Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Implementación de mejoras en el sistema de navegación autónoma mediante paquetes ROS para una plataforma robotizada de Padrino Tecnológico

Autor: Sara García Navarro

Tutor: Marta Marrón Romera

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Trabajo Fin de Máster

Implementación de mejoras en el sistema de navegación autónoma mediante paquetes ROS para una plataforma robotizada de Padrino Tecnológico

Autor: Sara García Navarro

Director: Marta Marrón Romera

Tribunal:

Presidente: Manuel Ocaña Miguel

Vocal 1: Saturnino Maldonado Bascón

Vocal 2: Marta Marrón Romera

Calificacion: .....

Fecha: .....

A mi familia

Agradecimientos

Este trabajo de fin de máster supone el fin de una etapa, y no hubiera sido posible sin el apoyo incondicional de las personas que me rodean.

Quiero agradecer a mis seres queridos, en especial a mi familia y a mi pareja, todo el cariño y el apoyo que siempre me han brindado, impulsándome a seguir y a no rendirme nunca.

A Juan Carlos y a Alessandro, por toda la ayuda proporcionada a lo largo de este proyecto, y especialmente a Marta, mi tutora, por darme esta gran oportunidad, por su atención, paciencia y apoyo durante estos últimos meses para llevar a cabo este trabajo.

Resumen

El propósito de este trabajo de fin de máster es mejorar el sistema de navegación reactiva y deliberativa de la silla de ruedas robotizada SARA[1] mediante la fusión de la información obtenida por los distintos sensores que componen el sistema, de tipo odométricos, ultrasónicos e inerciales.

Para llevarlo a cabo, se toma como base el trabajo previo [2], donde se creó un sistema de navegación empleando únicamente en la estimación del posicionamiento relativo basado en la odometría. La fusión de sensores permite minimizar los errores aleatorios de los sensores, con el fin de obtener un resultado más fiable. De este modo, se consigue una navegación de la silla más segura para el usuario.

Es necesario trabajar en el bajo nivel, que está formado por los microcontroladores que acceden a las medidas de cada sensor y componente que forma el sistema completo, que se comunica con el alto nivel por bus CAN. El alto nivel está formado por un conjunto de paquetes ROS encargados de fusionar los datos de la odometría del sistema previo, con las medidas obtenidas por el sensor inercial, creando una nueva estimación de la posición de la silla. La nueva odometría resultante se emplea como referencia para calcular las trayectorias del sistema de navegación.

Palabras claves Silla de ruedas, IMU, odometría, ROS, fusión de sensores.

Abstract

The main objetive of this final master project is improve the reactive and deliberative navigation system of robotic wheelchair SARA [1], by the fuse of the different sensors of the system: odometric, ultrasonic and inertials.

To achieve this purpose, the previous final degree project [2] is taken as basis, where a navigation system based on odometry was created. The sensor fusion allows minimize aleatory errors of sensors, obtaining a more reliable estimation. In this way, safer navigation is achieved for the user.

It is necessary to work at the low level of the system, which is composed of the microcontrollers that access the measurements of each sensor. The low level communicates with the high level by CAN bus. The high level consists of a set of ROS packages in charge of fusing the odometry data from the previous system, with the measurements obtained by the inertial sensor, creating a new estimate of the position of the platform. The resulting odometry is used as a reference to calculate the trajectories of the navigation system.

Keywords: Wheelchair, IMU, odometry, ROS, sensor fusion.

Resumen extendido

El propósito de este trabajo de fin de máster es mejorar el sistema de navegación reactiva y deliberativa de la plataforma robotizada desarrollada en el grupo de investigación GEINTRA [3] (SARA y KATE) mediante la fusión de la información obtenida por los distintos sensores que componen el sistema, de tipo odométricos, ultrasónicos e inerciales.

Se toma como punto de partida el trabajo de fin de grado de Francisco Javier la Roda [2], donde se creó un sistema de navegación basado en el posicionamiento relativo basado en la odometría.

El sistema completo se divide en funcionalidad básica o de bajo nivel, formado por los sensores, motores y un sistema empotrado, y otro de funcionalidad avanzada o de alto nivel, que contiene un sistema operativo con librerías de tipo ROS para crear un sistema de navegación.

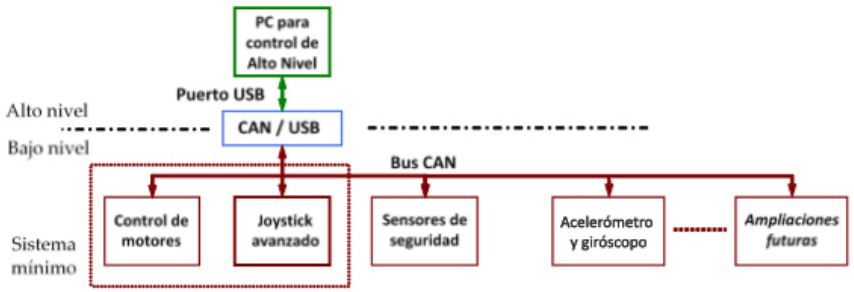

Figura 2: Relación alto nivel y bajo nivel

El sistema de bajo nivel, está formado por distintos nodos en función de sus componentes: nodo joystick, nodo motores y nodo sensores, programados en microcontroladores de núcleo tipo Cortex-M3.

En este trabajo, se modificará el **nodo sensores**, encargado de gestionar la información recibida por los sensores de ultrasonidos, añadiendo las funciones necesarias para incorporar un sensor de tipo inercial (IMU) al sistema. El nodo se encargará de leer las medidas del sensor inercial mediante el uso de comunicaciones I2C, procesarlas y enviarlas al sistema de alto nivel mediante comunicación por bus CAN, para posteriormente calcular la orientación que tiene la silla durante su navegación con las medidas procedentes del IMU.

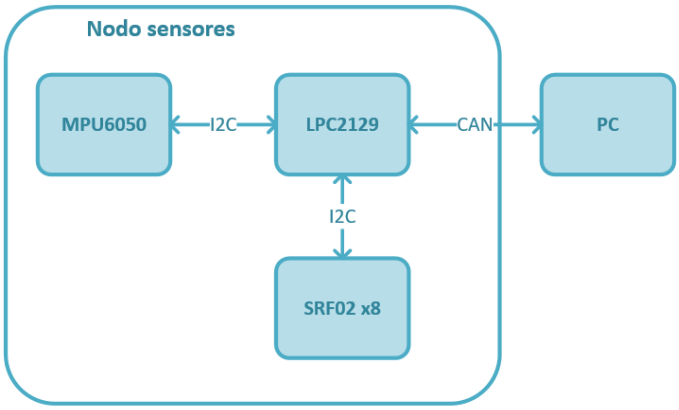

Figura 3: Diagrama de estructura del nodo sensores

El sistema de alto nivel es un sistema de tipo Linux, formado por paquetes de tipo ROS. La información de los sensores llega al alto nivel por bus CAN en forma de mensajes, los cuales se descomponen para poder trabajar con ellos. El sistema cuenta con dos grandes bloques para crear el sistema de navegación, que se comunican entre ellos:

- **Controlador e interfaz hardware**: se define la geometría del robot, y se calcula su odometría a partir de las consignas de velocidad necesarias para mover el robot.
- **Navegador**: a partir de la odometría calculada con la información de los encoders, se va creando una trayectoria con unas consignas de velocidad que el robot deberá seguir para llegar al destino definido.

La odometría calculada a partir de la información de los encoders de las ruedas acumula errores, de forma que la estimación de la posición de la plataforma no fuese correcta. Es por ello que se integra la información recibida por el acelerómetro y giróscopo del sensor inercial empleando la **fusión de sensores**, consiguiendo una estimación más precisa minimizando los errores sistemáticos de la odometría. Para ello, se emplea el paquete ROS *robot_localization*, que aplica un filtrado de Kalman para calcular esta estimación de forma sencilla.

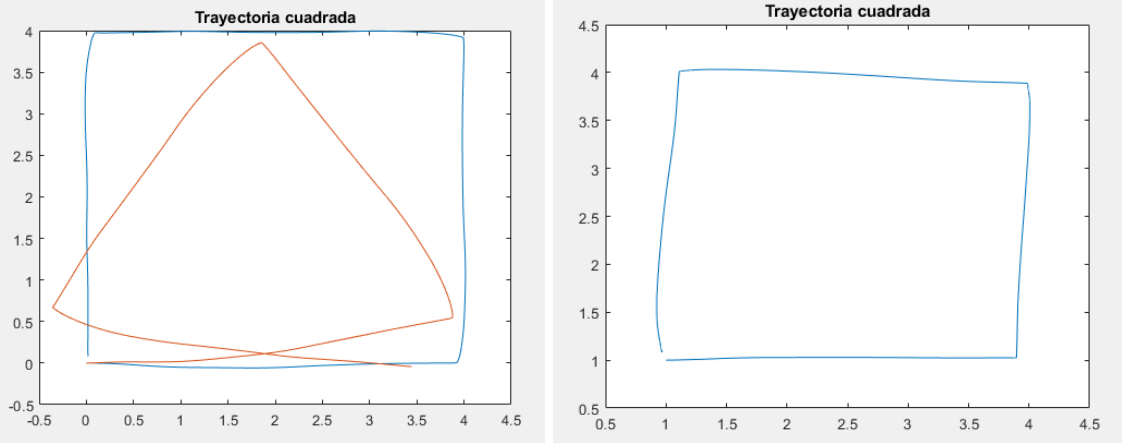

Figura 4: Comparativa de una trayectoria calculada por odometría o por fusión de sensores

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto del proyecto

En la actualidad en España, existen 2,5 millones de personas con movilidad reducida, y alrededor del 74 % de ellos precisan de ayuda para salir de sus casas, generalmente por la falta de accesibilidad de sus viviendas. Entre ellos, un 40 % emplea silla de ruedas manual o eléctrica, lo que hace que su dependencia sea mucho mayor.

Actualmente, no existen muchas soluciones avanzadas que favorezcan la independencia de las personas que emplean silla de ruedas, limitándose a añadir un joystick adaptado solo a algunos perfiles de movilidad. En los últimos años, se han desarrollado muchos proyectos de investigación para solucionar este problema mediante el empleo de la robótica.

El grupo GEINTRA [3] de la Universidad de Alcalá, desarrolló el proyecto SAR [1] (Sistema de Ayuda a la movilidad Robotizado), con el fin de favorecer la independencia de las personas con movilidad reducida mediante el empleo de sillas de ruedas robotizadas con navegación autónoma o semi-autónoma. El objetivo último de dicho proyecto es el de aportar soluciones avanzadas dentro del área de las tecnologías de apoyo, "Assistive Technologies" (ATs) a partir de productos comerciales estándar.

Como consecuencia del proyecto, se desarrolló un prototipo de silla de ruedas adaptable en función de las necesidades del usuario: mediante modificaciones de una silla eléctrica comercial, se obtuvo una plataforma robotizada, dotada de nodos inteligentes que controlan varios tipos de sensores y actuadores, comunicados a través de una red CAN [9]. A esta plataforma modular se le añadió además una interfaz Hombre-Máquina (HMI) y la funcionalidad de navegación autónoma.

En este contexto, este trabajo de fin de máster (TFM) parte del trabajo previo [2], donde se creó el sistema de navegación basado en la odometría, que permite estimar la posición relativa de la silla respecto a una posición inicial según navega. Se observó que dicha estimación, acumulaba errores en el tiempo (deslizamientos de las ruedas, cambios en la geometría en función del peso del usuario), creando así la necesidad de completar la estimación de la posición con otros sensores. En este TFM, se integra un sensor de tipo inercial.

A continuación, en la figura 1.1, se muestra un esquema de la silla de ruedas, formada por dos partes: silla de funcionalidad básica (SFB), y silla de funcionalidad avanzada (SFA). En este TFM, se actuará sobre las dos: mediante la integración de un IMU [10] se actuará sobre SFB, y a partir de los datos obtenidos y junto con datos obtenidos del resto de sensores que estiman su posicionamiento, se modifica el sistema de navegación autónoma sobre una plataforma que se ajuste en un procesador de tipo Raspberry Pi (SFA).

Figura 1.1: Esquema funcional de la silla de ruedas

Las pruebas y validación del prototipo diseñado se realizarán con la ayuda de la asociación "Padrino Tecnológico" [11], con el fin de su adaptación a los diferentes perfiles.

1.2 Estado de la cuestión

El posicionamiento relativo de un robot, es aquel que se calcula mediante el uso de los diferentes sensores embarcados en el robot (encoders, giróscopos, acelerómetros, etc). La odometría es un método más sencillo que permite estimar la posición relativa de un robot móvil, transformando el movimiento de sus ruedas en un desplazamiento lineal. Pero existen errores que se acumulan en el tiempo [4] que hacen que esta estimación no sea tan precisa. Se pueden dividir estos errores en dos tipos:

- **Errores sistemáticos**: se producen en todas las medidas, debido a fallos como una mala alineación en las ruedas, diámetros diferentes, resolución limitada del encoder, frecuencia de muestreo de datos limitada. Estos errores hacen que el resultado de la estimación se acumule en el tiempo.
- **Errores aleatorios**: aquellos que se producen bajo determinadas circunstancias y que no se pueden predecir: irregularidad del terreno, deslizamiento de las ruedas, impacto con obstáculos.

En la figura 1.2, se representan estos tipos de errores sistemáticos realizando una trayectoria cuadrada debido a (a) incertidumbre en la distancia entre ejes de las ruedas debido a errores en el modelado cinemático y (b) diámetros diferentes de las ruedas, donde se observa que el punto final de la trayectoria no es el esperado. Estos errores, se pueden compensar y corregir mediante el uso de software de procesamiento del sensado odométrico.

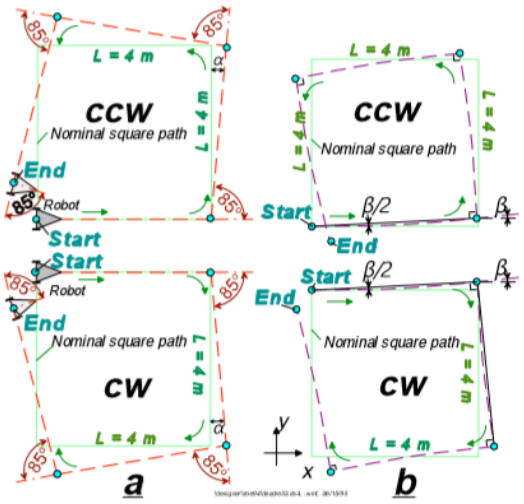

Figura 1.2: Representación de errores sistemáticos en odometría [4]

Los errores aleatorios no se pueden predecir, y por tanto, no se podrán compensar. Una solución para eliminar o reducir estos errores es la de fusionar la información de la odometría con la procedente de otros sensores. A este proceso se le denomina *sensor fusion* [12], y se basa en hecho probabilístico de que al combinar los datos de distintas fuentes (sensores), el resultado obtenido tiene menor incertidumbre y por tanto, sea más fiable que la obtenida individualmente.

En 1996, surge el concepto de Girodometría (*Gyrodometry* [5]), una forma eficaz de combinar los datos procedentes de los enconders con los de un giróscopo, que "*se basa en la hipótesis de que la discrepancia entre la curva de odometría y la curva de giro persiste solo durante un período de tiempo muy corto.*"De la misma forma que la odometría presenta errores sistemáticos, el uso del giróscopo produce un error denominado deriva que hace que el valor de la orientación del robot crezca de forma ilimitada y continua con el tiempo. La girodometría es capaz de reducir los problemas ocasionados por estos errores.

En la figura 1.3, se representan los errores que un robot tiene al estimar su orientación empleando el método de odometría (ϵ_{odo}), de la medida procedente del giróscopo (ϵ_{gyro}) y el error calculado empleando el método de girodometría (ϵ_{go}), observando que este último es prácticamente 0, compensando los errores de los dos sensores.

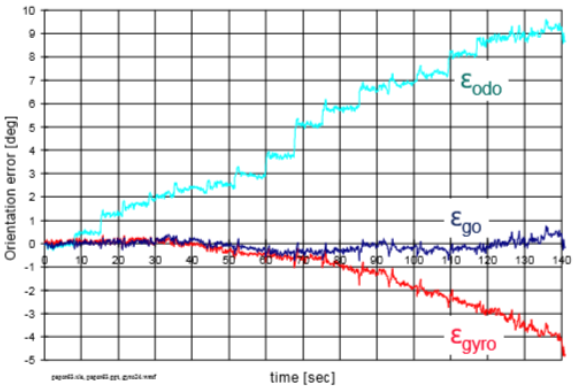

Figura 1.3: Representación de la compensación del error empleando el método Gyrodometry [5]

A partir de este concepto de Girodometría, en la Universidad de Keio surge el siguiente experimento [6], donde se combinan las estimaciones del giro obtenidas a partir de la odometría y del giróscopo para calcular la trayectoria que sigue un robot de dos ruedas de cinemática diferencial. Este método compensa el error por el deslizamiento de las ruedas, de modo el valor de la velocidad angular del movimiento se calcula mediante un promediado entre los valores estimados mediante la odometría y la basada en el giróscopo.

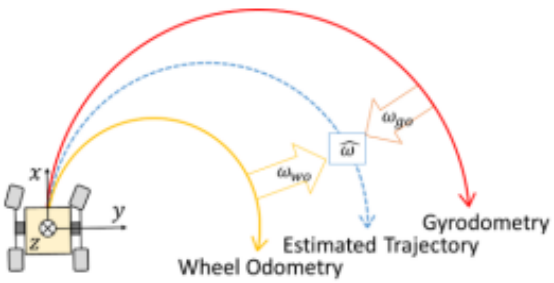

Figura 1.4: Estimación de la trayectoria de un robot de dos ruedas [6]

En la siguiente figura (1.5), se representa la trayectoria cuando el ángulo de deslizamiento del robot es de 10º, donde se muestra la trayectoria real (negro), la estimada por la odometría de las ruedas (verde) y por la odometría del giróscopo (azul). Se observa que, el valor estimado empleando únicamente el de la odometría calculada por los encoders, es bastante menor que el real. Al emplear el giróscopo, se obtiene una trayectoria más similar, aunque de valor mayor a la ideal. Al realizar el promediado de estas dos estimaciones (rojo) se consigue tener una trayectoria más próxima al valor real, obteniendo un error acumulado menor del 10 %.

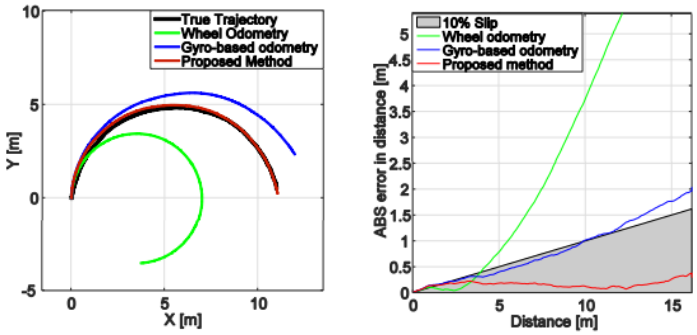

Figura 1.5: Comparativa de los diferentes métodos de estimación de una trayectoria cuando se produce deslizamiento [6]

En el caso de realizar una trayectoria cuadrada, la estimación basada únicamente en la odometría es bastante pobre, mientras que usando el giróscopo, la estimación obtenida consigue acercarse bastante a la trayectoria real. Tanto el giróscopo como el método del promediado, tienen un error acumulado menor del 10 %, como se puede observar en la figura 1.6.

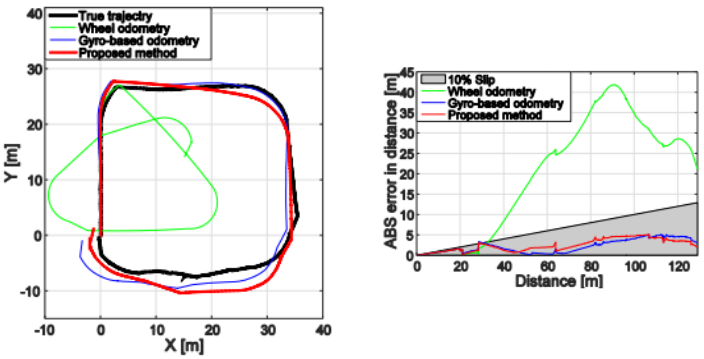

Figura 1.6: Comparativa de los diferentes métodos de estimación de la trayectoria cuadrada [6]

1.3 Objetivos

El objetivo principal de este trabajo de fin de máster, es el de mejorar el sistema de navegación reactiva y deliberativa, fusionando los datos procedentes de medidas inerciales con los de la odometría mediante el empleo de paquetes del sistema operativo ROS [13] para la silla de ruedas automatizada SARA. De este modo, la silla será capaz de navegar en cualquier entorno (conocido o desconocido) modificando su ruta para no colisionar con los obstáculos presentes, siempre de forma segura para el usuario. Se parte del desarrollo del trabajo previo realizado por el departamento de Electrónica. Para ello será necesario trabajar en dos partes:

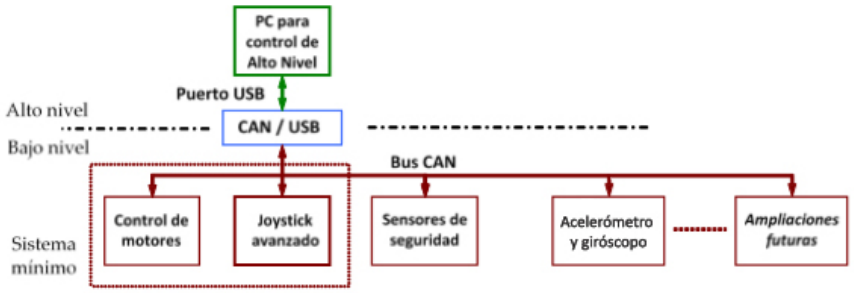

Figura 1.7: Relación alto nivel y bajo nivel

- El bajo nivel de la arquitectura es flexible y escalable en función de las necesidades de cada usuario, está formado por: encoders ópticos acoplados a los ejes de las dos ruedas de plataforma, sistema empotrado de control de bajo nivel (basado en bus CAN y microcontrolador [14] de núcleo tipo Cortex-M3), y complementado por sensores ultrasónicos para la detección de obstáculos. Se integra un sensor para estimar la posición relativa de un robot móvil: un IMU ("Inertial Measurement Unit", unidad de medición inercial), dispositivo electrónico formado por un acelerómetro y giróscopo.
- El alto nivel de la arquitectura está formado por un sistema empotrado con pantalla táctil, con sistema operativo estándar (Linux), y cuya aplicación se basa en librerías de código abierto tipo ROS. Se modifica el sistema de navegación autónoma, adaptado para el usuario, a partir de los datos obtenidos por los sensores de bajo nivel.

1.4 Organización de la memoria

Este TFM se centra en el diseño de un sistema de posicionamiento relativo/absoluto de la silla de ruedas mediante sensores de tipo odométricos, inerciales y ultrasónicos. Se documenta en esta memoria, que está estructurada de la siguiente manera:

- **Capítulo 1**: donde se realiza una introducción del TFM y se marcan los objetivos mediante el análisis de los antecedentes y proyectos previos.
- **Capítulo 2**: Descripción del bajo nivel de la silla de ruedas, donde se incorpora un sensor capaz de proporcionar la información necesaria para conocer la posición del prototipo.

  Éste será un IMU o unidad de medición inercial comercial: MPU6050 [8], con 6DOF (Degrees Of Freedom), formado por un acelerómetro de 3 ejes y un giróscopo de 3 ejes, proporcionando información de la aceleración y la velocidad de rotación en los ejes X, Y y Z. Los datos del IMU se reciben mediante el protocolo de comunicaciones I2C en el microcontrolador. Esta información se transformará y se enviará después mediante la red CAN.
- **Capítulo 3**: Descripción del alto nivel modificado a partir de paquetes ROS (*robot_localization*[15]). Permitirán una mejora del posicionamiento relativo del sistema mediante el uso del filtro de Kalman.

  El objetivo del filtro es fusionar y filtrar la información de posición de la silla y estimar la posición futura del robot, eliminando el ruido y recalculando la posición del robot móvil a partir de la posición actual del robot, y la trayectoria de éste.
- **Capítulo 4**: se definen los experimentos realizados y los resultados obtenidos.

- **Capítulo 5**: se muestran las conclusiones obtenidas a partir de los resultados y se definen futuras líneas de trabajo.
- **Capítulo pliego de condiciones**, con el hardware y software necesarios para realizar el proyecto.
- **Capítulo presupuesto**, donde se detallan los costes que suponen la realización del proyecto.

Capítulo 2

Adaptación del bajo nivel

2.1 Introducción

El sistema que forma la silla de ruedas automatizada está compuesta por un bajo nivel que se comunicará con el alto nivel mediante el protocolo serie CAN. El bajo nivel, a su vez, está formado por diferentes nodos encargados de gestionar y controlar los distintos componentes del sistema.

- Nodo motores: para navegar de forma automática o controlada por un joystick, la silla tiene instalados 2 motores eléctricos capaces de mover la plataforma, con una cinemática diferencial. Además, cada motor cuenta con un encoder acoplado que sirve para saber cuánto ha rotado cada rueda, y emplear esta información para obtener la estimación de la posición de la plataforma [16].

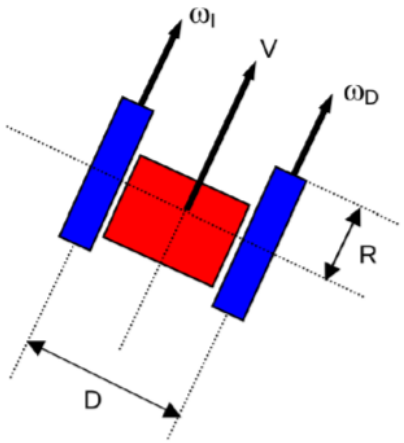

Figura 2.1: Cinemática de un robot diferencial

- Nodo sensores: la plataforma también cuenta con sensores que permiten conocer el entorno que la rodea y a qué distancia se encuentran los obstáculos, con el fin de poder evitarlos.
- Nodo joystick: el usuario tiene la opción de realizar una navegación "manual" además de la autónoma, formado por una interfaz hombre-maquina tipo joystick, de forma que pueda decidir la dirección y velocidad del sistema continuamente.

Estos nodos han sido implementados en trabajos previos [2]. En este trabajo, se modifica el nodo sensores añadiendo una unidad de medición inercial (IMU), formada por un acelerómetro y giróscopo, que permite conocer la orientación y detectar impactos en el sistema.

El nodo sensores es flexible y escalable en función de las necesidades de cada usuario, y está formado por un sistema empotrado de control de bajo nivel (basado en microcontrolador de núcleo tipo Cortex-M3) y por distintos sensores ultrasónicos.

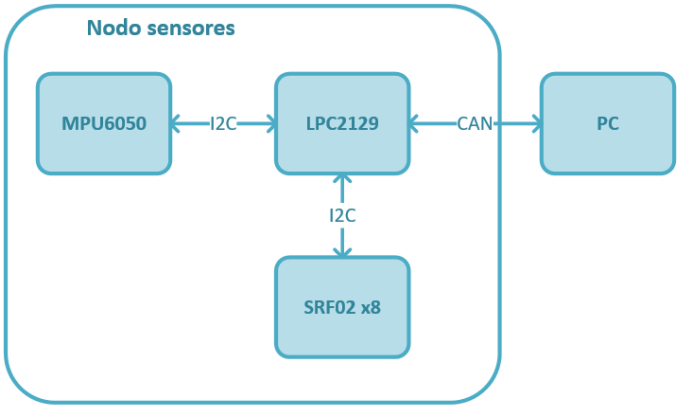

Figura 2.2: Diagrama de estructura del nodo sensores

2.2 Sistema empotrado

Cada uno de los nodos que forman el sistema está compuesto por un microcontrolador que permite comunicarse tanto con los distintos sensores y dispositivos que forman nodo como con el alto nivel, al que se enviarán los datos que proporcionan los sensores.

Se elige para ello el microcontrolador LPC2129 *CAN QuickStart* [17], que cuenta con distintos pines de E/S, un driver de buses CAN, otro de comunicación I2C y es capaz de proporcionar alimentaciones de 3.3 y 1.8 V a partir de sus 5 V de alimentación externa. Además, su configuración y la programación de sus distintas funcionalidades es sencilla, empleando el lenguaje de programación C.

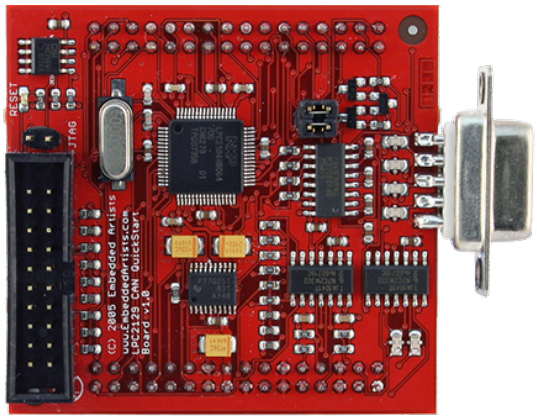

Figura 2.3: LPC2129 CAN QuickStart Board

Al nodo sensores, se le incorpora un sensor capaz de proporcionar la información necesaria para

conocer la orientación de la plataforma. Éste será un IMU o unidad de medición inercial: el módulo MPU6050 de 6DOF, formado por un acelerómetro de 3 ejes y un giróscopo de 3 ejes, proporcionando medidas de aceleración y velocidad de rotación en los ejes X, Y y Z.

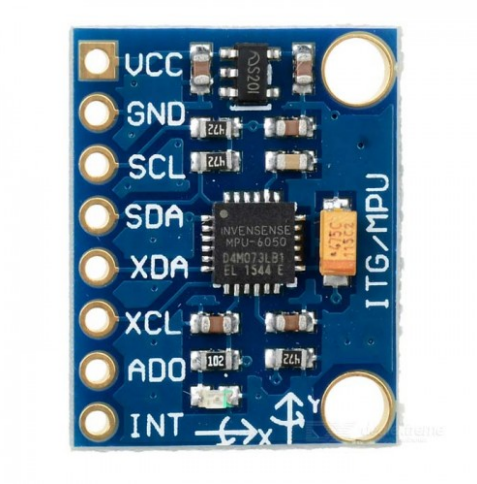

Figura 2.4: Módulo MPU6050

Un acelerómetro sirve para medir las aceleraciones de un cuerpo (relacionadas con su peso), por lo que si la plataforma donde se instala se encuentra en una superficie plana, medirá únicamente aceleración en el eje Z, mientras que si el cuerpo está inclinado, el peso estará distribuido entre los distintos ejes. Si el cuerpo se encuentra en caída libre, todos los ejes se mide un valor de aceleración nulo. Por su parte, un giróscopo sirve para medir la orientación de un objeto mediante la medición de su velocidad de rotación en sus ejes (X, Y, Z).

Los datos generados por ambos sensores (acelerómetro y giróscopo) se envían desde el módulo mediante el protocolo de comunicaciones I2C. Esta información se transformará en el nodo sensores y se enviará mediante la red CAN del microcontrolador al sistema de funcionalidad avanzada (1.7).

2.3 Protocolo de comunicaciones I2C

El bus serie de datos I2C (*Inter-Integrated Circuit*) es ampliamente utilizado para la comunicación entre un controlador y otros circuitos periféricos. En este caso, servirá para comunicar el sensor IMU MPU6050 con el microcontrolador LPC2129, de modo que el microcontrolador sea el maestro que controle el inicio y fin de la transferencia de datos procedentes de o hacia el esclavo (MPU6050).

Tanto en la lectura como en la escritura de datos, se podrá enviar/recibir un solo byte o una ráfaga de ellos, como se puede ver en los siguientes diagramas de las figuras 2.5 y 2.6 de la hoja característica del módulo MPU6050 [8].

El proceso de escritura cuenta con la siguiente secuencia:

1. En la escritura de datos, el maestro da la orden de comienzo de la comunicación, seguido de 8 bits correspondientes a los 7 de la dirección del esclavo y el bit de escritura (0).

2. El esclavo reconoce la transferencia, por lo que el maestro envía la dirección del registro interno al que quiere enviar el dato.

3. El esclavo responde de nuevo con ACK, por lo que el maestro envía el dato (DATA), que después de ser reconocido (ACK) por el esclavo, acaba con la secuencia de finalizado (P).

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Figura 2.5: Secuencia de escritura de un byte o una ráfaga de bytes

El proceso de lectura es similar al de escritura (2.5).

1. En primer lugar se realiza el mismo proceso de escritura, donde el maestro indica de qué registro leerá (AD+W).
2. El maestro no envía después ningún dato, sin embargo, repite la condición de start (S) y a continuación envía la dirección del esclavo (AD+R), seguido del bit de lectura (RA).
3. El esclavo responde con ACK y envía el dato solicitado hasta que el maestro decida finalizar la lectura de datos, respondiendo con ACK en caso de que quiera recibir más datos o con NACK y la señal de finalización de transmisión de datos cuando decida finalizar.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Figura 2.6: Secuencia de lectura de un byte o una ráfaga de bytes

En la tabla 2.1 se recogen las señales empleadas en la lectura y escritura de datos entre el maestro y el esclavo, junto con una breve descripción de ellas.

Señal	Descripción
S	Comienzo de comunicación
AD	Dirección del esclavo (0x68 o 0x69 en función del valor de AD0 en el módulo MPU6050)
W	Bit de escritura (0)
R	Bit de lectura (1)
ACK	"Acknowledge"
NACK	"Not-Acknowledge"
RA	Dirección de registro interno
DATA	Dato a transmitir o a recibir de 8 bits
P	Fin de comunicación

Tabla 2.1: Definición de las señales de comunicación I2C

La transmisión de datos se realiza mediante dos líneas de comunicación: SCL (*Serial CLock*) y SDA (*Serial Data*), que se encuentran tanto en el maestro como en el esclavo, conectándose entre sí. En el siguiente cronograma (ver figura 2.7) se define el nivel de las señales SDA y SCL en los procesos anteriores.

- El proceso de comienzo de transmisión (S) empieza cuando se produce un flanco de bajada en la línea SDA si SCL se encuentra a nivel alto.
- Finaliza el proceso de transmisión al producirse un flanco de subida en SDA si SCL se encuentra a nivel alto (P).
- Cuando ha comenzado la transmisión de datos, en los siguientes 8 ciclos de la señal de reloj SCL, se enviará el dato (dirección del esclavo + R/W, registro interno o dato a leer o escribir), y el noveno ciclo corresponderá al bit de reconocimiento ACK (0) o NACK (1).

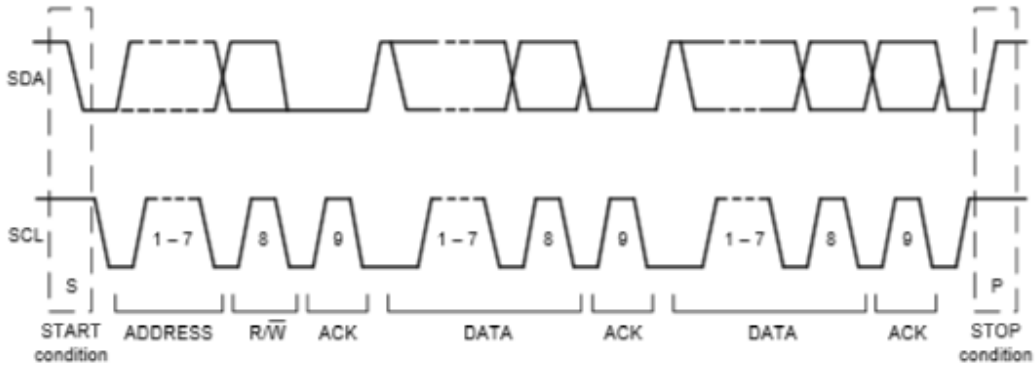

Figura 2.7: Comportamiento de las señales SDA y SCL

El conexionado del sensor con el microcontrolador para el envío y recepción de datos se define en la siguiente tabla 2.2:

MPU6050	LPC2129
VCC	5 V
GND	GND
SCL	P0.2 – SCL
SDA	P0.3 – SDA

Tabla 2.2: Conexionado entre MPU6050 y LPC2129

2.4 Protocolo de comunicaciones CAN

El microcontrolador seleccionado dispone de dos canales para la comunicación mediante el protocolo CAN (*Controller Area Network*). La trama de los mensajes CAN sigue la estructura definida en el apartado 3.2.5 del trabajo de fin de grado [2].

En este caso, se envían diferentes mensajes al alto nivel que contienen información procedente de los distintos nodos que componen la plataforma (2.1).

El nodo sensores se encarga de enviar por bus CAN los datos procedentes de los sensores de ultrasonidos (a qué distancia se encuentra un obstáculo de la plataforma), los datos procedentes del acelerómetro (aceleración de los ejes X, Y, Z) y giróscopo (velocidad de rotación de los ejes X, Y, Z). A cada mensaje se le ha asignado un identificador para poder procesarlo en el alto nivel.

2.5 Estructura Software del "Nodo Sensores"

El nodo sensores se encarga de recibir y procesar los datos procedentes de los objetos detectados por sensores de ultrasonidos. En este proyecto, se ha añadido un sensor de tipo IMU que ayudará a la navegación del sistema. La estructura del bajo nivel quedaría de la siguiente forma:

Figura 2.8: Estructura del código del "Nodo Sensores" con las modificaciones para el procesado de los datos del IMU

El bloque "System Calls" está formado por aquellos módulos que sirven para el envío y recepción de los protocolos I2C y bus CAN:

- can.c: contiene la función para enviar dos mensajes de 16 bits cada uno por el bus CAN, indicando el id de dicho mensaje.
- funciones_i2c.c: este código contiene las funciones para realizar la escritura y lectura mediante protocolo I2C. Se emplea la función "*leo_I2C*" que recibe la dirección del sensor (IMU o ultrasonidos) y el registro al que se quiere acceder en él, y devuelve el dato como un número entero de 8 bits.

- LPC_FullCAN_SW.c: contiene las funciones de configuración, envío y recepción de mensajes, necesarias para comunicar por bus CAN el bajo nivel y el alto nivel.
- i2c_sara.c: a diferencia de "funciones_i2c.c", con este módulo se puede elegir el número de bytes a transmitir o recibir por el bus I2C. De esta forma, si hay un dato cuyo tamaño es 16 bits, se puede proceder a su lectura de forma global.

Por su parte, el bloque "Source Code", está formado por el código principal que se ejecuta en el LPC2129:

- main.c: es el hilo principal que ejecuta el LPC2129, el cual es capaz de llamar a las funciones necesarias para obtener los datos de los sensores, y enviarlos mediante tramas CAN.
- setup.c: contiene la configuración del "timer" empleado para enviar los mensajes CAN de forma temporizada.
- SarIMU.c: se incluyen las funciones que sirven para realizar la configuración, calibración y conversión de unidades de las medidas proporcionadas por el sensor IMU mediante la lectura y escritura en sus registros, mediante la comunicación I2C.

La plataforma está formada por distintos nodos que sirven para controlarla y manejarla. Al "Nodo Sensores" de SARA, con ultrasonidos que detectan a qué distancia se encuentra un posible obstáculo, se le añade también un sensor inercial de tipo IMU para estimar la posición relativa de la plataforma. La información del IMU se alternará y enviará junto a las medidas de los ultrasonidos, mediante la comunicación por bus CAN entre el bajo y el alto nivel.

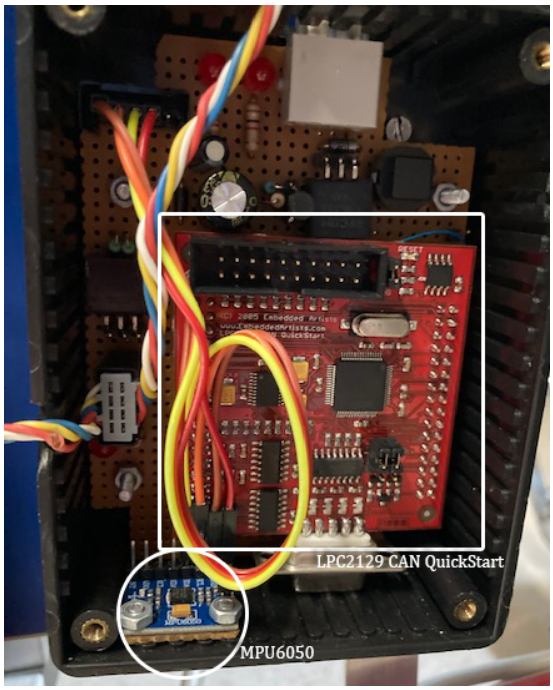

Figura 2.9: Incorporación del módulo MPU6050 al nodo sensores

La plataforma SARA cuenta con dos cajetines su lado derecho donde se encuentra el microcontrolador del "Nodo Sensores" y "Nodo Motores", con su correspondiente alimentación. En la figura 2.9, se muestra el cajetín del "Nodo Sensores", con el microcontrolador LPC2129 CAN Quickstart y el módulo MPU6050 en posición horizontal.

2.5.1 Registros del MPU6050

Mediante el empleo de un acelerómetro y un giróscopo del IMU, que es capaz de dar información sobre la velocidad, orientación y fuerzas gravitacionales del sistema en el que esté integrado, es posible estimar la posición de este sistema, integrando la información del IMU.

Para ello se emplea el sensor comercial MPU6050, con un acelerómetro y giróscopo que forman un sistema de 6 ejes. Dispone de dos convertidores ADC de 16 bits para digitalizar los valores tanto del giróscopo como del acelerómetro, con escalas programables, (± 250, ± 500, ±1000 y ±2000 °/s para el giróscopo) y (±2g, ±4g, ±8g y ±16g para el acelerómetro, donde 'g' representa el valor de la gravedad), en función de la precisión de medida requerida. Además, la calibración del dispositivo es sencilla, de forma que se puede emplear en cualquier tipo de superficie.

Estas medidas son almacenadas en los registros del IMU, a los que se accederá desde el microcontrolador mediante comunicación I2C con una velocidad máxima de 400 kHz. A continuación en la tabla 2.3, se recogen los registros empleados para esta aplicación, con su descripción y dirección.

Dirección	**Nombre**	**Descripción del registro**
0x75	WHO_AM_I	Devuelve la dirección del bus I2C. Por defecto, 0x68
0x6B	PWR_MGMT_1	Permite configurar el modo encendido y la fuente de reloj. Permite encender y apagar el sensor
0x68	SIGNAL_PATH_RESET	Resetea los valores del acelerómetro, giróscopo y termómetro
0x37	INT_PIN_CFG	Define el comportamiento de las interrupciones
0x1B	GYRO_CONFIG	Configura la sensibilidad de las medidas del giróscopo
0x1C	ACCEL_CONFIG	Configura la sensibilidad de las medidas del acelerómetro
0x1F	MOT_THR	Se fija el umbral para la habilitación de la interrupción de detección de impactos en el sistema
0x20	MOT_DUR	Se define el valor de la duración de la interrupción de detección de impactos
0x69	MOT_DETECT_CTRL	Delay
0x38	INT_ENABLE	Permite habilitar las interrupciones del sistema: detección de movimiento, de caída libre
0x3A	INT_STATUS	Sirve para leer el estado de las interrupciones habilitadas, que se pondrán a nivel alto cuando se produzcan
0x3B	ACCEL_XOUT_H	Almacena los valores del acelerómetro: **ax**, **ay**, **az**, en 16 bits
0x43	GYRO_XOUT_H	Almacena los valores del giróscopo: **gx**, **gy**, **gz**, en 16 bits

Tabla 2.3: Registros del módulo MPU6050 [8]

2.5.2 Código SarIMU.c

Existe un gran número de registros para configurar el sensor empleado, por lo que se decide crear un fichero ("SarIMU.c" y su correspondiente fichero cabecera "SarIMU.h") que englobe, de forma ordenada y estructurada, todas las funciones que serán necesarias para obtener las medidas del acelerómetro y giróscopo. El fichero tendrá la siguiente estructura, cuya funcionalidad se explicará en los siguientes apartados:

- configMPU6050()
- calibracion()
- leer_IMU()
  - getAcceleration()
  - getRotation()
- impacto()

2.5.3 void configMPU6050()

En primer lugar, se crea una función que inicializa y configura el módulo MPU6050, de forma que el módulo se habilite para el envío de datos, y además, se configura una interrupción que para detectar posibles golpes en el sistema. Para ello, es necesario emplear el mapa de registros del sensor.

En la tabla 2.4, se muestran los registros necesarios para realizar la configuración inicial del sensor, con el valor que se fijará en cada registro y la función que hará dicho valor.

Dirección	Valor	Función del registro
0x6B	0x80	Habilita y despierta el IMU, de forma que no esté dormido y sea capaz de comunicarse con el microcontrolador
0x68	0x07	Realiza un reseteo de los valores del acelerómetro, giróscopo y temperatura
0x37	0x20	Se configura el comportamiento de las señales de entrada de interrupción, en este caso, push-pull
0x1C	0x01	Se configura un filtro digital, cuya frecuencia de corte en este caso es de 5 Hz
0x1F	20	Se define el valor a superar en cualquier eje del acelerómetro para considerar que se ha producido un impacto en el sistema
0x20	40	Se define la duración de la interrupción en ms
0x69	0x15	Se añade un delay a los datos del acelerómetro. Cuando se produce un movimiento o caída libre del sensor, se incrementa un contador, en este caso con incrementos de 1, de ambos registros.
0x38	0x40	Se habilita la interrupción de detección de impactos

Tabla 2.4: Configuración de los registros de inicialización

A continuación, se muestra el código que realiza la configuración del sensor, mediante el uso de la función "*escribo_I2C*".

```
// Funcion para configurar IMU: MPU6050
void configMPU6050(){
        //      Habilitar interrupcion para choque:
        escribo_I2C((0x68<<1)&(0xFE),0x6B,0x80);
        escribo_I2C((0x68<<1)&(0xFE),0x68,0x07); //Reset ac, gy, temp
        escribo_I2C((0x68<<1)&(0xFE),0x37,0x20); //Active high, push-pull signal
        escribo_I2C((0x68<<1)&(0xFE),0x1C,0x01); //Digital pass filter: 5Hz.
        escribo_I2C((0x68<<1)&(0xFE),0x1F,20);   //Motion threshold
        escribo_I2C((0x68<<1)&(0xFE),0x20,40);   //40ms
        escribo_I2C((0x68<<1)&(0xFE),0x69,0x15); //
        escribo_I2C((0x68<<1)&(0xFE),0x38,0x40); //Enable motion detection interrupt
            .
}
```

2.5.4 void calibracion()

El proceso de calibración de un sensor, consiste en realizar unas medidas iniciales y compararlas con los valores de referencia, con el fin de modelar el error sistemático y aleatorio, a fin de poder eliminarlo en la medida de lo posible en el funcionamiento normal del sensor.

De este modo, cada vez que se reinicia el sistema global, se calibra el sensor para evitar que se produzca un error en las medidas, ya que el sensor puede estar desnivelado por un mal posicionamiento de este al instalarlo en la plataforma. La dirección de cada eje de medida del acelerómetro y giróscopo, viene definida por defecto en el módulo del sensor, como se representa en la figura 2.10:

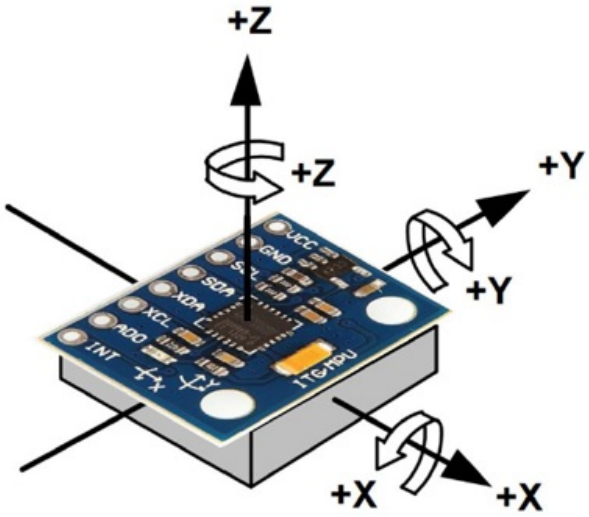

Figura 2.10: Direcciones de los ejes X, Y, Z del módulo MPU6050

De esta figura (2.10), se pueden deducir los valores esperados de cada variable con el sistema en reposo en posición horizontal (2.9), que se recogen a continuación:

Variable	Valor ideal	Valor sensor en 16 bits
Ax	0 m/s^2	0
Ay	0 m/s^2	0
Az	9.8 m/s^2	16384
Gx	$0^o/s$	0
Gy	$0^o/s$	0
Gz	$0^o/s$	0

Tabla 2.5: Valores del acelerómetro y giróscopo en reposo

Con la función *calibracion()* se pretende que, al iniciar el sistema, se haga un muestreo inicial de las medidas generadas por el sensor, y se ajuste su offset o error sistemático alcanzando los valores recogidos en la tabla 2.5.

El IMU dispone para realizar este ajuste, un conjunto de registros de 16 bits que será necesario configurar: 0x06, 0x08 y 0x0A para el offset del acelerómetro en los ejes X, Y, Z, y 0x13, 0x15 y 0x17 para los valores de offset del giróscopo en los ejes X, Y, Z.

El offset del acelerómetro cuenta con un margen de medida de ±16g, con pasos de 0.98mg, y el giróscopo cuenta con un span de $\pm 1000^o/s$. Para calcular el offset, será necesario tomar medidas al iniciar el sistema, y compararlas con el valor deseado. Las medidas del IMU se obtienen con una precisión de ±2g en caso del acelerómetro y $\pm 250^o/s$ en caso del giróscopo, por lo que será necesario realizar una conversión:

$$a_o = \frac{a_{ideal} - a_{medido}}{8} \tag{2.1}$$

$$g_o = \frac{g_{ideal} - g_{medido}}{4} \tag{2.2}$$

Para evitar posibles ruidos al iniciar el sistema, se pueden realizar 100 medidas con el fin de descartar valores erróneos que se puedan producir. Con la calibración, se consigue corregir el error sistemático de medida y que las posteriores lecturas sean lo más precisas posibles.

Para llevar a cabo el proceso de calibración descrito, se ha desarrollado la función *calibracion()* mostrada a continuación, a la que se llamará al iniciar el sistema o en cualquier momento que se desee realizar una calibración mediante el uso de un pulsador externo.

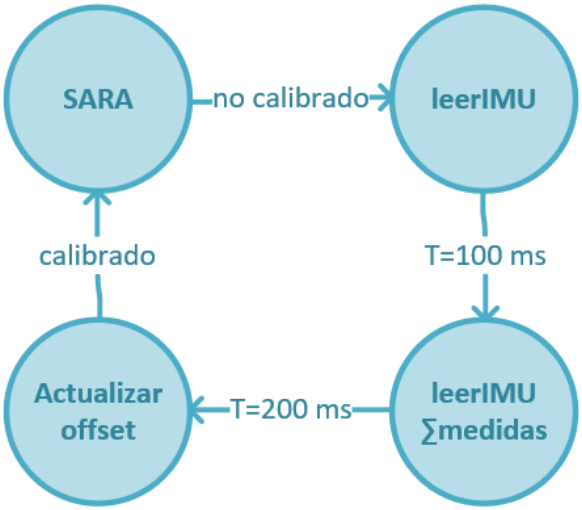

Figura 2.11: Diagrama funcional del proceso de calibración

2.5.5 void leer_IMU()

Los datos del acelerómetro y giróscopo procedentes del módulo se envían al microcontrolador mediante comunicación I2C. Para ello, se crea la función *leer_IMU()* que devuelve las medidas del sensor. Para ello, se deberá comprender cómo se obtienen los datos del acelerómetro y giróscopo.

2.5.5.1 Lectura de los valores del acelerómetro

El módulo dispone de 3 registros de 16 bits que almacenan la última medida de aceleración en los 3 ejes: *ACCEL_XOUT*, *ACCEL_YOUT*, *ACCEL_ZOUT*. A estos registros se accederá con la función *getAcceleration()* mediante el protocolo de comunicaciones I2C. Cada registro de cada eje, a su vez, está formado por 2 registros de 8 bits cada uno, como se puede observar en la figura 2.12:

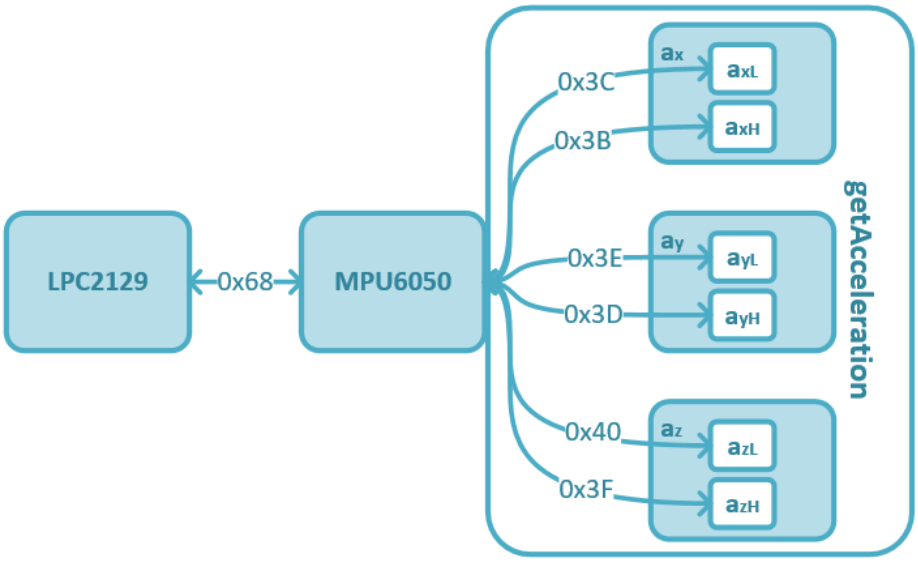

Figura 2.12: Diagrama de acceso a las medidas del acelerómetro

Como se puede observar en la figura anterior, cada medida de cada eje está formada por dos valores de 8 bits almacenados en 2 registros. Posteriormente, habrá que concatenarlos para leer el valor completo de la medida de la siguiente forma:

$$a_X = (a_{XH} \ll 8)|(a_{XL}) \tag{2.3}$$

$$a_Y = (a_{YH} \ll 8)|(a_{YL}) \tag{2.4}$$

$$a_Z = (a_{ZH} \ll 8)|(a_{ZL}) \tag{2.5}$$

Estas medidas codificadas en 16 bits podrán tomar distintos valores en función de la configuración del registro ACCEL_CONFIG (0x1C). Los bits 4 y 3 (AFS_SEL) permiten seleccionar la sensibilidad de las lecturas, pudiendo tomar valores entre ±2g, ±4g, ±8g y ±16g. Con el fin de ser más preciso, se decide que tome valores entre -2g y 2g, configurando estos bits a 0.

Esto implica que un valor de 2^{15} corresponde a 2g, por lo que para obtener la medida real en unidades de aceleración, se aplicará una constante de conversión denominada *accScale*.

$$accScale = \frac{2{,}0 * 9{,}81}{32768{,}0} = [m/s^2] \tag{2.6}$$

El protocolo de I2C se puede hacer de 2 formas: accediendo a cada uno de los registros de forma individual (escritura + lectura en cada uno), o accediendo a los 6 directamente.

En esta última forma, se indica mediante un proceso de escritura el registro inicial (en este caso 0x3B) que se almacena en la variable buffer, y a continuación, mediante el proceso de lectura, se indica cuántos bytes se leerán a continuación. De este modo, al estar los registros seguidos, se leerán 6x8 bits correspondiendo a los 6 registros de los 3 ejes del acelerómetro. Para ello, se emplean las funciones *i2cWrite* e *i2cRead*.

```
1 //Funcion para leer valor de aceleraci n del IMU, del registro 0x3B (ACCEL_XOUT_H)
2 void getAcceleration(tS16* x, tS16* y, tS16* z) {
```

```
3                   tU8  buffer[8];
4           tS8  retCode;
5                   i2cWrite(((0x68<<1)&(0xFE)), 1, 0x3B, buffer, 0); //Acceso registro
                        ACCEL_XOUT_H
6       retCode = i2cRepeatStart();
7       if (retCode == I2C_CODE_OK)
8                           retCode=i2cRead(((0x68<<1)|(0x01)), buffer, 6);//Leer valor
                                del registro ACCEL_XOUT_H
9       *x = (((int)buffer[0]) << 8) | buffer[1]; //Aceleraci n en X
10      *y = (((int)buffer[2]) << 8) | buffer[3];//Aceleraci n en Y
11      *z = (((int)buffer[4]) << 8) | buffer[5];   //Aceleraci n en Z
12  }
```

2.5.5.2 Lectura de los valores del giróscopo

El módulo dispone de 3 registros de 16 bits que contienen la última medida de los valores del giróscopo en sus 3 ejes: *GYRO_XOUT*, *GYRO_YOUT*, *GYRO_ZOUT*. A estos registros se accederá con la función *getRotation()* mediante el protocolo de comunicaciones I2C. De la misma forma que en el acelerómetro, cada eje tiene 2 registros asociados de 8 bits:

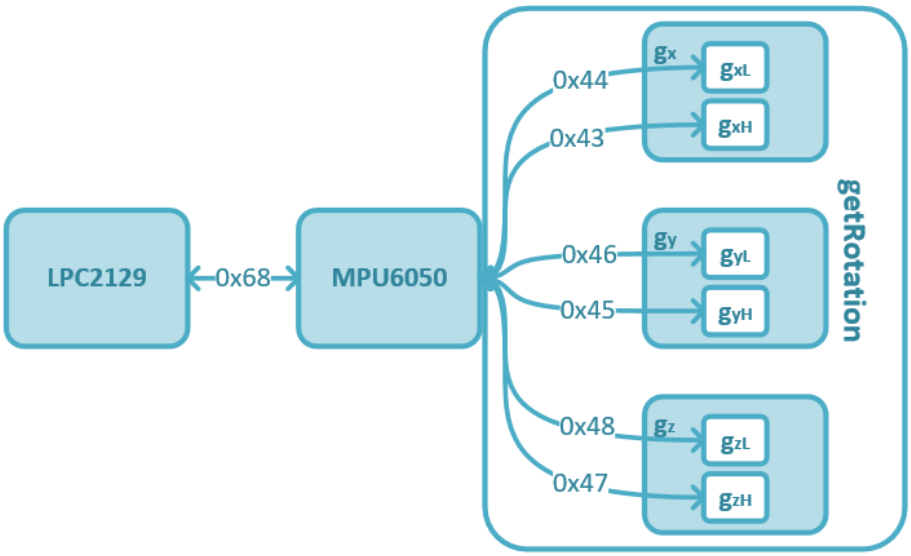

Figura 2.13: Diagrama de acceso a las medidas del giróscopo

Como se puede observar en la figura 2.13, cada medida de cada eje está formada por dos valores de 8 bits almacenados en 2 registros. Posteriormente, habrá que concatenarlos para leer el valor completo de la medida de la siguiente forma:

$$g_X = (g_{XH} \ll 8)|(g_{XL}) \tag{2.7}$$

$$g_Y = (g_{YH} \ll 8)|(g_{YL}) \tag{2.8}$$

$$g_Z = (g_{ZH} \ll 8)|(g_{ZL}) \tag{2.9}$$

Estas medidas en 16 bits podrán tomar distintos valores en función de la configuración del registro GYRO_CONFIG (0x1B). Los bits 4 y 3 (FS_SEL) permiten seleccionar la sensibilidad de las lecturas, pudiendo tomar valores entre $\pm 250^o/s$, $\pm 500^o/s$, $\pm 1000^o/s$ y $\pm 2000^o/s$. Con el fin de ser más preciso, se decide que tome valores entre -250 y 250, configurando los bits 3 y 4 a 0.

Esto implica que un valor de 2^{15} corresponde a $250^o/s$, por lo que para obtener la medida real en unidades de velocidad angular, se aplicará una constante de conversión denominada *gyroScale*:

$$gyroScale = \frac{250}{32768{,}0} = [^o/s] \tag{2.10}$$

De la misma forma que se hizo con el acelerómetro, para leer los datos del giróscopo se accederá a los 6 registros mediante el uso de las funciones *i2cWrite* e *i2cRead*, siendo el registro inicial 0x43.

```
1  //Funcion para leer valor del gir scopo del registro 0x43 (GYRO_XOUT_H)
2  void getRotation(tS16* x, tS16* y, tS16* z) {
3            tU8  buffer[8];
4          tS8  retCode;
5                          i2cWrite(((0x68<<1)&(0xFE)), 1, 0x43, buffer, 0); //Acceso
                               registro GYRO_XOUT_H
6      retCode = i2cRepeatStart();
7      if (retCode == I2C_CODE_OK)
8                          retCode=i2cRead(((0x68<<1)|(1<<0)), buffer, 6);//Leer valor
                               del registro GYRO_XOUT_H
9      *x = (((int)buffer[0]) << 8) | buffer[1];//Rotaci n en X
10     *y = (((int)buffer[2]) << 8) | buffer[3];//Rotaci n en Y
11     *z = (((int)buffer[4]) << 8) | buffer[5];//Rotaci n en Z
12 }
```

2.5.5.3 Conversión de los valores

Las funciones anteriores se emplean para leer de los registros las medidas del acelerómetro y giróscopo del módulo. La función *leer_IMU()* se encarga de convertir los valores a las unidades correspondientes de aceleración y velocidad angular mediante el empleo de las constantes de conversión descritas en los apartados anteriores. Estos valores tienen cifras decimales (tipo float), por lo que para trabajar con ellos en el alto nivel, se convierten a tipo **short** para ser enviados por el bus CAN, cuyos valores mínimos y máximos son -32768 y 32767. Posteriormente, los valores del acelerómetro se convertirán a milésimas y los del giróscopo a centésimas, para no perder precisión en las cifras decimales de las lecturas.

El código de la función es el siguiente, donde se realizan las conversiones para tener las unidades de aceleración y velocidad angular, y el paso a milésimas o centésimas:

```
1  void leer_IMU(short* ax_m, short* ay_m, short* az_m, short* gx_m, short* gy_m, short
       * gz_m){
2                  // Escala de los resultados:
3          const float accScale=2.0*9.81/32768.0; //[m/s^2]
4          const float gyroScale=250.0/32768.0;//[  /s]
5                  getAcceleration(&ax, &ay, &az); //Leer valor aceleracion
6          //Conversion a m/s^2
7                  ax_f=(float)ax*accScale;
```

```
8                   ay_f=(float)ay*accScale;
9                   az_f=(float)az*accScale;
10                  //Obtencion de milesimas de a para enviar por bus CAN:
11                  *ax_m=(short)(ax_f*1000.0);
12                  *ay_m=(short)(ay_f*1000.0);
13                  *az_m=(short)(az_f*1000.0);
14                  getRotation(&gx, &gy, &gz); //Leer valores de rotaci n
15          //Conversion   /s
16                  gx_f=(float)gx*gyroScale;
17                  gy_f=(float)gy*gyroScale;
18                  gz_f=(float)gz*gyroScale;
19                  //Obtencion de centesimas de g para enviar por bus CAN:
20                  *gx_m=(short)(gx_f*100.0);
21                  *gy_m=(short)(gy_f*100.0);
22                  *gz_m=(short)(gz_f*100.0);
23  }
```

2.5.6 void getRotation()

El módulo elegido es capaz de proporcionar únicamente información de la velocidad angular con el giróscopo y de la aceleración lineal con el acelerómetro. La orientación del sistema se puede calcular a partir de la aceleración lineal y la velocidad angular que proporcionan el acelerómetro y el giróscopo.

Existen dos formas de obtener la orientación del sistema a partir de estos valores, y mediante resultados experimentales, se elegirá cuál proporciona la orientación del sistema de forma más precisa.

2.5.6.1 Aplicación de un Filtro Complementario

Aplicar un filtro complementario consiste en combinar las variaciones lentas de las medidas del acelerómetro, aplicando un filtro paso bajo, con las variaciones bruscas de las medidas proporcionadas por el giróscopo, aplicando un filtro paso alto.

En primer lugar, se explica cómo obtener cuánto han rotado los ejes X e Y a partir de la información proporcionada por el acelerómetro:

$$\theta_x = tan^{-1}\frac{a_y}{\sqrt{a_x^2 + a_z^2}} \tag{2.11}$$

$$\theta_y = tan^{-1}\frac{-a_x}{\sqrt{a_y^2 + a_z^2}} \tag{2.12}$$

El giróscopo proporciona información sobre la velocidad angular del sistema. La velocidad angular mide la velocidad de rotación de un cuerpo, obteniéndose a partir de la derivada del ángulo de rotación: ω=dθ/dt, por lo que para obtener el ángulo de rotación en cada eje, basta con integrar la velocidad medida por el giróscopo empleando las siguientes expresiones:

$$\theta_x = \theta_{x0} + \omega_x dt \tag{2.13}$$

$$\theta_y = \theta_{y0} + \omega_y dt \tag{2.14}$$

$$\theta_z = \theta_{z0} + \omega_z dt \tag{2.15}$$

Se aplica un filtro complementario de forma que los ángulos anteriores no acumulen errores: con un filtro paso bajo (B) en la medida del acelerómetro se amortiguan las variaciones bruscas en las medidas de aceleración, y con un filtro paso alto (A) en el giróscopo se eliminan los posibles errores al tener rotaciones rápidas. Se combinan de esta forma, los resultados obtenidos en las ecuaciones 2.11 y 2.13 para el ángulo de rotación en el eje X, y para el ángulo de rotación en el eje Y, los resultados obtenidos en las ecuaciones 2.12 y 2.14, de la siguiente forma:

$$\theta = A(\theta_{prev} + \theta_{gyro}) + B\theta_{acc} \tag{2.16}$$

La suma de las constantes empleadas en la ecuación 2.16 deberá ser igual a 1. Estos valores se irán ajustando de forma manual, tomando la constante A valores cercanos a 0.9.

En el caso de calcular θ_z, debido a que cuando el sistema gire en el eje Z no se existirán cambios en la coordenada Z del acelerómetro (ya que, en el caso nominal, coincide con la componente de la gravedad), únicamente se podrá usar la información del giróscopo, es decir, la tasa de cambio. Se empleará la fórmula 2.15 para su cálculo.

2.5.6.2 Aplicación del Filtro de Madgwick

Conocidas las condiciones iniciales de un sensor, la orientación se puede obtener mediante la integración en el tiempo de la velocidad angular. Los errores en las mediciones del giróscopo, de la misma forma se integrarán, produciéndose un error acumulado a lo largo del tiempo en el cálculo de los ángulos. Es por ello, que se debe incorporar también las medidas del acelerómetro, el cuál se ve afectado por altos niveles de ruido, debidos a variaciones en el movimiento. Es necesario aplicar un filtro a estas medidas con el fin de optimizar la estimación de la orientación.

El filtro de Madgwick [18] realiza la estimación de la orientación del sistema mediante el uso de cuaternios. Los cuaternios son una representación matemática capaces de obtener la rotación de un objeto de forma simple, eficaz y estable.

2.5.7 int impacto()

Pueden existir ocasiones en que la plataforma de navegación no sea capaz de detectar obstáculos bien porque sean de tamaño reducido o se encuentren en un punto ciego, y los sensores de ultrasonidos sean incapaces de detectarlos. El sensor MPU6050 se puede configurar de forma que sea capaz de detectar cuándo se produce un impacto en la plataforma al chocar con un obstáculo que no se ha podido esquivar o que, por las imperfecciones del terreno, produzcan inestabilidad en la navegación del móvil.

En condiciones normales en las que no se produce un impacto ni una caída, la aceleración permanecerá constante o con pequeños incrementos si aumenta la velocidad de la navegación. En caso de que se produzca una caída o un choque, el incremento de la aceleración en cualquiera de los 3 ejes será mucho mayor. Para prevenir estas situaciones anómalas de la navegación se ha de estudiar cómo es el incremento de la aceleración cuando se produce un golpe , que será menor que cuando se produce la caída.

Para ello, se crea una función que detecte los impactos mediante la lectura del registro de interrupciones (0x3A INT_STATUS), cuyo bit 6 indica si se ha producido detección de movimiento brusco. En la función de configuración del MPU6050, en primer lugar se define el umbral a partir del cual se considera que se está produciendo un movimiento brusco (en el registro 0x1F MOT_THR), cuyo valor se configura en incrementos de mg. De forma experimental, se obtiene el valor de 20mg (siendo g unidad gravitacional), para cualquiera de los 3 ejes del acelerómetro. Después, se habilita la interrupción correspondiente a este

umbral (mediante la escritura de 1 en el bit 6 del registro 0x38 INT_ENABLE) como se puede observar en la función de configuración del módulo:

```
24 void configMPU6050(){
25         ...
26         escribo_I2C((0x68<<1)&(0xFE),0x1F,20);//Motion threshold
27         ...
28         escribo_I2C((0x68<<1)&(0xFE),0x38,0x40); //Enable motion detection interrupt
               .
29 }
```

De este modo, cuando se produce una lectura de los valores de aceleración, se llama a la función impacto() que comprobará si se ha superado el umbral considerado como un impacto o golpe en la plataforma:

```
30 int impacto(){
31         int valor;
32         //Leer estado de las interrupciones:
33         valor=leo_I2C((0x68<<1)&(0xFE),0x3A); //Lee valor de un bit.
34         if ((valor&(1<<6))>0) //Se ha producido impacto
35                 return 1;
36         else
37                 return 0;
38 }
```

Con este mecanismo, se puede generar una alarma o aviso cuando se detecte que hay un impacto en la plataforma, pudiendo al menos informar del problema con objeto de intentar que el usuario caiga haciendo que, si el sistema se encuentra en navegación automática, se detenga o se disminuya la velocidad de navegación.

2.6 Código principal

El código principal se encarga de llamar a las funciones de inicialización del sistema: se realiza la configuración del bus CAN, de la comunicación I2C y del módulo MPU6050 (junto con su calibración).

De forma cíclica, se realiza la lectura de datos de los sensores que componen el nodo de sensores. Para ello, se establecen dos turnos de procesamiento en este nodo para la recepción de las medidas y el envío de éstas del bajo nivel al alto nivel por el bus CAN. Cada turno tiene una duración de 100ms y de este modo, se divide el envío de datos de los 9 sensores de ultrasonidos y el de los datos correspondientes al acelerómetro y giróscopo en 2 paquetes.

- Turno 1: se reciben los datos de los sensores de ultrasonidos denominados SDI, STD, SLDD y SLIT.
- Turno 2: se reciben los datos de los sensores de ultrasonidos denominados SLID, SDD, SLDT, STI, SJO.

Las medidas de ultrasonidos se realizan y envían al alto nivel cada 200ms, mientras que los datos procedentes del IMU pueden transmitirse a más frecuencia, pudiéndose intercalar cada 10ms. Para ello,

en el fichero setup() se ha creado un TIMER de 10ms que realiza la lectura de las medidas con esa periodicidad.

Para las distintas pruebas realizadas, se envía la información procedente del acelerómetro, giróscopo y los ángulos obtenidos de éstas por los dos métodos seleccionados (Filtro Complementario y Filtro de Madgwick). Debido a que por cada mensaje CAN únicamente se pueden enviar con 4x16bits, se establecen 4 turnos de 10ms para enviar los 4 mensajes, de forma que no se solapen unos mensajes con otros.

Para las pruebas realizadas, se proponen distintas temporizaciones: la más rápida, donde se procesan los datos del IMU cada 50ms (2.14), y la más lenta, donde se procesan cada 200ms (2.15).

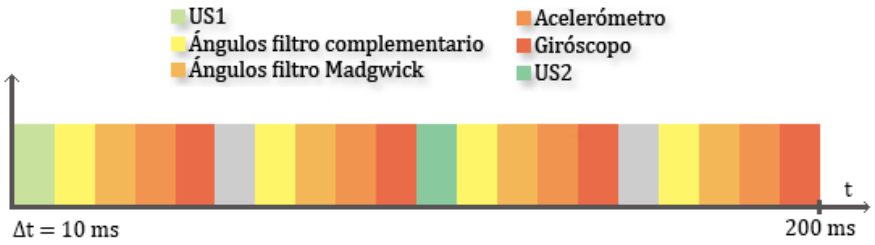

Figura 2.14: Temporización de los mensajes enviados por bus CAN con los datos IMU cada 50ms

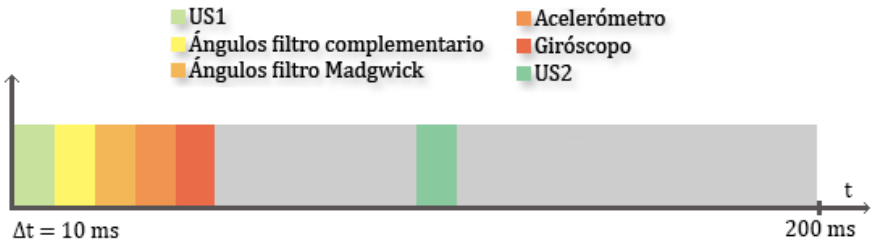

Figura 2.15: Temporización de los mensajes enviados por bus CAN con los datos IMU cada 200ms

Debido a que no se aprecia ninguna diferencia entre obtener las medidas a más velocidad, se selecciona la configuración de la figura 2.15, para que los datos del IMU tengan la misma frecuencia de llegada al alto nivel que los sensores de ultrasonidos, y así la medida del sensor de ultrasonidos ubicado en el Joystick (SJO), que se transmite con las procedentes del acelerómetro, se enviará cada 200ms al igual que el resto.

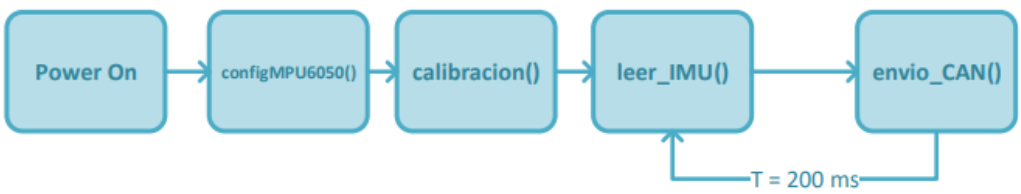

Figura 2.16: Flujograma de envío de las medidas del IMU

Cada mensaje de la trama CAN diseñada para el nodo de sensores de SARA y KATE [2] puede tener una longitud de hasta 64 bits, de forma que cada mensaje esta formado por la medida de 4 sensores diferentes, ya que cada dato leído de la trama tiene una longitud de 16 bits. A cada mensaje, se le ha asignado un identificador del 0x201 al 0x206. Los identificadores 0x201 y 0x202 corresponden a los datos procedentes de los sensores ultrasonidos, reservándose el resto para los procedentes del IMU: 0x203 corresponde a los del acelerómetro, 0x204 del giróscopo, 0x205 a los ángulos calculados mediante el filtro complementario y 0x206 a los ángulos calculados usando el Filtro de Madgwick.

En la tabla 2.6, se exponen los mensajes enviados por bus CAN desde el nodo sensores, incluida la modificación del TFM, correspondientes a cada identificador:

Identificador	Mensaje	Tipo	Tamaño
0x201	SDI	short	16 bits
	STD	short	16 bits
	SLDD	short	16 bits
	SLIT	short	16 bits
0x202	SLID	short	16 bits
	SDD	short	16 bits
	SLDT	short	16 bits
	STI	short	16 bits
0x203	a_z	short	16 bits
	SJO	short	16 bits
	a_x	short	16 bits
	a_y	short	16 bits
0x204	g_z	short	16 bits
	alarma	short	16 bits
	g_x	short	16 bits
	g_y	short	16 bits
0x205	r_x	short	16 bits
	-	short	16 bits
	r_z	short	16 bits
	r_y	short	16 bits
0x206	roll	short	16 bits
	-	short	16 bits
	yaw	short	16 bits
	pitch	short	16 bits

Tabla 2.6: Identificadores y estructura de los sensores para los mensajes CAN

El código *main()*, donde se llaman a las funciones que configuran el sensor MPU6050 y la modificación de la configuración del microcontrolador (bus CAN e I2C) del nodo de sensores, así como los temporizadores necesarios de éste para la ampliación realizada, quedarán de la siguiente manera:

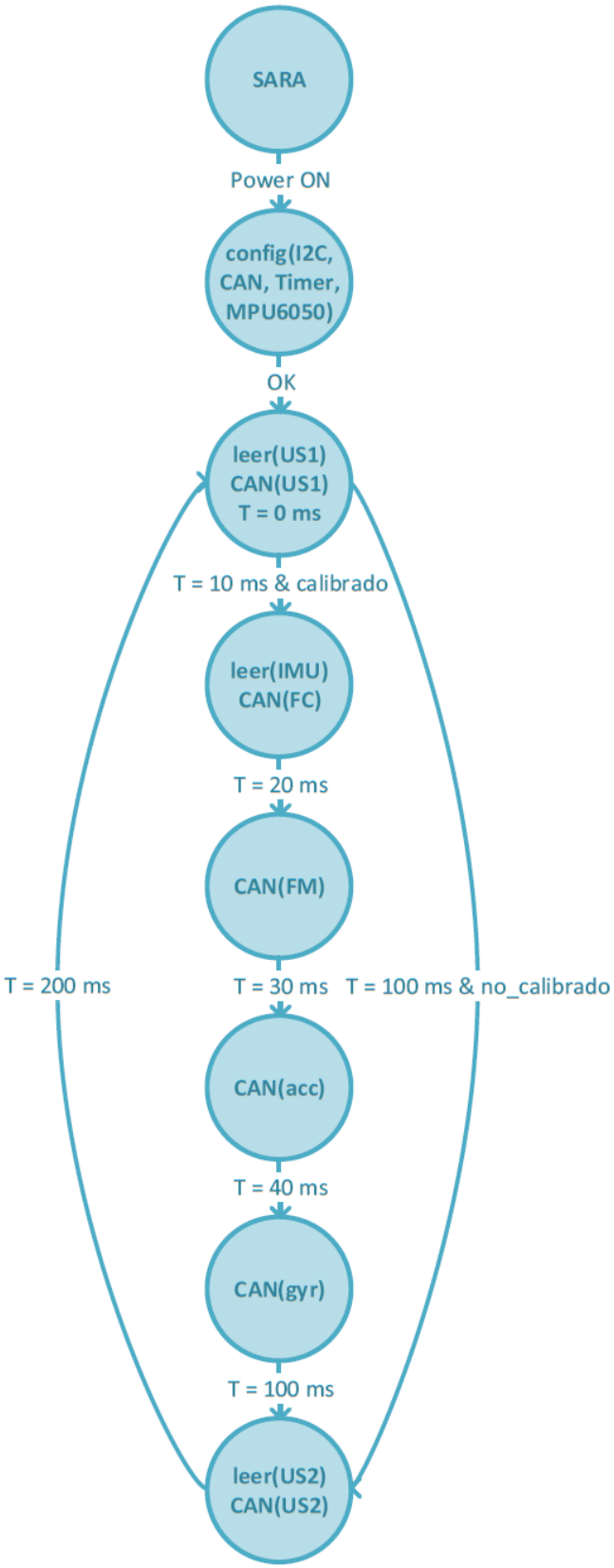

Figura 2.17: Diagrama funcional del código principal

Capítulo 3

Adaptación del alto nivel

3.1 Introducción

Se diseña el sistema para la funcionalidad avanzada, formado por un sistema empotrado con pantalla táctil, con sistema operativo estándar (Linux), y cuyas aplicaciones se basan en librerías de código abierto tipo ROS. Se crea un sistema de navegación autónoma, adaptado para el usuario, a partir de los datos obtenidos por los sensores de bajo nivel (encoders, IMU).

Este sistema diseñado será capaz de obtener un posicionamiento absoluto y relativo, mediante los datos obtenidos por los sensores y recibidos por el bus CAN y de realizar una navegación autónoma o semi-autónoma empleando el paquete "robot_localization" del sistema operativo ROS.

3.2 Tipos de mensajes

Los tipos de datos con los que trabaja el Sistema Operativo Robótico se denominan mensajes. ROS emplea un lenguaje de descripción de mensajes simplificado para describir los diferentes tipos de datos o mensajes que publican los nodos de ROS. Para llevar a cabo el sistema de funcionalidad avanzada es necesario conocer los diferentes tipos de mensajes empleados. En esta sección, se explicarán los mensajes fundamentales para crear el sistema de navegación.

3.2.1 Paquete sensor_msgs

El paquete "*sensor_msgs*" [19] contiene diferentes tipos de mensajes para poder trabajar con todo tipo de sensores. Este paquete será empleado por "*robot_localization*".

Para el sensor inercial, existe el mensaje de tipo **/Imu**, que permite trabajar con la orientación en radianes (en forma de cuaternios), la velocidad angular y la aceleración lineal. Las unidades de la aceleración lineal deberán ser m/s^2, y las de velocidad angular rad/s.

```
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

Figura 3.1: Mensaje tipo del paquete sensor_msgs

Como se puede observar en la figura 3.1, además del mensaje de datos, existen unas matrices de covarianza para cada tipo de mensaje que deberán de ser ajustadas con el fin de obtener más precisión en la obtención del resultado final.

3.2.2 Paquete nav_msgs

El paquete "*nav_msgs*" [20] contiene diferentes tipos de mensajes empleados para la navegación de los robots. Se empleará el mensaje de tipo /Odometry para almacenar la información de la odometría del robot, que almacenará la estimación de la posición y velocidad del robot a partir de la información que proviene de los diferentes sensores, en este caso, de los encoders.

```
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Figura 3.2: Mensaje tipo /Odometry del paquete nav_msgs

Los encoders de los dos motores envían unos pulsos y unos rangos de tiempo por la trama CAN que servirán para calcular las velocidades angulares de las ruedas. Estos valores se almacenan en unos mensajes de tipo "*enc_msg*" creados en trabajos previos. El contenido de estos mensajes se emplea para calcular la posición y velocidad de las ruedas, publicándose en un mensaje de tipo *sensor_msgs/JointState*:

```
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Figura 3.3: Mensaje tipo /JointState del paquete sensor_msgs

A partir de esta información y la consigna de velocidad enviada al controlador *diff_driver_controller*, se calcula la odometría que se fusionará junto con la información del IMU para realizar la estimación y corrección de la velocidad y posición del sistema.

3.3 Paquete canserial

Los datos de los diferentes sensores que componen el sistema se envían mediante comunicación bus CAN desde el bajo nivel al alto nivel. Para ello, se usa un adaptador CAN-USB de forma que se realice una conexión sencilla del sistema con el PC. Para trabajar con los datos, se empleará el paquete de ROS "*canserial*" [21], que permite la comunicación de las tramas CAN con ROS. Este paquete, está formado por los siguientes ficheros que se emplearán en este proyecto:

- canserial.py: se encarga de la recepción y transmisión de los mensajes de las tramas CAN, mediante la creación de dos topics: canrx para los mensajes recibidos del bajo nivel, y cantx para el envío de mensajes al bajo nivel.
- read.py: se encarga de la recepción de los mensajes enviados desde el bajo nivel, mediante la suscripción al topic canrx. Para los mensajes procedentes del IMU, será necesario acceder al id 0x203 (515) y 0x204 (516), 0x205 (517) y 0x206 (518), demultiplexarlos para obtener los valores de cada eje, y posteriormente publicarlos en un topic de tipo Imu denominado */imu/data*. Como no se puede acceder a todos los mensajes a la vez, se intercalan cada 10 ms y en el último se recopilan todas las medidas para publicarlas.

```
elif msg.stdId == 516:
        (alarm,) = struct.unpack('H', msg.data[0:2])
        (gx,) = struct.unpack('h', msg.data[6:8])
        (gy,) = struct.unpack('h', msg.data[4:6])
        (gz,) = struct.unpack('h', msg.data[2:4])


        #imu orientacion con cuaternios

        #calculo de rz:
        self.rotz=(((((gz-self.offset)/self.kgyro)*0.2)*pi/180)+self.thz_ant) # [rad
            /s]
        self.thz_ant=self.rotz
        q = quaternion_from_euler((pi/180)*self.rx/1000, (pi/180)*self.ry/1000, self
            .rotz*1.09)
        (th_x, th_y, th_z) = euler_from_quaternion (q)

        #mensaje tipo Imu:
        data_imu=Imu()
        data_imu.header.frame_id = "base_link" #eje de la silla
        data_imu.header.stamp = rospy.Time.now()


        data_imu.linear_acceleration.x=self.ejex / self.kacc #[m/s^2]
        data_imu.linear_acceleration.y=self.ejey / self.kacc #[m/s^2]
        data_imu.linear_acceleration.z=self.ejez / self.kacc #[m/s^2]

        data_imu.angular_velocity.x=(3.14/180)*gx / self.kgyro #[deg/s]to[rad/s]
        data_imu.angular_velocity.y=(3.14/180)*gy / self.kgyro #[deg/s]to[rad/s]
        data_imu.angular_velocity.z=(3.14/180)*gz / self.kgyro #[deg/s]to[rad/s]

        data_imu.orientation.x=q[0]
        data_imu.orientation.y=q[1]
        data_imu.orientation.z=q[2]
        data_imu.orientation.w=q[3]


        #Covariance matrix
        data_imu.linear_acceleration_covariance[0] = 0.0032
```

```
38          data_imu.linear_acceleration_covariance[4] = 0.0011
39          data_imu.linear_acceleration_covariance[8] = 0.0020
40          data_imu.angular_velocity_covariance[0] = 0.0084
41          data_imu.angular_velocity_covariance[4] = 0.0091
42          data_imu.angular_velocity_covariance[8] = 0.0051
43          data_imu.orientation_covariance[0] = 0.001
44          data_imu.orientation_covariance[4] = 0.001
45          data_imu.orientation_covariance[8] = 0.0098
46
47          self.pubImu.publish(data_imu)
```

3.4 Paquete robot_localization

El paquete "*robot_localization*"[7] de ROS permite la estimación de la posición de un robot en movimiento mediante la fusión de la información procedente de distintos tipos de sensores. Con la combinación de estos datos, se espera obtener una mejor estimación de la posición del sistema frente a usar los datos de forma individual, empleando las mejores propiedades de cada sensor. En las figuras 3.5, 3.6 y 3.7 procedentes de la documentación del paquete "*robot_localization*", se compara cómo al fusionar las medidas de diversos sensores con los datos de la odometría, se mejora la estimación de la posición respecto a utilizar únicamente los datos procedentes de la odometría (3.4).

Figura 3.4: Estimación de la posición empleando odometría [7]

Figura 3.5: Estimación de la posición empleando odometría y un IMU [7]

Figura 3.6: Estimación de la posición empleando odometría y dos IMU [7]

Figura 3.7: Estimación de la posición empleando odometría, dos IMU y un GPS [7]

El paquete contiene 2 nodos de estimación: *ekf_localization_node* y *ukf_localization_node*, basados en dos extensiones del filtro de Kalman para sistemas no lineales: Extended Kalman Filter y Unscented Kalman Filter.

- ekf_localization_node

  Implementa el filtro de Kalman extendido, que utiliza un modelo de movimiento omnidireccional para obtener el estado en el tiempo, corrigiendo la estimación con los datos proporcionados por los sensores. El EKF linealiza alrededor de una estimación actual, de forma que se obtiene una aproximación. Si el modelo es severamente no lineal, se obtendría unas covarianzas muy distintas a las del modelo real, de forma que se obtiene un filtro muy pobre.

- ukf_localization_node

  Implementa el filtro de Kalman 'unscented'. El UKF resulta de incorporar una transformación unscented al EKF para mejorar las aproximaciones. Esto consiste en la propagación de una variable aleatoria mediante una transformación no lineal. Los cálculos son más sencillos lo que hace que el filtro sea más estable, por lo que es ampliamente utilizado.

Estos nodos de estimación de estado, son capaces de predecir el estado en 15 dimensiones, ya que admiten mensajes de odometría, IMU, GPS:

$$(X, Y, Z, roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z}).$$

Figura 3.8: Predicción de estados

Uno de los usos más empleados es el de fusionar los datos de la odometría (calculada a partir de la información procedente de los encoders de las ruedas) con los del sensor inercial (IMU).

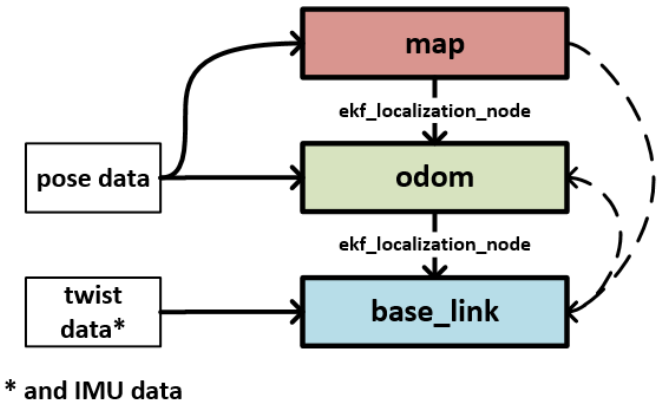

Figura 3.9: Ejemplo de uso de robot_localization

La odometría consiste en la estimación de la posición de un sistema móvil mediante la información proporcionada sobre la rotación de las ruedas. Las ruedas del sistema son accionadas mediante unos motores que tienen acoplados unos encoders. Un encoder es un transductor rotativo que sirve para indicar la posición, velocidad y aceleración angulares del eje del motor empleando una señal eléctrica. Esto permite obtener cuánto ha girado la rueda del sistema y conociendo su geometría, permitirá calcular la velocidad de dichas ruedas, pudiendo obtener así cuánto se ha desplazado en los ejes X e Y.

La Unidad de Medición Inercial proporciona información sobre la aceleración en los 3 ejes y la velocidad angular, lo que permitirá calcular a partir de ellas el ángulo de rotación de cada eje.

Es por tanto, que la odometría proporciona información sobre la posición en X e Y y a partir de éstas, la velocidad angular de guiñada (yaw o θ_z), mientras que el IMU proporciona información sobre los 3 ángulos de rotación de euler, la velocidad angular y la aceleración en los ejes X, Y, Z.

El paquete de ROS "*robot_localization*" permite hacer la fusión de los diferentes sensores que tenga instalado el sistema de forma sencilla mediante el uso de unas matrices de configuración, en las que se especifica qué variables de cada sensor serán consideradas para el cálculo de las estimaciones:

```
<rosparam param="odom0_config">
      [true, true, false,     x, y, z
      false, false, false,    θx, θy, θz
      false, false, false,    vx, vy, vz
      false, false, true,     ωx, ωy, ωz
      false, false, false]    ax, ay, az
</rosparam>
<rosparam param="imu0_config">
      [false, false, false,   x, y, z
      true, true, true,       θx, θy, θz
      false, false, false,    vx, vy, vz
      true, true, true,       ωx, ωy, ωz
      true, true, true]       ax, ay, az
</rosparam>
```

Figura 3.10: Ejemplo de matrices de configuración de la odometría y de un IMU

Para utilizar este paquete ROS, es necesario crear un fichero de configuración de tipo ".*yaml*", donde se deberán especificar los parámetros de configuración para realizar la fusión con los sensores del robot.

Para especificar los sensores que se fusionarán, se deberá crear una variable de nombre *sensor* que indicará el topic al que debe suscribirse el nodo, donde se publicarán las medidas procedentes directamente del sensor. A continuación, se crean unas matrices de configuración *sensor_config* de dimensión 5x3 (3.10), donde se especificará qué estados de cada sensor se fusionarán en el filtro, marcándolos con el valor *true*. Con el parámetro *sensor_queue_size* se especifica el tamaño de las medidas que se fusionarán cuando se envíen a altas frecuencias. El parámetro *sensor_differential* indicará si dos sensores envían medidas sobre un mismo estado, y el parámetro *sensor_relative* indicará si se quiere que dicho sensor comience sus mediciones en 0.

A continuación, se expone un ejemplo de configuración, utilizando un nodo EFK, para la plataforma SARA en un entorno 2D, cuyas medidas a fusionar serán la velocidad angular en Z y el ángulo de rotación en Z del IMU, junto con la posición en XY y velocidad lineal en X obtenidos de la odometría calculados a partir de los encoders de las ruedas.

```
39 #Configuation for robot odometry EKF
40 #
41 frequency: 50 # Same as controller
42
43 two_d_mode: true   #2D
44
45 publish_tf: true
46
47 transform_time_offset: 0.05
48
49 # Whether to publish the acceleration state. Defaults to false if unspecified.
50 publish_acceleration: true
51
52 # Complete the frames section
53 odom_frame: odom
54 base_link_frame: base_link
55 world_frame: map
56 map_frame: map
57
58 # Complete the odom0 configuration: x, y, vx
59 odom0: /diff_drive_controller/odom
60 odom0_config: [true, true, false,
61  false, false, false,
62  true, false,  false,
63  false, false, false,
64  false, false, false]
65 odom0_differential: false
66
67 odom0_queue_size: 20
68
69 # Complete the imu0 configuration: yaw, wz
70 imu0: /imu/data
71 imu0_config: [false, false, false,
72  false, false, true,
```

```
73    false,  false,  false,
74    false, false, true,
75    false, false, false]
76  imu0_differential: false
77
78  imu0_nodelay: false
79  imu0_relative: false
80  imu0_queue_size: 20
81
82  imu0_remove_gravitational_acceleration: true
```

La fusión de los datos se realizará a una frecuencia de 50 Hz, y se publicará la odometría resultante en un topic denominado */odometry/filtered*.

3.5 Sistema de navegación

En esta sección, se detallan los paquetes ROS necesarios para llevar a cabo el sistema de navegación autónoma, y la relación existente entre ellos. Será necesario emplear un controlador, un navegador y un localizador.

3.5.1 diff_drive_controller y hardware_interface

El paquete ROS "*diff_drive_controller*" es un controlador para robots de ruedas diferenciales. Forma parte del sistema de paquetes de ros_control [22], y es capaz de obtener la cinemática inversa de un robot a partir de una consigna de velocidad y la definición de los parámetros dinámicos y geométricos de la plataforma.

- Topics suscritos:
  - **cmd_vel** (geometry_msgs/Twist): El controlador, para calcular la odometría del sistema, se suscribe a las consignas de velocidad (tanto lineal como angular) enviadas por el sistema de navegación.
  - **joint_states** (): a partir de la interfaz del hardware, se proporciona el estado actual de las ruedas.
- Topics publicados:
  - **odom** (nav_msgs/Odometry): tras aplicar la cinemática inversa y el dead reckonig, se crea un topic de odometría para publicar la velocidad y la posición actual de la plataforma, que se enviará al sistema de navegación. El topic se publicará bajo el nombre de */diff_drive_controller/odom*.
  - **/tf** (tf/tfMessage): transformación entre los frames base_link (plataforma) y odom.
  - **/joint_commands**: son los comandos de velocidad obtenidos del controlador, que son posteriormente enviados a la plataforma a través del hardware interface.

3.5.2 move_base

El paquete ROS "*move_base*" [23] permite mover a un robot a una posición deseada usando el stack de navegación [24] de ROS. Mediante la configuración del robot, éste intentará llegar a una posición objetivo dentro de unas tolerancias (en posición XY y ángulo de rotación *yaw* o θ_z) fijadas en caso de que no haya ningún obstáculo que se interponga en su meta.

- Topics suscritos:
  - **move_base_simple/goal** (geometry_msgs / PoseStamped): es el objetivo o posición final que debe alcanzar el robot.
- Topics publicados:
  - **cmd_vel** (geometry_msgs / Twist): se crea una consigna de velocidad basada en las propiedades del robot, que se enviará al controlador. Este nodo realiza su tarea de navegación global vinculando un planificador global con uno local, con dos mapas de costos (uno para el planificador global y otro para local).

Además de los topics publicados y suscritos, son necesarios unos archivos de configuración que definirán el comportamiento del robot (en este caso la plataforma SARA) y del entorno en el que navega. Estos ficheros son de tipo *.yaml*.

- **base_local_planner_params**: Se definen los parámetros de comportamiento dinámico de la plataforma robotizada. Se definen los valores límites máximos y mínimos de velocidad tanto lineal, como angular (en avance y en el sitio) y de aceleraciones lineales y angulares. Estos valores deben encontrarse dentro del rango de movimiento de la plataforma, y la consigna de velocidad para alcanzar la meta se calculará en función de ellos. También existen unos parámetros para considerar que se ha alcanzado el objetivo, mediante la definición de unas tolerancias: xy_goal_tolerance es la distancia máxima en metros de la plataforma al objetivo, y yaw_goal_tolerancia es el ángulo de rotación en Z máximo del robot en su objetivo.
- **global_costmap_params** y **local_costmap_params**: para que el robot navegue, es necesario que se proporcione un mapa de ocupación, tanto para el marco global (*map*) como para el local (*odom*). Permite configurar la frecuencia de actualización de cada uno de los mapas y las capas de cada uno. En el mapa local, se añade la capa *ultrasonic* que irá generando los obstáculos que detecten los sensores de ultrasonidos.
- **costmap_common_params**: es el fichero de configuración común de los mapas de costos. En el se definen las dimensiones del robot (*footprint*), y las distintas capas que componen los mapas, donde se definirán los parámetros para la detección de obstáculos.

3.5.3 Incorporación de robot_localization

En el sistema de navegación previo empleaba el paquete "*fake_localization*" [25], como sustituto de un sistema de localización real. Este paquete se suscribía al topic de odometría para comunicarse con el paquete ROS "*move_base*", indicándole la posición actual del robot mediante una simulación.

Este sistema se sustituye por el paquete ROS "*robot_localization*", un sistema de localización formado por distintos nodos de estimación de la posición relativa de un robot. Este paquete, se suscribe a las medidas del IMU enviadas desde el bajo nivel por bus CAN y a la odometría calculada por el controlador

a partir de la información enviada por los encoders desde el bajo nivel, como se explicó en el apartado 3.4. La odometría resultante de aplicar el filtro de Kalman extendido, se enviará al navegador, para que a partir de ella, se calculen las consignas de velocidad para los motores de la plataforma.

La transformación entre global_frame y odom_frame se realizaba empleando el nodo de localización "*fake_localization*", por lo que en el nuevo sistema se realizará empleando el paquete "*robot_localization*", cuyo nodo se denomina "*/ekf_se*".

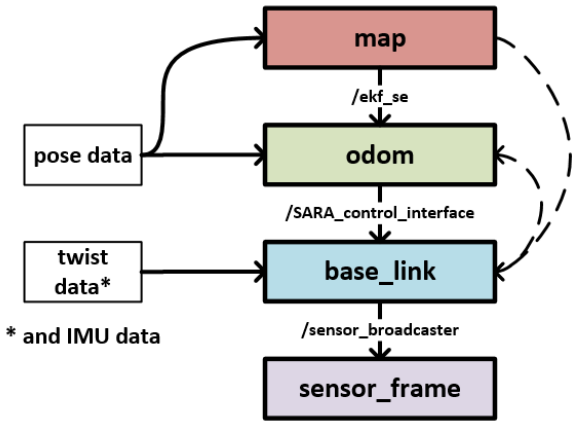

Figura 3.11: Transformaciones entre los distintos frames

Para realizar dicha transformación, es necesario configurar los parámetros del archivo de configuración del nodo EKF, donde se especifica que se realizará la transformación sobre el topic /tf (*publish_tf* = *true*), el offset de la transformación, ya que *robot_localization* se comunica con otros paquetes. También se especifican cuáles son los frames o marcos del robot (*base_link*), de la odometría (*odom*), y el global, que en este caso será *map*.

En el diagrama 3.12, se representa la relación existente entre los nodos *move_base*, *sara_control*, *robot_localization*, que forman el sistema de navegación en el alto nivel, y cómo dichos nodos se comunican mediante el bus CAN a través del paquete *canserial* con la plataforma SARA:

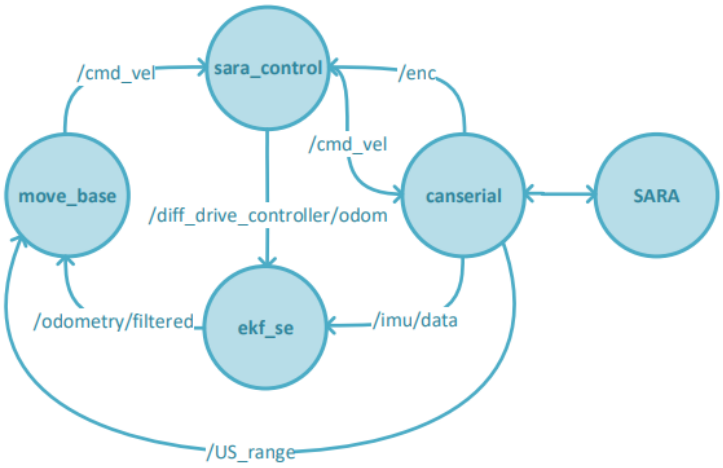

Figura 3.12: Relación de los nodos del sistema de navegación

3.6 Navegación

Una vez creado el sistema de navegación y establecida la relación entre los distintos nodos que forman el sistema, se procede a realizar la navegación de la plataforma.

Al ser un robot diferencial, la plataforma navegará en un entorno 2D, de forma que el objetivo que tendrá que alcanzar será de la forma (X,Y,θ_z).

Para ello, será necesario establecer dos parámetros:

- **Posición inicial**: para navegar por un entorno 2D, el planificador de trayectorias deberá conocer la posición inicial de la plataforma sobre la que se calculará la trayectoria. Esta posición inicial, viene dada por el nodo *ekf_se*, que es el encargado de publicar la posición que va tomando la plataforma en su navegación, procedente de fusionar la odometría con la información del IMU. Por defecto, el filtro se inicializa en la posición (X_0,Y_0,θ_{z0}) = (0, 0, 0). El nodo de estimación puede inicializarse con los valores deseados de dos formas:
  1. ~*initial_state*: es un parámetro que sirve para inicializar el filtro en un determinado estado, inicializado en el fichero de configuración .yaml del nodo. Es un vector en 15D, donde se pueden inicializar los 15 estados del filtro (3.8).
  2. */set_pose*: es un topic de tipo geometry_msgs / PoseWithCovarianceStamped [26] publicado a través del rosservice SetPose. Permite modificar la posición sobre la que se inicializa la trayectoria en cualquier momento de la navegación. Además, es compatible con la herramienta de visualización 3D RViz [27].
- **Posición final**: la posición objetivo se recibe el el nodo *move_base* a través del topic move_base/goal, de tipo PoseStamped, y a partir de ella y la posición actual durante la navegación, el planificador calcula la trayectoria.

3.6.1 Navegación por script

Cuando no se posee un mapa por donde la plataforma pueda navegar, la forma más sencilla para realizar una trayectoria es mediante la creación de un script .py. Este método es útil cuando se quiere comprobar con qué precisión la plataforma alcanza los objetivos. Para ello, en el script se especifica la posición X e Y (ya que es diferencial), y el cuaternio correspondiente a la orientación que tomará la plataforma.

```
84  # Creates a new goal with the MoveBaseGoal constructor
85      goal = MoveBaseGoal()
86      goal.target_pose.header.frame_id = "map"
87      goal.target_pose.header.stamp = rospy.Time.now()
88
89      goal.target_pose.pose.position.x = 4
90      goal.target_pose.pose.position.y = 1
91
92      quat = tf.transformations.quaternion_from_euler(0, 0, pi/2)
93      goal.target_pose.pose.orientation.x = quat[0]
94      goal.target_pose.pose.orientation.y = quat[1]
95      goal.target_pose.pose.orientation.z = quat[2]
96      goal.target_pose.pose.orientation.w = quat[3]
97      # Sends the goal to the action server.
98      client.send_goal(goal)
```

3.6.2 Navegación utilizando RViz

RViz es una herramienta de visualización 3D para aplicaciones ROS, que permite visualizar el modelo de un robot, la información de sus sensores en tiempo real y realizar simulaciones de navegación.

La interfaz posee dos botones que permiten fijar los topics de la posición inicial ("2D Pose Estimate") y la posición final ("2D Nav Goal") cuando un robot, en este caso la plataforma SARA, se encuentra situado en un mapa conocido.

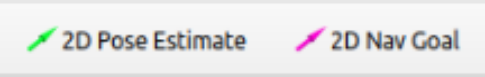

Figura 3.13: Botones de la interfaz RViz para la navegación

En la figura 3.14, se observa un ejemplo de navegación empleando el entorno RViz, donde se observa el modelo 3D de SARA en un mapa, la información de los sensores de ultrasonidos, el mapa de obstáculos inflados, el objetivo (goal) a alcanzar y la trayectoria planificada.

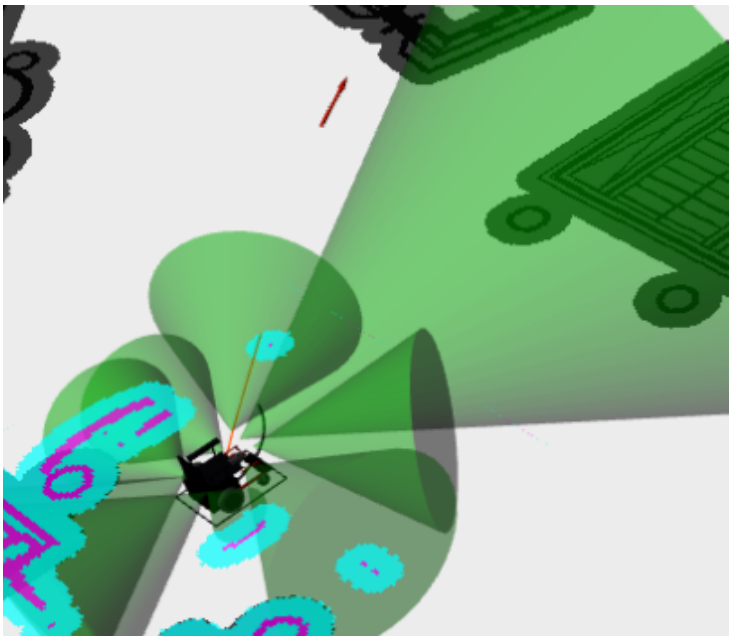

Figura 3.14: Ejemplo de navegación de SARA en RViz

Para comunicar la orientación inicial de la plataforma con el valor de RViz, existen dos opciones:

1. Fijar el parámetro '*imu0_differential*' a true, de forma que fusione la información de RViz con la procedente del IMU para calcular θ_z.

2. Actualizar el offset del ángulo θ_z en el nodo lectura (3.3) y fijar el parámetro 'imu0_differential' a false, de forma que el filtro sólo emplee la medida del IMU para la estimación de la orientación.

   Para ello, el nodo se suscribe al topic /set_pose y se crea una función de tipo *callback*, de forma que cada vez que se actualice la posición inicial se modifique el offset del valor del ángulo de orientación de la plataforma.

   La función para actualizar el offset quedaría de la siguiente forma:

```
99  def update_initial_pose(self, sara_goal):
100 
101         q1 = sara_goal.pose.pose.orientation.x # q1
102         q2 = sara_goal.pose.pose.orientation.y # q2
103         q3 = sara_goal.pose.pose.orientation.z # q3
104         q4 = sara_goal.pose.pose.orientation.w # q4
105 
106         quaternion=(q1, q2, q3, q4)
107 
108         # Yaw from quaternion:
109 
110         (e1,e2,self.thz_ant) = tf.transformations.euler_from_quaternion(
                quaternion)
111 
112         # Update yaw offset MPU6050:
113 
114         rospy.loginfo("initial_state received")
```

Capítulo 4

Resultados

4.1 Introducción

En este capítulo, se exponen los resultados obtenidos al incorporar el IMU en SARA, así como las modificaciones que se han realizado para obtener los resultados deseados.

4.2 Calibración del MPU6050

Un error sistemático de medida es aquel que se produce en todas las condiciones de medida y con la misma magnitud debido a un defecto en el diseño del sistema de medida o en su modelado. Un mal posicionamiento del sensor MPU6050 en la plataforma, puede producir un error sistemático en las medidas si el sensor se encuentra inclinado. Es por esto, que al iniciar el sistema o de forma periódica es necesario realizar una calibración del sensor con el fin de eliminar este error. Para ello, se analizará si en el estado de reposo, se produce un error con un patrón de repetición entre las medidas. Se modela el error en las medidas del módulo MPU6050, con el fin de eliminar el error sistemático para realizar posterior fusión de sensores para la estimación del posicionamiento de la plataforma.

4.2.1 Calibración del acelerómetro

Para que el acelerómetro esté calibrado, con el sistema en reposo, debería medir 0 m/s^2 en sus ejes X e Y, y 9,8 m/s^2 en el eje Z. Debido a una mala colocación del módulo en la plataforma, estos valores podrían ser diferentes, obteniendo posibles falsas medidas y produciendo un error sistemático.

Con el fin de evitar falsas medidas, se realiza un ensayo donde se obtienen medidas durante 1 minuto con el sistema en reposo, para comprobar si el sistema se encuentra inicialmente calibrado. Estas medidas se obtienen desde el sistema de alto nivel, obtenidas con ROS, se procesan en Matlab, y se representan en la figura 4.1:

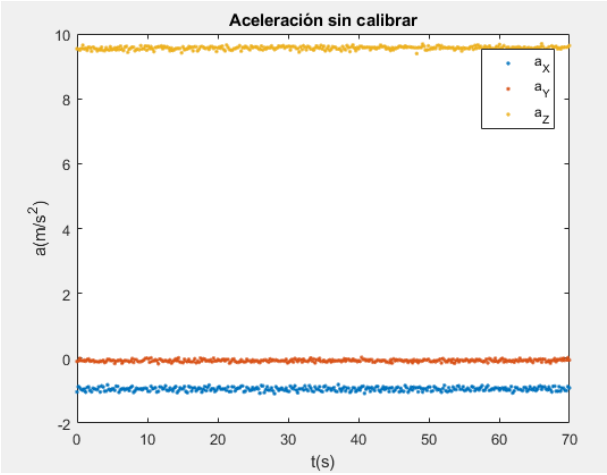

Figura 4.1: Medidas en reposo del acelerómetro antes de la calibración

Con un total de 340 muestras, se observa que las medidas en el eje X tienen un valor medio de = -0.9498 m/s^2, lo que supone un error sistemático de aproximadamente 1 m/s^2. Las medidas del eje Y tienen un error menor, con una media de -0.0752 m/s^2, siendo un valor más cercano a 0. En el eje Z, se obtiene una media de 9.5638 m/s^2, presentando un error de de 0.3 m/s^2 respecto a los 9.8 m/s^2 esperados del valor de la gravedad. En la tabla 4.1, se recogen el valor medio (μ) y la desviación estándar (σ) de cada variable, que representa la dispersión de las medidas respecto al valor medio.

Variable	μ (m/s^2)	σ
a_x	-0.9498	0.0576
a_y	-0.0752	0.0365
a_z	9.5638	0.0476

Tabla 4.1: Valor medio y desviación estándar de las medidas del acelerómetro sin calibrar

Se calculan los errores aleatorios de cada medida, a partir de la eliminación del error sistemático y se realiza un histograma con los resultados para ver si siguen una distribución normal, obteniendo las siguientes figuras (4.2):

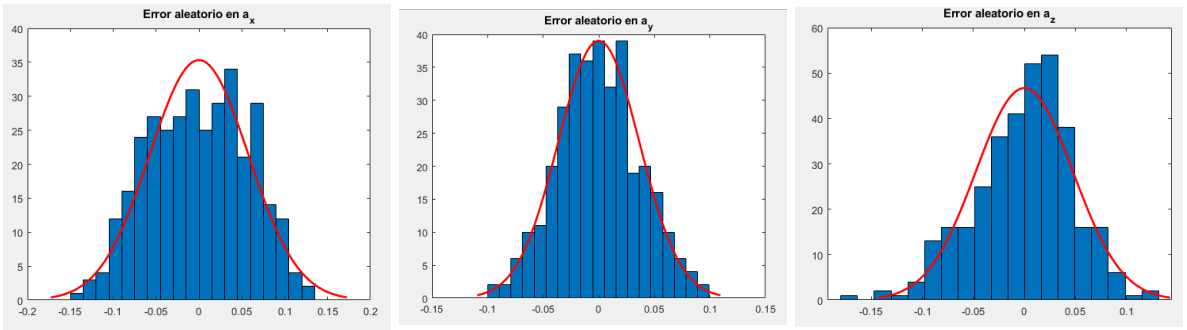

Figura 4.2: Histogramas de las medidas del acelerómetro en los ejes X, Y, Z antes de la calibración

En estos histogramas se puede ver que las medidas intentan seguir la distribución normal, ya que su forma está prácticamente dentro de la curva de Gauss. Se puede observar que la aceleración medida en el eje Y es la que más se ajusta a una distribución normal. En el eje Z, la gráfica se presenta ligeramente desplazada, debido a que las medidas presentan un error medio de +0.3 m/s^2.

Se procede por tanto, a realizar la calibración del sistema, accediendo desde el bajo nivel a los registros del módulo MPU6050 correspondientes que configuran el offset.

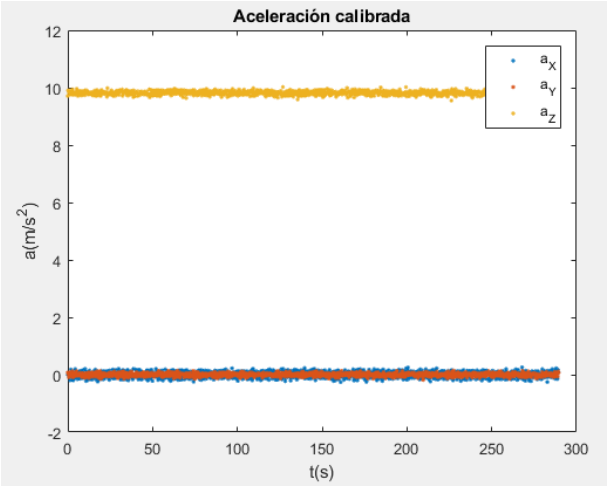

Figura 4.3: Medidas en reposo del acelerómetro después de la calibración

Realizada la reconfiguración del sistema de calibración, se toman de nuevo medidas, con el fin de observar si se ha corregido el error sistemático. En el eje X, que presentaba mayor error, se obtiene ahora una media de -0.0035 m/s^2, bastante menor que el error de 1 m/s^2 que había sin calibrar. En el caso del eje Y, que presentaba el menor error, se obtiene una media de -0.00032 m/s^2. En el eje Z, se obtiene una media de 9.8166 m/s^2. Aproximadamente, todas las medidas de los 3 ejes presentan un error $\leq 0{,}1m/s^2$, bastante menor que al tomar las medidas sin calibrar.

En la tabla 4.2, se representa el valor medio y la desviación estándar de las medidas tras realizar la calibración:

Variable	μ (m/s^2)	σ
a_x	-0.0035	0.1097
a_y	-0.00032	0.0594
a_z	9.8166	0.0628

Tabla 4.2: Valor medio y desviación estándar de las medidas del acelerómetro calibradas

A continuación (4.4), se representan de nuevo los histogramas de las medidas del acelerómetro en cada eje tras realizar la calibración:

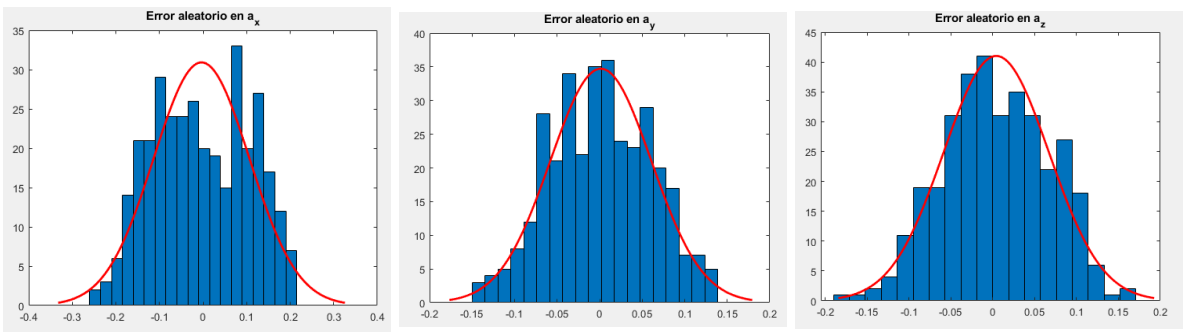

Figura 4.4: Histogramas de las medidas del acelerómetro en los ejes X, Y, Z después de la calibración

En los histogramas anteriores, se observa que en los 3 ejes sigue una distribución normal. La mayoría de las muestras tienen un error cercano a 0, ya que la mayoría de las muestras se encuentran dentro de la curva de Gauss.

A continuación, se representan en la tabla 4.3 los valores medios de cada medida cuando el sistema se encuentra en reposo, tanto cuando el sistema se encuentra sin calibrar y calibrado:

Valor medio	Sin calibrar	Calibrado
a_x	-0.9498	-0.0035
a_y	-0.0752	-0.00032
a_z	9.5638º	9.8166

Tabla 4.3: Comparativa valor medio de la aceleración sin calibrar y calibrado

4.2.2 Calibración del giróscopo

De la misma forma que se ha realizado en el acelerómetro, se realiza un ensayo con la plataforma en reposo con el fin de observar qué valores toman las medidas del giróscopo. En reposo, se espera que la medida del giróscopo en los 3 ejes sea de 0º/s, obteniendo los resultados de la figura 4.5.

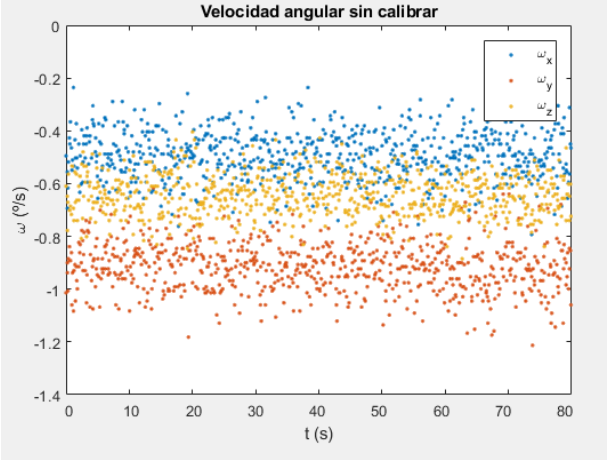

Figura 4.5: Medidas en reposo del giróscopo antes de la calibración

Con un total de 323 muestras del ensayo, se observa que las medidas en el eje X tienen un valor medio de -0.4924 º/s, las del eje Y tienen una media de -0.9131 º/s y en el eje Z una media de -0.6456 º/s. A diferencia del acelerómetro, se observa un error medio menor, ya que las medidas están cerca de 0, sin embargo, se observa un ruido considerable.

Se representa el valor medio y la desviación estándar de las medidas del giróscopo sin calibrar en la tabla 4.4:

Variable	μ (º/s)	σ
ω_x	-0.4924	0.0908
ω_y	-0.9131	0.0946
ω_z	-0.6456	0.0779

Tabla 4.4: Valor medio y desviación estándar de las medidas del giróscopo sin calibrar

A continuación, se obtiene el histograma con las medidas de cada eje, con el fin de observar si siguen una distribución normal.

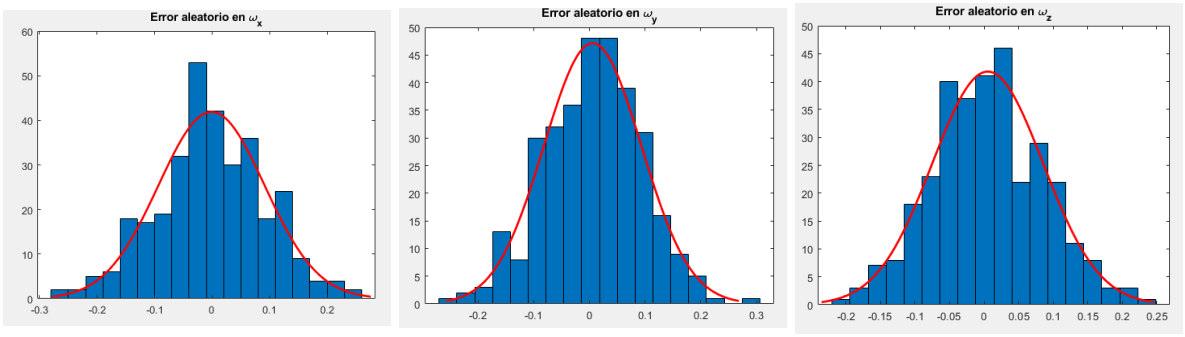

Figura 4.6: Histogramas de las medidas del giróscopo en los ejes X, Y, Z antes de la calibración

Al igual que con el acelerómetro, se realiza la calibración del giróscopo eliminando el error sistemático. Tras tomar medidas durante aproximadamente 1 minuto, se obtiene la siguiente gráfica (4.7):

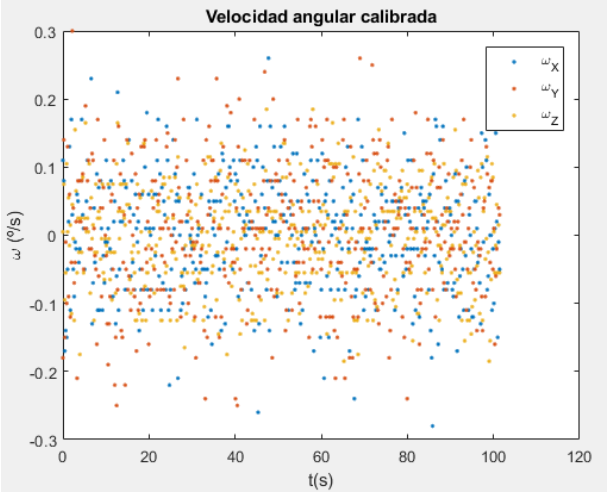

Figura 4.7: Medidas en reposo del giróscopo después de la calibración

Se obtiene una media de -0.004 º/s en el eje X, en el eje Y -0.0049 y en el eje Z -0.0026 º/s. Se observa que el error se ha reducido respecto al caso sin realizar la calibración. El error existente es reducido, pero puede ser lo suficientemente significativo como para afectar al cálculo de los ángulos de rotación.

En la tabla 4.5, se representa el valor medio y la desviación estándar de las medidas del giróscopo calibradas:

Variable	μ (º/s)	σ
ω_x	-0.004	0.0841
ω_y	-0.0049	0.0971
ω_z	-0.0026	0.0734

Tabla 4.5: Valor medio y desviación estándar de las medidas del giróscopo calibradas

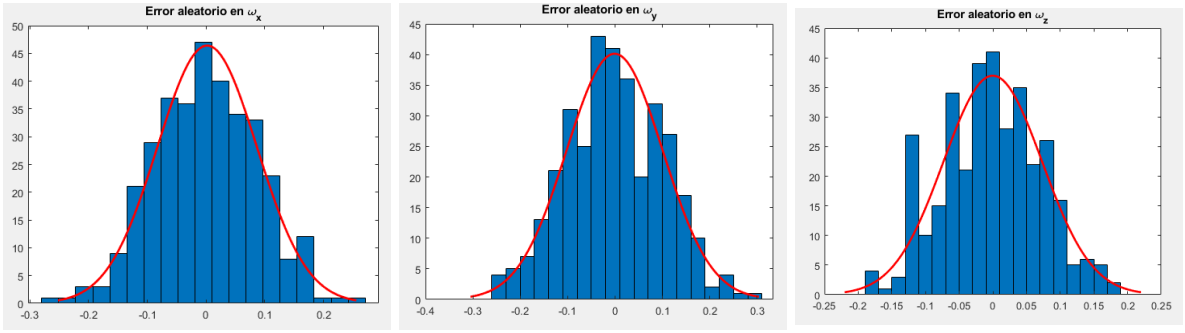

Figura 4.8: Histogramas de las medidas del giróscopo en los ejes X, Y, Z después de la calibración

Valor medio	Sin calibrar	Calibrado
ω_x	-0.0145	0.004
ω_y	-0.0155	0.0049
ω_z	-0.0108	-0.0026

Tabla 4.6: Comparativa valor medio de la velocidad angular sin calibrar y calibrado

4.2.3 Obtención de la rotación de la plataforma

Para obtener la orientación de la plataforma, en primer lugar es necesario tener calibrados el acelerómetro y el giróscopo del módulo, ya que los valores que éstos proporcionan serán empleados para calcular sus ángulos de rotación.

Uno de los errores más comunes que se producen al emplear este tipo de sensores es el llamado '*drift*' o deriva del giróscopo, producido por la acumulación de errores sistemáticos en su medida de la velocidad angular. Este error a lo largo del tiempo supone un problema, ya que crece de forma lineal con el tiempo, y en condiciones de reposo puede modificar la medida del ángulo de posicionamiento de la plataforma.

Existen dos formas de obtener estos ángulos evitando el drift: aplicando un Filtro Complementario o un Filtro de Madgwick a las medidas de la aceleración lineal y velocidad angular. En esta sección, se compararán los resultados obtenidos empleando los dos métodos mediante la realización de distintos ensayos, y se seleccionará el método que aporte resultados más estables y fiables.

4.2.3.1 Sistema en reposo

Una vez calibrados tanto el acelerómetro como el giróscopo del IMU, se procede a calcular la orientación de la plataforma empleando los métodos explicados en los apartados 2.5.6.1 y 2.5.6.2. Si el sistema se encuentra en reposo, se espera que su valor sea constante (ya que no se produce ningún movimiento).

(a) Filtro complementario: El valor de la orientación de la plataforma se obtiene a partir de los datos obtenidos por el giróscopo y el acelerómetro, mediante el cálculo del ángulo entre los ejes del acelerómetro y el ángulo calculado a partir de la integración de la velocidad angular. Para la integración es necesario tener un valor inicial del ángulo, al estar en el estado de calibración se impone que este valor sea 0.

Es necesario encontrar los valores de las constantes de los filtros alto y bajo empleados para obtener los ángulos de rotación.

El ángulo de rotación que indica la orientación que tiene la plataforma es el ángulo de rotación en el eje Z del sistema de coordenadas del entorno de movimiento. Éste ángulo únicamente depende del valor medido por el giróscopo, por lo que es de gran importancia que sus medidas estén calibradas correctamente, ya que, en caso contrario, se acumulará un error temporal de la forma $k{\cdot}t$, donde k es el valor medio de las medidas en reposo (offset), es decir, el error sistemático de medida.

Al arrancar la plataforma, las primeras 100 medidas son ignoradas, ya que el sistema tarda al menos 10 segundos en estabilizarse, por lo que estas medidas no son enviadas al alto nivel.

En las primeras pruebas, la calibración del giróscopo se realizó en el bajo nivel, como se explica en el apartado 2.5.4, por lo que el valor que se envía por el bus CAN es el calibrado en formato *short*. Además, estos valores están redondeados en centésimas de °/s, por lo que se obtiene un 0.001°/s de error.

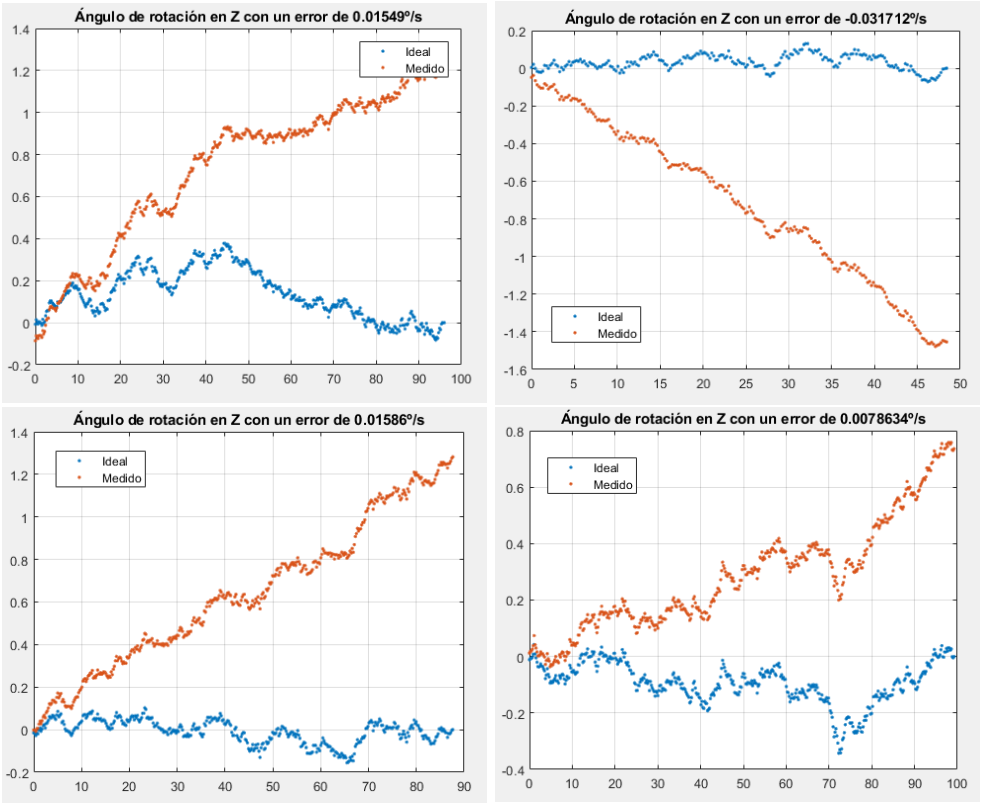

Figura 4.9: Ángulo de rotación en el eje Z con el sistema en reposo, con la calibración en el bajo nivel

Al tomar medidas con la plataforma en reposo, como se observa en las figuras 4.9, se obtiene un error de medida por encima de 0.01°/s, lo que hace que el ángulo de la plataforma, tras 1 minuto de tomar medidas, llegue a superar el valor de ±1°. Para tiempos cortos de funcionamiento, esta calibración es válida, pero cuando el sistema se encuentra en reposo durante periodos más largos, el valor del ángulo de rotación se aleja bastante de su valor ideal (0°).

A continuación, se realiza la calibración del giróscopo en el alto nivel, en el nodo de lecturas del bus CAN, apartado 3.3. El nodo sensores comienza a enviar las medidas sin calibrar del IMU al alto nivel. Una vez recibidas, el alto nivel comienza la calibración, mostrando más precisión que en el caso anterior, como se observa en las figuras 4.10.

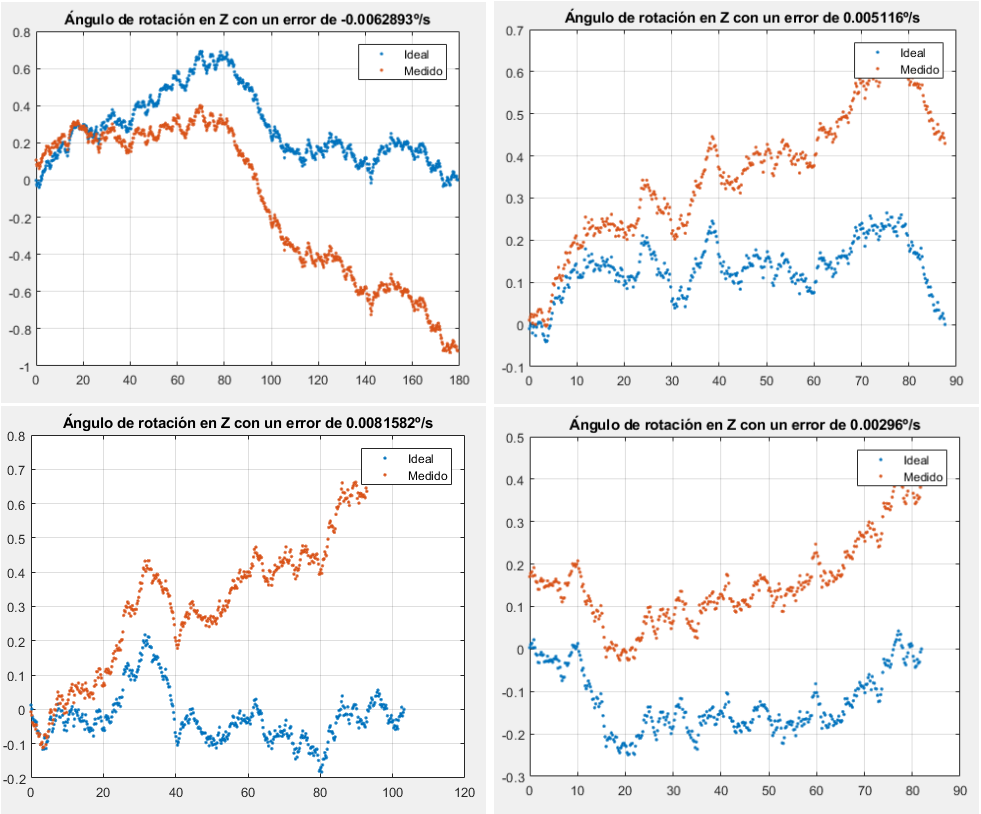

Figura 4.10: Ángulo de rotación en el eje Z con el sistema en reposo, con la calibración en el alto nivel

Se observa que, aun haciendo la calibración, el sistema sigue presentando errores menores a $\pm 0.001^{o}/s$, lo que provoque que el ángulo de la plataforma resultante se incremente a lo largo del tiempo (en menor medida que en el caso anterior). En la tabla 4.7, se representa el offset calculado en cada caso para hacer la calibración del sensor, y el error existente en los experimentos anteriores:

Medida	Offset	Error
1	-0.6442	-0.0063
2	-0.6449	-0.0051
3	-0.6512	0.0082
4	-0.6488	0.0030

Tabla 4.7: Error en el ángulo θ_z tras la calibración en el alto nivel

En el siguiente experimento, se fija el offset de la medida de la velocidad angular del eje Z del giróscopo con un valor de -0.645 $^{o}/s$, obteniendo los siguientes resultados:

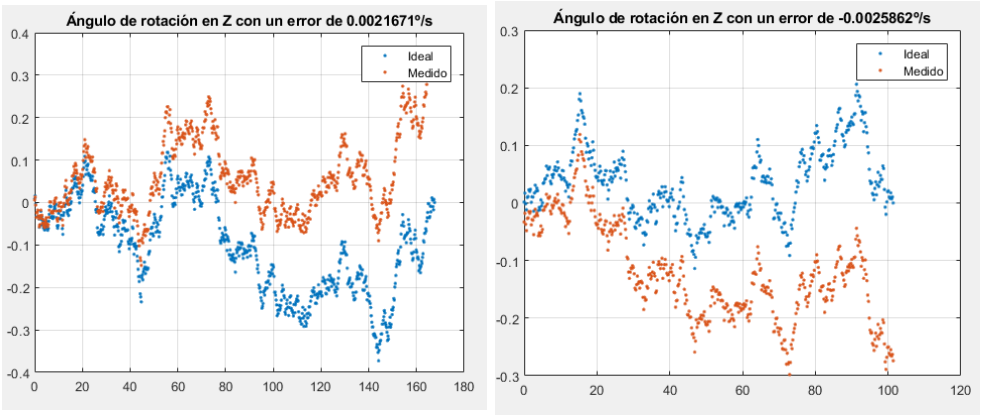

Figura 4.11: Ángulo de rotación en el eje Z con el sistema en reposo, con la calibración en el Alto Nivel

En la tabla 4.8, se muestra una comparativa de los errores de medida del sensor cuando se realiza la calibración del giróscopo en el bajo nivel o en el alto nivel. La media de cada error se calcula en valor absoluto. El error se reduce un 75 % al realizar la calibración en el alto nivel.

Calibración	**|Media del error|**
Bajo Nivel	0.0177 º/s
Alto Nivel	0.0045 º/s

Tabla 4.8: Comparativa error en el ángulo θ_z calibraciones bajo/alto nivel

(b) Filtro de Madgwick: Con los valores medidos por el IMU de aceleración lineal y velocidad angular, se procede a calcular los cuaternios de orientación de la plataforma con el Filtro de Madgwick que permitirán obtener los ángulos de rotación de la plataforma.

Tras tomar medidas durante 1 minuto, se observa el comportamiento de la figura 4.12:

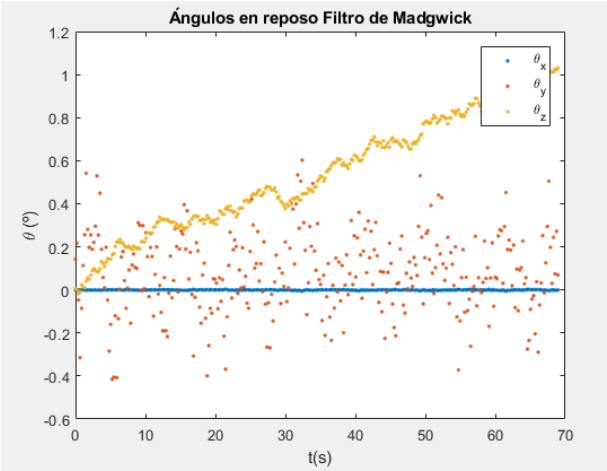

Figura 4.12: Ángulos de rotación con el sistema en reposo - Filtro Madgwick

Se observa que en ambos casos aparecía el problema de la deriva en el ángulo θ_z. En el filtro de Madgwick se observa en la figura 4.12 que trata de corregirlo y mantenerlo constante, pero a lo largo del tiempo el ángulo se incrementa con el paso del tiempo, al igual que ocurría con el Filtro Complementario.

Como el Filtro de Madgwick supone mayor gasto computacional que el filtro Complementario, se descarta su uso y para la aplicación final se utilizará el filtro Complementario para calcular los ángulos de rotación.

Con el fin de alcanzar una mayor precisión en las medidas, se decide que el valor de las velocidades angulares se muestree cada 10 ms y se obtenga el valor medio, para posteriormente enviarlo en décimas de º/s, eliminando así el error acumulado que se observaba en la tabla 4.8.

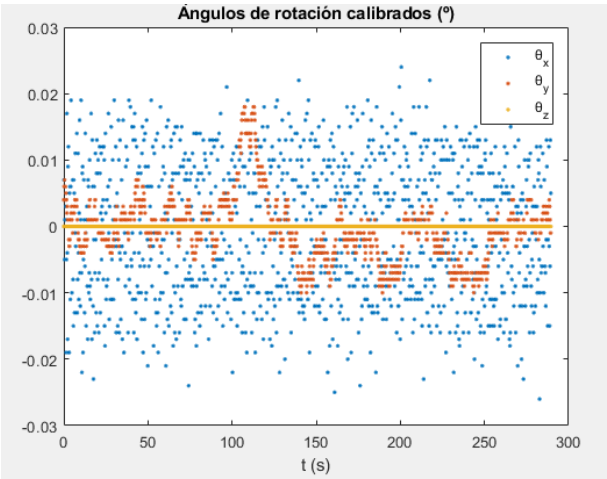

Figura 4.13: Ángulos de rotación con el sistema en reposo calibrados

En la figura 4.13 se muestran los ángulos de rotación de la plataforma tras la calibración, donde se observa que los ángulos en X e Y oscilan entre ±0.02º.Se amplía la gráfica del ángulo en Z, con valores muy reducidos debido a los valores infinitesimales que procesa el alto nivel, concluyendo que el problema de la deriva ha sido eliminado.

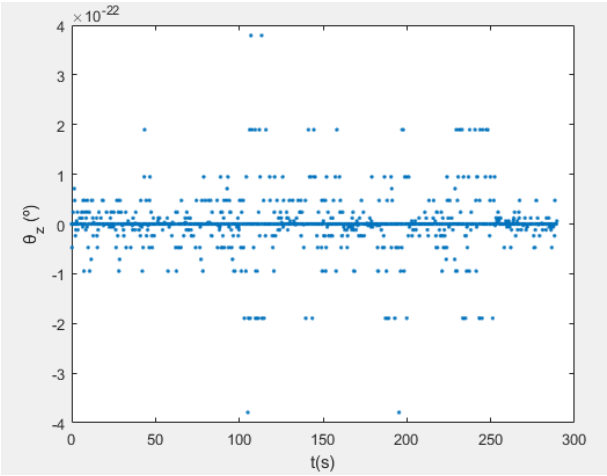

Figura 4.14: Ángulo de rotación en Z calibrado

En la tabla 4.9, se muestran los valores medios de las medidas de los ángulos de rotación de la plataforma tras realizar la calibración en el bajo nivel y empleando el filtro de Madgwick:

Valor medio	Calibrado
θ_x	-6.3794·10^{-4}
θ_y	-5.0435·10^{-4}
θ_z	0

Tabla 4.9: Media del ángulo de rotación calibrado

Se obtienen los histogramas con las medidas de los ángulos de rotación de los ejes X e Y, observando que sigue una distribución normal y que sus medidas se encuentran dentro de la curva de Gauss.

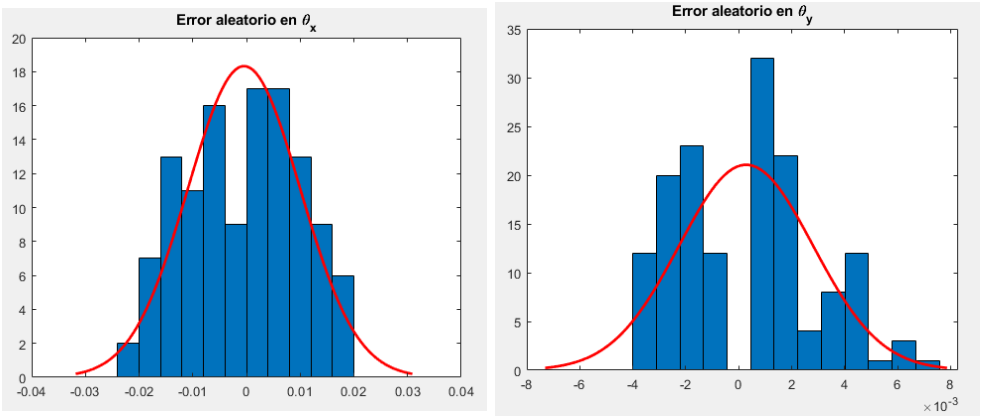

Figura 4.15: Histogramas de la rotación en reposo

4.2.3.2 Matrices de covarianza

Una matriz de covarianza es una matriz cuadrada que contiene la variación conjunta de dos variables aleatorias entre sí. Se calcula dicha matriz para cada conjunto de variables (acelerómetro, giróscopo y rotación) una vez que el sensor está calibrado. Estos valores son necesarios para calcular la estimación de la localización de la plataforma mediante la fusión de sensores, empleando el paquete ROS *robot_localization* (fig. 3.1).

$$\Sigma = \begin{pmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \cdots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \cdots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \cdots & \mathrm{Var}(X_n) \end{pmatrix}$$

Figura 4.16: Estructura de una matriz de covarianza

Debido a que cada una de las variables es independiente del resto, las matrices resultantes tendrán únicamente los elementos de la diagonal principal no nulos. Dichos valores son la varianza de cada variable, que se recoge en la tabla 4.10:

Sensor	Variable	Varianza
Acelerómetro	a_x	0.0032
	a_y	0.0011
	a_z	0.0020
Giróscopo	ω_x	0.0084
	ω_y	0.0091
	ω_z	0.0051
Rotación	θ_x	0.01
	θ_y	0.01
	θ_z	0.0098

Tabla 4.10: Varianzas del IMU

4.2.3.3 Giro completo y corrección del ángulo

El siguiente experimento consiste en realizar un giro en el eje Z de la plataforma completa tanto en el sentido de las agujas del reloj como en sentido contrario, con el observar si toma todos los valores de ángulo posibles.

Para ello, se debe tener en cuenta la definición de los ejes del IMU de la figura 2.10, para ver si los resultados tienen sentido. Cuando el sistema gira en sentido contrario a las agujas del reloj, la velocidad angular es positiva, y cuando gira en sentido a las agujas del reloj, es negativa.

Al realizar un giro de la plataforma completo, en ambos sentidos, se observa la necesidad de aplicar un filtro de corrección, ya que el valor medido es menor que el valor esperado. En la tabla 4.11, se compara el valor ideal o esperado y la media del valor medido para cada uno de los casos, con el factor de corrección necesario:

Ideal	Medido	k
90º	$\approx 82^o$	1.09
180º	$\approx 167^o$	1.08
270º	$\approx 250^o$	1.08
360º	$\approx 332.5^o$	1.08

Tabla 4.11: Comparativa valores ideal y medido de θ_z

En la tabla (4.11), se observa que el error en la medida es lineal, pues aplicando un factor de corrección k $\simeq$ 1.08, se alcanzarían los valores ideales. Este factor de corrección es muy próximo a 1, por lo que no afectará a la medida cuando la plataforma se encuentra en reposo.

En las figuras 4.17 se representa la velocidad angular en el eje Z, y el ángulo de rotación θ_Z, tanto el valor medido (azul) como el corregido (rojo) en distintos ensayos.

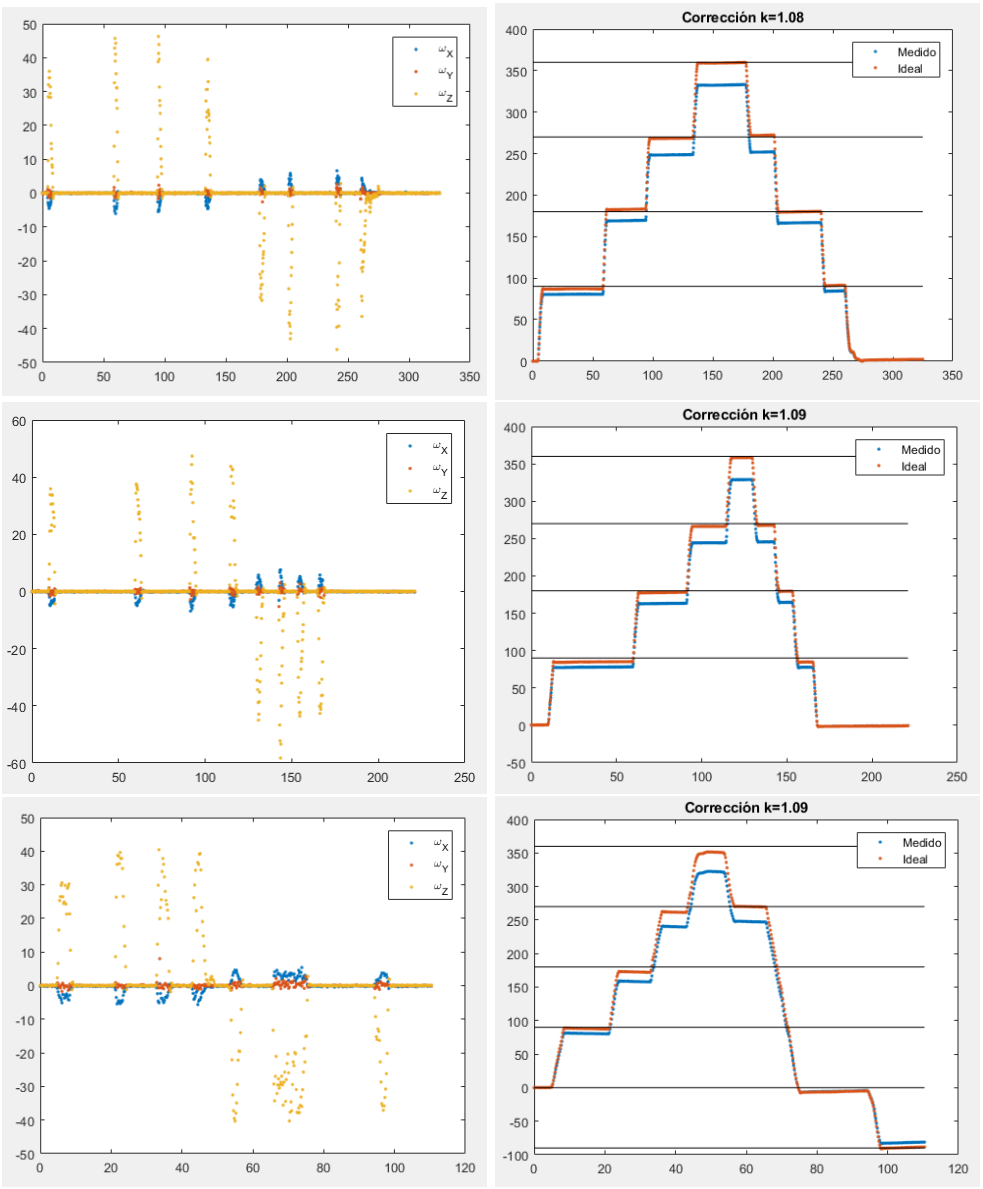

Figura 4.17: Velocidad angular y ángulo en Z corregido

En caso de emplear el Filtro de Madgwick, se observa un comportamiento similar al de las figuras 4.17. En la figura 4.18, se observa que sería necesario emplear una constante de corrección, por lo que aplicar este filtro no supone ninguna mejora.

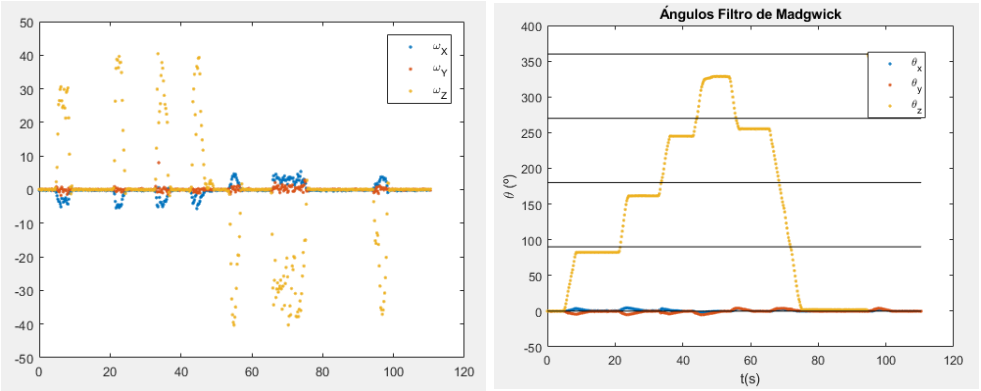

Figura 4.18: Velocidad angular y ángulo empleando el Filtro de Madgwick

4.3 Sistema de Navegación

El sistema actual, tiene implementado un sistema de navegación autónoma basado en el posicionamiento relativo estimado por la odometría de los enconders: en función de cuánto rotan las ruedas, se puede calcular cuánto ha avanzado la plataforma en los ejes X e Y. Esta información se utiliza para calcular y enviar las consignas de navegación, mediante una posición final (*goal*) a la que se tendrá que desplazar. Para ello, el sistema de navegación envía una consigna de velocidad con la que podrá alcanzar la meta al controlador.

En el caso de realizar una trayectoria en línea recta, la estimación por odometría es buena, pero cuando el sistema tiene que girar, existen errores producidos por el deslizamiento de las ruedas que hacen que esta estimación no lo sea.

4.3.1 Giro completo

De la misma forma que en el apartado anterior, se realiza un giro completo en la plataforma SARA usando la pila de navegación de ROS, con el fin de observar si la transición de una orientación a otra es suave y correcta.

Para ello, desde la posición inicial (0º), se hará girar en el sitio a la plataforma, pasando por las siguientes orientaciones: 90º, 180º y 270º, terminando de nuevo en la posición inicial. La velocidad angular para el caso en el que la plataforma gire en el sitio se ha fijado a 0.2 rad/s (11.46 º/s).

- Sentido antihorario: Desde la posición de reposo, se gira desde 0º en pasos de 90º en sentido antihorario, obteniendo un sentido de giro positivo. Cuando llega a los puntos clave, el sistema espera 2 segundos para cambiar a la siguiente posición.

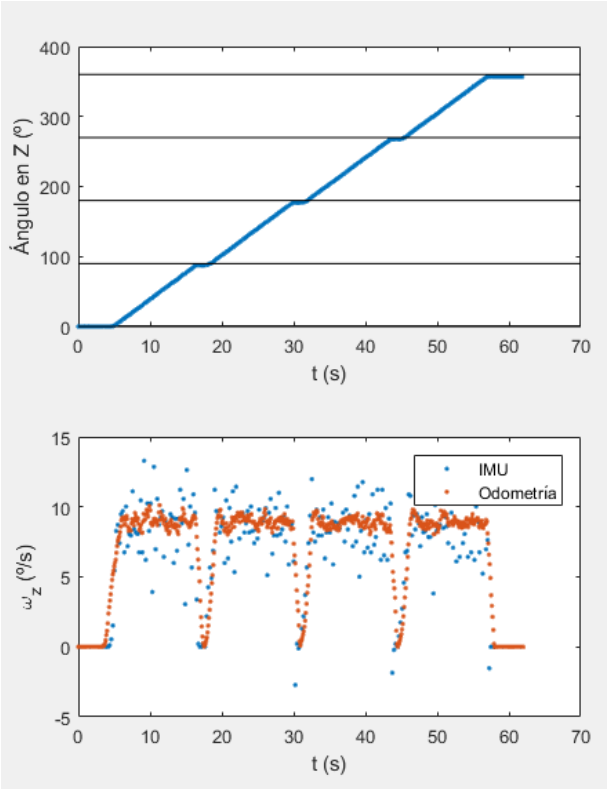

Figura 4.19: Rotación en sentido antihorario con navegación autónoma

- Sentido horario: Desde la posición de reposo, el sistema gira en pasos de 90º en sentido horario, con una velocidad angular negativa. Cuando llega a los puntos clave, el sistema espera 2 segundos para cambiar de posición.

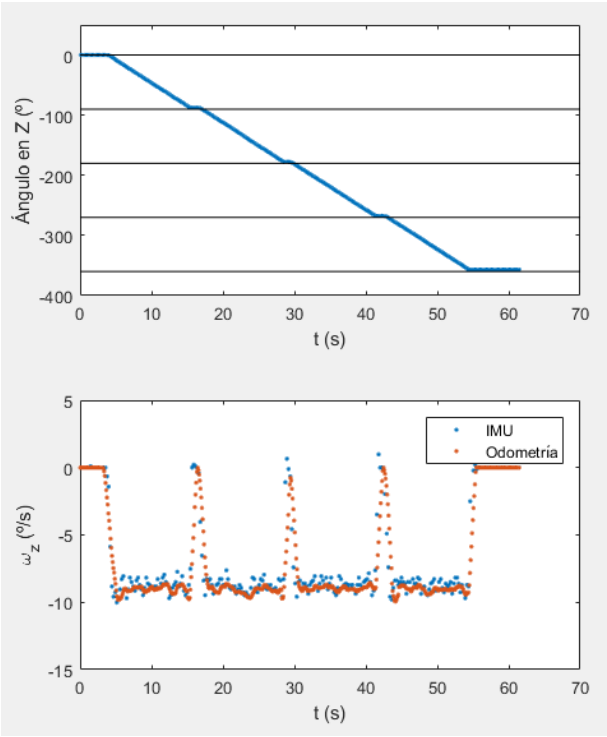

Figura 4.20: Rotación en sentido horario con navegación autónoma

En ambos sentidos de giro, se observa una transición suave: se obtienen los 4 puntos de orientación sin ningún movimiento brusco.

4.3.2 Trayectoria cuadrada

Empleando el método del apartado 3.6.1, se crea una trayectoria cuadrada de 4x4m empleando el paquete ROS *move_base*: se envían las posiciones finales en X e Y, que se transformarán en desplazamientos lineales para los tramos rectos de la trayectoria, y la orientación final que tendrá la plataforma, enviada en forma de cuaternio.

Punto 1	x = 5	y = 1	$\theta_z = 90^o$
Punto 2	x = 5	y = 5	$\theta_z = 180^o$
Punto 3	x = 1	y = 5	$\theta_z = 270^o$
Punto 4	x = 1	y = 1	$\theta_z = 0^o$

Tabla 4.12: Definición de los puntos de la trayectoria cuadrada

En la figura 4.21, se representa la trayectoria resultante que debe seguir la plataforma, donde se realiza un cuadrado de 4x4 m. El paquete de ROS, junto con los datos procedentes de los sensores de la plataforma (odometría), realizará los ajustes necesarios para seguir dicha trayectoria.

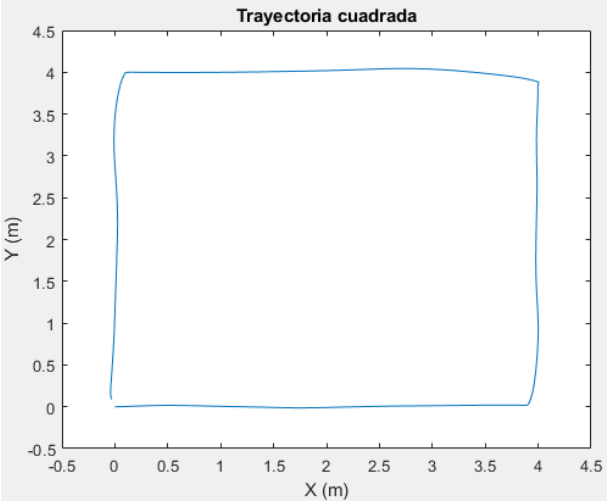

Figura 4.21: Definición de la trayectoria cuadrada en 2D

Al ejecutar el programa, se observa que la plataforma no es capaz de seguir el cuadrado, en lugar de realizar un giro de 90º sobre sí misma, realiza un giro menor debido al deslizamiento de las ruedas, lo que produce que los siguientes puntos finales no se alcancen. Es necesario realizar un estudio de los resultados obtenidos empleando sólo la odometría (sistema inicial) y del IMU de manera aislada, con el fin de analizar como el IMU corregiría los fallos de la odometría.

En el caso de la velocidad angular en el eje Z, se observa que tanto la obtenida por el IMU como la de la odometría siguen una forma similar, con un error de ganancia que hace que la velocidad angular calculada por odometría sea menor que la real. Esto puede provocar que el ángulo estimado por la odometría sea menor que el real.

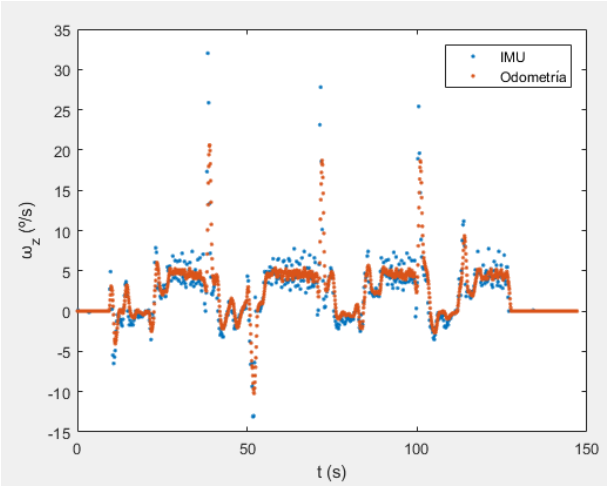

Figura 4.22: Velocidad angular en Z del IMU y de la odometría

En la figura (4.23), se representa la variación del ángulo de rotación en el eje Z (estimado por la odometría y por el IMU) y la variación de las posiciones en X e Y respecto del tiempo.

En la gráfica inferior, se observa que desde la posición de reposo (0,0), avanza en primer lugar en el eje X una longitud total de 4 m. En el instante de tiempo en el que la posición varía, el ángulo de rotación proporcionado por la odometría se mantiene constante a 0º.

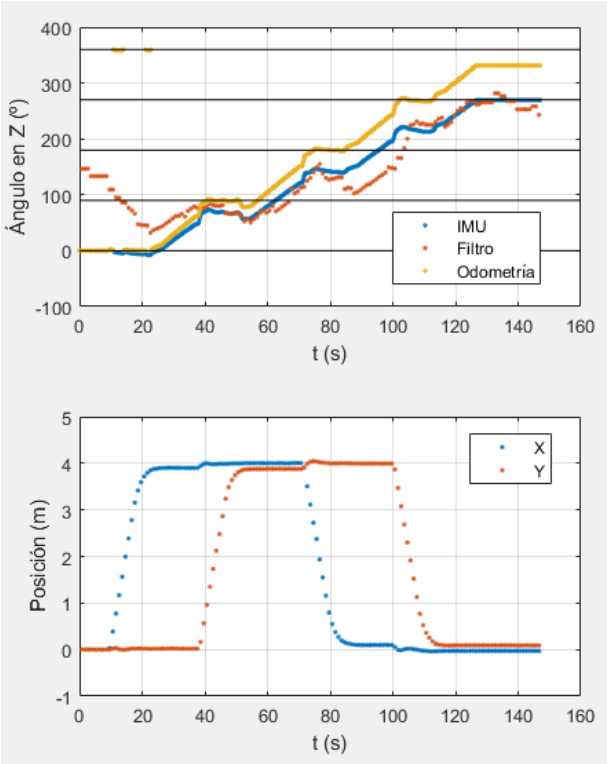

Figura 4.23: Variación de θ_z del IMU y de la odometría

Cuando se alcanza la posición final (punto 1), la gira rota sobre sí misma 90º, como marca la estimación de la odometría. El ángulo que rota es menor en todos los casos, como se representa en la tabla (4.13), con un error de ganancia de que se encuentra entre 1.25 y 1.30.

Odometría	IMU
90º	70º
180º	143º
270º	216º
360º	270º

Tabla 4.13: Estimación de θ_z por odometría e IMU

4.3.3 Fusión de sensores

La fusión de las medidas obtenidas por el IMU y la odometría de los encoders, se realiza empleando el paquete de ROS *robot_localization*.

La plataforma se mueve en un entorno de 2D en un sistema XY, avanzando con una velocidad lineal en el eje X y rotando sobre el eje Z, con velocidad angular en Z. Estos serán los parámetros fundamentales del sistema que se tendrán en cuenta para fusionarlos de cada sensor, con el fin de obtener mejores resultados que empleando unicamente la odometría de los encoders.

El paquete *robot_localization* da la opción de seleccionar qué parámetros de cada sensor se fusionarán para obtener la nueva odometría. Se probará con distintas combinaciones hasta obtener el resultado más preciso. Debido a que la orientación siempre es más precisa con el IMU, la estimación de θ_z será siempre la proporcionada por el sensor inercial.

- **Velocidad linear:** la plataforma se mueve en XY mediante una consigna de velocidad cmd_vel que se calcula combinando la pila de navegación de ROS con el controlador de la plataforma, para lograr su objetivo. En una primera estimación, se fusiona la información del IMU de la orientación con la velocidad lineal en X (al ser un robot diferencial, su velocidad en Y es nula).

  En la figura 4.24 se representa la trayectoria calculada por la odometría de los sensores, donde el sistema traza una trayectoria cuadrada, y la trayectoria con la fusión de la información del IMU y la velocidad lineal v_x, que sería la trayectoria real. Al no fusionar la información de la posición en X e Y, se pierde la referencia de dónde se encuentra la plataforma y no es capaz de conseguir realizar el cuadrado.

  Esta misma trayectoria en rojo, es la que el sistema realizaba realmente cuando no se realizaba la fusión de los sensores.

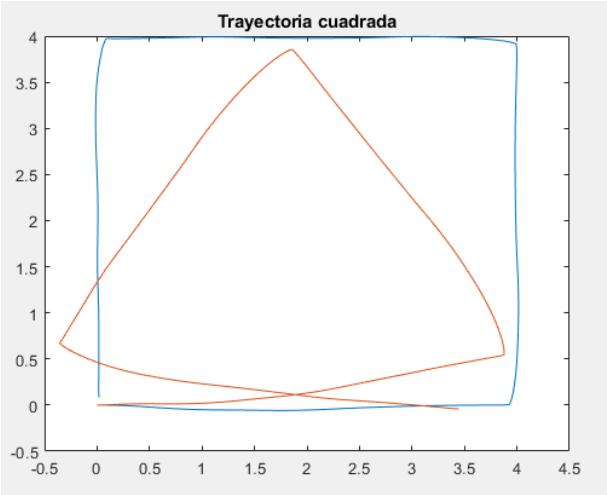

Figura 4.24: Trayectoria obtenida fusionando la velocidad lineal

La velocidad angular calculada tanto por la odometría de los encoders como por el IMU, tiene una forma muy similar como se observa en la figura 4.25, donde realiza rotaciones en sentido antihorario (positivo).

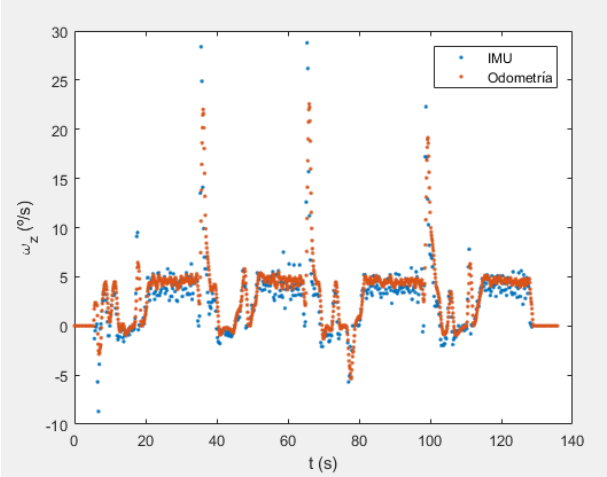

Figura 4.25: Velocidad angular en Z del IMU y de la odometría fusionando la velocidad lineal

Finalmente, se observa la orientación que toma la plataforma en cada punto: para obtenerla, la plataforma gira de forma suave hasta obtener la orientación deseada, por lo que el problema de conseguir el ángulo deseado se corrige.

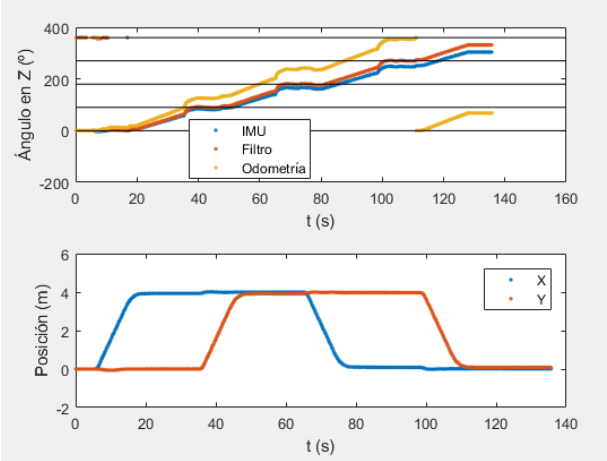

Figura 4.26: Variación de θ_z del IMU y de la odometría fusionando la velocidad lineal

- **Posición:** Para calcular la consigna de velocidad, el controlador necesita conocer la posición dada por la odometría. En esta ocasión, se fusiona la información del IMU con la posición que tiene la plataforma en X e Y.

  En la figura 4.27, se representa la trayectoria obtenida, donde ya se observa una forma más cuadrada. En los tramos rectos, se obtiene una pequeña desviación que hace que su forma no sea del todo lineal. Esto ocurre porque el navegador calcula la trayectoria entre dos puntos, y no tiene por qué ser lineal.

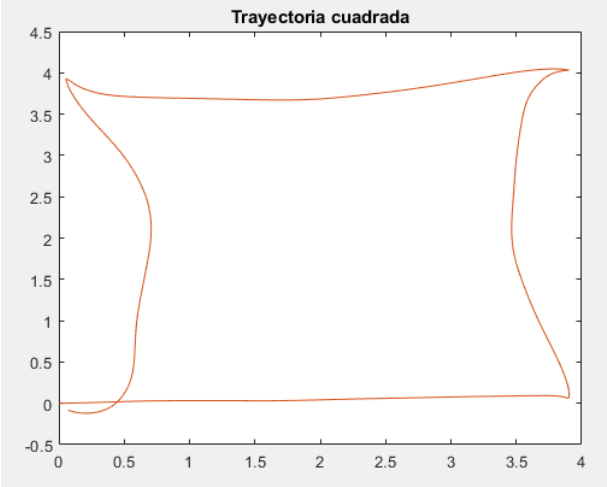

Figura 4.27: Trayectoria obtenida fusionando la posición

La velocidad angular calculada tanto por la odometría de los encoders como por el IMU, tiene una forma muy similar como se observa en la figura 4.28, donde realiza rotaciones en sentido horario (negativo).

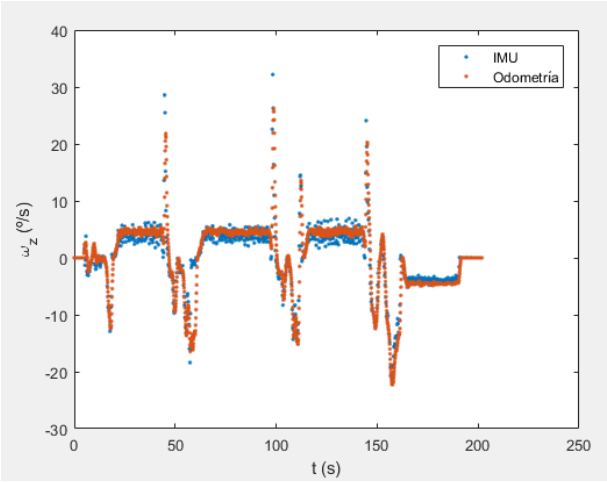

Figura 4.28: Velocidad angular en Z del IMU y de la odometría fusionando la posición

Estas rotaciones en sentido horario, hacen que cuando la plataforma llega al punto marcado, en lugar de realizar la rotación por el tramo más corto, se desvía hasta casi 90º en sentido contrario al que debería girar, y después toma la orientación marcada, como se observa en la figura 4.29.

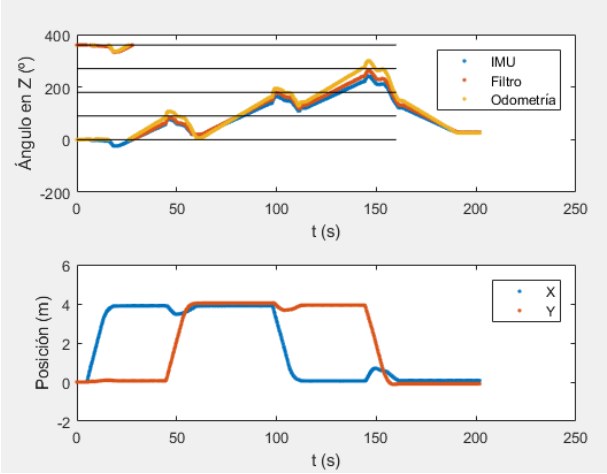

Figura 4.29: Variación de θ_z del IMU y de la odometría fusionando la posición

- **Posición y velocidad lineal:** Para conseguir mejores resultados, se fusiona con la información del IMU la velocidad lineal y la posición en X e Y, con el fin de obtener una mejor trayectoria tanto en los tramos rectos como en los giros en el sitio. De esta forma, el sistema tendrá una mejor referencia de dónde se encuentra y qué consigna de velocidad debe enviar.

  En la figura 4.30, se representa la trayectoria obtenida, donde se suavizan los tramos rectos respecto al caso donde solo se fusiona la información de la posición (4.27).

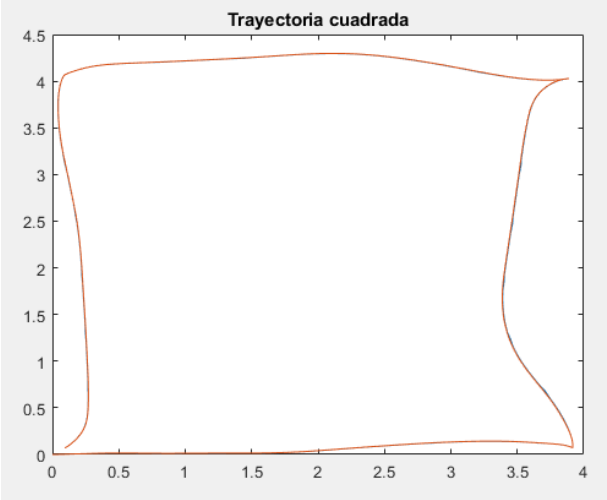

Figura 4.30: Trayectoria obtenida fusionando la posición y velocidad lineal

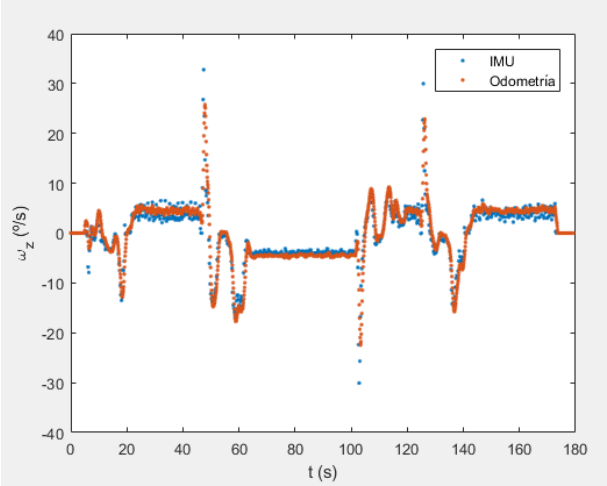

Figura 4.31: Velocidad angular en Z del IMU y de la odometría fusionando la posición y velocidad lineal

En la velocidad angular, se observan tramos positivos y negativos 4.31.

En este caso, 4.32, se observan rotaciones más suaves, aunque en el caso de lograr los 180º realiza prácticamente una rotación completa, tomando el tramo más largo. Observando la trayectoria 4.30, es el tramo menos recto, ya que antes de llegar al punto final, la plataforma se desvía y comienza a rotar.

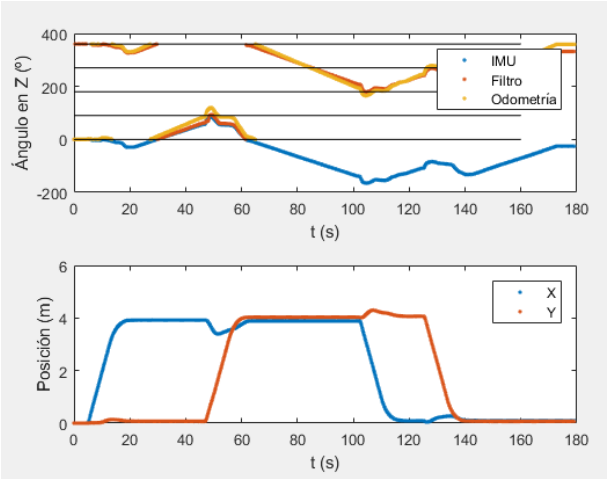

Figura 4.32: Variación de θ_z del IMU y de la odometría fusionando la posición y velocidad lineal

Con el fin de tratar de suavizar estos tramos, se modifica el tiempo de cálculo de la trayectoria del planificador (*sim_time*). Se observa que si el tiempo es menor, la plataforma oscila más en los tramos rectos, y al aumentarlo se consigue que la plataforma se desvíe menos y vaya prácticamente recta.

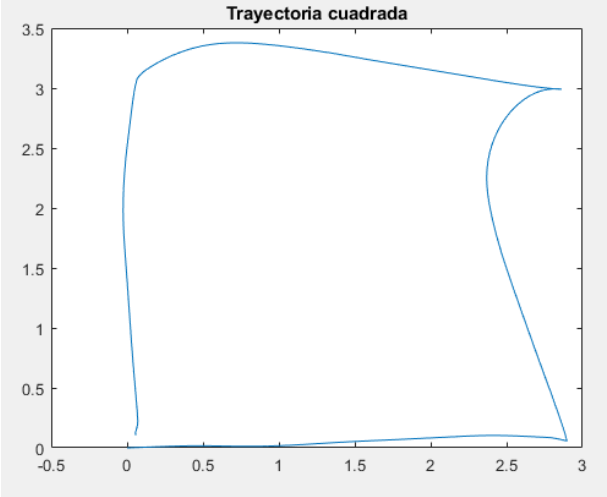

Figura 4.33: Trayectoria fusionando la posición y velocidad lineal

Los cambios de rotación son suaves, pero se sigue presentando la desviación en 180º aunque el tramo lineal es más suave que en el caso anterior.

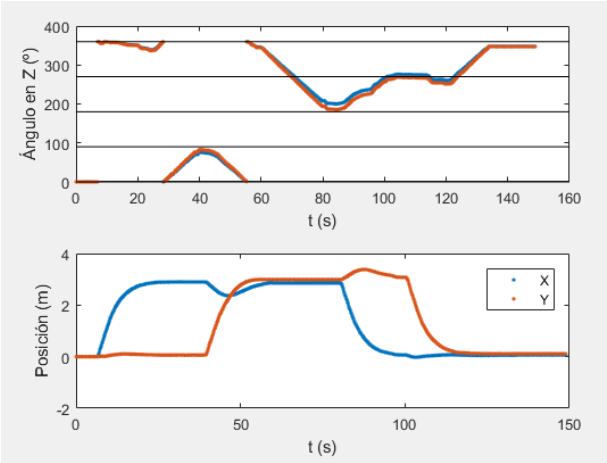

Figura 4.34: Variación de θ_z del IMU y de la odometría fusionando la posición y velocidad lineal

Se define una nueva trayectoria cuadrada, donde se define un nuevo sentido de giro: en el caso anterior, el sentido de giro era antihorario, y para esta nueva trayectoria, se define con sentido horario, de forma que su orientación se decrementa desde 360º hasta 0º.

Punto 1	x = 3	y = 0	θ_z = 270º
Punto 2	x = 3	y = -3	θ_z = 180º
Punto 3	x = 1	y = -3	θ_z = 90º
Punto 4	x = 0	y = 0	θ_z = 0º

Tabla 4.14: Definición de los puntos de la trayectoria cuadrada 2

La trayectoria que sigue el sistema es la siguiente (4.35), donde se observa que en el último tramo el robot no sigue una trayectoria lineal.

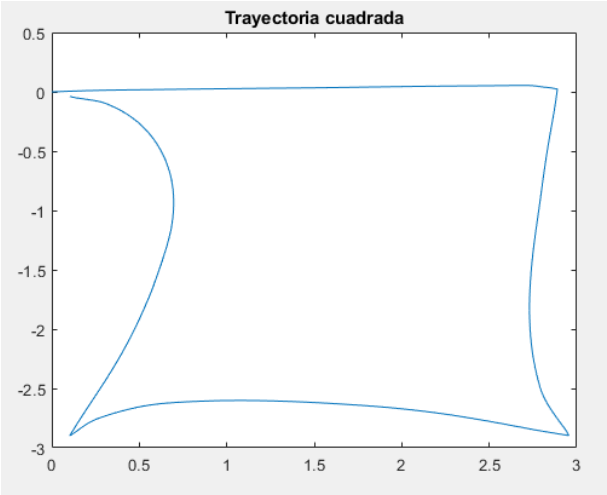

Figura 4.35: Trayectoria fusionando la posición y velocidad lineal

Los cambios de rotación son suaves, aunque en el tramo 3 se observa que el robot se desvía de los 180º en el tramo lineal, y en el tramo 4 es incapaz de seguir una trayectoria lineal, cambiando de los 90º a los 180º y realizando medio giro hasta los 0º, lo que hace que la rotación sea más larga de lo esperado.

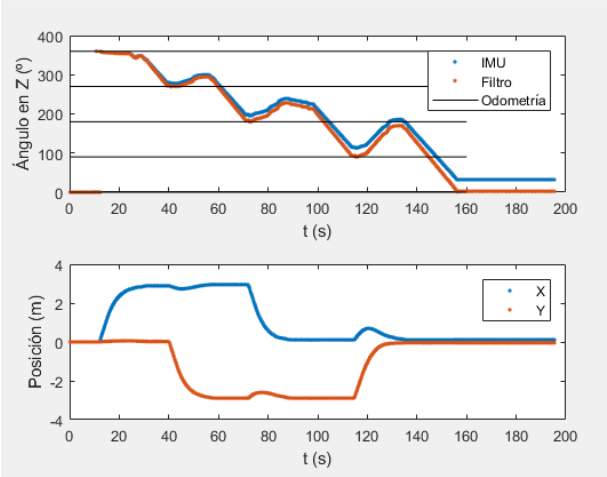

Figura 4.36: Variación de θ_z del IMU y de la odometría utilizando la trayectoria 2

Con el fin de suavizar las transiciones, se modifican los parámetros del sistema de navegación: se incrementa el tiempo que tarda en realizar el cálculo de la trayectoria, consiguiendo una velocidad menor pero con una trayectoria sin desviación, como se observa en la figura 4.37. Para este caso, se fusiona la posición XY y velocidad lineal en el eje X obtenida por la odometría de las ruedas, con la velocidad angular y el ángulo de rotación en Z. Además, se configuran los parámetros *imu0_differential* y *odom0_differential* como false, para que solo se tengan en cuenta sus medidas.

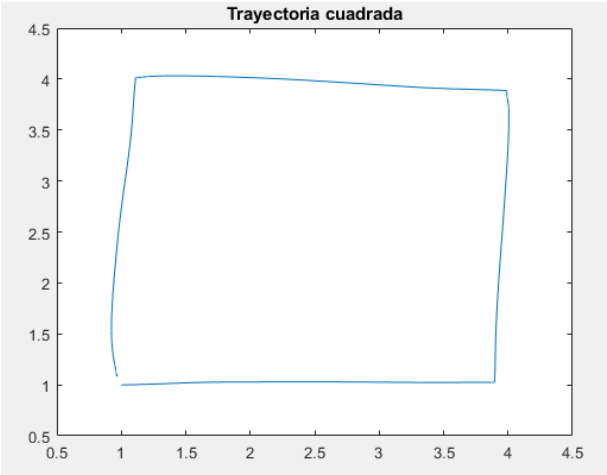

Figura 4.37: Trayectoria cuadrada aumentando el tiempo de simulación

Se observa que la velocidad angular (4.38) se mantiene con valores similares en cada tramo, con sentido siempre positivo, lo que implicará que para lograr los giros los realizará siempre en el mismo sentido y por el trayecto más corto.

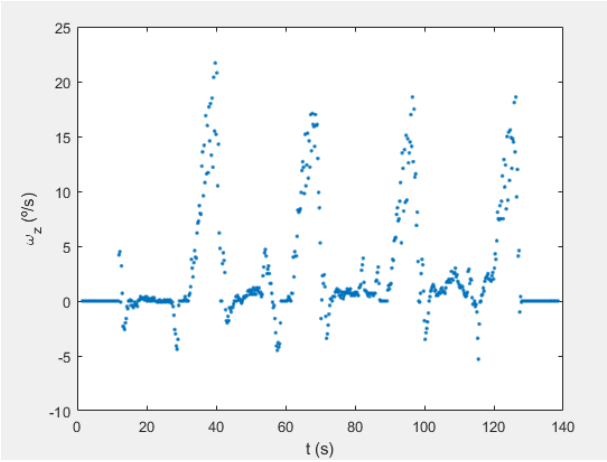

Figura 4.38: Velocidad angular en Z aumentando el tiempo de simulación

Por último, se observa las rotaciones son suaves y que no busca su orientación de referencia inicial para realizar los giros.

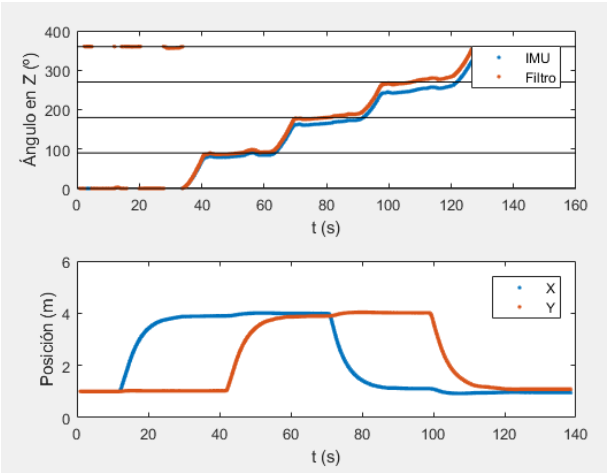

Figura 4.39: Variación de θ_z aumentando el tiempo de simulación

4.3.4 Navegación en un entorno real

Se puede navegar en un entorno real empleando el método de navegación del apartado 3.6.2. Debido a que la plataforma no dispone de ningún sensor de visión, en la herramienta RViz se cargan los mapas de la Escuela Politécnica Superior con el paquete ROS *"map_server"* [28]. De este modo, la plataforma navegará empleando los sensores de ultrasonidos, y la odometría obtenida a partir de la fusión de sensores, junto con la información proporcionada por el mapa cargado. Se probarán dos métodos para la navegación:

1. La orientación medida por el IMU, al inicializar los nodos, está fijada a 0º. Al posicionar la plataforma en el inicio, el topic /set_pose tomará la orientación del mapa de RViz. Durante la navegación, el nodo /ekf_se es capaz de reajustar el offset entre la medida del IMU y la salida del filtro, ya que el parámetro imu0_differential del archivo de configuración del paquete *robot_localization* (3.4) se inicializa a true, para que al estimar la orientación de la plataforma, tome la información del IMU y de RViz.

   En la figura 4.40, se representa la evolución temporal del ángulo de orientación en el eje Z de la plataforma, donde se observa dicho suceso.

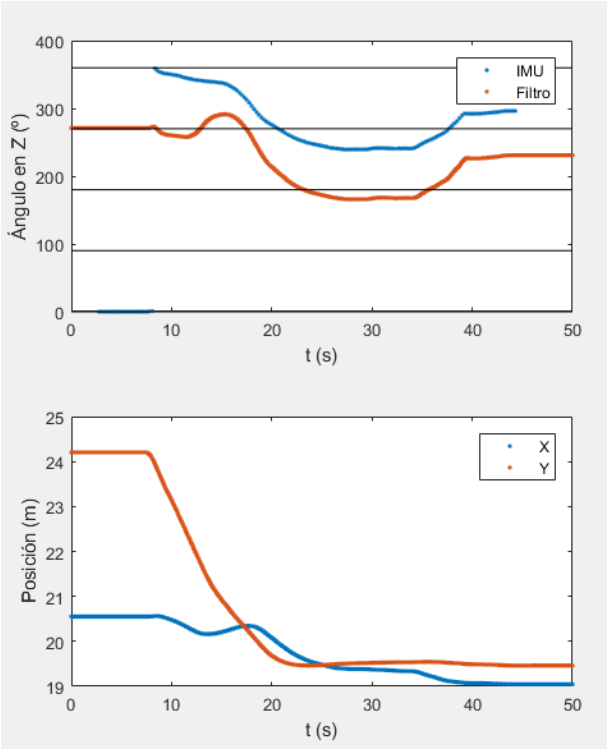

Figura 4.40: Evolución temporal de θ_z navegando en el entorno RViz

En la figura 4.41, se observa la trayectoria de SARA cuando navega en el entorno de RViz. Se observa que la trayectoria no se realiza en línea recta, si no que se produce una pequeña oscilación. Esto es debido a que, para que la plataforma se posicione en la orientación correcta, se producen unos saltos entre la información procedente del IMU y la de RViz.

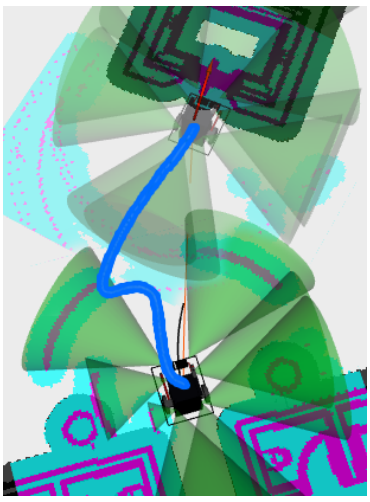

Figura 4.41: Trayectoria de navegación en el entorno RViz

2. Con el fin de eliminar las oscilaciones en las trayectorias, se fija el parámetro imu0_differential del archivo de configuración del paquete *robot_localization* (3.4) a false, de forma que la única fuente de información de la orientación θ_z sea el IMU. Para inicializar la orientación a otro valor distinto de 0, se procede a introducir la función creada en el método 2 del apartado 3.6.2, para que cada vez que se inicialice la posición con RViz, se reajuste el offset.

En la figura 4.42, se representa la evolución temporal del ángulo de orientación en el eje Z de la plataforma con la nueva configuración, donde se observa que el ángulo de la orientación de la plataforma se mantiene constante, debido a que la trayectoria calculada es lineal, frente al ángulo calculado en la figura 4.40.

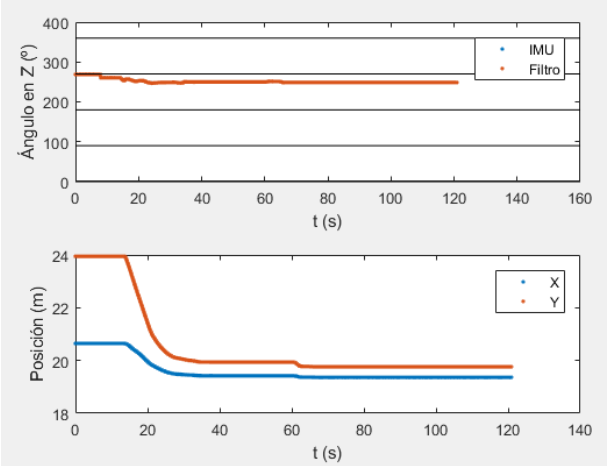

Figura 4.42: Evolución temporal de θ_z corregido navegando en el entorno RViz

En la figura 4.43, se observa la trayectoria de SARA con la nueva configuración. Respecto a la figura 4.41, se observa una trayectoria mucho más lineal si oscilaciones, de forma que llega a la posición final de manera más exacta.

Figura 4.43: Trayectoria de navegación en el entorno RViz corregida

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

En este trabajo de fin de máster se fijó el objetivo de continuar con la mejora del sistema de navegación autónoma de la silla de ruedas SARA [1], partiendo de trabajos previos [2].

Para ello, se integra un sensor inercial de tipo IMU que contiene un acelerómetro y un giróscopo. Es capaz de proporcionar información de la aceleración lineal y la velocidad angular que lleva el sistema, pudiéndose calcular la orientación de la silla mediante integración. Para poder trabajar con estos datos, se modifica el Nodo Sensores que se encarga de leer la información recibida por el sensor y transformarla en unidades que tengan un significado físico. También se realiza la calibración del IMU, ya que se observa que el sensor tiene errores sistemáticos (debido a un mal posicionamiento del mismo o por ruidos externos) que causan que las medidas sean erróneas y produzcan una falsa rotación del sistema cuando se encuentra en reposo.

Se ha actuado también sobre el alto nivel, formado por paquetes ROS que emplean la información de los distintos sensores que componen el sistema para realizar la navegación de la silla. La información de la odometría previa se fusiona con los datos del IMU de forma sencilla mediante el uso del paquete *robot_localization* mejorando la estimación de la posición, corrigiéndose así los errores provocados por la estimación mediante la odometría de las ruedas.

Se observa que, al fijar una trayectoria al navegador, en ocasiones el planificador empleado no calcula la trayectoria más corta o más lineal, causando que el sistema se desvíe de la trayectoria ideal, pero alcanzando el punto final marcado.

5.2 Trabajos futuros

A continuación, se exponen algunas propuestas de mejoras o trabajos futuros que pueden partir de este trabajo de fin de Máster:

- Llevar a cabo este trabajo sobre la plataforma KATE, ya que ambas plataformas poseen la misma estructura flexible y escalable, de modo que se pueda mejorar la estimación de su localización.
- Combinar la información proporcionada por el IMU con un magnetómetro. Un magnetómetro es capaz de medir el campo magnético de la tierra y situar el norte, sur, este y oeste. De esta forma existiría un sistema de referencia, y el cálculo de la orientación sería más preciso.

- Incorporación de un GPS de forma que se mejore la estimación de la posición de la plataforma, ya que es compatible con el paquete ROS *robot_localization*.
- Incorporación de sistemas de visión capaces de crear un mapa del entorno en tiempo real según navegue la plataforma, de forma que se puedan detectar obstáculos dinámicos y mejorar el cálculo de trayectorias.

Capítulo 6

Pliego de condiciones

En este capítulo se detallará tanto el software como el hardware necesarios para llevar a cabo el trabajo de fin de máster.

6.1 Equipos físicos

- Silla de ruedas:
  - Dimensiones 108 cm x 58 cm x 110 cm (largo - ancho - alto)
  - Radio de la rueda 15.5 cm
  - Distancia entre ejes 52.5 cm
  - Reductora x32
  - Batería. Pack de dos baterías de 12V en serie con tecnología AGM.
  - Display LCD serie i2cS310118.
  - Teclado matricial 4x3 teclas S310119.
  - Joystick.
- Sensores:
  - Encoders "HEDS-5500". Dos discos de 500 pulsos: 2000 flancos por vuelta.
  - Sensor de soplido *awm2000v*.
  - Cargador de baterías *EMSPower7969*.
  - Conversor DC-DC *Mascot8862000079*.
  - Controlador de motores *RoboteqAX3500*.
  - Microcontrolador Philips *LPC2129*.
  - Sensores de ultrasonido *SRF02*.
  - Unidad de medición inercial *MPU6050*.
  - Conversor USB-CAN *LawicelCANUSB*.
- Portátil Lenovo Yoga

6.2 Software

- Sistema Operativo Ubuntu 20.04 LTS
- Keil uVision4: programación del microcontrolador LPC2129.
- Matlab: procesamiento y análisis de datos.
- ROS Noetic.
- OverLeaf (Latex): procesador de textos.

Capítulo 7

Presupuesto

En este capítulo se detalla la estimación del coste total que supone la realización de este proyecto. En los siguientes apartados se dividen los recursos en función de su origen.

7.1 Recursos Hardware

Para la realización del trabajo, se ha empleado la silla de ruedas SARA, a la que se ha incorporado el sensor IMU MPU6050. También ha sido necesario el uso de un ordenador portátil Lenovo Yoga. Los costes de hardware son los siguientes:

Concepto	**Coste**
Silla de ruedas	1000,00 €
Ordenador portátil	700,00 €
MPU6050	10,00 €
TOTAL	1710,00 €

Tabla 7.1: Recursos Hardware

7.2 Recursos Software

Para realizar el trabajo, ha sido necesario el siguiente software que permite la programación de los distintos nodos que componen el sistema.

- Bajo nivel: se ha empleado el software *Keil uVision4* para la programación del microcontrolador de Philips *LPC2129*, que realiza la lectura de las medidas de los distintos sensores que tiene la silla, en este caso, del IMU *MPU6050*. Este software, tendrá un coste 0 ya que se usa la versión de estudiante.
- Alto nivel: para interpretar los datos obtenidos por el bajo nivel y crear una comunicación entre los componentes de la silla y el sistema de navegación, se usa el software libre (sin coste) del sistema operativo *ROS*, que integra los paquetes necesarios para la lectura/escritura, fusión, controlador y sistema de navegación.

- Análisis de datos: para analizar las medidas y los resultados obtenidos en el alto nivel, se ha usado el software de MATLAB, ya que es capaz de interpretar y trabajar con los mensajes de tipo *ROS*. Este software, tendrá un coste 0 ya que se usa la licencia de la Universidad.

Concepto	Coste
Keil uVision4	0,00 €
ROS	0,00 €
MATLAB	0,00 €
TOTAL	0,00 €

Tabla 7.2: Recursos Software

7.3 Recursos de mano de obra

A continuación, se exponen los costes de la mano de obra empleada para realizar el proyecto, formada por un ingeniero que se encarga del desarrollo hardware, software y de la escritura de la memoria de dicho trabajo.

Concepto	Precio por hora	Horas trabajadas	Coste total
Ingeniería	50,00 €/h	500	25000,00 €

Tabla 7.3: Recursos humanos

7.4 Presupuesto de ejecución material

En la siguiente tabla (7.4) se realiza un resumen de los costes totales que implica la realización de este trabajo.

Concepto	Coste
Recursos Hardware	1710,00 €
Recursos Software	0,00 €
Costes mano de obra	25000,00 €
TOTAL	26710,00 €

Tabla 7.4: Presupuesto de ejecución material

7.5 Importe de la ejecución por contrata

A continuación, se exponen los costes de la ejecución por contrata, que deben incluir los gastos derivados del uso de las instalaciones donde se desarrolla el proyecto, las cargas fiscales, los gastos financieros, las tasas administraciones y las obligaciones de control del proyecto. El beneficio industrial también se incluye.

Para cubrir estos gastos, se asume un recargo del 22 % de los costes totales de ejecución material.

Concepto	Coste
22 % del coste total de ejecución material	5875,1€

Tabla 7.5: Importe de ejecución por contrata

7.6 Honorarios facultativos

Para este proyecto, se fija un 7 % sobre el coste total de ejecución por contrata.

Concepto	Coste
7 % del coste total de ejecución material	2281,034€

Tabla 7.6: Honorarios facultativos

7.7 Presupuesto total

En la tabla 7.7, se realiza un resumen de los costes totales que implica la realización de este trabajo.

Concepto	Coste
Presupuesto de ejecución material	26710,00 €
Importe de la ejecución por contrata	5875,10 €
Honorarios facultativos	2281,03 €
TOTAL sin IVA	34866,13 €
IVA (22 %)	7650,55 €
TOTAL	42536,68 €

Tabla 7.7: Presupuesto total

Bibliografía

[1] SAR, "Memoria explicativa del proyecto SAR (sistema de ayuda a la movilidad robotizado)," 2018.

[2] F. J. Roda, "Módulos software para un sistema de asistencia a la movilidad en entorno ROS (Robot Operating System)," 2018.

[3] GEINTRA, "Grupo de ingeniería electrónica aplicada a espacios inteligentes y transporte," http://www.geintra-uah.org.

[4] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," 1996.

[5] ——, "Gyrodometry: a new method for combining data from gyros and odometry in mobile robots," 1996.

[6] T. Nagata and G. Ishigami, "Experimental evaluation of gyro-based odometry focusing on steering characteristics of wheeled mobile robot in rough terrain," 2012.

[7] T. Moore and D. Stouch, "A generalized extended Kalman filter implementation for the Robot Operating System," 2016.

[8] "Datasheet del IMU MPU6050," https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf.

[9] "Bus CAN," https://es.wikipedia.org/wiki/Bus_CAN.

[10] "Unidad de medición inercial," https://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial.

[11] "Asociación Padrino Tecnológico," http://padrinotecnologico.org/.

[12] F. Gustafsson, *Statistical Sensor Fusion*. Studentlitteratur, 2018.

[13] "Sistema operativo robótico," http://www.ros.org/.

[14] "Datasheet del microcontrolador LPC2129," http://www.keil.com/dd/docs/datashts/philips/lpc2119_2129.pdf.

[15] "Paquete ROS robot_localization," http://docs.ros.org/melodic/api/robot_localization/html/index.html.

[16] A. Melino, "Plataforma motorizada de apoyo a la movilidad para niños con discapacidad motriz," 2020.

[17] "Guía del usuario del microcontrolador LPC2129 CAN QuickStart," https://www.embeddedartists.com/wp-content/uploads/2018/07/LPC2129_CAN_QuickStart_Board_Users_Guide.pdf.

[18] S. O. Madgwick, "An efficient orientation filter for inertial and intertial/magnetic sensor arrays," 2010.

[19] "Paquete ROS sensor_msgs," http://wiki.ros.org/sensor_msgs.

[20] "Paquete ROS nav_msgs," http://wiki.ros.org/nav_msgs.

[21] "Repositorio canusb," https://github.com/spiralray/canusb.

[22] "Controladores de ROS," http://wiki.ros.org/ros_control.

[23] "Paquete ROS move_base," http://wiki.ros.org/move_base.

[24] "Pila de navegación de ROS," http://wiki.ros.org/navigation.

[25] "Paquete ROS fake_localization," http://wiki.ros.org/fake_localization.

[26] "Mensaje ROS PoseWithCovarianceStamped," http://docs.ros.org/en/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html.

[27] "Herramienta de visualización 3D para ROS," http://wiki.ros.org/rviz.

[28] "Paquete ROS map_server," http://wiki.ros.org/map_server.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá