Trabajo Fin de Máster

Máster en Ingeniería Electrónica, Robótica y Automática

Caracterización y puesta a punto de un Sistema de Radiobalizas para localización interior en robots de inspección.

Autor: Francisco Casas Reyes

Tutor: Samuel Yanes Luis

Tutor: Sergio Toral Marín

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

ACE-TI
Grupo de investigación
Ingeniería Electrónica

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Caracterización y puesta a punto de un Sistema de Radiobalizas para localización interior en robots de inspección.

Autor:
Francisco Casas Reyes

Tutor:
Samuel Yanes Luis
Investigador Predoctoral FPU

Tutor:
Sergio Toral Marín
Catedrático

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Máster: Caracterización y puesta a punto de un Sistema de Radiobalizas para localización interior en robots de inspección.

Autor: Francisco Casas Reyes

Tutores: Samuel Yanes Luis, Sergio Toral Marín

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2022

Agradecimientos

En primer lugar, a toda mi familia, que siempre ha estado presente en todas las fases de mis estudios, tanto en el Grado como en el Máster como en estudios anteriores, y que en cualquier proyecto que me surgía, cualquier idea, cualquier inquietud, ahí estaban ellos apoyándome y dándome su más sincera opinión.

Sin ellos, y mis amigos, que ahora se han convertido en familia, ni este trabajo, ni ninguno de ellos hubiera sido posible, gracias a todos por haber estado ahí siempre, en lo bueno y en lo malo, intentando que no me ahogara en un vaso de agua cuando algo no salía cuando yo quería. Con esa familia me refiero tanto a mis antiguos amigos, de Cabra y Córdoba, como a mis compañeros de Máster, que han constituido un apoyo indispensable.

Esta es la última fase de mi paso por la Universidad, y tengo que agradecer a la Universidad de Sevilla el gran número de oportunidades, tanto de evolución personal como laboral, que ha proporcionado. Han sido muchas horas dedicadas tanto a este trabajo como a otros muchos que he realizado en mi paso por esta escuela, las cuales creo que han merecido la pena una a una.

Por último y no menos importante, agradecer a mis directores de trabajo, Sergio y Samuel, su labor ayudándome y atendiéndome en cada ocasión que necesitaba consejo o directamente un poco de ayuda para seguir adelante. Siempre han estado presentes.

Resumen

En el presente documento se realiza la puesta a punto de un sistema de posicionamiento en interiores para su aplicación en robots de inspección en zonas interiores.

Para el caso que se tiene entre manos, se emplea un robot fabricado por NVIDIA, del cual se mencionan detalles a lo largo del documento, y que es ideal para demostrar cómo se puede plasmar el sistema de posicionamiento en un robot de este tipo.

El robot incorpora una cámara y varias librerías precargadas para implementar aplicaciones con redes neuronales, las cuales no se usarán en este trabajo. Durante el desarrollo sólo se empleará la cámara en una parte del programa de control, con el objetivo de fotografiar el escenario al llegar a un punto objetivo. En futuras ampliaciones del trabajo se podrán implementar aplicaciones de IA, haciéndolo aún más interesante.

Existen tres partes bien diferenciadas en este trabajo: la caracterización del sistema de posición, la caracterización del robot usado en el posicionamiento (tanto mecánica como electrónicamente) y el desarrollo del algoritmo de control de ruta basado en una baliza especial con IMU interna.

El sistema de posicionamiento consiste en una mezcla de RF y ultrasonidos, toda la información se irá exponiendo durante la confección del trabajo, en los sucesivos apartados.

Abstract

This document describes the setup of an indoor positioning system which can be used in inspection robots.

In this case, the robot is manufactured by NVIDIA, and details about it will be mentioned throughout the document. This robot is ideal to demonstrate the application of the positioning system.

The robot incorporates a camera and several preloaded libraries ready to implement neural networks applications, which will not be used during this work. The camera is only going to be used to take photos when the robot arrives to its destination point. In future implementations IA applications could be added to this work, making it even more interesting.

There are three well-differentiated parts in this work: the characterization of the positioning system, the characterization of the robot used in positioning (both mechanically and electronically) and the development of the main control algorithm based on a special beacon with internal IMU.

The positioning system consists of a mix of RF and ultrasound, all the information may be read during the work development.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

La localización de objetos móviles en interiores es un tema muy sonado en la actualidad, en la que el uso de robots en la industria está en auge [1].

El hecho de contar con robots en un espacio en el que conviven con obstáculos y con otros robots hace obligatoria la necesidad de poder controlar la posición de cada uno, con el objetivo de planificar rutas seguras (con seguras se hace referencia a evadir objetos y prevenir paradas indeseadas del robot durante su tiempo de operación).

Uno de los ejemplos más conocidos son los conocidos robots transpaleta que utilizan centros de distribución como los de Amazon. Son los llamados AGVs (*Automatic Guided Vehicles*).

Ilustración 1: Pegasus, uno de los robots empleados en Amazon. Imagen obtenida de [2].

Con estos robots se agiliza el transporte de paquetes por el centro de distribución, acelerando el depósito a otras empresas de transporte que deben llevarlos a su destino final.

El anterior es solo uno de los ejemplos que existen, dado que la localización en interiores se emplea en muchos más campos, como puede ser por ejemplo un entorno peligroso, al que los humanos no puedan acceder (radiactividad, exposición a agentes químicos, temperaturas excesivas, etc.).

En el posicionamiento en interior hay siempre un factor común que es la precisión de la posición, es decir, la diferencia entre la posición real del robot, la que se podría medir directamente respecto a otro objeto con algún instrumento de medición y la que se está viendo en el software de lectura de datos asociado. Existen muchas soluciones: en comunicación para el posicionamiento existen UWB (Ultra-Wide Band), bluetooth, etc.; y en detección de obstáculos durante la ruta: visión artificial, ultrasonidos, infrarrojos, etc. La diferencia entre ellas radica en varios aspectos: su dificultad de implementación, el coste, las capacidades que ofrecen, los requisitos de funcionamiento, etc. Por ejemplo, para el caso de visión artificial, los objetos localizados quedarán fuera de rango cuando algún otro objeto se encuentre en la línea de visión, por lo que habrá que combinarlo con algún otro sistema.

Los sistemas de guiado automático siempre funcionan de la misma manera: se especifican una serie de puntos (llamados en seguimiento de rutas *waypoints*) por los que el robot debe pasar, y su diseño de software interno se encarga de procesarlos y alcanzarlos. La manera de alcanzarlos también debe ser definida, disminuyendo la velocidad del robot antes de alcanzar cada uno de ellos, u orientándolo de manera específica antes de terminar la maniobra. También la manera de especificar *waypoints* puede ser dinámica o predefinida, dependerá de la aplicación. Si el robot está generando un mapa de obstáculos de manera dinámica, él mismo será el encargado de crear la ruta que debe seguir (conociendo siempre el punto final e inicial, junto con algunas referencias de posición).

Para complementar los sistemas de radio que se utilizan en estimación de la posición se emplean unidades de medición inercial (en inglés, IMU). Con estas unidades se pueden hacer correcciones sobre la posición. Por

ejemplo, si mediante trilateración (ilustración 2) se ha estimado que la posición del robot ha cambiado pero la IMU no ha detectado cambio de velocidad significa que ha habido un error, y por tanto debe ignorarse la nueva posición (puede haberse interpuesto algún objeto o haberse perdido la conexión momentáneamente).

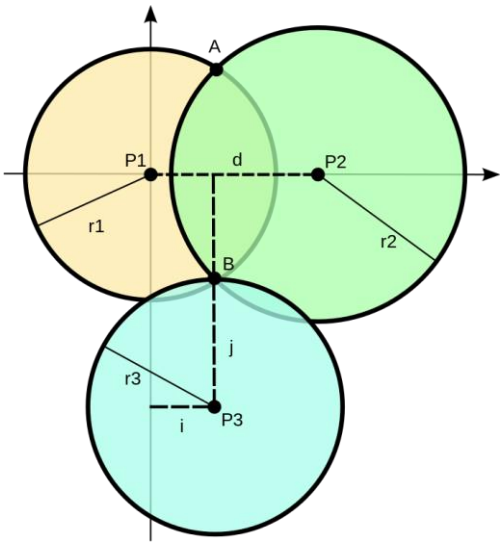


Ilustración 2: Técnica de localización por trilateración. Imagen obtenida de [3].

La estimación de la posición a partir de medidas de algunos métodos como trilateración o triangulación se combina con las anteriormente mencionadas IMUs, mediante el uso de los llamados filtros de Kalman [4]. Estos filtros tienen numerosos usos, entre los que destaca su uso en guiado, navegación y control de vehículos. Tienen su mayor relevancia en el control de naves espaciales.

Sobre la información anterior tratará el presente trabajo, el cual será dividido en varios puntos centrales, en los cuales se irán describiendo paso a paso los pasos seguidos:

- **Objetivos.** Se detallarán aquellos que se persiguen con el siguiente trabajo, haciendo hincapié en los detalles concretos que se tratarán.
- Antecedentes. Se darán una serie de datos relativos a los principios empleados y cómo se aplican al trabajo, sin entrar en profundidad.
- Puesta a punto del sistema de localización en interior. Previo al uso del sistema de localización que se emplea en el trabajo son necesarios ajustes para que el funcionamiento sea el correcto.
- Caracterización del sistema de localización. Se comprueban los errores que comete el sistema al localizar el objeto móvil, haciendo estudios estadísticos y varias pruebas adicionales bajo diferentes condiciones.
- Implementación de sistema de guiado autónomo dados puntos de una trayectoria predefinida. Se pretende diseñar un sistema de autoguiado del vehículo móvil usando datos de posición y medidas inerciales.

- Valoración económica del proyecto. Recogida de los precios de los equipos empleados, para poder ver el coste real de un sistema de seguimiento y guiado tal y como el que se ha implementado.
- Trabajo futuro y mejoras. Aplicaciones del sistema y posibles continuaciones del trabajo realizado, enfocado a futuros TFGs o TFMs.
- Conclusiones. Derivadas del resultado obtenido con el trabajo realizado, se analiza la consecución de los objetivos y la información adicional que se obtiene durante la realización del mismo.

1.2. Objetivos.

En este proyecto se concibe como objetivo principal la puesta a punto de un sistema de localización de un objeto móvil en interiores, basado en ultrasonidos y radiofrecuencia 433 MHz.

Como objetivos secundarios del proyecto se plantean los siguientes:

- Dibujado de la trayectoria del objeto móvil basada en las coordenadas devueltas por el sistema de localización.
- Comprobación de las desviaciones en las medidas frente a las coordenadas reales.
- Uso del sistema de localización para el guiado de un robot móvil.
- Búsqueda de límites de funcionamiento del sistema de localización. Determinación de condiciones de posición fiable o poco fiable.

Al final de la realización se valorará la consecución de los mismos, incidiendo en la profundidad alcanzada, así como las dificultades que se han ido teniendo en el desarrollo del proyecto.

2. ANTECEDENTES

En esta sección se describen los conceptos teóricos en los que se basa el sistema empleado en el proyecto, de una manera breve y suficiente para tener una idea general sobre su funcionamiento, sin entrar en excesivos detalles.

2.1. Uso de ultrasonidos en localización de objetos.

Su principio de funcionamiento es ampliamente conocido [5], se basa en emitir pulsos de frecuencia superior a la máxima audible por el oído humano (que tiene un límite de alrededor de 20 KHz), y medir la diferencia de tiempo entre su envío y recepción, pudiendo estimar con ello la distancia al objeto que los ha reflejado.

$$d = \frac{t_{send} - t_{recv}}{2} \cdot v_{aire} \qquad (1)$$

Donde:

- t_{send} es el instante de envío del pulso.
- t_{recv} es el instante de recepción del pulso.
- v_{aire} es la velocidad del sonido en el aire para la frecuencia y condiciones ambientales dadas.
- d es la distancia medida por el sensor.

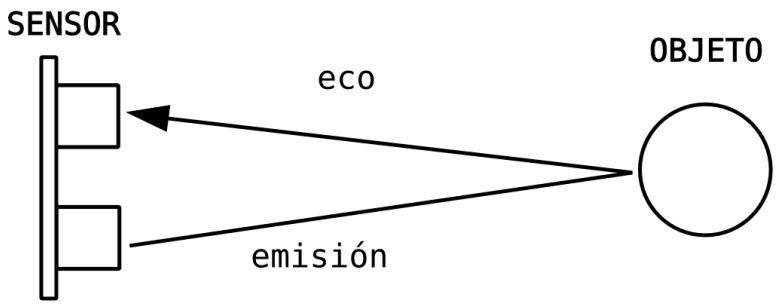


Ilustración 3: Funcionamiento del sensor de ultrasonidos. Imagen obtenida de [6].

La frecuencia que suelen usar suele estar en torno a los 40 KHz, aunque en el sistema que se emplea en el proyecto se usa una frecuencia de 31.9 KHz, como se verá en el apartado siguiente de puesta en funcionamiento.

El dispositivo de ultrasonidos puede ser emisor, receptor o ambos, en los dispositivos que componen el sistema empleado (se verá que son balizas) se incluyen 5 de ellos, junto con una antena RF que permite la transmisión de datos con el módem.

2.2. Técnicas de posicionamiento.

En este apartado se hará una breve introducción a algunas de las técnicas de localización que existen, destacando una de ellas, que será la que emplea el sistema que se usará en este proyecto.

- **Técnicas basadas en triangulación.** Es la que emplea el sistema que se usa en el proyecto. La triangulación consiste en tomar medidas respecto a fuentes de señal (en el caso de este proyecto serán balizas) desde la ubicación objetivo.

  Dentro de la triangulación hay varios métodos: tiempo de llegada (en inglés TOA), diferencia de tiempo de llegada (TDOA), método basado en la atenuación de señal, método basado en el tiempo de retorno de la señal, método de la fase de la señal recibida (ROTF) y ángulo de llegada (AOA).

  El método empleado por el sistema que se usa es *TOA*: con las medidas de distancia obtenidas desde las fuentes de señal se trazan circunferencias de radio la distancia medida, y el lugar geométrico donde se crucen tres circunferencias será el lugar donde se encuentra el objeto que se pretende localizar.

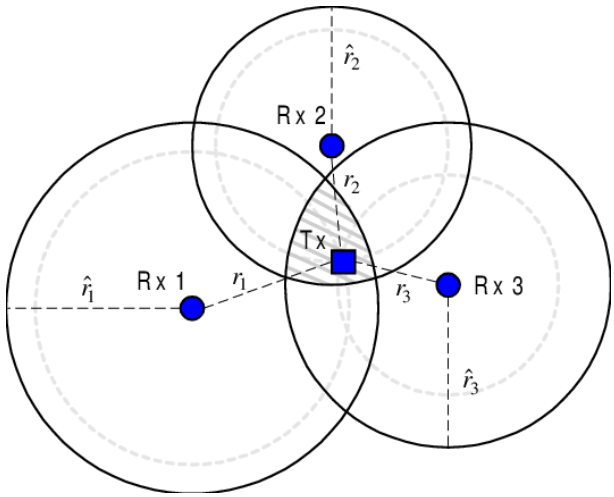


Ilustración 4: Funcionamiento de TOA. Imagen obtenida de [7].

  Respecto a los demás métodos [8] [9] [10]:

  - TDOA se basa en medir la diferencia de tiempo entre la llegada de dos señales entrantes hacia dos estaciones base (fijas).
  - Método basado en la atenuación de la señal. Se tiene en cuenta la diferencia de potencia entre la señal enviada y la recibida.
  - AOA. Existen muchos métodos para implementarlo. Se basa en medir las direcciones angulares desde un dispositivo colocado en una ubicación conocida (respecto a dispositivos fijos (balizas)).
  - ROTF. Basado en medir el tiempo desde que una señal es emitida hasta que se recibe, de manera similar al TOA.

- Técnicas basadas en el análisis del escenario. Se recopilan medidas sobre el escenario, y en tiempo real se comparan las medidas anteriores con las que se están obteniendo. Ejemplos: KNN, redes neuronales, SVM, SMP, métodos probabilísticos.

- Técnicas de proximidad. Se emplea una amplia red de antenas o sensores, y se mide la potencia que recibe la antena o sensor más cercanos, que recibirá la mayor señal posible.

2.3. Uso de RF para la transmisión de información.

Para la comunicación entre el dispositivo central (módem) y los demás terminales (dispositivos con una localización fija, en adelante llamados balizas fijas), se debe establecer algún tipo de comunicación. El módem es el elemento central que procesa los datos en crudo de los demás elementos (por ejemplo, en un sistema de localización, las distancias medidas), y es capaz de devolver a los dispositivos que están enviando datos su localización (u otros datos) basada en los datos que ha recibido.

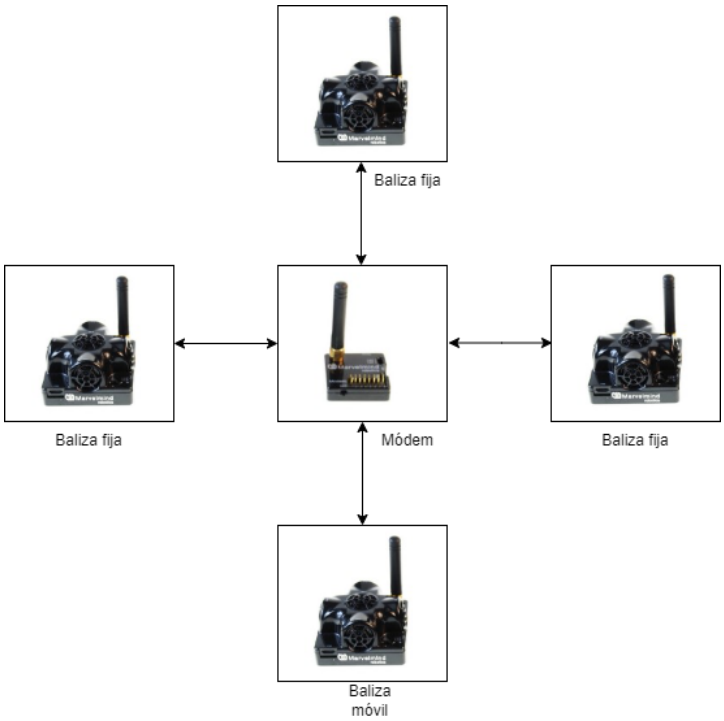


Ilustración 5: Esquema de comunicación de un módem con unas balizas radio.

Los ultrasonidos se usan para calcular distancias, pero estas deben ser procesadas para calcular mediante *TOA* la localización del objeto móvil (baliza móvil). El dispositivo encargado de procesarlas y enviar datos es el módem.

En el sistema que se emplea la comunicación módem-balizas se realiza mediante RF 433 MHz, una comunicación muy empleada en redes inalámbricas de corto alcance [11]. Esta banda de frecuencia tiene usos destacados en controles remotos, RFID, dispositivos de radar, radio y satélites *amateur,* etc.

Ilustración 6: Módulos comunes RF 433, receptor (izquierda) y transmisor (derecha). Imagen cedida por [12].

2.4. Fundamentos de guiado automático dadas coordenadas de destino y actuales.

Como se mencionó anteriormente, mediante el sistema que se emplea en el proyecto se tiene la capacidad de localizar un objeto móvil en el espacio (tanto en 2D como en 3D).

Si se tienen las coordenadas actuales y las de destino del objeto se puede implementar algún tipo de control que gobierne los accionadores de movimiento del robot (en el caso que se trata serán dos motores), y conseguir con ellos tanto el avance como el giro del robot (mediante diferencia de velocidades de rotación de las ruedas en caso de que sea de 2 ruedas, o en caso de que sea de 4 mediante avance o retroceso de las ruedas motrices y un control de la dirección con las otras dos ruedas mediante servomotor o similar).

Dado que la velocidad de avance y el giro deben ser controlados para no tener deslizamientos, se hace imprescindible implementar una estrategia de control adecuada, buscando una respuesta concreta (inicio de trayectoria suave, corrección de desvío de la ruta, velocidad limitada, etc.).

Por ejemplo, puede implementarse un controlador encargado de orientar el robot dados los datos de la IMU mencionada anteriormente, y que otro controlador se encargue del avance. Dado que la trayectoria no será ideal (leves diferencias de velocidad en los motores, excentricidades, etc.) también habrá que realizar correcciones de orientación durante el avance, cambiando entre ambos controladores.

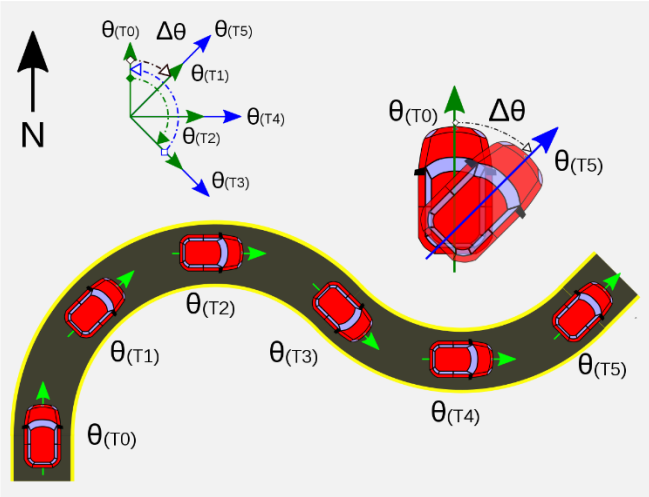


Ilustración 7: Guiado de un vehículo por una trayectoria. Imagen obtenida de [13].

La implementación del sistema de guiado se verá en un apartado posterior de manera detallada, dando por ahora una breve pincelada de sus fundamentos.

2.5. Filtrado y fusión.

Filtrado

Los datos que devuelven los sensores (giroscopio, brújula, acelerómetro) suelen ser datos con ruido, que pueden provocar un malfuncionamiento de los algoritmos que hagan uso de ellos.

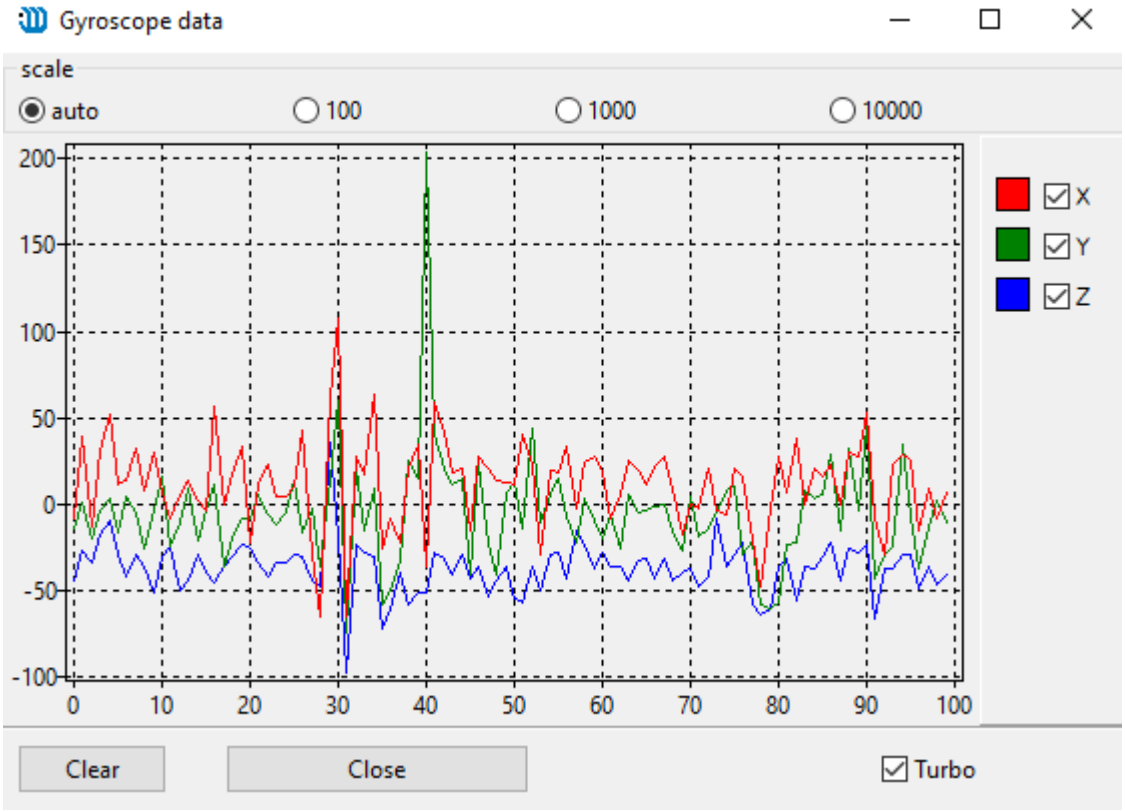


Ilustración 8: Ejemplo de datos RAW (en crudo) leídos de un giroscopio.

Este ruido es una desviación respecto al valor real (que nunca será conocido), con lo cual la única acción a a realizar con los datos es intentar estimar este valor real atendiendo al ruido de la señal y el ruido auxiliar que pudiera adicionarse a esta señal. Para cuantificar estos ruidos por ejemplo puede usarse su varianza.

Un filtrado muy usado es ya mencionado filtro de Kalman [14], que hace uso de estas varianzas para estimar un valor de salida del filtro. Es un filtro iterativo, que dará la estimación cuando se hayan calculado sus ecuaciones varias veces sobre el valor que se pretende estimar.

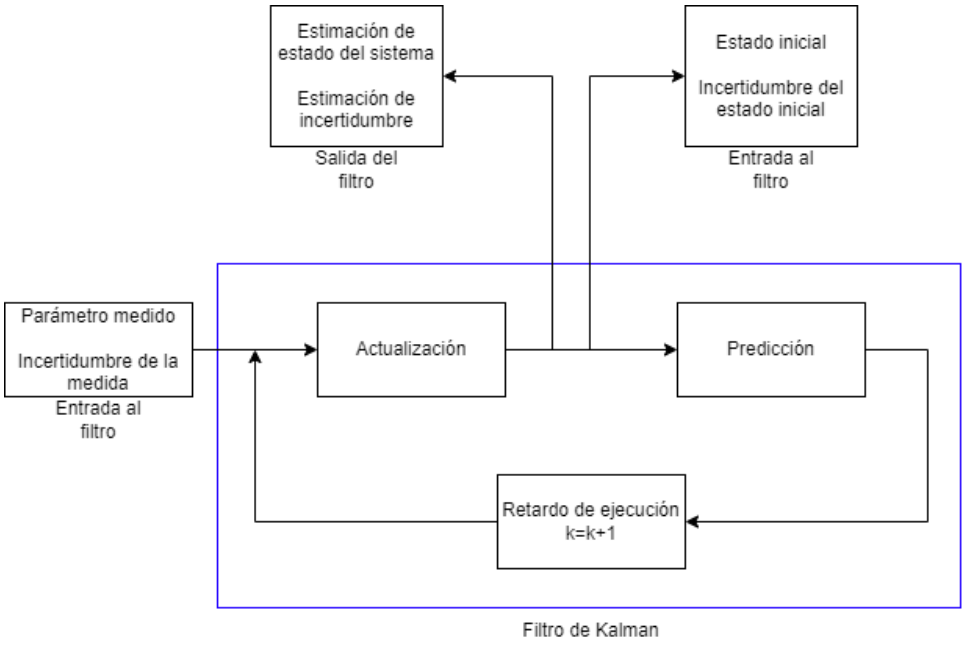


Ilustración 9: Filtro de Kalman en diagrama de bloques. Inspirado en [15].

En el algoritmo de este proyecto se incluye una simplificación del filtro de Kalman [16] para una única señal, y viene dado por:

Fase de predicción:

$$MSE_{min}(k) = a^2 \cdot MSE_{min}(k-1) + \sigma_v^2 \qquad (2)$$

$$K(k) = \frac{MSE_{min(k)}}{\sigma_n^2 + MSE_{min(k)}} \qquad (3)$$

Fase de corrección:

$$S(k) = S(k) + K(k) \cdot [x(k) - S(k-1)] \qquad (4)$$

$$MSE_{min}(k) = (1 - K(k)) \cdot MSE_{min}(k) \qquad (5)$$

Donde:

- K: ganancia de Kalman.
- k: instante de cálculo.
- k-1: instante anterior.
- a: constante del filtro, debe ser elegida, por ejemplo, podría ser la primera observación (x(0)).
- σ_v^2: varianza del ruido de la señal de entrada.
- σ_n^2: varianza del ruido adicional a la señal de entrada.
- S: predicción.
- x: valor observado.
- MSE_{min}: error cuadrático medio medio mínimo.

Cuando se pretende actuar sobre varias señales o variables se emplea el filtro de Kalman matricial, que funciona de la misma manera, con una fase de predicción y otra de corrección o actualización, esta vez usando en lugar de variables unidimensionales matrices que constituyen la expresión del sistema en espacio de estados.

Fusión

Una vez que se han filtrado las señales de los sensores, es necesario estimar un valor teniendo en cuenta varios valores dados por el filtrado a distintos sensores, que, en teoría, por estar midiendo la misma variable, deberían devolver el mismo valor (hecho que no ocurrirá).

Para saber la lectura de qué sensor debe ser la que debe tener una mayor ponderación en la estimación final se puede emplear por ejemplo la covarianza en base a un experimento: si se rota el objeto 45º, se pueden comparar estos 45º con los que devuelve el filtrado de: los datos de la brújula, los datos de la integración del giroscopio y finalmente los datos devueltos por el ángulo entre la posición en x, y anterior y una nueva posición, tras haber avanzado un poco.

Para ponderarlos, por ejemplo, se podría hacer de la manera que sigue:

$$\hat{\theta} = \frac{\frac{\hat{\theta}_{FK1}}{cov_{FK1}} + \frac{\hat{\theta}_{FK2}}{cov_{FK2}} + \frac{\hat{\theta}_{FK3}}{cov_{FK3}}}{\frac{1}{cov_{FK1}} + \frac{1}{cov_{FK2}} + \frac{1}{cov_{FK3}}} \qquad (6)$$

Donde:

- $\hat{\theta}_{FKi}$ es la estimación de ángulo del filtro de Kalman i.
- $\hat{\theta}$ es el ángulo estimado tras ponderar las medidas.
- cov_{FKi} es la covarianza entre los valores del resultado y los valores reales teóricos del experimento.

2.6. Estimación del ángulo de giro del eje Z (yaw) mediante cuaternios.

Con los datos filtrados que se han visto en el apartado anterior se puede calcular el ángulo de giro del eje Z (llamado *yaw*), pero el sistema de posicionamiento que se emplea también incorpora el cálculo interno de los llamados cuaternios o cuaterniones [17], números hipercomplejos que tienen 4 componentes, una parte real y 3 partes imaginarias, cada una con una dirección, con lo cual son capaces de indicar una dirección de manera tridimensional.

Si se tiene el valor de las componentes del cuaternio se puede calcular el ángulo *yaw* de la manera que sigue [18]:

$$sin_y = 2 \cdot (q_w \cdot q_z + q_x \cdot q_y) \qquad (7)$$

$$cos_y = 1 - 2 \cdot (q_y^2 + q_z^2) \qquad (8)$$

$$yaw = atan2(sin_y, cos_y) \qquad (9)$$

Teniendo como resultado el valor del *yaw* en rad.

2.7. Controladores PI. Aproximaciones discretas.

En este proyecto se emplearán controladores PI, los cuales, para ser implementados de manera discreta requieren de una discretización [19] de la forma continua de la acción integral.

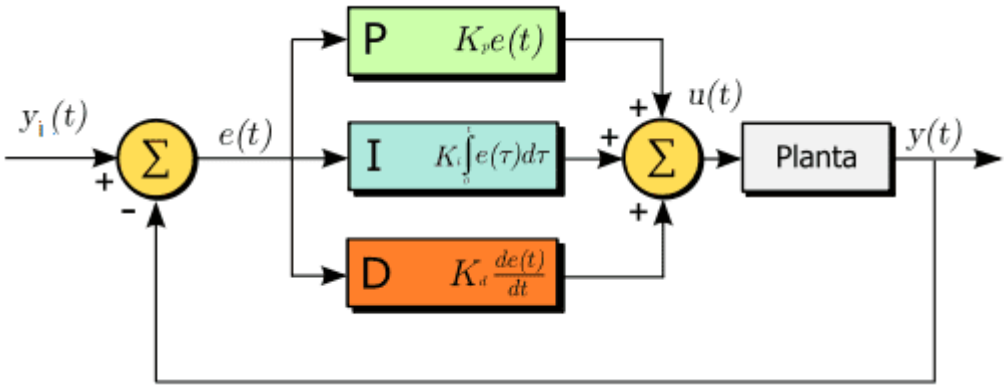


Ilustración 10: Diagrama de bloques de un PID. Imagen obtenida de [20].

De la ilustración anterior solo se emplean las acciones P e I, y la acción de control u, se aplica al sistema controlado, llamado en el esquema "Planta".

Existen muchas aproximaciones:

- *Backward Euler*.
- *Forward Euler*.
- En rampa.
- Bilineal o de Tustin.

En base a otras implementaciones se ha comprobado que la aproximación por *Backward Euler* ofrece muy buenos resultados en la mayoría de casos, con lo cual es la que se implementará en el proyecto, y tiene inicialmente la siguiente forma:

$$p(k) = Kp \cdot e(k) \qquad (10)$$

$$i(k) = i(k-1) + Ki \cdot T \cdot e(k) \qquad (11)$$

Donde:

- p es la acción proporcional (no necesita aproximación).
- e es el error (referencia-lectura).
- Kp es la constante proporcional del controlador.
- Ki es la constante integral del controlador.
- T es el periodo de cálculo del controlador.

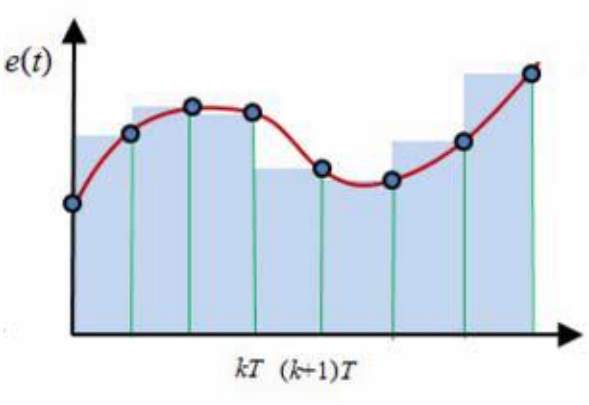


Ilustración 11: Aplicación de Backward Euler a una señal. Imagen obtenida de [19].

3. DESARROLLO DEL SISTEMA DE LOCALIZACIÓN

3.1. Descripción del hardware empleado.

El equipo que se emplea en el proyecto es un set de balizas radio con comunicación RF 433MHz combinado con ultrasonidos, tal y como el que se observa en la siguiente imagen:

Ilustración 12: Set de balizas SUPER-NIA-3D. Imagen obtenida de [21].

El uso de RF les permite la transmisión de datos entre balizas (son los 5 dispositivos iguales que se ven en el set de la ilustración 12) y módem (el dispositivo más pequeño que conforma el set de la ilustración 12), el cual coordina todos los datos que recibe, permitiendo saber la ubicación relativa de una baliza a otra procesando los datos de ultrasonidos que recaba. Por tanto, el sistema requiere que el módem esté encendido, sin él las balizas no comienzan su funcionamiento.

El objetivo de estas balizas es su colocación en un espacio donde haya un objeto móvil que se desee localizar, en este caso será un robot llamado Jetbot de la mano de Waveshare, combinado con la plataforma de desarrollo Jetson Nano, de la mano de NVIDIA.

Las balizas permiten dos arquitecturas, NIA, e IA. Estas arquitecturas definen la manera de funcionar y comunicarse entre ellas, y vienen determinadas por el set que se adquiera. En este caso se usará la arquitectura NIA, dado que es sobre la que funcionan los equipos adquiridos. Esto se debe tener en cuenta en la puesta en funcionamiento de las mismas.

El fabricante señala que para la localización 2D de un objeto es necesario tener al menos 3 balizas: la baliza objetivo o móvil (que se pretende localizar), y al menos dos más. con una línea de visión directa sobre la baliza objetivo. Para el caso 3D, en el que el sistema también puede devolver la altura a la que se encuentra la baliza objetivo, es necesario colocar una baliza estática más (teniendo 4 en total).

La colocación de las balizas estáticas debe ser, tal y como se menciona anteriormente, en una ubicación desde la que se tenga una línea directa de visión sobre el objeto localizado. De no tener una visión directa por haber objetos intermedios se producirán errores en la medida, los cuales serán observados en apartados posteriores.

En cuanto a características de interés, se construye una tabla con las más destacadas, que hacen elegir este sistema frente a otros muchos que hay en el mercado:

Característica	Valor
Número máximo de balizas comunicadas	250
Precisión de la localización	±2 cm
Método de localización de la baliza móvil	*TOF*
Puede construir el mapa de localización de las balizas de manera automática	Sí, en el caso de arquitectura *NIA* y casos simples de localización
Tiempo de construcción del mapa	De 7 a 10 segundos
Batería de las balizas fijas y móvil	1000 mAh, Litio
Tiempo de actualización de la localización	De 1 a 25 Hz
Distancia recomendada entre balizas	Máx. 50 m
Área de cobertura	Hasta 1000 m^2 con el kit que se emplea. Posibilidad de áreas más grandes creando sub-mapas.
Tiempo de funcionamiento	De 2 días a varios meses, depende del uso: tiempo en stand- by y tiempo transmitiendo posición, frecuencia de actualización, frecuencia de ultrasonidos, etc.

Tabla 1: Resumen de características del sistema.

La denominación “casos simples de localización” se refiere a la ausencia de un entorno ruidoso, o distancias muy elevadas.

En el caso que se trata en este proyecto, compuesto por 3 balizas fijas y una móvil, el mapa podrá ser autogenerado (entendiéndose como autogenerado que es capaz de calcular las distancias entre balizas fijas por sí mismo, sin introducirlas manualmente), aunque podrían introducirse las posiciones, para ello existe una opción en el *dashboard* (herramienta que permite la calibración, verificación, y puesta en marcha del sistema de posicionamiento), llamada *table of distances*. Si en esta tabla se ve alguna celda en color amarillo/rojo esta distancia no se está midiendo correctamente, y debe ser introducida manualmente. Ejemplo:

HIDE	6	22	66	77
6		8.000	4.200	11.400
22	8.000		13.100	6.500
66	4.200	13.100		10.800
77	11.400	6.500	10.800	

HIDE	6	22	66	77
6		8.000	27.378	5.054
22	8.000		28.688	18.739
66	33.772	29.794		18.741
77	17.315	7.585	3.522	

Ilustración 13. Ejemplos de tablas de distancias correcta (izquierda) e incorrecta (derecha). Imagen obtenida de [21].

La introducción de distancias se hace al pulsar en la celda, colocando en su interior la distancia entre la baliza marcada por el número de columna y la marcada por el número de fila.

3.2. Puesta en funcionamiento del sistema de posicionamiento.

Firmware

Como ya se ha mencionado anteriormente, la arquitectura del sistema determina el modo de funcionamiento de las balizas. Se debe tener en cuenta de cara a cargarles el firmware antes de su primer uso.

Con anterioridad a la actualización se precisa la carga de todas las balizas radio, para poder poner en funcionamiento el sistema al terminar todas las actualizaciones. Para la carga del firmware se conectan mediante el microUSB que poseen, previamente habiendo colocado en la posición ON la parte del microswitch *Power*, si esta opción no funciona, se debe colocar en ON la otra mitad del microswitch (modo DFU).

MarvelMind incluye la ya mencionada herramienta llamada *dashboard* con la que se pueden realizar varias funciones, entre ellas, actualizar fácilmente el firmware de las mismas. En este caso se ha cargado el firmware hw49 tanto para *beacons* (balizas) como para el módem. Para saber qué versión de hardware se emplea se debe consultar la etiqueta debajo de cada baliza.

La pestaña *"Firmware"* permite la actualización de las mismas. Puede verse en la ilustración siguiente, donde se muestra la herramienta en su pantalla principal.

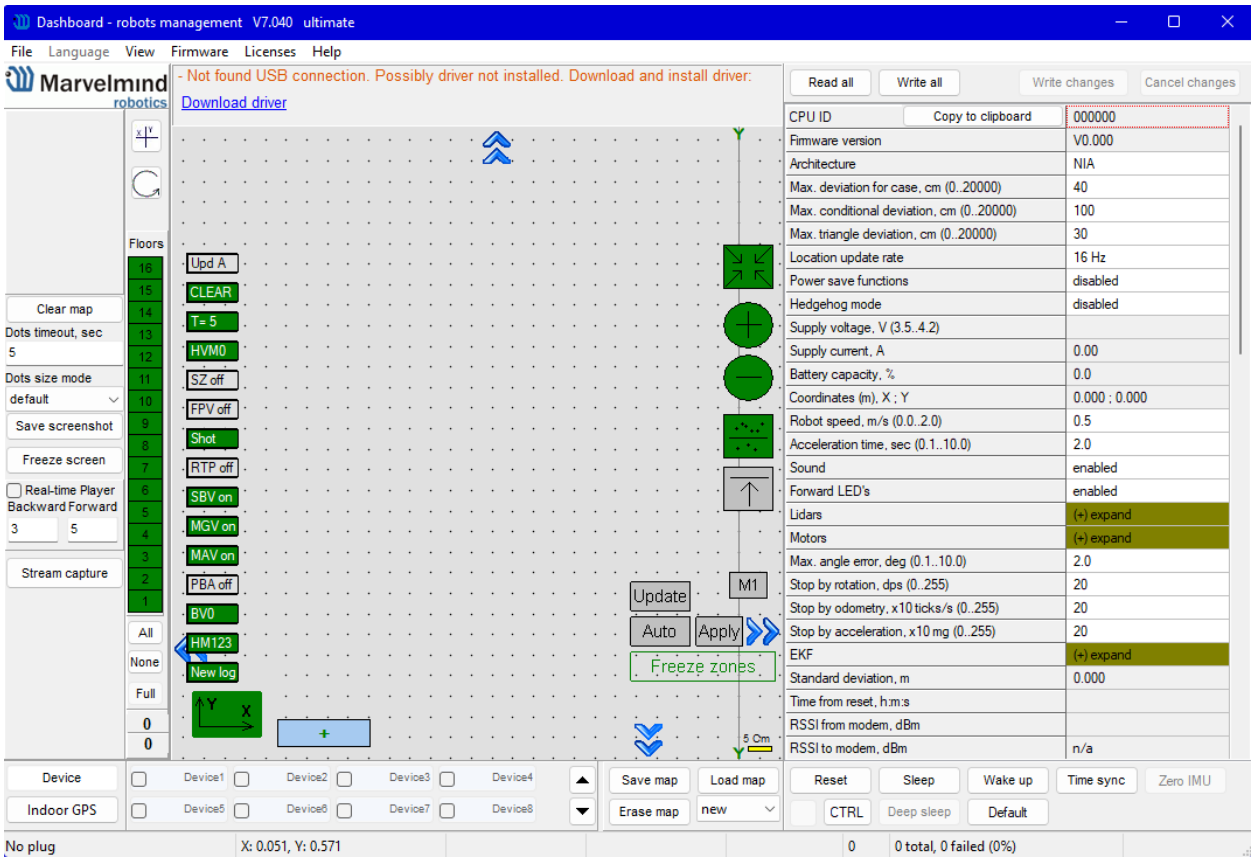


Ilustración 14: Herramienta dashboard.

Distancias relativas

En cuanto a las distancias que se mencionaron anteriormente, si se produce un error de medidas entre las balizas, se debe hacer de forma manual la medida de distancia entre ellas, hasta conseguir que fondo de la celda se vea en color blanco, momento en el que los datos devueltos por ultrasonidos e introducidos coincidirán.

Mapas, sub-mapas, zonas de servicio y zonas de entrega *(handover zones)*

El mapa es la unidad espacial que utiliza el sistema para referirse a una visión global de la instalación: todas las balizas fijas y su distribución en sub-mapas, junto con la baliza móvil localizada.

En cuanto a los sub-mapas, son unidades más pequeñas que luego formarán el mapa concreto, y sirven para seleccionar las balizas que van a funcionar cuando se quiera localizar el objeto dentro de la zona que cubren. Se pueden construir mapas tanto de una sola dimensión hasta mapas 3D, empleando hasta 4 balizas para ello. Pueden construirse mapas distintos que usen varias veces las mismas balizas: esto ayuda a casos en los que haya obstáculos entre una de ellas y quiera seguirse usando el mayor número de ellas cercanas, sin cambiar a otras distintas.

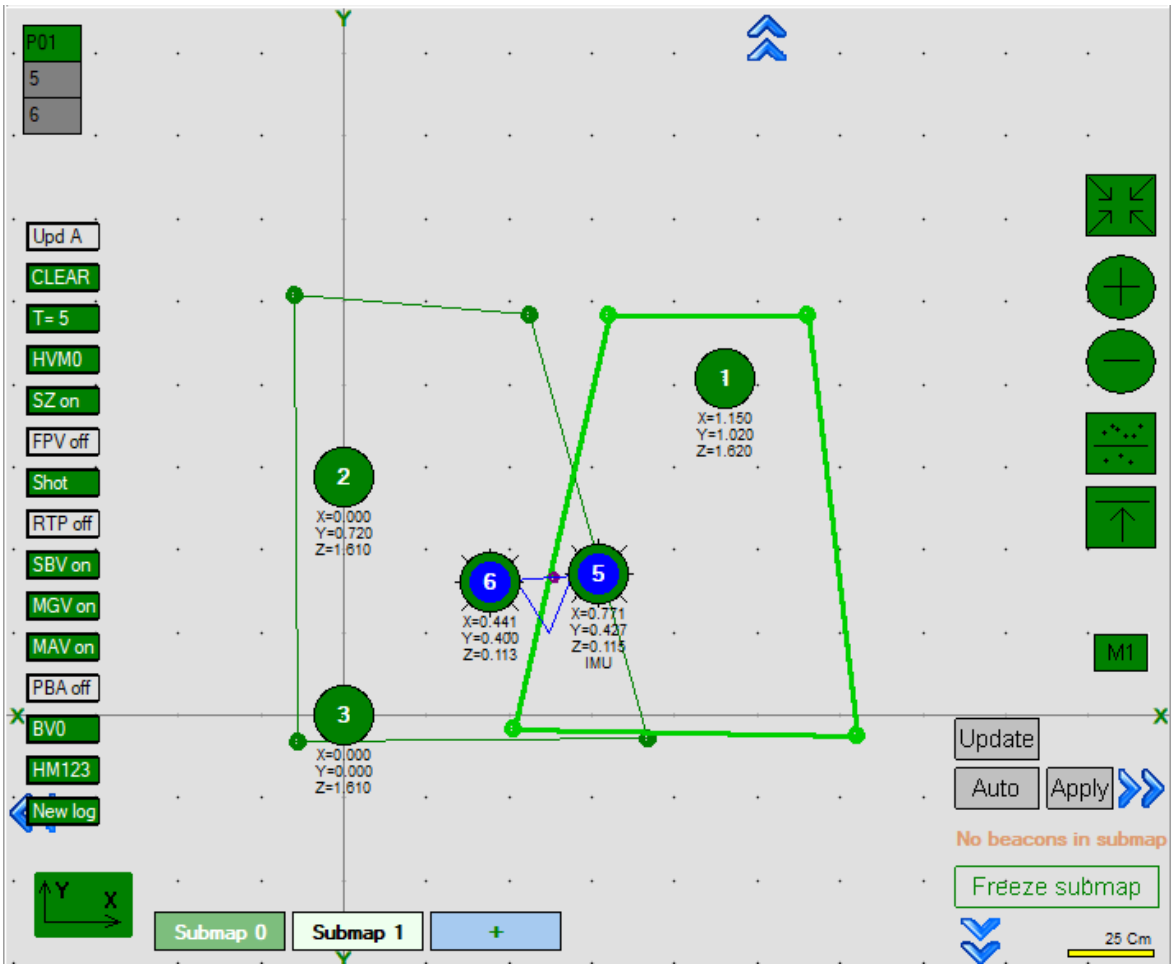


Ilustración 15: Vista de configuración con 2 submapas.

La zona de servicio es el área que cubre el sub-mapa y a la hora de crearlo será necesario también especificar el área que cubre ésta también. En la ilustración 15 pueden verse 2 sub-mapas (*"Submap 0"* y *"Submap 1"*), estando seleccionado en ese momento el sub-mapa 1. La zona delimitada por puntos es su zona de servicio. La del sub-mapa 0 es la que se ve a la izquierda, delimitada por una línea de menor grosor.

En cuanto a las zonas de entrega son las zonas intermediarias entre el cambio entre un sub-mapa y otro, y su función es permitir que cuando se cambie del cálculo de la posición del objeto móvil de un sub-mapa a otro la diferencia sea muy leve, evitando un salto brusco de coordenadas que ocurriría hasta que las nuevas balizas devolvieran la nueva posición del objeto móvil. En la ilustración 15 puede verse una zona de entrega entre los sub-mapas 0 y 1. El fabricante señala que la zona debe ser lo suficientemente grande para que a la frecuencia de actualización que está funcionando el sistema se produzcan al menos 4 actualizaciones de posición. Esto se deberá comprobar sobre el campo, de manera experimental.

El mapa completo es el que alberga todo lo que se ve en la ilustración 15.

ID de las balizas fijas y móvil

Dentro de la red RF de comunicación que se establece entre módem y balizas cada una de ellas tiene un ID asociado, que viene preasignado. Se puede usar o bien la predefinida o cambiarla según preferencia, por ejemplo, para tener un ID bajo y no tener que navegar en el *dashboard* a los números más altos.

En el caso que se ocupa, se han usado los IDs 1,2,3 para las balizas fijas y el ID 5 para la baliza móvil.

Activación de la IMU

Para que los datos de la IMU sean devueltos junto a los de posición, la unidad IMU de la baliza móvil debe ser activada en el *dashboard*. También existe la opción de calibrar esta unidad en caso de que los datos que devuelva sean muy incorrectos (cambios muy bruscos entre valores, valores excesivamente altos, medición fantasma (es decir, valores cambiando en ausencia de cambio real), etc).

Data-rate de RF (radio profile)

Un parámetro que debe ser de imprescindible configuración: si el valor que se emplea en alguna de las balizas es distinto al de las demás ésta no se va a comunicar con las demás.

Para la configuración inicial que se ha establecido se ha empleado un enlace con un *datarate* de 38kbps, en la interfaz de usuario se llama "radio profile". La configuración de cada baliza se hace individualmente antes de construir la red de balizas.

Frecuencia de ultrasonidos

Algunas balizas permiten su configuración, pero otras tienen la frecuencia preestablecida. Para este sistema, la frecuencia predefinida es de 31.9 kHz, la cual es variable desde el *dashboard*, pero se deja en 31.9 kHz.

Baliza *hedge*

Para saber qué baliza es la móvil y cuáles son las fijas, en el momento de configurar los parámetros anteriores en cada una también será necesario especificar en el campo "Hedge" si se quiere que la baliza sea *hedge* o no.

La denominación "hedge" hace referencia a una baliza que se quiere localizar. En el sistema que se emplea, en este caso será la baliza IMU, que se ha configurado con ID 5.

Uso de los scripts de Python

Aunque con la herramienta *dashboard* se tienen muchas opciones, como pueden ser la vista de la ruta actual de la baliza monitoreada, la posición actual de la misma, o un CSV con las coordenadas de la ruta que se va realizando, se hace necesario disponer de algún script que permita poder leer la posición en un instante, para tomar decisiones sobre ella, o interpretar su valor.

Para esto, MarvelMind ha desarrollado varios ficheros [22] escritos en Python, que se encargan de leer la posición de la baliza y devolverla. Estos ficheros permiten interpretar los datos si, por ejemplo:

- Se pretende hacer una fusión de datos haciendo uso de un filtro de Kalman aplicado a las lecturas de posición y las lecturas de la IMU de la baliza móvil.
- Se pretende dibujar una trayectoria mediante cualquier otro software: Matlab, ROS, ROS2, etc.
- Se quiere comparar la lectura que da la baliza frente a una lectura manual parametrizada para calcular el error que comete en el cálculo de la posición.
- Se quiere comprobar cómo afecta la interposición de obstáculos en la línea de visión de algunas balizas.

En los sucesivos experimentos se detallará el código empleado, así como las adiciones al mismo que permitan el funcionamiento que se observará.

3.3. Caracterización de las balizas radio.

En esta sección se hará una serie de pruebas al sistema, para ver cómo se desenvuelve en diferentes situaciones.

3.3.1. Prueba de localización con líneas de visión libre.

En este capítulo se hará una prueba sencilla de localización, teniendo las 3 balizas fijas con una línea de visión directa sobre la móvil, y esta se irá variando manualmente dentro de una pequeña área delimitada. Se comparará el incremento de posición medido por la baliza con el real.

También se va a comprobar, estando en una posición fija, la desviación típica que tienen las medidas, así como su varianza, con objeto de poder tener una idea del error admisible en la medida que podrá tenerse en futuros usos, y su mejora mediante el uso de filtrado.

Ilustración 16: Vista de la ubicación automática de las balizas sin configurar posiciones manualmente.

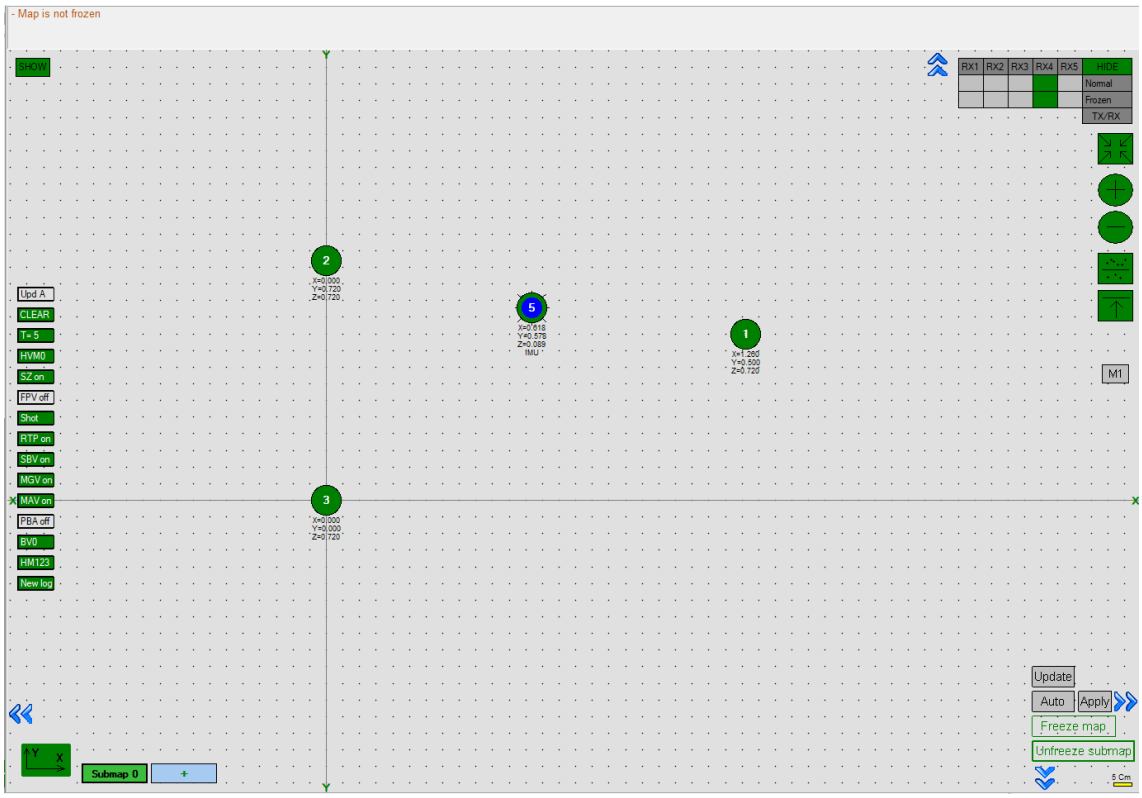


Ilustración 17: Vista de la ubicación tras insertar las coordenadas de forma manual en cada baliza.

La introducción manual de la ubicación de las balizas se ha hecho teniendo en cuenta que el centro de la baliza es el origen de coordenadas, y que su altura se mide desde la base.

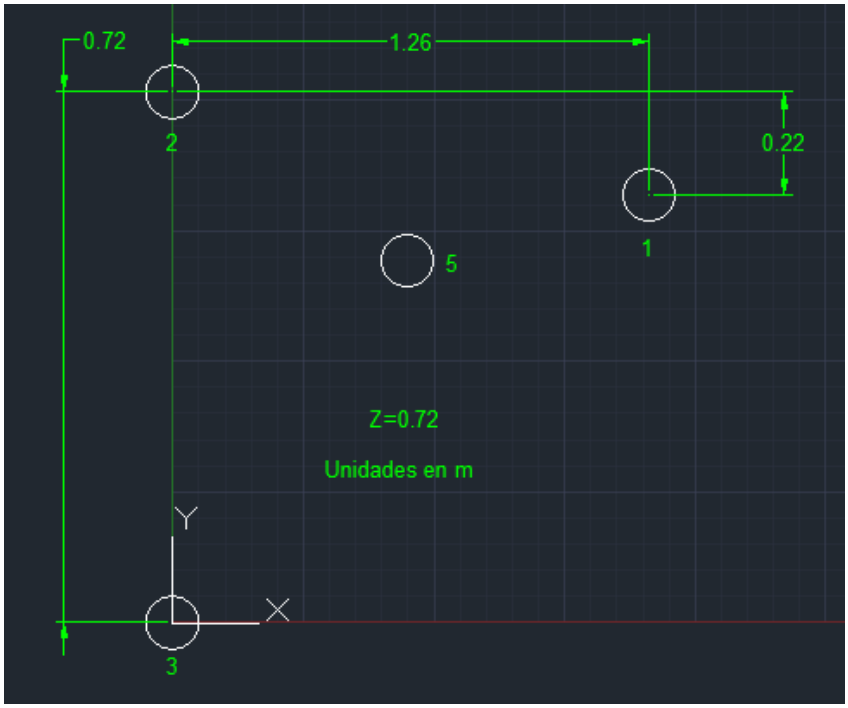


Ilustración 18: Esquema de colocación de las balizas para la prueba diseñado en AutoCAD.

Para la colocación que se ha esquematizado anteriormente su colocación real es la siguiente:

Ilustración 19: Colocación de las balizas para el ensayo, donde para Bi, i es el ID de la baliza.

La primera prueba que se va a realizar es la desviación típica que experimentan las medidas de posición que devuelve la baliza, junto con su varianza. Para ello, hay dos opciones: o bien se usa directamente el *dashboard*, que guarda la posición de la baliza hedge y los datos de la IMU, o bien usar los scripts dados por MarvelMind para la obtención de posición [21]. Se usará la segunda opción. Para obtener la posición y poder procesarla lo primero es ver cómo se devuelve. Usando el script "example.py" que se encuentra en el directorio "marvelmind.py-master" puede verse cómo se obtiene la posición añadiéndole unas leves modificaciones:

```
23
24              if (hedge.positionUpdated):
25                  hedge.print_position()
26                  print("Ahora se imprime posicion")
27                  posicion=hedge.position()
28                  print(posicion)
29                  sleep(2)
30                  print("Fin de impresión de posicion")
31                  #writecsvobj.writerow(hedge.position())
32
33              if (hedge.distancesUpdated):
34                  hedge.print_distances()
35
36              if (hedge.rawImuUpdated):
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Ahora se imprime posicion
[5, 0.614, 0.257, 0.091, 0, 579130]
Fin de impresión de posicion
Raw IMU: AX:0, AY:0, AZ:0,   GX:0, GY:0, GZ:0,   MX:0, MY:0, MZ:0,   at time T:
 581.411
Hedge 5: X: 0.614, Y: 0.256, Z: 0.091, Angle: 0 at time T: 581.096
Ahora se imprime posicion
[5, 0.614, 0.256, 0.091, 0, 581096]
Fin de impresión de posicion
```

Ilustración 20: Vista del formato de impresión de la posición.

Como puede verse se obtiene un vector con el ID de la baliza móvil, su posición en X, en Y, y en Z, junto con su ángulo y el tiempo en el que se hizo el registro del dato.

Para guardar los datos en CSV se usa el módulo csv de Python, creando una lista que luego se escribe a un fichero de nombre "lastPathID5.csv". Este fichero tendrá inicialmente 100 elementos, con los cuales se calcularán la desviación típica y varianza en X, Y, Z.

Los valores calculados que se obtienen son:

	Varianza [m^2]	Desviación típica [m]
X	6.8798e-07	0.00082945
Y	2.0908e-06	0.001446
Z	1.3636e-06	0.0011677

Tabla 2: Varianza y desviación típica para 100 valores.

Puede verse que tanto la varianza y desviación típica tienen valores muy bajos. Estos resultados se mantienen incluso para una muestra de datos menor, ya que podría pensarse que al ser un número tan elevado de datos las diferencias positivas se compensan con las negativas:

	Varianza [m^2]	Desviación típica [m]
X	2e-07	0.00044721
Y	1.3e-06	0.0011402
Z	8e-07	0.00089443

Tabla 3: Varianza y desviación típica para 5 valores.

Como puede observarse, los indicadores se mantienen en el mismo orden. Dado que la baliza lleva bastante tiempo en el mismo lugar, puede pensarse que al cambiar de lugar las variaciones van a ser más distintas hasta que se mantenga estable. Se hace la comprobación moviéndola:

	Varianza [m^2]	Desviación típica [m]
X	2e-07	0.00044721
Y	2e-07	0.00044721
Z	0	0

Tabla 4: Repetición del ensayo anterior con un movimiento de la baliza.

Puede comprobarse que las medidas de posición para este entorno estable son muy poco dispersas, haciéndolo ideal para un seguimiento en estas condiciones. En posteriores apartados se comprobará el efecto de obstruir la línea de visión.

Tras terminar los ensayos anteriores se preparará un área delimitada en la que probar la baliza móvil.

Se hará circular a lo largo de un cuadrado de 30x30 cm, medidos manualmente, y se comparará con los datos que se irán registrando a medida que se realiza el experimento, dibujando posteriormente también la trayectoria

que se ha seguido.

Ilustración 21: Cuadrado de 30x30 cm donde se colocará la baliza, moviéndose a cada una de las cuatro esquinas.

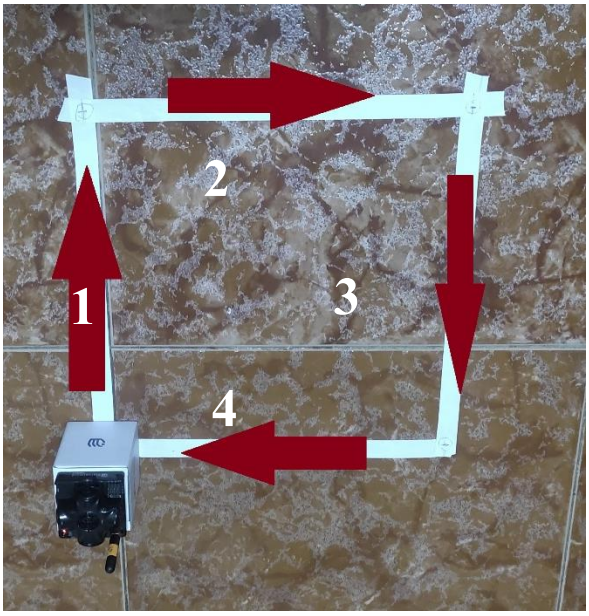


Ilustración 22: Trayectoria que seguirá la baliza móvil.

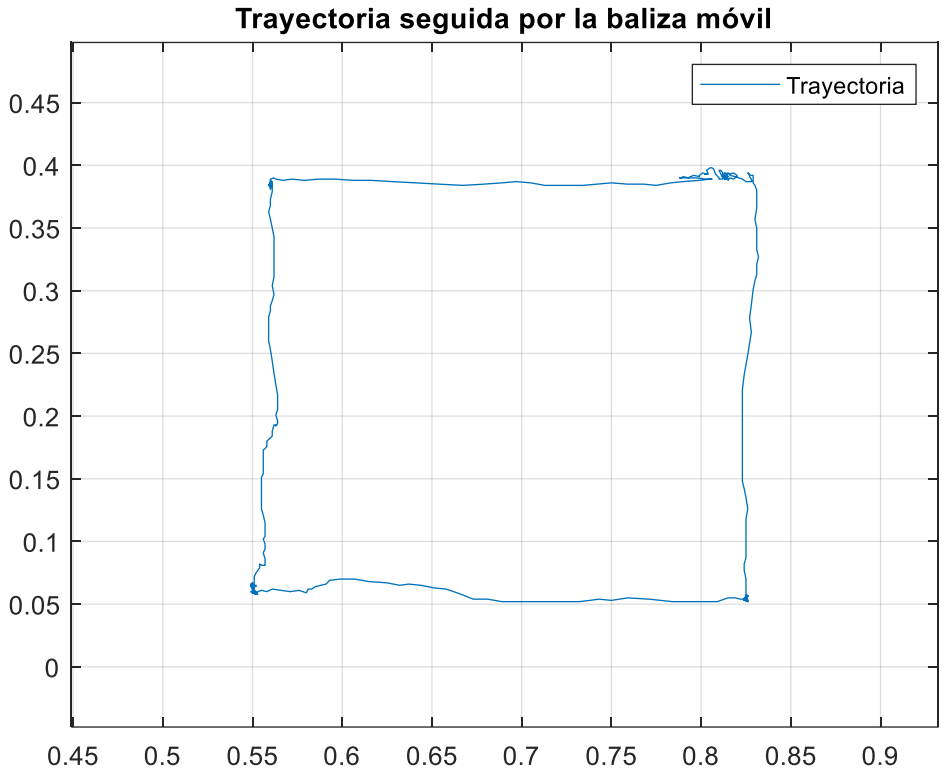


Ilustración 23: Vista de la trayectoria que ha seguido la baliza móvil.

Viendo los datos obtenidos, para el primer avance, hay una diferencia en Y de 0.328 m, frente a la medida real de aproximadamente 0.3 m. En cuanto al avance en X para el segundo movimiento hacia la derecha, la diferencia entre medidas es 0.268 m. En el tercer movimiento la diferencia en Y es de 0.339 m, y finalmente para el movimiento de regreso al punto inicial, la diferencia es de 0.275 m. Reuniendo todas las cifras en una tabla:

	Xi [m]	Xf [m]	Yi [m]	Yf [m]	$\Delta X\ [m]$	$\Delta Y\ [m]$
Tramo 1	0.551	0.562	0.059	0.39	0.11	0.331
Tramo 2	0.562	0.829	0.39	0.392	0.267	0.02
Tramo 3	0.829	0.825	0.392	0.058	0.04	0.334
Tramo 4	0.825	0.551	0.058	0.059	0.274	0.01

Tabla 5: Valores obtenidos durante el experimento.

Teniendo en cuenta que el cuadrado es de aproximadamente 30 cm de lado, puede comprobarse que se obtiene un error de aproximadamente 3 cm en el movimiento, acercándose mucho al anunciado de ±2 cm.

3.3.2. Pruebas de localización con alguna de las balizas con línea de visión obstruida.

En esta ocasión se verá cómo cambia la varianza y la desviación típica de las medidas frente a la real en varios casos de prueba.

Para la continuación de estos experimentos, por razones de desplazamiento, el escenario de prueba ha cambiado

respecto al de la ilustración 19, y ahora se deben reajustar las balizas tanto en altura como en posición. La ubicación ahora es la siguiente:

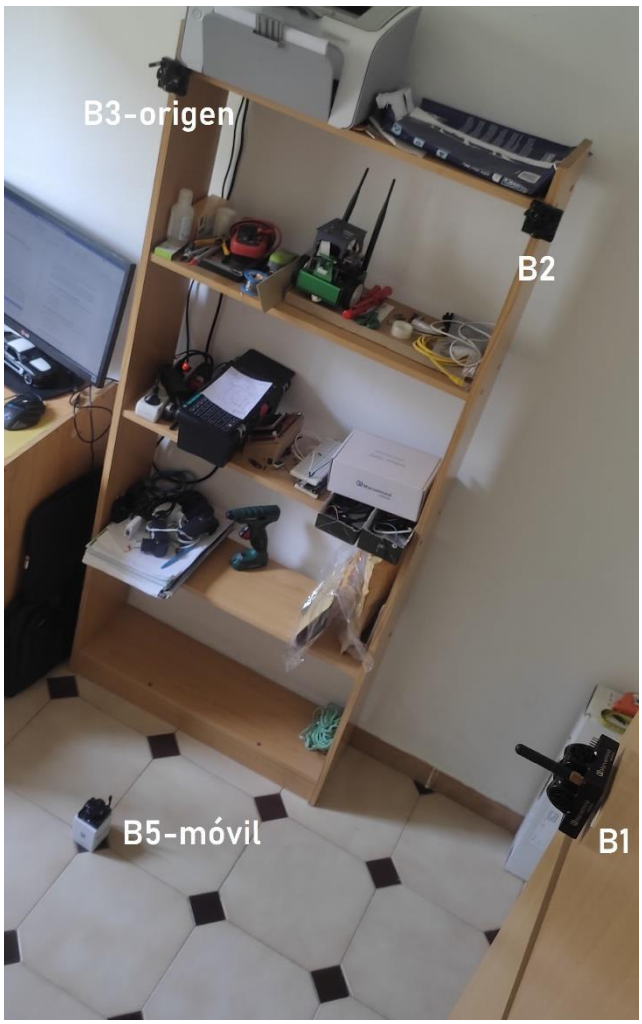


Ilustración 24: Vista superior de la nueva colocación de las balizas para los sucesivos experimentos.

Estando también el esquema de colocación que se mostraba anteriormente en la ilustración 18 también cambiado:

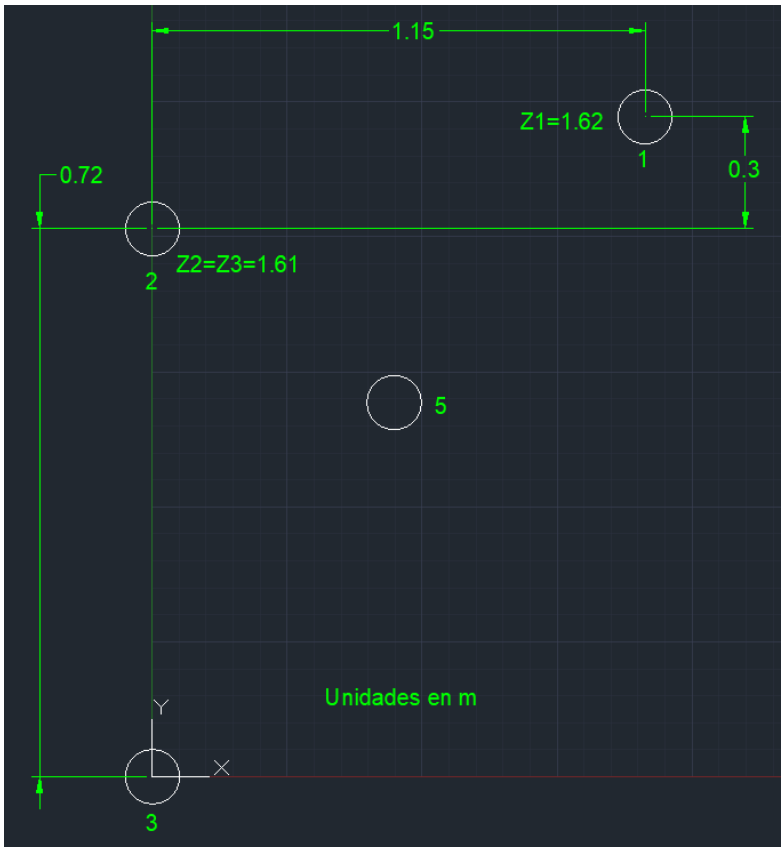


Ilustración 25: Esquema de la nueva localización de las balizas.

Introduciendo de nuevo al programa las nuevas coordenadas de las balizas (ilustración anterior), se proceden a realizar los casos de prueba que se ven a continuación.

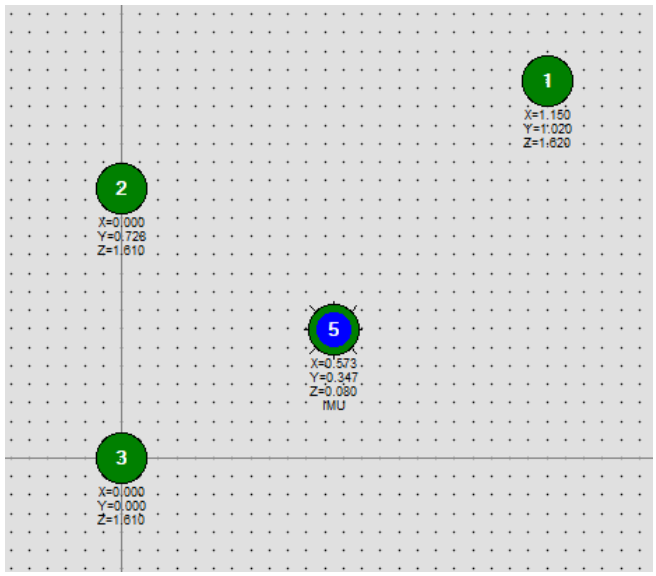


Ilustración 26: Nuevo sub-mapa para la nueva ubicación.

Caso 1: Una de las balizas no tiene línea de visión directa,

Para el caso en que una de ellas no tiene línea de visión, se realizará una serie de 10 medidas, comparando la

desviación típica que ofrecen y la ruta que se dibujó con la obtenida ahora. En este caso se hará obstruyendo por ejemplo la baliza 2.

Ilustración 27: Baliza no obstruida (izquierda). Baliza obstruida (derecha).

Para la obstrucción se ha empleado un sobre acolchado, capaz de absorber los ultrasonidos y servir como barrera, cubriendo todos los laterales de la baliza. Para comprobar su correcto funcionamiento tapando la baliza, se ha observado el movimiento provocado en la posición de la baliza móvil, y se ha visto que se producen oscilaciones al taparla. Tras estas oscilaciones se estabiliza, pero las variaciones de posición son a simple vista aparentemente mayores.

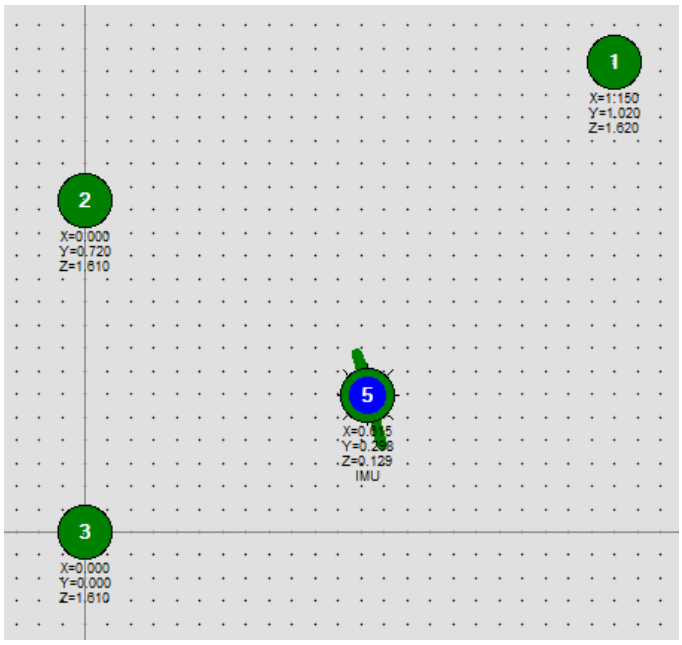


Ilustración 28: Vista del desplazamiento derivado de la obstrucción de la baliza.

Haciendo la prueba de la desviación típica y varianza se obtiene:

	Varianza [m²]	Desviación típica [m]
X	2.0111e-06	0.0014181
Y	6.2667e-06	0.0025033
Z	5e-07	0.00070711

Tabla 6: Varianza y desviación típica de una posición fija con la baliza 2 obstruida.

Respecto a los mismos valores sin obstrucción:

	Varianza [m²]	Desviación típica [m]
X	1.7889e-06	0.0013375
Y	3.8222e-06	0.0019551
Z	4.5556e-07	0.00067495

Tabla 7: Varianza y desviación típica de una posición fija con la baliza 2 libre de obstáculos.

Puede verse que se obtienen aproximadamente los mismos valores para la obstrucción de una misma baliza, pese a que inicialmente viendo la posición actual en el mapa dibujado en el *dashboard* pareciera que la variación era mayor.

Donde se observa la mayor diferencia es en la posición, dado que previamente a la obstrucción se detecta una posición, y se detecta otra distinta que indica que se ha producido un movimiento:

	Posición anterior [m]	Posición final [m]
X	0.55	0.72
Y	0.41	0.22
Z	0.09	0.12

Tabla 8: Cambios en la posición estimada por las balizas.

Para ver un indicativo del cambio se harán los porcentajes de cambio respecto a valor inicial:

$$\%_{total} = \frac{PosicionFinal - PosicionInicial}{PosicionInicial} \cdot 100 \qquad (12)$$

Obteniéndose lo siguiente:

	Porcentaje de variación [%]	Porcentaje absoluto de variación total [%]
X	30.9091	36.8613
Y	-46.3415	
Z	33.3333	

Tabla 9: Porcentajes de variación de valores.

El porcentaje absoluto es la media del valor absoluto de los porcentajes de variaciones, que da una idea de la variación total de las medidas.

Como puede verse la obstrucción de una baliza proporciona una variación de posición muy considerable.

En cuanto a la representación de la misma trayectoria que se veía en la ilustración 23 (ahora con leves variaciones debidas al cambio de escenario de prueba y pequeños movimientos manuales) se observa la siguiente:

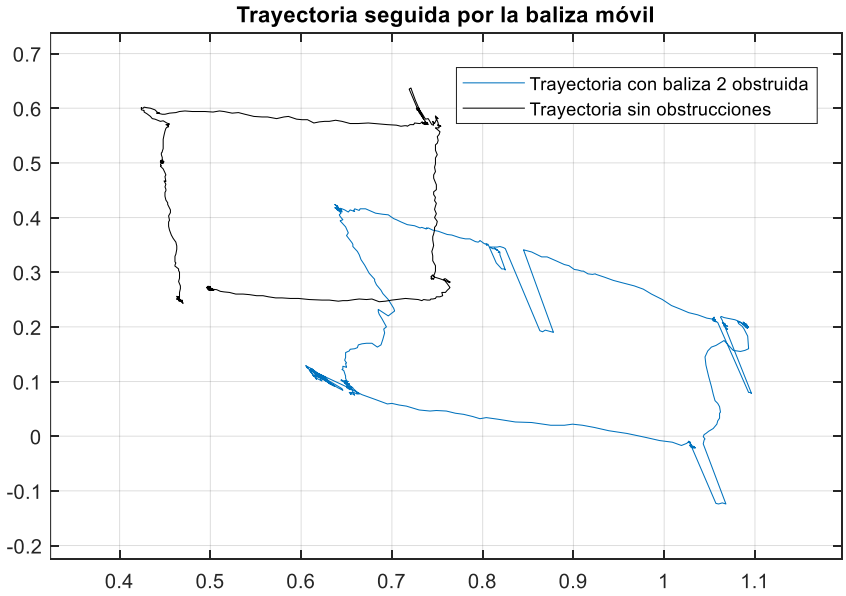


Ilustración 29: Vista de la trayectoria con la baliza 2 obstruida.

Pese a que hay una variación de la posición estimada, el fabricante indica que con 2 balizas estáticas es posible utilizar correctamente el sistema de localización en 2D, con lo cual, ignorando la coordenada Z y utilizando algún tipo de filtrado puede ser posible seguir usando el sistema, por ejemplo, si una de las balizas se ha quedado sin batería y no es posible acceder a ella.

Caso 2: Dos de las balizas no tienen línea de visión directa.

Para el caso en que dos de ellas no tengan línea de visión, se realiza la misma prueba que en el caso 1. En este caso se hará obstruyendo la línea de visión de 1 y 3.

Tomando como referencia la Tabla 7, en la que se veían los valores para la ubicación de la baliza (que no se ha modificado para esta prueba), se comparan con la siguiente tabla de datos obtenida al obstruir las balizas 1 y 3:

	Varianza [m²]	Desviación típica [m]
X	2.4889e-06	0.0015776
Y	1.4489e-05	0.0038064
Z	7.2222e-07	0.00084984

Tabla 10: Vista de varianza y desviación típica para la obstrucción de las balizas 1 y 3.

Puede verse que de nuevo varianza y desviación típica han aumentado, como era de esperar, al obstruir la línea de visión de otra baliza adicional. Por otro lado, las oscilaciones en la posición que se veían en la ilustración 28 ahora son más pronunciadas y frecuentes, haciendo que la localización sea extremadamente poco fiable.

	Posición anterior [m]	Posición final [m]
X	0.55	0.46
Y	0.41	0.52
Z	0.09	0.04

Tabla 11: Estimación de posición con B1 y B3 obstruidas.

	Porcentaje de variación [%]	Porcentaje absoluto de variación total [%]
X	-16.3636	32.9162
Y	26.8293	
Z	-55.5556	

Tabla 12: Estimación de porcentaje de variación de la posición respecto a la obtenida sin obstrucciones.

Puede verse de nuevo que la variación de posición de nuevo es considerable, no siendo viable la utilización de la estimación de posición dada en estas condiciones.

La trayectoria sobre el cuadrado que se veía en la ilustración 21 ahora queda como sigue:

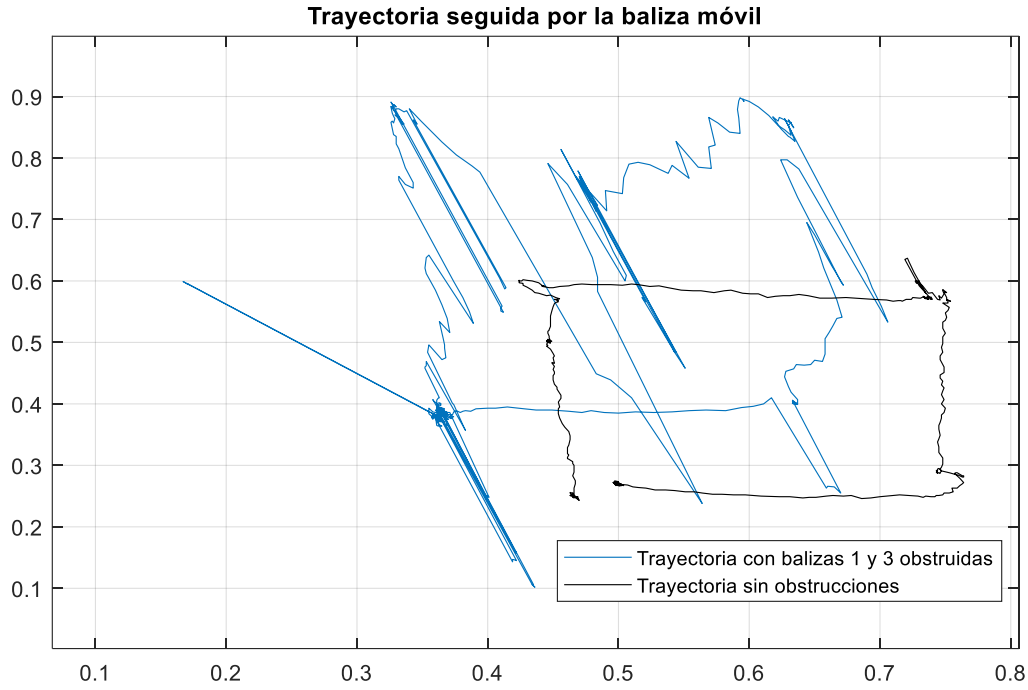


Ilustración 30: Vista de la trayectoria con las balizas 1 y 3 obstruidas.

En el caso de la obstrucción de una baliza sí que era posible usar el sistema de posicionamiento con cierta fiabilidad, pero ahora que se tienen 2 balizas obstruidas las medidas que devuelven no son fiables, además de tener oscilaciones muy pronunciadas y frecuentes de posición, que harán fluctuar incluso el resultado de aplicar un filtro.

3.3.3. Pruebas de localización con todas las balizas con línea de visión obstruida.

En este caso ya no hay visión directa sobre la baliza móvil. Se comprobará cuál es el efecto sobre la trayectoria dibujada.

Ilustración 31: Trayectoria dibujada cuando las 3 balizas no tienen línea de visión.

Puede verse que la trayectoria dibujada tanto en el caso de todas las balizas obstruidas como en el caso en el que 2 de ellas están obstruidas está muy deformada, haciendo muy difícil su aprovechamiento como estimación de posición fiable.

Aunque con las 3 balizas el sistema no se puede utilizar, durante el desarrollo del trabajo se detectó una baliza defectuosa, la cual ha influido sobre los resultados de 1 y 2 balizas obstruidas. La comparación con los resultados aquí obtenidos se comparará con los nuevos experimentos tras sustituirla, en el apartado 5. Problemas durante el desarrollo.

4. MONTAJE DE LA BALIZA MÓVIL SOBRE UN VEHÍCULO

Como ya se adelantó, se usará el robot JetBot basado en Jetson Nano para el montaje de una de las balizas móviles y la lectura por parte del robot de su ubicación estimada.

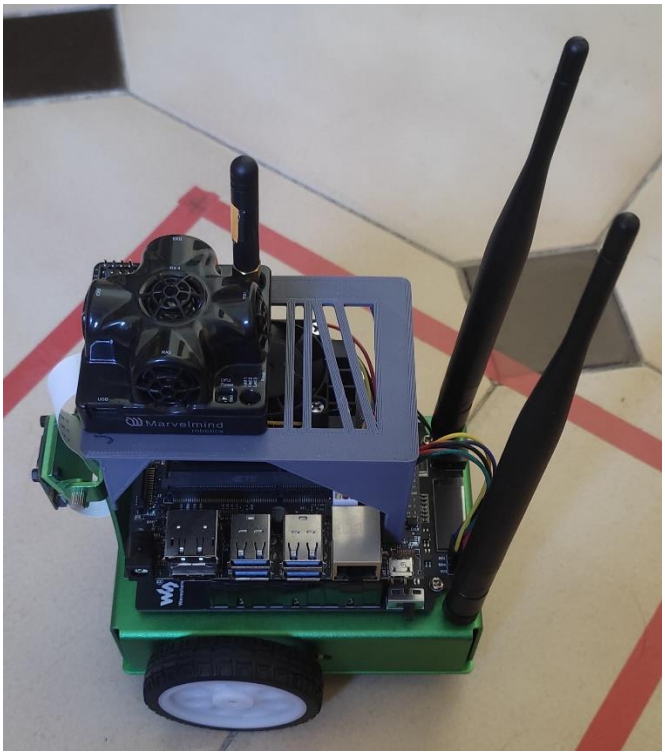


Ilustración 32: Imagen del JetBot con la baliza IMU montada.

Antes de realizar cualquier guiado es necesario caracterizar el tren motriz que posee el vehículo, basado en dos motores, uno colocado en cada rueda. De este sistema mecánico es necesario conocer la correspondencia de la señal que se le aplique a su controlador con la velocidad que desarrolla.

En cuanto al sistema operativo se emplea Ubuntu 18.04, con una interfaz E/S que permite el uso de componentes externos tales como el puente H que controla a los motores. En cuanto a memoria RAM, es el modelo con mayor capacidad, siendo en este caso de 4 GB, los cuales se verá que son más que suficientes para funcionamiento de cualquier programa. El sistema se carga en una memoria microSD, que en este caso se ha usado de 32 GB, espacio de sobra para albergar tanto al SO como a los programas y datos que se usan en esta aplicación.

La constitución mecánica no es excesivamente favorable: las ruedas a veces deslizan, las ruedas locas de apoyo delantera y trasera al ser pequeñas y metálicas suelen atascarse en huecos, los motores tienen una reducción suficiente para poder mover el vehículo, pero a veces con demasiada velocidad, etc. Pese a esto, el robot es controlable y los efectos de estas limitaciones se mencionarán posteriormente.

Tanto el proceso de prueba de los motores como los resultados de velocidad obtenidos se presentarán en un subapartado.

4.1. Caracterización del funcionamiento de los motores.

Inicialmente es necesario determinar el régimen de funcionamiento de los motores, dado que habrá un rango de señal PWM en la que no respondan por no poder generar el par necesario para comenzar a moverse.

El código de prueba se verá en uno de los Anexos, es un código sencillo en el que se carga en un script de Python la librería que incluye la interfaz del Jetson Nano con el puente H. Para usarlo simplemente basta con indicar el *duty cycle* de la señal PWM aplicada.

Tras varias pruebas se obtiene lo que sigue:

PWM [%]	Superficie de la prueba	Dirección	Motor funcionando
20	Vacío	Hacia delante	Izquierdo sí, derecho a veces
22	Vacío	Hacia delante	Izquierdo sí, derecho a veces
25	Vacío	Hacia delante	Ambos
30	Vacío	Hacia delante	Ambos
20	Plano	Hacia delante	No
22	Plano	Hacia delante	No
25	Plano	Hacia delante	Ambos
22	Plano	Hacia detrás	Izquierdo sí, derecho a veces
25	Plano	Hacia detrás	Ambos
30	Plano	Hacia detrás	Ambos

Tabla 13: Funcionamiento de los motores según PWM aplicado.

Comprobando que para el funcionamiento se necesita aplicar al menos un 25% para garantizar que ambos motores se mueven.

Teniendo los límites definidos es necesario conocer qué velocidad desarrollan las ruedas para poder saber qué distancia se va a desplazar el robot en cada momento.

Con el objetivo de conocerla, se plantean varias posibilidades:

1. Recorrer una distancia fija conocida, y calcular el tiempo que tarda desde que comienza el recorrido hasta que acaba.
2. Medir la velocidad de rotación de las ruedas con una referencia en la rueda y otra referencia fija, por ejemplo, en el chasis del robot.
3. Desarrollar un prototipo de medidor de RPM bastante preciso, que permita estimar una velocidad con una proximidad mucho mayor a la real.

De las tres opciones anteriores, se han probado las 3, pero se han encontrado problemas a cada una de ellas:

- Recorrer una distancia fija conocida es una opción plausible, siempre que la distancia sea suficientemente grande, el robot tenga su dirección lo suficientemente calibrada como para no desviarse (leves diferencias de rotación en los motores), y se disponga de una pista de pruebas lo suficientemente grande para realizar las pruebas. En el escenario que se tenía, no se tenía una pista acorde a las necesidades, dado que a altos valores de PWM el robot desarrolla una velocidad considerable. Adicionalmente, se hicieron unas pruebas en una pista pequeña a bajas RPM, y debido a las diferencias de rotación mencionadas anteriormente se producían desviaciones muy considerables en su dirección de desplazamiento.
- Medir la velocidad angular de las ruedas con una referencia puede ser buena opción, si se tuviera un sistema de captura de vídeo lo suficientemente rápido para temporizar una vuelta. En el caso que se maneja, la cámara de la que se disponía no era capaz de capturar el momento exacto de paso de la referencia móvil por la referencia fija:

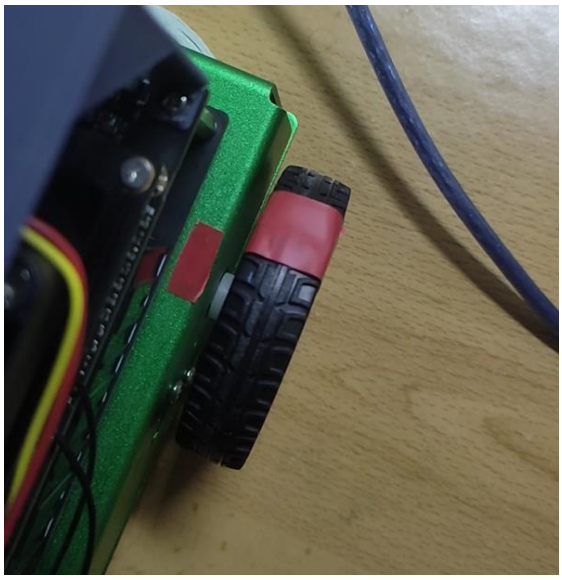


Ilustración 33: Vista de la colocación de una referencia fija en el chasis verde, respecto a la móvil, situada en la rueda.

- En cuanto al desarrollo de prototipo de medidor de RPM, es la mejor opción. En la web existen ejemplos hechos usando una interrupción de cambio de nivel en una entrada [23], mientras que, para una medida precisa, tal y como funcionan los medidores de altas RPM comerciales, es necesario poder conocer el tiempo entre pulsos de rotación.

  Para poder tener un medidor lo suficientemente preciso se ha desarrollado un sketch en Arduino, con el microcontrolador ATMega328P montado sobre una placa UNO. En lugar de emplear únicamente una interrupción al cambio de nivel, se ha empleado un temporizador hardware, de manera que se conoce su periodo de temporización al momento de dispararse la interrupción de cambio de nivel en una entrada.

  Al hablar de interrupción de cambio de nivel en la entrada se hace referencia a una entrada digital al microcontrolador en la que se ha colocado la salida de un optoacoplador CNY70.

  Este optoacoplador devuelve un nivel alto al reflejar la luz sobre una superficie blanca que emite un diodo led que posee, y un nivel bajo en caso de que la luz se absorbida por una línea negra. En base a esto se construye el prototipo:

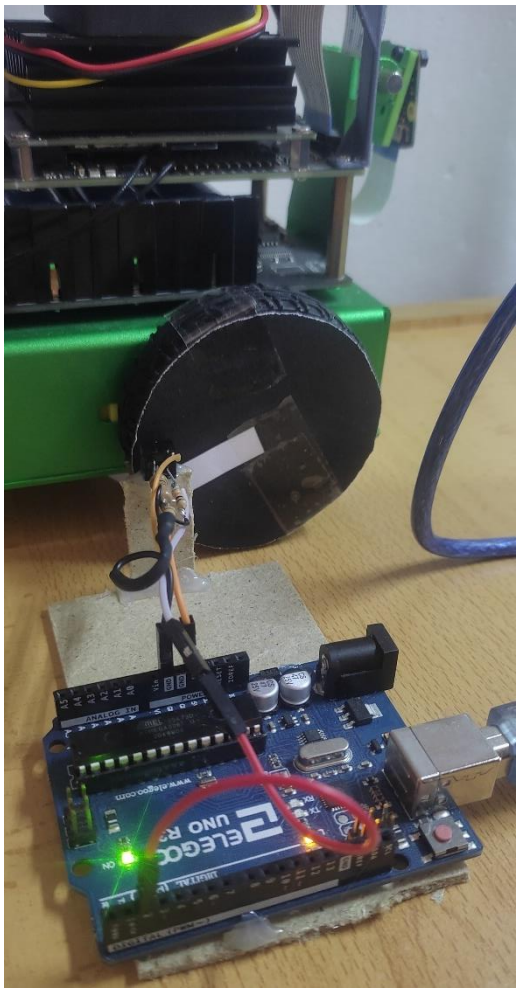


Ilustración 34: Vista del montaje con el microcontrolador.

Para conseguir los pulsos de salida del optoacoplador se usa una circunferencia de color negro, con una banda blanca. Cuando el optoacoplador se coloca sobre la banda blanca su salida está a nivel alto. Si se temporiza el tiempo entre pulso y pulso se pueden conocer las revoluciones por segundo, y teniendo esto se hace el producto por 60 y se obtienen las revoluciones por minuto (RPM).

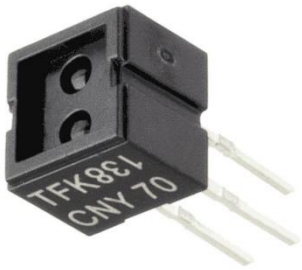


Ilustración 35: Vista del sensor CNY70 montado perpendicularmente a la rueda. Imagen cedida por [24].

El código que se carga a la placa se dejará de nuevo en uno de los Anexos, de manera que en el desarrollo sólo se mostrarán resultados.

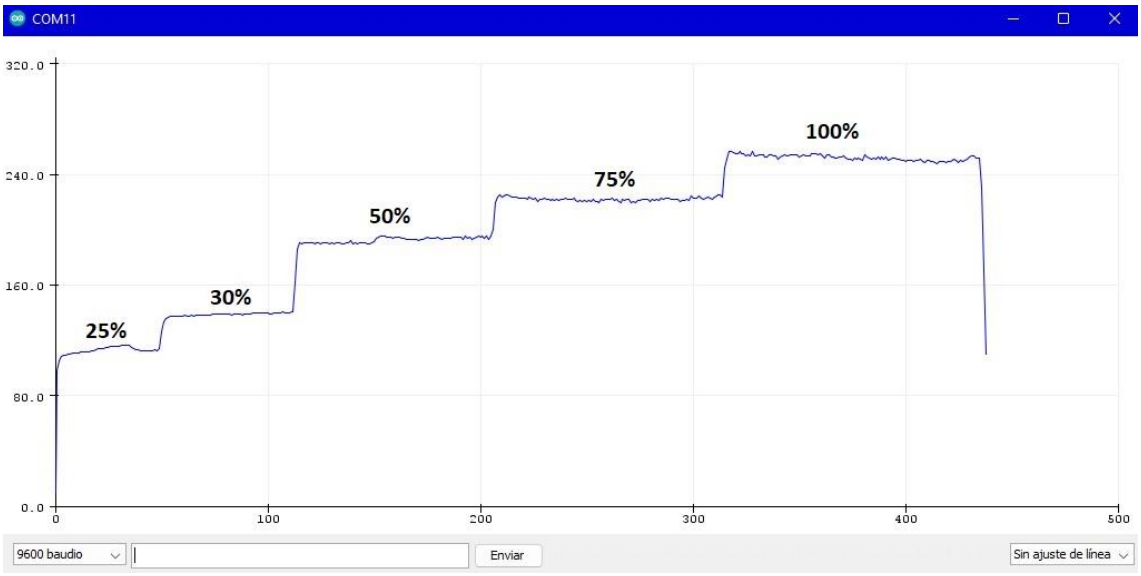


Ilustración 36: Vista de las RPM que se alcanzan para cada PWM.

Dado que en la ilustración 36 anterior no se aprecian de manera detallada las RPM medidas, se han recopilado los datos en la siguiente tabla para poder comprobar la velocidad desarrollada por el robot. También, teniendo en cuenta el radio de la rueda, 3.2 cm, se calcula la velocidad lineal mediante $v = \omega \cdot r$:

PWM [%]	Rango observado [RPM]	Velocidad de rotación media [RPM]	Velocidad lineal del robot [m/s]
25	124, 125	124.5	0.4172
30	149, 150	149.5	0.5010
50	205, 208	206.5	0.6920
75	233-234	233.5	0.7825
100	240-250	245	0.8210

Tabla 14: Velocidad obtenida del robot.

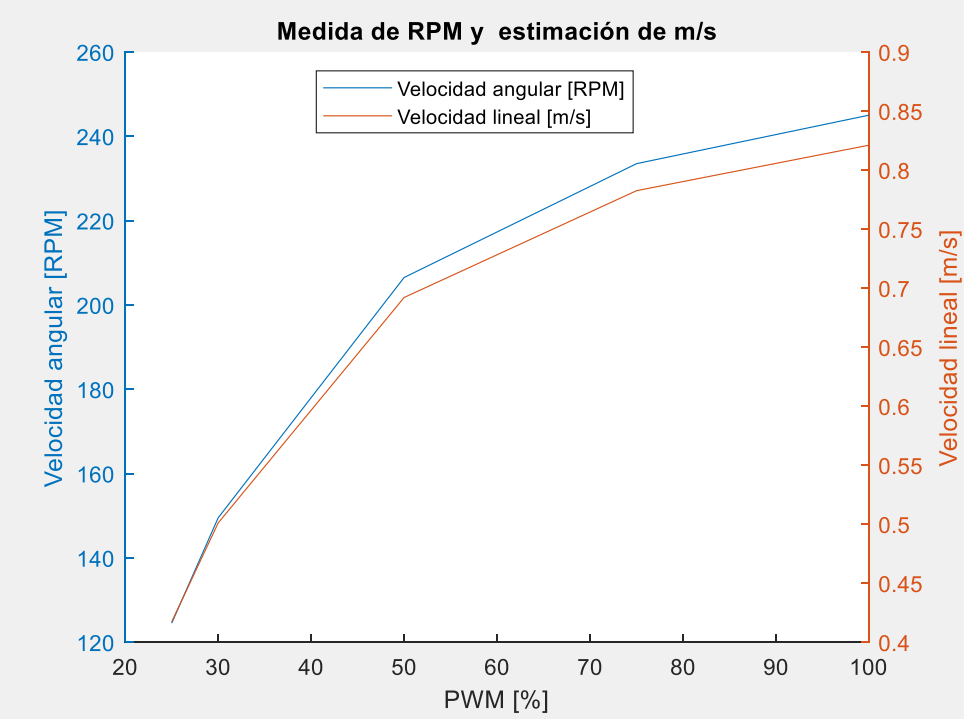


Ilustración 37: Vista de la gráfica generada al representar las velocidades.

Puede verse que no se ajusta a una función lineal, respondiendo a una función aproximadamente logarítmica. Si fuera necesario en el desarrollo del código, se podría calcular una función para estimar la velocidad de salida dada la entrada en PWM.

Para la realización de este ensayo de velocidad se ha comprobado que la velocidad en vacío (sin estar el robot en el suelo) es aproximadamente la misma que cuando se sitúa en el suelo, dado que la reducción de velocidad que lleva a cabo en mecanismo reductor hace que tenga suficiente par de avance como para no ralentizarse con el par resistente del peso y el rozamiento.

Una vez que se tiene el sistema mecánico caracterizado ya se puede operar el robot con todos los datos que se poseen.

4.2. Descripción del algoritmo de control del robot.

En la creación del algoritmo de control del robot han participado tanto los algoritmos desarrollados por Marvelmind para la obtención de datos de las balizas como los algoritmos de control de motor desarrollados por NVIDIA, ambos en el lenguaje de programación Python.

Para describir el algoritmo de funcionamiento del robot, de una manera breve, se hará uso de diagramas de flujo y esquemas que permitan distinguir claramente la arquitectura que sigue.

Con el objetivo de aunar todo el código y poder usar el robot de manera sencilla se ha creado un archivo llamado “robotClass_vX.py”, en el que se concentran todas las funciones necesarias para poder hacer uso de todas las posibilidades que se han implementado dentro del robot. La letra “X” en “vX” dentro del nombre del ficrero hace referencia a la versión de él que se está usando. Para usar una versión u otra es imprescindible especificar el nombre correcto al importarlo al programa. La clase se llama DifferentialRobot, y consta de los siguientes métodos:

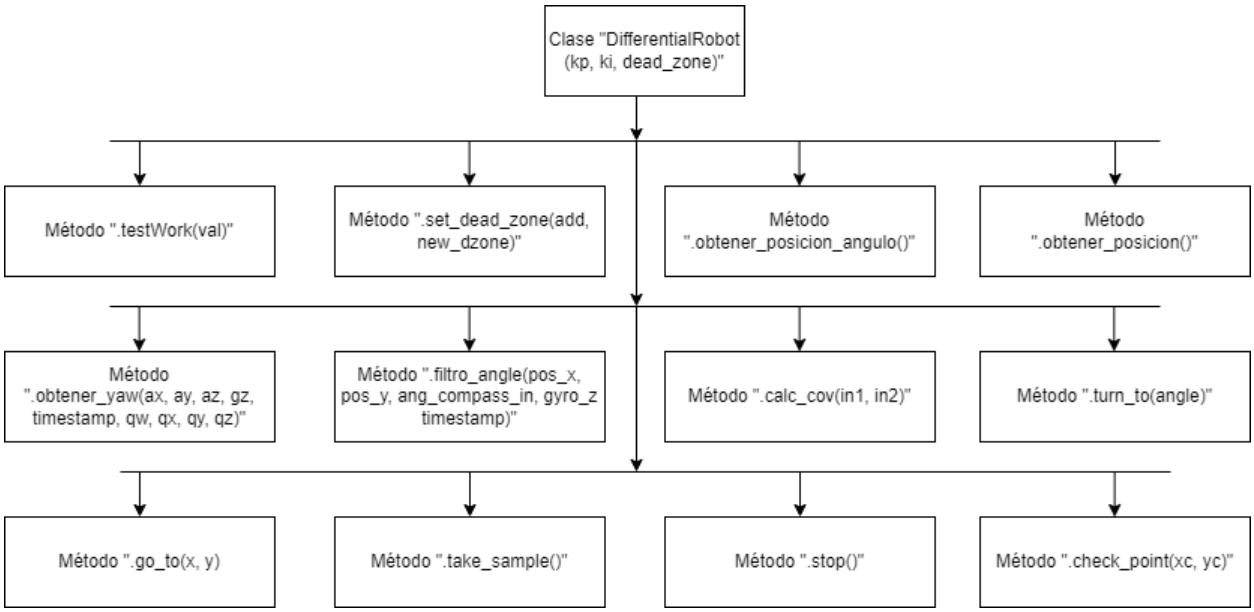


Ilustración 38: Arquitectura de la clase creada.

Al instanciar un objeto de la clase, por ejemplo: "robot = DifferentialRobot(kp=[0.5, 0.3], ki=[0.01, 0.001] dead_zone=[0.1, 2.7 0.05, 3.2]", se indican los siguientes parámetros:

- Los controladores PI emplearán, para el de ángulo, una constante kp=0.5, y una constante ki=0.3, y para el de posición, una constante kp=0.3, junto con una ki=0.01.
- Habrá una zona muerta delimitada por dead_zone, tal que: dead_zone=[xmin, xmax, ymin, ymax]. Con el ejemplo de instanciación dado, el robot no podrá acceder a una zona que rebase ni la coordenada x mínima "xmin", ni la coordenada x máxima "xmax". Lo mismo ocurre para la coordenada y.

Se hará una breve descripción de los demás métodos, profundizando en aquellos que han resultado más exhaustivos de realizar.

1. Método ".testWork(val)"

   Es un método sin ninguna función sobre el robot, simplemente indica si se ha instanciado correctamente la clase, y los parámetros que se han elegido. Toda esta información es imprimida por un terminal Python.

   Si se emplea este método con val=1, se devolverá la respuesta anterior, en caso contrario, se devolverá error.

2. Método ".set_dead_zone(add, new_dzone)"

   Si add=True se carga la restricción especificada en "new_dzone" dentro del vector de restricciones interno a la librería llamado "self.dead_zone", para permitir saber si un punto que se pretenda alcanzar se ha salido de las zonas permitidas.

3. Método ".obtener_posicion_angulo()"

   Empleado en las primeras versiones del código, es encargado de volcar en variables de la clase ("self.IMUfusion" y "self.rawIMU") tanto las lecturas de la fusión de datos con IMU como las lecturas RAW (en crudo) de datos.

4. Método “.obtener_posicion()”

   Dedicado al guardado en una variable de la clase “self.pos_act” la posición actual registrada por la baliza. Es el método que se emplea en la versión final para calcular desplazamientos.

5. Método “".obtener_yaw(ax, ay, az, gz, timestamp, qw, qx, qy, qz)"

   Tiene distinto comportamiento según se configure en una variable local a la clase “self.way”. Si “self.way=True” se calcula el ángulo al que está orientado el robot según el valor de los cuaternios que devuelve la fusión de datos de la IMU. Si “self.way=False” devuelve el ángulo calculado con un filtro complementario que actúa sobre los valores de la lectura del acelerómetro y de la lectura del *yaw* proporcionada por el giroscopio.

6. Método ".filtro_angle(pos_x, pos_y, ang_compass_in, gyro_z, timestamp)"

   Aplica tres filtros de Kalman a los parámetros de entrada: posición, ángulo de brújula (“ang_compass_in”) y aceleración del eje Z en el giroscopio (“gyro_z”).

   Filtro 1: Se calcula el ángulo como el obtenido al computar el arco tangente entre el punto registrado en el momento actual y el punto leído anteriormente.

   Filtro 2: Brújula. Inicialmente, si se devuelve en grados, simplemente se escala el ángulo y se aplica el filtrado de Kalman.

   Filtro 3: Se realiza la integral de la velocidad angular en º/s para obtener el ángulo integrado.

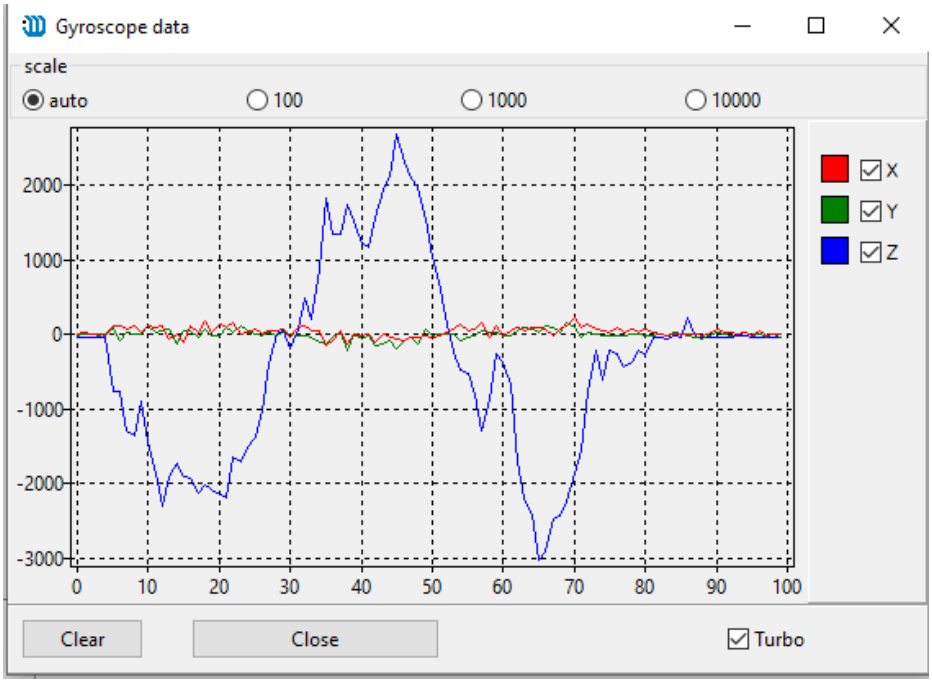


Ilustración 39: Vista de los valores que devuelve el giroscopio al girar la baliza. Velocidad angular en º/s.

Para que estos filtros funcionen es necesario especificar la varianza de la señal de entrada (se deben hacer ensayos). A continuación, se debe especificar una varianza para un posible ruido que entre como perturbación al proceso (debe ser una varianza lógica, no puede ser un parámetro elegido al azar, puede estar inspirado en las varianzas de entrada, para ser, por ejemplo, de un orden 10 veces menor).

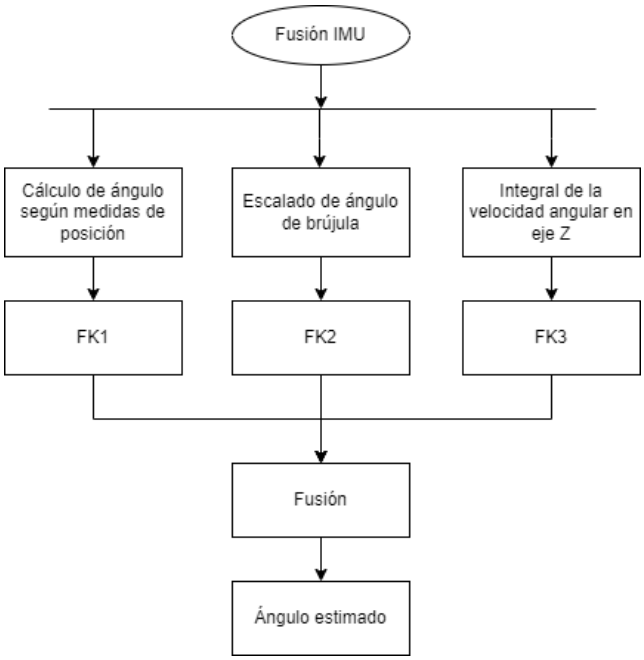


Ilustración 40: Funcionamiento del algoritmo de fusión.

Finalmente se fusionan los resultados, usando por ejemplo el cálculo de la covarianza de cada uno respecto a un ángulo girado. El filtro que haya arrojado mayor correspondencia con el ángulo girado será el que más ponderación tendrá sobre la salida final conjunta de los 3 filtros.

7. Método ".calc_cov(in1, in2)"

   Simplemente se trata de una función implementada para calcular la covarianza entre dos vectores de entrada "in1" e "in2", con el objetivo de poder calibrar los filtros de Kalman creados sin tener que recurrir a cálculos manuales ni programas de terceros.

8. Método ".turn_to(angle)"

   Es el método principal dentro de la arquitectura de cualquier programa de movimiento del robot, siempre se ejecuta para poder seguir una ruta. Su función es reorientar el robot hacia su próxima posición.

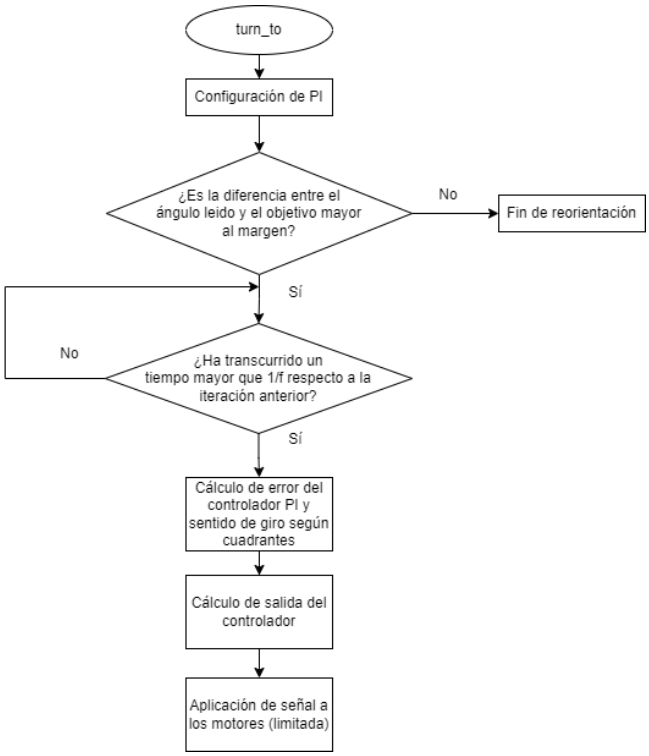


Ilustración 41: Funcionamiento interno de ".turn_to(angle)".

Hay varias partes del algoritmo que requieren explicación:

- Configuración de PI.

  Los controladores PI serán objetos de una clase llamada "classPI", que alberga en su interior la estructura de 2 tipos de controladores: un PI con saturación interna, y otro con saturación externa. Para calcular la salida del PI es necesario únicamente especificar el error de entrada, dado que las constantes kp y ki se declaran al instanciar el objeto.

  Están inspirados en la aproximación por *Backward Euler*, que calcula el término integral en base al error anterior. Para que funcione correctamente es necesario un periodo de muestreo constante, es por ello que el bucle se ejecuta cada 1/f s, donde f será la frecuencia de refresco de ángulo leído de las balizas.

  Control de saturación y antiwindup.

  Características importantes que se deben añadir a los controladores y que garantizan que el robot nunca se va a salir de control.

  Se ha implementado un control de saturación que hace que, si la acción de control es mayor a un máximo, declarado en el programa como "PWMmaxP", de valor 0.3 en este caso (30% de PWM), se limite la salida a este máximo.

  El problema del *windup* radica en que, al producirse la saturación, el controlador no es consciente dado que la limitación no le afecta, y esto resulta en una acumulación de error en el término integral que ocasiona una saturación prolongada en el tiempo. Se ha resuelto deteniendo la acumulación de término integral.

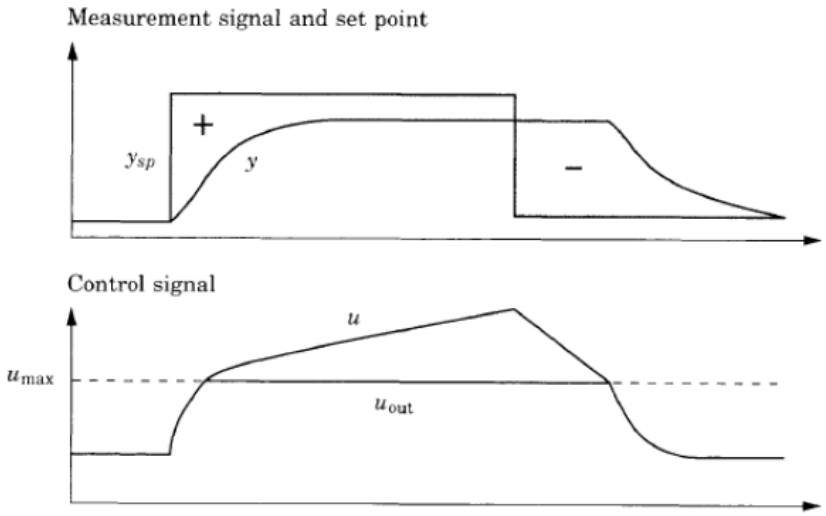


Ilustración 42: Vista del efecto del windup. Imagen obtenida de [25].

En la ilustración anterior puede verse el efecto del *windup*. En la parte inferior puede verse la salida teórica del PI (u) y la que realmente está obteniéndose (uout) ya limitada. Esto hace que el controlador siga acumulando el cálculo de la acción integral al ver un error que no se reduce. Derivada de esta acumulación es la respuesta que se obtiene cuando el error ya cambia de signo, que como puede verse (parte superior de la ilustración) es un retardo hasta que la acción integral vuelve a descender desde el valor que ha acumulado.

- **Margen de ángulo.** Es necesario dado que será difícil alcanzar el ángulo exacto, de manera que, si el ángulo alcanzado se encuentra dentro de este margen, se acepta como bueno. Esta aceptación deriva de que el ángulo será corregido en el siguiente paso, dentro del algoritmo go_to(x, y), con lo cual no es imprescindible alcanzar el ángulo exacto. Paralelamente este margen también se aplica a la tolerancia del ángulo que devuelve la lectura de las balizas, que es levemente variable. En este caso se ha empleado un margen de 5º en ángulo, este margen se preconfigura dentro de la clase "DifferentialRobot", en el parámetro "self.angle_margin".

  Relacionado con este aspecto, es necesario profundizar en la idea de utilizar inicialmente una baliza a la solución actual, dos de ellas actuando en modo *paired hedge*. De todo esto se hablará en el apartado siguiente 5. Problemas durante el desarrollo, pero por ahora considerar que, el ángulo que se obtiene al ubicar estas dos balizas de la manera que se ve en la ilustración 44 (se verá en posteriores líneas), es el que se emplea para reorientar el robot (sistema YB, XB).

- Espera de actualización de acción de controlador.

  Se ha implementado una tasa de refresco de los controladores igual a la de refresco de ángulo, que es la que determinará el cambio en la salida del controlador PI. La tasa de refresco se especifica de nuevo al inicializar la clase, con el parámetro "self.TactAngle".

- Cálculo del error del controlador PI y sentido de giro según cuadrantes.

  Es necesario tener en cuenta los cuadrantes de los ángulos, para decidir cuándo debe moverse en sentido horario y antihorario, optimizando el tiempo de reorientación yendo siempre por el camino más corto. Además, no se puede introducir directamente al controlador PI el error como la diferencia entre ángulo de referencia y ángulo actual, dado

que en el paso de 359º a 0º el controlador experimenta un cambio de signo que se debe tener en cuenta. Para que el controlador tenga un funcionamiento suave, se ha implementado un método de optimización de ángulo girado y un cálculo de error según ángulo diferencial, de manera que el controlador nunca verá saltos de ángulo que generen señales de control inestables. Este algoritmo podrá verse en el código completo, dentro de los Anexos, pero su funcionamiento puede resumirse en el siguiente diagrama de flujo:

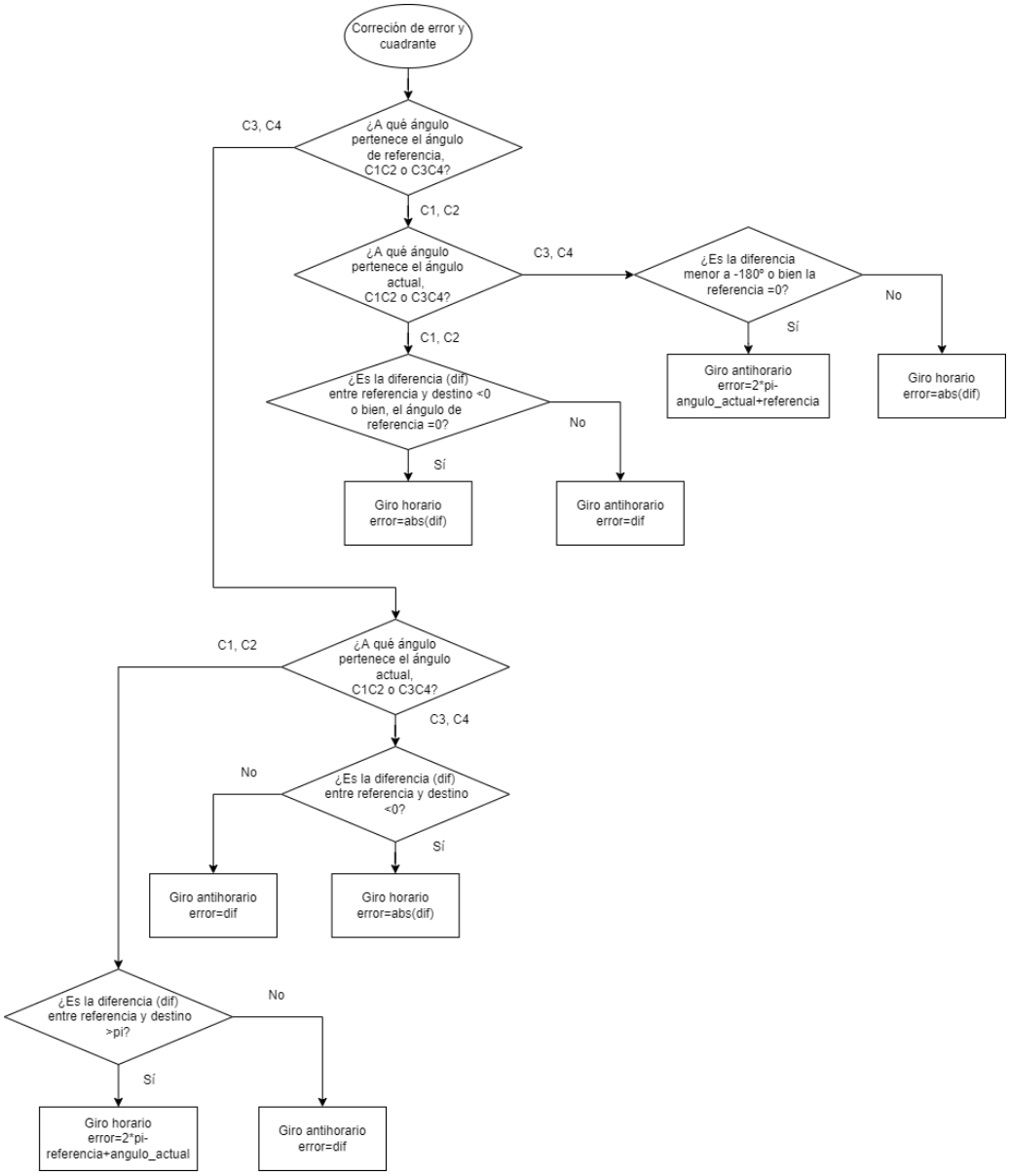


Ilustración 43: Diagrama de flujo de corrección de sentido y cuadrante.

Dado que la dirección de avance del robot es aquella en la que tiene la cámara, se debe tener en cuenta lo siguiente:

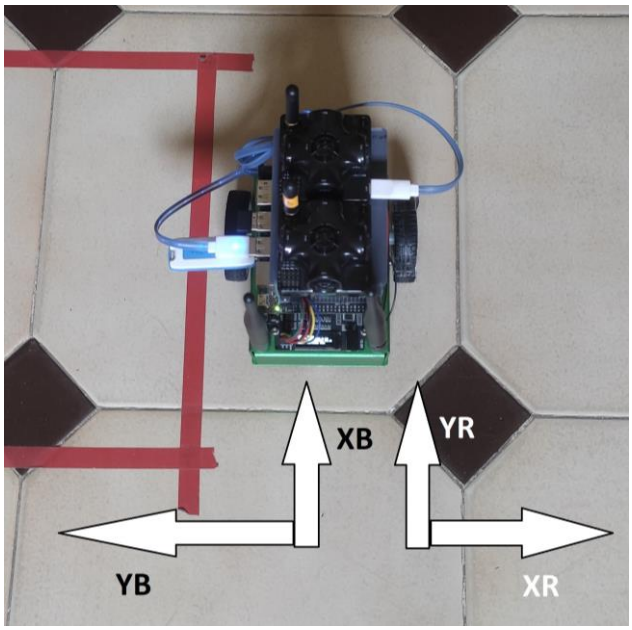


Ilustración 44: Orientación del sistema de referencia de las balizas.

En la ilustración anterior, el robot se encuentra alineado con el eje Y de referencia del mapa que se creó en el *dashboard* (sistema de referencia que puede verse en la ilustración 25), de manera que, la lectura de ángulo que debería devolver es de 90º (la parte delantera del robot se encuentra alineada con el eje Y original). Esto no ocurrirá, dado que el ángulo que devuelven las balizas en esta posición viene dado por YB, XB, de manera que el ángulo leído será 180º (puede verse que el eje YB se encuentra desfasado +90º respecto al eje Y de referencia, YR, XR). Puede verse que aparecen dos balizas en lugar de una, la razón de esto se explicará en líneas posteriores.

Para solventar esta diferencia de ángulos, y para poder leer el ángulo que forman las balizas en modo *paired hedge*, se ha modificado el archivo "marvelmind.py", llamado ahora "marvelmind_mod.py", que incluye la clase "MarvelmindHedge", conteniendo un nuevo método llamado "return_angle()", que selecciona la información necesaria de la trama de datos recibida para devolver el ángulo, escalarlo, teniendo en cuenta que viene expresado en decigrados, y finalmente restarle los 90º de desfase que tienen el eje de avance del robot respecto al de las balizas.

Dado que cuando el eje Y del sistema de las balizas se encuentra en el primer cuadrante el ángulo leído será menor a 90º el resultado de hacer la diferencia con 90º será negativo, hecho que debe ser corregido para que al controlador entre siempre una referencia positiva y un ángulo actual positivos, expresados en [0º, 360º]. Esto también se encuentra corregido dentro de la clase modificada "MarvelmindHedge".

Dado que la lectura de ángulo se realiza de 0 a 360º, en el cálculo mediante arcotangente (función atan2, que es capaz de diferenciar el cuadrante de la orientación calculada) del ángulo formado de un punto a otro, ha sido necesario reescalarlo a ángulos positivos, de manera que, si el ángulo obtenido por arcotangente era negativo, se le han sumado 2π rad para situarlo en los cuadrantes inferiores con ángulo positivo.

De esta manera, teniendo en cuenta todas las consideraciones anteriores, se consigue que el robot se reoriente siempre por el camino angular más corto, consiguiendo una optimización de tiempo de giro.

9. Método "go_to(x, y)"

Recurre al método turn_to(angle) tras calcular el ángulo desde un punto inicial al punto siguiente, y tras hacer esto y reorientar el robot, entra en un bucle con dos controladores PI en paralelo, que se encargan, uno de generar la acción de control derivada de la diferencia entre el ángulo objetivo y el ángulo actual, y otro de generar la acción de control derivada de la distancia que haya entre el punto actual y el objetivo. Todo se ve más claro en un diagrama de flujo que resuma su funcionamiento.

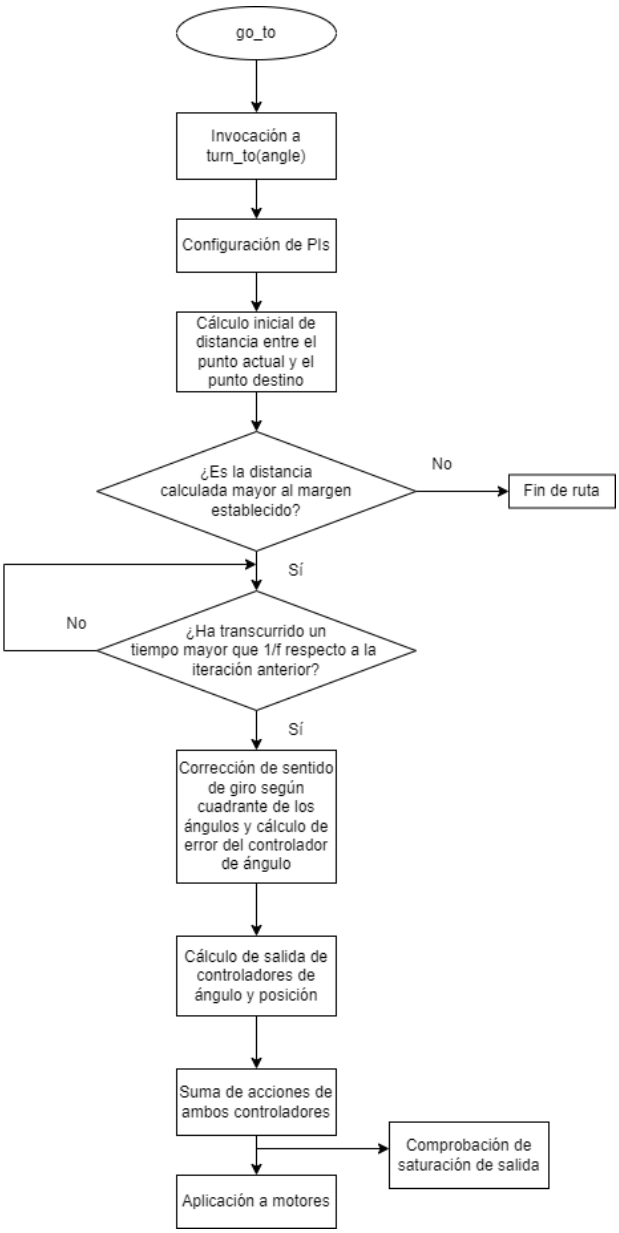


Ilustración 45: Diagrama de flujo de "go_to(x,y)".

Puede verse que es un diagrama similar al de "turn_to(angle)", y es que hereda parte de su funcionamiento, de hecho, antes de entrar en el bucle de control se hace una recolocación inicial del robot. El controlador de ángulo es el mismo que se implementa en el primero, con la salvedad de que no es el único, funciona en paralelo con un controlador que tiene en cuenta la distancia al punto destino. También se diferencia en que los controladores PI instanciados esta vez tienen control de saturación externo, ya que no se puede calcular internamente al ser la acción final la suma de dos controladores distintos.

En este código se deben tener en cuenta aspectos similares a los que se tenían en el algoritmo "turn_to(angle)":

- Configuración de los PI.

  Como se ha mencionado, ahora se usan dos controladores, los cuales se inicializan con las constantes kp y ki que se declararon al crear el objeto de la clase "DifferentialRobot".

- Tiempo mayor que 1/f para el recálculo de la acción de los controladores.

  Ahora este parámetro es distinto al de turn_to(angle), dado que ahora debe emplearse tanto el ángulo como la posición actualizados, para los cuales como máximo se ha conseguido una frecuencia de actualización de 17.8 Hz. Este parámetro se declara al inicializar la clase como "self.TactPos".

- Influencia del cuadrante de los ángulos. Misma consideración que se tenía en turn_to(angle).

- **Margen de distancia.** Antes era un margen de ángulo, ahora lo es de distancia. Se especifica un margen de distancia para que cuando el robot alcance un punto de distancia al objetivo menor a este margen se detenga, considerando que ha llegado al objetivo. Si en el momento de llegar al punto destino el robot no está orientado correctamente al ángulo al que debe estar orientado por acción de los controladores para llegar al punto destino, se puede hacer uso de nuevo de "turn_to" para recolocarlo. En este caso se ha establecido un margen de 10 cm, que se declara de nuevo al inicializar la clase como "self.dmargin". La distancia al punto se calcula como el ángulo formado entre la posición actual y la de destino, de manera que se garantiza que el robot seguirá siempre el camino de mínima distancia.

10. Método "take_sample()"

    Su única función es guardar una foto en el momento que se ejecuta, en una ubicación definida por programa. Dado que viene completamente implementada por NVIDIA, no se explicará su funcionamiento, únicamente mencionar que es importante tener instalados los *packages* necesarios para su funcionamiento.

11. Método "stop()"

    Es un método que debe ejecutarse en todos los procesos que se hagan con la librería desarrollada para cerrar el puerto serie y permitir usarlo por parte de otro programa, como por ejemplo el *dashboard*. Además, se encarga de terminar la ejecución del programa de Python.

12. Método "check_point(xc, yc)"

    Se encarga de aplicar la restricción de coordenadas a las que puede acceder el robot. Simplemente compara el punto de entrada que se le proporcione con las restricciones que se hayan aplicado. Se debe colocar al principio de "go_to(x,y)" para comprobar los puntos.

Todos estos métodos son los que conforman la clase "DifferentialRobot", los cuales dependen de otras variables internas a la clase empleadas a modo de memoria y otras empleadas a modo de configuración. Todos los detalles sobre la clase en su última versión se verán en un Anexo dedicado a este fin.

4.3. Gestión de PWM de motores.

4.3.1. Aplicación de señal.

Como se verá en el apartado siguiente 5. Problemas durante el desarrollo, ha sido necesario cambiar la filosofía de aplicación de la señal PWM al puente H que controla los motores. Los motivos se verán en dicho apartado.

Tanto en "turn_to(angle)", como en "go_to(x, y)" hay una parte del programa llamada "Aplicación a motores", la cual es una sección del algoritmo dedicada a asignar la señal PWM calculada en los controladores a los motores.

En lugar de aplicarla de manera directa se aplica con un ciclo de trabajo concreto, es decir, si el controlador calcula una señal del 25% de PWM, esta no se aplica de manera continua a los motores hasta la próxima actualización del controlador, sino que se aplica en forma de tren de pulsos. Se puede apreciar mejor con un ejemplo gráfico.

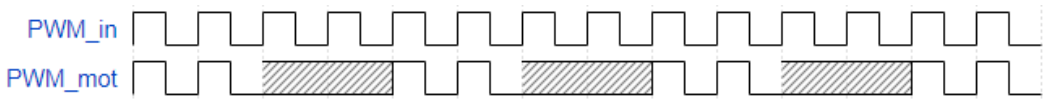


Ilustración 46: PWM continuo (superior) vs trenes de pulsos de PWM (inferior).

En el caso de "turn_to(angle)", se aplica el tren PWM con un ciclo de trabajo del 50%, tal y como se ve en la ilustración 46, y para el caso de "go_to(x,y)" se emplea un tren de pulsos con ciclo de trabajo del 66% (2/3 de periodo). La razón de esto es que se consiga una velocidad menor en el giro que en el desplazamiento final, para optimizar de nuevo el tiempo de viaje del robot de un punto a otro.

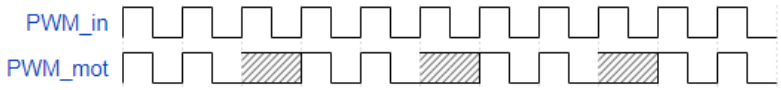


Ilustración 47: PWM por tren de pulsos con ciclo de trabajo del 66%.

El ciclo de trabajo se calcula como:

$$DC = \frac{T_{PWMalto}}{T_{PWMtren}} * 100 \qquad (13)$$

Donde, $T_{PWMalto}$ es el periodo en el que se está aplicando la señal PWM, y $T_{PWMtren}$ es el periodo de la señal del tren completa, incluyendo el periodo en que se aplica PWM y el periodo en que no se aplica PWM a los motores (a efectos de código equivale a asignarle un PWM de 0).

4.3.2. Selección de funcionamiento PWM continuo o con tren de pulsos.

Al aplicar la señal PWM anterior se pierde velocidad de movimiento del robot, lo cual puede ser un problema si el robot necesita desplazarse en un área extensa. Esta variable se emplea no sólo porque se esté utilizando el robot en un espacio pequeño, sino por limitaciones de ejecución del código que se mencionarán en un apartado posterior.

Para poder permitir al robot seleccionar entre un espacio grande (en el que no utilizar PWM en tren de pulsos, sino un PWM continuo) o un espacio pequeño, se ha incluido en la inicialización de la clase "DifferentialRobot" una variable "self.SS" (de *small space*) que delimita el funcionamiento.

- Si "self.SS"=True, el robot aplica a los motores una señal PWM en tren de pulsos, como se veía en las ilustraciones 46 y 47.
- Si "self.SS"=False", el robot se desplaza con una señal PWM continua, sin periodos de señal a 0.

Todo esto se podrá ver en el Anexo al código, en el que se encuentran todas las funciones comentadas.

4.4. Requisitos y restricciones de SW.

Durante el desarrollo del trabajo se han encontrado impedimentos de cara al software: directrices que hay que seguir para la correcta ejecución, módulos que no se encontraban previamente instalados o necesidades de actualización de firmware críticas.

Como se mencionó anteriormente, es necesario actualizar tanto balizas como módem a la última versión del firmware v4.9, el cual se puede encontrar en la web de Marvelmind.

En cuanto al robot, comenzando por los motores, se debe comentar que:

- Se debe inspeccionar la placa para ver qué puentes H tiene montados, tal y como puede verse en las ilustraciones siguientes.

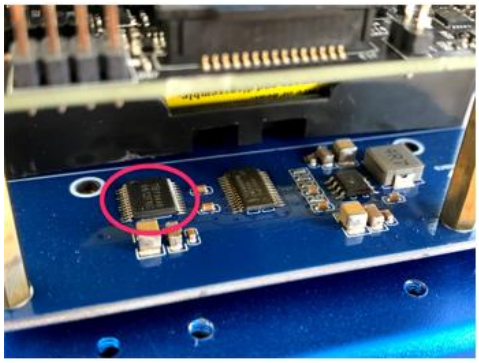


Ilustración 48: Revisión antigua, con un único controlador de motores. Imagen obtenida de [26].

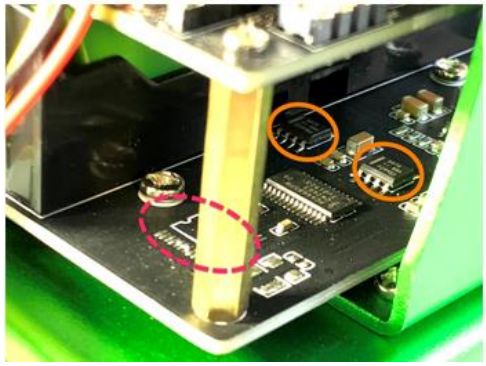


Ilustración 49: Revisión nueva, con dos controladores de motores. Imagen obtenida de [26].

  Si se está usando la revisión nueva de la PCB empleada en el Jetbot, es necesario que la versión de software instalada sea superior a la versión 0.4.1. de firmware Jetbot, es decir, debe ser al menos la versión 0.4.2. Si se emplea una versión inferior, las librerías de control de motores están enfocadas a la revisión antigua de la PCB, y, por tanto, los motores no responderán.

  Tanto para actualizarlo, como para ver qué versión instalar ante este posible fallo, se puede revisar [26].

- Cuando se instala una nueva versión superior a la 0.4.1., la interfaz gráfica de Ubuntu viene por defecto desactivada con el objetivo de optimizar recursos hardware.

  Si no se está muy familiarizado con la programación por terminal, o supone una serie de retrasos de cara a manejar archivos o bloques grandes de código, puede ser imprescindible activar la GUI (*Graphical User Interface)*, o interfaz gráfica de usuario.

Para realizar esto simplemente hay que escribir en el terminal, tras actualizar el firmware de la Jetson Nano [27]:

```
sudo systemctl set-default graphical.target
```

Dependencias de librerías

Son necesarias algunas librerías de Python para asegurar el funcionamiento de todos los módulos que se emplean. Estas librerías son:

- Numpy, crcmod, vispy.
- Pyserial.
- Pytorch, torch, packaging.

Además, se deben instalar determinadas dependencias extra [28], sin las cuales el funcionamiento será limitado. Se deben instalar al menos: traitlets, jupyter lab y misc dependences (dependencias misceláneas).

En el cambio de versión del Jetbot, además, se cambia el directorio de la carpeta "jetbot", la cual incluye todos los módulos para permitir el uso de las características del robot. Pasa de estar en el directorio /opt a estar en el directorio /home/jetbot. Esto servirá de cara a la instalación de las dependencias mencionadas anteriormente.

Además, es necesario que para la instalación de todos los módulos se escriba antes de la instrucción de instalación *"sudo"* para ejecutar los comandos con permiso de superusuario. La contraseña, tal y como se anuncia en el la web del robot es "Jetbot".

Ejecución de scripts

Para ejecutar cualquier script de Python se debe tener en cuenta que debe ser también en modo superusuario, debido a que una ejecución normal no permite abrir los puertos serie, lo cual es necesario de cara a la comunicación del robot con las balizas.

5. PROBLEMAS DURANTE EL DESARROLLO

Durante la implementación del sistema se han enfrentado numerosos problemas, los cuales han llevado a unas u otras soluciones según su naturaleza. En este apartado se resumen los mismos, previamente a presentar los resultados de las implementaciones.

5.1. Funcionamiento de motores.

Con el código cargado al Jetbot, se prueba en primera instancia el funcionamiento con un PWM continuo (sin pausas en el movimiento), pero el robot no es estable, comprobando que:

- Con un giro de ángulo a cualquier consigna (en cualquier cuadrante), el robot se descontrola, rectificando el movimiento cuando ya se ha pasado de la consigna. Ejemplo: se le pide un giro a 45º y acaba en 0º, al no poder disminuir la señal PWM a tiempo.
- Con el desplazamiento a cualquier punto, que implica una reorientación inicial del robot para luego emplear ambos controladores, de ángulo y de posición, ocurre lo mismo, el robot sigue siendo inestable.

Para comprobar que no es problema de configuración de los controladores PI se prueba una constante proporcional baja (0.05) y una constante integral muy baja (0.002), comprobando que la inestabilidad es independiente de la configuración del controlador, pudiéndose suavizar un poco, pero sin alcanzar el objetivo requerido.

Para comprobar cuál era la razón de una ejecución de código tan tardía se hace una prueba de tiempos de ejecución, devolviendo lo siguiente:

```
Angulo RAW:  48
Angulo REAL leido:  -85.2
Controlador:  0.0749802173303024 Angulo objetivo:  315.0 angulo leido:  274.8
t de ejecucion de iteracion completa=  44902.56309509277
t de lectura de angulo:  1817.2264099121094
t de calculo de controlador:  17.404556274414062
t de correccion de signo:  4.5299530029296875
t de asig puente H:  42934.17930603027
Angulo RAW:  0
Angulo REAL leido:  -90.0
Controlador:  0.07616878655091053 Angulo objetivo:  315.0 angulo leido:  270.0
t de ejecucion de iteracion completa=  44734.954833984375
t de lectura de angulo:  16862.869262695312
t de calculo de controlador:  17.642974853515625
t de correccion de signo:  6.9141387939453125
t de asig puente H:  25555.37223815918
```

Ilustración 50: Vista de los tiempos en µs.

Donde puede verse que la asignación de valores al puente H de los motores tras haberse calculado en los controladores era la que demandaba la mayoría de tiempo (del orden de 45 ms).

Para aislar estos resultados, se hace una prueba de tiempo de asignación por separado, en un script cuya única función es asignar un valor de PWM y parar los motores, para ver la máxima velocidad de ejecución que alcanza.

```
jetbot@nano-4gb-jp441:~/Desktop/JetBot_Balizas/jetbotMotion$ sudo python3 basicMotion_v2_pruebaTiempos.py
t de asignacion a motores:  17.930269241333008  ms
t de detencion de motores:  23.828983306884766  ms
t de asignacion a motores:  21.062612533569336  ms
t de detencion de motores:  42.566537857055664  ms
t de asignacion a motores:  22.515535354614258  ms
t de detencion de motores:  19.207239151000977  ms
t de asignacion a motores:  19.136428833007812  ms
t de detencion de motores:  18.214702606201172  ms
t de asignacion a motores:  52.533626556396484  ms
t de detencion de motores:  19.654035568237305  ms
t de asignacion a motores:  42.87838935852051  ms
t de detencion de motores:  36.17405891418457  ms
t de asignacion a motores:  44.33035850524902  ms
t de detencion de motores:  25.249958038330078  ms
t de asignacion a motores:  71.75683975219727  ms
t de detencion de motores:  42.46973991394043  ms
t de asignacion a motores:  36.31472587585449  ms
t de detencion de motores:  43.488264083862305  ms
t de asignacion a motores:  40.81106185913086  ms
t de detencion de motores:  19.768953323364258  ms
```

Ilustración 51: Vista de tiempo de asignación de % de PWM a los motores.

En el script puede verse que el mínimo tiempo que se requiere es de 17 ms, pero el máximo es de 71 ms.

En vista a esto, se decide explorar la librería donde se hace el control del puente H de los motores, ubicada en el script “motor.py”.

5.1.1. Optimización de la librería de control de motores.

Para optimizar la librería el primer paso es observar su composición. En este caso se podía ver que dentro de ella se incluían numerosas funciones relacionadas con otras utilidades del Jetbot que en este caso no se utilizaban (todo lo que fuese distinto a la comunicación I2C que emplea, el canal de cada motor y la asignación de entradas necesaria para utilizar un motor hacia delante o hacia detrás).

De esta manera, tras analizarla detenidamente, se encuentra que también depende de otro script “Robot.py” que también carga otros módulos innecesarios para este uso. Se elimina la dependencia de este otro script.

Finalmente se sintetiza todo en una librería con una clase llamada “Motor” que incluye los siguientes métodos e inicialización:

- Motor(channel).

  Únicamente requiere un parámetro obligatorio, “channel”, cuyo valor decide sobre qué canal I2C se quiere instanciar el control del motor (y por tanto sobre qué motor se actúa). Para los dos motores es necesario por tanto instanciar dos objetos de esta clase. Esto se hace en la clase “DifferentialRobot” dentro de su inicialización. Examinando el código puede verse fácilmente que la asignación de canales es:

  - Motor izquierdo: canal 1.
  - Motor derecho: canal 2.

- “._write_value(value)”.

  Al emplear este método sobre un objeto de motor se permite especificar en value el % de PWM a asignar [-1, 1], tal y como en la librería original.

  Al ser un valor en valor absoluto entre 0 y 1 se puede multiplicar directamente por la escala del controlador de motores, que es [0,4080].

  Para interpretar el signo se deja por defecto, viendo que el valor escalado se escribe en la entrada “ina” cuando el movimiento del motor es hacia detrás y cuando es hacia delante el valor del escalado se

escribe en la entrada “inb”, asignando un 0 a las entradas que no se aplica el valor del escalado, por ejemplo: motor hacia delante valor 2000 (aprox. 50% PWM): ina=0, inb=2000.

- “._release()”.

  Sirve para detener los motores sin necesidad de escribir un PWM=0, aunque tiene la misma función.

Tras reescribir el código y probarlo, se obtienen los siguientes tiempos de ejecución, bajo la misma prueba que la que se observa en la ilustración 51.

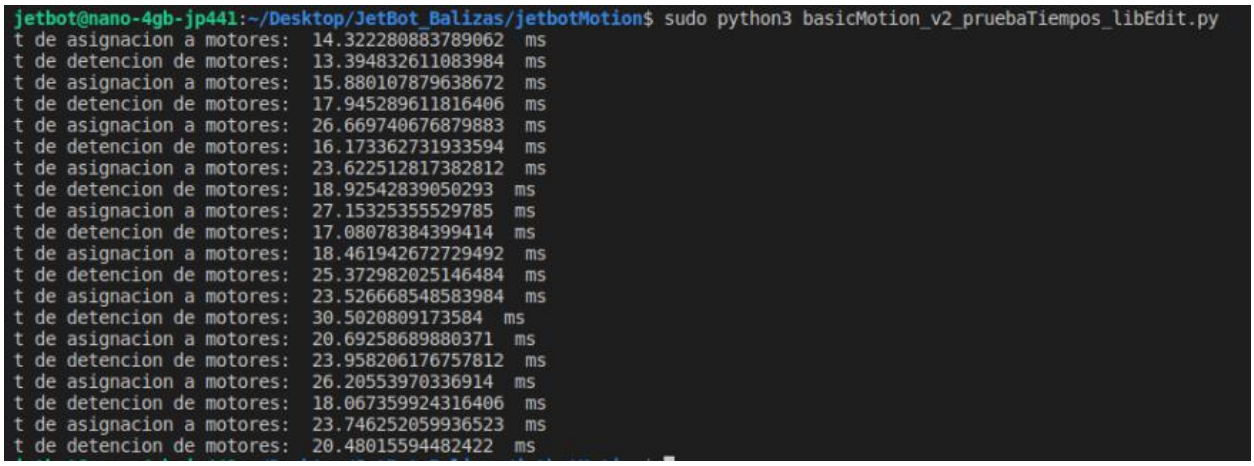


Ilustración 52: Vista del resultado de la prueba con la nueva librería.

Puede verse que se han disminuido los tiempos de ejecución, yendo desde 13 ms a 30 ms, siendo el máximo de 30ms, un 57% más rápida. Adicionalmente, su tiempo de carga (desde que se ejecuta el script hasta que comienzan a moverse los motores) es mucho más bajo, lo cual tiene explicación al haber eliminado tantas dependencias de código.

Pese a esto, tras hacer una prueba con el sistema completo, moviendo el robot, los tiempos de ejecución que arroja son los siguientes:

```
Angulo RAW:  2447
Angulo REAL leido:  154.7
Controlador:  -0.30853057852129767 Angulo objetivo:  135.0 angulo leido:  154.7
t de ejecucion de iteracion completa=  16175.74691772461
t de lectura de angulo:  176.1913299560547
t de calculo de controlador:  16.689300537109375
t de correccion de signo:  1.6689300537109375
t de asig puente H:  15891.551971435547
Angulo RAW:  2458
Angulo REAL leido:  155.8
Controlador:  -0.3104417140522315 Angulo objetivo:  135.0 angulo leido:  155.8
t de ejecucion de iteracion completa=  72826.38549804688
t de lectura de angulo:  5855.083465576172
t de calculo de controlador:  1690.6261444091797
t de correccion de signo:  2.384185791015625
t de asig puente H:  60042.381286621094
Angulo RAW:  2458
Angulo REAL leido:  155.8
Controlador:  -0.31225685647430557 Angulo objetivo:  135.0 angulo leido:  155.8
t de ejecucion de iteracion completa=  38831.94923400879
t de lectura de angulo:  1274.8241424560547
t de calculo de controlador:  18.596649169921875
t de correccion de signo:  2.384185791015625
t de asig puente H:  36475.419998168945
Angulo RAW:  2458
Angulo REAL leido:  155.8
Controlador:  -0.31407199889637966 Angulo objetivo:  135.0 angulo leido:  155.8
t de ejecucion de iteracion completa=  47285.55679321289
t de lectura de angulo:  1492.7387237548828
t de calculo de controlador:  17.642974853515625
t de correccion de signo:  2.1457672119140625
t de asig puente H:  44704.437255859375
Angulo RAW:  2427
Angulo REAL leido:  152.7
Controlador:  -0.3153460892503355 Angulo objetivo:  135.0 angulo leido:  152.7
t de ejecucion de iteracion completa=  26823.997497558594
t de lectura de angulo:  612.2589111328125
t de calculo de controlador:  18.11981201171875
t de correccion de signo:  1.6689300537109375
t de asig puente H:  26116.609573364258
```

Ilustración 53: Vista de los nuevos tiempos de ejecución con el código completo, de nuevo en µs.

Donde puede verse, que en ocasiones el tiempo de ejecución vuelve a ser muy alto, de 72 ms, lo cual es excesivo.

Para ver hasta qué punto este tiempo es excesivo se hacen unos sencillos cálculos con las características mecánicas del robot.

- PWM máximo configurado: 0.35.
- Velocidad de rotación de la rueda a ese PWM: mayor a 149.5 RPM, dado que en los ensayos de prueba de zona muerta de los motores no se hizo para el 35%, pero sí para el 30%, que equivale a 149.5 RPM. Es el caso más favorable:

$$149\ RPM \cdot \frac{360º}{1\ rev} \cdot \frac{1\ min}{60s} \cdot 0.07s = 62.58º$$

  En 70 ms la rueda ya ha girado alrededor de 60º, lo cual equivale a más de un cuarto de vuelta de giro del robot. Esta es la causa de la inestabilidad, al reprocesar el nuevo ángulo el controlador está corrigiéndolo constantemente, sin poder llegar a él porque cuando vuelve a hacer la comprobación de cercanía al margen, éste ya se ha excedido por muchos grados.

  Es por esto que resulta obligatorio usar el tren de pulsos con PWM, para evitar el giro excesivo entre iteración e iteración de cálculo.

  Esto es un efecto directo del Teorema de Muestreo de Nyquist-Shannon, el cual afirma que la frecuencia de muestreo debe ser como mínimo el doble que la frecuencia máxima de la señal medida (esta frecuencia de la señal medida es la que se conoce como frecuencia de Nyquist, y es la frecuencia de refresco de los valores leídos de las balizas).

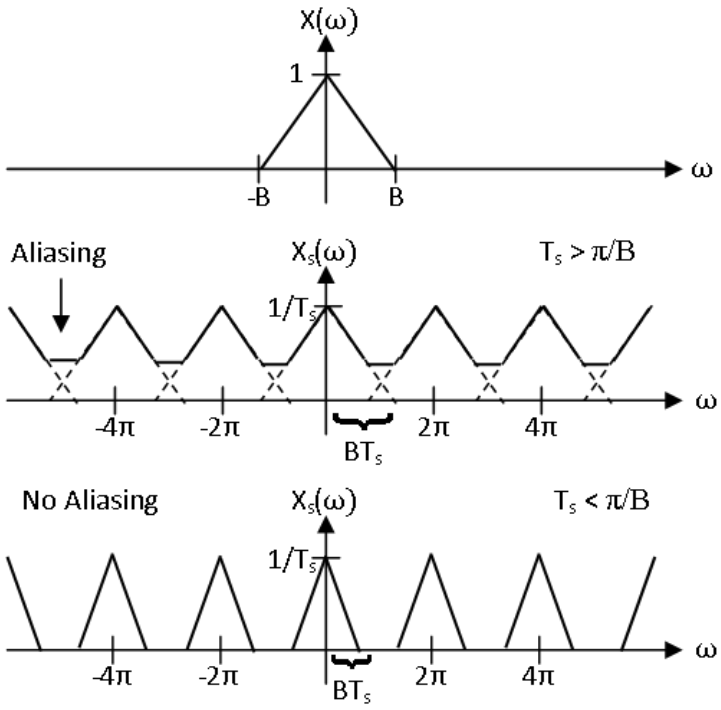


Ilustración 54: Vista del efecto de no cumplir el teorema de Nyquist. Imagen obtenida de [29].

Como puede verse en la ilustración anterior, cuando la frecuencia entre muestreos no es suficiente para poder separar los datos leídos, se produce el llamado *aliasing*, que consiste en la pérdida de datos por efecto del no cumplimiento del teorema de Nyquist-Shannon:

$$f_S \geq 2 \cdot f_N \qquad (14)$$

Donde:

- f_S es la frecuencia de muestreo (en el caso del sistema empleado, es la frecuencia de cálculo del algoritmo).
- f_N es la frecuencia de Nyquist (en el caso de este sistema, la frecuencia de actualización de las lecturas de las balizas).

Al no cumplirse este teorema resulta justificada la obligación de compensar la baja frecuencia de muestreo con una restricción en el efecto de la acción de control (uso de PWM en tren de pulsos).

5.2. Actualización de librerías de control.

En las primeras versiones de desarrollo del algoritmo de control, la librería principal "robotClass_vX", que alberga la clase "DifferentialRobot" se encontraban problemas de rendimiento debido a:

- Exceso de variables globales.

  El acceso a memoria de las variables es más lento que prelocalizarlo cuando se inicializa la librería, tal y como se hace en las últimas versiones del código (actualmente v10). Derivado de este acceso a variables globales se propone la idea de implementar la librería de los controladores PI separada "classPI", para poder instanciar los controladores necesarios fácilmente y encapsular el código.

  Con esto se obtiene una aceleración del código, que seguía limitada por el accionamiento de los motores.

- Operaciones repetidas.

  Haciendo un análisis del código se vería que había varias operaciones repetidas: cálculo de errores de controladores, obtención de posición, etc. Todas estas operaciones han sido optimizadas al mínimo, disminuyendo levemente el tiempo de ejecución.

- Logging de tiempos de ejecución y salidas de controladores.

  En las versiones iniciales siempre se imprimían diversos valores de la ejecución en curso, de manera que el programa debía estar continuamente devolviendo estos valores, ocasionando una leve (pero adicional) espera innecesaria.

  Se han incluido unas variables de nuevo en la inicialización de la clase "DifferentialRobot" llamadas "self.debug" y "self.debugTime", que al encontrarse en valor True devuelven los datos de controladores y de ángulo leídos, y de tiempo, respectivamente. Cuando no son necesarios estos datos simplemente con colocar el valor False en ambas variables es suficiente. También pueden utilizarse de manera unitaria, no deben tener el mismo valor ambas.

5.3. Deriva *(drift)* en *yaw*.

Este ha sido uno de los problemas principales del robot desde el principio, la estimación del ángulo de orientación del eje Z en el plano XY, también conocido, como viene mencionándose, como *yaw*.

Como se anunció en el apartado de antecedentes, concretamente en la sección de cuaternios, es posible calcular el ángulo *yaw* a partir de ellos. Esto también se encuentra implementado en el método ".obtener_yaw" que ya se vio en los métodos de la clase creada.

Al emplear esta forma de obtener el *yaw* se observaba un *drift* (deriva) muy alta en la estimación del ángulo.

```
Angulo posicional calculado:  45.0 Angulo gyro calculado:  45.74774389413859
Angulo posicional calculado:  0.0 Angulo gyro calculado:  45.82200715923623
Angulo posicional calculado:  0.0 Angulo gyro calculado:  45.80026823460117
Angulo posicional calculado:  18.43494882292209 Angulo gyro calculado:  45.69528739322255
Angulo posicional calculado:  0.0 Angulo gyro calculado:  45.67879727915654
Angulo posicional calculado:  0.0 Angulo gyro calculado:  45.735404012386844
Angulo posicional calculado:  45.0 Angulo gyro calculado:  45.820204543666755
Angulo posicional calculado:  0.0 Angulo gyro calculado:  46.05597860786506
Angulo posicional calculado:  90.0 Angulo gyro calculado:  47.180651996033994
Angulo posicional calculado:  90.0 Angulo gyro calculado:  47.333600674289194
Angulo posicional calculado:  0.0 Angulo gyro calculado:  47.333600674289194
Angulo posicional calculado:  90.0 Angulo gyro calculado:  47.68204906687802
Angulo posicional calculado:  0.0 Angulo gyro calculado:  47.990245304314136
Angulo posicional calculado:  135.0 Angulo gyro calculado:  48.56249866183442
Angulo posicional calculado:  135.0 Angulo gyro calculado:  48.75078895110423
Angulo posicional calculado:  180.0 Angulo gyro calculado:  48.86193200172145
Angulo posicional calculado:  135.0 Angulo gyro calculado:  48.882210512409486
Angulo posicional calculado:  135.0 Angulo gyro calculado:  48.97104515902187
Angulo posicional calculado:  153.43494882292185 Angulo gyro calculado:  49.223940669632256
Angulo posicional calculado:  0.0 Angulo gyro calculado:  49.223940669632256
Angulo posicional calculado:  0.0 Angulo gyro calculado:  49.223940669632256
Angulo posicional calculado:  141.34019174590992 Angulo gyro calculado:  50.24432657526519
Angulo posicional calculado:  146.30993247402023 Angulo gyro calculado:  50.927760771957644
Angulo posicional calculado:  146.30993247402023 Angulo gyro calculado:  51.28944562939081
Angulo posicional calculado:  153.43494882292185 Angulo gyro calculado:  51.620661815858085
Angulo posicional calculado:  135.0 Angulo gyro calculado:  51.84409668608822
Angulo posicional calculado:  153.434948822922 Angulo gyro calculado:  52.122157202697096
Se ha finalizado la captura de datos
```

Ilustración 55: Ejemplo de prueba con un ángulo girado de 45º.

La prueba anterior se empleó para calibrar los filtros de Kalman que se mencionaron anteriormente, y como se puede ver, la estimación de ángulo, respecto a los 45º girados, va aumentando.

La razón de esto es el llamado *drift* o deriva, que se produce debido al método de obtención del *yaw*, que consiste en la integración de la aceleración angular que mide el giroscopio integrado en la baliza IMU.

Consultando diversas fuentes [30] [31] [32] puede verse que la deriva en *yaw* es un tema muy tratado en sistemas de navegación y posicionamiento como el que se maneja.

Este *drift* puede eliminarse de varias maneras, pero la probada, de hacerlo pasar por el filtro de Kalman, no dio resultados satisfactorios: no era posible calcular una covarianza baja debido a que las medidas estaban continuamente cambiando. Lo mismo ocurría con la varianza del ruido, estimada, pero de un valor poco fiable y sin posibilidad de compararlo con otros experimentos.

Desechada la posibilidad de usar esta medida, se analizan las medidas de la estimación posicional de ángulo, que se basa en calcular el ángulo cuya tangente es:

$$yaw = atan2\left(\frac{yf-y0}{xf-x0}\right) \qquad (15)$$

Donde:

- yf, xf, son las coordenadas de la posición actual.
- y0, x0, son las coordenadas de la posición anterior.

De nuevo, al hacer pasar estas medidas por el filtro, para una posición estática en la que el robot se mantiene en un punto, resultaba en valores muy dispares, derivados de la tolerancia en posición devuelta por las medidas cedidas por el sistema de las balizas.

Este método funcionaría bien durante una trayectoria del robot, en la que se tiene certeza de que la posición siguiente del robot se encontrará lo suficientemente separada de la inicial para poder calcular un ángulo bastante aproximado al buscado.

Finalmente, se analiza la posibilidad de usar la brújula integrada en la baliza IMU.

Uso de la brújula

Para la fusión de ángulos se planteó el uso de la brújula, un sistema sencillo de estimación de ángulo instantáneo, basado en la lectura de campos magnéticos en las direcciones X e Y [33]. Se ha comprobado que el ángulo se calcula de la manera que se ve en el texto referenciado para cualquier modelo de magnetómetro, viendo el método de cálculo en el modelo integrado en la baliza: LSM303 [34].

Tras leer los manuales del sistema IndoorGPS, se advertía que la brújula integrada en la baliza IMU era muy sensibles en campos magnéticos, pero merecía la pena probarla para ver cómo reaccionaba.

Una prueba útil es hacer un giro completo de la misma, esperando encontrar un ángulo entre 0 y 360º, calculado de la siguiente manera:

$$\theta = atan\,2\left(\frac{My}{Mx}\right) \qquad (16)$$

Donde:

- θ es el ángulo estimado
- My es la magnitud de campo magnético en el eje Y.
- Mx es la magnitud de campo magnético en el eje X.

Pero los ángulos que se obtenían no recorrían los 4 cuadrantes, y esto era debido a lo siguiente:

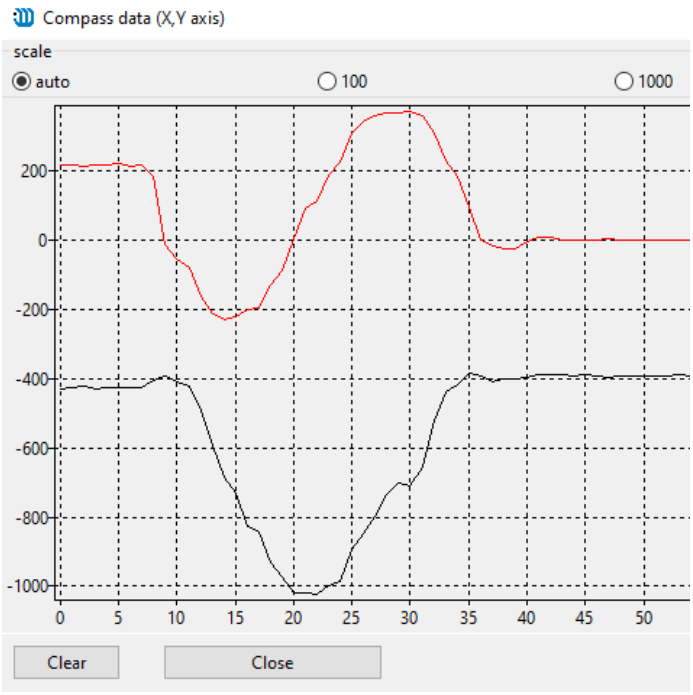


Ilustración 56: Prueba real de magnitudes en ejes X (línea roja) e Y (línea negra). Giro completo de la baliza.

Donde puede verse que la magnitud en el eje Y siempre era negativa, excepto en puntos muy pequeños. Esto hacía que siempre se calculase un ángulo en los cuadrantes 3 y 4 [180º, 360º].

Todo esto se debe a la anunciada sensibilidad al campo magnético que se señalaba en los manuales del sistema de posicionamiento, pero que merecía la pena probar para ver su respuesta.

Finalmente, tras un elevado número de pruebas, se encontró una alternativa que arrojó unos resultados mucho mejores que todas las anteriores, en un tiempo relativamente corto.

5.3.1. Modo *paired-hedge.*

Resuelve todos los problemas de sensorización anteriores. Aunque el filtrado del *yaw* podría haber resultado funcional con un elevado número de pruebas adicionales y ajustes, se decidió implementar este método, que permite un cálculo mucho más rápido y fiable del ángulo del robot. El uso de este método ya venía anunciado desde el apartado 4.2. Descripción del algoritmo de control del robot.

Este método tiene un funcionamiento sencillo: estima el ángulo en función de la ubicación de dos balizas próximas entre sí. Emplea la ecuación 15. La diferencia de posición entre ambas permite calcular el ángulo que forman, y en la mediatriz de la distancia entre ellas determina la orientación del grupo de balizas. Al ángulo que se obtiene con la ecuación 13 se le restan 90º, y ya se tiene el ángulo de orientación de las balizas.

Las balizas que se han colocado en el JetBot son la 5 (IMU) y la baliza 6.

Los datos se obtienen de la baliza 6, la cual es la que proporciona junto con los datos de posición el ángulo de ambas. La baliza IMU no proporciona el ángulo que forman ambas, sino el calculado según la fusión de sensores que posee, el cual tiene un comportamiento totalmente errático. Sobre intervalos de actualización se hablará en el apartado siguiente.

Para configurar el modo paired hedge se deben seguir una serie de pasos:

1. Iniciar el *dashboard* con todas las balizas en el funcionamiento normal.
2. Configurar la baliza 6 en modo hedge (la baliza 5 ya lo estaba).
3. Seleccionar de la lista de balizas inferior la baliza 5 e indicar a la derecha en “Pair mode” la baliza 6, e

indicar la separación entre ambas en cm (entre centro y centro).

4. Mismo procedimiento anterior ahora seleccionando en la lista la baliza 6 y en “Pair mode” la baliza 5 con la misma distancia anterior.

Si el ángulo que se observa es muy inestable puede deberse a la gran cercanía de una baliza a otra, esto se resuelve descendiendo un poco la potencia de TX. En el caso del proyecto esto no ha dado problemas (con 6 cm de separación entre centro y centro de la baliza) y se ha quedado por defecto.

Comprobación de desfase respecto al ángulo girado

Aunque visualmente (en el *dashboard*) puede verse que el ángulo es bastante cercano al real, es posible que la acumulación de medidas proporcione un valor demasiado distinto al real. Por ello, es necesario realizar una prueba para observar la desviación de ángulo respecto a uno teórico.

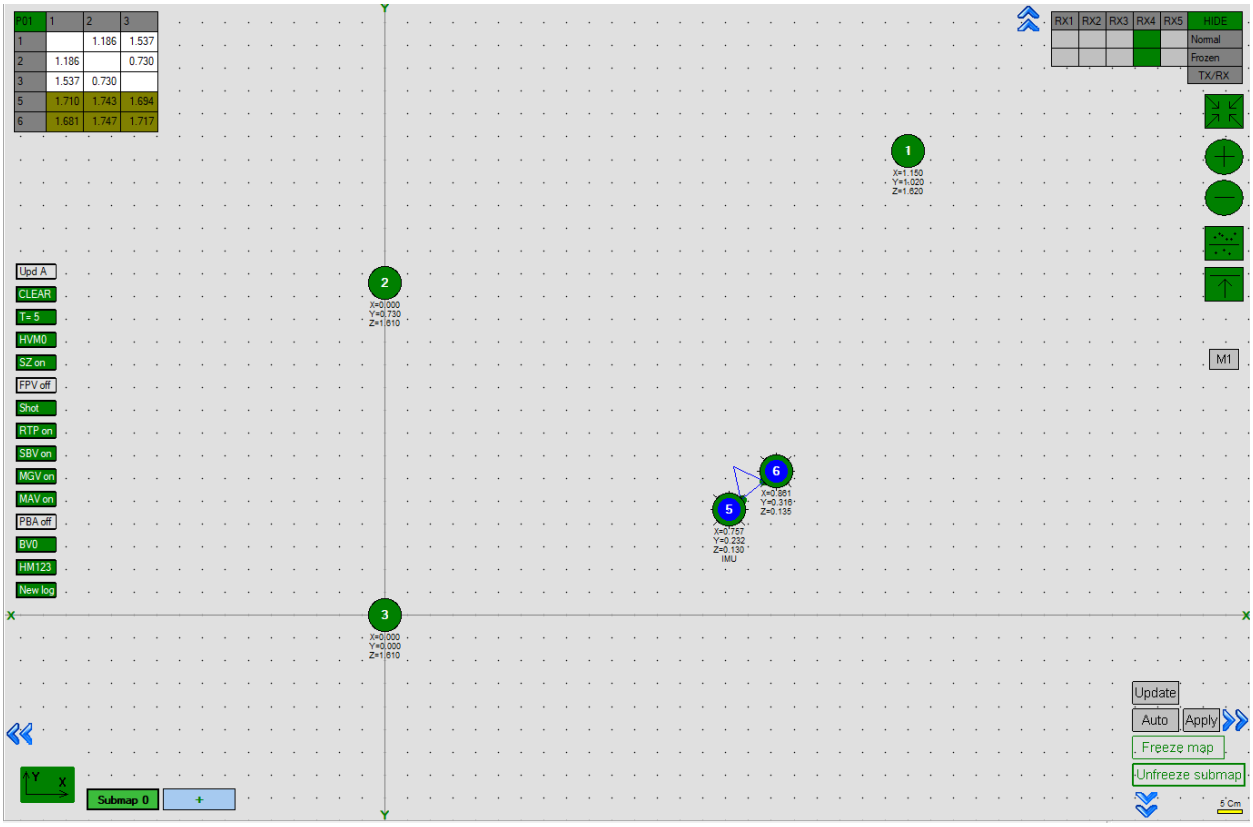


Ilustración 57: Vista de las balizas en modo paired hedge en el dashboard.

En este caso, se ha probado a colocar el robot a 90º (cuando se dice colocar el robot es la orientación de su dirección de avance, con la cámara en la parte delantera), por tanto, las balizas miden 90º más, debido al desfase entre sistemas de referencia que ya se comentó en el apartado 4.2. Descripción del algoritmo de control del robot.

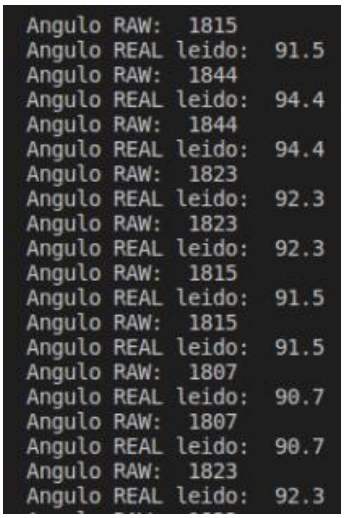


Ilustración 58: Datos obtenidos durante la realización de la prueba.

En la ilustración 58 aparece un fragmento del ensayo, mientras que para procesarlo se ha empleado el guardado de estos datos en un archivo CSV y un programa sencillo en Matlab.

Número de muestras	**500**
Media	92.3º
Diferencia respecto a 90º	2.3º

Tabla 15: Prueba de desviación de ángulo respecto a 90º.

Dado que la desviación es menor a 5 grados, que es el margen que se ha estipulado en el programa para el reposicionamiento, se considera que no hay que corregir el ángulo, y se puede leer directamente del valor proporcionado por las balizas.

5.4. Frecuencia de actualización de las lecturas de las balizas.

En las versiones iniciales, el programa actualizaba la posición a una frecuencia de 8.4 Hz, lo cual, equivale a un periodo de 119 ms, y como ya se pudo observar en el apartado 5.1.1. Optimización de la librería de control de los motores, es un tiempo excesivo que no permite un control suave.

Para aumentar la frecuencia de actualización tanto de ángulo como de posición se han realizado varias operaciones:

- Se ha aumentado la velocidad de comunicación RF de las balizas al máximo permitido, 500 kbps. Esta modificación puede hacer que en grandes espacios se pierda la comunicación, pero para el espacio de prueba ha funcionado perfectamente.
- Se ha delimitado una zona de servicio en el mapa. Si esto no se hace, el sistema de posicionamiento se prepara para detectar cualquier baliza móvil en un área de 30 m2.

La zona de servicio creada puede verse como sigue:

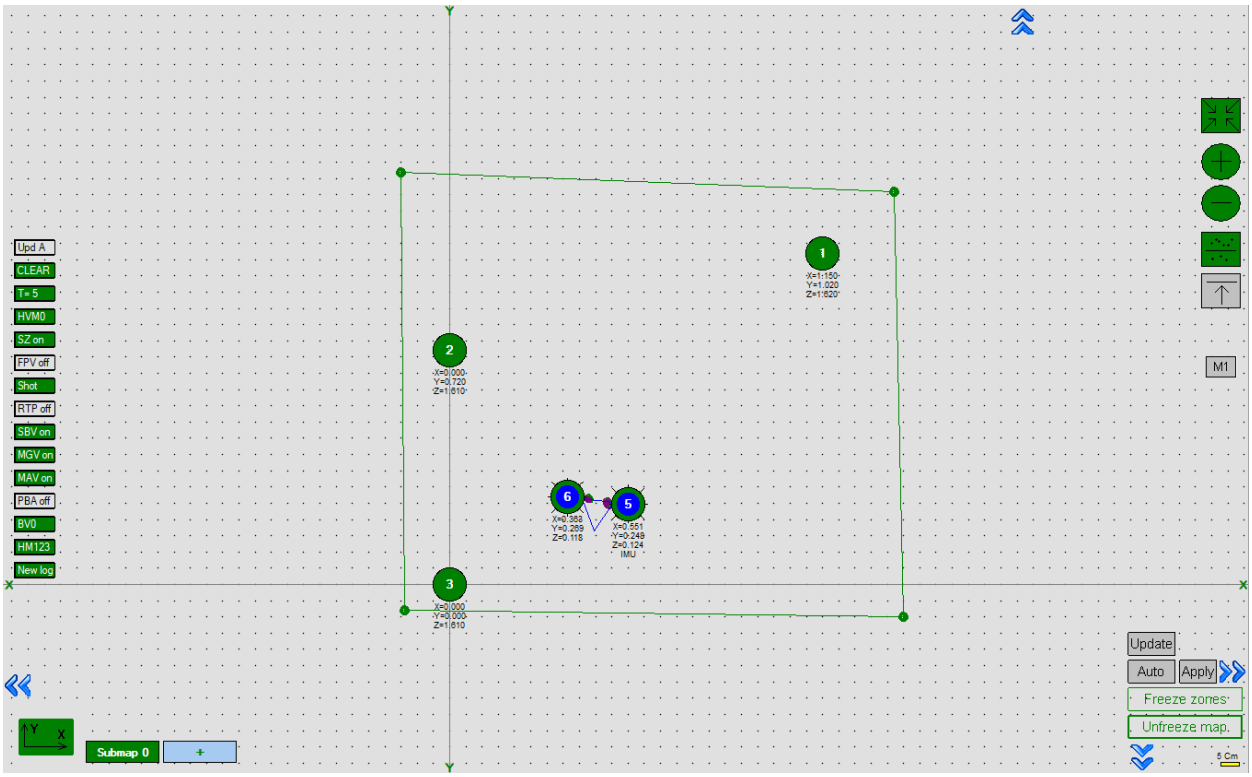


Ilustración 59: Vista de la zona de servicio creada (delimitada por líneas verdes).

Con esto se ha conseguido que la frecuencia de actualización ascienda hasta 17.8 Hz, lo cual equivale a un periodo de actualización de 56.17 ms.

Aunque este periodo sigue siendo relativamente alto, en el manual de Marvelmind se indica que el periodo de actualización del ángulo es menor, de hasta 10 ms (100 Hz), si se obtiene de los datos de fusión de la baliza IMU.

Para comprobarlo, se hace una prueba esta vez obteniendo los datos de la baliza IMU (5), y se cuantifica el tiempo de obtención del ángulo (mediante la bandera ".fusionImuUpdated()"). Se comprueba que el máximo tiempo que tarda en actualizarse la lectura de fusión IMU para estas condiciones de funcionamiento es de 1/66.6 Hz, que equivalen a 15ms. El problema es que el ángulo que devuelve esta baliza del par no es el que forman ambas, sino que devuelve el calculado mediante fusión de sensores, lo cual, como se comentó en el apartado anterior, devuelve medidas erráticas y no es posible su uso.

Se comentarán las mejoras necesarias para abarcar este aspecto en el apartado 8. Mejoras en el sistema.

5.5. Superficies. Sistema mecánico.

Las superficies de trabajo también juegan un papel importante en el desempeño del robot. Hay varios factores que pueden provocar un malfuncionamiento del sistema diseñado, derivado de una mala actuación del sistema mecánico:

- Superficie resbaladiza.

  El movimiento de las ruedas se transforma en un alto deslizamiento. Esto provoca que el sistema vea que un aumento en la señal de control no surta efecto, de manera que la acción de control aumentará, desestabilizando el sistema y dificultando la corrección de orientación.

- Atranque en las ruedas delantera o trasera.

Si el suelo no es totalmente liso, el atranque de las ruedas en algún surco puede provocar el mismo efecto que una superficie resbaladiza, la aplicación de una señal de control innecesaria.

- Obstáculos en la superficie.

  El robot no está preparado para evasión de obstáculos actualmente, de manera que para las pruebas se debe preparar un área lo suficientemente amplia, teniendo en cuenta que no se produzca ninguno de los dos casos anteriores, manteniendo una vigilancia activa mientras se estima el espacio necesario para pruebas de mayor magnitud.

5.6. *Jumps*. Efecto de una baliza estropeada.

En las ilustraciones 29, 30 y 31, se veía que la salida de datos arrojaba lo que parecían "saltos" de coordenadas muy elevados, es decir, en un instante se medía un punto y en el instante siguiente se medía un punto muy alejado, sin haber movido el robot. En la ilustración 28 también se veía un desplazamiento excesivo a modo de salto, que posteriormente se ha descubierto que no era ocasionado por lo que al principio se le atribuía: únicamente la obstrucción de la baliza.

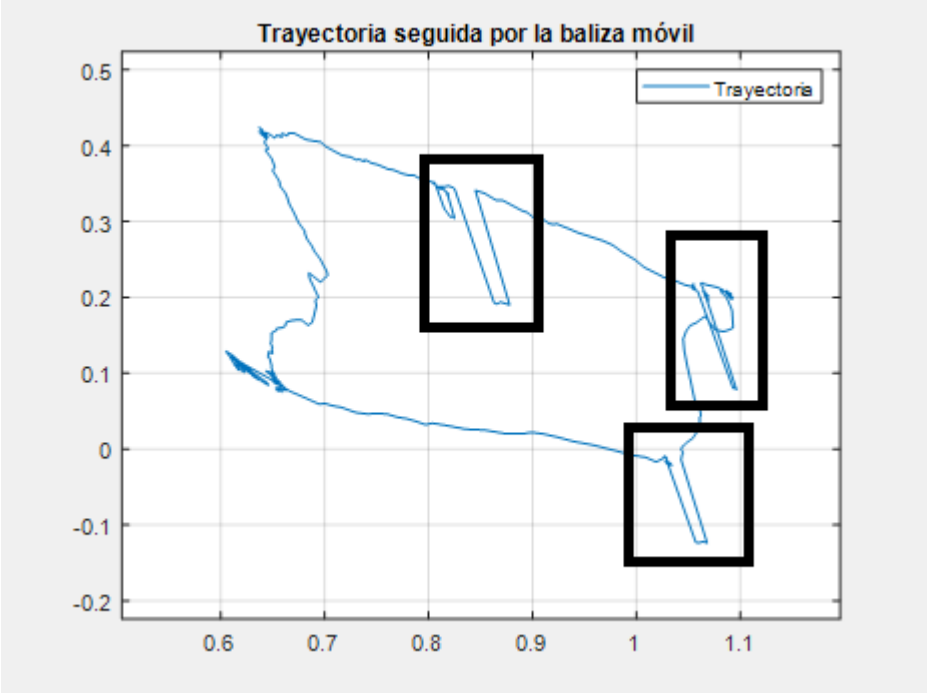


Ilustración 60: Vista de los saltos que se mencionan. Prueba con B2 obstruida.

Buscando información de por qué se producían estos saltos se vio que había varias razones [35]:

- Pérdida de línea de visión:
- Ruido cerca de la baliza receptora.
- Falta de correspondencia de software.

Viendo cómo se verificaba el funcionamiento de la baliza se advirtió de la función "Oscilloscope" dentro del *dashboard*. Esta función permite ver la señal que está recibiendo y transmitiendo la baliza.

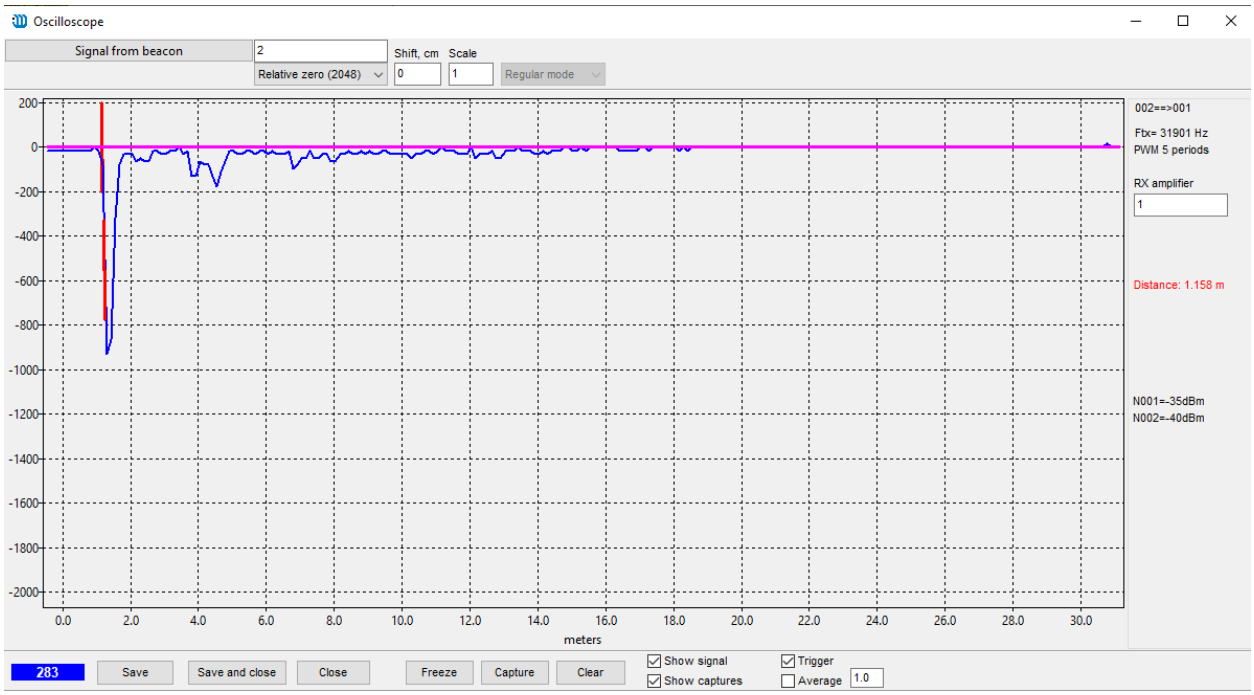


Ilustración 61: Vista de la señal de la baliza 2.

En todas las balizas la señal era parecida a la de la ilustración 61, pero al llegar a la baliza 1 se vio lo siguiente:

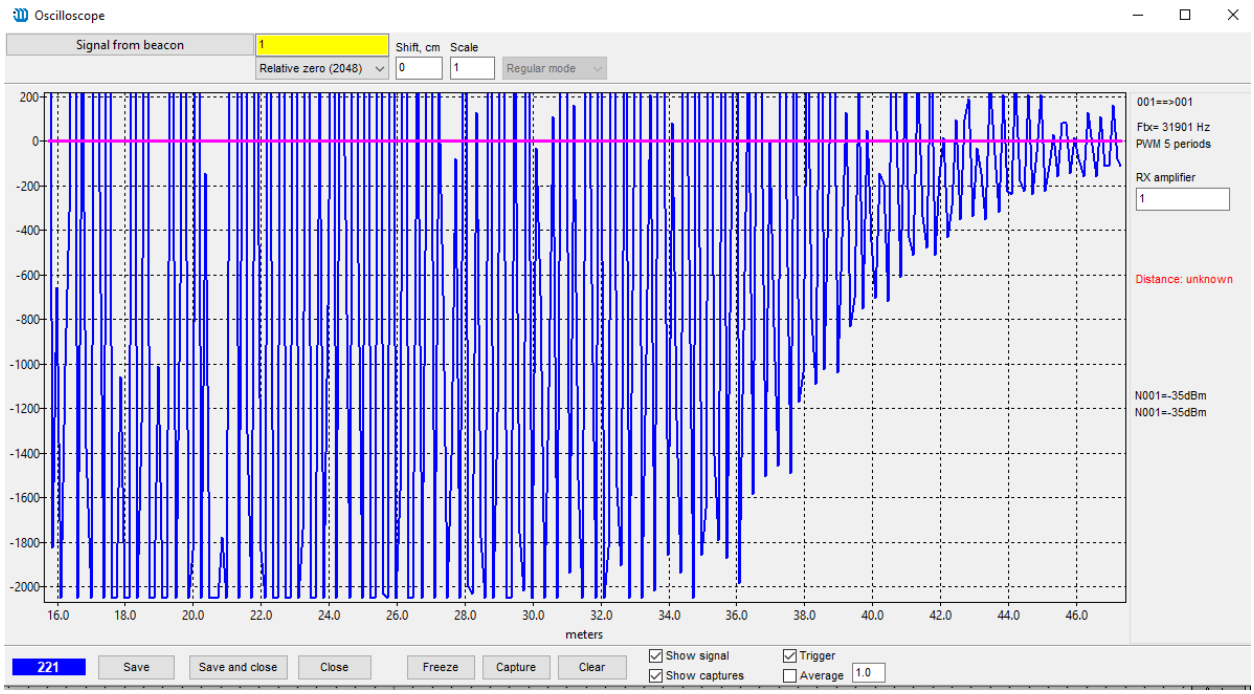


Ilustración 62: Vista de la señal extremadamente distorsionada de la baliza 1.

Lo cual era un suceso no visto anteriormente, y que podría estar relacionado con los saltos que se mencionaban. Pensando que esto era posible, se reconfiguró otra baliza para sustituirla por esta baliza con ID 1. Los resultados fueron los siguientes:

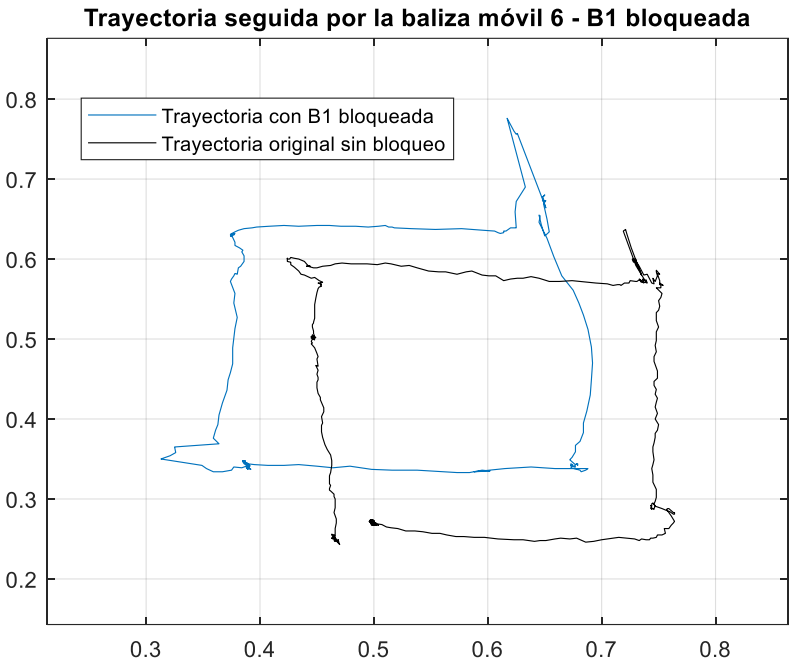


Ilustración 63: Vista de la nueva trayectoria con B1 bloqueada.

Puede verse, respecto a la ilustración 29, que los saltos han desaparecido por completo. Los picos que se ven abajo a la izquierda y arriba a la derecha se deben al movimiento manual del robot. Puede comprobarse que la trayectoria es mucho más suave que la que se veía.

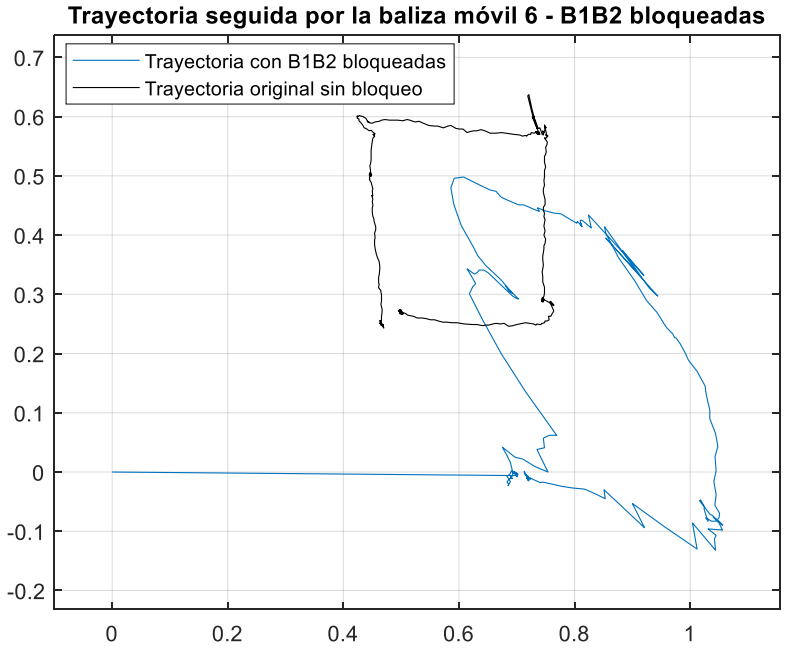


Ilustración 64: Prueba con B1 y B2 bloqueadas.

Aunque con esta configuración, como se sabe, el sistema no debe utilizarse por no poder realizar la trilateración correctamente (necesita al menos dos balizas fijas), se ha comprobado que la ruta que se dibuja es mucho menos ruidosa que la que se veía en la ilustración 30, la cual tenía numerosos saltos, debidos por un lado al malfuncionamiento de la baliza 1, y, por otro lado, a la pérdida de línea de visión con dos balizas.

6. PRUEBAS EXPERIMENTALES

Para concluir con todo lo descrito anteriormente se realizan una serie de pruebas tanto de orientación como de ruta, para ver la respuesta del robot ante las mismas.

6.1. Logging de datos.

En varias partes anteriores ya se mencionaba que se hacía un logging de los datos (recogida de datos) para su posterior procesado. En los sitios que se mencionó, simplemente se guardaba alguna variable aislada, no grupos completos de variables para su posterior análisis.

Se ha configurado una variable, de nuevo editable en la clase "DifferentialRobot", llamada "self.logSet", la cual tiene las siguientes funciones:

- Si logSet=0 el logging se encuentra desactivado.
- Si logSet=1 se hace logging de datos de reorientación (dentro del método "turn_to").

  Los datos que se guardan son:

Función de la variable	**Señal de control**	**Ángulo objetivo**	**Ángulo leído**	**Margen de ángulo**
Nombre en el programa	"u"	"self.angle"	"angulo_leido"	"self.angle_margin"

Tabla 16: Variables empleadas en el logging de "turn_to"

- Si logSet=2 se hace logging de los datos de seguimiento de ruta para los puntos especificados (dentro del método "go_to").

Función de la variable	**Nombre en el programa**
Señal de control del motor izquierdo	"uIzq"
Señal de control del motor derecho	"uDer"
Ángulo objetivo	"rot"
Ángulo leído	"angulo_leido"

Distancia al punto destino	“distance”
Margen de distancia	“self.dmargin”
Posición actual en X	“posición_leida[0]”
Posición actual en Y	“posición_leida[1]”

Tabla 17: Variables empleadas en el logging de “go_to”.

6.2. Pruebas de reorientación.

Se presentan los resultados de aplicar el control de giro de ángulo para unos ángulos concretos.

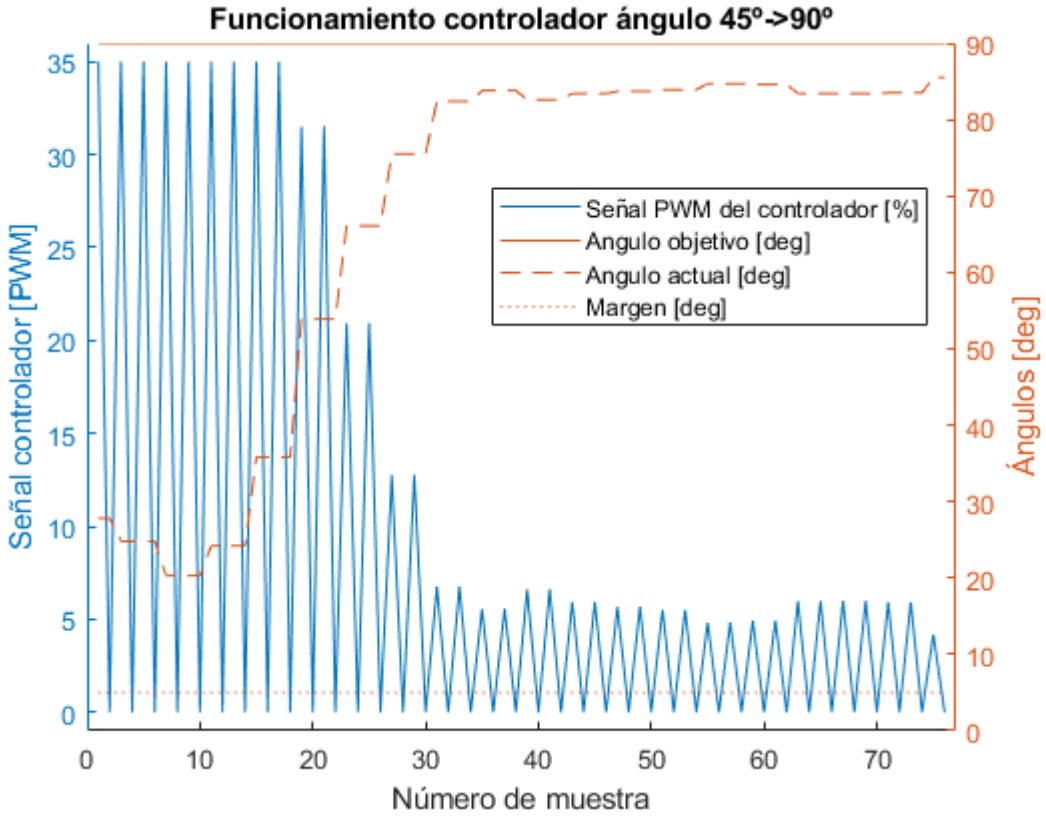


Ilustración 65: Vista de la acción del controlador PI sobre los motores para corregir el ángulo.

Puede verse que el ángulo actual se va acercando a la referencia (90º), y la prueba se detiene cuando se ha alcanzado el margen de 5º estipulado, que puede verse en línea continua en la parte inferior de la ilustración 65.

Ahora se realiza la misma prueba, pero al contrario. De 90º a 45, viendo la respuesta del controlador:

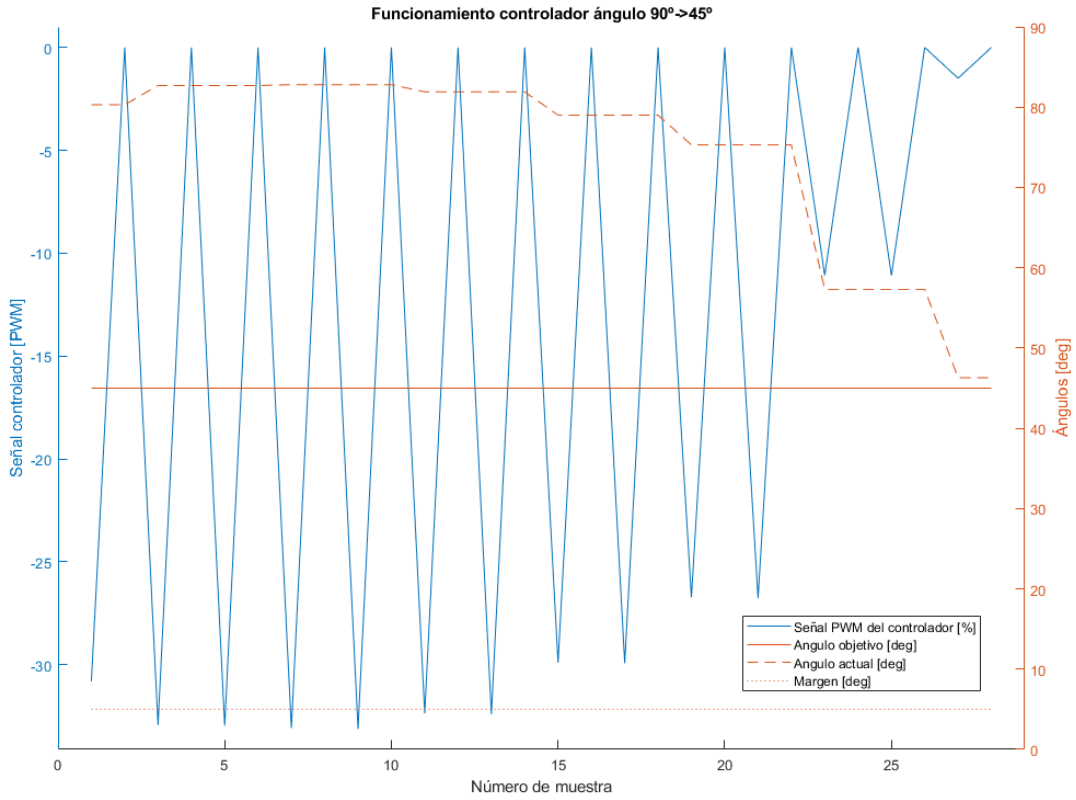


Ilustración 66: Nueva prueba donde puede verse de nuevo la correcta reorientación del robot.

Se ve lo mismo que anteriormente, cuando el robot se acerca a la referencia y pasa del margen, se detiene la prueba.

Los picos que se observan en el PWM son derivados de la acción del tren de pulsos, que varía entre 0 y el valor PWM.

Se comprueba el funcionamiento con otros ángulos para ver que el sistema de reorientación es estable:

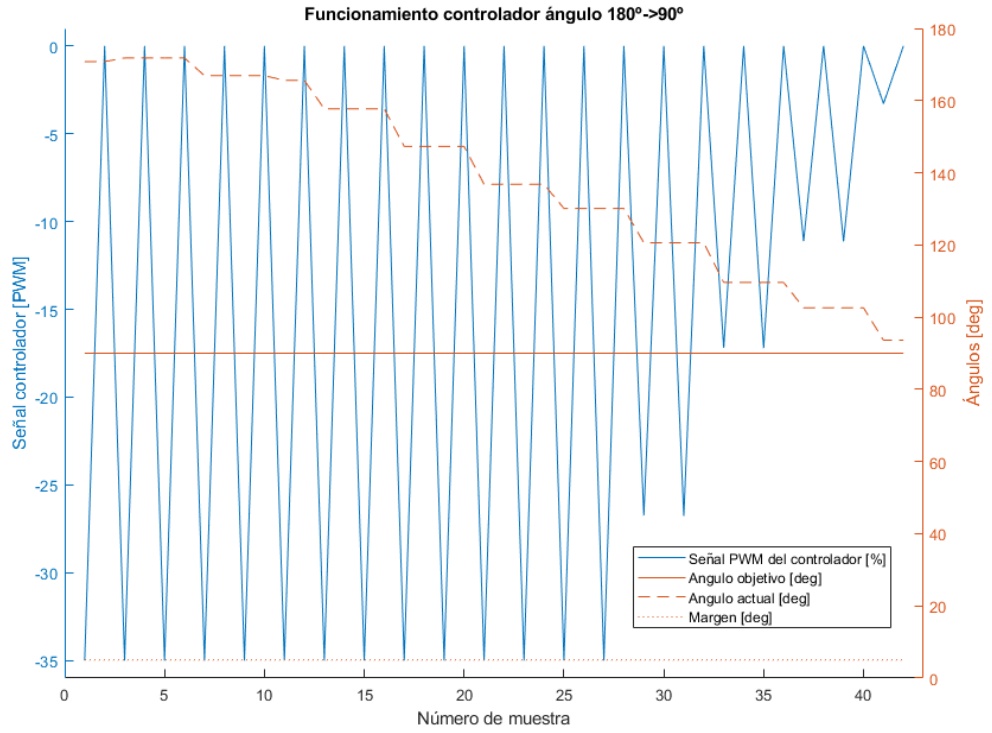


Ilustración 67: Reorientación desde 180º a 90º.

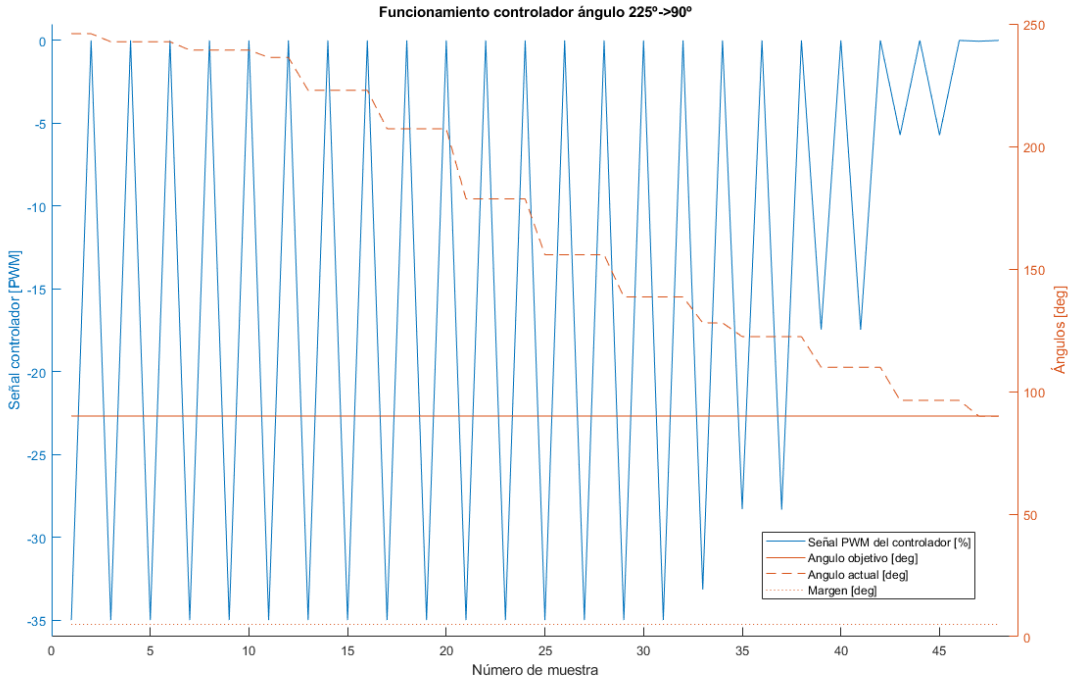


Ilustración 68: Reorientación desde 225º a 90º.

6.3. Pruebas de ruta.

Una vez se ha comprobado que la reorientación funciona correctamente, es interesante realizar la prueba que culmina con el control completo del robot: el seguimiento de una pequeña ruta.

Para no almacenar muchos datos y que sea posible visualizarlos mínimamente bien, se ha empleado una ruta con únicamente 2 puntos: ida y vuelta. Posteriormente en la exposición del trabajo se incluirá una ruta triangular, que sí se verá correctamente al tener mayor espacio de presentación.

La prueba consiste en desplazar el robot desde un punto cualquiera a un punto X=0.43 e Y=0.4, y luego volver hacia abajo, al punto X=0.43 Y=0.

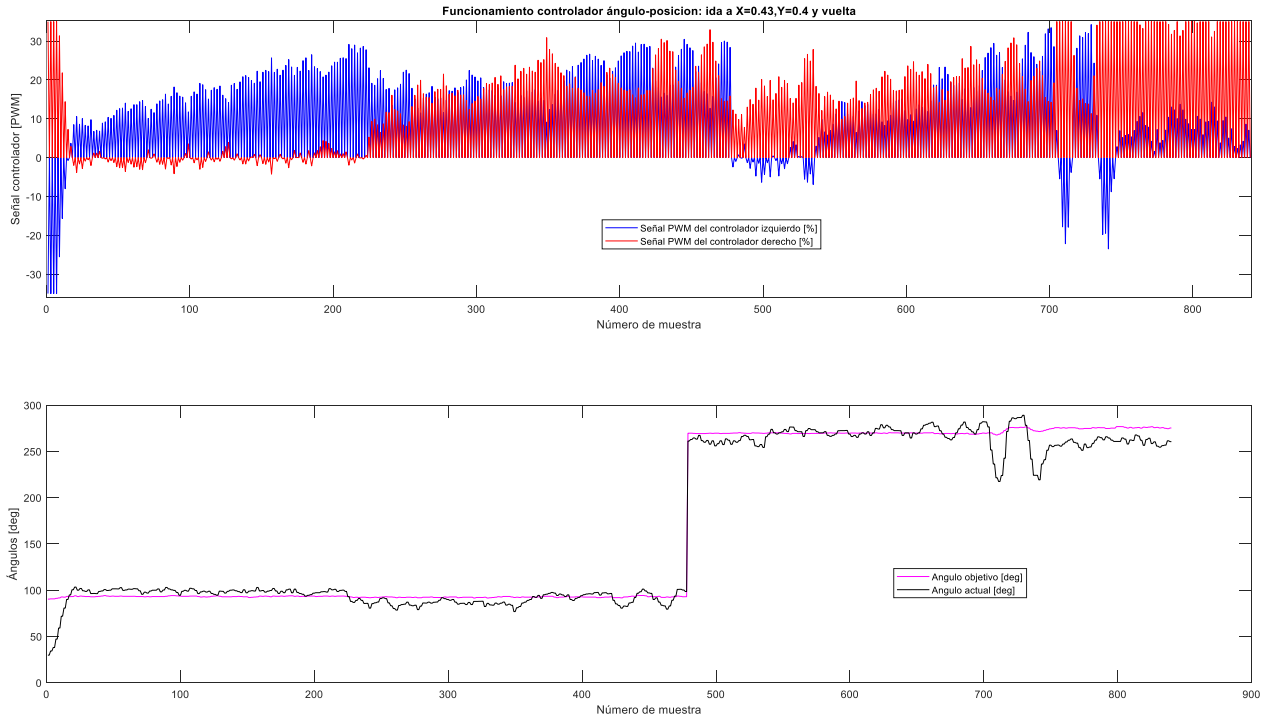


Ilustración 69: Funcionamiento del control de ángulo durante la prueba.

En la prueba realizada se ven 4 variables:

- En azul el control del motor izquierdo.
- En rojo el control del motor derecho.
- En morado la referencia de ángulo, recalculada al desviarse el robot.
- De color negro el ángulo de orientación actual del robot.

El cambio de referencia de ángulo que se ve en torno a la muestra nº 500 se debe a la reorientación del robot para volver a su punto de partida.

Puede verse que el robot sigue la referencia correctamente, compensando el ángulo. Si se observan las últimas muestras, puede verse que tiene mayor dificultad para corregirlo, y la acción de control aumenta su valor absoluto sobre el motor derecho e izquierdo para hacerlo. Esto se debe a la cercanía al punto, donde la señal de los motores ha descendido, pero para corregir el ángulo se necesita más de la que se tenía. Todo esto se ve en el análisis de la distancia.

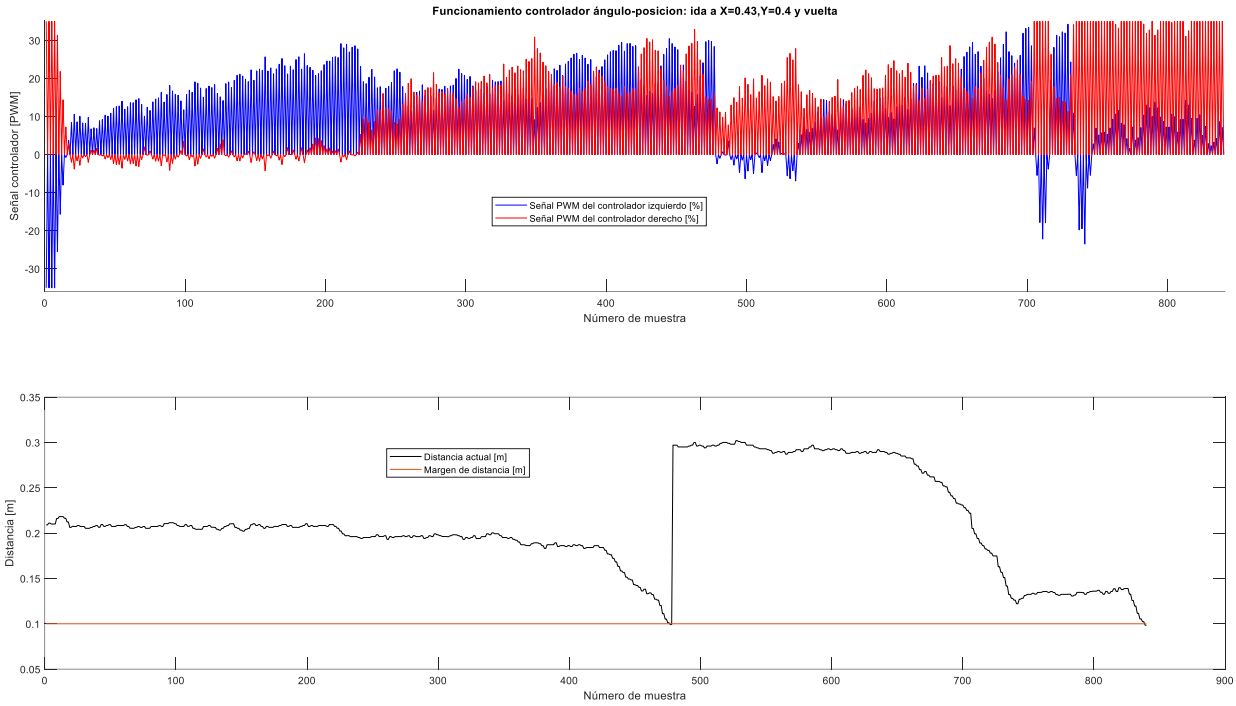


Ilustración 70: Funcionamiento del control de distancia durante la prueba.

En la prueba realizada se ven 4 señales:

- En azul y rojo las mismas señales PWM que se veían en la ilustración 69.
- En línea discontinua naranja el margen de distancia, establecido en 10 cm.
- De color negro la distancia actual hasta el punto.

Puede verse de nuevo el cambio de referencia al reorientar el robot, en torno a la muestra 500. El funcionamiento del robot es correcto, dado que va disminuyendo la distancia, manteniendo el ángulo hasta llegar al margen de distancia establecido.

Como puede verse, al final de la trayectoria de retorno la distancia al destino es muy pequeña, lo que causa que, siendo la señal de control de bajo valor, haya que aumentarla bastante para compensar ángulo y distancia, esto era lo que se comentaba en líneas anteriores.

Se repite el experimento, pero ahora en lugar de ir al punto anterior se usa una coordenada X mayor, de 0.63, manteniendo el mismo retorno en Y. Se obtiene lo siguiente:

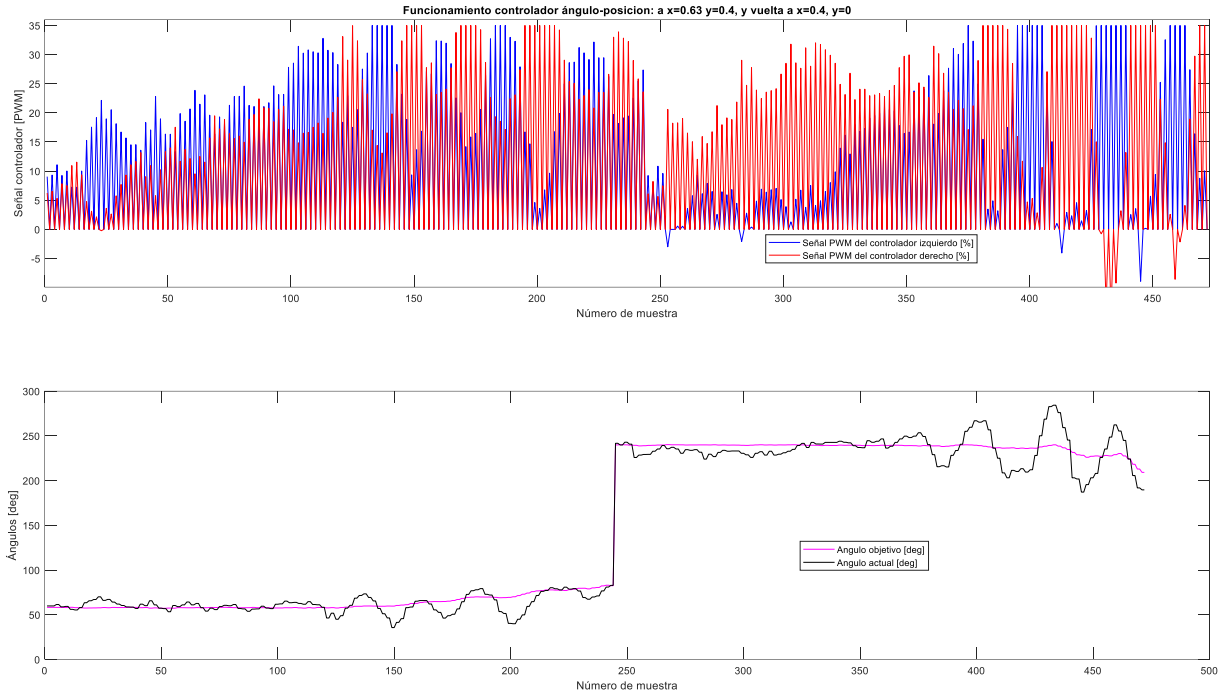


Ilustración 71: Prueba del sistema ahora para X=0.63. Prueba de ángulo.

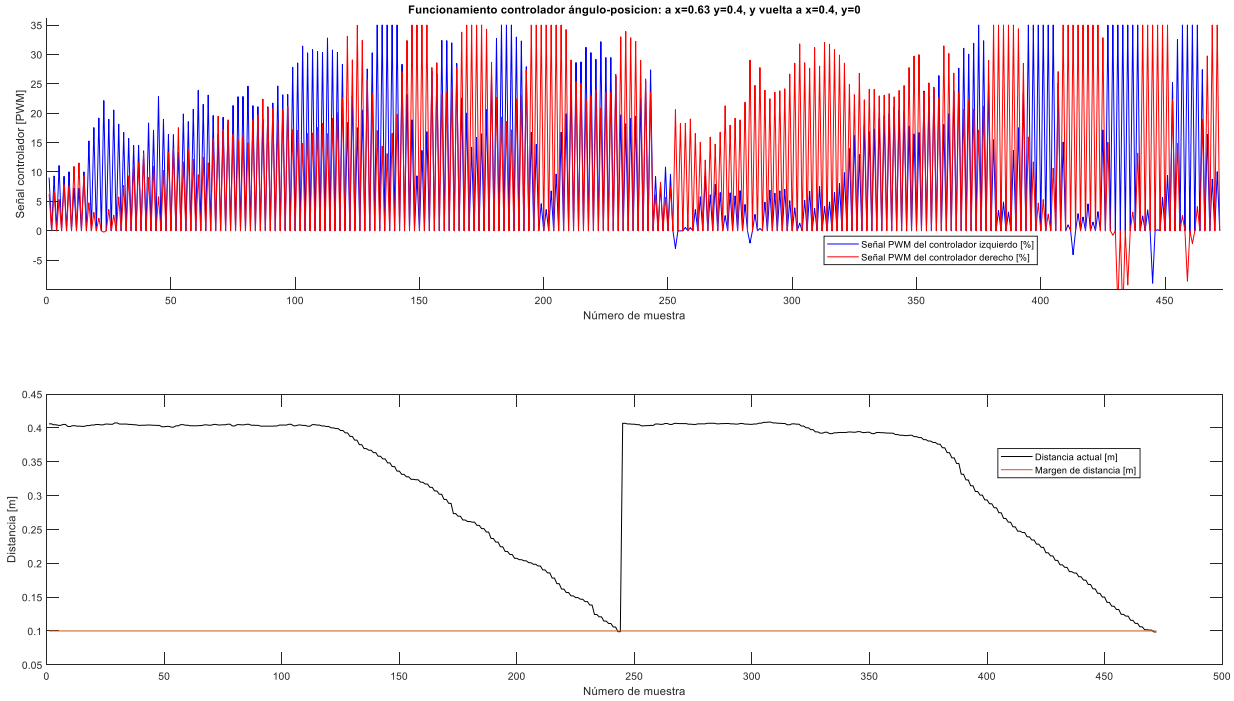


Ilustración 72: Prueba del sistema ahora para X=0.63. Prueba de distancia.

Puede verse de nuevo el correcto funcionamiento de los controladores. Esta vez se aprecia una mayor suavidad en el acercamiento al margen del punto 2 (final del experimento), a costa de un mayor cabeceo de reorientación,

que puede verse en la ilustración 71.

Las imágenes antes mostradas pueden ampliarse para ver más detalles, se han obtenido a alta resolución.

Prueba con 3 puntos con presentación de posición instantánea

Es la prueba más interesante del sistema. Se ha programado un script de Matlab que dibuja una trayectoria teórica y la compara con la real que ha seguido el robot.

Es una prueba triangular, en la que el robot debe retornar al punto de inicio, con un margen de distancia de 10cm, tal y como se veía.

El funcionamiento de los controladores no se mostrará en este documento por ser muy parecido a los ya vistos en ilustraciones anteriores. Se graficará la ruta seguida por el robot.

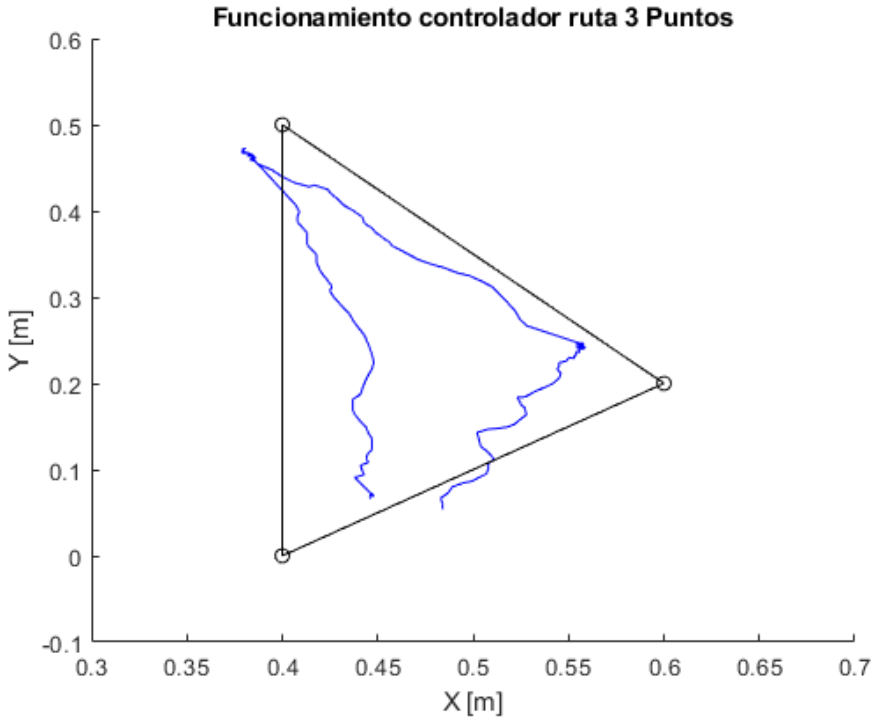


Ilustración 73: Ruta seguida por el robot a través de los 3 puntos de la ruta.

Puede verse que el robot respeta el margen establecido de 10 cm, quedándose siempre a una distancia un poco menor.

Al haberse empleado el método de recálculo de ángulo de referencia (cuando el robot se desplaza, se calcula un nuevo ángulo al punto destino, es por esto que se ve que la referencia de ángulo en los controles de ángulo no es constante), el robot al llegar al punto tendrá la última orientación que se calculó. Esto hace que siempre llegue por el camino de mínima distancia, evitando que se salga de la ruta por desviarse en exceso y perder el control de este.

Podría hacerse que se mantuviera la orientación calculada inicialmente, desde el punto de partida del robot al punto destino, pero esto no se hace por varias razones:

- Tolerancia de ángulo.

  Si el robot asegura que está reorientado correctamente hacia el punto, pero no lo está (teniendo un error de ángulo apreciable), la distancia al punto destino irá creciendo lateralmente (el robot se irá apartando de la trayectoria prevista), de manera que cuando se sobrepase el punto destino sin ser la distancia menor al margen establecido el robot perderá el control, viendo que el ángulo es el correcto, pero que la

distancia sigue aumentando constantemente.

- Desviaciones en la trayectoria.

  Relacionado con lo anterior, si durante el cabeceo que se produce en la trayectoria, el robot se orienta incorrectamente, ocurrirá lo mismo que antes, el robot verá un ángulo correcto de orientación, y si sigue avanzando se separará lateralmente de la trayectoria. Podrá desviarse del ángulo y volver a situarse, pero al ser un ángulo fijo, las pequeñas variaciones de distancia respecto al recorrido actual y el teórico podrían desembocar en el incumplimiento de llegar a una distancia dentro del margen.

Todo esto se soluciona con el control de mínima distancia que ya se comentó en el apartado 4.2. Descripción del algoritmo de control del robot.

De la trayectoria anterior se pueden hacer varias pruebas y ver que al no ser nada relacionado con movimientos preprogramados, el robot nunca sigue exactamente la misma ruta:

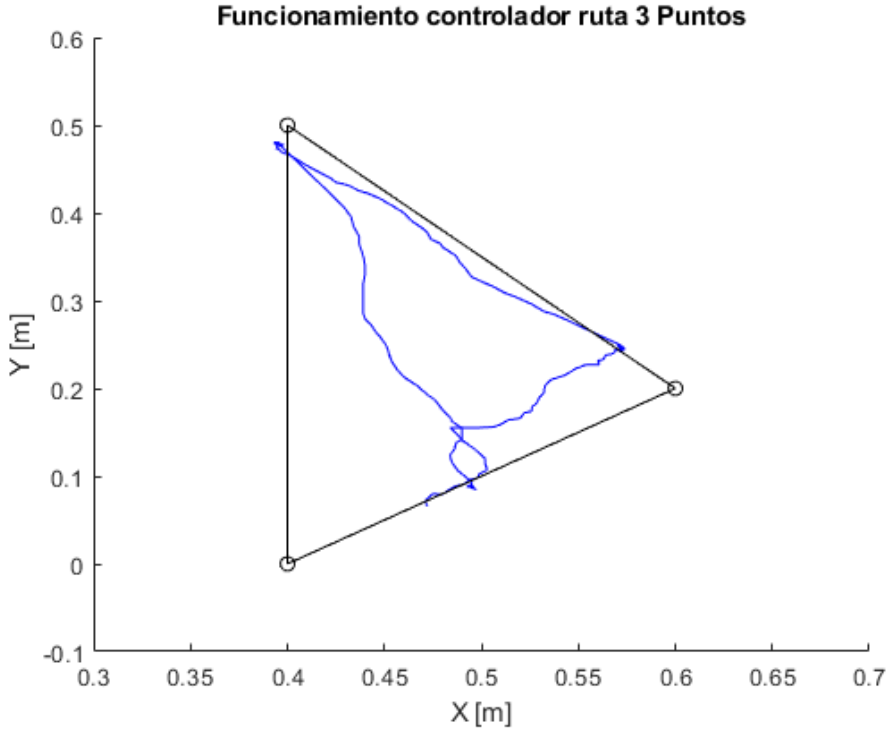


Ilustración 74: Misma ruta anterior, partiendo desde otro punto inicial.

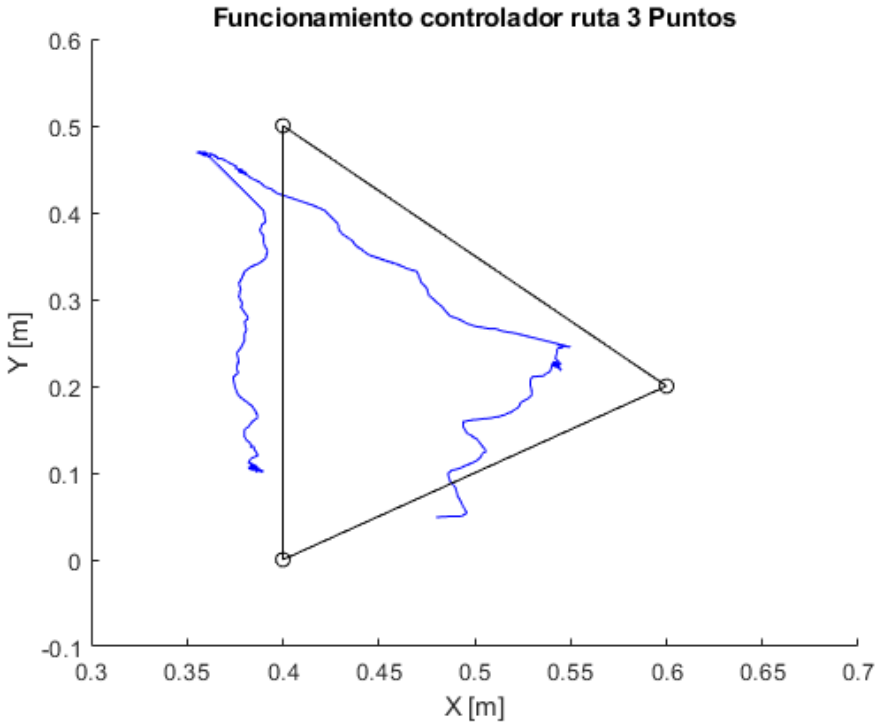


Ilustración 75: Misma prueba anterior, de nuevo desde otro punto inicial.

Puede comprobarse que la respuesta del robot es la correcta, manteniendo siempre el margen de distancia que se ha establecido.

7. VALORACIÓN ECONÓMICA

En este apartado se relaciona breve y aproximadamente el coste del proyecto, teniendo en cuenta el precio de los componentes que han sido necesarios para llevarlo a cabo.

Elemento	Precio unitario
Waveshare Jetbot	125 €
Jetson Nano 4GB	118 €
Starter Kit NIA HW 4.9	399 €
Baliza IMU HW 4.9	104 €
Arduino UNO	10 €
Sensor CNY	2 €
Componentes varios	1 €
Coste total de los componentes	759 €

Tabla 18: Tabla de precios aproximados de los componentes.

Puede verse que es un coste algo elevado, pero se debe tener en cuenta que sólo se ha empleado parte de la funcionalidad máxima de los elementos. En los siguientes apartados se verán formas de aprovechar el rendimiento del sistema, haciendo que su coste se vea menor respecto a la cantidad de aplicaciones que puede tener.

8. MEJORAS EN EL SISTEMA

Pese a que se han hecho todas las optimizaciones posibles al sistema desarrollado, existen numerosas mejoras a realizar, las cuales se relacionarán en este apartado para tenerlas en cuenta de cara a futuros nuevos usos del sistema.

8.1. Inclusión de la segunda baliza IMU (baliza 6).

Como ya se comentó en el apartado 5.4. frecuencia de actualización de las lecturas de las balizas, el uso de una segunda baliza IMU en modo *paired hedge* significaría aumentar la tasa de refresco de nueva ubicación y ángulo de manera muy significativa. Dado que en este proyecto sólo se disponía de una sola baliza IMU, esto no ha sido posible, pero se deja como opción para futuros usos.

8.2. Optimización del programa.

También se ha mencionado en el apartado 5.1.1. Optimización de la librería de control de motores, que la optimización llevada a cabo en el control de los motores no ha sido suficiente para disminuir el tiempo de procesado a algo razonable, de manera que se plantean las siguientes opciones:

- Emplear un microcontrolador sin sistema operativo para el control del robot.

  El sistema operativo es el que puede producir la ralentización del tiempo de ejecución. Programando en el microcontrolador la comunicación I2C de los motores y la comunicación serie con las balizas, se puede llegar a un resultado óptimo, aprovechando el código ya programado en Python.

- Emplear programación en C.

  Como se sabe, Python es un lenguaje más lento que C. C es lenguaje compilado, mientras que Python es interpretado. Esta diferencia puede desembocar en tiempos de ejecución mucho menores.

8.3. Cambio a motores paso a paso. Uso de *encoders* de posición.

El cambio de los motores con reductora actuales a motores paso a paso puede llevar a un control mucho más suave, en el que se conoce con exactitud a qué ángulo girado corresponde un paso. Se debería cambiar la estructura de los controladores, la forma de accionar los motores, pero el cuerpo del programa se podría mantener similar.

Otra opción es la de mantener los motores actuales, incluyendo un *encoder* de posición en cada rueda, de manera que se sabe qué ángulo ha girado cada rueda, lo mismo que ocurriría con un motor paso a paso. Teniendo el ángulo girado, ya se tiene, como se demostró en el apartado 5.1.1. Optimización de la librería de control de motores, la distancia que avanza el robot.

8.4. Mejora en el sistema mecánico.

Ya se mencionó en el apartado 5.5. Superficies. Sistema mecánico., la dificultad de movimiento del robot en algunos espacios.

Dado que se quiere mantener la topología de robot diferencial (girar y avanzar sólo con 2 ruedas motrices), la mejora en el movimiento puede consistir en sustituir las pequeñas ruedas de soporte delantera y trasera por unas ruedas locas más grandes, usando goma como material de contacto con el suelo, en lugar de metal (configuración actual), que permitan sobrepasar surcos, evitando el estancamiento del robot.

8.5. Indicador de batería.

Este es uno de los defectos más considerables del robot, y es que no es capaz de indicar la batería que tiene ni de calcular el tiempo estimado de funcionamiento que le queda para el consumo y batería actuales.

Esto se puede resolver comprobando si la interfaz de E/S del robot tiene alguna entrada analógica con la que leer tensión, y preparar un circuito de adaptación de la tensión de la batería a la admitida por esta entrada analógica, permitiendo saber el % de batería por ejemplo en función de la tensión.

El análisis según tensión es una idea sencilla pero efectiva, dado que el sistema de protección de las baterías que incorpora el robot (BMS) se basa en que la batería no rebase un umbral de tensión de descarga, desconectando la carga en ese momento. Sabiendo la tensión actual se puede saber cuándo el controlador BMS desconectará la carga y por tanto se apagará el robot.

Para estimar el tiempo de uso restante se deben tener más características en cuenta:

- Capacidad nominal de la batería.
- Intensidad instantánea de descarga.
- Tensión.

Con todo esto se puede estimar la capacidad restante, combinándola con la tensión, que era el sistema de detección de descarga sencillo, y estimar el tiempo de encendido que le queda.

9. FUTURAS IMPLEMENTACIONES

En el texto anterior se ha mencionado la capacidad del robot de usar aplicaciones de IA, lo cual, puede combinarse con el sistema de posicionamiento para lograr resultados muy buenos. Se plantean las siguientes opciones:

- Fusión de sensores de las balizas con ángulo detectado por diferencia de imágenes.

  Igual que se puede realizar una fusión de distintos sensores para estimar un ángulo *yaw* de giro, éste también es posible estimarlo mediante la cámara que posee el robot, aplicando un algoritmo de cálculo de imagen girada.

  El procesamiento de imágenes puede ser lento, pero este robot está optimizado para aplicaciones de IA, por lo que podría ser buena idea probar su funcionamiento y latencia.

- Evasión de obstáculos combinada con seguimiento de ruta.

  En este caso podría usarse la cámara para intercalar ambos controles: el de posición + ángulo con el de detección de objetos con la cámara. Esto podría completar el sistema pudiéndolo emplear en numerosas aplicaciones.

  Se deberá de implementar algún algoritmo de optimización de caminos, dado que cuando encuentre el objeto pueda recalcular un nuevo camino hacia el objetivo [36].

- Aplicación de alguna de las mejoras mencionadas en el apartado 8. Mejoras en el sistema.

  Como mínimo las que pueden dar mejor resultado con menor inversión de tiempo y dinero: la inclusión de una baliza IMU y la reescritura del programa en otro lenguaje. El programa es fácilmente configurable para una nueva baliza y la forma de obtener el ángulo (que ahora será mediante la fusión IMU). También se debe comprobar la frecuencia de actualización y configurar una fija en el tiempo de cálculo de los controladores que se vio en los diagramas de flujo de "turn_to" y "go_to".

- Pruebas exhaustivas de los algoritmos presentes.

  Aunque como se ha mencionado en el texto, algunos de los métodos implementados (filtro de Kalman, cálculo de *yaw* a partir de cuaternios, uso de brújula, etc.) han sido desplazados por el uso del método *paired hedge*, no se descarta la posibilidad de hacer un estudio exhaustivo de su posibilidad de uso, dado que son métodos que podrían dar buen resultado encontrando la manera de calibrarlos correctamente.

10. CONCLUSIONES

A continuación, se relaciona el grado de consecución de los objetivos propuestos al principio del trabajo. También se harán una serie de comentarios adicionales.

- Objetivo principal: Puesta a punto de un sistema de localización de un objeto móvil en interiores.

  Este objetivo se ha alcanzado con creces, además de la posición ha sido posible estimar la orientación del robot, útil en las implementaciones de algoritmos de control del mismo.

  Además de la medida de posición básica, devuelta por uno de los ejemplos de código que proporciona Marvelmind, también se han explorado otras posibilidades, por ejemplo: la posición mediante su fusión IMU interna, la estimación del ángulo en base a una fusión programada, la orientación en base al interpretado de los datos de fusión (cuaternios), etc.

- Dibujado de la trayectoria del objeto móvil basada en las coordenadas devueltas por el sistema de localización.

  Tanto en las trayectorias cuadradas como en la prueba final con la trayectoria triangular, se ha visto que de nuevo este objetivo se ha cumplido, haciendo uso de scripts programados en Matlab.

- Comprobación de las desviaciones en las medidas frente a las coordinadas reales.

  Todo esto se ha realizado en el apartado 3.3. Caracterización de las balizas radio., en el cual se calculaban varianzas y desviaciones típicas de posiciones estáticas, junto con desviaciones respecto a posiciones de una ruta (ensayo de la trayectoria cuadrada).

- Uso del sistema de localización para el guiado de un robot móvil.

  Este objetivo se ha cumplido pese a todas las dificultades enfrentadas, aunque con un resultado inferior al óptimo, empleando el PWM en tren de pulsos, que proporcionaba el tiempo necesario entre movimientos para obtener la nueva lectura de las balizas y poder hacer el cálculo de salidas de controladores.

- Búsqueda de límites de funcionamiento del sistema de localización. Determinación de condiciones de posición fiable o poco fiable.

  Todo esto se ha realizado también en el apartado 3.3. Caracterización de las balizas radio., donde se obstruían las líneas de visión de las balizas y se veía su efecto sobre la trayectoria. Aunque en este apartado no se vio correctamente debido a la baliza 1 estropeada, pudo verse en el apartado 5.6. *Jumps*. Efecto de una baliza estropeada., donde se cambió la baliza, viendo ya sólo el efecto de pérdida de línea de visión.

En general puede verse que los hitos se han cumplido, de manera que el resultado del trabajo es bastante bueno.

Como se ha comentado es mejorable teniendo en cuenta todo lo comentado, pero con los recursos presentes se ha llegado a la mejor solución posible.

Se han puesto en relieve varias conclusiones derivadas de toda la realización:

- Dificultad de la creación de un sistema de posicionamiento fiable.

- Necesidad de exploración de todas las posibilidades posibles para la valoración de la óptima, sin descartar ninguna.
- Necesidad de comprobación de los equipos ante funcionamientos inesperados (*jumps*).
- Indispensable conocimiento de arquitectura del código manejado para poder alcanzar la máxima optimización.

Finalmente indicar que, el trabajo realizado deja, tal y como se había mencionado, muchas posibilidades de ampliación y mejora, sentando el inicio a trabajos escalables en dificultad con infinidad de aplicaciones.

REFERENCIAS

[1] F. d. Zárate, «El uso de robots industriales se dispara en Estados Unidos y crece un 6% en España,» *El País,* 12 Junio 2022.

[2] A. P. Tella, «ALL the Amazon Warehouse Robots: From Fulfillment to Last-Mile Delivery,» AGVnetwork, [En línea]. Available: https://www.agvnetwork.com/robots-amazon. [Último acceso: 16 Junio 2022].

[3] «PNGwing,» PNGwing. [En línea]. [Último acceso: 27 Junio 2022].

[4] R. Negenborn, «Robot Localization and Kalman Filters,» Agosto 2003. [En línea]. Available: http://www.negenborn.net/kal_loc/. [Último acceso: 24 Mayo 2022].

[5] Revista Española de Electrónica, «Cálculo de la distancia. Cómo utilizar un dispositivo de ultrasonidos para detectar la distancia,» Revista Espñola de Electrónica, 10 Octubre 2014. [En línea]. Available: https://www.redeweb.com/articulos/componentes/calculo-de-la-distancia-como-utilizar-un-dispositivo-de-ultrasonidos-para-detectar-la-distancia/?cn-reloaded=1&cn-reloaded=1. [Último acceso: 20 Enero 2022].

[6] R. S. Mínguez, «Sensor de nivel de agua con ESP8266,» 2 Septiembre 2016. [En línea]. Available: http://rubensm.com/tag/ultrasonidos/. [Último acceso: 15 Mayo 2022].

[7] M. Ylianttila, «Time of Arrival geolocation method,» ResearchGate, [En línea]. Available: https://www.researchgate.net/figure/Time-of-Arrival-geolocation-method_fig7_221198350. [Último acceso: 15 Mayo 2022].

[8] C. G. T. S. A. D. R. L. B. R. C. H. Andrés Felipe García, «Algoritmos de Radiolocalización basados en ToA,» *Revista Ingeniería y Región,* nº 14(2), pp. 9-22, 2015.

[9] F. B. Barba, *TFM. Estudio de algoritmos de localización en interiores para tecnologías móviles de última generación,* Madrid. Universidad Politécnica de Madrid., 2012.

[10] Favendo, «Angle of Arrival – High-precision positioning with the AoA method,» Favendo, [En línea]. Available: https://www.favendo.com/angle-of-arrival#:~:text=AoA%20stands%20for%20Angle%20of,of%20the%20receiving%20radio%20signals.. [Último acceso: 27 Junio 2022].

[11] Data Alliance, «Banda de frecuencia 433MHz: 412-440 MHz – Usos,» [En línea]. Available: https://es.data-alliance.net/433mhz-frequency-band/. [Último acceso: 25 Mayo 2022].

[12] F. Nawazi, «FS1000A 433Mhz RF Transmitter Receiver Modules,» Circuits DIY, Febrero 2022. [En línea]. Available: https://www.circuits-diy.com/fs1000a-433mhz-rf-transmitter-receiver-modules/. [Último acceso: 25 Mayo 2022].

[13] Wiki LIC, «Variación del desplazamiento angular de un vehículo en movimiento,» Wikimedia Commons, 6 Diciembre 2019. [En línea]. Available: https://commons.wikimedia.org/w/index.php?curid=84646788.

[Último acceso: 10 Febrero 2022].

[14] G. B. Greg Welch, «The Kalman Filter,» University of Central Florida, University of North Carolina, 7 Julio 2016. [En línea]. Available: http://www.cs.unc.edu/~welch/kalman/. [Último acceso: 25 Mayo 2022].

[15] A. Becker, «Kalman Filter in One Dimension,» [En línea]. Available: https://www.kalmanfilter.net/kalman1d.html. [Último acceso: 27 Junio 2022].

[16] K. Steven, Fundamentals of Statistical Processing, Volume I: Estimation Theory, Financial Times Prentice Hall, 1993.

[17] B. M. L. Kamlofsky Jorge A., «Los cuaterniones en visión robótica,» Universidad Abierta Interamericana, Buenos Aires, Argentina, 2000.

[18] J. L. B. Claraco, «A tutorial on SE(3) transformation parametrizations and on-manifold optimization,» Universidad de Málaga, Málaga, 2010.

[19] F. V. Serrano, *Laboratorio de control de procesos. Aspectos prácticos del PID.,* Córdoba, 2021.

[20] S. A. C. Giraldo, «Controlador PI por Asignación de Polos,» Control Automatico Educación, [En línea]. Available: https://controlautomaticoeducacion.com/control-realimentado/controlador-pi-por-asignacion-de-polos/. [Último acceso: 27 Junio 2022].

[21] M. Robotics, «Precise (±2cm) Indoor Positioning and Navigation,» Marvelmind, [En línea]. Available: https://marvelmind.com/. [Último acceso: 10 Diciembre 2021].

[22] M. Robotics, «marvelmind.py,» GitHub, Abril 2022. [En línea]. Available: https://github.com/MarvelmindRobotics/marvelmind.py. [Último acceso: 15 Enero 2022].

[23] Arduino Forum, «Medir RPM ESP-32,» Julio 2021. [En línea]. Available: https://forum.arduino.cc/t/medir-rpm-esp-32/883095. [Último acceso: 22 Marzo 2022].

[24] MK Electrónica, «Sensor IR de reflexión CNY70,» MK Electrónica, [En línea]. Available: https://mkelectronica.com/producto/sensor-ir-reflexion-cny70/. [Último acceso: 22 Marzo 2022].

[25] Chegg, «Integrator Windup,» Chegg, [En línea]. Available: https://www.chegg.com/homework-help/questions-and-answers/2-integrator-windup-construct-pi-controller-system-simulation-program-illustrates-integrat-q86091620. [Último acceso: 27 Junio 2022].

[26] tokk-nv, «Waveshare JetBot kit's motors do not spin,» GitHub, 10 Noviembre 2020. [En línea]. Available: https://github.com/NVIDIA-AI-IOT/jetbot/issues/314. [Último acceso: 5 Junio 2022].

[27] jaydub NVIDIA, «My Jetbot will no longer boot with the GUI enabled,» NVIDIA Forum Developers, 3 Noviembre 2020. [En línea]. Available: https://forums.developer.nvidia.com/t/my-jetbot-will-no-longer-boot-with-the-gui-enabled/160243. [Último acceso: 5 Junio 2022].

[28] tokk-nv, «Dockerfile jetbot,» GitHub, 8 Enero 2021. [En línea]. Available: https://github.com/NVIDIA-AI-IOT/jetbot/blob/master/docker/base/Dockerfile#L72. [Último acceso: 7 Junio 2022].

[29] S. K. Romberg, «OpenStax CNX,» [En línea]. Available: https://cnx.org/contents/hoOAwzIw@1.7:9lzzjMl0@9/Sampling-Theorem. [Último acceso: 27 Junio

2022].

[30] F. S. J. C. P. J. G. Antonio R. Jimenez, «Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU,» Scinapse, 11 Marzo 2010. [En línea]. Available: https://www.scinapse.io/papers/1993793087. [Último acceso: 15 Junio 2022].

[31] «Yaw Drift,» Kauai Labs, Inc., 2020. [En línea]. Available: https://pdocs.kauailabs.com/navx-mxp/guidance/yaw-drift/. [Último acceso: 15 Junio 2022].

[32] M. L. Hoang, «Yaw/Heading optimization by drift elimination on MEMS gyroscope,» ScienceDirect, 1 Julio 2021. [En línea]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0924424721001540?via%3Dihub. [Último acceso: 15 Junio 2022].

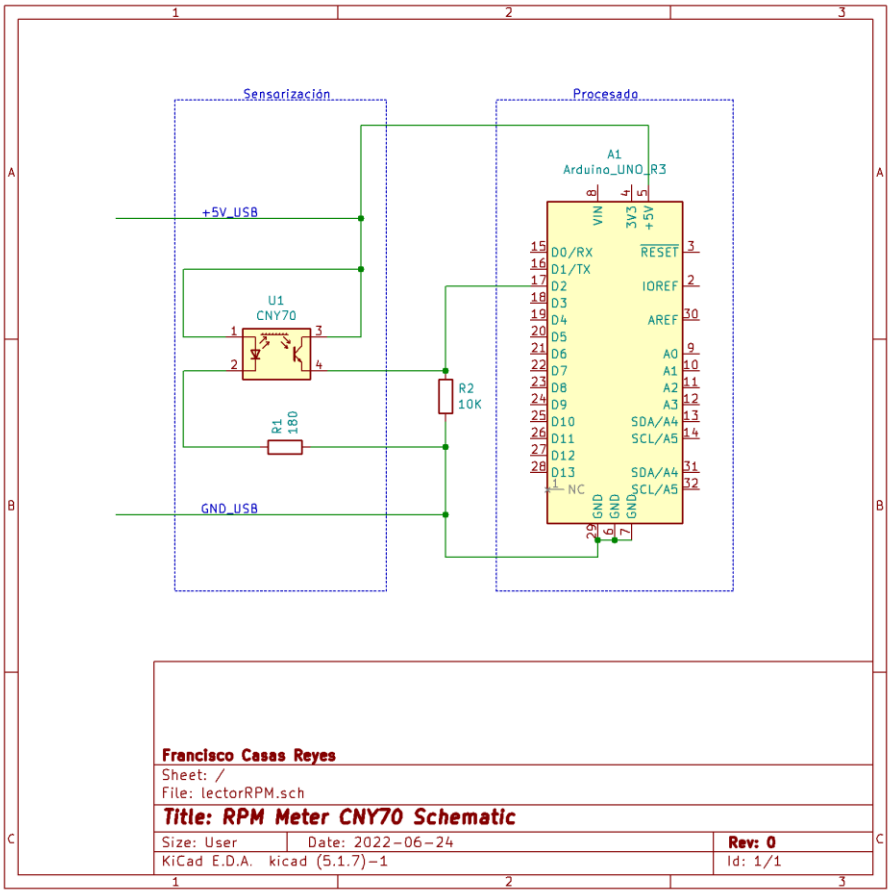
[33] Naylamp, «Tutorial Magnetómetro HMC5883L,» Naylamp, [En línea]. Available: https://naylampmechatronics.com/blog/49_tutorial-magnetometro-hmc5883l.html. [Último acceso: 16 Junio 2022].

[34] Learn Adafruit, «Making Tracks,» Learn Adafruit, 10 Marzo 2013. [En línea]. Available: https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/making-tracks. [Último acceso: 16 Junio 2022].

[35] Marvelmind Robotics, Marvelmind IndoorGPS. Typical mistakes with Precise Indoor "GPS", 2021.

[36] M. R. Arahal, Percepción en Automática y Robótica. Bloque 4., Sevilla, 2022.

ANEXO I: ESQUEMA DEL MEDIDOR DE RPM.



ANEXO II: CÓDIGO ARDUINO DEL MEDIDOR RPM.

```
//Sketch para usar como lector de RPM en una rueda, para posteriormente
calcular la velocidad lineal
//Debe ser una combinación de interrupcion+timer, para poder determinar una
frecuencia mas exacta
//Se usará el timer 0 de 8 bits, con 7812.5KHz de frecuencia de pulso
(prescaler a 1024)
//Cada pulso son 0.128ms -> Periodo total=32.64ms

//No se puede usar de 8 bits, con 16 bits y prescaler 1024, máximo periodo
medido: 0xFFFF(65535)*0.128ms= 8388ms
//8.38 es más que suficiente, ya que para el 25% de PWM se esperan como
máximo 400ms de periodo

//Autor: Fran Casas
//Revisión 1:
//-Corregida lectura de RPM errónea en subida de interrupcion, configurada a
bajada de nivel
//-Corregida lectura de pulsos del timer que no se actualizaba en while(1)

volatile unsigned int nCycles=0; //Guardada en RAM para no ser alterada
volatile unsigned int nCyclesPrev=0;
float tRevSegs=0, freqRev, freqRevMin;

//Prototipado
void contajeRev(void);

void setup()
{
  Serial.begin(9600);
  attachInterrupt(0,contajeRev,FALLING);  // Interrupcion 0 (pin2)
  //Timer config
  TCCR1A = 0; //Ext pins not used, 8 bit register at 0
  TCCR1B = 0; //Normal mode, 16 bits used, from 0 to MAX=0xFFFF
  TCCR1B |= 0b00000101; //1024 prescaler
  TCNT1 = 0; //Timer initial reset
  //For reading values, read TCNT1(16bit) register

}

void loop()
{
  //Serial.println(255*digitalRead(2));
  if (nCycles != nCyclesPrev) //Cambio de RPS
  {
    nCyclesPrev = nCycles;
    Serial.print("RPM medidas: ");
    tRevSegs=nCycles*0.000128;
    freqRev=1/tRevSegs; //Rev/s
    freqRevMin=freqRev*60; //Rev/min
    Serial.println(freqRevMin);
  }
}

void contajeRev()
```

```
{
  nCycles=TCNT1; //Saves the elapsed time on timer
  TCNT1=0; //Timer reset
}
```

ANEXO III: CÓDIGO DE PRUEBA DE LOS MOTORES.

```
# Código inicial de ejemplo para mover los motores
# En la librería final DifferentialRobot ya se emplea la clase
# "Robot" modificada (archivo libMotors_AdafruitBase_v0.py)

from jetbot import Robot
import time

robot = Robot()

testValue0=25
testValue1=30
testValue2=50
testValue3=75
testValue4=100
timeTest=int(input('Introduce el tiempo que quieres tener el motor
encendido[s]: '))

robot.left_motor.value = testValue0/100
robot.right_motor.value = testValue0/100
time.sleep(timeTest)

robot.left_motor.value = testValue1/100
robot.right_motor.value = testValue1/100
time.sleep(timeTest)

robot.left_motor.value = testValue2/100
robot.right_motor.value = testValue2/100
time.sleep(timeTest)

robot.left_motor.value = testValue3/100
robot.right_motor.value = testValue3/100
time.sleep(timeTest)

robot.left_motor.value = testValue4/100
robot.right_motor.value = testValue4/100
time.sleep(timeTest)
```

ANEXO IV: CÓDIGO PRINCIPAL "ROBOTCLASS.PY"

```
#############################################################
#                                                           #
# Class for movement algorithm of Marvel IndoorGPS system   #
# combined with Jetbot                                      #
#                                                           #
# Author: Fran Casas                                        #
#                                                           #
# Version: 10                                               #
# Fecha: 24-06-2022                                         #
#                                                           #
#                                                           #
#############################################################

##CHANGELOG##
#V0 (13-06-2022):
# Definidos atributos de la clase DifferentialRobot:
#  DifferentialRobot(kp=0.5, ki=0.5, dead_zone=[0]):
# .testWork(val=1) -- si se usa este atributo la clase indicará que se han
cargado correctamente los módulos
# .set_dead_zone -- se encuentra en desarrollo
# .obtener_posicion_angulo -- carga en las variables globales RAWimu e
IMUfusion los valores devueltos por la baliza
# .obtener_yaw -- calcula el angulo z según los cuateriones (way=True) o
según el giroscopio + acelerómetro (way=False)
# .turn_to -- incorpora un PI de corrección de ángulo, para el primer giro
(falta ver efecto de la deriva)
# .go_to -- incorpora un doble PI para alcanzar la ubicación corrigiendo
angulo + distancia al objetivo
# .take_sample -- guarda una imagen en la carpeta que se indique

#V1:
# -Corregido ángulo objetivo (initial_rot) sabiendo que queremos que el eje Y
es el que gire (es donde apuntará la parte
# -delantera del robot con la cámara)
# -Cambio de signos en los PIs para obedecer el sentido de giro de las ruedas
al corregir el movimiento (stop())

#V2:
# -Se corrige el paso de parámetros a las definiciones internas de la clase

#V3:
# -Se cambia la topología de las balizas a paired hedge (mediante cambio en
dashboard)
# -Se modifica la clase MarvelmindHedge para incluir un método que devuelve
exclusivamente ángulo (.return_angle)
# -Se adapta el código al nuevo método de obtención de orientación

#V4:
# -Arreglos menores: limites no aplicados en saturacion de PID pos + ang
# -Cambio de aplicación de señal en lugar de PWM continua a tren de pulsos
# -Incluida posibilidad de hacer logging de datos editando la variable logSet
# -Optimización de tiempo de giro según cuadrante
# -Se incorpora una variable SS que permite al controlador de posicion +
angulo
```

```
# funcionar en modo corta distancia (si SS=True) o en espacio amplio
(SS=False)
# -Se incluyen puntos de control para ver salida de controladores
# -Se corrige la clase MarvelmindHedge (marvelmind_mod.py) para no devolver
angulos negativos
# al devolver el angulo del eje Y robot (direccion de avance)
# -Dentro de dashboard se incrementa la velocidad de la comunciación radio a
154kbps para
# incrementar la frecuencia de actualizacion de medidas
# -Se sustituye la baliza N1 por otra, debido a su malfuncionamiento
# -Corrección del doble PI de posicion+angulo, que no actualizaba la nueva
direccion
# a la que reorientar el robot
# -Se limita la señal PWM en turn_to a 0.3 para evitar altas velocidades
(editable
# en la variable PWMmaxP), y a 0.35 en go_to, para evitar lo mismo pero
permitir una velocidad
# algo mayor (no excesiva)

#V5:
# -Se incluye el guardado de posicion instantánea en logging de ruta
(angulo+posicion)
# -Se elimina un retardo en el método go_to() que provocaba mayor lentitud en
la ejecucion de ruta

#V6:
# -Se corrigen los fallos de guardado puntuales del csv debidos a no
especificar ruta
# -Se incluye un método para detener tanto el puerto serie como la ejecución
en curso

#V7:
# -Se cambia la estructura de los controladores para poder usarlos con PWM
continuo y disminuir la latencia
# -Se usan controladores separados para ángulo y posición (antes compartían
parámetros)
# -Se usa una variable debug para devolver resultados por pantalla si se
desea
# -Se optimiza levemente el código para eliminar operaciones repetidas

#V8:
# -Se elimina el uso de variables globales, al usar variables internas de la
clase
# -Se crea una clase externa que alberga los PI

#V9:
# -Se reconstruye la librería Motor.py que alberga un controlador Adafruit
para
#  conseguir una mejor velocidad de respuesta, eliminando código inútil para
#  la aplicación que se desarrolla, dejando únicamente lo esencial para
activar
#  la señal PWM, cuya frecuencia ya se encuentra preconfigurada en la
librería
#  Adafruit
# -Adaptación del código a la nueva librería y forma de invocar el control de
los motores

#V10:
# -Se reconstruye la forma de actuar del PI para el ángulo, que a veces tenía
un funcionamiento errático
# -Se reconstruye la rutina de reorientación del robot completa
```

```
# -Se vuelve a activar la forma tren de pulsos PWM dado que factualizacion es
lenta debido al accionamiento
#  de los motores por parte de la librería ya optimizada.
# -Se completa el método .set_dead_zone con otro método .check_point que
comprueba si el punto especificado
#  está dentro de zonas prohibidas, indicandolo

# IPython Libraries for display and widgets
import re
import ipywidgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

#from turtle import up
# Camera and Motor Interface for JetBot
from jetbot import Camera, bgr8_to_jpeg
from libMotors_AdafruitBase_v0 import Motor
from marvelmind_mod import MarvelmindHedge
import math
import sys
import array
import numpy as np
from uuid import uuid1
import os
import json
import glob
import datetime
import cv2
import time
import csv
from classPI import classPI

#En primer lugar se debe hacer la lectura inicial tanto de coordenadas como
de
#ángulo, para calcular: la diferencia de coordenadas y el incremento de
ángulo

class DifferentialRobot():
    def __init__ (self, kp,ki,dead_zone):
        self.ki = ki #cte integral del PI [kiAng, kiPos]
        self.kp = kp #cte proporcional del PI [kpAng, kpPos]
        self.dead_zone = dead_zone
        self.hedge = MarvelmindHedge(tty = "/dev/ttyACM0", adr=6,
debug=False) # create MarvellogSetmindHedge thread
        if (len(sys.argv)>1):
            self.hedge.tty= sys.argv[1]
        self.hedge.start() # start thread

        #Control de motores
        self.mIzq=Motor(channel=1)
        self.mDer=Motor(channel=2)

        #Logging setup
        self.logSet=0 #Si logSet=0 no se hace logging de datos
                #Si logSet=1 se guardan los datos de posicionamiento a ángulo
                #Si logSet=2 se guardan los datos de angulo+posicion
        self.data=[] #Guardará los datos que se quieran colocar al CSV
        self.debug=True #Permite que se hagan todos los print de los metodos
        self.debugTime=True #Debug de tiempos de ejecucion

        #Memoria de KFs
```

```
        self.pos_x_prev=0
        self.pos_y_prev=0
        self.ang_int_prev=0
        self.timePrev=0
        self.accAngZprev=0
        self.y_pred1 = 0
        self.m_err1 = 0
        self.a1 = 0
        self.sigma_u1=2*10**(-7) #Varianza de la señal de posicion
        self.sigma_n1=0.01 #Varianza del ruido aplicado a la señal de entrada
        self.y_pred2 = 0
        self.m_err2 = 0
        self.a2 = 0
        self.sigma_u2=41.699 #Varianza de la señal de brujula
        self.sigma_n2=0.01
        self.y_pred3 = 0
        self.m_err3 = 0
        self.a3 = 0
        self.sigma_u3=83.602 #Varianza de la señal de gyro
        self.sigma_n3=0.01

        #Control de calculo de yaw (si se usa)
        self.way=True #False=metodo 1, correcion de deriva
                #True=metodo 2, estimacion por cuaternios

        #Position data
        self.bit_pos_upd=False
        self.pos_act=[]

        # IMU data
        self.rawIMU=[]
        self.IMUfusion=[]
        self.ang_z_prev=0
        self.lastTime_ang=0

        # PWM data
        self.PWMminP=0.25
        self.PWMmaxP=0.35

        #Variable SS (small space) que sirve para espacios pequeños o de
movimiento no libre
        self.SS=False

        #Márgenes de controladores
        self.angle_margin=5*(math.pi/180) # 5 deg de margen
        self.dmargin=0.1 #10cm de margen
        self.TactAngle=1/17.8 #Max 66Hz de update time
        self.TactPos=1/17.8

    def testWork(self, val):
        self.val = val
        if val:
            print('Clase iniciada correctamente \n')
            print('Todos los modulos se han cargado correctamente')
            print('Valores de kp elegidos: ', self.kp)
            print('Valores de ki elegidos: ', self.ki)
        else:
            print('El digito introducido no verifica los parametros')


    def set_dead_zone(self, add, new_dzone):
        # Dead zone será la zona donde el robot no puede acceder
```

```
        # Se construye una matriz cuadrada que definirá una región no
accesible
        # A partir de 4 esquinas (o 3)
        # [xMin, xMax, yMin, yMax]

        self.add=add
        self.new_dzone=new_dzone
        if add:
            self.dead_zone.append(new_dzone)

    def check_point(self, xc, yc):
        self.xc=xc
        self.yc=yc
        err=0
        for i in range(1,len(self.dead_zone)):
            if self.xc<self.dead_zone[i-1][0] or self.xc>self.dead_zone[i-
1][1] or self.yc<self.dead_zone[i-1][2] or self.yc>self.dead_zone[i-1][3]:
                err=err+1
        if err>0:
            return 1 #El punto no es alcanzable
        else:
            return 0 #No hay error en el punto, está fuera de zona muerta

    def obtener_posicion_angulo(self):
        while not self.bit_pos_upd:
            try:
                self.hedge.dataEvent.wait(1)
                self.hedge.dataEvent.clear()
                #Filtered values
                if (self.hedge.fusionImuUpdated):
                    #data in array [X,Y,Z, QW,QX,QY,QZ, VX,VY,VZ, AX,AY,AZ,
timestamp]
                    self.IMUfusion=self.hedge.imu_fusion()
                    #[AX,AY,AZ, GX,GY,GZ, MX,MY,MZ, timestamp]
                    self.rawIMU=self.hedge.raw_imu()
                    self.bit_pos_upd=True
                    #Stores position+quaternions
            except KeyboardInterrupt:
                self.hedge.stop()  # stop and close serial port
                sys.exit()
        self.bit_pos_upd=False

    def obtener_posicion(self):
        while not self.bit_pos_upd:
            try:
                self.hedge.dataEvent.wait(1)
                self.hedge.dataEvent.clear()
                #Filtered values
                if (self.hedge.positionUpdated):
                    #[AX,AY,AZ, GX,GY,GZ, MX,MY,MZ, timestamp]
                    self.pos_act=self.hedge.position() #hedgeID, X, Y, Z
                    self.bit_pos_upd=True
            except KeyboardInterrupt:
                self.hedge.stop()  # stop and close serial port
                sys.exit()
        self.bit_pos_upd=False

    def obtener_yaw(self, ax, ay, az, gz, timestamp, qw, qx, qy, qz):
        # yaw (z-axis rotation)
        # Falta el filtrado de deriva
        # El codigo siguiente incluye una corrección de deriva
        if not self.way:
```

```
            escalado=131 #suponiendo que ya está en g/s
            dt=timestamp-self.lastTime_ang
            self.lastTime_ang=timestamp
            acc_ang_z=math.atan2(az,math.sqrt(ax**2 + ay**2))
            ang_z=0.98*(self.ang_z_prev+(gz/escalado)*dt)+0.02*acc_ang_z
            self.ang_z_prev=ang_z
            return ang_z
        else:
            #Sin correccion de derivalogSet
            siny_cosp = 2 * (qw * qz + qx * qy)
            cosy_cosp = 1 - 2 * (qy * qy + qz * qz)
            return math.atan2(siny_cosp, cosy_cosp)

    def filtro_angle(self, pos_x, pos_y, ang_compass_in, gyro_z, timestamp):

        #Para que el filtrado sea correcto, esta función debe invocarse
varias veces, esto se debe hacer desde el
        #la funcion principal, para que el bucle de control principal sea la
causa de la detencion

        #Por ahora el papel del compass esta desactivado (alta influencia
campos magnéticos)

        #Preparacion de parametros
        self.pos_x=pos_x
        self.pos_y=pos_y
        self.ang_compass=ang_compass_in
        self.gyro_z=gyro_z
        self.timestamp=timestamp

        #Carga de entradas
        #1. Angulo posicional
        y_real_ang_pos=math.atan2(self.pos_y_prev-self.pos_y,
self.pos_x_prev-self.pos_x)
        self.pos_x_prev=self.pos_x
        self.pos_y_prev=self.pos_y

        """
        #2. Brújula: solo escalado
        offset=0 #se debe observar para alinear ejes
        max_scale=360
        ang_compass=(ang_compass_in-offset)/max_scale
        """
        #3. Giroscopio -> necesaria integracion deg/s -> deg
        #Metodo 2: rectangular (bxh)
        timestampDif=self.timePrev-timestamp
        ang_int= self.ang_int_prev+timestampDif*(self.accAngZprev-
self.gyro_z) #ang int es la entrada al KF3
        self.accAngZprev=self.gyro_z
        self.ang_int_prev=ang_int
        self.timePrev=self.timestamp

        #Variables que deben guardarse en memoria, con un subindice para cada
filtro:
        #Inicializacion
        #y_pred = 0 #Inicialmente la señal procesada se estima en 0 (irá
incrementando)
        #merr = 0 #Inicialmente se estima que el error cuadratico medio MSE
es 0
        #a = # bias (nivel de continua) sobre el que se inicializa el filtro
        #sigma_u = 0 #Varianza de la señal de entrada
        #sigma_n = 0 #Varianza del ruido aplicado a la señal (debe haberlo)
```

```
        # KF1: posicional
        # Fase 1: Prediccion
        self.y_pred1 = self.a1*self.y_pred1
        err = y_real_ang_pos-self.y_pred1 # error cometido en la prediccion
        self.m_err1 = (self.a1**2)*self.m_err1 + self.sigma_u1
        K=self.m_err1/(self.sigma_n1+self.m_err1) # calculo de la cte de
Kalman
        # Fase 2: Corrección de prediccion
        self.y_pred1 =  self.y_pred1+K*err
        y_est_ang_pos = self.y_pred1 # SALIDA DEL FILTRO
        self.m_err1 = (1-K)*self.m_err1 # actualizacion de MSE
        '''
        #KF2: Compass
        self.y_pred2 = self.a2*self.y_pred2
        err = ang_compass-self.y_pred2 # error cometido en la prediccion
        self.m_err2 = (self.a2**2)*self.m_err2 + self.sigma_u2
        K=self.m_err2/(self.sigma_n2+self.m_err2) # calculo de la cte de
Kalman
        # Fase 2: Corrección de prediccion
        self.y_pred2 =  self.y_pred2+K*err
        ang_compass_est = self.y_pred2 # SALIDA DEL FILTRO
        self.m_err2 = (1-K)*self.m_err2 # actualizacion de MSE
        '''
        #KF3: Gyro
        self.y_pred3 = self.a3*self.y_pred3
        err = ang_int-self.y_pred3 # error cometido en la prediccion
        self.m_err3 = (self.a3**2)*self.m_err3 + self.sigma_u3
        K=self.m_err3/(self.sigma_n3+self.m_err3) # calculo de la cte de
Kalman
        # Fase 2: Corrección de prediccion
        self.y_pred3 =  self.y_pred3+K*err
        ang_int_est = self.y_pred3 # SALIDA DEL FILTRO
        self.m_err3 = (1-K)*self.m_err3 # actualizacion de MSE

        #Fusión de estimaciones
        #Antes de fusionar se debe haber calculado la covarianza de cada
filtro
        #La covarianza se deberia de calcular con la medida real: ej: si se
gira 45°
        #la covarianza se calcula respecto a una variable cte de ese valor

        covPos=0.1 #Covarianzas que deben estimarse en base a experimentos
        covComp=0.1
        covGyro=0.1
        '''
        num=y_est_ang_pos/covPos+ang_compass_est/covComp+ang_int_est/covGyro
        den=1/covPos+1/covComp+1/covGyro
        return num/den #Estimación ponderada de los 3 filtrados según
covarianzas
        '''

        num=y_est_ang_pos/covPos+ang_int_est/covGyro
        den=1/covPos+1/covGyro
        return [num/den, y_est_ang_pos, ang_int_est] #Estimación ponderada de
los 3 filtrados según covarianzas

    def calc_cov(self, in1, in2):
        self.in1=in1
        self.in2=in2
        N=len(in1)
        med1=0
```

```
        med2=0
        cov=0
        #calculo de media
        for i in range(0,N-1):
            med1=med1+self.in1(i)
            med2=med2+self.in2(i)
        med1=med1/N
        med2=med2/N
        for i in range(0,N-1):
            cov2=cov2+(self.in1(i)-med1)*(self.in2(i)-med2)
        return cov2/N


    def turn_to(self, angle):
        # Se debe girar el robot hasta la posición dada

        self.angle = angle # Debe estar expresado en (-pi, ++pi)
        PI_ang = classPI(kp=self.kp[0], ki=self.ki[0], lim=self.PWMmaxP)

        #Cuando el robot está a 90° está alineado con el eje Y positivo, se
gira desde Y

        # Yaw respecto al SR del robot
        #Cte ángulo margen (configurable)

        err_PI=self.angle_margin*2 #Para entrar inicialmente al bucle
        angulo_leido=0
        t_prev=0

        if self.debug:
            print('Comienzo del reposicionamiento del robot')
        #Bucle de calculo del PI

        while (self.angle_margin < abs(err_PI)):
            if ((time.time()-t_prev)>=self.TactAngle):
                t_prev=time.time()
                if self.debugTime:
                    inicio=time.time()

                angulo_leido=self.hedge.return_angle() #Return angle ya
devuelve el ángulo del eje Y del robot (angulo leido -90)
                #print('Posicion actualizada')

                if self.debugTime:
                    treadAng=time.time()

                #Calculo controlador
                #El err introducido al PI siempre será positivo, el ángulo se
maneja segun convenga
                err_PI=0
                giro=1
                dif=self.angle-angulo_leido

                #Control de signo segun C1 o C2 (C1 son los cuadrantes
superiores 1 y 2, C2 los inferiores)
                if (self.angle>=0 and self.angle<=math.pi): #C1
                    if (angulo_leido>math.pi and angulo_leido<2*math.pi): #C2
                        if (dif<=-math.pi or self.angle==0):
                            giro=-1 #giro antihorario
                            err_PI=2*math.pi-angulo_leido+self.angle
                        else:
                            giro=1 #giro horario
```

```
                        err_PI=abs(dif)
                else: #si no, C1
                    if (dif<0 or self.angle==0):
                        giro=1 #giro horario
                        err_PI=abs(dif)
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif
            else: #Angulo referencia en C2
                if (angulo_leido>math.pi and angulo_leido<2*math.pi): #C2
                    if dif<0:
                        giro=1 #giro horario
                        err_PI=abs(dif)
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif
                else: #si no, C1
                    if dif>=math.pi:
                        giro=1 #giro horario
                        err_PI=abs(2*math.pi-self.angle+angulo_leido)
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif

            u=PI_ang.PI_SW(err_PI)

            if self.debugTime:
                tcalcPI=time.time()

            if self.debug:
                print('Controlador: ', u, 'Angulo objetivo: ',
self.angle*(180/math.pi), 'angulo leido: ', angulo_leido*(180/math.pi))

            #Asignacion a motores
            if not self.SS:
                self.mIzq._write_value(value=u*giro)
                self.mDer._write_value(value=-u*giro)
                if self.logSet == 1:
                    self.data.append([u, u, self.angle, angulo_leido,
self.angle_margin])
            else: #No usado  - PWM en tren de pulsos
                T=0.1
                self.mIzq._write_value(value=u*giro)
                self.mDer._write_value(value=-u*giro)
                if self.logSet == 1:
                    self.data.append([u, u, self.angle, angulo_leido,
self.angle_margin])
                time.sleep(T/2)
                self.mIzq._release()
                self.mDer._release()
                if self.logSet == 1:
                    self.data.append([0, 0, self.angle, angulo_leido,
self.angle_margin])
                time.sleep(T/2)

            if self.debugTime:
                final=time.time()
                print('t de ejecucion de iteracion completa= ', (final-
inicio)*10**6)
                print('t de lectura de angulo: ', (treadAng-
inicio)*10**6)
```

```
                    print('t de calculo de controlador: ', (tcalcPI-
treadAng)*10**6)

        #Detencion al alcanzar angulo
        self.mIzq._release()
        self.mDer._release()

        #Si el log está activado se guardan los datos
        if self.logSet == 1:
            csvfile =
open("/home/jetbot/Desktop/JetBot_Balizas/jetbotMotion/logAng.csv",'w',
newline='')
            with csvfile:
                writer = csv.writer(csvfile)
                writer.writerows(self.data)
            print('Se ha finalizado la captura de datos')


    def go_to(self, x, y):
        self.x = x
        self.y = y

        # Posicion inicial
        self.obtener_posicion()
        if self.debug:
            print('Posicion RAW ',self.pos_act)

        # pos_act: [hedgeID, X, Y....]
        posicion_leida=[self.pos_act[1], self.pos_act[2]]
        dify=self.y-posicion_leida[1]
        difx=self.x-posicion_leida[0]
        initial_rot=math.atan2(dify,difx)


        if initial_rot <0:
            initial_rot=initial_rot + 2*math.pi #Siempre se trabaja con
ángulo positivo (+360°)

        self.turn_to(initial_rot) #giro inicial
        # initial_rot guarda el ángulo de orientación respecto al SR balizas
        # Tras el primer giro se comienza el comando comjunto de
dirección+angulo

        # Distancia inicial
        #angle_margin=3*(math.pi/180) # 3 deg de margen
        distance=math.sqrt(dify**2+difx**2)
        # La parte de corrección de ángulo se ejecutará siempre

        # Instanciación del doble PI
        PIang=classPI(kp=self.kp[0], ki=self.ki[0], lim=self.PWMmaxP)
        PIpos=classPI(kp=self.kp[1], ki=self.ki[1], lim=self.PWMmaxP)
        sat=False
        t_prev=0

        # Bucle de cálculo de PIs
        while (distance > self.dmargin):
            if ((time.time()-t_prev)>=self.TactPos):
                t_prev=time.time()

            # Calculo de posicion y angulo iterativos
            self.obtener_posicion()
```

```
            self.hedge.dataEvent.wait(1)
            self.hedge.dataEvent.clear()
            while not self.hedge.positionUpdated:
                angulo_leido=0
            angulo_leido=self.hedge.return_angle() #Return angle ya devuelve
el ángulo del eje Y del robot (angulo leido -90)

            posicion_leida=[self.pos_act[1], self.pos_act[2]]
            dify=self.y-posicion_leida[1]
            difx=self.x-posicion_leida[0]
            distance=math.sqrt(dify**2+difx**2)
            rot=math.atan2(dify,difx)
            if rot <0:
                rot=rot + 2*math.pi #Siempre se trabaja con ángulo positivo
(+360°)

            # Doble PI (posicion+angulo)
            err_PI=0
            giro=1 #1: horario, -1:antihorario
            dif=rot-angulo_leido

            #Control de signo segun C1C2 o C3C4 (C1C2 son los cuadrantes
superiores 1 y 2, C3C4 los inferiores)
            if (rot>=0 and rot<=math.pi): #C1C2
                if (angulo_leido>math.pi and angulo_leido<2*math.pi): #C3C4
                    if (dif<=-math.pi or rot==0):
                        giro=-1 #giro antihorario
                        err_PI=2*math.pi-angulo_leido+rot
                    else:
                        giro=1 #giro horario
                        err_PI=abs(dif)
                else: #si no, C1C2
                    if (dif<0 or rot==0):
                        giro=1 #giro horario
                        err_PI=abs(dif)
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif
            else: #Angulo referencia en C3C4
                if (angulo_leido>math.pi and angulo_leido<2*math.pi): #C3C4
                    if dif<0:
                        giro=1 #giro horario
                        err_PI=abs(dif)
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif
                else: #si no, C1C2
                    if dif>=math.pi:
                        giro=1 #giro horario
                        err_PI=2*math.pi-rot+angulo_leido
                    else:
                        giro=-1 #giro antihorario
                        err_PI=dif

            err_pos=distance-self.dmargin
            uAng=PIang.PI_SW_ext(err=err_PI, sat_act=sat) #PIs con control de
saturacion externo
            uPos=PIpos.PI_SW_ext(err=err_pos, sat_act=sat)

            uIzq = uPos + uAng*giro
            uDer = uPos - uAng*giro
```

```
            # Anti windup + control de saturacion
            if uIzq>self.PWMmaxP:
                uIzq=self.PWMmaxP
                sat=True
            elif uIzq<-self.PWMmaxP:
                uIzq=-self.PWMmaxP
                sat=True
            else:
                if sat:
                    sat=False

            if uDer>=self.PWMmaxP:
                uDer=self.PWMmaxP
                sat=True
            elif uDer<=-self.PWMmaxP:
                uDer=-self.PWMmaxP
                sat=True
            else:
                if sat:
                    sat=False

            # Asignacion a motores

            if not self.SS:
                self.mIzq._write_value(value=uIzq)    # Cuando u<1 el robot
gira a la izq (giros con incAng>0)
                self.mDer._write_value(value=uDer)     # Cuando u>1 el robot
gira a la derecha (giros con incAng<0)
                if self.logSet == 2:
                    self.data.append([uIzq, uDer, rot, angulo_leido,
distance, self.dmargin, posicion_leida[0], posicion_leida[1]])
            else: #En tren de pulsos, no usado actualmente
                T=0.1
                self.mIzq._write_value(value=uIzq)    # Cuando u<1 el robot
gira a la izq (giros con incAng>0)
                self.mDer._write_value(value=uDer)     # Cuando u>1 el robot
gira a la derecha (giros con incAng<0)
                if self.logSet == 2:
                    self.data.append([uIzq, uDer, rot, angulo_leido,
distance, self.dmargin, posicion_leida[0], posicion_leida[1]])
                time.sleep(2*T/3)
                self.mIzq._release()
                self.mDer._release()
                if self.logSet == 2:
                    self.data.append([0, 0, rot, angulo_leido, distance,
self.dmargin, posicion_leida[0],posicion_leida[1]])
                time.sleep(T/3)
            if self.debug:
                print('Motor iz [u]: ', uIzq, 'Motor der [u]: ', uDer)
            #time.sleep(T) # Ajuste de t de muestreo

        #Detencion de motores
        self.mIzq._release()
        self.mDer._release()

        #Si el log está activado se guardan los datos
        if self.logSet == 2:
            csvfile =
open("/home/jetbot/Desktop/JetBot_Balizas/jetbotMotion/logAngPos.csv",'w',
newline='')
            with csvfile:
                writer = csv.writer(csvfile)
```

```
                writer.writerows(self.data)
            print('Se ha finalizado la captura de datos')


    def take_sample(self):
        #Se tomará una foto con la cámara en la ubicación alcanzada
        from jupyter_clickable_image_widget import ClickableImageWidget
        DATASET_DIR = 'dataset_xy'

        # we have this "try/except" statement because these next functions
can throw an error if the directories exist already
        try:
            os.makedirs(DATASET_DIR)
        except FileExistsError:
            print('Directories not created because they already exist')

        camera = Camera()

        # create image preview
        camera_widget = ClickableImageWidget(width=camera.width,
height=camera.height)
        snapshot_widget = ipywidgets.Image(width=camera.width,
height=camera.height)
        traitlets.dlink((camera, 'value'), (camera_widget, 'value'),
transform=bgr8_to_jpeg)

        # create widgets
        count_widget = ipywidgets.IntText(description='count')
        # manually update counts at initialization
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR,
'*.jpg')))

        def save_snapshot(_, content, msg):
            if content['event'] == 'click':
                data = content['eventData']
                x = data['offsetX']
                y = data['offsetY']

                # save to disk
                #dataset.save_entry(category_widget.value, camera.value, x,
y)
                uuid = 'xy_%03d_%03d_%s' % (x, y, uuid1())
                image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
                with open(image_path, 'wb') as f:
                    f.write(camera_widget.value)

                # display saved snapshot
                snapshot = camera.value.copy()
                snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
                snapshot_widget.value = bgr8_to_jpeg(snapshot)
                count_widget.value = len(glob.glob(os.path.join(DATASET_DIR,
'*.jpg')))

        camera_widget.on_msg(save_snapshot)

        data_collection_widget = ipywidgets.VBox([
            ipywidgets.HBox([camera_widget, snapshot_widget]),
            count_widget
        ])

        display(data_collection_widget)
```

```
    def stop(self):
        self.hedge.stop()   # stop and close serial port
        sys.exit()
```

ANEXO V: CÓDIGO "CLASSPI.PY".

```
#Codigo que alberga un controlador PI implementado mediante backward Euler
#Incluye control de saturacion y anti-windup

#Autor: Francisco Casas Reyes

# Implementacion PI: u=p+i -> Backward Euler
# p(k)=Kp*(c*ref(k)-med(k))
# i(k)=i(k-1)+(kp/Ti)*T*err(k-1)
# donde T es el periodo de muestreo

class classPI():
    def __init__(self, kp, ki, lim):
        self.kp=kp
        self.ki=ki
        self.err_prev=0
        self.i_prev=0
        self.wnd_i=False
        self.lim=lim

    def PI_SW(self,err):
        self.err=err
        p=self.kp*self.err

        #Proteccion anti-windup + calculo termino i
        if not self.wnd_i:
            i=self.i_prev+self.ki*self.err_prev
        else:
            i=self.i_prev
        self.err_prev=self.err
        self.i_prev=i # Actualizacion de i_prev

        #Salida del controlador
        u=p+i

        if u>self.lim: # Saturacion de la salida + antiwindup
            u=self.lim
            self.wnd_i=True #Pausa del cálculo de i
        elif u<-self.lim:
            u=-self.lim
            self.wnd_i=True
        else:
            if wnd_i:
                wnd_i = False #Puede seguir calculandose i

        return u

    def PI_SW_ext(self,err,sat_act):
        #PI con control de saturacion externo

        self.err=err
        self.wnd_i=sat_act
        p=self.kp*self.err

        #Proteccion anti-windup + calculo termino i
        if not self.wnd_i:
            i=self.i_prev+self.ki*self.err_prev
```

```
        else:
            i=self.i_prev
        self.err_ext=self.err
        self.i_prev=i # Actualizacion de i_prev

        #Salida del controlador
        u=p+i

        return u
```

ANEXO VI: CÓDIGO "MARVELMIND_MOD.PY" (SOLO MÉTODO AÑADIDO).

```
    def return_angle(self):
        #print('Angulo RAW: ', self.position()[4])
        #print('Angulo REAL leido: ',((self.position()[4]-900)/10))
        angle=(self.position()[4]-900)/10
        if angle < 0:
            angle = angle+360

        return  angle*(math.pi/180)  #Angle  located  at  position  5  (with  Y
correction) [rad]
```

ANEXO VII: LIBRERÍA MODIFICADA DE MOTORES ("LIBMOTORS_ADAFRUITBASE_V0.PY").

```
import atexit
from Adafruit_MotorHAT import Adafruit_MotorHAT


class Motor():
    def __init__(self, channel, *args, **kwargs):
        self.i2c_bus=1
        #Bus i2c: id=1
        self._driver = Adafruit_MotorHAT(i2c_bus=self.i2c_bus)
        self._motor = self._driver.getMotor(channel)
        #Canal motor izquierdo: 1
        #Canal motor derecho:2
        if(channel == 1):
            self._ina = 1
            self._inb = 0
        else:
            self._ina = 2
            self._inb = 3
        atexit.register(self._release)

    def _write_value(self, value):
        self.value=value #Valor entre 0 y 1, se reescala a [0, 4080]
        self.vEsc=abs(int(self.value*4080))
        if self.value < 0:
            self._motor.run(Adafruit_MotorHAT.FORWARD)
            self._driver._pwm.setPWM(self._ina,0,0)
            self._driver._pwm.setPWM(self._inb,0,self.vEsc)
        else:
            self._motor.run(Adafruit_MotorHAT.BACKWARD)
            self._driver._pwm.setPWM(self._ina,0,self.vEsc)
            self._driver._pwm.setPWM(self._inb,0,0)

    def _release(self):
        """Stops motor by releasing control"""
        self._motor.run(Adafruit_MotorHAT.RELEASE)
        self._driver._pwm.setPWM(self._ina,0,0)

        self._driver._pwm.setPWM(self._inb,0,0)
```

ANEXO VIII: CÓDIGO DE PRUEBA DE REORIENTACIONES.

```
#Script para prueba de giro

#Prueba de giro a 4 angulos

import math
import time
from robotClass_v10 import DifferentialRobot

move = DifferentialRobot(kp=[0.5,0.7], ki=[0.01, 0.001], dead_zone=[0])
coefRadDeg = math.pi/180

def main():
    move.turn_to(45*coefRadDeg)
    print('Angulo 1 alcanzado')
    time.sleep(3)
    move.turn_to(180*coefRadDeg)
    print('Angulo 2 alcanzado')
    time.sleep(3)
    move.turn_to(315*coefRadDeg)
    print('Angulo 3 alcanzado')
    time.sleep(3)
    move.turn_to(135*coefRadDeg)
    print('Angulo 4 alcanzado')
    move.stop()


main()
```

ANEXO IX: CÓDIGO DE PRUEBA DE RUTA.

```
#Script para prueba de giro+traslación del robot

#Prueba triangular: 3 puntos

from robotClass_v10 import DifferentialRobot

move = DifferentialRobot(kp=[0.5,0.7], ki=[0.01, 0.001], dead_zone=[0])

def main():
    #Prueba triangular:robot colocado en x=0.4, y=0
    move.go_to(x=0.4, y=0.5)
    print('Punto 1 alcanzado')
    move.go_to(x=0.6, y=0.2)
    print('Punto 2 alcanzado')
    move.go_to(x=0.4, y=0)
    print('Punto 3 alcanzado - fin de ruta')
    move.stop()

main()
```

ANEXO X: CÓDIGO MATLAB DE DIBUJADO DE TRAYECTORIA CUADRADA.

```
%Script para el dibujado de la trayectoria cuadrada, con objetivo de
%comparar con la trayectoria real seguida.

clear all
close all

%load('datosPosicionMovilCuadrado30x30_tabla.mat');
%load('datosPosicionMovilCuadrado30x30_Obs2.mat');
%load('datosPosicionMovilCuadrado30x30_escenario2.mat');
load('datosPosicionMovilCuadrado30x30_Obs1_3.mat');
load('rutaCuadradoB1B2bloqueadas.mat');
load('rutaCuadradoB1bloqueada.mat');
load('rutaSinObstruccion.mat');

Xdata=table2array(lastPathID5(:,2));
Ydata=table2array(lastPathID5(:,3));
XdataP=Xdata;
YdataP=Ydata;

plot(Xdata, Ydata)
hold on
grid on
title('Trayectoria seguida por la baliza móvil')
%Superposicion de trayectoria sin obstrucciones
Xdata=table2array(rutaSinObstruccion(:,2));
Ydata=table2array(rutaSinObstruccion(:,3));
plot(Xdata, Ydata, 'k-')
%axis([min(Xdata)-0.1   max(XdataP)+0.1   min(YdataP)-0.1
%max(Ydata)+0.1]); % Para B2 obstruida
axis([min(XdataP)-0.1   max(XdataP)+0.1   min(YdataP)-0.1
max(YdataP)+0.1]);
% Para B1 y B3 obstruidas
legend('Trayectoria con balizas 1 y 3 obstruidas', 'Trayectoria sin
obstrucciones')
%legend('Trayectoria con baliza 2 obstruida', 'Trayectoria sin
obstrucciones')

%% Parte 2: con la baliza 1 reemplazada y en modo paired hedge


figure()

Xdata=table2array(rutaCuadradoB1bloqueada(:,2));
Ydata=table2array(rutaCuadradoB1bloqueada(:,3));
XdataP=Xdata;
YdataP=Ydata;
plot(Xdata, Ydata)
hold on
grid on
Xdata=table2array(rutaSinObstruccion(:,2));
Ydata=table2array(rutaSinObstruccion(:,3));
axis([min(XdataP)-0.1 max(Xdata)+0.1 min(Ydata)-0.1 max(YdataP)+0.1]);
plot(Xdata, Ydata, 'k-')
legend('Trayectoria con B1 bloqueada', 'Trayectoria original sin bloqueo')
```

```
title('Trayectoria seguida por la baliza móvil 6 - B1 bloqueada')

%% Parte 3: con balizas 1 y 2 obstruidas


figure()

Xdata=table2array(rutaCuadradoB1B2bloqueadas(:,2));
Ydata=table2array(rutaCuadradoB1B2bloqueadas(:,3));
XdataP=Xdata;
YdataP=Ydata;
plot(Xdata, Ydata)
hold on
grid on
Xdata=table2array(rutaSinObstruccion(:,2));
Ydata=table2array(rutaSinObstruccion(:,3));
axis([min(XdataP)-0.1 max(XdataP)+0.1 min(YdataP)-0.1 max(Ydata)+0.1]);
plot(Xdata, Ydata, 'k-')
legend('Trayectoria con B1B2 bloqueadas', 'Trayectoria original sin bloqueo')
title('Trayectoria seguida por la baliza móvil 6 - B1B2 bloqueadas')
```

ANEXO XI: CÓDIGO MATLAB DE DIBUJADO DE ENSAYOS DE ÁNGULO Y POSICIÓN.

```
%% Script para graficar los resultados de una ruta seguida
%Incluye dibujado de la ruta original frente a la trazada
%Incluye tanto ángulo como posición

clear all
close all

load('logAngPosruta3puntos.mat')
load('logAngPos3puntosprueba2ki0.mat')
load('logAngPospruebaFinalki03.mat')

cteRadDeg=180/pi;

P0=[0.4, 0];
P1=[0.4, 0.5];
P2=[0.6,0.2];
P3=[0.4, 0];
xPuntos=[P0(1), P1(1), P2(1), P3(1)];
yPuntos=[P0(2), P1(2), P2(2), P3(2)];

%Datos capturados
uI=table2array(logAngPospruebaFinalki03(:,1));
uD=table2array(logAngPospruebaFinalki03(:,2));
rot=table2array(logAngPospruebaFinalki03(:,3))*cteRadDeg;
ang_leido=table2array(logAngPospruebaFinalki03(:, 4))*cteRadDeg;
distance=table2array(logAngPospruebaFinalki03(:, 5));
dmargin=table2array(logAngPospruebaFinalki03(:, 6));
xPos=table2array(logAngPospruebaFinalki03(:, 7));
yPos=table2array(logAngPospruebaFinalki03(:, 8));
m=1:length(dmargin);


subplot(2,1,1)
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
subplot(2,1,2)
plot(m, rot, 'm')
hold on
plot(m, ang_leido, 'k')
ylabel('Ángulos [deg]')
xlabel('Número de muestra')
legend('Angulo objetivo [deg]', 'Angulo actual [deg]')


figure()

subplot(2,1,1)
```

```
hold on
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
subplot(2,1,2)
plot(m, distance, 'k')
hold on
plot(m, dmargin)
ylabel('Distancia [m]')
xlabel('Número de muestra')
legend('Distancia actual [m]', 'Margen de distancia [m]')

figure()
%Dibujado de la ruta
hold on
title ('Funcionamiento controlador ruta 3 Puntos')
ylabel('Y [m]')
xlabel('X [m]')
plot(xPos, yPos, 'b-')
plot(xPuntos, yPuntos, 'ko-')
axis([min(xPuntos)-0.1 max(xPuntos)+0.1 min(yPuntos)-0.1 max(yPuntos)+0.1 ])

pause()
close all

%% Prueba con datos B

%Datos capturados
uI=table2array(logAngPos3puntosprueba2ki0(:,1));
uD=table2array(logAngPos3puntosprueba2ki0(:,2));
rot=table2array(logAngPos3puntosprueba2ki0(:,3))*cteRadDeg;
ang_leido=table2array(logAngPos3puntosprueba2ki0(:, 4))*cteRadDeg;
distance=table2array(logAngPos3puntosprueba2ki0(:, 5));
dmargin=table2array(logAngPos3puntosprueba2ki0(:, 6));
xPos=table2array(logAngPos3puntosprueba2ki0(:, 7));
yPos=table2array(logAngPos3puntosprueba2ki0(:, 8));
m=1:length(dmargin);

subplot(2,1,1)
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
subplot(2,1,2)
plot(m, rot, 'm')
hold on
plot(m, ang_leido, 'k')
ylabel('Ángulos [deg]')
xlabel('Número de muestra')
legend('Angulo objetivo [deg]', 'Angulo actual [deg]')
```

```
figure()

subplot(2,1,1)
hold on
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
subplot(2,1,2)
plot(m, distance, 'k')
hold on
plot(m, dmargin)
ylabel('Distancia [m]')
xlabel('Número de muestra')
legend('Distancia actual [m]', 'Margen de distancia [m]')

figure()
%Dibujado de la ruta
hold on
title ('Funcionamiento controlador ruta 3 Puntos')
ylabel('Y [m]')
xlabel('X [m]')
plot(xPos, yPos, 'b-')
plot(xPuntos, yPuntos, 'ko-')
axis([min(xPuntos)-0.1 max(xPuntos)+0.1 min(yPuntos)-0.1 max(yPuntos)+0.1 ])

pause()
close all

%% Prueba con datos C

%Datos capturados
uI=table2array(logAngPosruta3puntos(:,1));
uD=table2array(logAngPosruta3puntos(:,2));
rot=table2array(logAngPosruta3puntos(:,3))*cteRadDeg;
ang_leido=table2array(logAngPosruta3puntos(:, 4))*cteRadDeg;
distance=table2array(logAngPosruta3puntos(:, 5));
dmargin=table2array(logAngPosruta3puntos(:, 6));
xPos=table2array(logAngPosruta3puntos(:, 7));
yPos=table2array(logAngPosruta3puntos(:, 8));
m=1:length(dmargin);

subplot(2,1,1)
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
subplot(2,1,2)
plot(m, rot, 'm')
hold on
plot(m, ang_leido, 'k')
```

```
ylabel('Ángulos [deg]')
xlabel('Número de muestra')
legend('Angulo objetivo [deg]', 'Angulo actual [deg]')


figure()

subplot(2,1,1)
hold on
plot(m, uI*100, 'b-')
hold on
plot(m, uD*100, 'r-')
title ('Funcionamiento controlador ruta 3 puntos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
legend('Señal PWM del controlador izquierdo [%]', 'Señal PWM del controlador
derecho [%]')
axis([m(1)-1 m(end)+1 min(uI*100)-1 max(uI*100)+1 ])
subplot(2,1,2)
plot(m, distance, 'k')
hold on
plot(m, dmargin)
ylabel('Distancia [m]')
xlabel('Número de muestra')
legend('Distancia actual [m]', 'Margen de distancia [m]')

figure()
%Dibujado de la ruta
hold on
title ('Funcionamiento controlador ruta 3 Puntos')
ylabel('Y [m]')
xlabel('X [m]')
plot(xPos, yPos, 'b-')
plot(xPuntos, yPuntos, 'ko-')
axis([min(xPuntos)-0.1 max(xPuntos)+0.1 min(yPuntos)-0.1 max(yPuntos)+0.1 ])
```

ANEXO XII: CÓDIGO MATLAB DE DIBUJADO DE VARIAS REORIENTACIONES.

```
%% Script para graficar los resultados de un reposicionamiento en angulo

clear all
close all

load('logAng180a90.mat')
load('logAng225a90.mat')
load('logAng45a90.mat')
load('logAng90a45.mat')
load('logAngpruebaFinal4angulos.mat')

cteRadDeg=180/pi;

u=table2array(logAng180a90(:,1));
angle=table2array(logAng180a90(:,3))*cteRadDeg;
ang_leido=table2array(logAng180a90(:, 4))*cteRadDeg;
margen=table2array(logAng180a90(:, 5))*cteRadDeg;
m=1:length(margen);

hold on
yyaxis left
title ('Funcionamiento controlador ángulo 180°->90°')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
plot(m, u*100)
axis([m(1)-1 m(end)+1 min(u*100)-1 max(u*100)+1 ])
yyaxis right
ylabel('Ángulos [deg]')
plot(m, angle)
plot(m, ang_leido)
plot(m, margen)
legend('Señal PWM del controlador [%]', 'Angulo objetivo [deg]', 'Angulo
actual [deg]', 'Margen [deg]')

%% Plot 2
figure()

u=table2array(logAng225a90(:,1));
angle=table2array(logAng225a90(:,3))*cteRadDeg;
ang_leido=table2array(logAng225a90(:, 4))*cteRadDeg;
margen=table2array(logAng225a90(:, 5))*cteRadDeg;
m=1:length(margen);

hold on
yyaxis left
title ('Funcionamiento controlador ángulo 225°->90°')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
plot(m, u*100)
axis([m(1)-1 m(end)+1 min(u*100)-1 max(u*100)+1 ])
yyaxis right
ylabel('Ángulos [deg]')
```

```
plot(m, angle)
plot(m, ang_leido)
plot(m, margen)
legend('Señal PWM del controlador [%]', 'Angulo objetivo [deg]', 'Angulo
actual [deg]', 'Margen [deg]')

%% Plot 3
figure()

u=table2array(logAng45a90(:,1));
angle=table2array(logAng45a90(:,3))*cteRadDeg;
ang_leido=table2array(logAng45a90(:, 4))*cteRadDeg;
margen=table2array(logAng45a90(:, 5))*cteRadDeg;
m=1:length(margen);

hold on
yyaxis left
title ('Funcionamiento controlador ángulo 45°->90°')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
plot(m, u*100)
axis([m(1)-1 m(end)+1 min(u*100)-1 max(u*100)+1 ])
yyaxis right
ylabel('Ángulos [deg]')
plot(m, angle)
plot(m, ang_leido)
plot(m, margen)
legend('Señal PWM del controlador [%]', 'Angulo objetivo [deg]', 'Angulo
actual [deg]', 'Margen [deg]')

%% Plot 4
figure()

u=table2array(logAng90a45(:,1));
angle=table2array(logAng90a45(:,3))*cteRadDeg;
ang_leido=table2array(logAng90a45(:, 4))*cteRadDeg;
margen=table2array(logAng90a45(:, 5))*cteRadDeg;
m=1:length(margen);

hold on
yyaxis left
title ('Funcionamiento controlador ángulo 90°->45°')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
plot(m, u*100)
axis([m(1)-1 m(end)+1 min(u*100)-1 max(u*100)+1 ])
yyaxis right
ylabel('Ángulos [deg]')
plot(m, angle)
plot(m, ang_leido)
plot(m, margen)
legend('Señal PWM del controlador [%]', 'Angulo objetivo [deg]', 'Angulo
actual [deg]', 'Margen [deg]')

%% Plot 5: Prueba final con 4 angulos
%Mantener nombre de archivo

figure()

u=table2array(logAngpruebaFinal4angulos(:,1));
angle=table2array(logAngpruebaFinal4angulos(:,3))*cteRadDeg;
ang_leido=table2array(logAngpruebaFinal4angulos(:, 4))*cteRadDeg;
```

```
margen=table2array(logAngpruebaFinal4angulos(:, 5))*cteRadDeg;
m=1:length(margen);

hold on
yyaxis left
title ('Funcionamiento controlador 4 angulos')
ylabel('Señal controlador [PWM]')
xlabel('Número de muestra')
plot(m, u*100)
axis([m(1)-1 m(end)+1 min(u*100)-1 max(u*100)+1 ])
yyaxis right
ylabel('Ángulos [deg]')
plot(m, angle)
plot(m, ang_leido, 'k-')
plot(m, margen, 'r')
legend('Señal PWM del controlador [%]', 'Angulo objetivo [deg]', 'Angulo
actual [deg]', 'Margen [deg]
```