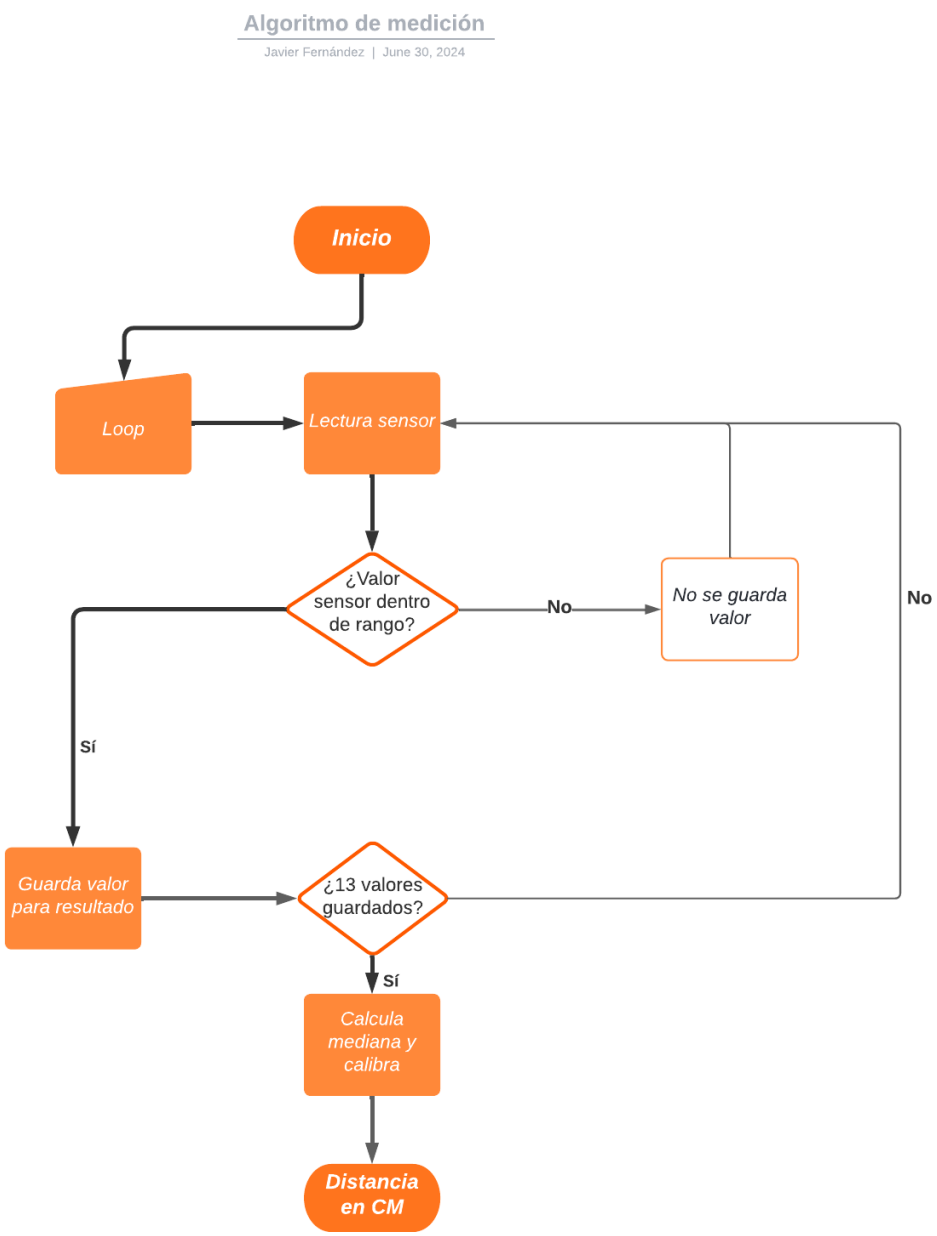




*Figuras 5,6 y 7. Valores de calibración para ultrasonido izquierdo, derecho y frontal*

Estos valores se implementan en la función ***Calibrate_US.*** Finalmente, con la función ***Read_Distance*** englobamos el proceso entero. Leemos los valores que obtenemos del sensor, calibramos y con un algoritmo de ajuste optimizamos la medición.

El ajuste de la medición se realiza con un algoritmo distinto. Cada 5ms se obtienen 13 valores con los que calculamos su mediana. Para evitar valores espurios, se impone una condición de que el valor actual debe estar entorno a $\pm 8.5mV$ (unidades del ultrasonido típico de 16.5 $mV/cm$) respecto al valor anterior. Es decir, se impone una ventana para los posibles valores.




*Figuras 8, Diagrama de flujo algoritmo de medición*

De esta manera se consigue reducir el error con un margen de entre 1.5-2cm (véase anexo apartado A.3.2 de mediciones) de la medida y estabilizarla. Tras realizar varios ensayos con los ultrasonidos y midiendo las distancias reales con una cinta métrica, se ha comprobado que los ultrasonidos por debajo de los 18 cm obtienen valores que difieren de la distancia real (más de 2cm de error). Al ser sensores con bastante uso (años de universidad y prácticas acumuladas de distintos estudiantes) las membranas de estos se deterioran y empeora la operación de estos. Cabe destacar, que las mediciones de los ultrasonidos son cruciales para el correcto funcionamiento del robot, ya que los algoritmos creados para la navegación y la odometría dependen directamente de estos datos.

Por este motivo, y tras los ensayos realizados, se establece una **distancia mínima** de medición de **25cm**. Además, como se ha mencionado, la tarea principal de los ultrasonidos es la de medir distancias de objetos próximos al robot, por lo tanto, no será necesario realizar medidas de distancias pequeñas porque se quiere evitar que el robot se acerque demasiado a un objeto donde no pueda maniobrar.

Una vez determinado el *driver*, desde el programa ***main_robot*** creamos una tarea ***(TaskUS)*** para poder aplicar las funciones del *driver* periódicamente y poder usarlo para el resto de las tareas prioritarias. En este caso la tarea de US se ejecutará cada 5ms y con una prioridad de 6, es decir, con la mayor prioridad ya que es una de las tareas principales.

### 3.1.2 *Driver* "GYRO":

Otro de los sensores de los que dispone el robot es el giroscopio *PmodGYRO*, el cual es un módulo que incluye el sensor de movimiento de bajo consumo L3G4200D que permite medir la velocidad angular en los tres ejes (apartado 2.2.1). El giroscopio, situado en la parte inferior del robot (figuras 10 y 11), nos devuelve los valores de referencia angular según la indicación (figura 9) (STMicroelectronics 2010): $\overrightarrow{\Omega X}$ $\overrightarrow{\Omega Y}$ $\overrightarrow{\Omega Z}$

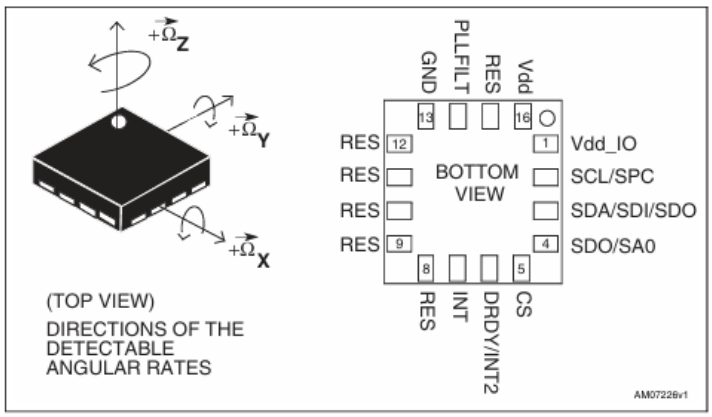
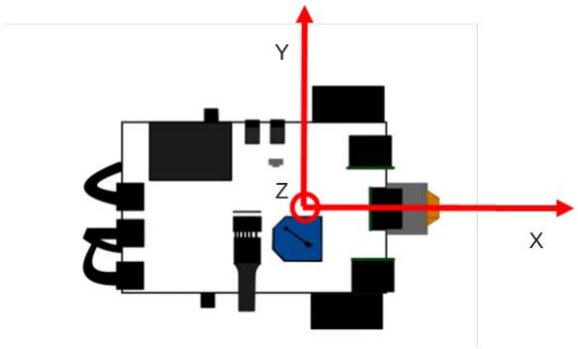
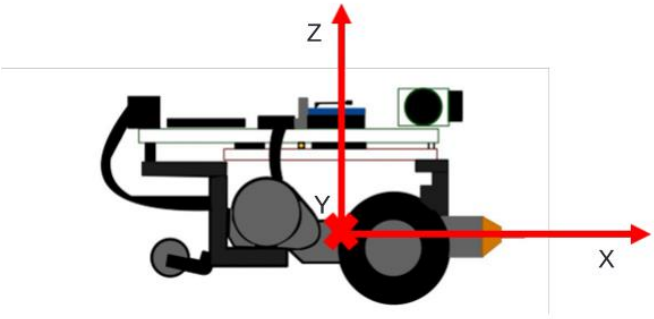



*Figura 9, Dirección de los ángulos del sensor según datasheet L3G42000D.*

De esta forma, según la posición del sensor en el robot, su referencia resulta:




*Figuras 10 y 11. Planta y perfil del robot con representación de los ejes según Pmodgyro.*

Desde el *driver GYRO* controlamos desde la obtención del dato del sensor hasta la conversión a rad/s (ω) ya que se trabaja con unidades del SI.

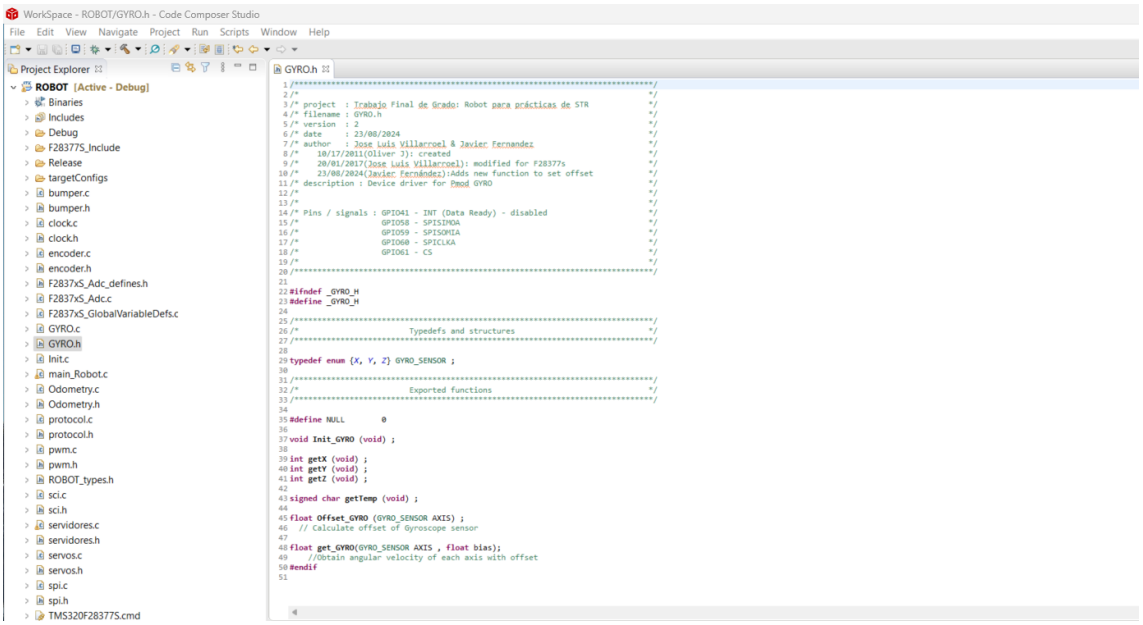



*Figura 12, Captura de pantalla del script GYRO.h y sus funciones*

La importancia del giroscopio es conocer el valor de la velocidad angular en cada instante, ya que de esta manera se obtiene una medida de los giros que realiza el robot, pudiéndose usar para controlar el giro de las ruedas. Esto se profundizará en la sección de la aplicación de navegación (apartado 3.2.3).

El giroscopio del que se dispone es de 16 bit (rango de lectura de -32768 a 32767). Dispone de diferentes escalas tipo **dps** (*degrees per second*) En este caso se opta por la más básica de $-250$ º/s a $250$º/s, ya que no va a ser necesario una escala mayor, siendo que la velocidad del robot se limita en el algoritmo de navegación y nunca excederá esos valores.

Con estos valores y el dato de muestreo del sensor hacemos la transformación (***Calcula_Gyro***) a º/s y de ahí lo transformamos a rad/s.

Dependiendo de la posición del robot el sensor no se va a encontrar totalmente en una posición horizontal respecto al suelo, es decir, está desnivelado y acumula un error en la medida que se obtenga.

Para solucionar este problema, se crea una función para calcular el ***offset*** y poder compensar el error durante el movimiento del robot. ***Offset_Gyro*** obtiene en un array 50 valores del sensor para el eje Z (giros del robot respecto al plano de movimiento) y se realiza una media de los valores con el que obtenemos el valor de *offset*. Esta función se aplica en la navegación al inicio de manera que se obtenga el valor de *Offset del* giroscopio para ese instante, además se puede recalcular con la función Calibración de la consola de telemanipulación (apartado 3.3.1). Una vez el robot está en movimiento, con ***get_Gyro*** se obtienen los valores del sensor, se compensan con el offset y se transforma a **rad/s:**

$$W(º/s) = (Valor\_act - offset) * \frac{250\ º/s}{32767.0}$$

$$\mathrm{W(rad/s)} = W(º/s) * 0.0174$$

En el programa principal se ejecuta la tarea encargada de gestionar este *driver* ***(TaskGYRO)*** la prioridad de esta tarea es de 5 con un periodo de 5ms.

### 3.1.3 *Driver* "Odometry":

La odometría nos permite estimar la posición y orientación del vehículo a partir del uso de datos de sensores de movimiento, en este caso de los ***encoders*** situados en los motores de las ruedas. Sabiendo la velocidad lineal V y angular W podemos calcular unas coordenadas X e Y que representan la posición del robot sobre sí mismo de manera que podemos interpretar en un espacio la trayectoria que realiza. Con el ángulo θ sabremos los giros que realiza para un periodo T de control.

*Odometry* se encarga de calcular estas coordenadas a partir de las velocidades obtenidas por el robot.

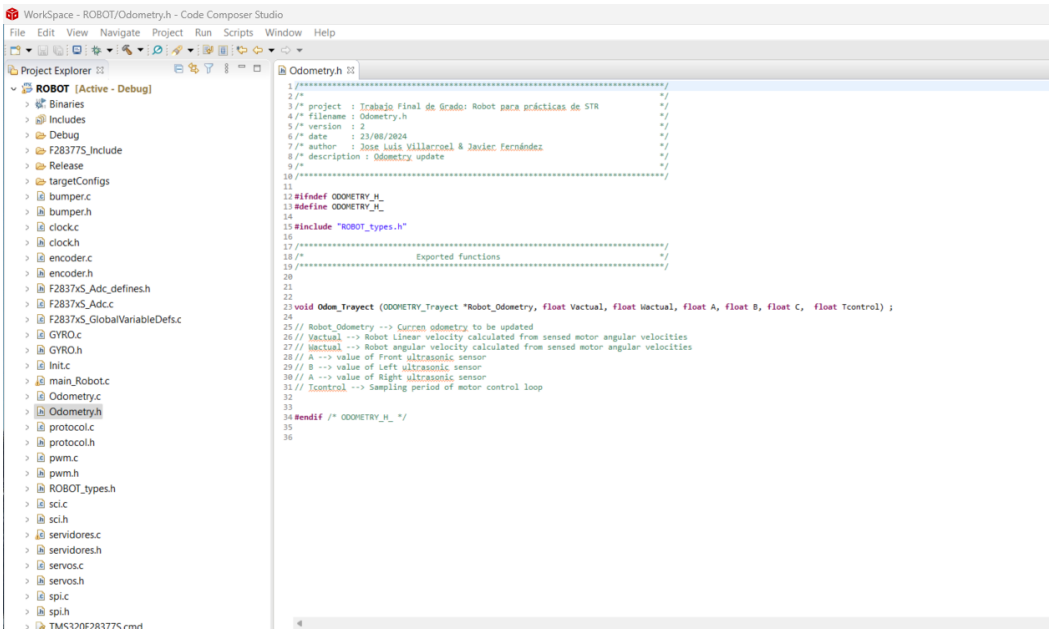


*Figura 13, Captura de pantalla del script Odometry.h y sus funciones*

*Odom_update* se encargaba de calcular los valores de la odometría en las clases prácticas de la asignatura, pero como se ha mencionado en el apartado 1.2, como mejora planteada del trabajo se quiere representar la nube de puntos respecto a la trayectoria del robot para realizar un mapa de la zona. Para ello, era necesario ampliar las variables obtenidas. En este *driver* se añade la función ***"Odom_Trayect"*** que es capaz de gestionar las 6 variables necesarias para poder mostrar la trayectoria y su mapa en el ordenador:

Coordenadas X e Y y ángulo θ: Obtenidas por cálculo a partir de las velocidades V y W del robot.

Valores de los ultrasonidos: datos de los sensores ultrasonidos frontal (dF), izquierdo (dL) y derecho (dR) para cada posición calculada del robot.

Una vez determinado el *driver*, desde el programa ***main_robot*** creamos una tarea ***(TaskODO)*** para poder aplicar las funciones del driver periódicamente y poder usarlo para el resto de las tareas prioritarias. En este caso la tarea de *Odometry* se ejecutará cada 250ms y con una prioridad de 1, es decir, con la menor prioridad al ser la tarea con periodo mayor y no es crucial para el funcionamiento del resto de aplicaciones.

### 3.1.4 *Driver* "Protocol":

*Protocol* se encarga de gestionar el envío y recepción de datos entre el robot y la consola de telemanipulación a través del módulo de comunicación XBee.

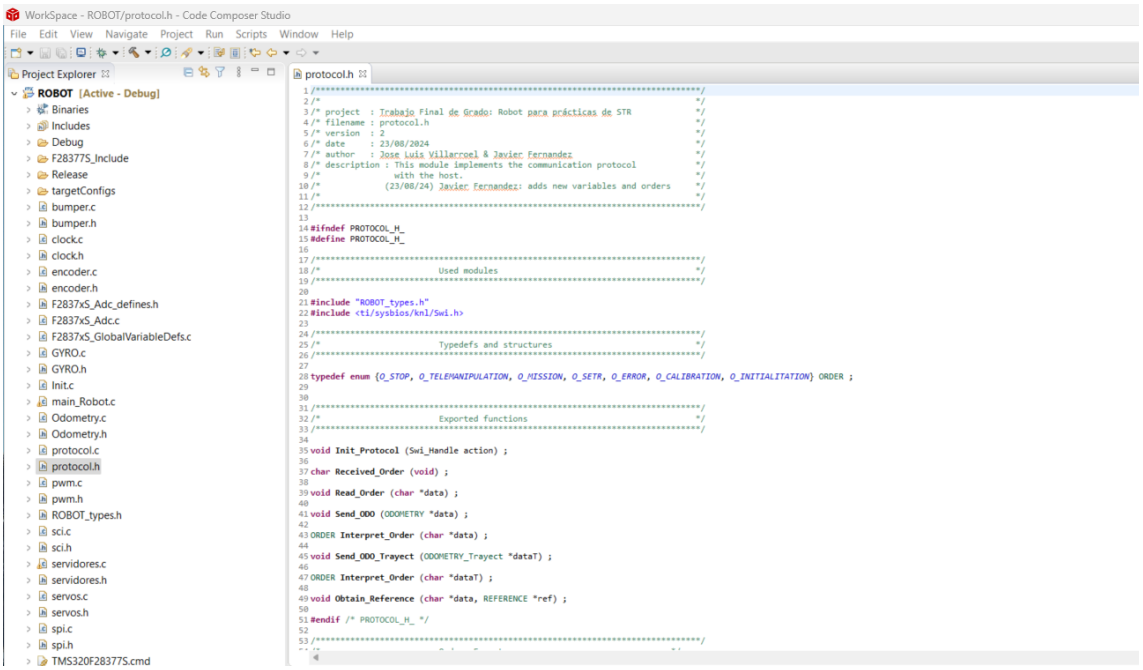



*Figura 14, Captura de pantalla del script Protocol.h y sus funciones*

En *protocol* encontramos las funciones que envían los valores de la odometría y de la nube de puntos obtenidos por el *driver Odometry*, así como las funciones encargadas de recibir las órdenes de control enviadas desde la consola de telemanipulación.

La función ***Send_Odo*** envía los valores obtenidos en *Odometry.* Para ello, se genera un mensaje o *frame* de caracteres no binarios, es decir, un array de caracteres. Este mensaje se genera con una estructura definida de manera que luego se pueda interpretar correctamente en la consola de telemanipulación. La estructura es la siguiente:

"#ODO X,Y, θ,dF,dL,dR\n"

*Figura 14, Estructura del mensaje*

Donde "#" indica el inicio del mensaje, los caracteres "ODO" hacen referencia a que la estructura siguiente es la odometría y a continuación se escriben los datos de la odometría, las coordenadas X e Y, el ángulo de giro θ y los valores de los ultrasonidos dF, dL, dR (en orden, sensor frontal, izquierdo y derecho). Por último, los caracteres "\n" marcan el final de mensaje y salto de línea para escribir el siguiente mensaje. Posteriormente estos mensajes se interpretan en la consola de telemanipulación donde se estructuran para poder ser usados cómodamente.

*X,Y,theta,dF,dR,dL*

*0,0,0,650,427,1922*

*50,0,0,643,432,1914*

*100,0,0,644,423,1917*

*150,0,0,650,424,1920*

*200,0,0,652,421,1914*

*250,0,0,646,412,1921*

*Figura 15, Ejemplo de mensajes enviados por el robot e interpretados por la consola*

Los datos de la odometría y de los ultrasonidos se obtienen de variables declaradas como tipo flotante o *float.* Hay que tener en cuenta que al enviar este tipo de datos como caracteres puede suponer la perdida de información ya que los *float* admiten valores decimales y se perderían en la conversión. Para ello, primero se realiza una conversión o *casting* de las variables a tipo entero o *int* de manera que tengamos únicamente valores enteros.

Para no perder información, antes de la conversión se multiplica el valor por 10 de manera que en la consola una vez obtengamos los datos del mensaje se divida por 10 consiguiendo recuperar un decimal.

En un principio por mantener dos decimales de las medidas, los datos se multiplicaban por 100, esto llevó a un problema con los valores de la odometría donde las coordenadas

X , Y y el ángulo de giro con el tiempo alcanzaban el valor máximo permitido por las variables tipo *int* de 16 bits (32767), es decir, se producía desbordamiento o *Overflow.*

Para evitarlo, se envían los valores de la odometría sin decimales ya que al fin y al cabo para representar la trayectoria no supone un problema, en cambio para las distancias de los ultrasonidos sí que se ha mantenido al ser centímetros necesitamos esos decimales para obtener una distancia más exacta.

Por otro lado, *Interpret_order* gestiona la recepción de las ordenes de control por parte del robot. De manera similar al envio de los datos de odometría, en este caso en la consola a través del ordenador se envían las distintas cadenas de caracteres que corresponden a las ordenes que posteriormente se ejecutan en el programa *main* desde la tarea **TaskORD**.

```
143 ORDER Interpret_Order (char *data) {
144
145     if (data[0]==Begin_Frame && data[1]=='S' && data[2]=='T' && data[3]=='O' && data[4]=='P')
146         return O_STOP ;
147     else if (data[0]==Begin_Frame && data[1]=='M' && data[2]=='I' && data[3]=='S' && data[4]=='N')
148         return O_MISSION ;
149     else if (data[0]==Begin_Frame && data[1]=='T' && data[2]=='E' && data[3]=='L' && data[4]=='E')
150         return O_TELEMANIPULATION ;
151     else if (data[0]==Begin_Frame && data[1]=='S' && data[2]=='E' && data[3]=='T' && data[4]=='R')
152         return O_SETR ;
153     else if (data[0]==Begin_Frame && data[1]=='C' && data[2]=='A' && data[3]=='L' && data[4]=='B')
154             return O_CALIBRATION ;
155     else if (data[0]==Begin_Frame && data[1]=='I' && data[2]=='N' && data[3]=='I' && data[4]=='T')
156             return O_INITIALITATION ;
157     else return O_ERROR ;
158 }
```

*Figura 16, Interpret_Order y las ordenes de control*

Por último, *Obtain_reference* permite recibir valores de las velocidades V y W para ajustar la velocidad del robot desde la consola de telemanipulación. Todas las funciones y características de la consola se explicarán en el apartado 3.3.1.

### *3.1.5* Servidores:

Este módulo se encarga de implementar los servidores con herencia de prioridad que gestionan la información compartida entre las diversas tareas. Según la figura 17, estos servidores son:

Ultrasonidos (US): Este servidor recibe los valores medidos por los ultrasonidos en la tarea *TaskUS*. Por lo tanto, es uno de los servidores más importantes y crucial para el control de navegación del robot.

Giroscopio (GYRO): Recibe el valor de la velocidad angular $w_Z$, medido por el giroscopio y lo envía a la tarea de navegación.

Referencia (REF): Gobernado por la tarea navegación, la cual manda que referencias debe seguir el robot, el servidor REF se encarga de enviar este control a la tarea *TaskSERVO* la cual modificara los valores de V y W (velocidad lineal y angular) para la correcta navegación del robot.

Estados (*STATUS*): Recibe de la tarea de órdenes los distintos estados en los que puede encontrarse el robot (Stop, Mision…) y así controlar en cual se encuentra y así, poder controlar a que estado moverse.

Odometría (ODO): Recibe los valores de la posición y distancias de la tarea de odometría y así poder describir la trayectoria del robot y la nube de puntos.

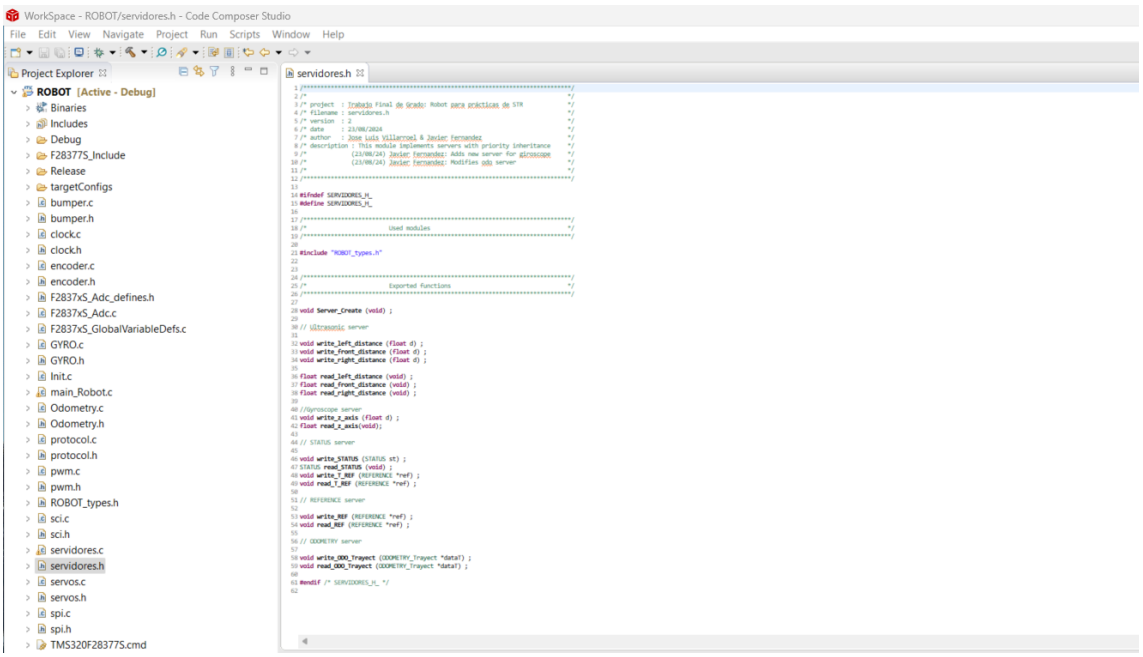



*Figura 17, Captura de pantalla del scrpit Servidores.h*

Con la creación de una tarea específica para el giroscopio se ha creado un servidor que gestione su información compartida. Además, a añadirse nuevas variables para su envío en el *driver* de "Odometry", ha sido necesario modificar las funciones que controlan este servidor, *"write_odo_trayect"* y *"read_odo_trayect",* de manera que se puedan gestionar 6 variables (X, Y, θ, df, dr, dl).

No es uno de los cambios más extensos, pero sí de los más importantes, ya que es el encargado de trabajar con los punteros de las variables para poder recibir y enviar su información respetando las características de un sistema de tiempo real con herencia de prioridad (apartado 1.1.1).

## 3.2 Navegación: Aplicación demostrativa

Uno de los puntos importantes de este proyecto es el de poder ampliar el trabajo de las prácticas de laboratorio de STR, de ahí que la navegación autónoma sea uno de los aspectos clave.

Partiendo del trabajo de la asignatura (apartado 2.1), se ha conseguido una aplicación demostrativa donde el robot es capaz de moverse en un ambiente no "critico" es decir, zonas donde el robot pueda moverse con sus capacidades por lo general en sitios cerrados, con obstáculos sin distintas alturas. Obviamente esta navegación se encuentra dentro de los propios límites de los que dispone el robot como es su interacción con el entorno a través de 3 ultrasonidos y el giroscopio. La estructura planteada es la siguiente:

### 3.2.1 Diseño del algoritmo:

El robot siempre va a moverse hacia delante (velocidad constante), a no ser que no pueda. Esto puede ser debido a que hay un obstáculo delante o se encuentra en un espacio cerrado donde debe moverse en otra dirección, es decir, si el ultrasonido delantero, o el ***bumper*** detecta algo de frente. Esta decisión se ha tomado ya que la prioridad de la aplicación es que **el robot pueda moverse**, e intentar adaptarse a la zona en la que se encuentra. La idea de seguir un muro se había explorado en la propia asignatura y esto suponía un reto más complicando al intentar crear un algoritmo que fuera capaz de desenvolverse en la gran mayoría de las situaciones, teniendo en cuenta la ejecución de las tareas en tiempo real.

Mientras el robot avanza, constantemente se realiza lecturas de los tres ultrasonidos y del giroscopio. Si detecta que se está acercando a "algo" de frente (la distancia del ultrasonido frontal se reduce) empieza a disminuir su velocidad hasta poder interpretar la situación en la que puede encontrarse. Para seguir moviéndose se han englobado las situaciones en 5 casos:

**Sigue**: Todo funciona con normalidad con lecturas adecuadas, el robot avanza recto.

**Frente**: En caso de que se detecte frontalmente una distancia menor a la de seguridad del robot (40cm, permiten al robot poder maniobrar) este se detiene, puede ser debido a un obstáculo, un objeto en movimiento, persona, animal… Si es momentáneo, en cuanto recupere el valor normal, continuará avanzando. En caso contrario, significa que no puede avanzar de frente y debe moverse en una dirección transversal, lo que nos lleva a los siguientes estados izquierda y derecha.

**Izquierda y Derecha**: midiendo las distancias transversales del robot se elige dirección de giro aquella cuya distancia es mayor, es decir, para el instante donde el robot parado está midiendo los ultrasonidos, se moverá en la dirección donde haya un obstáculo más lejos, girará y volverá a navegar en el estado Sigue.

**Rodeado**: Si el robot por algún motivo excede la distancia de seguridad (más cerca frontalmente de 40cm), se interpreta que el robot ha llegado a una situación en la que no puede avanzar de manera frontal, ya sea por un obstáculo un camino sin salida, fallo de lectura… en este caso el robot se moverá marcha atrás hasta alcanzar la distancia de seguridad (40cm), volviendo a redirigir su dirección.

Mientras el robot avanza, si este se va acercando a una pared u obstáculo tanto por el lateral izquierdo como por el derecho, rectifica la dirección para no acercarse más y evitar un choque. Es decir, aplica el **seguimiento de muros**.

Mostrando el desarrollo en un diagrama de flujo:

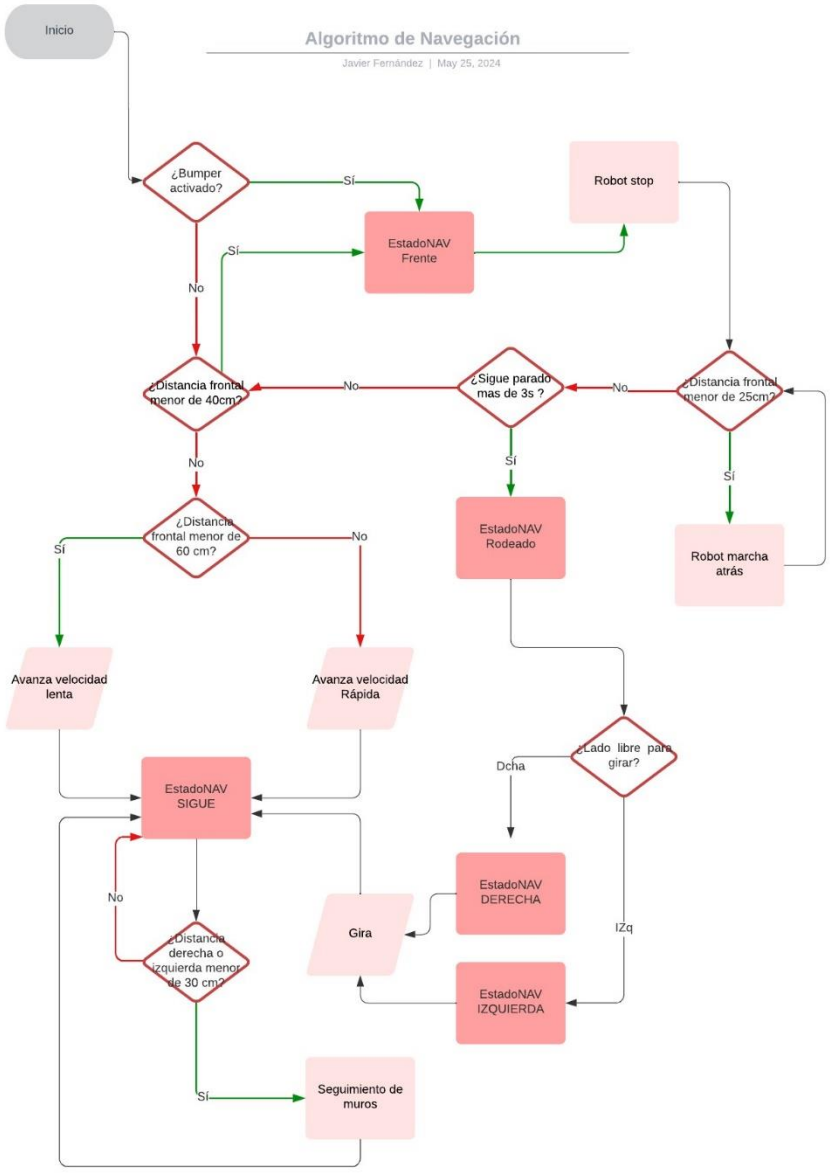



*Figura 18, Flujo algoritmo de navegación*

Este algoritmo es el resultado de realizar diversas prácticas en distintas zonas y suelos donde se han ido corrigiendo los diferentes estados de la navegación hasta alcanzar una aplicación con la que el robot puede desenvolverse sin mayores problemas. Pero, a la vez que se ha optimizado el algoritmo, han surgido problemas más profundos.

### 3.2.2 Puntos Ciegos:

Como se ha explicado anteriormente, el robot para poder maniobrar depende prácticamente de los ultrasonidos, pero estos tienen un rango determinado.

Los ultrasonidos emiten y reciben una señal con la cual se mide la distancia que ha recorrido hasta el objeto más próximo. Para conseguir que la medida sea lo más fiel

posible, el objeto a medir debe encontrarse en dirección **perpendicular** al sensor (caso **A** figura 19). Conforme esta dirección o ángulo varía, nos encontramos con un problema y es que el ultrasonido no es capaz de medir correctamente haciendo variar notoriamente la distancia medida de la real produciendo en el robot un **punto ciego,** al no saber la distancia real de lo que le rodea.

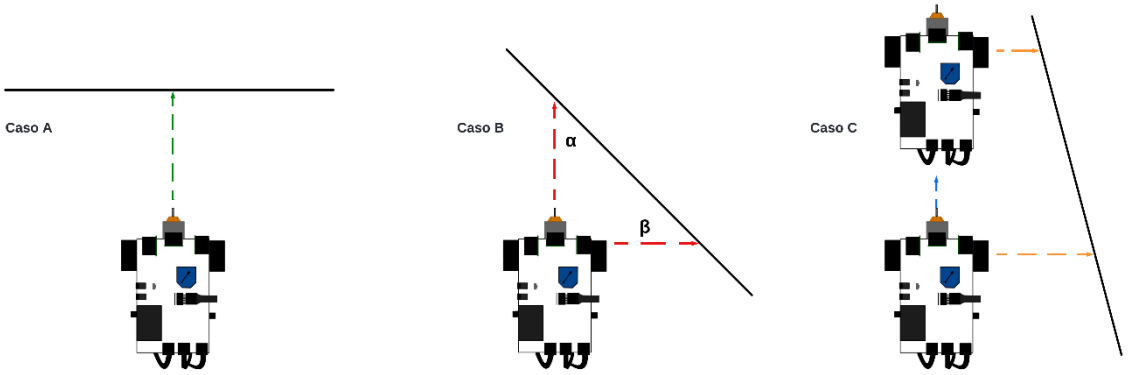



*Figura 19, 20, 21, Ejemplos para puntos ciegos.*

Para cierta angulación (caso **B,** figura 20), la onda de retorno se pierde o rebota provocando que la lectura del ultrasonido sea errónea (normalmente superior a la real), si el objeto está en el área de este ángulo de detección, el sensor puede no detectarlo correctamente o puede medir una distancia errónea, lo cual desestabiliza el funcionamiento de la aplicación de navegación, provocando posibles choques laterales o frontales, es decir, la configuración actual dispone de **puntos ciegos**.

Para evitarlo, se han intentado usar diferentes algoritmos aplicando trigonometría. Donde usando los ángulos α y β y los catetos que forman las distancias frente al objeto a medir, se busca una relación lineal entre ellos. Este algoritmo ha resultado ser demasiado complejo donde había que realizar varios cálculos complejos que para el resultado obtenido no merecía la pena usarlo.

Finalmente, siguiendo la idea de usar dos sensores a modo de apoyo en vez de aplicar trigonometría llegamos al caso **C** (figura 21)**.** Donde sabiendo si nos alejamos o nos acercamos a un objeto lateral y frontalmente podemos medir esa variación para saber en qué dirección movernos y evitar la colisión.

Con este algoritmo se han conseguido buenos resultados y disminuir el problema, pero para α y β entre valores de 40-50º, el punto ciego del robot persiste. Una posible solución a este problema es la de modificar la configuración actual de sensores, la cual se estudiará en el apartado 4.1.

### 3.2.3 Control de Velocidad y Posición:

En el sistema original las dos entradas que se usan para controlar al robot son la velocidad lineal **V** y la velocidad angular **W**. Para un entorno típico (suelo no deslizante) la velocidad lineal es fácil de controlar, pero conforme ha sido necesario modificar los giros

del robot para la aplicación de navegación, aparece un **problema de deslizamiento**, ya que, al realizar giros más bruscos, es más factible que se produzcan patinajes de las ruedas.

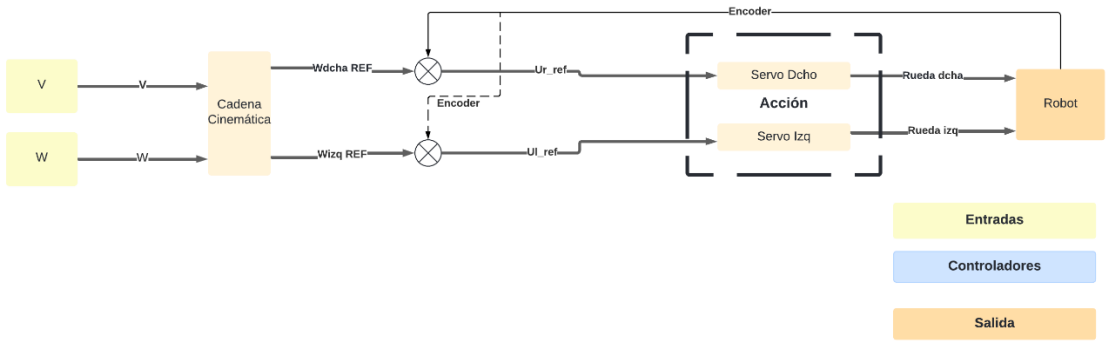



*Figura 22, Configuración original del servosistema del robot.*

Como se puede apreciar en el diagrama del servosistema (figura 22), para realimentar la velocidad del robot se usa la velocidad calculada por el propio *encoder*. Esto supone un problema porque si las ruedas deslizan repercutirá a la vez en la medición del *encoder*. Para solventarlo, hay que controlar estos deslizamientos. Se usarán los sensores de los que disponemos, en este caso los ultrasonidos y el giroscopio.

En la aplicación de navegación (apartado 3.2) se controla a qué velocidad debe girar el robot para que se desvíe de su trayectoria o esquive un objeto, pero ante un suelo muy resbaladizo, el robot intentará girar a la velocidad asignada durante el tiempo determinado, pero patinando, por lo que el giro será distinto al calculado, normalmente resultando en un derrape constante del coche.

La primera solución que se toma es la de compensar la velocidad angular ω con la calculada por el giroscopio, de esta manera mediante un controlador podemos regular $\boldsymbol{\omega}$ que usamos en los giros que realiza el robot.

Con un controlador de **posición**, utilizando un ángulo $\boldsymbol{\theta}$ como referencia, la velocidad angular se ajustará en función de los grados que el robot deba girar, consiguiendo reducir el deslizamiento y realizados giros con menor error de posición.

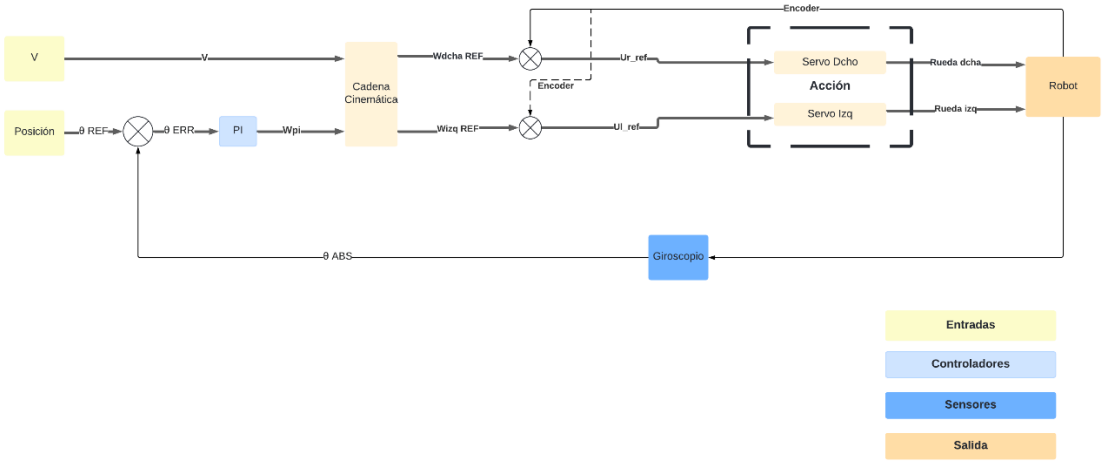


*Figura 23, Configuración servosistema con controlador de posición.*

El controlador que se ha usado para la posición es un proporcional integrador o **PI.** Se ha elegido este tipo de controlador por ser más estable que un controlador proporcional y a la vez es más fácil de aplicar que un tipo proporcional integrador-derivativo o PID. Para prevenir saturaciones en la acción integral se ha usado una estructura tipo ***Anti-Windup***. (Ogata, Katsuhiko, 1998).

La constante proporcional **Kp** y constante de integración **Ki** regulan la acción del controlador. Para su correcto funcionamiento, es necesario **sintonizarlas** (Ogata, Katsuhiko, 1998)**,** de manera que, con un buen ajuste de las constantes, la acción sea la adecuada.

Para la sintonización, se parte de valores Kp=Ki=0, donde se comprueba que la acción sea nula. Una vez se confirma, primero se aumenta el valor de la Kp poco a poco hasta alcanzar el valor máximo permitido por el controlador sin llegar a saturar la acción. Por último, con la Kp ajustada, se sintoniza Ki, para ello, se realiza el mismo proceso, pero en este caso hasta alcanzar el valor máximo para una buena respuesta en el transitorio sin comprometer el tiempo de respuesta (Véase anexo 4 con la sintonización de Kp y Ki).

Como se ha comentado se usa una estructura *anti-windup* donde los valores máximos determinados son de $\pm 1.57\ rad$. Ya que en este caso el robot no va a realizar un giro único de más de 90º.

Buscando otro tipo de regulación, se ha planteado el uso del ultrasonido frontal para compensar la velocidad lineal **V**. Para ello usaremos las distancias medidas con el ultrasonido y durante un intervalo calculamos su derivada para así saber a qué velocidad lineal nos estamos moviendo:

$$V = \frac{d_{act} - d_{ant}}{Tc}$$

*Figura 24, Cálculo de la velocidad*

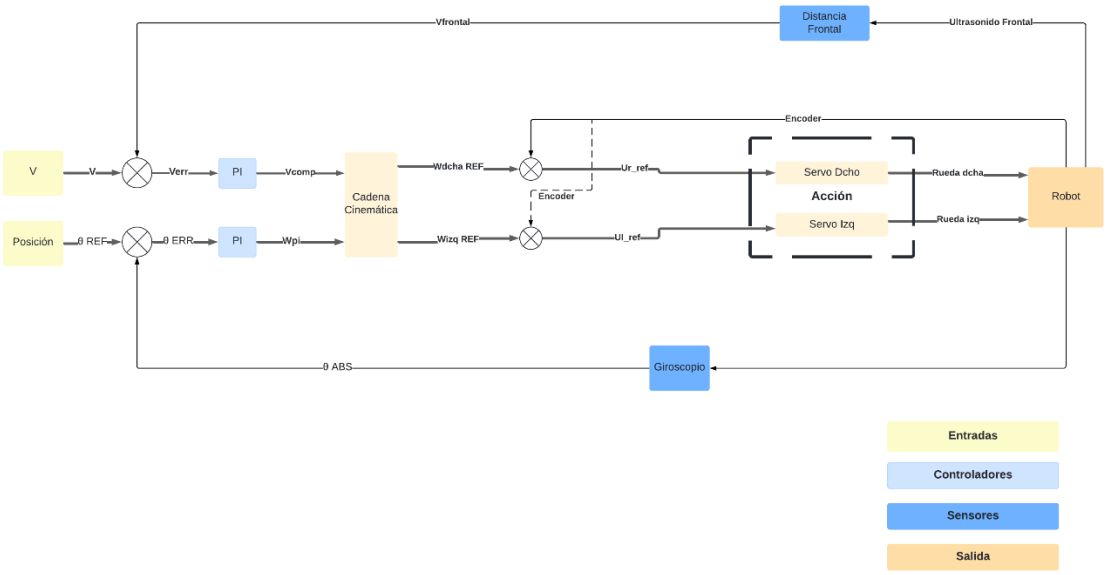



*Figura 25, Configuración servosistema con controlador de posición y velocidad lineal.*

Como se aprecia en la figura 25, la estructura elegida para el controlador de velocidad lineal es la misma que la anterior, la de un controlador tipo PI con estructura *anti-windup*. Se ha realizado la sintonización y el ajuste de la Kp y Ki y se ha elegido unos valores de saturación de $\pm 0.3\ m/s$.

El problema de esta implementación a diferencia del uso con el giroscopio es que debe ser en un entorno muy delimitado.

Para que la medida sea lo suficientemente buena, es necesario que la distancia medida por el ultrasonido se encuentre en el rango establecido, entre 20-80cm lo cual ya limita la distancia de uso, además esto supone que sea necesario comprobar esta situación constantemente antes de poder calcular la compensación. Este proceso al ser más enrevesado se ha descartado por ser limitado.

Por lo tanto, la configuración final elegida es la de la figura 23, aplicando únicamente el controlador de posición.

Desde el programa principal *main* se ejecuta la tarea ***(TaskNAVIGATION)*** que gestiona las funcionalidades de la navegación.

## 3.3 Control y Odometría:

### 3.3.1 Consola de manipulación

#### Creación nueva consola en Python

A la vez que se han extendido las aplicaciones del robot era necesario adaptar el control desde el ordenador.

Originalmente partíamos de una simple consola de comandos con los controles básicos, donde ciertas teclas del teclado activaban las diversas ordenes al robot. Las ordenes que originalmente podíamos mandar eran:

***STOP (S):*** Ajusta las referencias de velocidad lineal y angular (V y ω) a 0.0.

***MISION (A):*** Inicia aplicación seguimiento de muros

***TELE (T):*** Facilita manejo del robot manualmente mediante el uso de las teclas del teclado IJKLM (véase figura 26) donde se marca una referencia fija de velocidad que seguirá el robot con la función SETR (SETR vv.vv, ww.ww). I y M para aumentar y disminuir la velocidad lineal, JKL para la angular.

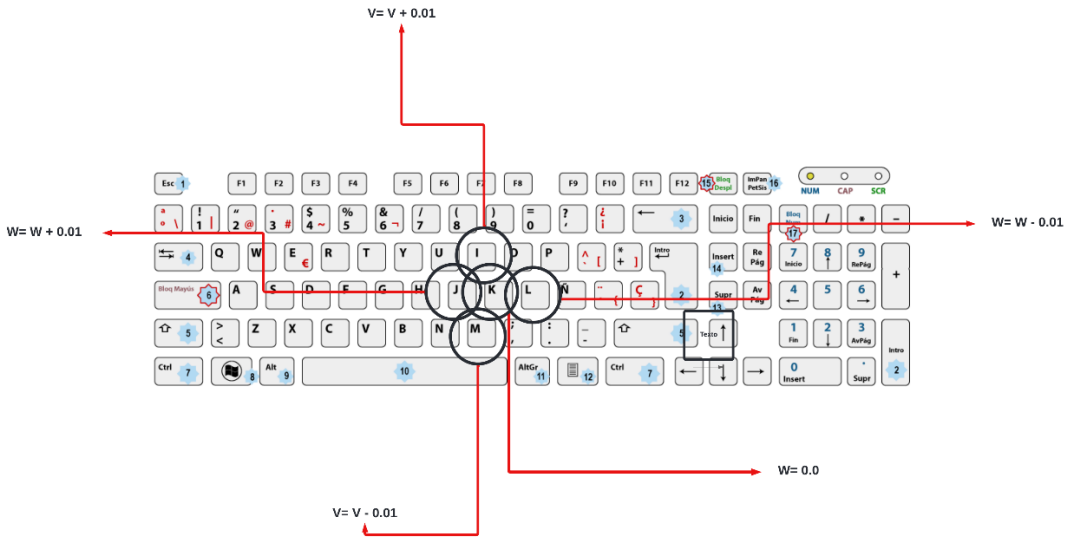



*Figura 26, Uso de teclas con TELE.*

La consola era funcional pero básica. Para ello, se ha diseñado una consola de manipulación nueva con Python. Donde disponemos de una interfaz gráfica con las ordenes modificadas y nuevas funciones añadidas.

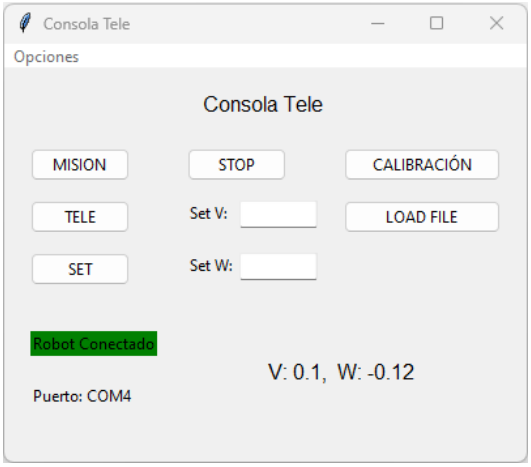



*Figura 27, Nueva consola de telemanipulación.*

Antes de abrir la consola, esta dispone un archivo de configuración "configuracion.txt" donde está indicado el puerto COM (serial para conectarnos al módulo **Xbee**) y en la

siguiente línea el nombre del archivo de texto donde se guardarán los valores enviados por el robot. Por defecto *"*trayectoria.txt*"*. Nada más iniciar la consola, si el módulo de comunicación no está conectada al ordenador, se abrirá una ventana emergente avisando de que no hay nada conectado al puerto con "Error puerto". Si está conectada, pero no hay conexión con el robot, la consola marcará que el puerto esta desconectado con un mensaje. Una vez el puerto esté preparado aparecerá el mensaje de **robot conectado** en verde, como se indica en la figura 27.

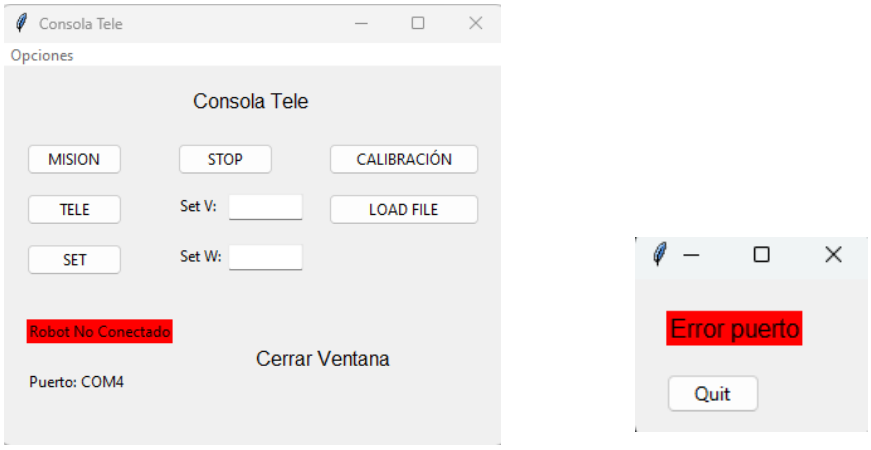



*Figura 28, Errores de inicio*

La consola dispone de pulsadores que activan las distintas opciones disponibles:

***MISION:*** Inicia la aplicación de navegación (apartado 3.2) y muestra ventana emergente con la odometría del robot en tiempo real.

***TELE***: Control manual del robot mediante las teclas de flechas indicando su dirección de movimiento. De manera similar de uso de la antigua consola, pero con las flechas de dirección, además en la consola se muestra la velocidad actual.

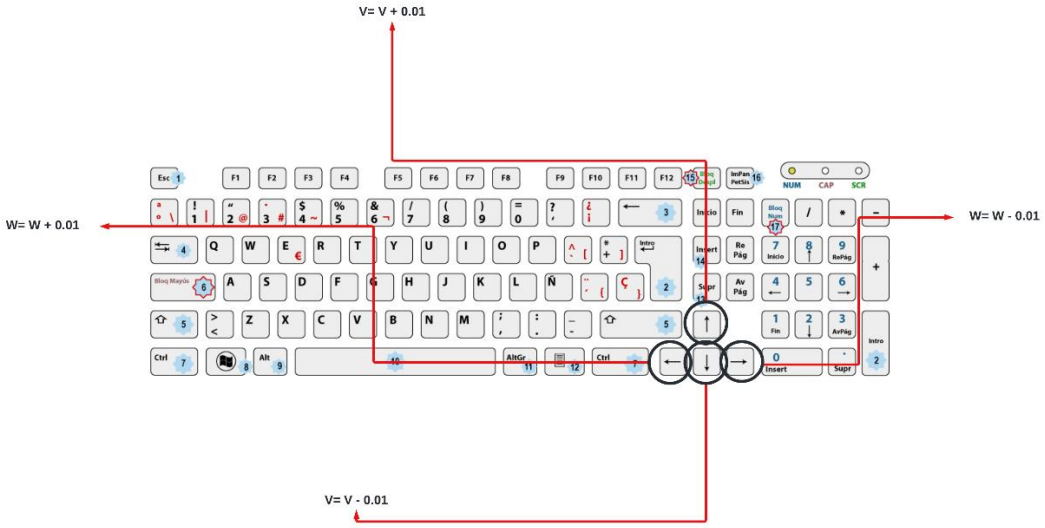



*Figura 29, Teclas de movimiento de TELE.*

***SET:*** Permite asignar una velocidad lineal y angular del robot con las ventanas de introducción de texto *set V* y *set W*.

***STOP:*** Ajusta la velocidad lineal y angular (V y ω) del robot a 0.

**CALIBRACIÓN*:*** Detiene el robot y activa el cálculo del *offset* del giroscopio (apartado 3.1.2) durante 5 segundos.

***LOAD FILE:*** Permite cargar un archivo de la trayectoria en formato txt generado durante la aplicación de navegación del robot y mostrarla gráficamente.

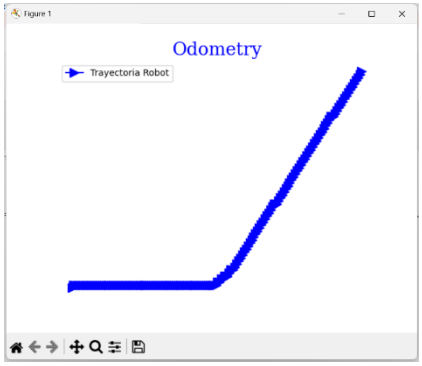



*Figura 30, Trayectoria robot de fichero txt.*

Además, se ha añadido un menú de opciones con herramientas extra y la opción de cerrar la consola de telemanipulación.

Como herramientas extra, se ha añadido la posibilidad de limpiar el fichero txt generado, cambiar el puerto de comunicación y el nombre del fichero. Por último, "*inicializar el robot"* inicializa todas las variables y funciones del robot en caso de cualquier fallo de código o del propio robot.

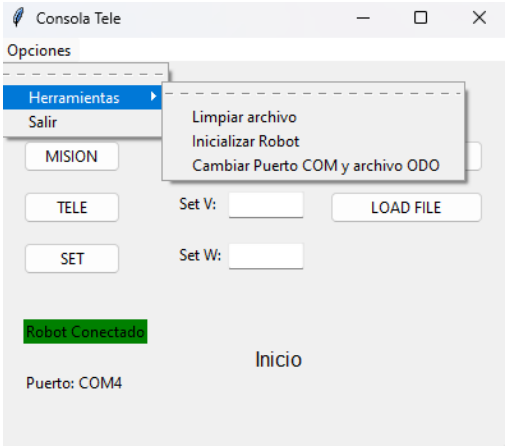
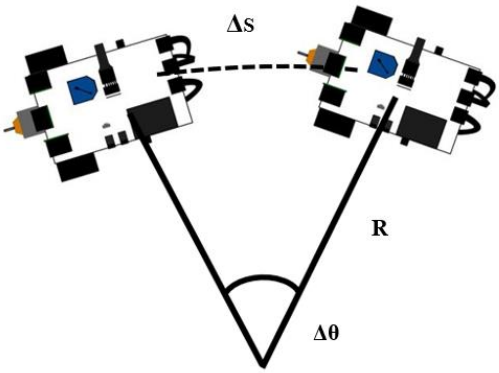



*Figura 31, Menú de opciones.*

### 3.3.2 Aplicación de odometría:

Para poder obtener la odometría, primero es indispensable disponer de la información necesaria. De ahí que obtengamos los datos necesarios del robot gracias a los *drivers Odometry* y *Protocool* (apartado 3.1.3 y 3.1.4).

La odometría se calcula a partir de la velocidad lineal V y angular W medidas por el robot en ese instante. De esta forma se obtienen las coordenadas X,Y y el angulo de giro $\theta$.




$$R = \frac{L}{2}\frac{(wd + wi)}{(wd - wi)} = \frac{V}{W}$$

*Figura 32, Estimación de la localización y modelo cinemático*

Para calcular estos valores, se calculan sus variaciones (dx, dy y dtheta) y con un periodo de control $\boldsymbol{T_{control}}$ de manera que:

$$\text{dtheta} = (\text{Wactual} * \text{Tcontrol})$$

$$Si\ Wactual^2 < 0.001 \rightarrow Wactual \approx 0$$

$$ds = Vactual * Tcontrol$$

$$Si\ Wactual^2 > 0.001 \rightarrow Wactual \neq 0$$

$$ds = \frac{Vactual}{Wactual} * Tcontrol$$

$$dx = ds * \cos\left(\theta + \frac{dtheta}{2}\right)$$

$$dy = ds * \sin\left(\theta + \frac{dtheta}{2}\right)$$

Así con las variaciones podemos calcular los valores de la odometría:

$$\boldsymbol{\theta} = \theta + dtheta$$
$$\boldsymbol{X} = X + dx$$
$$\boldsymbol{Y} = Y + dy$$

Los valores obtenidos son relativos, es decir tomando como sistema de referencia el propio robot (ROB), para poder representarlos, se cambia a un sistema de referencia general, denominado ABS donde podemos observar la posición del robot:

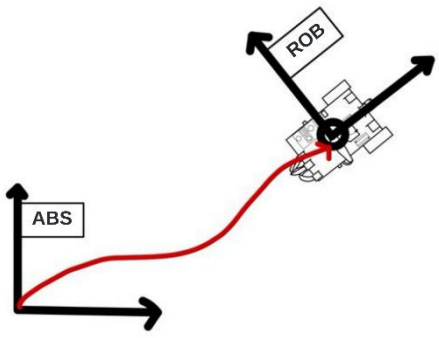



*Figura 33, cambio de base*

A la vez que se representa la trayectoria del robot, para cada posición se obtienen los valores de los sensores ultrasonidos, de esta manera se representa una nube de puntos junto a la trayectoria dibujando las superficies alrededor del robot véase figura 34. Para ello, hay que hacer su correspondiente transformación para poder representarlo.

Tomando los valores de los ultrasonidos como dL ultrasonido izquierdo y dR ultrasonido derecho, realizamos el cambio de base y lo añadimos a la posición:

$$(X\ Y) + (dL\ dL) * \begin{pmatrix} \cos(\theta + 90º) \\ \cos(\theta + 90º) \end{pmatrix} = \begin{pmatrix} dLx' \\ dLy' \end{pmatrix}$$
$$(X\ Y) + (dR\ dR) * \begin{pmatrix} \cos(\theta - 90º) \\ \cos(\theta - 90º) \end{pmatrix} = \begin{pmatrix} dRx' \\ dRy' \end{pmatrix}$$

Al inicio, el programa de odometría se planteó en ***Matlab*** donde se realizaron pruebas mientras se usaba la antigua aplicación de consola. Una vez se creó la nueva consola en Python, todo el cálculo de odometría se migró a esta consola, de manera que pudieran trabajar de manera conjunta, mejorando mucho la experiencia de usuario (apartado 3.3.1).

Una vez se active en la consola la orden de MISION una ventana emergente aparece mostrando la odometría en tiempo real y la nube de puntos donde se dibuja lo que esta "viendo" el robot. Pudiendo interpretar la zona que está recorriendo.

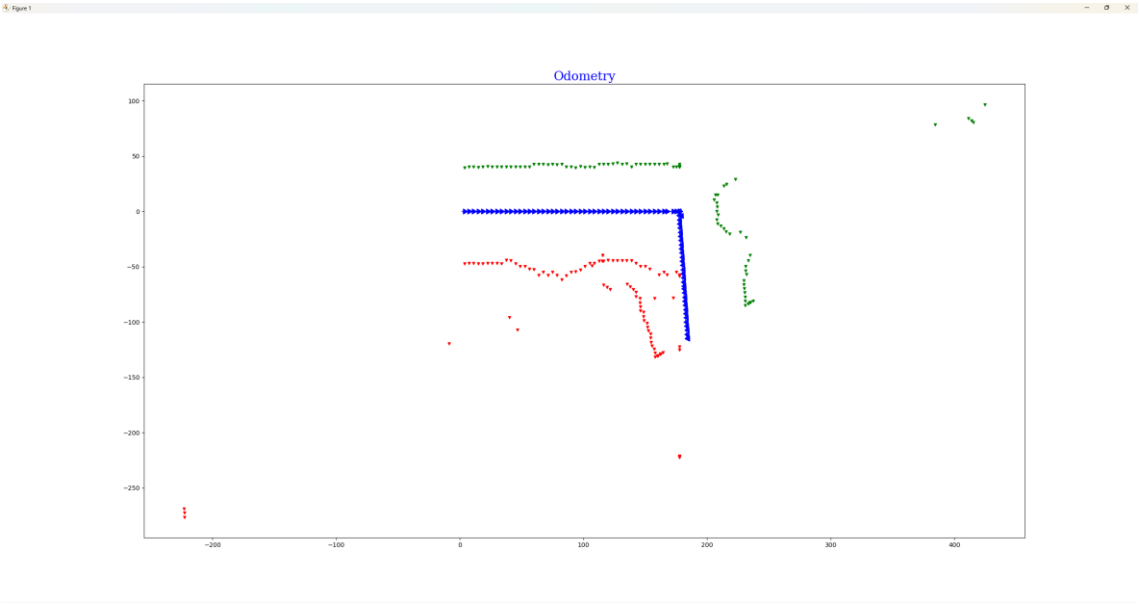



*Figura 34, Ventana gráfica de odometría, en azul trayectoria del robot, en verde valores sensor ultrasonido izquierdo y en rojo del derecho*

La odometría depende totalmente de las medidas obtenidas de los *encoders*, estos sensores dependen de una acción mecánica, en este caso de la fuerza transmitida a las ruedas, esto significa que un deslizamiento del robot o una rotación excesiva conlleva un error en la medida y, por lo tanto, en la odometría. De ahí que sea importante la aplicación de navegación y el control de las velocidades (apartado 3.2).

La nube de puntos es práctica para entornos tipo pasillos donde el robot obtiene medidas similares de las paredes dibujando la forma de la zona. En cambio, para una sala como puede ser un salón de un hogar, al encontrarse con diferentes objetos tales como sofás, sillas, mesas… conforme el robot avance a lo largo de la sala en distintas direcciones, la nube de puntos será muy difusa.

## 3.4 Análisis de tiempo Real:

Una vez se han realizado las distintas mejoras y actualizaciones del robot, se procede a analizar el sistema con los correspondientes cálculos para verificar su funcionamiento:

### 3.4.1 Estructura de las tareas y servidores

Como se ha mencionado en el apartado 1.1.1, la estructura de un sistema de tiempo real consta de la panificación de distintas tareas (apartados 3.1 y 3.2) y servidores (apartado 3.1.5) para compartir información y gestionar las diversas funciones de las que disponen:

La estructura de las tareas con sus prioridades y la relación con los servidores queda planificada tal que:

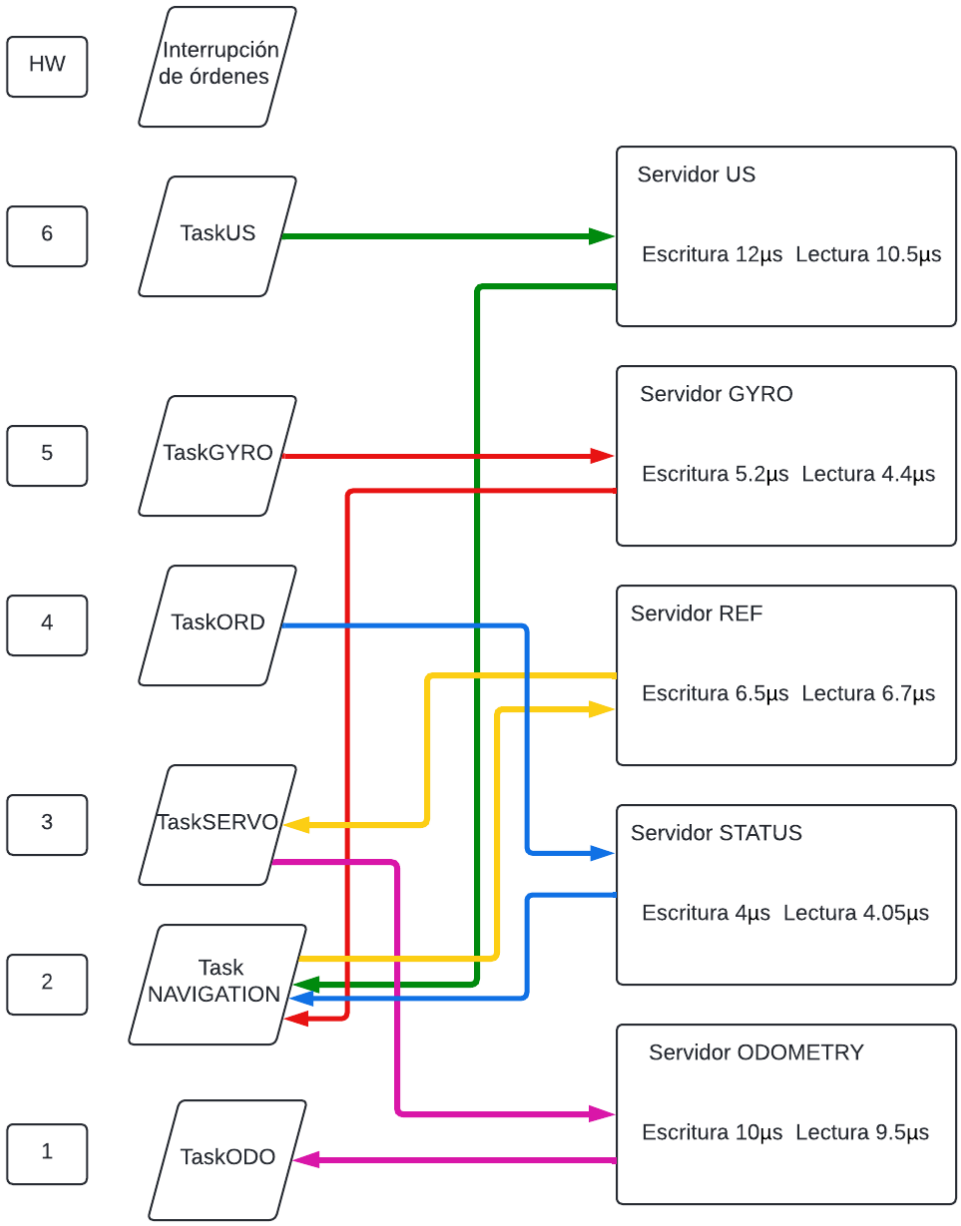



*Figura 35, Esquema tareas y servidores con prioridades.*

Para el control de los semáforos y tareas, se dispone de dos tipos de interrupciones *Hardware HW*, el **reloj** del propio sistema (*Clock*) y una **interrupción esporádica** de las órdenes recibidas del ordenador.

### 3.4.2 Medida de tiempos de cómputo

El siguiente requerimiento de un sistema de tiempo real es el de controlar eventos y su temporización (apartado 1.1.1). Para Garantizar los plazos especificados para cada tarea, se realiza un análisis de los tiempos de ejecución de las tareas críticas.

Para poder realizar la medida de tiempos en CCS, gracias al módulo de ***SYS/BIOS*** (apartado 1.3.2) se dispone de las funciones *"log_write1" y "log_write3"* con las cuales se puede realizar registros a modo de semáforo (activando a inicio de la tarea y deteniendo al final) para poder comprobar los tiempos de ejecución máximos de cada tarea y poder estudiar el peor escenario posible.

```
313 Void TaskUS (UArg a0, UArg a1)
314 {
315
316     Init_US () ;                                  // Initialize Ultrasound
317     //NUEVAS CALIBRACIONES
318     Calibrate_US(LEFT,0.0551,4.3516); //1º pendiente y 2º offset calibrar al principio
319     Calibrate_US(RIGHT,0.0569,1.5336);
320     Calibrate_US(FRONT,0.0564,2.3776);
321    while (1) {
322        Semaphore_pend (TaskUS_sem, BIOS_WAIT_FOREVER) ;
323
324      Log_write1(UIABenchmark_start, (xdc_IArg)"TaskUS");
325      Log_write3(UIABenchmark_startInstanceWithAdrs, (IArg)"Func: id=%x, Fxn=%x", 0, (UArg)&TaskUS);
326        Left_distance = Read_Distance (LEFT) ;
327        Front_distance = Read_Distance (FRONT) ;
328        Right_distance = Read_Distance (RIGHT) ;
329
330        write_left_distance (Left_distance) ;
331        write_front_distance (Front_distance) ;
332        write_right_distance (Right_distance) ;
333      Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskUS");
334      Log_write3(UIABenchmark_stopInstanceWithAdrs, (IArg)"Func: id=%x, Fxn=%x", 0, (UArg)&TaskUS);
335
336     }
337 }
```

*Figura 36, ejemplo de uso de las funciones log_write para TaskUS.*

Así con la herramienta de "*RTOS Analysis*" con *Execution Graph* y *Duration* se aprecia gráficamente como se activan las distintas tareas y los tiempos que están trabajando.

*Live Session | *Execution Graph | *Duration: Summary | *Printf Logs

| | Source | Count | Min | Max | Average | Total | Percent |
|---|---|---|---|---|---|---|---|
| 1 | C28xx_CPU1, TaskGYRO | 8 | 42560 | 43290 | 42,948.8 | 343,590 | 16.9 |
| 2 | C28xx_CPU1, TaskNAV | 1 | 27760 | 27760 | 27,760.0 | 27,760 | 1.4 |
| 3 | C28xx_CPU1, TaskODO | 1 | 861680 | 861680 | 861,680.0 | 861,680 | 42.3 |
| 4 | C28xx_CPU1, TaskSERVOS | 2 | 28870 | 28870 | 28,870.0 | 57,740 | 2.8 |
| 5 | C28xx_CPU1, TaskUS | 7 | 105260 | 107280 | 106,602.9 | 746,220 | 36.6 |

*Figura 37, Captura de pantalla de CCS, uso de las tareas con execution graph.*

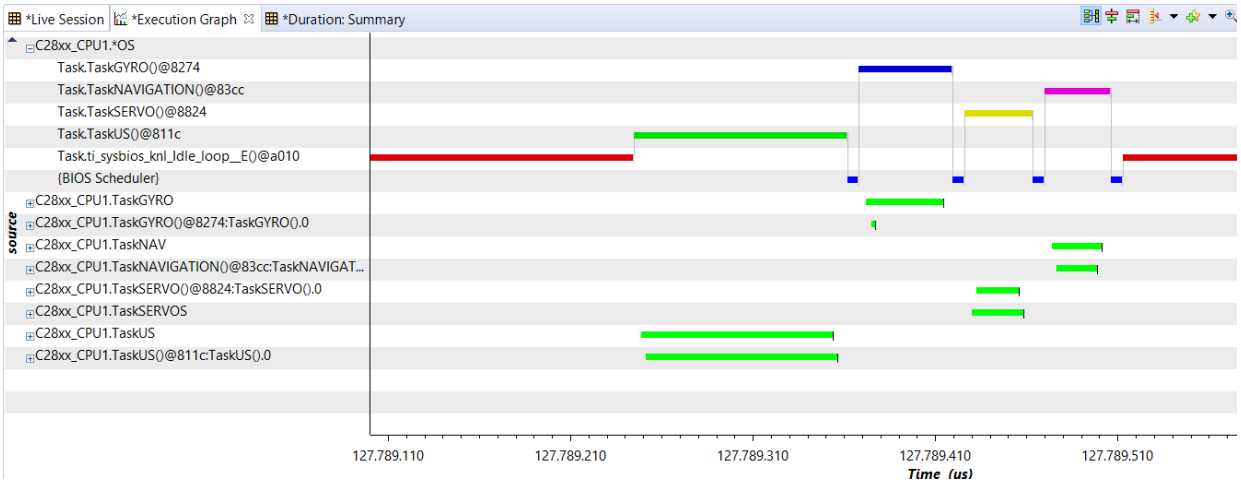



*Figura 38, captura de pantalla de CCS con los tiempos de cómputo en µs de las tareas con duration.*

De esta manera, en la figura 39 se recogen los tiempos principales de cada tarea y su prioridad:

| Tareas | Prioridad | Tiempo de Computo C (µs) | Periodo T (ms) | Plazo de Respuesta D (ms) |
|---|---|---|---|---|
| Clock | HW | 5 | 10 | - |
| Interrupción ORD | HW | 7 | 2 | - |
| TaskUS | 6 | 106.55 | 5 | 5 |
| TaskGYRO | 5 | 42.97 | 5 | 5 |
| TaskORD | 4 | 27.54 | 6.25 | 6.25 |
| TaskSERVO | 3 | 20.91 | 25 | 25 |

| | | | | |
|---|---|---|---|---|
| TaskNAVIGATION | 2 | 29.37 | 50 | 50 |
| TaskODO | 1 | 872.50 | 250 | 250 |

*Figura 39, Tabla con prioridades y tiempos de las interrupciones.*

Para calcular el periodo T de la tarea de órdenes ***TaskORD***, la cual como se ha comentado (apartado 3.4.1) es esporádica, se sabe que las órdenes se transmiten a 9600bps (apartado 2.2.1), donde cada byte de información son 8 bits de información y además se incluyen +1 bit de start y +1 bit de stop, por lo tanto, la velocidad de envío de datos es:

$$9600bps = \frac{9600}{10bits} = 960\ bytes/s$$

El tiempo entre interrupciones será:

$$\frac{1}{960} = 1.0416\ ms$$

Si la orden más corta a enviar es " #STOP /r" que corresponde a 6 bits, el periodo será:

$$Tord = 6 * 1.0416 = 6.25\ ms = Dord$$

### 3.4.3 Cálculo de bloqueos

Al tratarse de un sistema con herencia de prioridad, se pueden producir bloqueos entre las distintas tareas. Se estudian los posibles bloqueos producidos.

Para realizar los cálculos de los bloqueos, es necesario conocer las prioridades de los servidores. Estas se eligen mediante el criterio de techo de prioridad, es decir, los servidores heredarán la prioridad de la tarea más prioritaria que los use, según la planificación en la figura 35:

| **Servidor** | **Prioridad** |
|---|---|
| US | 6 |
| GYRO | 5 |
| REF | 3 |
| STATUS | 4 |
| ODOMETRÍA | 3 |

*Figura 40, tabla con prioridades asignadas a los servidores.*

Teniendo en cuenta estos datos se calculan los bloqueos:

Para el reloj y la interrupción de órdenes el bloqueo es 0 al ser de prioridad por HW.

Para el resto de las tareas se aplican bloqueos por techo de prioridad, donde afectan los tiempos de escritura y lectura de los servidores con misma o menor prioridad que los llame una tarea inferior a la que se calcule el bloqueo (figura 35):

$$TaskUS \rightarrow Servidores = 12\mu s, B_{us} = 12\mu s$$

$$TaskGYRO \rightarrow Tareas = 5.2\mu s + 12\mu s = 17.2\mu s$$
$$\rightarrow Servidores = 5.2\mu s + 12\mu s = 17.2\mu s, B_{GYRO} = 17.2\mu s$$

$$TaskORD \rightarrow Tareas = 4.05\mu s + 12\mu s + 17.2\mu s = 33.25\mu s$$
$$\rightarrow Servidores = 4.05\mu s + 12\mu s + 17.2\mu s = 33.25\mu s,$$
$$B_{ORD} = 33.25\mu s$$

$$TaskSERVO \rightarrow Tareas = 6.7\mu s + 10\mu s = 16.5\mu s$$
$$\rightarrow Servidores = 6.7\mu s + 10\mu s = 16.5\mu s, B_{SERVO} = 16.5\mu s$$

$$TaskNAVIGATION \rightarrow Tareas = 10\mu s, B_{NAV} = 10\mu s$$

$$TaskODO \rightarrow No\ tiene\ bloqueos\ al\ ser\ la\ tarea\ de\ menor\ prioridad, B_{ODO} = 0\mu s$$

Por lo tanto, los bloqueos resultan:

| Tareas | Bloqueos |
|---|---|
| Clock | 0 |
| Interrupción ORD | 0 |
| TaskUS | 12 µs |
| TaskGYRO | 17.2 |
| TaskORD | 33.25 µs |
| TaskSERVO | 16.5 µs |
| TaskNAVIGATION | 10 µs |
| TaskODO | 0 |

*Figura 41, Tabla con valores de bloqueo para las tareas.*

### 3.4.4 verificación de plazos

Para garantizar el cumplimiento de plazos de las tareas se calculan los tiempos de trabajo ***W.*** Para ello, los tiempos de trabajo de las tareas deben ser menores al plazo de respuesta marcado ***D*** (figura 39). En este sistema con 6 tareas periódicas la condición suficiente, es el cumplimiento de los plazos de respuesta ($D$) determinados:

$$W_i = \left\lceil \frac{D_k}{T_k} \right\rceil C_k + \left\lceil \frac{D_j}{T_j} \right\rceil C_j + \cdots + C_i + B_i < D_i$$

Para cada tarea resulta:

**Ultrasonidos (US):**

$Wus(Dus) = Cus + Bus = 106.55 + 12 = 118{,}55\ \mu s < Dus = 5ms$

**Giroscopio (GYRO):**

$Wgyro(Dgyro) = \left\lceil \frac{Dgyro}{Tus} \right\rceil * Cus + Cgyro + Bgyro = \left\lceil \frac{5}{5} \right\rceil * 106.55 + 42.97 + 17.2 = 166.72\ \mu s < Dgyro = 5ms$

**Órdenes (ORD):**

$Word(Dord) = \left\lceil \frac{Dord}{Tus} \right\rceil * Cus + \left\lceil \frac{Dord}{Tugyro} \right\rceil * Cgyro + Cord + Bord = \left\lceil \frac{6.25}{5} \right\rceil * 106.55 + \left\lceil \frac{6.25}{5} \right\rceil * 42.67 + 27.54 + 33{,}25 = 247.315\ \mu s < Dord = 6.25ms$

**Servos (SERVO):**

$Wservo(Dservo) = \left\lceil \frac{Dservo}{Tus} \right\rceil * Cus + \left\lceil \frac{Dservo}{Tgyro} \right\rceil * Cgyro + \left\lceil \frac{Dservo}{Tord} \right\rceil * Cord + Cservo + Bservo = \left\lceil \frac{25}{5} \right\rceil * 106.55 + \left\lceil \frac{25}{5} \right\rceil * 42.97 + \left\lceil \frac{25}{6.25} \right\rceil * 27.54 + 20.91 + 16.5 = 895.17\ \mu s < Dservo = 25ms$

**Navegación (NAVIGATION):**

$$Wnav(Dnav) = \left\lceil \frac{D_{nav}}{T_{us}} \right\rceil * Cus + \left\lceil \frac{D_{nav}}{T_{gyro}} \right\rceil * Cgyro + \left\lceil \frac{D_{nav}}{T_{ord}} \right\rceil * Cord + \left\lceil \frac{D_{nav}}{T_{servo}} \right\rceil * Cservo + Cnav + Bnav$$
$$= \left\lceil \frac{50}{5} \right\rceil * 106.55 + \left\lceil \frac{50}{5} \right\rceil * 42.97 + \left\lceil \frac{50}{6.25} \right\rceil * 27.54 + \left\lceil \frac{50}{25} \right\rceil * 20.91 + 29.37 + 10 = 1796.71\ \mu s = 1{,}79ms < Dnav = 50ms$$

**Odometría (ODO):**

$$Wodo(Dodo) = \left\lceil \frac{D_{odo}}{T_{us}} \right\rceil * Cus + \left\lceil \frac{D_{odo}}{T_{gyro}} \right\rceil * Cgyro + \left\lceil \frac{D_{odo}}{T_{ord}} \right\rceil * Cord + \left\lceil \frac{D_{odo}}{T_{servo}} \right\rceil * Cservo + \left\lceil \frac{D_{odo}}{T_{nav}} \right\rceil * Cnav + Codo + Bodo$$
$$= \left\lceil \frac{250}{5} \right\rceil * 106.55 + \left\lceil \frac{250}{5} \right\rceil * 42.97 + \left\lceil \frac{250}{6.25} \right\rceil * 27.54 + \left\lceil \frac{250}{25} \right\rceil * 20.91 + \left\lceil \frac{250}{50} \right\rceil * 29.37 + 872.50 = 9806.05\mu s = 9.8ms < Dodo = 250ms$$

Por lo tanto, queda concluido que todas las tareas cumplen sus plazos de respuesta.

Además, por añadir, calculamos el **tiempo libre** del microprocesador de las tareas (figura 39):

$$Tiempo\ U = \Sigma \frac{Ci}{Ti}$$

$$U = \frac{0{,}005}{10} + \frac{0{,}007}{2} + \frac{0{,}106}{5} + \frac{0{,}043}{5} + \frac{0{,}028}{6{,}25} + \frac{0{,}021}{25} + \frac{0{,}029}{50} + \frac{0{,}872}{250} = 0{,}043188$$

De esta manera:

$$Tiempo\ Libre\ CPU = 1 - U = 0{,}9568 = 95{,}68\%\ de\ tiempo\ libre$$

Con el análisis realizado se concluye que el funcionamiento de este sistema de tiempo real es **correcto.**

# 4. Actualizaciones:

## 4.1 Introducción nuevos sensores

Hasta ahora todas las mejoras añadidas al robot se han realizado sin modificar su hardware, pero durante el proyecto, han surgido necesidades que se podrían haber solventado de manera más rápida con nuevos sensores pudiendo conseguir mejores resultados en ciertas áreas.

Aquí un despliegue y explicaciones de sensores que podrían ser de ayuda:

### 4.1.1 US

Añadir más ultrasonidos: La configuración actual de 3 sensores (frontal, derecha e izquierda) supone un problema al tener puntos ciegos como se ha podido comprobar con la navegación (apartado 3.2.2). cambiando la configuración de **3 a 5** sensores se añaden dos en las diagonales, de manera que se puedan evitar los puntos ciegos del robot.

Lógicamente este cambio no sería directo ya que habría que adaptar la placa actual para introducir los nuevos sensores al igual que el *driver*, pero eliminaría el problema de los puntos ciegos actual además de facilitar el manejo del robot durante la navegación.

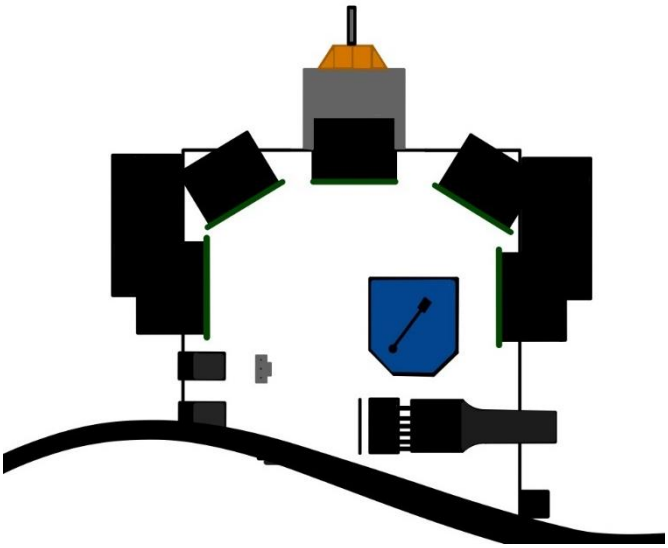


*Figura 42, Configuración nueva de ultrasonidos.*

### 4.1.2 Acelerómetro

El giroscopio nos permite medir la velocidad angular del robot y como se ha visto en el apartado 3.2.3 se ha usado para implementar un lazo cerrado de ω.

Con un acelerómetro, se podría medir las aceleraciones de este y así poder crear un lazo de control de la velocidad más factible que el diseñado con ultrasonidos (apartado 3.2.3). Además, la configuración de giroscopio + acelerómetro es de las más típicas en este tipo de aplicaciones por lo que hay mucha documentación al respecto y aplicaciones ya desarrolladas para estos entornos.

Otro posible sensor de uso similar son los sensores laser tipo **LiDAR** *(light detection and ranging)*, donde, con un uso parecido al que podría realizar un robot aspiradora tipo *roomba*, donde el láser realiza una medida en 360º (figura 43) de lo que rodea al vehículo, de manera que constantemente podamos saber dónde se sitúa. Con este tipo de sensores, la aplicación de navegación podría ser mucho más extensa, consiguiendo además un mejor resultado de mapeado en la parte gráfica y pudiendo implementar algoritmos de tipo SLAM (*simultaneous localization and mapping*).

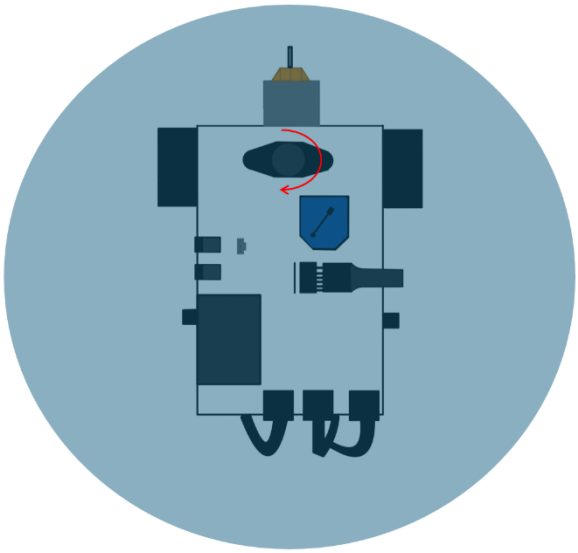


*Figura 43, Configuración nueva con LiDAR*

### 4.2.3 Placa arduino

La placa de TI dispone de una entrada tipo **SCIB** con la que es posible compartir información o comunicarse, es decir, se puede añadir una placa externa de tipo Arduino que pueda comunicarse con la placa del robot.

Se podría utilizar un módulo con antena wifi y bluetooth incorporado lo que podría mejorar la comunicación del robot con otro ordenador, donde en vez de usar el módulo Xbee (alcance limitado) se puede crear un protocolo cerrado a través de wifi o bluetooth para establecer la conexión.

### 4.2.4 Cámara CMOS

También, se puede implementar una **cámara** tipo CMOS que, junto a un algoritmo de detección basado en visión por computación, se establezca un algoritmo que pueda reconocer objetos, animales o personas. De esta manera, la aplicación de navegación mejoraría notoriamente ya que esta cámara se encargaría del reconocimiento de la zona que rodea al robot, además del resto de sensores.

# 5. Conclusiones

Como se mencionaba al inicio de este trabajo, el *hardware* del robot y la situación inicial de la que se partía era básica, pero se ha demostrado cuanto puede mejorar el funcionamiento de este tipo de sistemas sin entorpecer su correcta ejecución.

Para la aplicación demostrativa de navegación la adición de un controlador para la posición facilita el uso de un algoritmo más complejo para mejorar el movimiento del robot, así como facilitar nuevos desarrollos por parte de los estudiantes.

De la misma forma, la nueva consola facilita el manejo del robot, así como comprender mejor su movimiento y el entorno que lo rodea gracias a la aplicación de la odometría.

Con los resultados obtenidos en el análisis del sistema se facilita el trabajo a los próximos estudiantes de la asignatura ya que se verifica el cumplimiento de plazos de respuesta para los distintos *drivers* del robot, teniendo solo que analizar el respectivo código que añada cada alumno nuevo.

Por último, en el desarrollo de este trabajo se ha aprendido sobre la importancia de maximizar el uso de los sensores y funciones de los que se dispone, y no depender de añadir componentes más caros o complejos. Esto ha permitido explorar nuevas vías y buscar la mejor solución y alternativa para los problemas planteados.

# 6. Bibliografía:


[1] Stankovic, J. (1988, 1 octubre). *Misconceptions about real-time computing: a serious problem for next-generation systems*. IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/7053

[2] Burns, A y Wellings,A. (2009). *Real-Time Systems and Programming Languages. Addison Wesley*

[3] Villarroel, José Luis y Abadía, David (2023). *Presentación Sistemas de Tiempo Real [Diapositivas de PowerPoint]. Departamento de Informática e Ingeniería de Sistemas. Universidad de Zaragoza.*

[4] Villarroel, José Luis y Abadía, David (2023). SYS/BIOS Concurrencia y Tiempo *[Diapositivas de PowerPoint]. Departamento de Informática e Ingeniería de Sistemas. Universidad de Zaragoza.*

[5] MaxBotix. (2015). *LV-MaxSonar-EZ Series* [Dataset] https://maxbotix.com/pages/lv-maxsonar-ez-datasheet?srsltid=AfmBOooS_bW_Lhlco1W5A-rMtXIq26u07i1FscP5CwxXC-jpbYhO5IrY

[6] Digilent. (2011).P*modGYRO Reference Manual* [Dataset] https://digilent.com/reference/pmod/pmodgyro/reference-manual?srsltid=AfmBOor3aIiNSgqpkkaG2SwSc66cHYQNUMkafvr0QIeQB4AuOn81vEAu

[7] STMicroelectronics. (2010). *L3G4200D MEMS motion sensor* [Dataset] https://www.alldatasheet.es/datasheet-pdf/pdf/332531/STMICROELECTRONICS/L3G4200D.html

[8] ON Semiconductor. (2013). *LB1836M ow-Saturation Bidirectional Motor Driver for Low-Votage Drive* [Dataset] https://www.alldatasheet.com/datasheet-pdf/pdf/536778/ONSEMI/LB1836M.html

[9] Digi International. (2009). *XBee/XBee-PRO RF Modues* [Dataset] https://www.digi.com/resources/library/data-sheets/ds_xbeemultipointmodules

[10] Texas Instruments. (2015). *LLAUNCHXL-F28377S User's Guide*. https://www.ti.com/lit/ug/sprui25d/sprui25d.pdf?ts=1724743633311

[11] Texas Instruments. (2015). *F28377S DS* [Dataset] https://www.ti.com/lit/ds/symlink/tms320f28377s.pdf?ts=1724778832838&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMS320F28377S%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Depd-c2x-null-44700045336317350_prodfolderdynamic-cpc-pf-google-wwe_int%2526utm_content%253Dprodfolddynamic%2526ds_k%253DDYNAMIC+SEARCH+ADS%2526DCM%253Dyes%2526gad_source%253D1%2526gclid%253DCjwKCAjw8rW2BhAgEiwAoRO5rM_QvuVdl3-lqu4ueHqpz3c7PqJ-Za1dcabGoDv2-ygc8QMrZMO7EhoC-iEQAvD_BwE%2526gclsrc%253Daw.ds

[12] LEGO. (2006). *LEGO Education Sensor 9843 NXT*. https://www.lego.com/en-es/product/touch-sensor-9843

[13] LEGO. (2006). *LEGO Education Sensor 9842 NXT* https://www.electricbricks.com/lego-education-mindstorms-nxt-servo-motor-interactivo-nxt-p-252.html

[14] Ogata, Katsuhiko. (1998). Capítulo 10: Controladores PID y sistemas de control con dos grados de libertad. Sebastián Dormido Canto. *Ingeniería de control moderna* (681-752). Cuarta edición. Pearson (Ed.).

# Anexos:

## Anexo 1: Código CCS

### A.1.1 Programa *Main*:

```
/*****************************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : main_Robot.c
*/
/* version  : 4
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernández
*/
/* description : Main program of control and navigation of the robot
*/
/*
*/
/*****************************************************************************
*/



/*****************************************************************************
*/
/*                          Used modules
*/
/*****************************************************************************
*/

#include <stdio.h>

#include <xdc/std.h>

#include <xdc/runtime/Error.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>

#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Clock.h>

#include <xdc/runtime/Log.h>
#include <ti/uia/events/UIABenchmark.h>

#include "F2837xS_device.h"
#include "pwm.h"
#include "encoder.h"
#include "us.h"
#include "GYRO.h"
#include "bumper.h"

#include "ROBOT_types.h"
#include "servos.h"
#include "servidores.h"
```

```
#include "protocol.h"
#include "Odometry.h"
#include <ti/sysbios/knl/Swi.h>
#include "Wall_Follower.h"
#include "clock.h"

/*************************************************************************
*/
/*                        Global variables
*/
/*************************************************************************
*/

Clock_Handle clockTaskUS ;
Task_Handle TaskUS_Handle ;
Semaphore_Handle TaskUS_sem ;

Clock_Handle clockTaskGYRO ;
Task_Handle TaskGYRO_Handle ;
Semaphore_Handle TaskGYRO_sem ;

Clock_Handle clockTaskNAVIGATION ;
Task_Handle TaskNAVIGATION_Handle ;
Semaphore_Handle TaskNAVIGATION_sem ;

Clock_Handle clockTaskSERVO ;
Task_Handle TaskSERVO_Handle ;
Semaphore_Handle TaskSERVO_sem ;

Task_Handle TaskORD_Handle ;
Semaphore_Handle TaskORD_sem ;
Swi_Handle mySwi;

Clock_Handle clockTaskODO ;
Task_Handle TaskODO_Handle ;
Semaphore_Handle TaskODO_sem ;

const unsigned int US_Period = 5 ;
const unsigned int GYRO_Period = 5 ;
const unsigned int Sevo_Period = 25  ;
const unsigned int Navigation_Period = 50 ;
const unsigned int Odometry_Period = 250;


/*************************************************************************
*/
/*                        Tasks                                    */
/*************************************************************************
*/

extern void DelayUs (volatile Uint16 Usec) ;
extern void Init_System (void) ;

Void TaskUS(UArg arg0, UArg arg1) ;
void TaskUS_release (void) ;

Void TaskGYRO(UArg arg0, UArg arg1) ;
void TaskGYRO_release (void) ;

Void TaskNAVIGATION(UArg arg0, UArg arg1) ;
void TaskNAVIGATION_release (void) ;

Void TaskSERVO(UArg arg0, UArg arg1) ;
```

```
void TaskSERVO_release (void) ;

Void TaskORD (UArg a0, UArg a1);
Void TaskORD_release (void);

Void TaskODO (UArg a0, UArg a1);
Void TaskODO_release (void);

////////////////////////////////****Functions****////////////////////////////
////////////

void Action (float V, float U); //Set V and W to servo motors
void MISION_4 (void);//Algorythm of navigation
float Comp(int i,float dF_ant);//Algorythm of compensation lineal velocity by
US
void CalibGyro(float tiempo);//Get offset to gyroscope by t seconds

void Lineal_velocity (void);
void velocity (float Wref, float Vlineal); //Velocity Control;
void Position(float titaref, float Vlineal);//Position Control
float rectifica(float distL,float distR,float d_ant);//Blind Spots
void Seacerca(float dref,float US);
float P_ciego(float us, int t,float dmax, float dmin);
void Inicializa(void); //Initialize all variables to zero

////////////////////////////////****Variables****////////////////////////////
////////////

//Counters
unsigned int caso=0; //rotate
unsigned int caso2=0; //reverse
unsigned int caso3=0; //error
unsigned int cuenta_gyro=0;
int Cuenta_US=0;

//Const of robot and board
const float r = 0.027 ;   // Wheel radius
const float d = 0.13 ;    // Distance between wheels
const float dhalf=0.065;
const float Tsw = 0.0025; //Clock timer 5ns
char bumper;

REFERENCE Robot_Reference_N, Robot_Reference_S ;  // Global for debugging
STATUS et_status;
////////////////////////////////*******US*******////////////////////////////
////////////
float dl, df, dr;
float Left_distance, Front_distance, Right_distance ;

////////////////////////////////******ODOM******////////////////////////////
////////////
ODOMETRY_Trayect odoT= {0,0,0,0,0,0};

enum {INICIAL, SIGUE,FRENTE,IZQUIERDA,DERECHA,RODEADO,MARCHATRAS,SIGUEMUROS}
EstadoNav ;  // States of Robot
unsigned int Ti, siguiente,periodo,j; //set timer

////////////////////////////////******GYRO******////////////////////////////
////////////
float vX_b,vY_b,vZ_b=0;
float wZ=0,wZ_task=0;
float Wpi=0;
```

```
////////////////////////////////////*******PI*******//////////////////////////////
///////////
float PI (float xref, float x,float kp, float ki,float* xenc, float Umax,
float Umin);

////////////////////////////////////******Tuned Constants for
PI******//////////////////////

float kp=0.5; // For W
float ki=0.01;

float kp_V=0.1; //For Lineal V (not used)
float ki_V=0.001;

float kpx=1; //For position theta
float kix=0.0001;

float* Ix=0;
float* Iw=0;
float* Iwl=0; //Pointers
float* Iwr=0;
float* Iv=0;

float tita_abs=0;
float Wpos=0;
float titaref=0;
float Wl=0, Wr =0,Wl_ref=0, Wr_ref=0,Wl_ref_pi=0,
Wr_ref_pi=0,U_l=0,U_r=0,Wabs=0, Vl_T=0, Vr_T=0;

float Wciego=0;
float d_ciegoL=0,d_ciegoR=0;
float cont=0;
float alpha=0;

////////////////////////////////////***LINEAL_COMPENSATION**/////////////////////
///////////
float V_comp=0;
float dF_ant=0;
float V_pi=0;
unsigned int hayComp=0;
////////////////////////////////////////////////////////////////////////////////
//////////


/*****************************************************************************
*/
/*                          Main
*/
/*****************************************************************************
*/


Void main() {




    Clock_Params clockParams;
    Task_Params taskParams ;

    Swi_Params swiParams;
```

```
    Clock_Params_init(&clockParams);
    clockParams.period = US_Period;
    clockParams.startFlag = TRUE;                   // start immediately
    clockTaskUS = Clock_create((Clock_FuncPtr)TaskUS_release, 10,
&clockParams, NULL); //10 US period

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 255 ; // si no cabe ampliar
    taskParams.priority = 6 ; //5 OEM>> numero mas prioridad
    TaskUS_Handle = Task_create((Task_FuncPtr)TaskUS, &taskParams, NULL);

    TaskUS_sem = Semaphore_create(0, NULL, NULL);

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



    Clock_Params_init(&clockParams);
    clockParams.period = GYRO_Period;
    clockParams.startFlag = TRUE;                   // start immediately
    clockTaskGYRO = Clock_create((Clock_FuncPtr)TaskGYRO_release, 10,
&clockParams, NULL); //10 US period

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 255 ; // si no cabe ampliar
    taskParams.priority = 5 ; //4 OEM >> numero mas prioridad
    TaskGYRO_Handle = Task_create((Task_FuncPtr)TaskGYRO, &taskParams, NULL);

    TaskGYRO_sem = Semaphore_create(0, NULL, NULL);




////////////////////////////////////////////////////////////////////////////
///////////
    Clock_Params_init(&clockParams);
    clockParams.period = Navigation_Period ;
    clockParams.startFlag = TRUE;                   // start immediately
    clockTaskNAVIGATION = Clock_create((Clock_FuncPtr)TaskNAVIGATION_release,
10, &clockParams, NULL);

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 1024 ; //512 OEM
    taskParams.priority = 2 ; //1
    TaskNAVIGATION_Handle = Task_create((Task_FuncPtr)TaskNAVIGATION,
&taskParams, NULL);

    TaskNAVIGATION_sem = Semaphore_create(0, NULL, NULL);

    Clock_Params_init(&clockParams);
    clockParams.period = Sevo_Period ;              // every 50 Clock ticks
    clockParams.startFlag = TRUE;                   // start immediately
    clockTaskSERVO = Clock_create((Clock_FuncPtr)TaskSERVO_release, 10,
&clockParams, NULL);

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 512 ;
    taskParams.priority = 3 ;
    TaskSERVO_Handle = Task_create((Task_FuncPtr)TaskSERVO, &taskParams,
```

```
NULL);

    TaskSERVO_sem = Semaphore_create(0, NULL, NULL);

    Clock_Params_init(&clockParams);
    clockParams.period = Odometry_Period ;             // every 50 Clock ticks
    clockParams.startFlag = TRUE;                   // start immediately
    clockTaskODO = Clock_create((Clock_FuncPtr)TaskODO_release, 10,
&clockParams, NULL);

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 512 ;
    taskParams.priority = 1 ; //5
    TaskODO_Handle = Task_create((Task_FuncPtr)TaskODO, &taskParams, NULL);

    TaskODO_sem = Semaphore_create(0, NULL, NULL);

    Swi_Params_init(&swiParams);
    swiParams.priority = 1 ;
    mySwi = Swi_create((Swi_FuncPtr)TaskORD_release, &swiParams, NULL);

    Task_Params_init(&taskParams) ;
    taskParams.stackSize = 512 ;
    taskParams.priority = 4 ;
    TaskORD_Handle = Task_create((Task_FuncPtr)TaskORD, &taskParams, NULL);

    TaskORD_sem = Semaphore_create(0, NULL, NULL);

////////////////////////////////////******INIT******////////////////////////////
///////////

    Init_Protocol (mySwi) ;
    Init_Bumper();
    Init_Clock();
    siguiente= Get_Time();
    Set_Timer(5);
    Server_Create () ;


    BIOS_start();                                   // enable interrupts and
start SYS/BIOS
}

void TaskUS_release (void) {
      Semaphore_post(TaskUS_sem);
}

Void TaskUS (UArg a0, UArg a1)
{

     Init_US () ;                                    // Initialize Ultrasound
     //New Calibrations
     Calibrate_US(LEFT,0.0551,4.3516);
     Calibrate_US(RIGHT,0.0569,1.5336);
     Calibrate_US(FRONT,0.0564,2.3776);
    while (1) {
        Semaphore_pend (TaskUS_sem, BIOS_WAIT_FOREVER) ;

      //Log_write1(UIABenchmark_start, (xdc_IArg)"TaskUS");
     //Log_write3(UIABenchmark_startInstanceWithAdrs, (IArg)"Func: id=%x,
Fxn=%x", 0, (UArg)&TaskUS);
        Left_distance = Read_Distance (LEFT) ;
        Front_distance = Read_Distance (FRONT) ;
```

```
        Right_distance = Read_Distance (RIGHT) ;

        write_left_distance (Left_distance) ;
        write_front_distance (Front_distance) ;
        write_right_distance (Right_distance) ;
     // Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskUS");
    // Log_write3(UIABenchmark_stopInstanceWithAdrs, (IArg)"Func: id=%x,
Fxn=%x", 0, (UArg)&TaskUS);

     }
}




void TaskGYRO_release (void) {
      Semaphore_post(TaskGYRO_sem);
}


void TaskGYRO (UArg a0, UArg a1){

    Init_GYRO();                          // Initialize Gyroscope
    vZ_b=Offset_GYRO (Z);                 // Get offset value of axis Z

    while (1) {
            Semaphore_pend (TaskGYRO_sem, BIOS_WAIT_FOREVER) ;
          // Log_write1(UIABenchmark_start, (xdc_IArg)"TaskGYRO");
          //  Log_write3(UIABenchmark_startInstanceWithAdrs, (IArg)"Func:
id=%x, Fxn=%x", 0, (UArg)&TaskGYRO);
          wZ_task=get_GYRO( Z,vZ_b);


          write_z_axis(wZ_task);
          //  Log_write3(UIABenchmark_stopInstanceWithAdrs, (IArg)"Func:
id=%x, Fxn=%x", 0, (UArg)&TaskGYRO);
          //  Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskGYRO");

    }
}



void TaskNAVIGATION_release (void) {
      Semaphore_post(TaskNAVIGATION_sem);
}

Void TaskNAVIGATION (UArg a0, UArg a1)
{


    while (1) {
           Semaphore_pend (TaskNAVIGATION_sem, BIOS_WAIT_FOREVER) ;
           Log_write1(UIABenchmark_start, (xdc_IArg)"TaskNAV");
           df = read_front_distance () ; //1°
           dl = read_left_distance () ;  //2°
           dr = read_right_distance () ; //3°

           et_status=read_STATUS();

           bumper = Read_Bumper(); //0 press,

           wZ=read_z_axis();//Get value of TaskGyro
```

```
            tita_abs=wZ*0.005;

            if(et_status==STOP || bumper==0){

                Action(0 , 0);

                titaref=0;
            }
            if(et_status==CALIBRATION || bumper==0){

                Action(0 , 0);
                CalibGyro(5);

                titaref=0;
            }
            if(et_status==INITIALITATION || bumper==0){

                Action(0 , 0);
                Inicializa();

                titaref=0;
            }
            if(et_status==MISSION){
                EstadoNav=INICIAL;

                titaref=89;
                MISION_4(); //Algorythm of Navigation


            }//end of mision

            if(et_status==TELEMANIPULATION){
              read_T_REF(&Robot_Reference_N);
              write_REF (&Robot_Reference_N);
              if(bumper==0){
                  Action(0,0);
              }
            }


            write_REF (&Robot_Reference_N) ;
            Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskNAV");

           //siguiente = siguiente + periodo ; //Al terminar el while
          //delay_until (siguiente) ;
        }

}



void TaskSERVO_release (void) {
      Semaphore_post(TaskSERVO_sem);
}

Void TaskSERVO (UArg a0, UArg a1)
{
    Init_EPWM2 () ;
    Init_EPWM7 () ;

    Init_Encoder_right () ;
    Init_Encoder_left () ;
```

```
    Init_Servos (Sevo_Period) ;



    while (1) {
        Semaphore_pend (TaskSERVO_sem, BIOS_WAIT_FOREVER) ;
      //  Log_write1(UIABenchmark_start, (xdc_IArg)"TaskSERVOS");

        read_REF (&Robot_Reference_S) ;

        //Kinematic chain

        Wl_ref = (Robot_Reference_S.V - Robot_Reference_S.W*dhalf)/r;
        Wr_ref = (Robot_Reference_S.V + Robot_Reference_S.W*dhalf)/r ;

        // left motor control loop


        Wl = left_velocity () ;
        U_l = Rl (Wl_ref, Wl) ;


        left_action (U_l) ;

        // right motor control loop


        Wr = right_velocity () ;
        U_r = Rr (Wr_ref, Wr) ;


        right_action (U_r) ;

       //Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskSERVOS");
    }
}

Void TaskORD_release (void) {
    Semaphore_post(TaskORD_sem);
    }


Void TaskORD (UArg a0, UArg a1)
{

char Order_frame[200]=Received_Order();
ORDER Order;
REFERENCE Local_ref ;
    while (1) {
           Semaphore_pend (TaskORD_sem, BIOS_WAIT_FOREVER) ;

         //  Log_write1(UIABenchmark_start, (xdc_IArg)"TaskORD");

           Read_Order (Order_frame) ;
           Order = Interpret_Order (Order_frame) ;

    switch (Order) {
        case O_STOP: write_STATUS (STOP) ;

        break ;

        case O_MISSION: write_STATUS (MISSION) ;
```

```
        break ;

        case O_TELEMANIPULATION: write_STATUS (TELEMANIPULATION) ;

        break ;

        case O_SETR: Obtain_Reference (Order_frame, &Local_ref) ;

        write_T_REF (&Local_ref) ;
        break ;
        case O_CALIBRATION: write_STATUS (CALIBRATION) ;

                break ;
        case O_INITIALITATION: write_STATUS (INITIALITATION) ;

                break ;
        default: ;
        }
          //  Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskORD");
    }
}

Void TaskODO_release (void) {
    Semaphore_post(TaskODO_sem);
    }

Void TaskODO (UArg a0, UArg a1)
{

    while(1){
    Semaphore_pend (TaskODO_sem, BIOS_WAIT_FOREVER) ;

   // Log_write1(UIABenchmark_start, (xdc_IArg)"TaskODO");

    read_ODO_Trayect(&odoT);
    Odom_Trayect(&odoT, Robot_Reference_N.V,Robot_Reference_N.W,df ,dl ,dr,
Odometry_Period);
    write_ODO_Trayect(&odoT);
    Send_ODO_Trayect(&odoT);


   // Log_write1(UIABenchmark_stop, (xdc_IArg)"TaskODO");
    }
}

void Action (float V,float W){  // Set V & W to servo motors
    Robot_Reference_N.V = V;
    Robot_Reference_N.W = W;

    }



void MISION_4 (void){ //Algorythm of Navigation
 if(bumper==0){

     Position(0,0);

     caso2=caso2+1;

     if(caso2>60){
         Position(0,-0.1);
```

```
    }
 }else{
    if ( df<45){

                  Position(0,0);
                    EstadoNav= FRENTE;

                    if(df<25){
                        caso2=caso2+1; // counter
                    } else { caso=caso+1; // counter
                    }

                    }
    else if ( df<60) {


                    Position(0,0.1);
                    EstadoNav= SIGUE;

                    caso=0;
                    caso2=0;

                    if( (dl < 30 && dr>30) ){Position(-2,0.1);}
                    if( (dr < 30 && dl>30) ){Position(2,0.1);}
    }

    else{
                    Position(0,0.15);
                    EstadoNav= SIGUE;

                    caso=0;
                    caso2=0;

                    if( (dl < 30 && dr>30)  ){Position(-2,0.15);}
                    if( (dr < 30 && dl>30)  ){Position(2,0.15);}


        }




                if ( caso>60){ //If you are free, go to the right to change
direction

                    EstadoNav= RODEADO;




                if (dl > dr){ //Go left
                    Position(89,0.1);
                    EstadoNav= IZQUIERDA;

                    }
                else if (dr > dl){ //Go right
                    Position(-89,0.1);
                    EstadoNav= DERECHA;

                    }
                }
                if(caso2>60){
```

```
                    Position(0,-0.1); //Reverse
                }
    }

}


float PI (float xref, float x, float kp, float ki, float* xenc, float Umax,
float Umin){ //Proporcional-Integral Controller + AntiWind-up

    float err,pi;
    err=xref-x;
    pi=(err*kp)+ *xenc;
    if(pi>Umax){
        pi=Umax;

    }else if (pi<Umin){
        pi=Umin;
    }else {
       *xenc=*xenc+ki*err*Tsw;
    }


    return pi;
}



float Comp(int i,float dF_ant){ //Calculate Lineal velocity every 5s, only if
distance is correct
    float  dF_act, X,V;


      if(i>50){

      dF_act=(float)Read_Channel (1);
      X =  (dF_act-dF_ant)/100; //meters
      }
     if(X<=0){
         V=0;
      }else {

          V = X/5;
      }



      return V;
}
void CalibGyro(float tiempo){ //Calibrate offset value giroscope
    float t=tiempo*20;
    if(cuenta_gyro>t){ //  5s

        vZ_b=Offset_GYRO (Z);
        cuenta_gyro=0;
        }
    else{
        cuenta_gyro++;

                }

}
```

```
void Lineal_velocity (void){ //Algorythm of lineal velocity compesation



              if(Cuenta_US<10){
                        dF_ant = read_front_distance () ; //1° read value
                     }
                     if(Cuenta_US>50){ //100

                     if((25<df<90) && (wZ<=1)){
                                                  //2° read other value 5s after
                        V_comp=Comp(Cuenta_US,dF_ant);  //Get V

                        hayComp=1;
                     }else {
                         V_comp=0;
                         Cuenta_US=0;
                         hayComp=0;}
                     Cuenta_US=0;

                    }
                              else{
                                  Cuenta_US++;
                              }

        V_pi=PI(Robot_Reference_S.V ,V_comp ,kp_V ,ki_V ,Iv,0.3, -0.3); //PI
to feedback
        Action(V_pi,0); //Set V compensate

}

void velocity (float Wref, float Vlineal){  //Angular velocity compensation
    float Wpi=0;

     Wpi=PI(Robot_Reference_S.W,wZ,kp,ki,Iw,0.5,-0.5);
     Action (Vlineal,Wpi);

}

void Position(float titaref, float Vlineal){ //Position Compensation
    float tita_ref, Wpos,cont_pos=0;

    tita_ref=titaref*0.0174; //° a rad


    cont_pos=cont_pos+1;
    if(cont_pos>=60){
        Wpos=0;
        cont_pos=0;
    }else{
    Wpos=PI(tita_ref,tita_abs,kpx,kix,Ix,1.57,-1.57); }

    Action (Vlineal,Wpos);


}



float P_ciego(float us, int t, float dmax, float dmin){ //Blind Spot
    float  d_act=0,d_ant=0,dt;
    int temp=0;
    temp=t*20;
```

```
        if(cont>=temp){


            cont=0;
        } else {
            d_act=us;
        cont++;
}


dt=(d_act-d_ant)/temp;

d_ant=d_act;

return dt;
}
void Inicializa(void){ //Set values to 0

    caso=0;
    caso2=0;
    caso3=0;
    cuenta_gyro=0;
    Cuenta_US=0;
    dl=0;
    df=0;
    dr=0;
    Left_distance=0;
    Front_distance=0;
    Right_distance =0;



    Ti=0;
    siguiente=0;
    periodo=0;
    j=0;


    vX_b=0;
    vY_b=0;
    vZ_b=0;
    wZ=0;
    Wabs=0;


    Ix=0;
    tita_abs=0;
     Wpos=0;
    titaref=0;
    Wl=0;
    Wr=0;
    Wl_ref=0;
    Wr_ref=0;
    U_l=0;
    U_r=0;

    Vl_T=0;
    Vr_T=0;

     Wciego=0;
     d_ciegoL=0;
     d_ciegoR=0;
```

```
     cont=0;
     alpha=0;



    V_comp=0;
    dF_ant=0;
    V_pi=0;
    hayComp=0;

}
```

### A.1.2 *Driver* US

```
/**********************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : us.c
*/
/* version  : 2
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernandez
*/
/* description : This module is a device driver for the ultrasound sensors
*/
/*
*/
/**********************************************************************
*/

/**********************************************************************
*/
/*                          Used modules
*/
/**********************************************************************
*/

#include <stdlib.h>
//#include <stdio.h>
//#include <math.h>
//#include <string.h>

#include "F2837xS_Device.h"
#include "F2837xS_Adc_defines.h"
#include "us.h"

extern void AdcSetMode(uint16 adc, uint16 resolution, uint16 signalmode) ;



/**********************************************************************
*/
/*                          Local variables
*/
```

```
/*****************************************************************************
*/

static float R_Calib_m = 1.0 ; static float R_Calib_b = 0.0 ;
static float F_Calib_m = 1.0 ; static float F_Calib_b = 0.0 ;
static float L_Calib_m = 1.0 ; static float L_Calib_b = 0.0 ;


/*****************************************************************************
*/
/*                          Local functions
*/
/*****************************************************************************
*/

extern void DelayUs( volatile uint16 Usec ) ;

/*****************************************************************************
*/
/*                          Exported functions
*/
/*****************************************************************************
*/

void Init_US (void) {

    EALLOW;

    CpuSysRegs.PCLKCR13.bit.ADC_A = 1;

    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;                  // ADCCLK = SYSCLK/4 =
200MHz/4 = 50MHZ
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;               // Set pulse positions
to late

    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;                  // Power up the ADC A

    DelayUs(1000);                                      // delay for 1ms to ADC
to power up

    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2 ;                 // SOC0 will convert
ADCINA 2
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 19 ;                // sample window =
(19+1)*5ns = 100ns

    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3 ;                 // SOC1 will convert
ADCINA 3
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 19 ;                // sample window =
(19+1)*5ns = 100ns

    AdcaRegs.ADCSOC2CTL.bit.CHSEL = 4 ;                 // SOC2 will convert
ADCINA 4
    AdcaRegs.ADCSOC2CTL.bit.ACQPS = 19 ;                // sample window =
(19+1)*5ns = 100ns

      GpioCtrlRegs.GPCDIR.bit.GPIO69 = 1 ;                // GPIO69 as output,
start of measure pulse
    GpioDataRegs.GPCDAT.bit.GPIO69 = 0 ;
    DelayUs(1000);                                      // delay for 1ms

    EDIS;
```

```
    GpioDataRegs.GPCDAT.bit.GPIO69 = 1 ;                // Start measuring
    DelayUs(25);                                        // Pulse of 25 us
    GpioDataRegs.GPCDAT.bit.GPIO69 = 0 ;
    DelayUs(10);

    EALLOW;
      GpioCtrlRegs.GPCDIR.bit.GPIO69 = 0 ;                // GPIO69 as output,
start of measure pulse
    EDIS;


}

void Calibrate_US (US_SENSOR USS, float slope, float bias) {
       switch (USS) {
       case LEFT:
             L_Calib_m = slope ; L_Calib_b = bias ; break ;
       case FRONT:
             F_Calib_m = slope ; F_Calib_b = bias ; break ;
       case RIGHT:
             R_Calib_m = slope ; R_Calib_b = bias ; break ;
       }
}

unsigned int Read_Channel (char CH) {

    EALLOW;
    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = CH ;            // end of SOC_CH will
set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1 ;               // enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1 ;             // make sure INT1 flag
is cleared
    EDIS;

    AdcaRegs.ADCSOCFRC1.all = (0x0001) << CH ;          // Start of conversion

    while(AdcaRegs.ADCINTFLG.bit.ADCINT1 == 0) ;
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1 ;

    if (CH == 0) return AdcaResultRegs.ADCRESULT0 ;
    if (CH == 1) return AdcaResultRegs.ADCRESULT1 ;
    if (CH == 2) return AdcaResultRegs.ADCRESULT2 ;

    return 0 ;
}


float Read_Distance (US_SENSOR USS) {

       float distance = 0.0 ;
       unsigned char i,j,k;
       float v[12];
       float result;
       float vR[12];
       float resultR;
    float vF[12];
    float resultF;

       switch (USS) {
         case LEFT:

                for (i=0; i<12; i++) {
                            v[i]=Read_Channel(0);
```

```
                        if(v[i-1]>v[i]+5 && v[i]>350){
                       v[i-1]=v[i];
                                                    }
                        result += v[i];
                        }
                    distance=result/2.3;
                    distance = L_Calib_m * distance + L_Calib_b ;
              break ;
        case FRONT:

              for (k=0; k<12; k++) {
                        vF[k]=Read_Channel(1);
                        if(vF[k-1]>vF[k]+5 && vF[k]>350){
                        vF[k-1]=vF[k];
                                                   }
                        resultF += vF[k];
                        }
                     distance=resultF/2.3;
                     distance = F_Calib_m * distance + F_Calib_b ;
            break ;
        case RIGHT:

             for (j=0; j<12; j++) {
                                  vR[j]=Read_Channel(2);
                                  if(vR[j-1]>vR[j]+5 && vR[j]>350){
                                      vR[j-1]=vR[j];
                                  }
                                  resultR += vR[j];
                                  }
                               distance=resultR/2.3;
                               distance = R_Calib_m * distance + R_Calib_b ;
              break ;
      }
      return distance ;
}

float median (float v[], unsigned char n) {
      float aux [20] ;
      unsigned char i, j ;
      float temp ;

      memcpy (aux, v, 4*n) ;
      for (i=1; i<n; i++) {
            for (j=0; j<n-i; j++) {
                  if (aux[j]>aux[j+1]) {
                        temp = aux[j+1] ;
                        aux[j+1] = aux[j] ;
                        aux[j] = temp ;
                  }
            }
      }
      if (n%2) return aux[n/2] ;
      else return (aux[n/2]+aux[n/2+1])/2.0 ;
}
  // Returns the median of the n elements stored in array v

float average (float v[], unsigned char n) {
      float result ;
      unsigned char i ;

      for (i=0; i<n; i++) {
            result += v[i] ;
```

```
        }
        return result/n ;
}
```

### A.1.3 *Driver Gyro*

```
/*****************************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : GYRO.c
*/
/* version  : 2
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernandez
*/
/*       10/17/2011(Oliver J): created
*/
/*    20/01/2017(Jose Luis Villarroel): modified for F28377s
*/
/*    23/08/2024(Javier Fernández):Adds new function to set offset
*/
/* description : Device driver for Pmod GYRO
*/
/*
*/
/*
*/
/* Pins / signals : GPIO41 - INT (Data Ready) - disabled
*/
/*                  GPIO58 - SPISIMOA
*/
/*                  GPIO59 - SPISOMIA
*/
/*                  GPIO60 - SPICLKA
*/
/*                  GPIO61 - CS
*/
/*
*/
/*****************************************************************************
*/


/* ------------------------------------------------------------ */
/*                        Include File Definitions
        */
/* ------------------------------------------------------------ */

#include "F2837xS_device.h"
#include "GYRO.h"
#include "spi.h"

/*****************************************************************************
*/
/*                        Defines
*/
/*****************************************************************************
```

```
*/

//  Register Addresses

#define WHO_AM_I          0x0F
#define CTRL_REG1         0x20
#define CTRL_REG2         0x21
#define CTRL_REG3         0x22
#define CTRL_REG4         0x23
#define CTRL_REG5         0x24
#define REFERENCE         0x25
#define OUT_TEMP          0x26
#define STATUS_REG        0x27
#define OUT_X_L                 0x28
#define OUT_X_H                 0x29
#define OUT_Y_L                 0x2A
#define OUT_Y_H                 0x2B
#define OUT_Z_L                 0x2C
#define OUT_Z_H                 0x2D
#define FIFO_CTRL_REG     0x2E
#define FIFO_SRC_REG      0x2F
#define INT1_CFG          0x30
#define INT1_SRC          0x31
#define INT1_TSH_XH       0x32
#define INT1_TSH_XL       0x33
#define INT1_TSH_YH       0x34
#define INT1_TSH_YL       0x35
#define INT1_TSH_ZH       0x36
#define INT1_TSH_ZL       0x37
#define INT1_DURATION     0x38

//     CTRL_REG1

#define REG1_DR1          0x80
#define REG1_DR0          0x40
#define REG1_BW1          0x20
#define REG1_BW0          0x10
#define REG1_PD                 0x08
#define REG1_ZEN          0x04
#define REG1_YEN          0x02
#define REG1_XEN          0x01

//     CTRL_REG3

#define REG3_I1_INT1      0x80
#define REG3_I1_BOOT      0x40
#define REG3_H_LACTIVE    0x20
#define REG3_PP_OD        0x10
#define REG3_I2_DRDY      0x08
#define REG3_I2_WTM       0x04
#define REG3_I2_ORUN      0x02
#define REG3_I2_EMPTY     0x01

//     CTRL_REG4

#define REG4_BDU        0x80

//     CTRL_REG5

#define REG5_FIFO_EN    0x40

//     INT1_CFG
```

```
#define INT1_ANDOR         0x80
#define INT1_LIR           0x40
#define INT1_ZHIE          0x20
#define INT1_ZLIE          0x10
#define INT1_YHIE          0x08
#define INT1_YLIE          0x04
#define INT1_XHIE          0x02
#define INT1_XLIE          0x01


#define NULL        0

#define WRITE              0x00
#define MULTI_WRITE 0x40
#define READ        0x80
#define MULTI_READ  0xC0

/*****************************************************************************
*/
/*                        Local variables
*/
/*****************************************************************************
*/

static int Z_dc_offset = 0 ;

int buff[100] ;
int i = 0 ;
int aux = 0 ;

char temp = 0 ;

static float X_Calib_b = 0.0 ;
static float Y_Calib_b = 0.0 ;
static float Z_Calib_b = 0.0 ;


/*****************************************************************************
*/
/*                        Prototypes of local functions
*/
/*****************************************************************************
*/

int writeReg (unsigned char reg, unsigned char value) ;
void readReg (unsigned char reg, unsigned char *recv, int count) ;

extern void DelayUs( volatile uint16 Usec ) ;


/*****************************************************************************
*/
/*                        Exported functions
*/
/*****************************************************************************
*/

void Init_GYRO (void)
{
       int n ;

       Init_SPIA () ;
       Set_CS (HIGH) ;
```

```
    // else interrupts disabled
    writeReg(CTRL_REG4, REG4_BDU) ;             // Block Data update, 250 dps,
SPI interface
       writeReg(CTRL_REG5, REG5_FIFO_EN) ;          // FIFO enabled
       writeReg(FIFO_CTRL_REG, 0x00) ;              // Bypass mode
       writeReg(CTRL_REG1, REG1_PD | REG1_ZEN) ;    // Normal mode, Z axis
enabled, 100Hz


       for (n=0; n<100; n++) {

       buff[i] = getZ () ;
          aux += buff[i] ;
       i = (i+1)%100 ;
       DelayUs (20000) ;
       }
       Z_dc_offset = aux/100 ;

       return ;
}


int getX (void)
{
       unsigned char temp[2] = {0,0} ;
       unsigned int xAxis = 0 ;

       readReg(OUT_X_L, temp, 2) ;

       xAxis = temp[0] ;
       xAxis |= (temp[1] << 8) ;
       xAxis= xAxis + X_Calib_b;
       return xAxis;
}

int getY (void)
{
       unsigned char temp[2] = {0,0} ;
       unsigned int yAxis = 0 ;

       readReg(OUT_Y_L, temp, 2) ;

       yAxis = temp[0] ;
       yAxis |= (temp[1] << 8) ;
       yAxis= yAxis + Y_Calib_b;
       return yAxis;
}

int getZ (void)
{
       unsigned char temp[2] = {0,0} ;
       unsigned int zAxis = 0 ;

       readReg(OUT_Z_L, temp, 2) ;

       zAxis = temp[0] ;
       zAxis |= (temp[1] << 8) ;
       zAxis= zAxis + Z_Calib_b;
       return ((int)zAxis - Z_dc_offset) ;
}

//Changes made:
```

```
//Changed temp to int8_t from uint8_t
//Casted &temp to (uint8_t *) in readReg call
//Added temp = 44 - temp; to get a more accurate temperature

signed char getTemp (void)
{
	signed char temp = 0;

	readReg(OUT_TEMP, (unsigned char *) &temp, 1);

	temp = 44 - temp;
	return temp;
}


int writeReg (unsigned char reg, unsigned char value)
{
	unsigned char temp;

	Set_CS (LOW);
	DelayUs (20) ;
	SPI_transfer(reg);
	SPI_transfer(value);
	Set_CS (HIGH);

	Set_CS (LOW);
	SPI_transfer(READ | reg);
	temp = SPI_transfer(0x00);
	Set_CS(HIGH);

	if(temp != value)
		return 0;

	return 1;
}

void readReg (unsigned char reg, unsigned char *recv, int count)
{
	if(count == 1)
	{
		// Retrieve current state of register
		Set_CS(LOW);
		SPI_transfer(READ | reg);
		*recv = SPI_transfer(0x00);
		Set_CS(HIGH);
	}
	else
	{
		int i = 0;
		Set_CS(LOW);
		SPI_transfer(MULTI_READ | reg);
		for(i=0;i < count;i++)
		{
			*recv = SPI_transfer(0x00);
			recv++;
		}
		Set_CS(HIGH);
	}
}
float Offset_GYRO (GYRO_SENSOR AXIS){ //Calculate the average of 50 values for
each axis
    float Calib_b [50];
    float resultX=0; float resultY=0; float resultZ=0;
```

```
    float bias_Gyro;
    unsigned char i;
    switch (AXIS){
    case X:
        for (i=0; i<50; i++) {
           Calib_b [i]=getX();
           resultX += Calib_b [i];
        }
        bias_Gyro=resultX/50;
    break;
    case Y:
        for (i=0; i<50; i++) {
            Calib_b [i]=getY();
            resultY += Calib_b [i];
               }
        bias_Gyro=resultY/50;
    break;
    case Z:
        for (i=0; i<50; i++) {
             Calib_b [i]=getZ();
             resultZ += Calib_b [i];
               }
        bias_Gyro=resultZ/50;
    break;

    }
    return bias_Gyro;
}
float get_GYRO(GYRO_SENSOR AXIS, float bias){ //Transfomr the data to angular
velocity

    float g_deg_s,W,V,Wabs; //GYRO

    switch (AXIS){
    case X:
        V=getX();
        g_deg_s=((V-bias)*(250.0/32767.0));
        W=g_deg_s*0.0174;

        break;
    case Y:
        V=getY();
        g_deg_s=((V-bias)*(250.0/32767.0));
        W=g_deg_s*0.0174;

        break;
    case Z:
        V=getZ();
        g_deg_s=((V-bias)*(250.0/32767.0));
        W=g_deg_s*0.0174;
        if (W<=0.01){
            Wabs=0.0;
        }else {Wabs=W;}
        break;

    }

      return Wabs;
}
```

### A.1.4 *Driver Odometry*

```
/*****************************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : Odometry.c
*/
/* version  : 2
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernández
*/
/* description : Odometry update
*/
/*
*/
/*****************************************************************************
*/

#include <math.h>
#include "ROBOT_types.h"
float titaMAX=0;
/*****************************************************************************
*/
/*                          Exported functions
*/
/*****************************************************************************
*/


void Odom_Trayect (ODOMETRY_Trayect *Robot_Odometry, float Vactual, float
Wactual, float A, float B, float C, float Tcontrol){


    float dtita, ds, dx, dy ;


      dtita = (Wactual*Tcontrol)/1000 ;
      if (Wactual*Wactual < 0.0001){
          ds = Vactual*Tcontrol ;
          Robot_Odometry->tita = Robot_Odometry->tita;
          dtita=0;
      }
      else {
       ds = Vactual/Wactual * dtita ;
       Robot_Odometry->tita = Robot_Odometry->tita + dtita ;
      }

       dx = ds*cos(Robot_Odometry->tita*0.78+(dtita/2)) ;
       dy = ds*sin(Robot_Odometry->tita*0.78+(dtita/2)) ;


       Robot_Odometry->X = Robot_Odometry->X + dx ;
       Robot_Odometry->Y = Robot_Odometry->Y + dy ;

     Robot_Odometry->df = A;
     Robot_Odometry->dl = B ;
     Robot_Odometry->dr = C ;
```

```
    if (fabs(Wactual)>titaMAX){
        titaMAX=fabs(Wactual);
    }

}
```

### A.1.5 *Driver Protocol*

```
/*****************************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : protocol.c
*/
/* version  : 2
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernandez
*/
/* description : This module implements the communication protocol
*/
/*               with the host.
*/
/*             (23/08/24) Javier Fernandez: adds new variables and orders
*/
/*
*/
/*****************************************************************************
*/

/*****************************************************************************
*/
/*                          Used modules
*/
/*****************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/knl/Swi.h>


#include "ROBOT_types.h"
#include "protocol.h"
#include "XBee.h"

/*****************************************************************************
*/
/*                          Local variables
*/
/*****************************************************************************
*/

const char Begin_Frame = '#' ;
const char End_Frame = '\r' ;

static char Aux_Frame [80] ;
```

```
/*****************************************************************************
*/
/*                          Local functions
*/
/*****************************************************************************
*/

static int Move_to_comma (int i, char *data) ;


/*****************************************************************************
*/
/*                          Exported functions
*/
/*****************************************************************************
*/

void Init_Protocol (Swi_Handle action) {

       Init_XBee () ;
       Receive_Frames (Begin_Frame, End_Frame) ;
       Install_Received_Frame_Action (action) ;

}

char Received_Order (void) {
       return Received_Frame () ;
}

void Read_Order (char *data) {

       Read_Frame (data) ;
}


void Send_ODO_Trayect (ODOMETRY_Trayect *dataT) {

    int j = 0, n = 0 ;

    Aux_Frame [0] = Begin_Frame ;
    Aux_Frame [1] = 'O' ;
    Aux_Frame [2] = 'D' ;
    Aux_Frame [3] = 'O' ;
    Aux_Frame [4] = ' ' ;

    j = 5 ;

    n=System_sprintf (Aux_Frame+j, "%d", (int)  (dataT->X));

    j += n ;
    Aux_Frame [j] = ',' ; j++ ;

    n=System_sprintf (Aux_Frame+j, "%d", (int) (dataT->Y));
    j += n ;
    Aux_Frame [j] = ',' ; j++ ;

    n=System_sprintf (Aux_Frame+j, "%d", (int) (dataT->tita));
    j += n ;
    Aux_Frame [j] = ',' ; j++ ;

    n=System_sprintf (Aux_Frame+j, "%d", (int) (dataT->df*10.0));
    j += n ;
```

```
    Aux_Frame [j] = ',' ; j++ ;

    n=System_sprintf (Aux_Frame+j, "%d", (int) (dataT->dr*10.0));
    j += n ;
    Aux_Frame [j] = ',' ; j++ ;

    n=System_sprintf (Aux_Frame+j, "%d", (int) (dataT->dl*10.0));
    j += n ;
    Aux_Frame [j] = '\n' ;
    Aux_Frame [j+1] = End_Frame ;

    Send_Frame (Aux_Frame, j+2) ;

}



ORDER Interpret_Order (char *data) {

      if (data[0]==Begin_Frame && data[1]=='S' && data[2]=='T' && data[3]=='O'
&& data[4]=='P')
            return O_STOP ;
      else if (data[0]==Begin_Frame && data[1]=='M' && data[2]=='I' &&
data[3]=='S' && data[4]=='N')
            return O_MISSION ;
      else if (data[0]==Begin_Frame && data[1]=='T' && data[2]=='E' &&
data[3]=='L' && data[4]=='E')
            return O_TELEMANIPULATION ;
      else if (data[0]==Begin_Frame && data[1]=='S' && data[2]=='E' &&
data[3]=='T' && data[4]=='R')
            return O_SETR ;
      else if (data[0]==Begin_Frame && data[1]=='C' && data[2]=='A' &&
data[3]=='L' && data[4]=='B')
             return O_CALIBRATION ;
    else if (data[0]==Begin_Frame && data[1]=='I' && data[2]=='N' &&
data[3]=='I' && data[4]=='T')
           return O_INITIALITATION ;
      else return O_ERROR ;
}

void Obtain_Reference (char *data, REFERENCE *ref) {

      int j ;

      ref->V = atof (data+5) ;
      j = Move_to_comma (5, data) ;
      ref->W = atof (data+j+1) ;
}

static int Move_to_comma (int i, char *data) {

      int j ;

      for (j=i+1; data[j] != ','; j++) ;
      return j ;
}
```

### A.1.6 Servidores

```
/*****************************************************************************
*/
/*
*/
/* project  : Trabajo Final de Grado: Robot para prácticas de STR
*/
/* filename : servidores.c
*/
/* version  : 2
*/
/* date     : 23/08/2024
*/
/* author   : Jose Luis Villarroel & Javier Fernandez
*/
/* description : This module implements servers with priority inheritance
*/
/*             (23/08/24) Javier Fernandez: Adds new server for giroscope
*/
/*             (23/08/24) Javier Fernandez: Modifies odo server
*/
/*
*/
/*****************************************************************************
*/


/*****************************************************************************
*/
/*                          Used modules
*/
/*****************************************************************************
*/

#include <ti/sysbios/gates/GateMutexPri.h>

#include "servidores.h"
#include "ROBOT_types.h"


/*****************************************************************************
*/
/*                          Local variables
*/
/*****************************************************************************
*/

static STATUS ROBOT_Status = STOP ;
static REFERENCE Telemanipulation_Reference = {0.0, 0.0} ;
static GateMutexPri_Handle Status_server ;

static float ROBOT_dl = 0.0, ROBOT_df = 0.0, ROBOT_dr = 0.0 ;
static GateMutexPri_Handle US_server ;

static float ROBOT_wz = 0.0;
static GateMutexPri_Handle GYRO_server ;

static REFERENCE ROBOT_Reference = {0.0, 0.0} ;
static GateMutexPri_Handle Reference_server ;

static ODOMETRY ROBOT_Odometry = {0.0, 0.0, 0.0} ;
static GateMutexPri_Handle Odometry_server ;
```

```
static ODOMETRY_Trayect ROBOT_Odometry_T = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0} ;
static GateMutexPri_Handle Odometry_Trayect_server ;

static GateMutexPri_Params prms ;



/*****************************************************************************
*/
/*                          Exported functions
*/
/*****************************************************************************
*/

void Server_Create (void) {

       GateMutexPri_Params_init (&prms) ;
       Status_server = GateMutexPri_create (&prms, NULL) ;
       GateMutexPri_Params_init (&prms) ;
       US_server = GateMutexPri_create (&prms, NULL) ;
       GateMutexPri_Params_init (&prms) ;
       GYRO_server = GateMutexPri_create (&prms, NULL) ;
       GateMutexPri_Params_init (&prms) ;
       Reference_server = GateMutexPri_create (&prms, NULL) ;
       GateMutexPri_Params_init (&prms) ;
       Odometry_server = GateMutexPri_create (&prms, NULL) ;
       GateMutexPri_Params_init (&prms) ;
       Odometry_Trayect_server = GateMutexPri_create (&prms, NULL) ;
}

// Status_server

void write_STATUS (STATUS st) {
       IArg key ;

       key = GateMutexPri_enter (Status_server) ;
       ROBOT_Status = st ;
       GateMutexPri_leave (Status_server, key) ;
}

STATUS read_STATUS (void)  {
       IArg key ;
       STATUS aux_st ;

       key = GateMutexPri_enter (Status_server) ;
       aux_st = ROBOT_Status ; ;
       GateMutexPri_leave (Status_server, key) ;
       return aux_st ;
}

void write_T_REF (REFERENCE *ref) {
       IArg key ;

       key = GateMutexPri_enter (Status_server) ;
       Telemanipulation_Reference.V = ref->V ;
       Telemanipulation_Reference.W = ref->W ;
       GateMutexPri_leave (Status_server, key) ;
}

void read_T_REF (REFERENCE *ref) {
       IArg key ;

       key = GateMutexPri_enter (Status_server) ;
```

```
	ref->V = Telemanipulation_Reference.V ;
	ref->W = Telemanipulation_Reference.W ;
	GateMutexPri_leave (Status_server, key) ;
}

// Ultrasonic server

void write_left_distance (float d) {
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	ROBOT_dl = d ;
	GateMutexPri_leave (US_server, key) ;
}

void write_front_distance (float d) {
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	ROBOT_df = d ;
	GateMutexPri_leave (US_server, key) ;
}

void write_right_distance (float d) {
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	ROBOT_dr = d ;
	GateMutexPri_leave (US_server, key) ;
}

float read_left_distance (void) {
	float d ;
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	d = ROBOT_dl ;
	GateMutexPri_leave (US_server, key) ;
	return d ;
}

float read_front_distance (void) {
	float d ;
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	d = ROBOT_df ;
	GateMutexPri_leave (US_server, key) ;
	return d ;
}

float read_right_distance (void) {
	float d ;
	IArg key ;

	key = GateMutexPri_enter (US_server) ;
	d = ROBOT_dr ;
	GateMutexPri_leave (US_server, key) ;
	return d ;
}


//Gyroscope_server
```

```
void write_z_axis (float d) {
    IArg key ;

    key = GateMutexPri_enter (GYRO_server) ;
    ROBOT_wz = d ;
    GateMutexPri_leave (GYRO_server, key) ;
}

float read_z_axis (void) {
    float d ;
    IArg key ;

    key = GateMutexPri_enter (GYRO_server) ;
    d = ROBOT_wz ;
    GateMutexPri_leave (GYRO_server, key) ;
    return d ;
}

// Reference_server

void write_REF (REFERENCE *ref) {
       IArg key ;

       key = GateMutexPri_enter (Reference_server) ;
       ROBOT_Reference.V = ref->V ;
       ROBOT_Reference.W = ref->W ;
       GateMutexPri_leave (Reference_server, key) ;
}

void read_REF (REFERENCE *ref) {
       IArg key ;

       key = GateMutexPri_enter (Reference_server) ;
       ref->V = ROBOT_Reference.V ;
       ref->W = ROBOT_Reference.W ;
       GateMutexPri_leave (Reference_server, key) ;
}

// ODOMETRY server

void write_ODO_Trayect (ODOMETRY_Trayect *dataT) {
    IArg key ;

    key = GateMutexPri_enter (Odometry_server) ;
    ROBOT_Odometry_T.X = dataT->X ;
    ROBOT_Odometry_T.Y = dataT->Y ;
    ROBOT_Odometry_T.tita = dataT->tita ;
    ROBOT_Odometry_T.df = dataT->df ;
    ROBOT_Odometry_T.dr = dataT->dr ;
    ROBOT_Odometry_T.dl = dataT->dl ;


    GateMutexPri_leave (Odometry_server, key) ;
}

void read_ODO_Trayect (ODOMETRY_Trayect *dataT) {
    IArg key ;

    key = GateMutexPri_enter (Odometry_server) ;
    dataT->X = ROBOT_Odometry_T.X ;
    dataT->Y = ROBOT_Odometry_T.Y ;
    dataT->tita = ROBOT_Odometry_T.tita ;
```

```
    dataT->df = ROBOT_Odometry_T.df ;
    dataT->dr = ROBOT_Odometry_T.dr ;
    dataT->dl = ROBOT_Odometry_T.dl ;

    GateMutexPri_leave (Odometry_server, key) ;
}
```

## Anexo 2: Código Consola Telemanipulación Python

```
################################################################################
######################
#
#   project  : Trabajo Final de Grado: Robot para prácticas de STR
#   filename : main.py
#   version  : 1.6180339887
#   date     : 23/08/2024
#   author   : Javier Fernandez
#   description : This program shows the console and the odometry point in a
map
# Press Mayús+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows,
actions, and settings.
################################################################################
######################
from __future__ import annotations
import random
from typing import Type
import threading
import serial
import time
import numpy as np
from io import open
import tkinter as tk
from tkinter import ttk
from serial import SerialException
from msvcrt import getch
import keyboard
from itertools import islice
import struct
import sys
import csv
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import math

#Port COM info
cfg = open("C:/D\Documentos/4 año/TFG/logs trayectoria/configuracion.txt",
"r")

info= cfg.read()     #Leer cfg para saber puerto
puerto=info[0:4]  # tipico en windows X= un entero positivo
archivo=info[29:45]
cfg.close()
plt.ion()
bool=0
baudrate = 9600 #(o el baudrate adecuado/usado en putty)

#Open the console window
try:
    ser = serial.Serial(port=puerto, baudrate=baudrate,timeout=0.25)
except SerialException:
    pront="Error puerto"
    print(pront)
    Err = tk.Tk()
    Err.title("ERROR")
    Err.config(width=150, height=100)
    etiqueta_Err = ttk.Label(text=pront, background="red",font="10")
    etiqueta_Err.place(x=20, y=20)
    boton_convertirErr = ttk.Button(text="Quit", command=quit)
    boton_convertirErr.place(x=20, y=60, width=60)
    Err.mainloop()
```

```
print("Iniciada Tele")
ser.write(b'#STOP\r')  # el \r corresponde al CR carriage return, \n a LF line
feed
time.sleep(0.5)
print("Robot No Conectado...")
time.sleep(0.5)
print("Buscando Robot")

V=000.00 #resetear valores de TELE
W=000.00
i=000.00
j=000.00
datos=0
fram=[]
pront="Inicio"
pront1="V:   , W:"

#Reads and interprate the values sended by the robot
def leerOdo():
    trama = open("C:/D/Documentos/4 año/TFG/logs trayectoria/trayectoria.txt",
"w")
    trama.write("X,Y,theta,dF,dR,dL" + "\n")
    trama.close()
    global pront
    pront="Sending Data"
    print(pront)
    trama = open("C:/D/Documentos/4 año/TFG/logs trayectoria/trayectoria.txt",
"a")
    frame=''
    frame2=''

    while True:
        ventana.update()
        #  time.sleep(1) # tiempo en segundos.
        size=ser.inWaiting()
        complete=False

        if True:
            frame = ser.read(size) # si sabes la cantidad de bytes recibidos,
puedes especificarlo dentro de los parentesis.
            #print(frame)
            frame2+=frame.decode('utf-8')


            if "#ODO" in frame2:
                pos=frame2.index("#ODO")
            else:
                pos=-1
            if pos>-1:
                frame2=frame2[pos+5:]

            if '\r' in frame2:
                    pos = frame2.index("\r")
            else:
                pos = -1
            if pos > -1:
                frame2 = frame2[:pos - 1]

                frame3 = ''
                if frame2.count(',') == 5:
                    complete = True

                    try:
                        pos = frame2.index(',')


                        trama.write(frame2 + "\n")
```

```
                        #leer txt
                        fram=Data(frame2)
                        Data.plotMemory()
                        print(fram)



                    except Exception as e:
                        print(str (e))



            elif frame2 == None:
                ventana.quit()
        if keyboard.is_pressed("esc"):
            break
    trama.close()
def getColumns(filename,column):

    results = csv.reader(open(filename), delimiter=",")
    return [result[column] for result in results]

#Load a old .txt file of odometry and shows de map with points
def CSV():
    x = []
    y = []
    ang = []


    x=getColumns("C:/D\Documentos/4 año/TFG/logs
trayectoria/trayectoriaa.txt",0)
    y = getColumns("C:/D\Documentos/4 año/TFG/logs
trayectoria/trayectoriaa.txt", 1)
    ang = getColumns("C:/D\Documentos/4 año/TFG/logs
trayectoria/trayectoriaa.txt", 2)



    if True:
        font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
        plt.title("Odometry", fontdict=font1)
        font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
        plt.plot(x, y,marker=">", color="blue",
markersize=10,label="Trayectoria Robot")
        plt.axis('off')
        plt.legend(loc="upper left")
        plt.show()
        plt.pause(0.05)



class Data: #Obtener valores del frame del robot

    def __init__(self,s:str):
        if s.count(",")!=5:
            return None

        res=[]

        for i in range(5):
            pos=s.index(",")
            res.append(int(s[0:pos]))
            s=s[pos+1:]

        self.x=res[0]/10 #cm
```

```
        self.y=res[1]/10
        self.ang=res[2]*180/math.pi #rad?
        self.dF=res[3]/10 #cm
        self.dR=res[4]/10
        self.dL=int(s)/10

        self.time=datetime.datetime.now()
        Data.memory.append(self)

    memory: list[Data]
    memory = []


#Plot de points of the odometry in a map with a matrix transformation
    @staticmethod
    def plotMemory():

        x = []
        y = []
        ang = []
        sensorX = []
        sensorY = []
        sensorXL = []
        sensorYL = []
        xx = []
        yy = []
        #for m in Data.memory:

        if True:
            m=Data.memory[-1]
            x.append(m.x)
            y.append(m.y)
            ang.append(m.ang)

            #sensorX.append(m.x + m.dF * math.cos(m.ang / 180 * math.pi))
            #sensorY.append(m.y + m.dF * math.sin(m.ang / 180 * math.pi))

            sensorXL.append(m.x + m.dL * math.cos((m.ang + 90) / 180 *
math.pi))
            sensorYL.append(m.y + m.dL * math.sin((m.ang + 90) / 180 *
math.pi))
            sensorX.append(m.x + m.dR * math.cos((m.ang - 90) / 180 *
math.pi))
            sensorY.append(m.y + m.dR * math.sin((m.ang - 90) / 180 *
math.pi))

            font1 = {'family': 'serif', 'color': 'blue', 'size': 20}

            plt.title("Odometry",fontdict=font1)
            plt.plot(x, y, marker=(2,2,m.ang-90), color="blue", markersize=10)
            plt.plot(x, y, marker=(3, 1, m.ang - 90), color="blue",
markersize=10,label="Odo")

            #plt.arrow(x,y,1,1,width =0.05)
            # plt.plot(x,sry,"ro")
            # plt.plot(x, sly, "go")
            plt.plot(sensorX, sensorY, "r.", markersize=5,label="Right US")
            plt.plot(sensorXL, sensorYL, 'g.', markersize=5, label="Left US")
            #plt.legend()
            plt.show()
            plt.pause(0.05)



            #print("estoy bien")
    def __repr__(self):
        return f"Time: {self.time}, x: {self.x}, y: {self.y}, angle:
```

```
{self.ang}, dF: {self.dF}, dL: {self.dL}, dR: {self.dR}"


#Commands on the console

def Mision():
    ser.write(b'#MISN\r')  # el \r corresponde al CR carriage return, \n a LF
line feed
    global pront
    pront="Robot en misión"
    print(pront)
    leerOdo()

def Stop():
    global pront
    ser.write(b'#STOP\r')  # el \r corresponde al CR carriage return, \n a LF
line feed
    pront="Robot Parado"
    print(pront)
    command = 'Stop'
def Inicializa():
    global pront
    ser.write(b'#INIT\r')
    time.sleep(1)
    pront="Robot Inicializado"
    print(pront)

def Tele():
    global pront
    pront="Control Manual..."
    print(pront)
    time.sleep(0.5)
    ser.write(b'#TELE\r')
    global i,j
    while ser.isOpen():
        ventana.update()
        Teclas()
        c=round(i,3) #redondeo de valores
        d=round(j, 3)
        V = bytes(str(c), "utf-8")
        W = bytes(str(d), "utf-8")
        time.sleep(0.1)
        ser.write(b'#SETR')
        ser.write(V)
        ser.write(b',')
        ser.write(W)
        ser.write(b'\r')
        pront="V: " + str(c) + ",  W: " + str(d) #mostrar por pantalla los
valores de V y W
        #print("V: " + str(c) + ",  W: " + str(d))
        if keyboard.is_pressed("esc"):
            break
    #print(pront1)

def Teclas():
    global i,j
    if keyboard.is_pressed("down"):

        i = i - 000.01
    elif keyboard.is_pressed("up"):

        i = i + 000.01
    elif keyboard.is_pressed("left"):

        j = j + 000.03 #W avanza algo mas rapido para mayor comodidad en uso
manual
    elif keyboard.is_pressed("right"):
        print(W)
```

```
        j = j - 000.03
    elif keyboard.is_pressed("esc"):
        quit()


def Set():
    global pront
    pront="Set Valor"
    print(pront)
    ser.write(b'#TELE\r')
    time.sleep(0.5)
    V = bytes(caja_setV.get(), 'utf-8')
    W = bytes(caja_setW.get(), 'utf-8')
    # SETR vvv.vv, www.ww\r
    ser.write(b'#SETR')
    ser.write(V)
    ser.write(b',')
    ser.write(W)
    ser.write(b'\r')
    print("Valor de V: "+str(V)+", Valor de W: " + str(W))

def Calibra():
    global pront
    pront="Robot Calibrando"
    print (pront)
    time.sleep(0.5)
    ser.write(b'#CALB\r')
    pront = "No mover"
    print(pront)
    time.sleep(6)
    pront = "Robot Calibrado"
    print(pront)


def reload():
    global pront
    etiqueta_setTxt.configure(text=pront, font="8")
    etiqueta_setTxt.after(40,reload)

def ReseteaTXT():
    trama = open("C:/D\Documentos/4 año/TFG/logs trayectoria/trayectoria.txt",
"w")
    trama.close()


def ChngPort():

    vent = tk.Tk()
    vent.title("Cambiar Puerto COM y archivo ODO")
    vent.config(width=400, height=150)
    etiqueta = ttk.Label(vent,text="Puerto COM:")
    etiqueta.place(x=60, y=20)
    caja = ttk.Entry(vent)
    caja.place(x=160, y=20, width=60)
    etiquetaA = ttk.Label(vent, text="Archivo ODO:")
    etiquetaA.place(x=60, y=60)
    cajaA = ttk.Entry(vent)
    cajaA.place(x=160, y=60, width=60)

    def WriteCFG():
        cfg = open("C:/D\Documentos/4 año/TFG/logs
trayectoria/configuracion.txt", "w")
        port = "COM" + caja.get()+" # COM donde esta el XBee"+"\n"
        cfg.write(port)
        arch=cajaA.get()+".txt"+" # nombre de archivo TXT"+"\n"
        cfg.write(arch)
        cfg.close()
    boton_bool = ttk.Button(vent,text="Aplicar", command=WriteCFG)
```

```
    boton_bool.place(x=60, y=100)
    vent.mainloop()


ventana=tk.Tk()
ventana.title("Consola Tele")
ventana.config(width=400, height=300)



etiqueta_Ord = ttk.Label(text="Elegir Orden:")
etiqueta_Ord.place(x=20, y=20)
boton_convertir = ttk.Button(text="MISION", command=Mision)
boton_convertir.place(x=20, y=60)

boton_convertirS = ttk.Button(text="STOP", command=Stop)
boton_convertirS.place(x=140, y=60)

boton_convertirT = ttk.Button(text="TELE",command=Tele)
boton_convertirT.place(x=20, y=100)
boton_convertirSet = ttk.Button(text="SET",command=Set)
boton_convertirSet.place(x=20, y=140)

etiqueta_setV = ttk.Label(text="Set V:")
etiqueta_setV.place(x=140, y=100)
caja_setV = ttk.Entry()
caja_setV.place(x=180, y=100, width=60)



etiqueta_setW = ttk.Label(text="Set W:")
etiqueta_setW.place(x=140, y=140)
caja_setW = ttk.Entry()
caja_setW.place(x=180, y=140, width=60)



boton_convertirC = ttk.Button(text="CALIBRACIÓN",command=Calibra)
boton_convertirC.place(x=260, y=60, width=120)

boton_convertirCSV = ttk.Button(text="LOAD FILE",command=CSV)
boton_convertirCSV.place(x=260, y=100,width=120)

etiqueta_setTxt = ttk.Label(text=str(V))
etiqueta_setTxt.place(x=200, y=220)
etiqueta_setTxt_ = ttk.Label(text="Consola Tele", font="10")
etiqueta_setTxt_.place(x=150, y=15)

barra_menu=tk.Menu(ventana)   #Menu de opciones
ventana.config(menu=barra_menu)
menu=tk.Menu(barra_menu)
submenu=tk.Menu(menu)
barra_menu.add_cascade(label="Opciones",menu=menu)


menu.add_cascade(label="Herramientas", menu=submenu)    #Opciones
submenu.add_command(label="Limpiar archivo",command=ReseteaTXT)
submenu.add_command(label="Inicializar Robot",command=Inicializa)
submenu.add_command(label="Cambiar Puerto COM y archivo ODO",command=ChngPort)
menu.add_command(label="Salir",command=quit)




reload()


if ser.read(): #Si el robot esta conectado, se puede leer atraves del puerto
```

```
        time.sleep(2)  # tiempo en segundos.
        etiqueta_Ord.config(text="Robot Conectado", background="green")
        EstOrd = "Robot Conectado"
        etiqueta_Ord.place(x=20, y=200)
        etiqueta_Ord = ttk.Label(text="Puerto: " + puerto)
        etiqueta_Ord.place(x=20, y=240)
        print(EstOrd)


else:
        etiqueta_Ord.config(text="Robot No Conectado", background="red")
        EstOrd = "Robot No Conectado"
        etiqueta_Ord.place(x=20, y=200)
        etiqueta_Ord = ttk.Label(text="Puerto: " + puerto)
        etiqueta_Ord.place(x=20, y=240)
        pront="Cerrar Ventana"



print(ser.isOpen)

ventana.mainloop()

def quit(): #Cerrar ventana y tele
    ventana.destroy()
    sys.exit()
```

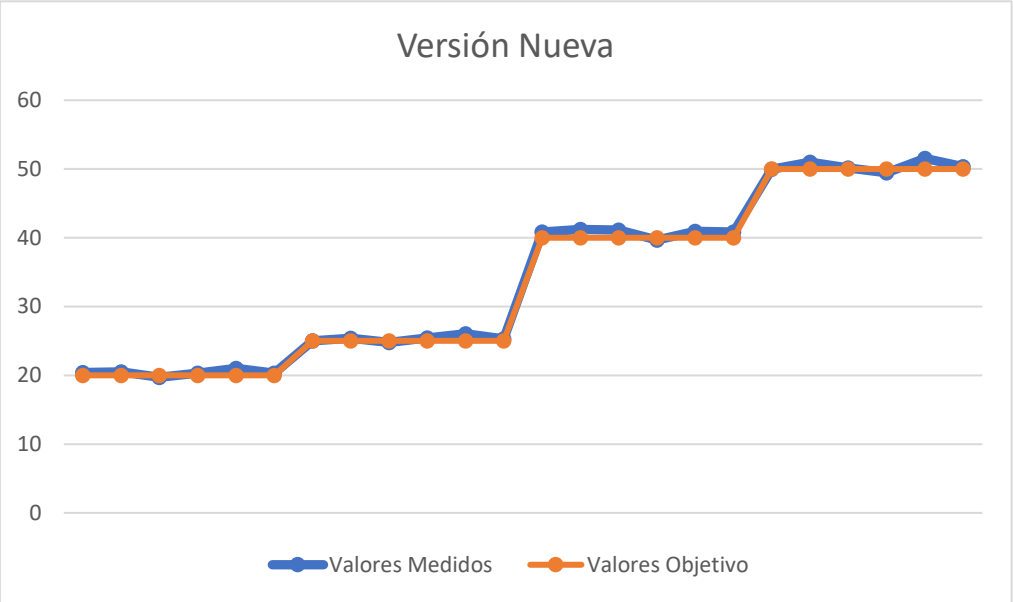
# Anexo 3: Valores Ultrasonidos

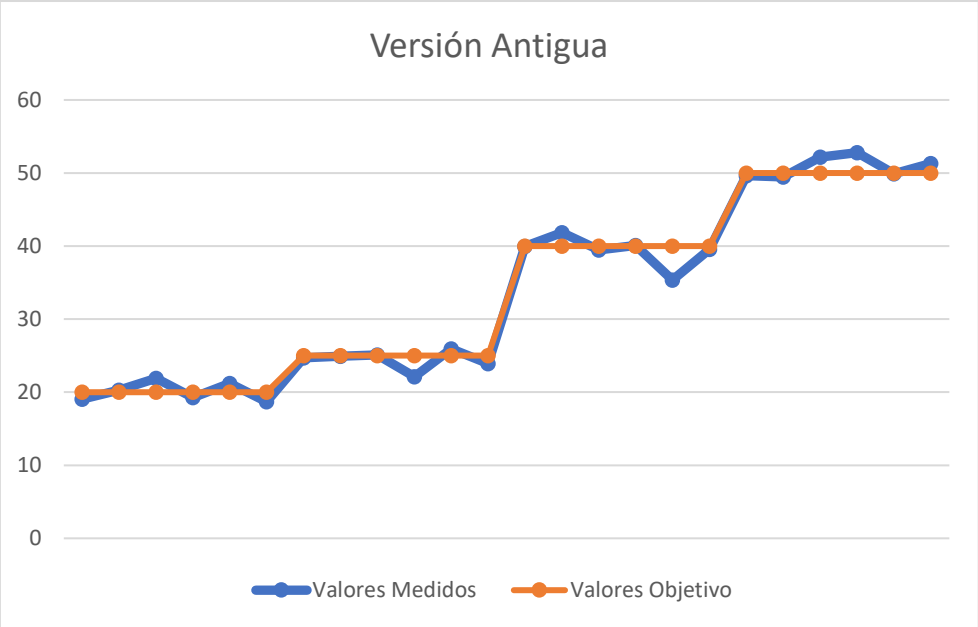
## A.3.1 Mediciones para calibración

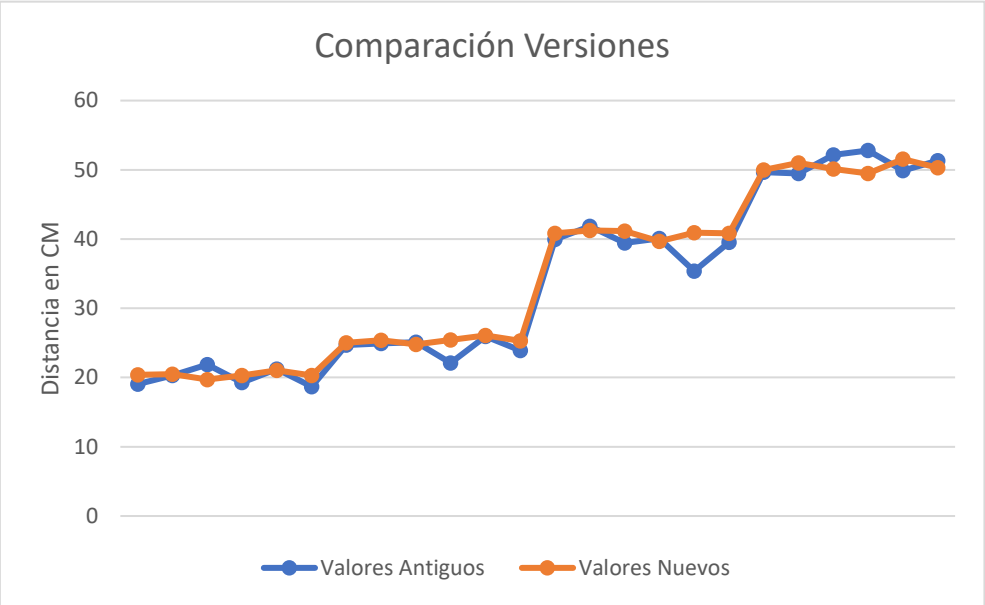
| Distancia CM | Ultra Izq | Ultra Front | Ultra Dcha |
|---|---|---|---|
| 20 | 354 | 313 | 341 |
| 20 | 359 | 314 | 330 |
| 20 | 372 | 323 | 339 |
| 20 | 362 | 316 | 360 |
| 20 | 364 | 315 | 324 |
| 20 | 361 | 331 | 322 |
| 20 | 368 | 309 | 347 |
| 20 | 366 | 321 | 326 |
| 20 | 351 | 278 | 335 |
| 20 | 365 | 279 | 339 |
| 25 | 405 | 401 | 406 |
| 25 | 409 | 405 | 403 |
| 25 | 416 | 400 | 377 |
| 25 | 366 | 417 | 412 |
| 25 | 405 | 433 | 441 |
| 25 | 408 | 398 | 418 |
| 25 | 405 | 400 | 427 |
| 25 | 397 | 394 | 425 |
| 25 | 410 | 399 | 409 |
| 25 | 408 | 407 | 422 |
| 25 | 383 | 408 | 389 |
| 40 | 530 | 669 | 680 |
| 40 | 534 | 662 | 672 |
| 40 | 549 | 676 | 684 |
| 40 | 523 | 678 | 675 |
| 40 | 535 | 667 | 679 |
| 40 | 400 | 643 | 674 |
| 40 | 521 | 662 | 685 |
| 40 | 559 | 661 | 685 |
| 40 | 522 | 669 | 677 |
| 40 | 513 | 680 | 676 |
| 40 | 547 | 671 | 685 |
| 50 | 820 | 849 | 850 |
| 50 | 840 | 842 | 858 |
| 50 | 844 | 827 | 849 |
| 50 | 852 | 810 | 860 |
| 50 | 845 | 831 | 855 |
| 50 | 833 | 840 | 857 |
| 50 | 851 | 852 | 861 |
| 50 | 825 | 845 | 849 |
| 50 | 868 | 836 | 855 |
| 50 | 834 | 867 | 875 |

| | | | |
|---|---|---|---|
| 50 | 849 | 854 | 841 |
| 60 | 1023 | 1015 | 1050 |
| 60 | 1020 | 1042 | 1035 |
| 60 | 1027 | 1011 | 730 |
| 60 | 1019 | 1012 | 1026 |
| 60 | 1023 | 1013 | 1074 |
| 60 | 1013 | 1023 | 1033 |
| 60 | 1050 | 1027 | 1034 |
| 60 | 1025 | 1001 | 986 |
| 60 | 1056 | 1006 | 1056 |
| 60 | 1042 | 1014 | 1040 |
| 60 | 986 | 1067 | 1039 |

### A.3.2 Comparación de algoritmos





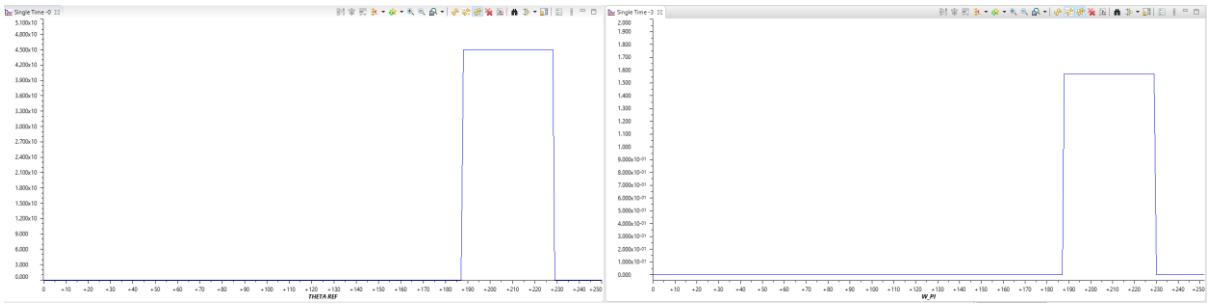




| Ultra Frontal OLD (CM) | | Ultra Frontal NEW (CM) | | Distancia (CM) |
|---|---|---|---|---|
| 19,07 | | 20,4 | | 20 |
| 20,3 | | 20,5 | | 20 |
| 21,89 | | 19,72 | | 20 |
| 19,29 | | 20,32 | | 20 |
| 21,21 | | 21,04 | | 20 |
| 18,69 | | 20,3 | | 20 |
| 24,71 | | 25,02 | | 25 |
| 24,93 | | 25,37 | | 25 |
| 25,1 | | 24,78 | | 25 |

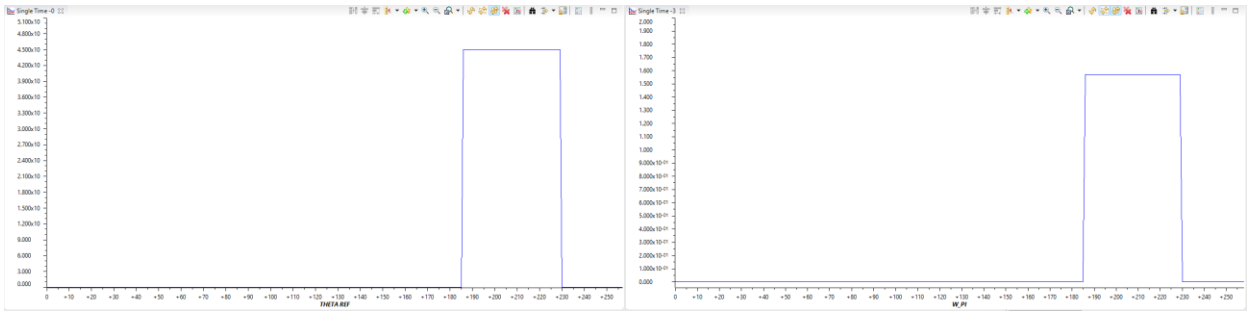
| | | |
|---|---|---|
| 22,11 | 25,42 | 25 |
| 25,92 | 26,06 | 25 |
| 23,92 | 25,28 | 25 |
| 39,94 | 40,85 | 40 |
| 41,85 | 41,24 | 40 |
| 39,454 | 41,14 | 40 |
| 40,1 | 39,675 | 40 |
| 35,37 | 40,92 | 40 |
| 39,54 | 40,85 | 40 |
| 49,64 | 49,99 | 50 |
| 49,47 | 51,01 | 50 |
| 52,17 | 50,14 | 50 |
| 52,79 | 49,48 | 50 |
| 49,886 | 51,54 | 50 |
| 51,32 | 50,32 | 50 |

## Anexo 4. Sintonización del controlador PI:

θ ref = 45º, Kp = 100 y Ki=0 (Valor elevado, satura acción a valor máximo de 1.57 $rad/s$)



θ ref = 45º, Kp=1, Ki=100 (Valor elevado, satura acción a valor máximo de 1.57 $rad/s$)



θ ref = 45º, Kp=1, Ki=10 (Valor adecuado, alcanza referencia de w=0.8 rad/s)