TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Desarrollo de un sistema de localización en interiores basado en balizas UWB con aplicación en robótica móvil"

Curso: 2021/2022

Modalidad TFG: Trabajo técnico o experimental

Alumno/a:

Guillermo Martínez Martínez

Director/es:

José Carlos Moreno Úbeda

Agradecimientos

Al director José Carlos Moreno Úbeda, por su apoyo y ayuda durante el desarrollo de todo el proyecto.

A Francisco Mañas Álvarez por su orientación en los sistemas robóticos controlados por ROS.

Y por último a Gines Martínez y Salvador Belmonte por darme la posibilidad de realizar pruebas en un invernadero real.

Este trabajo fin de grado ha sido financiado con el Proyecto Agricultural Collaborative Robot Inside IoT (AGRICOBIOT) UAL2020-TEP-A1991 del Programa Operativo FEDER Andalucía 2014-2020

Dedicatoria

A mis padres y mi hermana por estar siempre a mi lado apoyando y respetando mis decisiones y a todos esos amigos con los que he crecido y que me han acompañado a lo largo de gran parte de mi vida.

Índice de figuras

Índice de tablas

Tabla de abreviaturas

Abreviatura	**Significado**
2D	2 Dimensiones
3D	3 Dimensiones
6D-EKF	*6 Dimensions- Extended Kalman Filter*
AoA	*Angle of Arrival*
API	*Application Programing Interface*
BLE	*Bluetooth Low Energy*
GNSS	*Global Navigation Satellite System*
GPL	*General Public License*
GPS	*Global Positioning System*
ICP	*Iterative Closet Point*
IMU	*Inertial Measurement Unit*
INS	*Inertial Navigation System*
ISM	*Industrial Scientific and Medical*
MEMS	*Microelectromechanical System*
RA	Realidad Aumentada
RBPF	*Rao-Blackwellized Particles Filter*
RFID	*Radio Frequency Identification*
ROS	*Robot Operating System*
RSSI	*Received Signal Strenth Indicator*
RTLS	*Real Time Location System*
SLAM	*Simultaneous Locationing and Mapping*
TFG	Trabajo de Fin de Grado
TFE	Trabajo de Fin de Estudios
TDoA	*Time Difference of Arrival*
ToA	*Time of Arrival*
ToF	*Time of Flight*
UWB	*Ultra Wide Band*
VCS	*Version Control System*
WLAN	*Wireless Local Area Network*

Índice general

Resumen

La robótica se ha convertido en los últimos años en la solución más empleada a la hora de resolver los diferentes problemas que surgen tanto en los procesos industriales como en otros ámbitos. Uno de los campos de estudio dentro de la robótica que está en expansión es el de la robótica móvil, en gran parte por varias marcas de automóviles que están invirtiendo grandes sumas de dinero en estos estudios.

En la robótica móvil uno de los problemas recurrentes es el del posicionamiento de los robots, sobre todo cuando se necesita mucha precisión para evitar ciertos obstáculos y más aún si el lugar por donde el robot se debe mover es un recinto cerrado.

En aras de plantear una solución a este problema, en este proyecto se trata de buscar una solución para el posicionamiento en interiores de robots móviles investigando primero los tipos de sistemas que existen y están disponibles en el mercado para posteriormente seleccionar el que más convenga para realizar pruebas.

Además de esto, se implementará un algoritmo de seguimiento de trayectorias en un robot para probar el funcionamiento del sistema seleccionado en conjunto con un robot móvil.

Abstract

Currently, robotics has become the most widely used solution to solve the different problems that appear in industrial processes and other areas. One of the fields of study within robotics that is expanding is that of mobile robotics, due to several automobile brands that are investing large sums of money in these studies.

In mobile robotics one of the recurring problems is the positioning of robots, especially when high precision is needed to avoid certain obstacles and even more so if the place where the robot must move is an indoor area.

In order to propose a solution to this problem, this project aims to find a solution for indoor positioning of mobile robots by first investigating the types of systems that exist and are available on the market and then select the most suitable for testing.

In addition to this, a trajectory tracking algorithm will be implemented in a robot to test the performance of the selected system in conjunction with a mobile robot.

1. Introducción

1.1. Motivación del trabajo de fin de estudios

La característica principal de la robótica y la automatización industrial desde los principios de su historia ha sido la del cambio brusco de los métodos de trabajo populares, los cuales siempre han estado estrechamente ligados con la economía mundial por razones obvias. El uso de robots en la industria combinados con sistemas de diseño asistido por ordenador (CAD) y manufactura asistida por ordenador (CAM) son la tónica dominante en la automatización de procesos, todas estas tecnologías están llevando a este campo de la ciencia hacia otra nueva transición de alcances inimaginables [1].

Uno de los grandes dilemas de esta nueva era tecnológica nace del drástico cambio que se ha dado en la última década a la hora de adquirir productos a través de internet en lugar de comprarlos en una tienda física, esto se debe a la globalización la cual permite no solo adquirir productos de la zona en la que se vive sino de todo el mundo y también se debe en parte a la situación de pandemia en la que se encuentra la población desde 2020, la cual ha causado que se emplee más aún medios informáticos para la compra de todo tipo de artículos. Con toda esta situación, almacenes y centros logísticos de todo el mundo se ven en la necesidad de renovar todo el sistema tanto de organización como de trasporte para poder abastecer la demanda exigida.

Amazon es la empresa por excelencia que ha sufrido y resuelto con gran mérito el problema del se está hablando, la solución que ellos proponen y ponen en marcha es la del empleo de grandes almacenes de paquetería controlados por vehículos con guiado automático (AGV) que son los que distribuyen y ordenan de forma autónoma las estanterías en estos almacenes. Esto reduce considerablemente el espacio de almacenaje ya que evita que los operarios tengan que pasar entre ellas, además disminuye también el tiempo de intercambio de paquetes ya que el propio AGV acercaría la estantería que contiene el paquete en cuestión hasta la posición del operario.

Además de esto, existen infinitas posibilidades para el mundo de la robótica móvil, otro sector que se encuentra en auge actualmente es el de la agricultura, sobre todo en Almería, las oportunidades que nacen de la unión entre estos dos mundos son muy amplias ya que la forma de trabajo que se lleva ahora mismo depende demasiado del operario y es muy laboriosa y fatigante.

Por todo esto, el enfoque de este trabajo de fin de estudios se centra en la investigación y desarrollo de un sistema que combine un sistema de localización para interiores preciso como es el kit de desarrollo basado en UWB “MDEK 1001” [2] con un robot móvil como es el Robot iCreate 2 [3].

1.2. Objetivos

El principal objetivo de este trabajo de fin de estudios es la integración de un sistema de posicionamiento para interiores en el iRobot Create®2 para lograr así eliminar el error generado por los encoders implementados ya en el robot a la hora de determinar la posición actual.

En primer lugar, se realizará un estudio preliminar del estado del arte, analizando varias tecnologías de posicionamiento indoor así como métodos de cálculo para determinar la posición, además se estudiarán trabajos previos realizados que hagan uso del robot que se va a emplear.

Por otra parte, se ha elegido ROS (Robot Operation System) como sistema operativo base para controlar el robot y para comunicarse con los diversos periféricos que se emplean, la elección se debe a la gran versatilidad que ROS aporta y del cual se hará también un estudio antes de comenzar con la implementación.

También se realizará un estudio del robot en cuestión de sus como: sensores y actuadores, así como de sus limitaciones

El hecho de aunar todos los sistemas en uno solo será una parte importante de este trabajo ya que se debe conseguir la unión tanto física como por software de todos los dispositivos empleados (balizas de posicionamiento, Raspberry Pi 4 Model b [4], batería…)

Una vez se consiga esto, se debe trasladar todo el trabajo realizado al robot real para la implementación final de todo el sistema en conjunto con su correspondiente programación previa que permitirá cumplir las tareas exigidas.

Tras las correspondientes pruebas pertinentes, se analizarán los resultados obtenidos para poder validar si finalmente se consigue el propósito que del TFE.

Finalmente, se elaborará la documentación del proyecto recogiendo todos los procesos seguidos y resultados obtenidos a lo largo del presente proyecto.

1.3. Planificación temporal

Este TFG se ha estructurado en ocho actividades diferenciadas que se reflejan en la Tabla 1, por otra parte, en la Tabla 2 aparecen el número de horas que se han empleado en cada actividad divididas por meses.

Actividades	Descripción
A	Investigación sobre métodos de posicionamiento basados en BLE
B	Investigación sobre sistemas de posicionamiento para interiores
C	Investigación sobre sistema de posicionamiento DECAWAVE
D	Investigación y aprendizaje de ROS
E	Desarrollo y programación del sistema
F	Pruebas con el robot y el sistema de posicionamiento
G	Pruebas en el invernadero
H	Redacción de memoria del proyecto

Tabla 1: Descripción de actividades realizadas en el proyecto

	Actividad	A	B	C	D	E	F	G	H	Total
Número de horas de trabajo por mes	**Febrero**	25								
	Marzo	25								
	Abril		25							
	Mayo		15							
	Junio			10	10					
	Julio				20					
	Agosto				20	20				
	Septiembre					20				
	Octubre			5		10	10		5	
	Noviembre			5	5	10	15		10	
	Diciembre			5	5	5	15	5	10	
	Enero			5				5	30	
	Total									350

Tabla 2: Número de horas empleadas en cada actividad por mes

1.4. Competencias utilizadas en el TFE

A lo largo de todo el transcurso del grado de Ingeniería Electrónica industrial, se adquieren ciertas competencias que te dan pie tanto a la superación de las propias asignaturas de dicho grado como posteriormente a la habilitación profesional que se adquiere tras la graduación, se podrían diferenciar entre tres grandes bloques de competencias, las cuales son: competencias básicas, transversales y específicas. A continuación, se expondrán todas aquellas que han sido requeridas para la composición del presente Trabajo de Fin de Estudios o TFE.

Las competencias básicas para todos los títulos de grado vienen definidas en el R.D. 1393/2007, de 29 de octubre y están dirigidas a la adquisición por parte del estudiante de una formación general, las competencias básicas desarrolladas en este proyecto son las siguientes:

- CB1: Poseer y comprender conocimientos.
- CB2: Aplicación de conocimientos.
- CB3: Capacidad de emitir juicios.
- CB4: Capacidad de comunicar y aptitud social.
- CB5: Habilidad para el aprendizaje.

A continuación, se muestran las competencias transversales de la Universidad de Almería, que fueron aprobadas por el Consejo del Gobierno, de 17 de junio de 2008, desarrolladas en la confección de este trabajo de fin de estudios.

- UAL1: Conocimientos básicos de la profesión.

- UAL2: Habilidad en el uso de las TIC.
- UAL3: Capacidad para resolver problemas.
- UAL4: Comunicación oral y escrita en la propia lengua.
- UAL5: Capacidad de crítica y autocrítica.
- UAL6: Trabajo en equipo.
- UAL7: Conocimiento de una segunda lengua.
- UAL8: Compromiso ético.
- UAL9: Capacidad para aprender a trabajar de forma autónoma.
- UAL10: Competencia social y ciudadanía global.

Por último, las competencias específicas del grado que conciernen a este trabajo son:

- CT3: Conocimiento en materias básicas y tecnológicas, que les capacite para el aprendizaje de nuevos métodos y teorías, y les dote de versatilidad para adaptarse a nuevas situaciones.
- CT4: Capacidad de resolver problemas con iniciativa, toma de decisiones, creatividad, razonamiento crítico y de comunicar y transmitir conocimientos, habilidades y destrezas en el campo de la Ingeniería Industrial.
- CT5: Conocimientos para la realización de mediciones, cálculos, valoraciones, tasaciones, peritaciones, estudios, informes, planes de labores y otros trabajos análogos.
- CT6: Capacidad para el manejo de especificaciones, reglamentos y normas de obligado cumplimiento.
- CT7: Capacidad de analizar y valorar el impacto social y medioambiental de las soluciones técnicas.
- CT10: Capacidad de trabajar en un entorno multilingüe y multidisciplinar.
- CB1: Capacidad para la resolución de los problemas matemáticos que puedan plantearse en la ingeniería. Aptitud para aplicar los conocimientos sobre: álgebra lineal; geometría; geometría diferencial; cálculo diferencial e integral; ecuaciones diferenciales y en derivadas parciales; métodos numéricos; algorítmica numérica; estadística y optimización.
- CB2: Comprensión y dominio de los conceptos básicos sobre las leyes generales de la mecánica, termodinámica, campos y ondas y electromagnetismo y su aplicación para la resolución de problemas propios de la ingeniería.

- CB3: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.

- CB5: Capacidad de visión espacial y conocimiento de las técnicas de representación gráfica, tanto por métodos tradicionales de geometría métrica y geometría descriptiva, como mediante las aplicaciones de diseño asistido por ordenador.

- CRI4: Conocimiento y utilización de los principios de teoría de circuitos y máquinas eléctricas.

- CTEE9: Conocimientos de principios y aplicaciones de los sistemas robotizados.

- CTEE10: Conocimiento aplicado de informática industrial y comunicaciones.

- CTEE11: Capacidad para diseñar sistemas de control y automatización industrial.

- CTEQ2: Capacidad para el análisis, diseño, simulación y optimización de procesos y productos.

- TFG: Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería Industrial de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

2. Estado del arte

En este capítulo se hablará sobre las diferentes problemáticas que acarrea el hecho de trabajar con robots y querer conocer su posición en todo momento y además se presentarán las diferentes tecnologías que existen hoy en día para el posicionamiento en interiores, así como los métodos empleados para calcular la posición con dichas tecnologías.

2.1. El problema de la localización en robótica

En el panorama actual de la robótica, y en concreto de la robótica móvil, que es la parte que se trata en el presente trabajo, existe una problemática común, la de conocer la localización junto con la orientación del robot en tiempo real. Esto se debe a que para obtener dichos datos se debe hacer uso de información sensorial aportada por el propio robot empleando sistemas de medición de magnitudes físicas como por ejemplo los encoders, sin embargo, este tipo de sistemas, nos son todo lo fiables que deberían. Esta baja fiabilidad es debida a que estos sensores hacen uso de los datos obtenidos anteriormente para obtener la posición y orientación actual del robot, lo que implica tener dos condiciones previas al uso de este robot en cualquier entorno fijo (no dinámico).

- Conocer la posición inicial del robot
- Conocer la orientación inicial del robot

Además del hecho de tener estas condiciones previas, el error en ambas mediciones por parte de los sensores implementados en el error se va acumulando conforme este navega por el entorno en cuestión, lo que en casos en los que se necesite una gran precisión a la hora de posicionar el robot en una ubicación concreta conllevará a un fracaso casi asegurado por parte del sistema.

A raíz de la casuística anteriormente mencionada, surge la opción de llevar a cabo la integración de sistemas de posicionamiento globales en los robots móviles. Con esto lo que se consigue es tener una posición en tiempo real del robot dentro del entorno en el que se encuentre. Sin embargo, esto no solucionaría del todo el problema de la orientación, que se debería solucionar empleado sensores como las IMUs o giróscopos o bien empleando el mismo sistema de localización de forma que se conozca la posición de la parte frontal y trasera del robot, con esto y un simple cálculo se podría obtener la orientación del móvil, aunque este método quedaría restringido a robots móviles que se muevan en el plano XY ya que en otros casos se podrían obtener resultados erróneos como por ejemplo que el sistema no interprete correctamente si el robot móvil se encuentra en la posición original o dado la vuelta.

2.2. Sistemas de localización en exteriores

El estudio del posicionamiento se ha centrado en gran parte en el conocimiento de la posición a nivel global, es decir una localización dentro de la Tierra. Estos sistemas son nombrados por las siglas GNSS (*Global Navigation Satellite System)*, lo cual abarca todos los sistemas de posicionamiento geoespacial y temporal. Los diferentes sistemas GNSS que existen, difieren entre ellos en síntesis al país o comunidad de países a los que estos satélites pertenecen. Estos

sistemas son: GPS (*Global Positioning System*) controlada por los Estados Unidos, Galileo administrada por Europa, GLONASS perteneciente a Rusia y Beidou que es propiedad de China.

El funcionamiento de un GNSS, sea el que sea, se basa en una combinación de tres segmentos imprescindibles [8].

- Segmento espacial: Este está compuesto por dos tipos de satélites, unos que se encargan de la navegación situándose en diferentes planos orbitales alrededor de la Tierra, los segundos forma los sistemas de aumento que ayudan a la corrección de errores de posicionamiento.

- Segmento de control: Consiste en las estaciones terrestres que se encargan de recoger los datos tomados por los diferentes satélites, cada país lleva cabo esta función acorde a sus especificaciones y necesidades, pero sus funciones básicas son el garantizar el correcto funcionamiento del sistema así como las correcciones orbitales necesarias para que cada satélite se encuentre en la posición correcta, estas tareas se realizan con el apoyo de un reloj atómico para la sincronización entre satélites .

- Segmento del usuario: Esta parte se compone de los equipos que toman las señales emitidas por el segmento espacial.

Entre todos los sistemas de posicionamiento en exteriores que se han mencionado el más famoso y utilizado es el GPS, cuyo funcionamiento se basa en una costelación de 31 satélites que orbitan a unos 20.200 km de altura, cada uno de los cuales completa una vuelta a la Tierra cada doce horas. Estos se sitúan en diferentes planos orbitales para de esta forma conseguir que en todo momento se cubran todas las zonas del planeta. La señal que estos satélites emiten, que se llama efeméride, es captada por los dispositivos electrónicos preparados para esta función. Con la acción combinada de las señales de varios satélites (3 como mínimo, necesitando un cuarto para conocer la altitud), se puede conocer la posición del dispositivo mediante el uso de métodos como la trilateración con una precisión de unos cinco metros, pudiendo aumentar hasta un metro con diversos sistemas de control y el uso de algunas bandas de comunicación. En este sistema es crucial la sincronización entre los satélites, ya que de esta sincronización depende la alta exactitud del sistema. Varias estaciones de las fuerzas aéreas de los Estados Unidos se encargan de colocarlos en hora con respecto a un reloj atómico y vigilar que se encuentran en la posición correcta [8].

2.3. Sistemas de localización en interiores

Una vez que la posición a localizar se encuentra dentro de una construcción como un edificio, una nave industrial o cualquier recinto cerrado, los sistemas de posicionamiento comienzan a perder precisión puesto que estos funcionan con ondas electromagnéticas las cuales rebotan en los obstáculos que se encuentras por el camino entre el emisor y receptor. Además de este problema, nos encontramos con que normalmente estos sistemas no tienen la precisión que se necesitas para ciertos trabajos destinados a realizarse bajo techo.

A raíz de esto surgen nuevos métodos de posicionamiento para estas condiciones con mayor versatilidad y precisión, los cuales suelen emplear diversos protocolos de comunicación u otras tecnologías, las cuales se dividirán en dos grandes grupos: las tecnologías basadas en radio

frecuencia y las que no están basadas en radio frecuencia. Que se combinan con algunos métodos matemáticos para dar lugar a sistemas de posicionamiento robustos y precisos.

2.3.1. Tecnologías de posicionamiento basadas en radio frecuencia

Los protocolos empleados para el posicionamiento en interiores no tienen por qué ser necesariamente diseñados para este propósito, sino que se emplean diversos protocolos de comunicación o tecnologías que interactúan con el medio físico a través de ciertos sensores y se adaptan para que funcionen como nos convenga. Todas las tecnologías que se van a esponer a continuación tienen en común que emplean ondas electromagnéticas de altas frecuencias para realizar comunicaciones entre dispositivos que compartan protocolo.

- **Wi-Fi**: Esta es una tecnología que se origina para la creación de redes de área local inalámbricas (*WLAN: Wireless local area networking*) de dispositivos que se basa en el estándar de red IEEE 802.11 y que opera en las bandas de radio ISM (*Industrial, Scientific and Medical*) de 2,4 y 5 GHz. En la actualidad existe un gran número de dispositivos electrónicos que emplean esta tecnología, como pueden ser: ordenadores, smartphones, televisiones inteligentes, impresoras, videoconsolas e incluso coches. Debido a que este protocolo es ampliamente utilizado y se encuentra por todas partes, desde hogares hasta espacios públicos. El estudio de los sistemas de posicionamiento en interiores basados en Wi-Fi está en auge por la gran cantidad de puntos de acceso existentes [8].
  Como principales ventajas, se podría destacar el gran número de puntos de acceso ya operativos, lo que reduce el coste de instalación para un nuevo sistema. Alguna de las desventajas serían por ejemplo que la distancia es limitada y el consumo de potencia es alto sobre dispositivos activos.

- **BLE (Bluetooth Low Energy)**: BLE se lanzó como una versión 4.0 de Bluetooth en junio de 2010 con el objetivo de ser usado por dispositivos que no requieran una gran transferencia de datos y está pensado para la trasmisión inalámbrica a corto alcance y con bajo consumo y coste. Al igual que Wi-Fi, BLE opera en una banda ISM de 2,4 GHz la cual se divide en 40 canales espaciados a 2 MHz [6]. Unos de los principales sistemas que emplean este protocolo son los iBeacos que más adelante se explicaran.
  Una de las principales ventajas de este protocolo es la facilidad de creación de redes inalámbricas AD-HOC entre dispositivos. Como desventaja se puede destacar que el alcance para el intercambio de información es reducido.

- **UWB (Ultra Wide Band)**: Esta tecnología se caracteriza por el bajo consumo de energía para las comunicaciones de corto alcance y por el gran ancho de banda en el que opera en una gran parte del espectro radioeléctrico. Una onda de radio emitida se considera UWB si su ancho de banda supera los 500 MHz o el 20% de la frecuencia portadora. Además de esto, esta tecnología tiene una penetración eficaz a través de materiales densos y una menor sensibilidad al efecto multi trayecto debido a una duración muy corta de los pulsos UWB lo que hace que sea idóneo para el desarrollo de sistemas de posicionamiento en interiores. Hasta el momento algunos de los últimos modelos de la marca Apple y Samsung poseen un chip UWB [6]. Esta es la tecnología con la que trabaja el sistema Decawave el cual se ha elegido para el desarrollo de este TFG.
  Las ventajas de UWB son sobre todo la capacidad de compartir el espectro de frecuencias y el buen rendimiento que tiene en espacios con diferentes elementos que generan la

reflexión de las ondas. Por otra parte, las desventajas que esta presenta es la distorsión de la forma del pulso debido al amplio rango de frecuencias cubierto y que también requiere de una alta velocidad de conversión analógica-digital.

- **RFID (Radio Frequency Identification):** Los sistemas basados en esta tecnología necesitan dos componentes esenciales, una etiqueta RFID y el lector RFID para cumplir con su tarea. El lector toma de forma inalámbrica la información almacenada en las etiquetas, este contiene un transceptor para trasmitir señales de radio frecuencia y leer los datos emitidos por las etiquetas. Dentro de las etiquetas existen dos categorías: pasivas y activas. Las etiquetas pasivas toman la energía de las señales de radio entrantes mientras que se necesita de una batería para alimentar las activas. RFID opera en cuatro bandas de frecuencia, baja frecuencia (125kHz), alta frecuencia (13,56MHz), ultra alta frecuencia (433,868-915 MHz) y frecuencia de microondas (2,45 GHz, 5,8Ghz). La propiedad de detectar y reconocer etiquetas cercanas hace de RFID una tecnología a tener en cuenta para los sistemas de posicionamiento en interiores. Algunos sistemas basados en esta utilizan etiquetas pasivas desplegadas en el suelo a una distancia conocida formando una cuadrícula y se estiman los resultados de localización mediante la detección de múltiples etiquetas [6].
  Dentro de las ventajas que proporciona RFID es resaltable el hecho de que la fabricación de las etiquetas está destinada a trabajar en ambientes hostiles por lo que la vulnerabilidad del sistema es menor, lo que le da una versatilidad notable. Como principales desventajas cabe destacar el alto precio en comparación a las demás tecnologías, también es común que las etiquetas sufran daños ya que deben estar expuestas en casi todos los casos al proceso industrial completo y problemas ocasionados por el solapamiento de activación y lectura, es decir, otros dispositivos que empleen radio frecuencia podrían interferir en las comunicaciones de este protocolo causando conflictos.

- **ZigBee:** Esta es una tecnología inalámbrica de corto alcance y bajo consumo que nace de la antigua alianza HomeRF y que se define como solución inalámbrica de baja capacidad para aplicaciones en el hogar como la seguridad y la automatización o como en este caso el posicionamiento. La meta de esta tecnología no está en alcanzar altas velocidades ya que solo puede alcanzar una tasa de 20 a 250Kbps en un rango de 10 a 75 metros, sino en obtener sensores cuyos transceptores tenga un consumo muy bajo para reducir el mantenimiento y ampliar las opciones de utilización, de hecho, existen dispositivos que alimentados con dos pilas AA pueden llegar a aguantar en activo hasta unos dos años. Las bandas en la que trabaja ZigBee son las libres de 2,4 GHz, 858MHz para Europa y 915MHz para Estados Unidos [7].
  Las ventajas de ZigBee son su bajo coste y consumo además con cada paquete de datos se adjunta el indicador de fuerza de señal lo que ayuda al desarrollo de sistemas de posicionamiento. Como desventajas, ZigBee necesita trabajar con una infraestructura y un hardware muy específico.

2.3.2 Tecnologías que no se basan en radio frecuencia

Todas estas tecnologías que se expondrán a continuación tienen en común que son usadas para crear sistemas de posicionamiento y además no emplean ondas electromagnéticas para llevar a cabo este cometido.

- **Ondas sónicas y ultrasónicas:** Esto se basa en ondas mecánicas que viajan por el aire, dichas ondas pueden ser longitudinales o transversales. En el caso de las transversales, las partículas se mueven perpendiculares a la dirección de las ondas. Sin embargo, en las ondas longitudinales, el movimiento de las partículas será en la misma dirección que el movimiento de la onda. Para medios acuáticos, las ondas son transversales, en contraposición a las ondas del sonido, las cuales son longitudinales. Dos de las propiedades más atractivas de las ondas sónicas sobre las ondas de radio son: la velocidad de propagación más lenta y la barrera efectiva formada por paredes y otros obstáculos. Las ondas sonoras viajan por la atmósfera a unos 343,2 m/s, mientras que las ondas de radio suelen tener una velocidad de 300.000.000 m/s. Por lo tanto, a la hora de hacer mediciones de tiempo de vuelo, nos resultará más sencillo si lo hacemos empleando ondas sonoras. Además, como estas se ven muy obstaculizadas por las paredes, se tiene la posibilidad de confinarlas mejor y así obtener más precisión en situaciones de salas pequeñas. Sin embargo, este tipo de ondas también presentan una serie de problemas, entre otros, la velocidad de propagación de estas ondas depende en cierto modo de la temperatura, humedad y presión barométrica, dando esto lugar a fluctuaciones entre 290 y 360 m/s. Otro factor que influye esta velocidad es el viento, el sonido viaja más rápido en la dirección del viento y más lento en la dirección opuesta. También este tipo de ondas se ven mayormente afectadas por el ruido de fondo en comparación con los sistemas de radiofrecuencia [8].

- ***Inertial Navigation System* (INS) :** Los sistemas de navegación inercial son utilizados comúnmente en naves espaciales, misiles guiados, submarino y aviones. Previamente de la llegada del GPS, el INS era el mecanismo más popular para el conocimiento de la posición. Estos sistemas se consideran autónomos, ya que todas las medidas se realizan por medio de sensores implementados en el propio sistema sin tener que recurrir a ninguna entrada externa. Este tipo de sistemas dio un gran paso hacia adelante con el desarrollo de los sistemas microelectromecánicos (MEMS), lo que ha conseguido la miniaturización de los sensores. Esto también ha permitido la implementación de sistemas como este en dispositivos que están al alcance de nuestra mano como pueden ser los Smartphones con fines de posicionamiento entre otros. Los tres principales componentes de los MEMS son: el acelerómetro, el giroscopio y la brújula magnética. Estos MEMS son implementados en lo que se conoce como unidad de navegación inercial (INU) o bien unidad de medición inercial (IMU), el método por el cual se obtiene la ubicación a través de las mediciones inerciales se denomina *Dead Reckoning* (DR). En el ámbito de interiores, la localización se basa en los peatones. El DR peatonal (PDR) halla la posición empleando esencialmente la velocidad, la orientación y la dirección [8].

- ***Simultaneous Locationing and Mapping* (SLAM):** La localización y mapeo simultáneos (SLAM) es una técnica en la que se construye un mapa al mismo tiempo que se localiza al robot en cuestión. Dicho mapa puede ser desconocido para el robot en un principio. El robot hace uso de sensores como: RADAR, LIDAR, SONAR, extracción de imágenes,

etc. Para medir distancias a ciertos puntos de referencia y, a medida que se desplaza, va construyendo un mapa de la zona. Al mismo tiempo, también emplea las señales de entrada proporcionadas por los sensores para ubicarse dentro de este mapa. El SLAM se desarrolló originalmente con el fin de que los robots pudieran navegar por terrenos desconocidos. También cabe la posibilidad de que previamente a la navegación se le incorpore al propio robot un mapa de la zona que se conoce a priori. Algunos de los algoritmos que se han empleado para el desarrollo de estos sistemas son: punto más cercano interactivo (ICP), el filtro de Kalman extendido a la 6ª dimensión (6D-EKF), el filtro de partículas Rao-Blackwellized (RBPF), el basado en gráficos, etc. Debido a las incertidumbres en el comportamiento de los sensores y actuadores, se necesitan técnicas estadísticas en lugar de deterministas para obtener buenos resultados [8].

- **Realidad Aumentada (RA):** En los últimos años, con la popularización sobre todo de los smartphones, la realidad aumentada se está convirtiendo en una tecnología práctica. El concepto básico de la RA es superponer información sobre una imagen en tiempo real. Uno de los primeros usos de estos sistemas ha sido en el ámbito deportivo. Durante los partidos de fútbol americano, por ejemplo. Existen dos enfoques de la RA. En uno de ellos, el GPS u otra tecnología es para la localización, la orientación de la dirección a la que apunta la cámara se determina mediante la información de una brújula si está disponible, a continuación, basándose en la información de la ubicación y orientación se superpone la información contextual. La alineación de la información contextual con la imagen es tan precisa como lo sean las medidas de posición y orientación, por lo tanto, la superposición puede estar muy desviada. El segundo enfoque no depende tanto del GPS o la brújula y aumentan esa información con la capacidad de reconocimiento de la imagen. Así si alguien está en Barcelona y mira hacia la Basílica de la Sagrada Familia, con el reconocimiento de la imagen, la ubicación precisa de la Basílica es identificada en la imagen. Por ende, la información superpuesta puede ser alineada con precisión. Varias empresas han realizado grandes avances en este ámbito, algunas con el objetivo evidente de la ayuda en lugares nuevos para ciertas personas como el turismo, búsqueda y rescate o la orientación de las fuerzas de seguridad. Otra aplicación es la de la superposición de la hoja de datos o manual de montaje de un producto sobre el propio producto. Esta nueva forma de trasmisión de información tiene un gran potencial a la hora de aumentar la productividad [8].

2.3.3. Métodos de posicionamiento empleados

En este apartado se explicarán los métodos empleados para el posicionamiento en interiores, aunque dichos métodos sean usados sobre todo en las tecnologías que emplean la radiofrecuencia, existen también algunos sistemas como los basados en ultrasonidos que también emplea alguno de estos métodos para determinar la posición. Los métodos más empleados son: Time of Flight (ToF), Angle of Arrival (AoA) y Recived Signal Strength Indicator (RSSI).

Algunas de las tecnologías de radio de gran fama emplean alguno de estos métodos, por ejemplo, GPS se basa en gran parte en ToF y el RADAR es una combinación de ToF y AoA.

- ***Time of Flight* (ToF):** Este ha sido el método más adoptado a la hora de medir la distancia a través de tecnologías basadas en radiofrecuencia. Esto se basa en la medición del tiempo

que tardan las ondas electromagnéticas en recorrer cierta distancia. Debido a que la velocidad de la luz en el vacío y en el aire son muy similares y altas, es necesario realizar estas mediciones en unidades muy pequeñas de tiempo como puede ser el sub-nanosegundo. A partir de este método surgen diferentes derivaciones que son: Time of Arrival (ToA), Time Differential of Arrival (TDoA) y Time Transfer.

En el método ToA, el emisor envía una señal en un momento conocido para el receptor, de algún modo el reloj del receptor se sincroniza con el del trasmisor, por lo tanto, el receptor puede restar el tiempo de transmisión del tiempo de recepción para obtener el ToF, tras esto, solo quedará multiplicarlo por la velocidad de la luz para obtener la distancia entre emisor y receptor. Si se midiera la distancia en dos dimensiones, tan solo con conocer la distancia de dos trasmisores es suficiente para conocer la localización que se quiere obtener. También se puede obtener la posición mediante un método no sincronizado, en este caso, el tiempo de ida y vuelta se calcula en el receptor mediante el intercambio de información entre receptor y trasmisor.

En el caso del TDoA, se elimina la necesidad de sincronizar receptor y trasmisor, pero requiere una sincronización entre los trasmisores. Puesto que la sincronización no es tan estricta, para dispositivos desarrollados con bajo presupuesto se emplea el TDoA. La ventaja que presentan las técnicas basadas en ToA y TDoA es que el error cometido o la varianza no aumenta con la distancia como en el caso de AoA y RSS.

En la Figura 1 se representa el posicionamiento de un objeto a través del método ToA en un sistema de referencia de dos dimensiones, para la localización en un sistema de tres dimensiones la matemática es similar, pero con mayor complejidad.

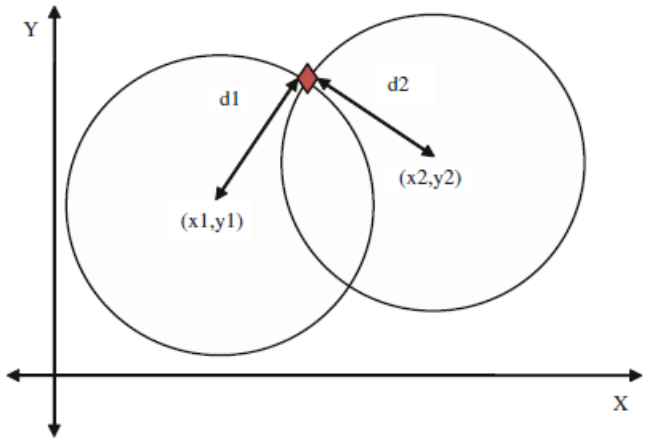


Figura 1: Método ToA en sistema de referencia 2D

La distancia al nodo cuya posición quiere ser determinada lo que se denomina *tag* desde dos nodos fijos a lo que se nombrarán *anchors*, esto situaría al *tag* en la intersección de dos círculos centrados alrededor de los *anchors*, como resultado de esto se obtienen dos puntos tal y como se observa en la figura los cuales se toman de las soluciones dadas por el siguiente conjunto de ecuaciones cuadráticas:

$$(x - x_1)^2 + (y - y_1)^2 = {d_1}^2 \quad (1)$$

$$(x - x_2)^2 + (y - y_2)^2 = {d_2}^2 \quad (2)$$

Para tres dimensiones se necesitan al menos la distancia de tres *anchors*, por lo tanto, se tendrían que resolver tres ecuaciones cuadráticas con lo que se obtiene igualmente dos puntos. La información adicional y el conocimiento del trazado pueden ayudar a eliminar uno de los puntos. En el empleo de técnicas como esta sobre el campo, se utilizan más *anchors* de los estrictamente necesarios por lo que la precisión aumentará. Hay varios métodos matemáticos para tratar sistemas sobre-determinados, uno de los más populares es la solución por mínimos cuadrados [8].

- ***Angle of Arrival* (AoA):** Este método trata de obtener la posición de un objeto mediante la medición del ángulo que existe entre los anchors y un tag. Para un sistema de dos dimensiones, con la obtención de ángulos θ_1 y θ_2 es suficiente para posicionar correctamente el tag, siendo (x_1, y_1) y (x_2, y_2) las posiciones de los *anchors* 1 y 2 respectivamente y (x, y) la posición del tag (Figura 2).

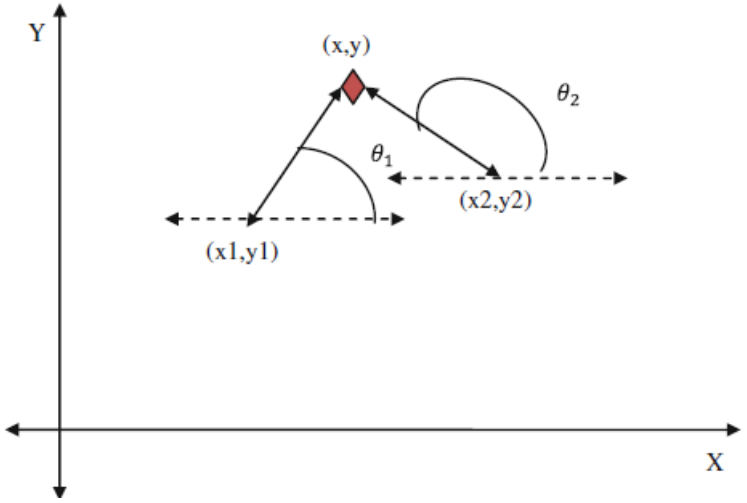


Figura 2: Representación del método AoA en sistema de referencia 2D

La ubicación del tag vendría dada por la resolución de las dos siguientes ecuaciones:

$$\tan(\theta_1) = \frac{y - y_1}{x - x_1}, \tan(\theta_2) = \frac{y - y_2}{x - x_2} \quad (3)$$

$$y_i - x_i \tan(\theta_i) = y - x\, tan(\theta_i) \quad (4)$$

Para trasladar esto a un sistema de referencia en 3D se deben proyectar los planos X-Y, e Y-Z o X-Z. Como representación, en el plano Y-Z el ángulo con el eje Z será φ y las ecuaciones serían:

$$y_i - x_i \tan(\theta_i) = y - x\, tan(\theta_i) \quad (5)$$

$$y_i - z_i \tan(\phi_i) = y - x\, tan(\phi_i) \quad (6)$$

Que se pueden transformar en las ec.(7) y ec.(8) y sería posible resolverlo mediante mínimos cuadrados.

$$y_i - x_i \tan(\theta_i) = y - x\, tan(\theta_i) - z * 0 \quad (7)$$

$$y_i - z_i \tan(\phi_i) = y - x * 0 - z tan(\phi_i) \quad (8)$$

La medición física de los ángulos, pueden presentarse diversas dificultades tales como las múltiples trayectorias, es decir, las ondas pueden verse sometidas a fenómenos de reflexión, refracción y otros que podrían inferir en el trayecto de la onda generando errores en el ángulo de recepción o el crecimiento del error de localización a medida que incrementa la distancia entre el *anchor* y el *tag*, esto se debe cuanto mayor sea la distancia peor será la medición del ángulo de llegada de la señal con lo que la posción obtenida se verá perjudicada [8].

- ***Signal Strength*:** El empleo de la intensidad de la señal es uno de los métodos más sencillos para la estimación de la distancia entre dispositivos, la señal del transmisor disminuye gradualmente su intensidad a medida que el receptor se aleja del mismo. En un ambiente completamente ideal, dicha intensidad disminuye como la inversa del cuadrado de la distancia tal y como se puede comprobar en la siguiente ec.(9). P_0 es la intensidad de la señal a la distancia r_0.

$$P(r) = \frac{r_0^2 P_0}{r^2} \tag{9}$$

Esta idealidad está evidentemente distorsionada por el multitrayecto, es decir la propagación de las ondas por diferentes caminos debido a los fenómenos de reflexión y refracción, y una antena no isotópica, es decir que no emite señales con la misma intensidad en todas las direcciones. Existen dos tipos de atenuaciones, una a gran escala y otra a pequeña escala, si esta última es eliminada o ignorada, numerosos estudios han hallado una relación logarítmica empírica entre la distancia y la intensidad como se ve en la ec.(10):

$$\log P(r) = a + b \log r \tag{10}$$

Los parámetros *a* y *b* se hallan tras tomar un pequeño número de mediciones. Una vez hecho esto, las distancias se pueden calcular empleando la ec.(10). Al igual que en otros métodos, en un sistema de referencia de tres dimensiones, son necesarias al menos tres *anchors* para definir la posición de un *tag*. El problema del multitrayecto puede originar diversos errores a pequeña escala debido a la variación de intensidades que se producen a lo largo de una distancia de longitud de onda, debido a que si una misma onda llega al receptor por dos o más diferentes caminos causaría conflictos a la hora de calcular la intensidad de esta señal introduciendo errores en la medida de la posición [8].

- **Otros métodos:** Además de todos los métodos ya mencionados, existen muchos otros, por ejemplo, midiendo las fases de dos o más longitudes de onda. Se transmite un tono de onda continua hacia un objetivo, el cual puede reflejar activa o pasivamente una señal. La diferencia de fase entre la señal transmitida y la recibida se emplea para medir la distancia, como se puede observar en la ec.(11). Este método es funciona bien en distancias del orden de la diferencia de longitud de onda. Aunque un problema patente en este método es que el hecho de emplear una un solo tono limita el alcance a la longitud de una sola onda, ya que la diferencia de fases se repite en intervalos de longitud de onda. Al utilizar dos tonos en lugar de uno se alivia este problema.
  La relación entre fase y distancia se ve reflejada en la ec.(11):

$$2d = \frac{(\theta_1 - \theta_2)}{2\pi(\frac{1}{\lambda_1} - \frac{1}{\lambda_2})} \quad (11)$$

Siendo *d* la distancia entre emisor y receptor, λ_1 y λ cada una de las longitudes de onda y θ_1 y θ_2 la fase de la onda en cuestión.

Por otro parte, existe también la posibilidad de medir distancias empleado un láser, aunque los métodos empleados por estos sistemas son adaptables también a las tecnologías basadas en radio frecuencia, por ejemplo, en el caso de ToF con láser, el sistema se compone de un trasmisor de pulsos láser, un fotodetector y un sistema de medición de sub-nanosegundos. Un pulso muy corto se envía hacia un objeto el cual refleja una parte que es detectada por el fotorreceptor y el sistema de medición de tiempo recoge el intervalo trascurrido en este proceso.

Otro enfoque para la solución con láser es lo que se conoce como *Beam-Modulation Telemetry*, que se basa en enviar un haz hacia un objetivo el cual reflejará parte del mismo, la diferencia de fase entre el haz emitido y la fase de la parte reflejada es proporcional a la distancia, la frecuencia de modulación puede ser sintonizada para que coincida con el rango de distancia a medir [8].

3. Materiales y métodos

En este capítulo se hará un repaso por todos los elementos que se han utilizados en el trascurso de este TFG tanto los elementos físicos como pueden ser las balizas de posicionamiento o el robot, como los métodos empleados para resolver problemas como el del seguimiento de trayectoriar para robots.

3.1. Sistemas de posicionamiento

A continuación, se explicarán las características de los sistemas de posicionamiento para interiores de los que se disponía, aunque en el caso de Pozyx e iBeacons se han realizado pocas pruebas ya que no ha habido la posibilidad de actualizar los firmwares de cada uno de estos para que puedan ser actualizados debido a que se adquirieron hace mucho tiempo.

3.1.1 Pozyx®

Pozyx es un RTLS basado en UWB creado en 2015 con el objetivo de ayudar al desarrollo de la industria 4.0 dando solución a las diferentes necesidades de localización que esta requiere en algunas ocasiones.

Este sistema está basado en UWB al igual que el sistema DECAWAVE como se verá más tarde, es decir, está compuesto por un sistema de balizas (Figura 3) donde unas funcionan como anchor y otras como tags siendo flexible a la hora de la elección del rol de las balizas. Uno de los puntos fuertes de este sistema es su amplia información que está disponible en internet además de sus estrechos lazos con sistemas mundialmente conocidos como Raspberry Pi o Arduino, de hecho, los sistemas de Pozyx suelen estar enfocados a un control por medio de uno de estos dos sistemas, aunque pueden ser controlados y configurados por otros medios.

Figura 3: Balizas Pozyx

3.1.2 iBeacons

Estos dispositivos, emplean BLE para realizar el posicionamiento en interiores de sus propios dispositivos, en cierta medida, en smartphones, por ejemplo, pueden ser utilizados para compensar la señal de GPS en interiores, pero el principio de posicionamiento de estos difiere al GPS ya que estos usan el método RSSI para obtener la distancia entre dispositivos. Estos pequeños dispositivos (Figura 4) están preparados para ser usados con un teléfono móvil que hace de intermediario entre ellos.

La trama de difusión de iBeacon utiliza una señal de "*Advertising*" (Anuncio) para enviar un mensaje, el aviso es emitido periódicamente a los dispositivos BLE, siempre y cuando estos puedan recibir la señal.

Los datos enviados por los iBeacos están por cuatro tipos de información. El Universal Unique Identifier (UUID), *Major* , *Minor* y la potencia medida. UUID se define como un identificador estándar de 128 bits ISO/IEC11578: 1996. Por ejemplo, las tiendas pueden utilizar el nombre UUID en la oficina central, *Major* en los representantes regionales y *Minor* en el nombre de una tienda en concreto. Por otro lado, la potencia medida da la información sobre la distancia entre el módulo emisor y receptor, lo que se conoce como Received Signal Strength Indicator (RSSI).

Este sistema de balizas no es demasiado preciso ya que no da localizaciones precisas, sino que juzga la distancia en tres categorías. Si los dispositivos están muy ceraca se considerará como "muy cercano", si están a un metro o menos como "cerca" y si se da el caso de que estén a más de un metro serán señalados como "lejos" [9].

Figura 4: Balizas iBeacons

3.1.3 Kit de desarrollo basado en UWB "MDEK 1001"

Como sistema de localización para el desarrollo de este proyecto se ha elegido este kit de desarrollo MDEK 1001 basado en el chip DWM1001 (Figura 5) desarrollado por la marca DECAWAVE®. Este módulo DWM1001 es un chip versátil que soporta el firmware Positioning and Networking Stack (PANS), software también propiedad de la marca.

Figura 5: Módulo DWM1001

Este módulo permite desarrollar e implementar sistemas de posicionamiento en tiempo real (RTLS). El módulo tiene la posibilidad de configurarse como anchor o como tag. Esta configuración puede realizarse por comunicación Bluetooth utilizando la aplicación complementaria (Decawave DRTLS Manager) o mediante SPI o UART desde un host externo. El módulo incorpora como parte principal el transceptor DW1000 UWB (Figura 6) que es controlado por el firmware para realizar los intercambios bidireccionales con los nodos de la etiqueta para la creación y gestión de la red, permitiendo que cada etiqueta calcule su propia ubicación.

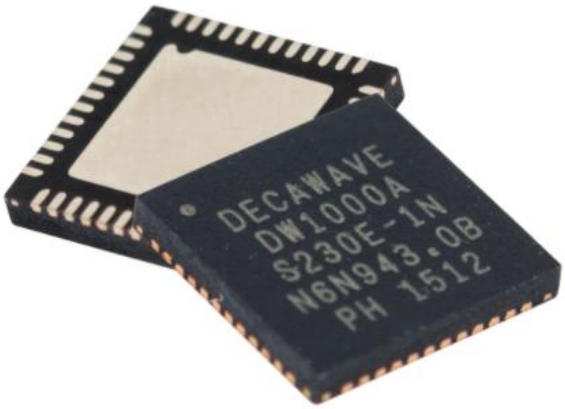


Figura 6: Transceptor DW1000

El DWM1001 posee también el IC NRF52832 de Nordic Semiconductor el cual da la posibilidad de conexión Bluetooth para la configuración y el microprocesador que ejecuta el firmware y proporciona a la función de habilitar el RTLS. Todo esto se puede montar sobre una PCB, sin embargo, la propia marca ofrece también la posibilidad de un kit de desarrollo llamado MDEK1001 que contiene lo siguiente:

- 12 balizas DWM1001 de desarrollo (Figura 7)
- Cable USB de 1m
- 4 conexiones USB en ángulo

- 8 Stickers de diversos colores
- 8 adhesivos para colocar las balizas en superficies verticales
- Manual de inicio rápido

Figura 7: Baliza DWM1001 de desarrollo

3.2 Plataforma utilizada

El centro neurálgico de todo este sistema se basa en el robot de limpieza iRobot Create® 2 y en el sistema operativo con el que será controlado que en este caso es el *Robot Operating System* (ROS)

3.2.1. iRobot Create® 2

Este es un robot de movimiento diferencial con batería recargable de y un peso de 3000mAh unos 3,5kg. Las dimensiones de dicho robot son las siguientes (Figura 8):

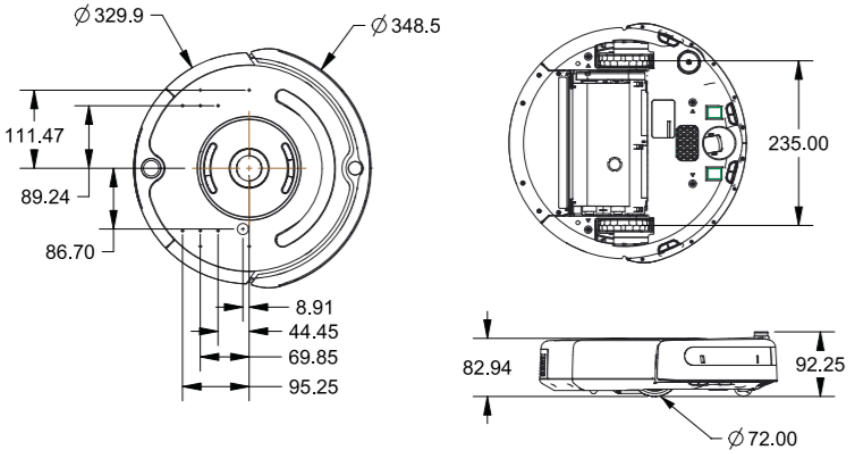


Figura 8: Dimensiones iRobot Create®2

Para tomar información del medio el robot posee sensores y actuadores para realizar acciones en consecuencia. Los sensores principales son [3]:

- Sensores infrarrojos: El robot tiene seis sensores infrarrojos, cuatro horizontales para la detección de obstáculos (Figura 9) y dos verticales para comprobar si el robot se dirige a un precipicio (Figura 10).

Figura 9: Sensores infrarrojos horizontales

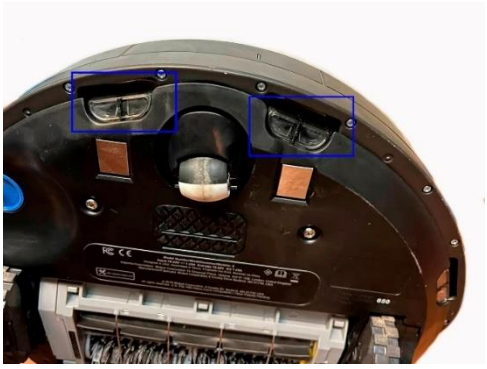


Figura 10: Sensores infrarrojos verticales

- Bumper: Este sensor está colocado en la parte frontal del robot (Figura 11) y sirve para detectar las colisiones de este con otros objetos.

Figura 11: Sensor bumper

- Sensores de caída de rueda: Este sensor (Figura 12) sirve para que el robot notifique si no se encuentra sobre el suelo, y se consigue dándole cierto juego vertical a las ruedas, de tal modo que, si el robot está apoyado en el suelo, las ruedas estarán dentro de su hueco y si el robot está suspendido en el aire, las ruedas estarán hacia afuera.

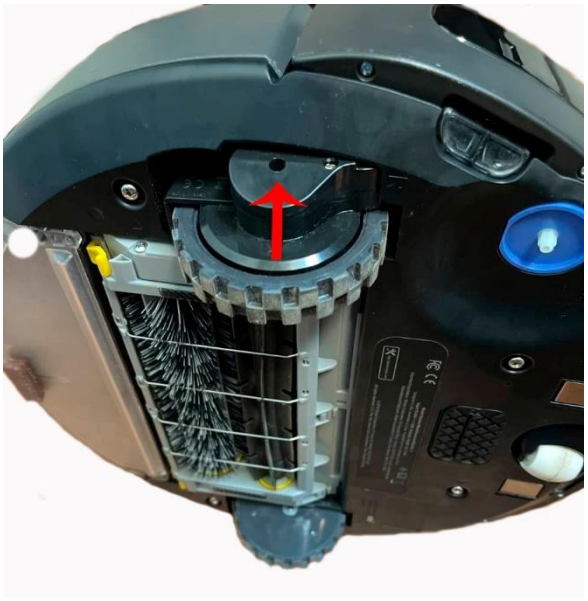


Figura 12: Sensor de caída

- Encoders: Estos sensores están implementados en los ejes de las ruedas y sirven para conocer la posición y orientación del robot. Estos tienen una resolución de 508,8 cuentas por vuelta.

- Sensor infrarrojo de comunicación (Figura 13): este sirve para comunicare con la estación de carga del robot y colocarse encima de ella automáticamente para proceder a cargarse.

Figura 13: Sensor infrarrojo de comunicación

- Otros sensores: A parte de estos, el robot tiene otros sensores de estado como pueden ser el que indica el nivel de batería.

Por otro lado, los principales actuadores son [3]:

- Motores DC: Posee dos motores de corriente continua para darle movilidad a las ruedas tanto linear como curva, alcanzan una velocidad máxima de 0,5 m/s y son capaces de describir curvas con un radio de hasta dos metros, se pueden ver señalados de color rojo en la (Figura 14).

- Cepillo y aspirador: Ambos sirven para llevar a cabo el trabajo para el que este robot fue diseñado que es el de barrer y aspirar la suciedad del suelo. Se puede ver reflejado de color azul en la (Figura 14).

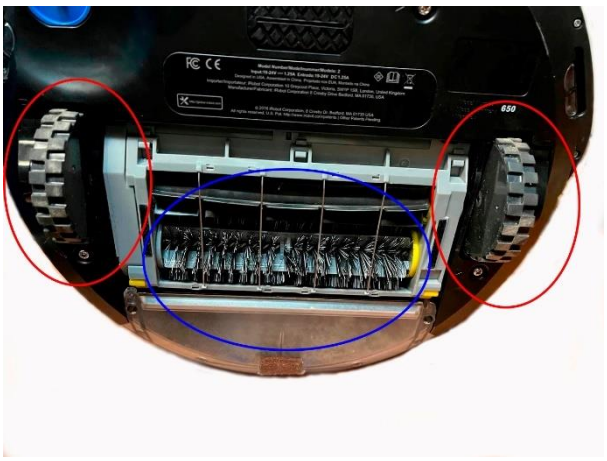


Figura 14: Actuadores motorizados del robot

- Altavoces: El robot tiene varios altavoces para dar señales acústicas en ciertos momentos de funcionamiento como cuando se conecta al cargador, por ejemplo.

- LEDs: Con estos, el robot se comunica con el usuario a través de señales lumínicas que pueden significar diferentes cosas como que le falta batería.

El robot tiene varios modos de funcionamiento automáticos que se describirán brevemente [3]:

- Modo apagado: este se ejecuta tras un cambio de batería y la primera vez que se enciende, el robot se queda en un estado de “off” a la espera de que se pulse el botón de “Start”. Una vez hecho esto, se pueden ejecutar cualquiera de los otros modos.

- Modo pasivo: En este modo se pueden recibir las lecturas de los sensores, sin embargo, no se pueden mandar consignas a los actuadores, para esto se debe ejecutar el modo seguro o completo.

- Modo seguro: En este modo se puede operar con el robot con libertad siempre y cuando no se den alguna de las siguientes condiciones: Conexión del cargador, detección de un precipicio o detección de caída de una rueda.

- Modo completo: Con este modo la libertad sobre el uso del robot es total, sin restricción alguna.

Aunque el propio sistema de iRobot permite la programación del robot a través de estructuras preensambladas, se ha decidido que esta plataforma no cubre las necesidades requeridas por el proyecto, debido a esto se empleará una Raspberry Pi como centro de control y ROS como sistema operativo para controlar todos los sistemas.

3.2.2 Robot Operating System (ROS)

ROS es un middleware robótico de código abierto que permite el desarrollo de sistemas robóticos complejos [10]. El éxito de este pseudo sistema operativo viene dado por el apoyo que muestra la comunidad de desarrolladores para mejorar las prestaciones de este a través de trabajo y esfuerzo.

¿Cómo nace ROS?

El principio de este proyecto aparece en 2007 bajo el nombre de Switchyard por el Laboratorio de Inteligencia Artificial de Stanford para da apoyo al proyecto del Robot con Inteligencia Artificial de Standford (STAIR). Tras esto, de 2008 a 2013, este intento de unificar los enfoques fragmentados de la robótica fuero tomados por Google y apoyado por Willow Garage, y más tarde por la OSRF (Fundación de Código Abierto de Google) hasta la actualidad. El objetivo de ROS siguió la tendencia moderna de software de código abierto y colaboración distribuida. Adeás se ha unido y aprovechado los esfuerzos paralelos de código abierto de proyectos tan famosos cómo OpenCV, PointCloudLibrary, Open Dynamics Engine, Gazebo y Eigen. Gracias al apoyo que Google le ha proporcionado, ROS ha sobrevivido al periodo de incubación crucial para cualquier proyecto de este tipo y actualmente es usado en todo el mundo y en todo tipo de industrias y empresas [11].

¿Qué es ROS?

Aunque ya se ha dado una definición aproximada de lo que es ROS, realmente es difícil dar una sola función a este que ROS abarca una gran cantidad de aspectos, como el estilo de programación; definiciones de interfaz y paradigmas para la comunicación entre nodos; definiciones de interfaz para la incorporación de bibliotecas y paquetes; una colección de herramientas para la visualización, depuración, registro de datos y diagnóstico del sistema; un repositorio de código fuente compartido; y puentes a múltiples bibliotecas útiles e independientes de código abierto. ROS es, por tanto, una forma de vida para los programadores de robots más que un simple sistema operativo [11].

¿Cómo funciona ROS?

Para entender el funcionamiento de este peculiar “sistema operativo”, primero se deben conocer los tres niveles de conceptos sobre los que este se basa: el nivel de sistema de archivos, el nivel del gráfico de computación y el nivel de la comunidad.

Nivel del sistema de archivos de ROS

En este nivel, se cubren principalmente los recursos de ROS que encuentra en el disco como [12]:

- **Paquetes:** Son la unidad principal para organizar el software en ROS, un paquete puede contener *runtime process* (nodos), librerías de dependencias de ROS, conjuntos de datos, archivos de configuración o en su defecto cualquier otro conjunto de elementos que se

útil almacenar en grupo. Los paquetes son el elemento de compilación más pequeño de ROS.

- **Metapaquetes:** Son paquetes especializados que sirven para representar un grupo de otros paquetes relacionados.

- **Manifiestos del paquete:** Proporcionan metadatos sobre un paquete, incluido su nombre. versión, descripción, información de licencia, dependencias etc.

- **Repositorios:** Son colecciones de paquetes que comparten un sistema de control de versiones (VCS). Los paquetes que comparten un VCS comparten la misma versión.

Nivel gráfico de computación de ROS

Esta es la red *peer-to-peer* (punto a punto) de procesos, este tipo de redes se basan en que la comunicación entre los elementos que la componen es descentralizada es decir que no dependen de un punto común aunque sí que necesitan de un registro de nombres que en este caso es el *Master*. Estos son los elementos básicos que lo componen [12].

- **Nodos:** Son procesos que realizan cálculos computacionales. ROS está diseñado para ser modular, por lo general, un sistema de control de robot comprende muchos nodos. Por ejemplo, un nodo controla un sensor de proximidad mientras otro tiene el control sobre los motores del mismo robot y otro sobre un sistema de cálculo de trayectorias. Un nodo ROS se escribe con una biblioteca de cliente ROS, como roscpp o rospy, con esto, se demuestra una de las grandes características de ROS que es la posibilidad del multilenguaje, con las configuraciones adecuadas, un proyecto no tiene por qué estar escrito con el mismo lenguaje, estos nodos se pueden escribir en lenguaje C++ o en su defecto en Python. El lenguaje elegido para la confección de los nodos creados para el sistema desarrollado ha sido Python del cual se hablará más tarde.

- **Master:** El ROS Master da registro de nombres y la búsqueda del resto del *Computation Graph.* Sin el Master, los nodos no serían capaces de encontrarse, intercambiar mensajes o invocar servicios.

- **Servidor de Parámetros:** Esto actualmente forma parte del Master y permite que los datos sean almacenados de forma encriptada en una ubicación central.

- **Mensajes:** Para comunicarse, los nodos usan los mensajes, esto es simplemente una estructurade datos. Son admitidos los tipos de datos primitivos tales como: enteros, flotantes, caracteres, cadenas, booleanos, etc. Al igual que las matrices o estructuras propias basadas en este tipo de datos.

- **Topics:** Los mensajes se enrutan a través de un sistema de publicación/suscipción. Un nodo "publica" un mensaje en *Topic* determinado y otro nodo suscriptor de este lo recibe. El nombre de un *Topic* se emplea para identificar el contenido del mensaje el cual será usado tanto por el publicador como por el suscriptor. Cabe la posibilidad de que el mismo *Topic* tenga varios publicadores y suscriptores simultáneos al igual que un nodo puede publicar y suscribirse a varios *Topics.* Normalmente, estos interlocutores que intercambian información no tienen por qué conocer la existencia de los demás publicadores o suscriptores. Se podría decir que un *Topic* es un bus de mensajes, cada

bus tiene un nombre y cualquiera puede conectarse al bs para enviar o recibir mensajes siempre y cuando sean del tipo correcto.

- **Servicios:** El modelo de publicación/suscripción es un paradigma de comunicación muy flexible, sin embargo, su trasporte unidireccional de muchos a muchos no es el más idóneo para las interacciones de solicitud/respuesta que son requeridas poen un sistema distribuido como este. La solicitud/respuesta se lleva a cabo mediante los servicios, que se definen con un par de estructuras de mensajes, una para la solicitud y otra para la respuesta. Un nodo proveedor da un servicio bajo un nombre y un cliente emplea ese nombre para a través de un mensaje de solicitud obtener una respuesta.

- **Bags:** Los *Bags* son un formato para guardar y reproducir datos de mensajes ROS, son un mecanismo de importancia para almacenar datos como por ejemplo lecturas de sensores o revoluciones de un motor.

El ROS Master actúa como servicio de nombres, almacena la información de registro de *Topis* y servicios para nodos. Los nodos se comunican con el Master para notificar su información de registro. A medida que estos nodos se comunican con el Master, pueden realizar conexiones entre ellos recibiendo y enviando datos. El Master también realiza devoluciones de llamada a estos nodos cuando cambie la información de registro, lo que permite que se creen conexiones dinámicas a medida que se ejecuten o maten los diversos nodos.

Los nodos se conectan entre sí directamente, el Master por lo tanto solo proporciona información de búsqueda como si de un servidor DNS se tratase. Los nodos que se suscriben a un *Topic* solicitarán conexiones de los nodos que publican en eso *Topic* y entablarán conexión mediante el protocolo acordado, en este caso, el más comúnmente utilizado es TCPROS, que hace uso de los sockets TCP/IP estándar.

Toda esta arquitectura, da la oportunidad de una operación desacoplada, donde los nombres son el medio principal por el cual se pueden construir sistemas más grandes y complejos. Estos nombres tienen un papel crucial en ROS: nodos, *Topics*, servicios y parámetro, todos tienen su nombre propio y único, dos elementos con un mismo nombre entrarían en conflicto cerrando la ejecución uno de otro [9]. A continuación, se muestra un pequeño gráfico del funcionamiento en síntesis de ROS (Figura 15)

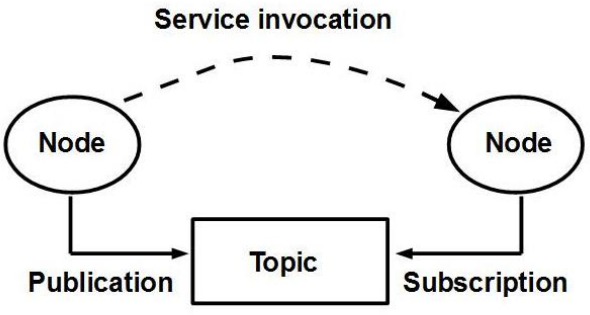


Figura 15: Gráfico explicativo del funcionamiento de ROS

Nivel de la comunidad de ROS

Los conceptos de la comunidad de ROS son recursos del mismo que permiten a comunidades separadas el intercambio de software y conocimientos. Estos recursos incluyen [12]:

- **Distribuciones:** Son colecciones de las diversas versiones de ROS que se pueden instalar. Cada una de las versiones va ligada a una versión de Linux, para la instalación en otros sistemas operativos habría que recabar más información.

- **Repositorios:** ROS está basado en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y ejecutar sus propios componentes del robot.

- **La Wiki de ROS:** Este es el foro principal para documentar información sobre ROS.

De entre todas las versiones de ROS que se ofrecen, para este proyecto se ha elegido *ROS Melodic Morereira* debido a su compatibilidad con diversos drivers que dan la posibilidad de conectarse al robot tal y como se verá más adelante, esta versión de ROS va ligada a Ubuntu 18.04 del cual se hablará posteriormente.

3.3. Hardware y software auxiliar

Además del robot y de ROS, han hecho falta varios elementos adicionales sin los cuales no hubiese sido posible llevar a cabo este proyecto.

3.3.1. Raspberry pi 4 Model B

Esta “mini computadora” (Figura 16) ha sido elegida como centro de control de todo el sistema robótico que se ha llevado a cabo debido a su gran versatilidad y la amplia red de información existente en torno a ella debido a la gran comunidad que engloba a Raspberry y Linux.

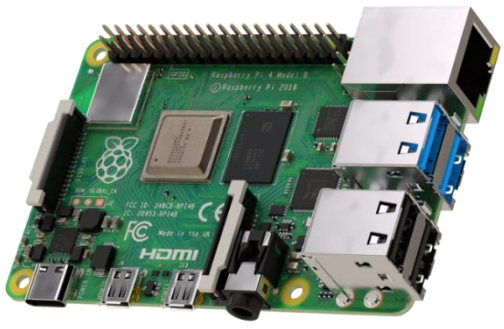


Figura 16: Raspberry Pi 4 Model B

Las especificaciones concretas de este modelo son:

- Procesador ARM Cortex-A72

- Frecuencia de reloj de 1,5GHz
- GPU VideoCore VI (con soporte para OpenGL ES 3.x)
- Memoria de 4 GB LPDDR4 SDRAM
- Conexión Bluetooth 5.0
- Conexión Wi-fi 802.11ac
- Conexión Gigabit Ethernet
- 40 pines GPIO con diversos usos
- 2 puertos micro HDMI
- 2 puertos USB 2.0 y 2 puertos USB 3.0
- CSI para la cámara de Raspberry.
- DSI para la pantalla táctil
- Ranura para microSD
- Conexión tipo C para alimentación (5V, 3A)

3.3.2 HP Pavilion Notebook

Este es el ordenador (Figura 17) con el que se han realizado toda la programación y las pruebas con el robot real y la Raspberry Pi, conectándose a ella mediante VNC.

Estas son las especificaciones de este modelo de HP:

- Procesador Intel Core i7-9750H 2,6GHz
- RAM de 8GB
- Tarjeta gráfica Nvidia Geforce GTX
- Disco duro de 500GB SSD

Figura 17: HP Pavilion Notebook

3.3.3 Samsung Galaxy Tab A

Esta Tablet (Figura 18) se ha utilizado para crear y configurar las redes para usar el sistema DECAWAVE, las características más relevantes de esta son:

- Procesador Exynos 7904 Octo-core (2x 1,8 GHz 6x 1,6 GHz).
- RAM de 2 GB.
- Almacenaciento de 32GB apliable por medio de tarjeta de memoria microSD
- Conectividad: Bluetooth 5.0, GPS Wi-fi y USB tipo C.
- Sistema operative Android 9.0 Pie

Figura 18: Samsung Galaxy Tab A

3.3.4. Baterías recargables RCR123a

Estas baterías recargables (Figura 19) han sido seleccionadas con el propósito de que alimente las balizas fijas del sistema de posicionamiento en tiempo real, con esto se consigue terne mayor libertad a la hora de crear una red de nodos de balizas ya que no se depende de una red eléctrica cercana. Estas baterías son de 3,7V y 2800mAh.

Figura 19: Bateriass recargables RCR123a

3.3.5. Power Bank HETP 26800mAh

Como último componente del sistema se ha adquirido una Power Bank (Figura 20) para la alimentación de la Raspberry, ya que de por si el iRobot Create® 2 no da la suficiente alimentación para que la Raspberry funcione correctamente.

Esta batería de polímero de litio tiene tres salidas:

- QC 3.0: 5V/3A
- PD: 5V/3A
- USB: 5V/2.1A

Y para la carga de la misma posee dos entradas diferentes:

- PD: 5V/3A
- Micro USB: 5V/2.1A

Para esta aplicación emplearemos la salida QC 3.0 con un cable USB-C para un correcto funcionamiento de la Raspberry.

Figura 20: Power Bank HETP

3.3.6. Linux

Un ordenador o una computadora se componen por la conjunción de diversos elementos independientes como un teclado, una pantalla, un procesador, una memoria, etc. Todos ellos dan la posibilidad al usuario del empleo de los servicios que estos elementos ofrecen. De entre todos los componentes, el procesador es el que más relevancia tiene dentro del conjunto puesto que es el que posee la capacidad de ejecución dentro del sistema. Al realizarse una petición, por parte del usuario, en la que se vean involucrados varios elementos, será necesario un elemento que sea el encargado de la coordinación de las diferentes partes, algo así como un "cerebro". Aquí es donde entra en juego el concepto de sistema operativo, que se define como un programa que determina las acciones a realizar por cada componente de la máquina y coordina la realización de dichas acciones, es decir el usuario se comunica con el sistema operativo por medio de instrucciones y este se encarga de transformarlas en la secuencia de instrucciones hardware

adecuadas. Por lo tanto el sistema operativo se puede ver también como un software que ayuda al usuario a no tener que lidiar de manera tediosa con el hardwar, así que los dos objetivos principales de este software serian por un lado facilitar la comunicación máquina-usuario y por otro lado gestionar los recursos de la máquina [13].

En lo que a este proyecto concierne, la elección del sistema operativo acarrea bastantes consecuencias ya que se trabajará con una computadora de hardware "limitado", así que se debe escoger dentro de las posibilidades el que más se ajuste a las necesidades requeridas.

Para profundizar un poco más, cuando se trabaja con Raspberry Pi, se suele emplear una distribución de Linux, esto es un sistema operativo Unix con licencia GNU GPL (Licencia Publica General de GNU). Linux aparece en octubre de 1991 de la mano de un estudiante finlandés llamado Linus Torvalds, el cual presenta la versión 0.011 de su kernel de sistema operativo orientado a máquinas Intel 386 y lo ofreció bajo licencia GPL a foros de programadores, para que fuera probado y mejorado por los mismos programadores, lo que causó sensación entre la comunidad, de esta forma en poco tiempo ya había una gran cantidad de programadores trabajando en el núcleo o en aplicaciones nuevas para este sistema operativo. Las principales características que separan a Linux de los demás sistemas operativos son [14]:

- **Sistema operativo de código abierto:** Esto significa que cualquier persona puede tener acceso a sus fuentes y por lo tanto modificarlas y crear nuevas versiones.

- **Portabilidad:** Al igual que el UNIX original, Linux se pensó para tener muy poca dependencia sobre la arquitectura de la máquina sobre la que se monta, así que tiene la capacidad de adaptarse a prácticamente cualquier arquitectura que tenga un compilador C.

- **Kernel de tipo monolítico:** El diseño del kernel está unido en una sola pieza, sin embargo, es modular en las diferentes tareas. El problema de los kernel monolíticos es que se vuelven intratables en el desarrollo cuando se hacen demasiado grandes, lo que se intentó solucionar con los módulos cargables.

- **Módulos dinámicos cargables:** Esto lo que permite es poner partes del sistema operativo como pueden ser *filesystems* o controladores de dispositivos, como pedazos externos que se cargan o enlazan con el kernel en tiempo de ejecución bajo demanda. Todo esto simplifica el kernel y permite programar estas funcionalidades por separado. Con motivo de esto, se podría considerar a Linux un kernel mixto, ya que es monolítico, pero ofrece una serie de módulos que complementan el kernel.

- **Desarrollo del sistema por una comunidad vinculada a internet:** Nunca antes en la historia de los sistemas operativos, había existido uno con tal desarrollo por parte de la misma comunidad que es usuaria de este sistema operativo. El fenómeno de la comunidad Linux da la posibilidad de que cada uno colabore en la medida que el tiempo los propios conocimientos del usuario se lo permitan. Como resultado, Linux es un laboratorio ideal para testear ideas de sistemas operativos al mínimo coste.

Dentro de este amplio sistema operativo llamado Linux, existen una amplia variedad de distribuciones que se ajustan a las necesidades que el propio sistema requiera, en este caso debe

de adecuarse en primer lugar a la Raspberry Pi y en segundo lugar a Robot Operating System (ROS) de lo que se hablará después. Las siguientes distribuciones son todas las que se han usado:

- **Ubuntu 18.04 Desktop:** Esta es una versión de escritorio de Ubuntu el cual está basado en Debian, basado a su vez en Linux. Este fue descartado tras varias pruebas debido a que era un sistema operativo demasiado pesado y por lo tanto daba bastantes fallos en su ejecución en Raspberry Pi. Sin embargo, este es el sistema operativo elegido para ser instalado de forma nativa en el ordenador en el que se desarrolla el proyecto el cual actuará de pueste entre el usuario y el sistema robótico.

- **Raspberry Pi OS:** Anteriormente llamado Raspbian Buster, es el sistema operativo basado en Debian propio de Raspberry Pi. La versión instalada es la de escritorio para así tener una interfaz gráfica. Este sistema operativo da algunos problemas como el hecho de que no tenga ciertos paquetes precompilados de ROS, los cuales fueron compilados posteriormente de forma local. Ha sido el sistema operativo elegido por la versatilidad y fluidez que proporciona al sistema.

3.3.7. Python

Python es en resumen un lenguaje de programación de alto nivel, interpretado y multipropósito. Últimamente el uso de este lenguaje ha ido en aumento siendo hoy en día uno de los mas utilizados en el desarrollo software.

Python tiene la capacidad de ser utilizado en una gran cantidad de plataformas lo que le da una versatilidad importante, dentro de las plataformas más importantes podemos destacar Windows, Mac Os X y Linux, pero además puede funcionar también en smartphones.

Algunos lenguajes de programación salen a la luz con un único objetivo, por ejemplo, PHP fue ideado para el desarrollo de aplicaciones web. Sin embargo, con Python es posible llevar a cabo gran cantidad de propósitos, aplicaciones científicas, comunicaciones de red, aplicaciones de escritorio con interfaz gráfica de usuario (GUI), creación de juegos y por supuesto para aplicaciones web. Este lenguaje es utilizado por grandes empresas como la NASA, Google, Nokia y muchas más para el desarrollo de productos y servicios, esto da una visión de la potencia y la extensión que tiene este lenguaje hoy día.

Además, Python no requiere dedicar tiempo a su compilación ya que este es interpretado. Por otra parte, este es *open source, es decir,* cualquier persona tienen la posibilidad de contribuir a su desarrollo y divulgación y no es necesario pagar ninguna licencia para distribuir software desarrollado con este lenguaje, es más, hasta su intérprete se distribuye de forma gratuita para diferentes plataformas, la última versión de este lenguaje tiene varios nombres entre ellos,Python 3000 y Py3k aunque comúnmente se le denomina con el nombre de Python 3 [15].

3.3.8. Matlab®

Matlab® es un potente programa con una infinidad de posibilidades, en este proyecto ha sido empleado como simulador para verificar algunos modelos matemáticos antes de ser probados en el sistema real, aunque con este también se pueden hacer simulaciones a mayor escala como pueden ser de modelos de sistemas, circuitos eléctricos, estructuras mecánicas etc.

Esta es una de las muchas y sofisticadas herramientas de computación que hay disponibles tales como Maple o Mathematica. Cada una de estas permie hacer cálculos matemáticos básicos, pero difieren a la hora de manejar los cálculos simbólicos y procesos matemáticos más complejos como la manipulación de matrices, por ejemplo. El nombre de Matlab® es una abreviatura de Matrix Laboratory, es decir laboratorio de matrices. Dado que la programación en l lenguaje propio de esta herramienta es más sencillo que C++ o FORTRAN, por ejemplo, muchas tareas se llevan a cabo con él. El programa destaca sobre todo en cálculos numéricos y especialmente en lo relacionado con matrices y gráficas, sin embargo, no es tan bueno en procesamiento de palabras. Como conclusión, si es necesario programar un script para la solución de un problema que involucre matrices y gráficas es conveniente emplear esta herramienta que será más rápida y eficaz que otros programas de propósito general como los mencionados anteriormente [16].

A modo ilustrativo, se muestra a continuación una pequeña explicación y una imagen (Figura 21) de la interfaz de Matlab® con sus respectivas partes:

- **Editor:** En esta zona será donde se deba escribir el código de los diferentes scripts que se necesiten.
- **Comand Window:** Aquí será donde por una parte se interactúa en tiempo real con las diferentes variables a través de una línea de código y a su vez también es donde aparecerán los diferentes elementos a mostrar por pantalla en un script que ejecutemos.
- **WorkSpace:** Aquí se verán reflejados los resultados obtenidos tras la ejecución de los scripts, tanto el valor que tienen los diversos elementos que componen el script como las dimensiones de las matrices, por ejemplo.
- **Folder Explorer:** Desde aquí se tiene la posibilidad de navegar a través del árbol de capetas del ordenador en el que se esté trabajando para poder abrir y cerrar los diversos archivos relacionados con Matlab®.

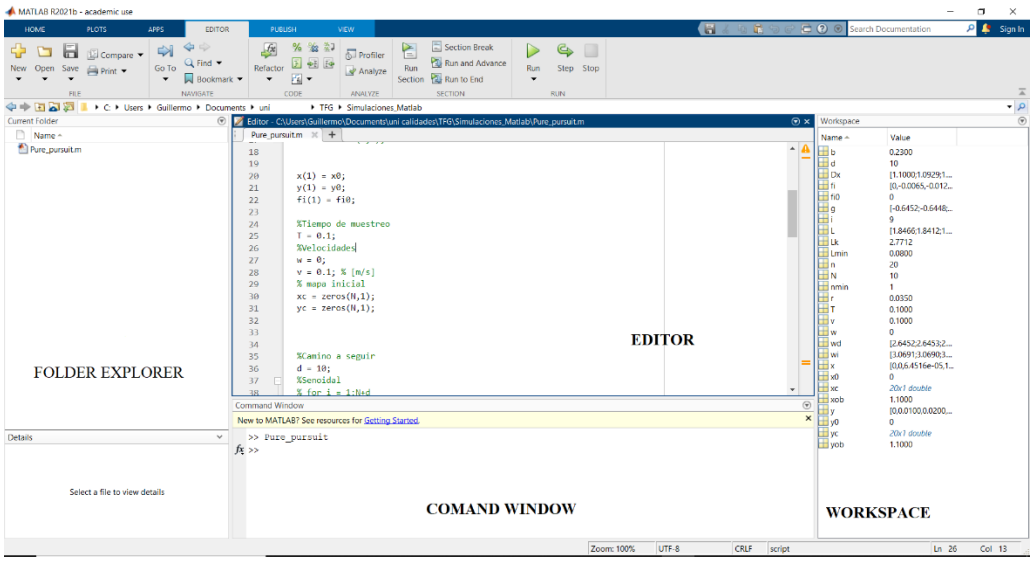


Figura 21: Interfaz de Matlab®

3.4. Algoritmo de seguimiento *Pure Pursuit.*

Este es un método de seguimiento de trayectorias para robot móviles, en este caso el robot con el que se está trabajando es un robot de tipo diferencial, lo cual se adecua a la perfección con el método operativo de *Pure Pursuit* que se describirá en las siguientes líneas.

Primeramente, se debe fijar un sistema de referencia local asociado al movimiento del propio robot (Figura 22). Se supe que en el intercalo de control, es decir donde se fija el objetivo, la curvatura tiene un valor constante, describiendo el vehículo una circunferencia. Del análisis de la figura se deduce:

$$r = \Delta x + d \tag{12}$$

$$d^2 + (\Delta y)^2 = r^2 \tag{13}$$

Si se despeja la *d* de la ec. (12) y se sustituye en la segunda:

$$(r - \Delta x)^2 + (\Delta y)^2 = r^2 \tag{14}$$

De donde el radio de curvatura necesario paraque el vehículo se desplace *Δx, Δy* es:

$$r = \frac{(\Delta x)^2 + (\Delta y)^2}{2\Delta x} \tag{15}$$

Por lo tanto, la curvatura que se le debe de suministrar al vehículo es:

$$\gamma_r = \frac{1}{r} = -\frac{2\Delta x}{L^2} \tag{16}$$

Donde el signo vendría dado por el sentido de giro necesario para alcanzar el punto objetivo. *L* es la distancia a la que se encuentra el punto objetivo y *Δx* es el desplazamiento lateral.

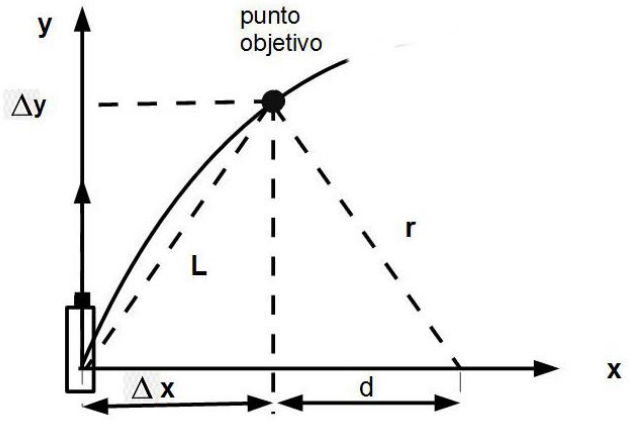


Figura 22: Sistema de referencia para control por *Pure Pursuit*

La ec.(16) constituye la ley de control de persecución pura o *Pure Pursuit*. Se puede ver como esta ley es proporcional al error latera (Δx) con respecto al punto objetivo y la constante de proporcionalidad o ganancia varia con la inversa del cuadrado de L.

Observando la figura se puede comprobar también que la curvatura de la persecución pura es la inversa del radio de la circunferencia que pasa por la posición actual del vehículo y por el punto objetivo.

Para usar este método en un robot móvil real. Primero hay que determinar un punto del camino real a una distancia previamente definida. Después se debe calcular el error lateral (Δx) con respecto a la posición actual del centro guiado del robot. Si las coordenadas están sujetas a un sistema global, que lo están, es necesario conocer la orientación del robot para obtener (Δx). Así que si el robot se encuentra en la posición (x,y) con una orientación φ (Figura 23) y el punto objetivo está en (x_{ob}, y_{ob}) entonces se tiene:

$$\Delta x = (x_{ob} - x)cos\phi + (y_{ob} - y)\, sen\phi \qquad (17)$$

En la práctica, para aplicar este método se suele bucar primeramente mediante un método iterativo u otros similares el punto perteneciente a la trayectoria más cercano al robot (x_{obm}, y_{obm}), y se toma un punto objetivo (x_{ob}, y_{ob}) a una distancia fija d previamente elegida (Figura 23). La elección de este parámetro d es crucial a la hora de que el sistema funcione correctamente ya que existe determinadas situaciones que podrían poner en apuros a este método. Por ejemplo, si el punto objetivo se encuentra muy alejado, la actuación suministrada por este método puede ser muy pequeña y que se tarde mucho en alcanzar este objetivo. En este caso se debería de sustituir esta distancia por una distancia máxima fijada. Otra situación problemática sería si se diera el caso de que el robot se encuentra sobre la trayectoria, pero con orientación opuesta, en este caso la curvatura resultante sería demasiado pequeña y sería posible salir de esta situación, en el caso de que se detectara esta situación, sería necesario sustituir dicha curvatura por otra arbitraria de mayor valor [17]

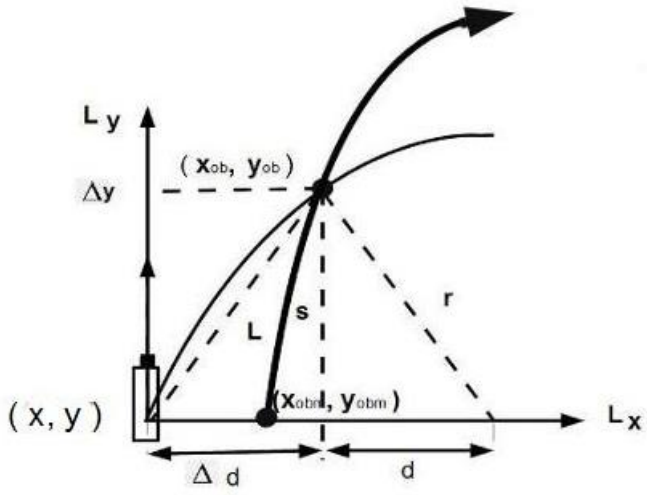


Figura 23: Aplicación de *Pure Pursuit*

Como más tarde se verá, las señales que se le envían al robot iCreate son tanto la de velocidad lineal como la de velocidad angular para que consiga seguir el camino marcado, por lo tanto, se deben trasformar estas ecuaciones para que fijando una de las dos velocidades, la lineal en este caso, se pueda obtener la angular para así darle una consigna que seguir a los actuadores [18].

Primero se debe obtener la velocidad angular de cada rueda, izquierda y derecha:

$$\omega_i = \frac{v}{r}\left(1 - \frac{b}{2}\right)\gamma \qquad \omega_d = \frac{v}{r}\left(1 + \frac{b}{2}\right)\gamma \tag{18}$$

Donde *b* es la separación de las ruedas del vehículo, *r* el radio de las ruedas y *v* la velocidad lineal previamente fijada, con este se hallaría:

$$\phi' = \omega = \frac{-r}{b}\omega_i + \frac{r}{b}\omega_d \tag{19}$$

Y con esto se obtendría el valor de la consigna a seguir en cada punto de control del algoritmo.

4.Resultados

En este capítulo se tratará de explicar la metodología seguida para el desarrollo del sistema robótico, dividiendo este en diferentes partes que irán abarcando todas las pruebas realizadas tanto a nivel de simulación como de ensayos con el sistema real.

4.1. Pruebas de localización con las balizas

Primeramente, se han comenzado las pruebas con el sistema de posicionamiento adquirido, del que ya se ha hablado anteriormente, que es el "kit de desarrollo MDEK 1001" de DECAWAVE. Antes que nada, se hará una breve explicación de como se ha configurado este sistema con el uso de la Tablet "Samsung Galaxy Tab A" y la aplicación que el propio sistema proporciona.

Una vez instalada la aplicación DRTLS, lo primero que se debe de hacer es encender el Bluetooth del dispositivo ya que la comunicación con las balizas será por medio de este protocolo, tras esto se debe configurar una nueva *Network* con el número de balizas que estén operativas. En este caso se ha creado una red de cuatro *anchors* y dos *tags* a la cual se ha llamado "Superficie 4m2" (Figura 24) ya que es la superficie que cubrirá esta red.

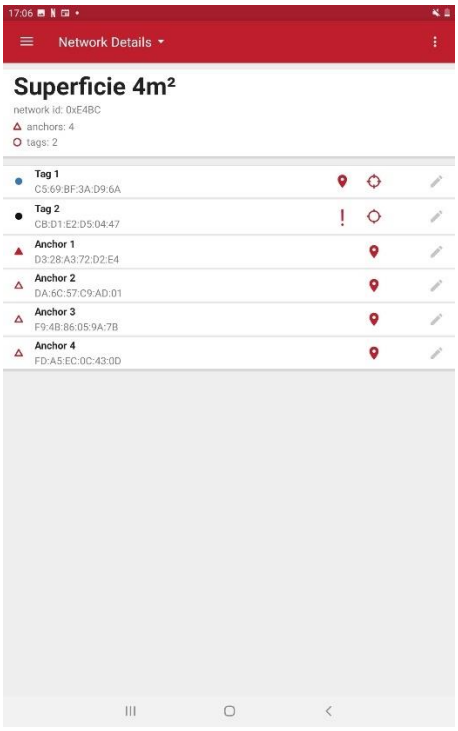


Figura 24: *Network* App DECAWAVE

Una vez introducidos todos los componentes de la red se podrían configurar, primero se debe configurar cada baliza según su rol en la red *anchor* o *tag.* Además de esto hay varios campos que se pueden configurar, como por ejemplo en el caso de los *tags* se puede cambiar la frecuencia de actualización de posición, con lo que se obtienen más o menos puntos. Para el caso de los *anchors,* la configuración es más crítica ya que esto definirá los puntos de referencia para la red, es decir cuanto más exacto sea el posicionamiento de estas balizas en esta red más precisas serán las lecturas de la localización de los *tags.* Para esta configuración primero se debe definir qué *anchor*

es el *initiator*, es decir el punto inicial de referencia (*x = 0, y = 0*), en esta red corresponde al *anchor 1* (Figura 25). A partir de aquí la aplicación da dos métodos para la posición relativa de las demás balizas, por un lado, se puede obtener con un método propio de la aplicación llamado *autopositioning*, o se puede ir introduciendo las coordenadas de cada baliza manualmente, este ha sido el método que se ha elegido para esta red en concreto ya que el otro no era lo suficientemente preciso tal y como recomienda la misma aplicación.

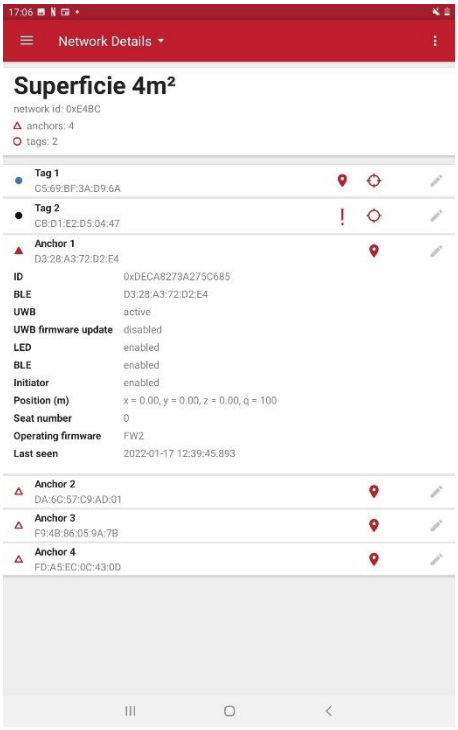


Figura 25: *Initiator* de la red y valores de configuración

Una vez que la red está configurada, se puede hacer uso de ella. En la aplicación se puede ver en tiempo real la posición de los *tags* junto con los *anchors* (Figura 26) así como la distancia entre ellos en un mapa generado por la aplicación al cual se le puede introducir una imagen de fondo con un mapa previamente diseñado.

Una vez hecha la configuración de la red y habiendo comprobado que funciona correctamente, ya se puede hacer uso de ella. Para poder emplear los datos obtenidos por este sistema de posicionamiento, lo primero que se debe de hacer es configurar un *tag* como elemento pasivo del sistema, este actuará como *listener*, es decir, estará conectado al PC o a la Raspberry a través del puerto USB y este recibirá los datos de posición del *tag* mediante BLE, con esto se consigue leer los datos toamados por el *tag* a través de este *listener* para poder usarlos como se desee. Tras esto se ha creado el programa `DWM.py` para poder leer los datos y almacenarlos en un archivo de texto, esto solo se ha realizado para esta prueba, para las demás se procede de otras formas.

En la primera parte del programa `DWM.py`, se importan todas las librerías que son necesarias para la ejecución de este, después de esto se define tanto el nombre del archivo y el nombre del puerto serie al que se conectará el programa (`/dev/ttyACM0`) junto con el valor del *Baudrate* (115200) después de esto se envían ciertos comandos por el puerto serie para poder recibir la información de las balizas, todo esto es gracias a la librería `serial` de Python3. Una vez que se reciben los datos, se deben tratar para obtener aquellos que se desean, en este caso, primero se separa la información por líneas y tras esto cada elemento de la línea se para por comas y se crea un vector

de donde se obtendrá la posición en (*x*, *y)* de la baliza en cuestión (para esta aplicación, el parámetro *z* es irrelevante, pero se podría capturar también). Cada vez que se recibe un dato de posición, este se copia en un archivo `.txt` que previamente se ha definido para tener un registro de estos valores. Por último, se crea una línea de control para cerrar la comunicación serie, el archivo y el programa en sí de forma correcta, esto se realiza con la variable `stop` que está configurada para cerrar el programa cuando el parámetro `y1` sea mayor a dos metros, sin embargo, este se puede configurar de varias formas, por ejemplo, para cerrar el programa cuando se hayan obtenido número determinado de datos, para esto sirve la variable `cont`.

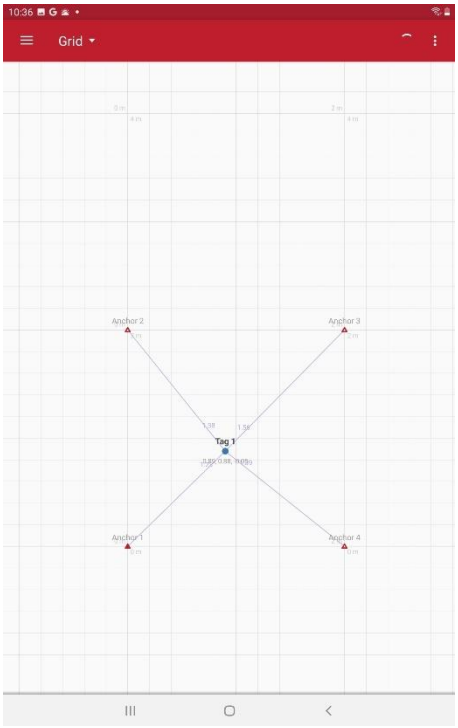


Figura 26: Mapeo de las balizas DECAWAVE

```
#DWM Serial Parser by Brian H. | Updated 3/6/19
import serial
import time
import datetime
import numpy as np
import math
import matplotlib.pyplot as plt
import os
import sys

cont = 0
stop = 2.0

file =
open("/home/guille/Escritorio/Pruebas_DWM/Dinamico/Superficie_2m2_dina
mico_1.txt","w")
DWM = serial.Serial(port="/dev/ttyACM0", baudrate=115200)
print("Connected to " +DWM.name)
DWM.write("\r\r".encode())
time.sleep(1)
DWM.write("lec\r".encode())
time.sleep(1)
x = []
y = []
```

```
while True:
    try:
        line=DWM.readline()
        if(line):
            parse=line.decode().split(",")
            if parse[0] == 'POS' and parse[1] == '0'and len(parse)> 4:
                x1 = float(parse[3])
                y1 = float(parse[4])
                file.write("{:.2f},{:.2f}".format(x1,y1))
                file.write(os.linesep)
                print(x1,y1)
                cont = cont + 1

                if y1 > stop :

                    DWM.write("\r".encode())
                    DWM.close()
                    file.close()
                    sys.exit()

        else:
            print("Distance not calculated : ",line.decode())
    except Exception as ex:
        print(ex)
        break
```

DWM.py

Ya que se ha programado el método de adquisición de datos, se puede comenzar a realizar pruebas con este sistema de localización. Se han colocado un total de cuatro *anchors* formando un cuadrado de 2x2m en el suelo de un terrado de una casa de planta baja (Figura 27). Aunque muchas de las pruebas se han realizado en una habitación, debido a diversas situaciones causadas por la COVID-19, en la fase final del proyecto no se ha podido disponer de ella por lo que los ensayos se han realizado en este lugar.

Figura 27: Montaje de sistema de localización

Una vez que se ha montado el sistema de localización, se procederá con dos tipos de pruebas diferentes para comprobar el funcionamiento y precisión de este. Primero se hará una prueba de forma estática colocando el tag_1 en la posición ($x = 1, y = 1$), todo esto medido con un metro en el suelo real. Tras esto, se ha realizado una prueba con la baliza en movimiento sobre una línea

definida y medida previamente que va desde la posición (*x = 1, y = 0*) hasta (*x = 1, y = 2*) este movimiento se ha realizado atando la baliza a una cuerda y tirando de esta para moverla por encima de la línea (Figura 28) con una velocidad aproximada de 0,2 *m/s*.

Para analizar y tratar los datos obtenidos con estas pruebas se ha utilizado el programa `Análisis_Datos_DWM.py`.

Este programa se divide en varias partes, una de las más importantes y críticas es la de aplicar un filtro a los datos procedentes de las balizas. Este filtro en síntesis coge un número previamente definido de datos los suma y hace la media de estos, con esto se consigue normalizar la gran cantidad de datos aportados por el sistema, este método se basa en el parámetro `nmed` que determinará el número de puntos que se sumarán para hacer la media.

Figura 28: Sistema de movimiento de baliza

A parte de esto se han creado dos funciones para por un lado crear la trayectoria que se ha seguido en el experimento llamada *recta()* y por otro lado la creación de un punto de referencia para la prueba en estático llamada *punto()*. Todos estos datos se guardan en las variables (`xt, yt`) que son dos variables de tipo lista. Por último, se calcula un error promedio con una función para poder comparar los diferentes resultados. Este error se calcula como la sumatoria de todos los errores absolutos dividiendo por el número de puntos tomados. Para que sea un parámetro normalizado la ecuación se obtendría la ec.(20):

$$e_p = \frac{\sum_{i=1}^{N} e_a}{N} \qquad (20)$$

donde e_a es el error absoluto de cada punto que se calcula como la distancia mínima a la trayectoria o punto de referencia y N es el número de puntos tomados. Además de esto se genera una gráfica con todos los datos recopilados para que pueda ser posteriormente visualizada y analizada.

A continuación, se mostrarán los datos obtenidos en las dos pruebas que se han realizado para cada una de las referencias y tras esto se hará una gráfica en Matlab para comparar los resultados obtenidos.

En la Figura 29 se muestran los resultados obtenidos en la prueba realizada en estático con los diferentes análisis variando el parámetro `nmed`.

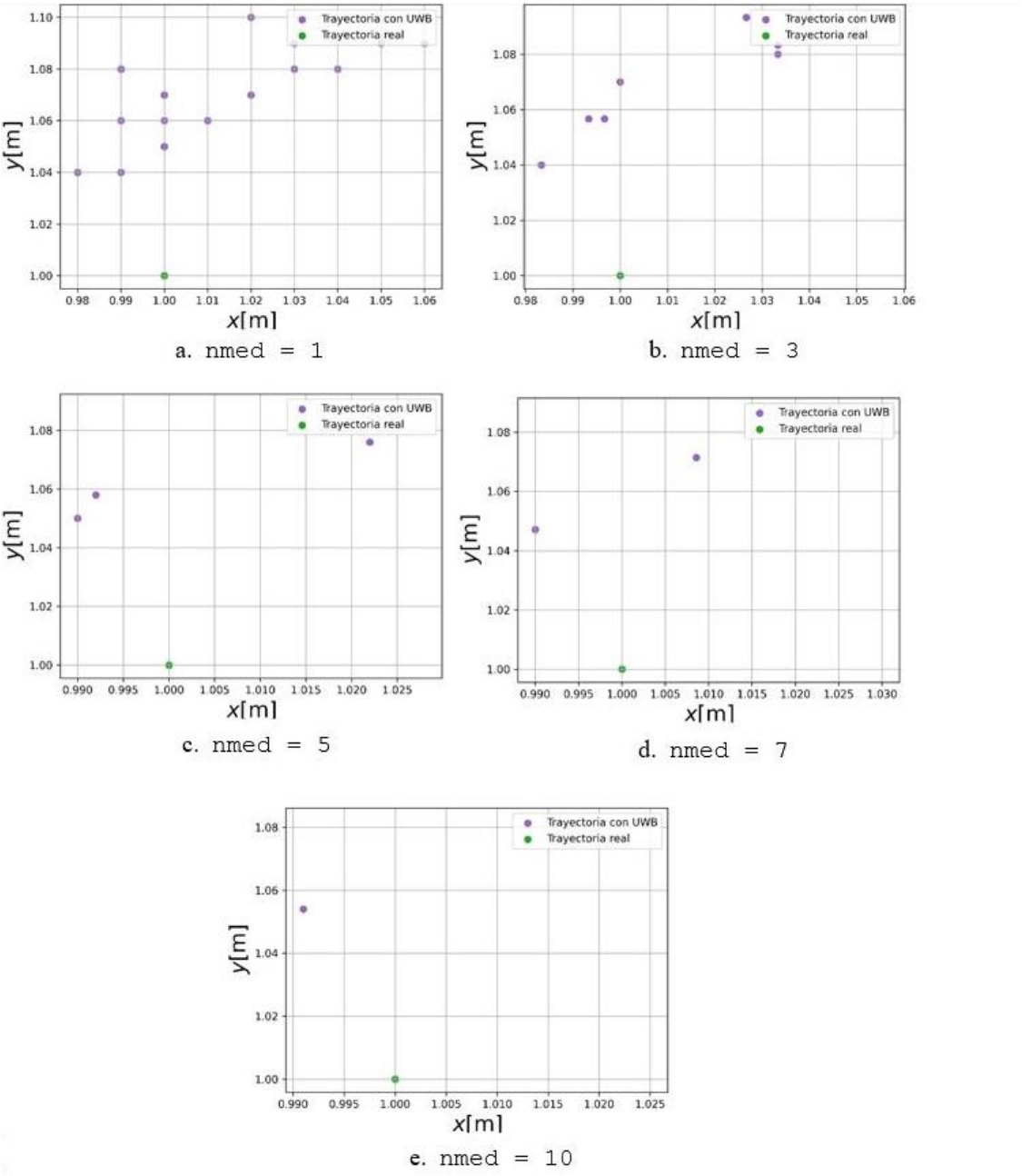


Figura 29: Resultados obtenidos en la prueba en estático en función del valor de `nmed`

Se puede observar cómo el número de puntos obtenidos va decreciendo conforme aumenta el valor de `nmed`, esto se debe a que cuantos más puntos se filtren haciendo la media, menos puntos finales se obtendrán. También se puede ver como el error máximo cometido es de unos pocos centímetros, lo cual está bastante bien, aunque todo esto se verá después analizando las gráficas de los errores promedio realizadas con Matlab.

En la Figura 30 se observa cómo se comporta el error promedio calculado en función del parámetro `nmed`.

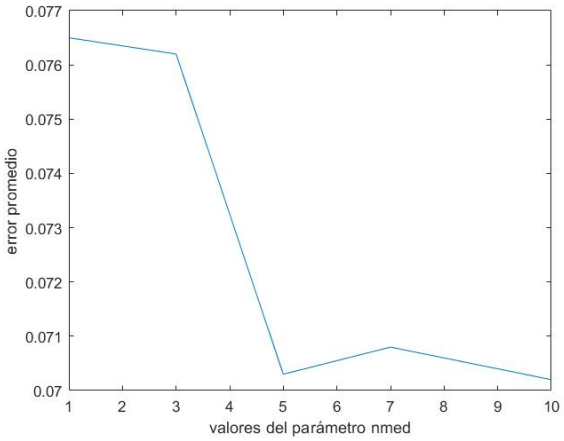


Figura 30: Error promedio en función de `nmed` para las pruebas en estático

Se ve como a medida que aumenta el valor de `nmed`, disminuye el error, como se observará en las demás pruebas.

Ahora se analizarán los resultados obtenidos en la prueba dinámica, los cuales se pueden ver en la Figura 31.

Se observa como al igual que en la prueba en estático, el error es de pocos centímetros y que conforme aumenta `nmed` disminuye el número de puntos. En la Figura 32 se verán los resultados obtenidos en la segunda prueba. En esta figura se aprecian resultados muy parecidos a la primera prueba asique En la Figura 33 se analizarán los errores promedio obtenidos con graficas en Matlab.

Se observa como los resultados obtenidos para estas pruebas son mejores y más estables en comparación con los de la prueba en estático, esto quiere decir que el sistema de localización se comporta de manera aceptable cuando a este se le aplica un movimiento, cabe mencionar que el error promedio en estas pruebas está en torno a cuatro centímetros.

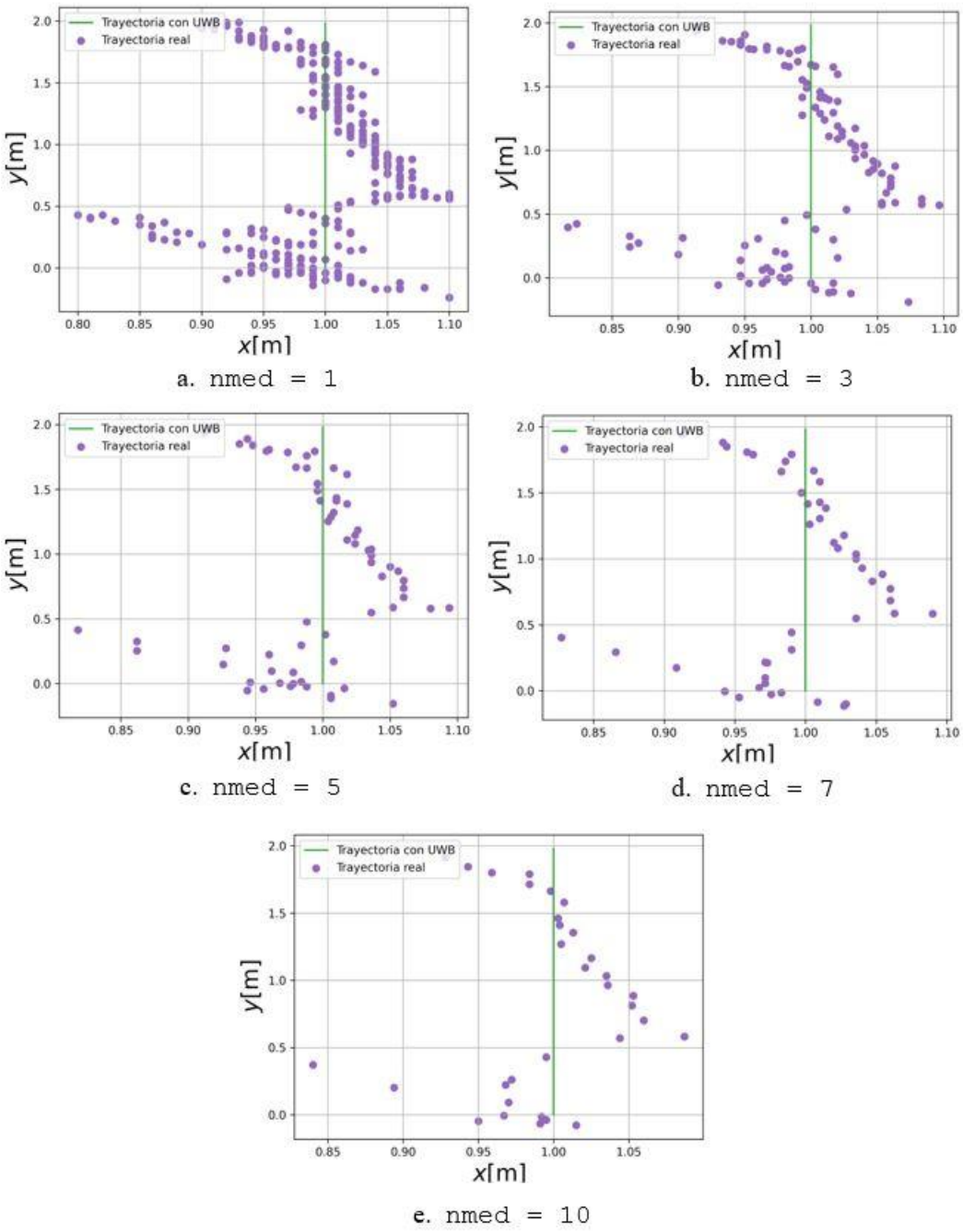


Figura 31: Resultados obtenidos en la prueba 1 en dinámico en función del valor de `nmed`

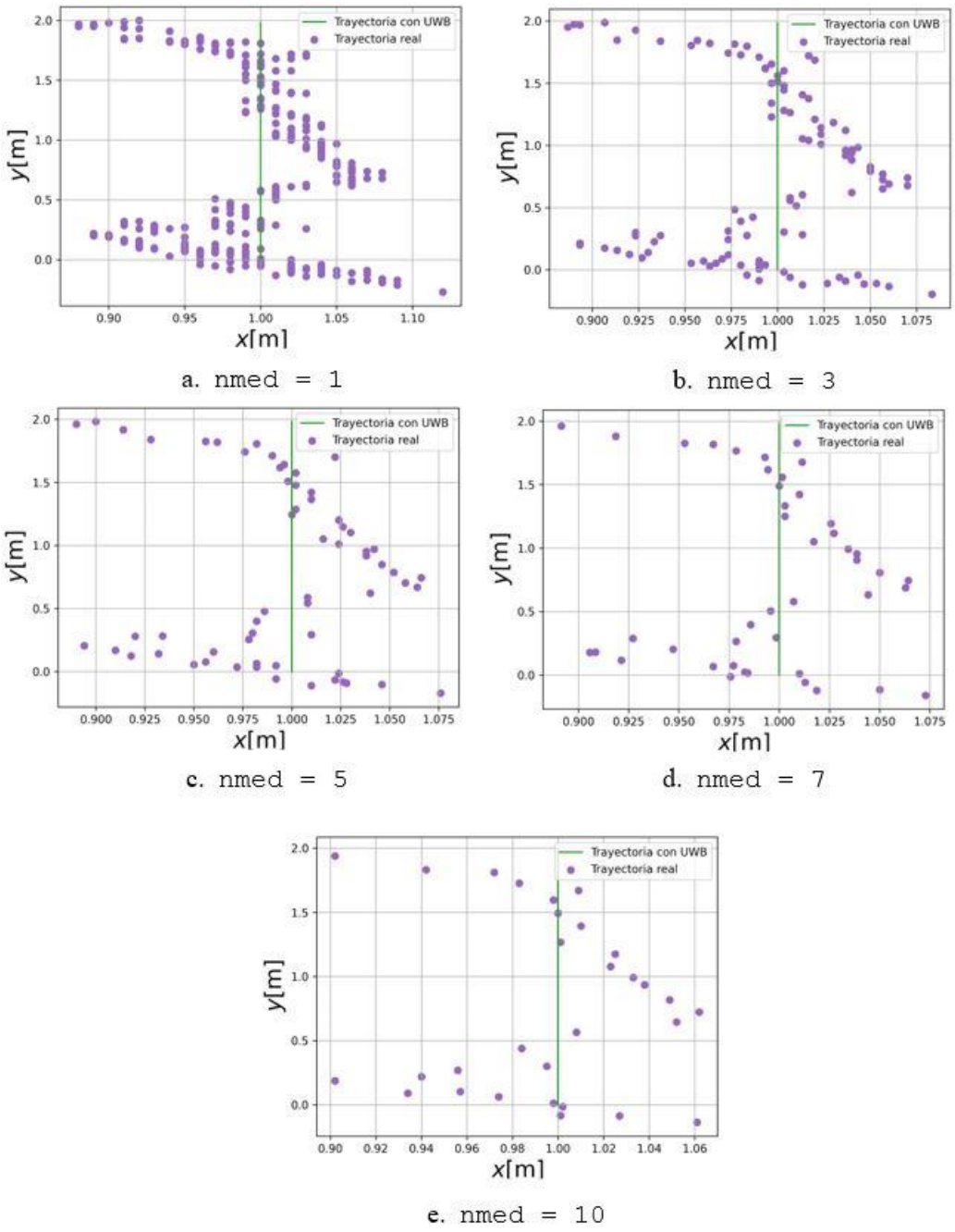


Figura 32: Resultados obtenidos en la prueba 2 en dinámico en función del valor de `nmed`

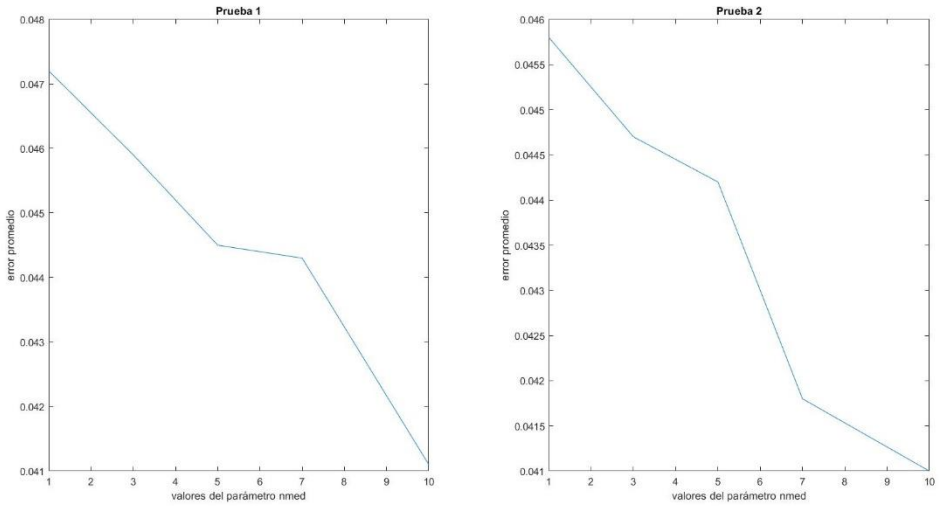


Figura 33: Error promedio en función de `nmed` para las pruebas en estático

4.2. Simulación del algoritmo *Pure Pursuit* en Matlab

Siguiendo los pasos definidos en el apartado 3.7.1, se ha construido un script en Matlab para previamente a las pruebas con el sistema robótico real realizar alguna simulación del método de seguimiento de trayectorias *Pure Pursuit*, el archivo en el que se encuentra este programa es `pure_pursuit.m` (ver anexo).

Con este script lo que se simula principalmente es un robot que parte de una posición y orientación inicial con una velocidad lineal fijada y el cual mediante el uso del *Pure Pursuit* se va calculando la velocidad angular necesaria para seguir la trayectoria.

Lo que se quiere comprobar también es cómo influye el parámetro *d* dentro de este método del cual se habló en capítulos anteriores. Se han realizado simulaciones tanto para una trayectoria completamente recta como para una que describe una senoide, se irán comparando las dos simultáneamente. En las siguientes gráficas, la línea roja corresponde a la trayectoria a seguir y las circunferencias azules a cada uno de los puntos alcanzados por el robot simulado, los ejes x e y corresponden a un sistema de referencia arbitrario creado para esta simulación.

Primeramente, se muestra una simulación para ambas trayectorias con un valor para el parámetro *d* pequeño (Figura 34) En este caso el valor de este parámetro es tres, esto lo único que significa es que el punto que se tomará como objetivo estará tres puntos por delante del punto más cercano al robot.

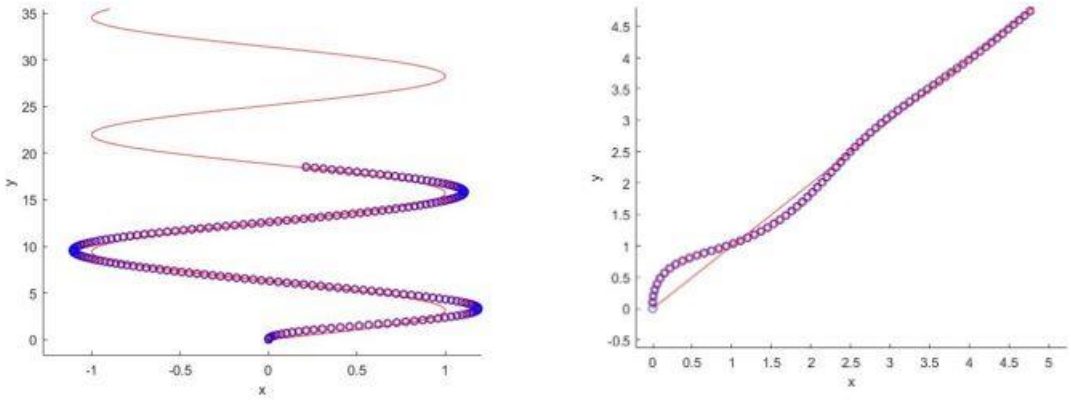


Figura 34: Seguimiento de trayectorias con d =3

Como se puede ver sobre todo en la trayectoria recta, antes de alcanzar el estado estacionario hace un par de oscilaciones, esto se debe a que como el punto objetivo está cerca del robot el giro de este debe de ser brusco para alcanzarlo.

A continuación, se realiza una simulación con un valor para la d igual a diez (Figura 35).

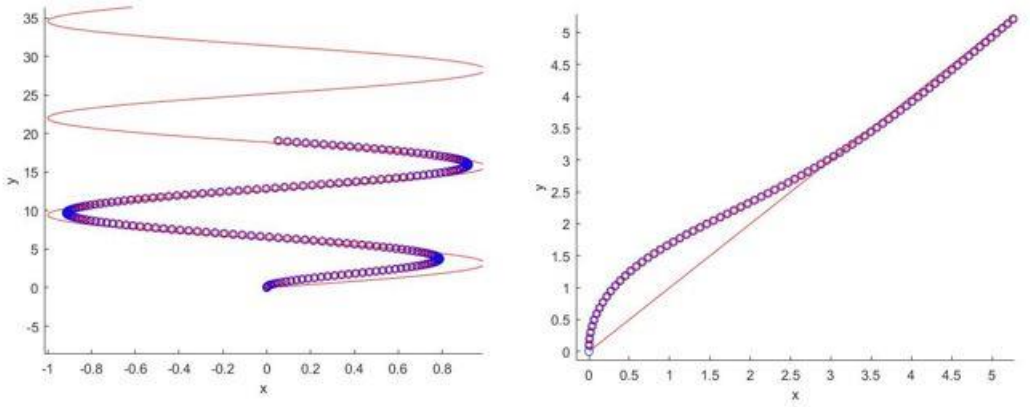


Figura 35: Seguimiento de trayectorias con d =10

Como se puede apreciar, ahora, en el caso de la trayectoria recta, hay menos oscilaciones y se tarda un poco menos en alcanzar el estado estacionario y como el punto objetivo se encuentra más lejos, la curva es más atenuada.

Por último, se hará una simulación con d igual a veinte para comprobar el efecto de un valor mayor de este parámetro (Figura 36).

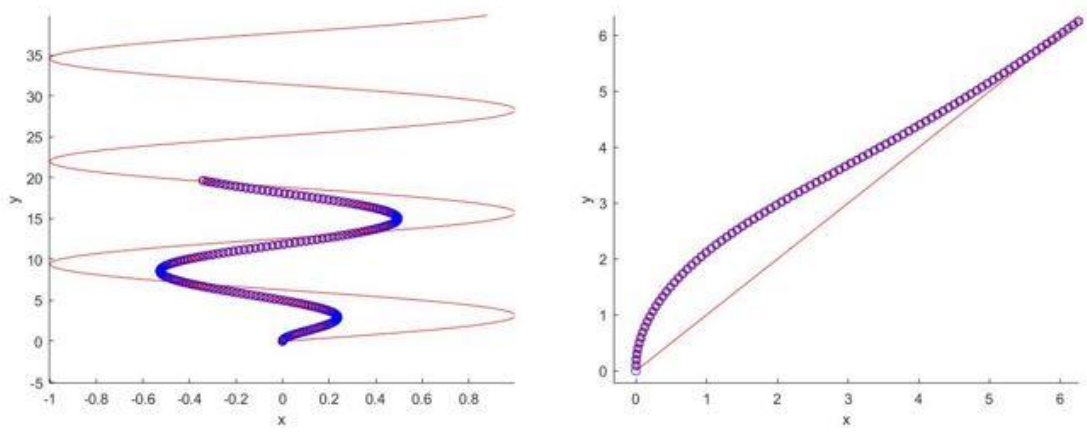


Figura 36: Seguimiento de trayectorias con d =10

Ahora se puede apreciar como por un lado en la trayectoria senoidal no se consigue un buen seguimiento y, por otro lado, en la trayectoria de forma recta se tarda demasiado en alcanzar el estado estacionario, ambos comportamientos se deben a que el punto objetivo se encuentra demasiado lejos del robot y por lo tanto el método comienza a fallar.

Como conclusión, se podría decir que la elección de este parámetro es crucial para el buen funcionamiento del sistema no debiendo ser ni demasiado pequeño ni demasiado grande.

4.3. Pruebas de navegación con odometría

En este apartado se implementará el algoritmo de seguimiento de trayectorias en el robot haciendo uso de los datos aportados por sus propios sensores y realizando la comunicación con ROS a través de la Raspberry Pi.

Antes de comenzar con las pruebas se debe de hablar del sistema de referencia que emplea el robot (Figura 37) para situarse en el plano, ya que para poder aplicar el algoritmo de seguimiento *Pure Pursuit* las coordenadas deben estar en el mismo sistema de referencia.

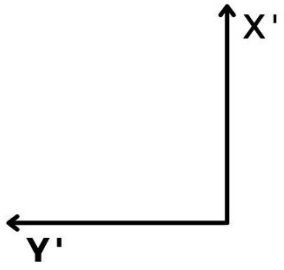


Figura 37: Sistema de referencia del robot

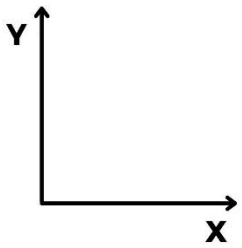


Figura 38: Sistema de referencia general

Puesto que el sistema de referencia del robot (*OX'Y'*) está girado 90º con respecto al sistema de referencia que se está usando (*OXY)* (Figura 38*),* para crear y seguir las trayectorias, se debe de generar una matriz de cambio para adecuar las coordenadas dadas por el robot a las del sistema de referencia empleado de tal modo que:

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} = R \begin{bmatrix} P_{x\prime} \\ P_{y\prime} \end{bmatrix} \tag{21}$$

Donde *P* son cada una de las coordenadas en sus sistemas de referencia y R es la matriz de cambio que se calcularía con el ángulo de giro θ que sería 90º como ya se ha dicho:

$$R = \begin{bmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \cos(90) & -sen(90) \\ sen(90) & \cos(90) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{22}$$

Con lo que obtendríamos:

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} = R \begin{bmatrix} P_{x\prime} \\ P_{y\prime} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} P_{x\prime} \\ P_{y\prime} \end{bmatrix} = \begin{bmatrix} -P_{y\prime} \\ P_{x\prime} \end{bmatrix} \tag{23}$$

Por otro lado, la información que da el robot de su orientación es en dos intervalos, uno negativo en sentido horario y otro positivo en sentido antihorario, ambos de 0 a π radianes. Para que el algoritmo de seguimiento funcione correctamente se ha tenido que invertir el signo de la orientación dada por los sensores del robot.

Una vez que se ha explicado esto, cabe mencionar que se ha empleado el paquete `create_robot` que es un driver basado en la librería `libcreate` programada en C++ y que se usa para controlar los iCreate 1 y 2. Esta en síntesis son un grupo de cuatro paquetes que facilitan mucho la comunicación con el robot [19]:

- `ca_description`: modelo URDF y mallas 3D para ambos robots.
- `ca_driver`: controlador ROS para los robots.
- `ca_msgs`: mensajes para el controlador.
- `ca_tools`: herramientas para el trabajo con los robots.

Los principales *Topics* que se han empleado en este proyecto son:

- `odom`: a través del uso de los encoders da los datos de posición y orientación actuales. Que emplea un tipo de mensaje `nav_msgs/Odometry`.

- `cmd_vel`: con este *Topic* es posible enviar consignas de velocidad tanto angular como lineal al robot para que este las siga. Que emplea un tipo de mensaje `geometry_msgs/Twist`.

Para la ejecución de este sistema, lo primero que hay que hacer es ejecutar el comando `roscore` en un terminal de la Raspberry Pi, con esto ejecutamos un master de ROS que será el que controlará todas las partes del sistema, después, en otra terminal se ejecutará utilizando el comando `roslaunch` el archivo `create_2.launch` que se encuentra en el paquete `create_bringup`. Esto servirá para ejecutar el controlador del robot del que se acaba de hablar, cuando esto se realice ya se tendrá comunicación con el robot y se podrán ejecutar los nodos que se deseen, en este caso se ejecutará el nodo `pure_pursuit_1.py` que se encuentra en el paquete `pure_pursuit` para llevar a cabo las pruebas en cuestión.

En este nodo, lo primero que se hace es definirlo como suscriptor del *Topic* `Odom` y publicador en el `cmd_vel` para poder recibir los datos de la posción y orientación y enviar las consignas de velocidad, tras esto se definen los tipos de trayectorias arbitrarias que se han empleado con las diferentes funciones `recta()`, `curva()` etc. Por último, se introduce el algoritmo *Pure Pursuit* anteriormente explicado, el cual se va continuamente ejecutando a cada paso del robot, se coloca una señal de control llamada `goal` para saber que el robot ha alcanzado el fin de la trayectoria con un error de ±5cm.

```
#!/usr/bin/env python
import rospy
import numpy as np
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import *
import matplotlib.pyplot as plt
import os

#Datos de inicializacion
file =
open("/home/guille/Escritorio/Pruebas_PP_sin_DWM/Curva/Vel_0.05_d_3.tx
t","w")
d = 3
r = 0.23 #23cm distancia entres ruedas
b = 0.035 #3.5cm radio de las ruedas
e = 0.05 #error
vel = Twist() #Velocidad para publicar en el Robot
vel.linear.x = 0.05#Velocidad lineal constante
vel.angular.z = 0 #Velocidad angular inicial
goal = False #Bit de senalizacion de final de trayectoria
#trayectoria
xt = [0.0]
yt = [0.0]
#Objetivo
xob = 0
yob = 0
#Posicion y orientacion actual
```

```
yaw = 0
pos = [0.0,0.0]


pub = rospy.Publisher('cmd_vel', Twist, queue_size=1) #con esto
publicaremos los mensajes de la velocidad

def begin(): #Inicializamos el nodo y definimos como subscriber del
topic /odom

    rospy.init_node('pure_pusuit_1', anonymous=True)
    rospy.Rate(5)
    rospy.Subscriber('odom',Odometry,Position)


#Funcion para la obtencion de la posicion y orientacion a traves del
topic /odom

def Position(odom_data):
    global yaw
    global pos
    global xs,ys

    curr_time = odom_data.header.stamp
    orientation = odom_data.pose.pose.orientation # orientation x y z
w
    quaternion = [
        odom_data.pose.pose.orientation.x,
        odom_data.pose.pose.orientation.y,
        odom_data.pose.pose.orientation.z,
        odom_data.pose.pose.orientation.w]
    euler = tf.transformations.euler_from_quaternion(quaternion)
    pos = [odom_data.pose.pose.position.y * (-1),
           odom_data.pose.pose.position.x]

    yaw = (-1)*euler[2]

    file.write("{:.2f},{:.2f}".format(pos[0],pos[1]))
    file.write(os.linesep)

#Definicion de la trayectoria (recta)

def Trayectoria(x0,y0,xf,yf):

    global xt,yt

    xt = [x0]
    yt = [y0]
    pdt = (y0-yf)/(x0-xf)
    org = y0-pdt*x0
    paso= 0.01
    pasofloat = float(paso)
    rng = int((xf-x0)/pasofloat)
    npuntos = list(range(rng))
    del npuntos[0]

    for i in npuntos:
        xt.append(xt[i-1]+pasofloat)
        num = pdt*xt[i] + org
        yt.append(num)
```

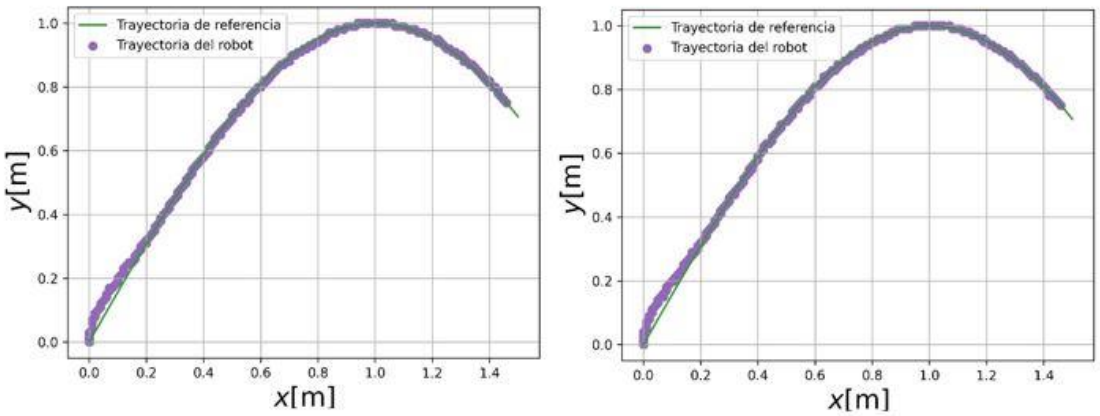
```
        xt.append(xf)
        yt.append(yf)

    #Definicion de la trayectoria (curva)

    def curva():
        global xt, yt

        for i in range(1,151):
            xt.append(float(xt[i-1] +0.01))
            yt.append(math.sin(xt[i]*(3.1416/2.0)))


    #Definicion de la trayectoria (senoidal)
    def senoide():
        global xt, yt

        for i in range(1,501):
            xt.append(math.sin(float(i)/75)*0.75)
            yt.append((float(i)-1)/175)



    #Obtencion del pto objetivo

    def objetivo():

        global pos,yaw,xob,yob,d

        Lmin = 10000.0
        n = 0

        for i in range(len(xt)):

            Lk = math.sqrt(pow((xt[i] - pos[0]),2) + pow((yt[i] -
    pos[1]),2))
            if Lk<Lmin:
                Lmin = Lk
                n = i
        if (n+d) < len(xt):
            xob = xt[n + d]
            yob = yt[n + d]
        else:
            xob = xt[-1]
            yob = yt[-1]


    #Calculo de la velocidad angular para alcanzar el pto objetivo

    def calculo_vel():
        global xob,yob,pos,vel,r,b

        L = math.sqrt(pow((xob - pos[0]),2) + pow((yob - pos[1]),2))
        Dx = (xob-pos[0])*math.cos(yaw) - (yob - pos[1]) *math.sin(yaw)
        if L == 0:
            g = 0;
        else:
            g = -2*Dx/(pow(L,2))
        #rr = 1/g #radio de curvatura
        wi = (vel.linear.x/r)*(1-(b/2)*g)
```

```
        wd = (vel.linear.x/r)*(1+(b/2)*g)
        vel.angular.z = (r/b)*(wd-wi)



if __name__ == "__main__":


        try:
            begin()
            #Trayectoria(0.0,0.0,2.0,2.0)
            curva()
            #senoide()
            while goal == False:
                objetivo()
                calculo_vel()
                pub.publish(vel)
                if (((pos[0] > (xt[-1]-e)) and (pos[0] < (xt[-1]+e)))
and ((pos[1] > (yt[-1]-e)) and (pos[1] < (yt[-1]+e)))):
                        goal = True


            print('Fin del recorrido')
            file.close()
            sys.exit()

        except rospy.ROSInterruptException as e:
            print(e)
            sys.exit()
        except KeyboardInterrupt:
            #file.close()
            sys.exit()
```

pure_pursuit_1.py

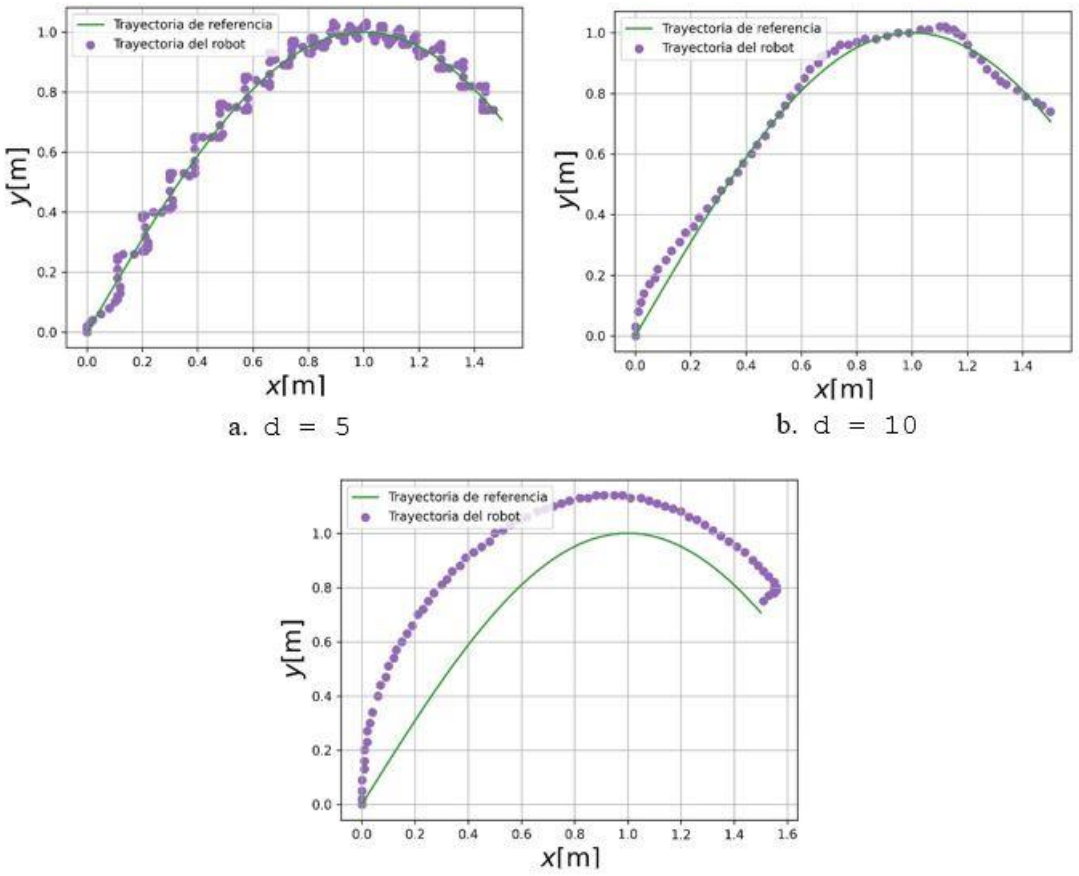
Con todo esto se han realizado una gran cantidad de pruebas, por lo que solo se mostrarán las que resultan más relevantes para su estudio, en concreto, se ha cogido una trayectoria curva que en principio es la que más problemas causaría, y se le han aplicado diversas velocidades lineales y variaciones del parámetro `d`. Primero se compraran los resultados obtenidos variando este parámetro a una velocidad baja (0.1 m/s) (Figura 39). Para analizar los datos obtenido en esta prueba se ha empleado el programa `Análisis_Datos_PP.py` que funciona de forma muy similar al que se vio en apartados anteriores.

Como se puede comprobar el seguimiento de la trayectoria objetivo es casi perfecto excepto al principio que al no estar orientado hacia el objetivo tiene que realizar una pequeña curva, pero los resultados son bastante buenos. Ahora se comprobará el comportamiento del sistema para la velocidad lineal máxima (0.5 m/s) y variando además el valor de `d` (Figura 40).



a. d = 5 b. d = 10

Figura 39: Resultados obtenidos con v = 0.1 m/s



a. d = 5 b. d = 10

c. d = 20

Figura 40: Resultados obtenidos con v = 0.5 m/s

En este experimente se puede observar que con esta velocidad alta el algoritmo sufre problemas, lo primero que se detecta es que con una d pequeña se generan muchas oscilaciones ya que el punto objetivo que se debe alcanzar está demasiado cerca del robot lo que produce giros demasiado bruscos, además para una d muy alta, como un valor de veinte, se nota como no se sigue la trayectoria en ningún momento ya que el punto objetivo se encentra demasiado lejos y la curva a describir es muy grande, por último señalar que con una d de valor intermedio el sistema no funciona mal del todo.

Para optimizar estos dos parámetros se han realizado varias pruebas variando ambos y se ha recogido el error promedio de cada experimento para después crear una gráfica en tres dimensiones con Matlab (Figura 41).

En la Figura 41, se coloca en el eje de la *X* los diferentes valores que se le dan a `d`, en el eje de la *Y* los valores de la velocidad lineal y en de la *Z* el error promedio cometido. Analizando la gráfica se observa que para valores altos de la velocidad y de `d` el error es muy alto, siendo menor cuando ambos disminuyen, el error mínimo se da en el de `d` = *5* y *v = 0.1* m/s, con un error de 0.0038cm. Sin embargo, se considera que esta velocidad es demasiado baja, por lo que para los siguientes experimentos se fijará una velocidad un poco más alta, de unos 0.15m/s.

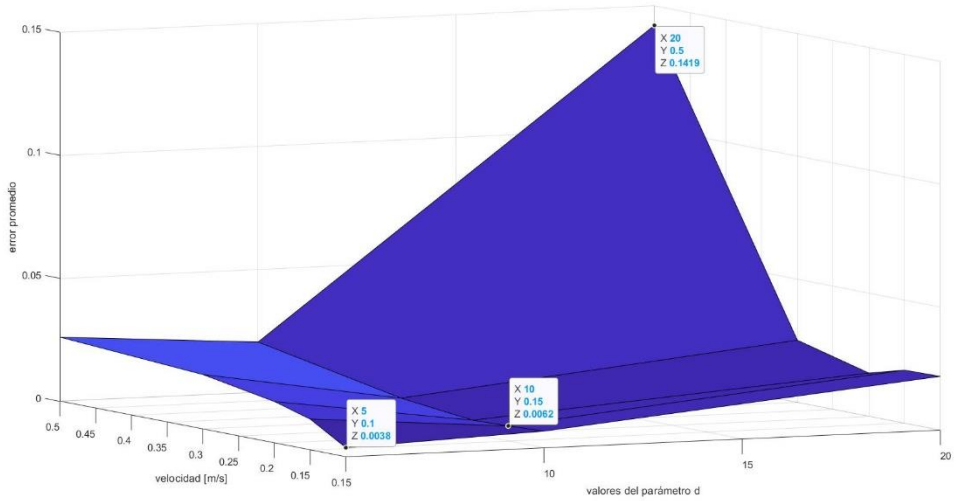


Figura 41: error medio en función de la velocidad lineal y el parámetro *d*

4.4 Pruebas de navegación con posicionamiento basado en balizas

En este apartado se juntará todo lo visto anteriormente para realizar pruebas juntando el sistema de posicionamiento con el robot para el seguimiento de trayectorias.

Primeramente, decir que para la aplicación del algoritmo en este apartado, se ha tenido que hacer uso del dato de la orientación proporcionado por el propio robot, se probaron varias soluciones como la de emplear dos balizas a la vez, una en la cabeza del robot y otra en la parte trasera para que mediante trigonometría se pudiera conocer la orientación, sin embargo todos los experimentos dieron malos resultados debido a la precisión del sistema de localización que aunque es bastante buena, no lo es lo suficiente para esta aplicación.

Al igual que en el apartado anterior, se ejecuta un ROS *Master* y el driver que controla el robot, tras esto se ejecuta el nodo `position_DWM.py` que publica en el *Topic* `Rposition`. Se ha creado un tipo nuevo de mensajes llamado `pos_1_2` que no deja de ser una estructura de datos que contiene datos de tipo `float64`. Este nodo está preparado para enviar datos de dos balizas diferentes, pero solo se emplea una de ellas.

Este nodo funciona de forma muy similar al que se vio en las pruebas con las balizas con la particularidad que publica sus datos en un *Topic*.

Una vez ejecutado este nodo, solo faltaría por ejecutar el nodo `pure_pursuit_dwm.py` para poder realizar las pruebas, los detalles del nodo pueden consultarse en el anexo ya que funciona de forma muy parecida al que se usó para realizar las pruebas con las balizas con la diferencia de que este nodo toma la posición actual del sistema de localización.

En este caso se han realizado pruebas variando el parámetro `d` y el `nmed`, este es el que determina el filtro que se le aplica a los datos de posición tomados, siendo la velocidad lineal siempre fija con un valor de 0.15 m/s. Para el tratamiento de estos datos se ha empleado el programa `Análisis_Datos_PP_DWM.py` que funciona de forma parecida a los demás programas de tratamiento de datos. A continuación, se muestran los resultados que se consideran relevantes en este experimento.

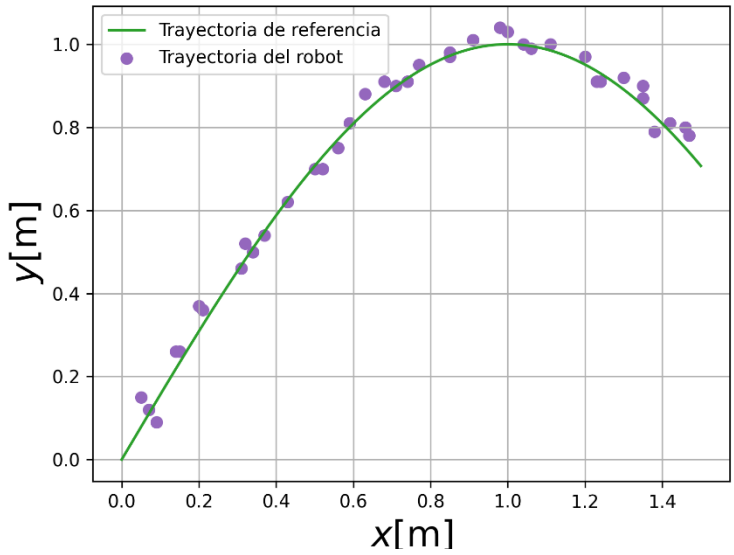


Figura 42: Resultado obtenido con `d`= 5 y `nmed` =3

En la Figura 42 se observa cómo con un valor bajo de `d` y de `nmed`, el número de puntos obtenido y el ajuste a la trayectoria son bastante buenos, ahora se aumentarán estos valores.

En la Figura 43, se ve cómo se reducen los puntos por el aumento del parámetro `nmed`, pero el ajuste sigue siendo bastante bueno, por último en la Figura 45, se ve como hay menos puntos también pero están todos cerca de la trayectoria, por lo que se puede decir que a esta velocidad el sistema trabaja bastante bien y se ve poco influenciado por las variaciones de estos parámetros, para comprobar esto se han recabado todos los datos del error promedio cometido y se han graficado en tres dimensiones con la variación de `d` y `nmed` (Figura44).

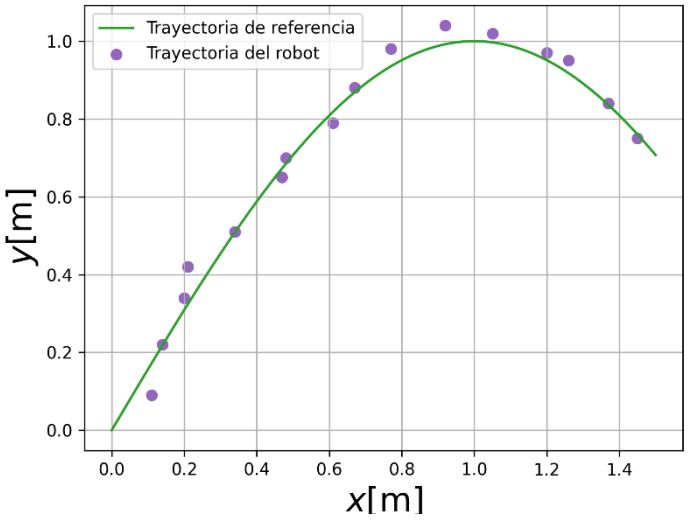


Figura 43: Resultado obtenido con `d`= 10 y `nmed` =7

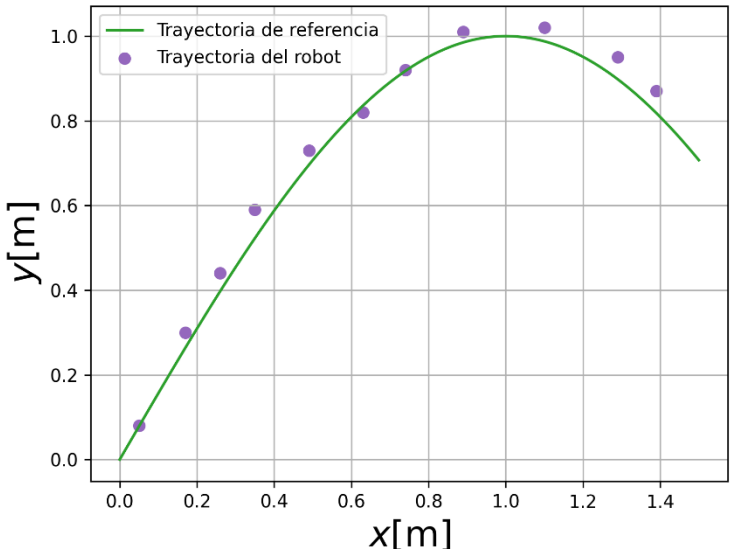


Figura 44: Resultado obtenido con `d`= 20 y `nmed` =10

En la Figura 45, se coloca en el eje *X* los valores de `nmed`, en el eje *Y* los valores de `d` y en el *Z* los valores del error cometido. Se puede ver como el error está entre dos y tres centímetros, estando el mínimo en `d` = *5*, *nmed* = *3* y el valor del error es de 0.0185cm. Esto demuestra que el conjunto total del sistema funciona de forma aceptable y valida el sistema de posicionamiento como bueno.

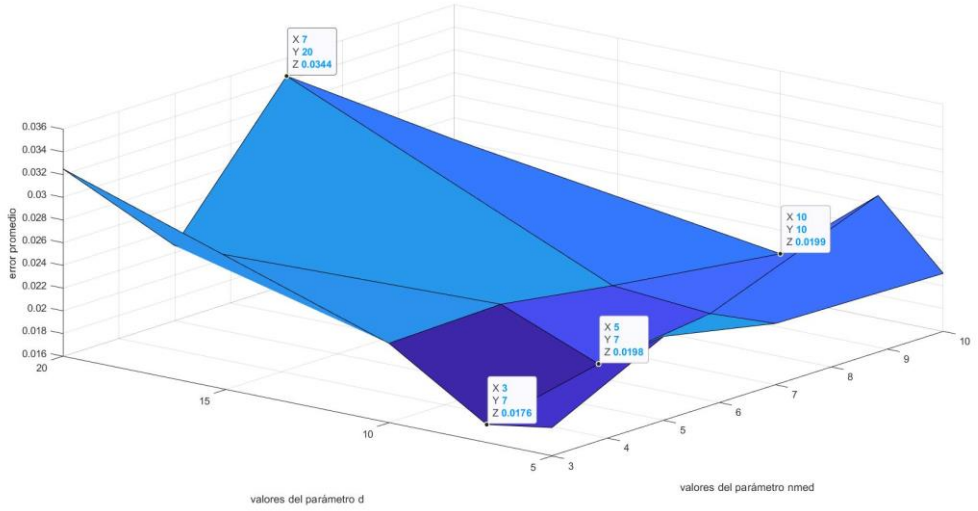


Figura 45: error medio en función del parámetro `d` y `nmed`

4.5 Pruebas en el invernadero

Para poner a prueba el sistema de posicionamiento, se han realizado ensayos en un invernadero de plantación de tomate Cherry situado en el municipio de Níjar, Almería. Con vistas a ser usado por la navegación en su interior de un robot móvil como el que se propone en el proyecto AGROBIOT [20], en este apartado se dará información acerca del tipo de invernadero en el que se ha trabajado, así como una exposición y análisis de los resultados obtenidos.

Tipo de invernadero.

El invernadero en el que se han tomado las muestras es del tipo multitunel, también llamado de tipo industrial. Su característica principal es la forma semicilíndrica de su cubierta y una estructura hecha de metal (Figura 46), este tipo de construcción se está expandiendo en el mundo de la agronomía almeriense debido a la capacidad de control de variables microclimáticas que proporciona.

Estos invernaderos se construyen con tubos de acero galvanizado (Figura 47), en su mayor parte de sección cilíndrica, con diámetros entre 25 y 60 mm y espesores de 1,5 a 3 mm. Los túneles tienen anchuras que varían entre los 6,5 y 9 m y la separación entre apoyos bajo las canales suelen ser de 4 o 5m. Gran cantidad de estos invernaderos se construyen con cerramiento lateral rígido de policarbonato ondulado, lo que aporta mayor resistencia al viento por la parte lateral y frontal de estos [21].

Figura 46: Invernadero tipo multitunel.

Figura 47: Estructura del invernadero

Las ventajas que aportan este tipo de construcción son [21]:

- La gran separación entre los apoyos facilita la entrada y salida de maquinaria agrícola.
- La elevada altura de estos invernaderos ayuda a la circulación del aire.
- Presentan una buena estanqueidad a la lluvia y al aire, esto da la posibilidad de usar métodos de control activo del clima, como la calefacción, paneles evaporadores o enriquecimiento carbónico.
- Permite la instalación de ventilación cenital situada a sotavento (que aumenta la tasa de ventilación) y facilita su accionamiento mecánico.
- La cubierta curva facilita el reparto de la luminosidad en el invernadero.
- Da la posibilidad de utilizar una parte de la estructura como zona de recepción o de almacén.

En contraposición, sus inconvenientes son [21]:

- El elevado coste de construcción, que ronda entre los 12 y 25 €/m^2.
- El plástico empleado requiere mayores exigencias mecánicas debido a que está poco sujeto a la estructura.

Pruebas realizadas con el sistema de posicionamiento

Para realizar este ensayo, se ha creado una nueva red en la aplicación de *DECAWAVE* llamada *DRTLS*, a la cual se le ha puesto el nombre de "Invernadero" (Figura 48), en esta red se incluirán diez *anchors* y dos *tags*, uno que funcionará como *listener* y otro que será del que obtengamos la posición.

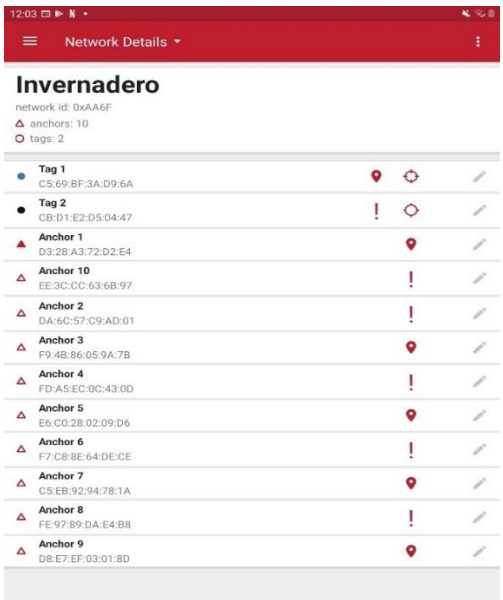


Figura 48: Red "Invernadero"

Para tener una buena sujeción de las balizas con una altura adecuada, se han colocado las balizas sobre los alambres que sujetan las plantas de los tomates atando estos dos elementos con bridas de plástico (Figura 49).

Figura 49: Sujeción de las balizas

Una vez decidida la localización de las balizas, se decide posicionar las dos filas de balizas paralelas con una distancia entre balizas a lo largo de la línea de unos dos metros (Figura 50), colocando un total de cinco balizas a cada lado. En este caso, todo el proceso de posicionar cada *anchor*, es decir, introducirle sus coordenadas, debe de realizarse a mano, ya que la opción de *autopositioning* que ofrece la aplicación solo permite crear redes cuadradas y no con muchas balizas en línea como en este caso.

Figura 50: Colocación de las balizas

Tras colocar las balizas se plantean tres tipos de disposiciones diferentes, que se conseguirán apagando y encendiendo ciertas balizas para conseguir la disposición deseada, se ha probado por una parte el empleo de tres configuraciones diferentes que son: Emplear todas las balizas tal y como están colocadas (Figura 51) , apagar las balizas con número par para conseguir una disposición en zig-zag (Figura 52) y apagar pares de balizas para que la distancia entre ellas sea de cuatro metro en lugar de dos (Figura 53). Por otro lado, se han realizado pruebas de movimiento tanto la línea en la que se ha colocado el sistema como a lo largo de la línea que se encuentra inmediatamente a la derecha para ver el comportamiento del sistema.

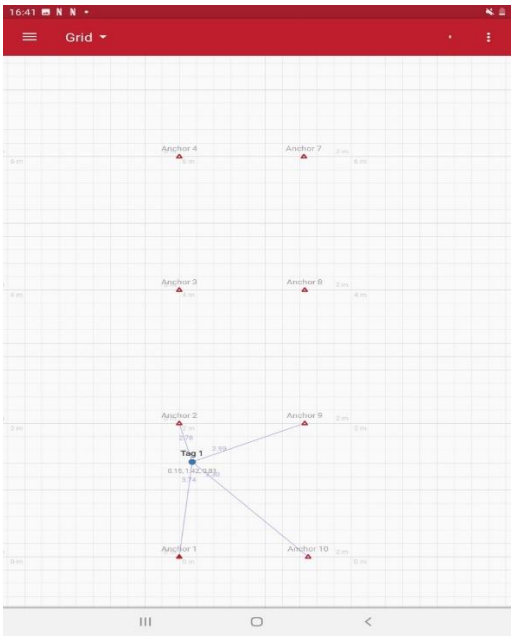


Figura 51: Disposición con todas las balizas

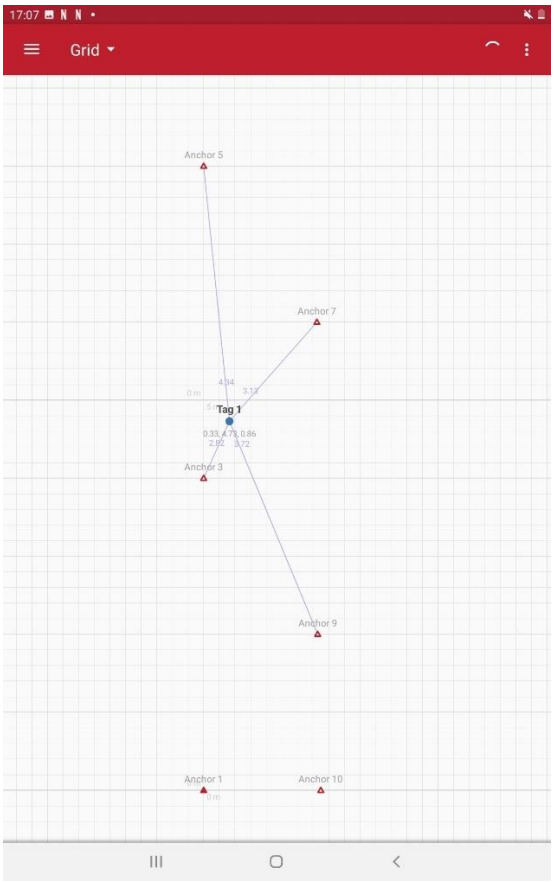


Figura 52: Disposición en zig-zag

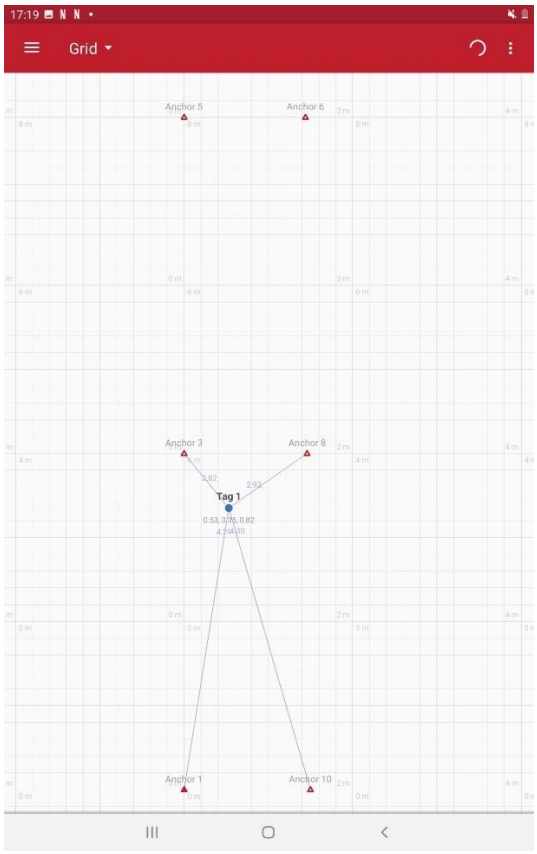


Figura 53: Disposición con una separación de cuatro metros entre balizas

Para realizar la adquisición y análisis de todos los datos se han empleado los mismos programas que en el aparatado 4.1. Sin embargo, en este caso, como referencia de trayectoria, se ha colocado una línea recta que va desde ($x = 0.75$, $y = 0$) hasta ($x = 0.75$, $y = 10$) para el caso de la línea central y otra desde ($x = 3.0$, $y = 0$) hasta ($x = 3.0$, $y = 10$) para la línea contigua a esta, estas coordenadas se toman midiendo la distancia desde la primera baliza hasta el centro de la línea. Tras tomar todas las medidas, se han procesado los datos para obtener el error promedio en cada experimento.

Primero se analizarán los datos tomados para la línea central en función de las diferentes disposiciones y el valor de `nmed` (Figura 54), a cada una de las disposiciones se le ha asignado un número del uno al tres siendo el uno la configuración con todas las balizas, el dos en zig-zag y el tres la configuración con una distancia de cuatro metros entre balizas.

Observando los resultados, se ve como la configuración en zig-zag es la que peor funciona de todas dando errores en torno a 30cm y cuando se colocan las balizas separándolas cuatro metros se obtiene los mejores resultados con errores de 12cm aproximadamente, esto lleva a pensar que el hecho de introducir muchas balizas como se ha hecho en la primera configuración, no asegura mejores resultados, esto se debe a los errores cometidos a la hora de introducir las coordenadas de cada una de las balizas manualmente, es decir, el error a la hora de posicionar el *anchor* influirá en la distancia que se mida entre este y el *tag*, y por lo tanto, cuantos más *anchors* haya en la red mayor será el error acumulado. También cabe mencionar que existen errores cometidos a la hora de obtener los datos ya que se han realizado caminando por el centro de la línea con el *tag* en la mano. También cabe mencionar que conforme aumenta el valor de `nmed` disminuye ligeramente el error.

Ahora se analizarán los resultados obtenidos para la línea contigua (Figura 55).

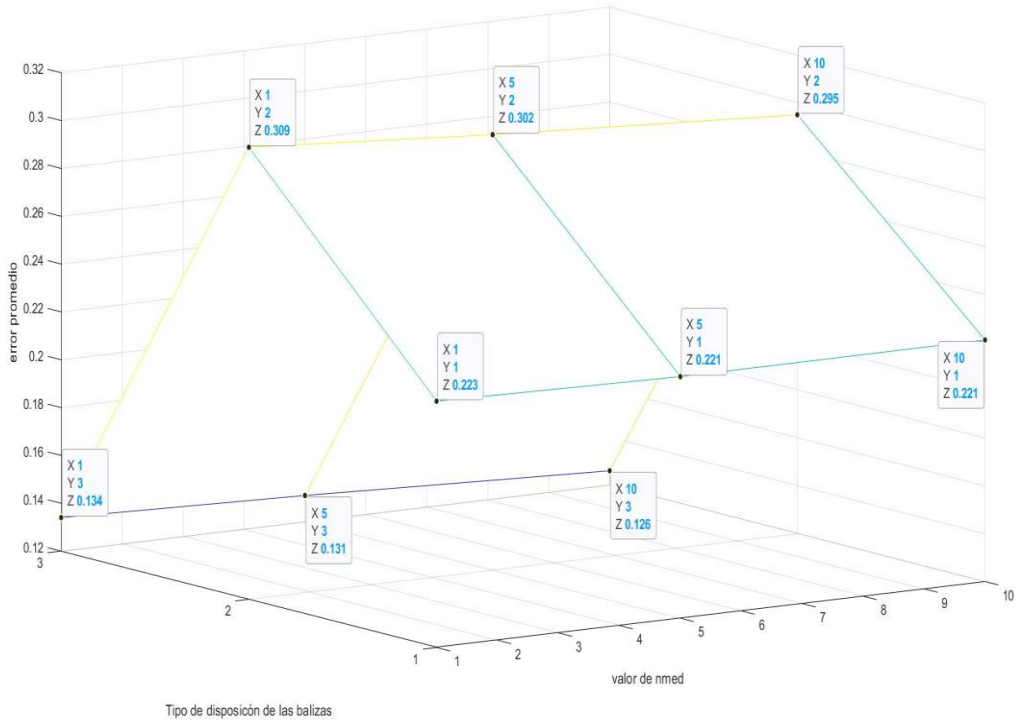


Figura 54: Error promedio en la línea central función de la disposición de las balizas y `nmed`

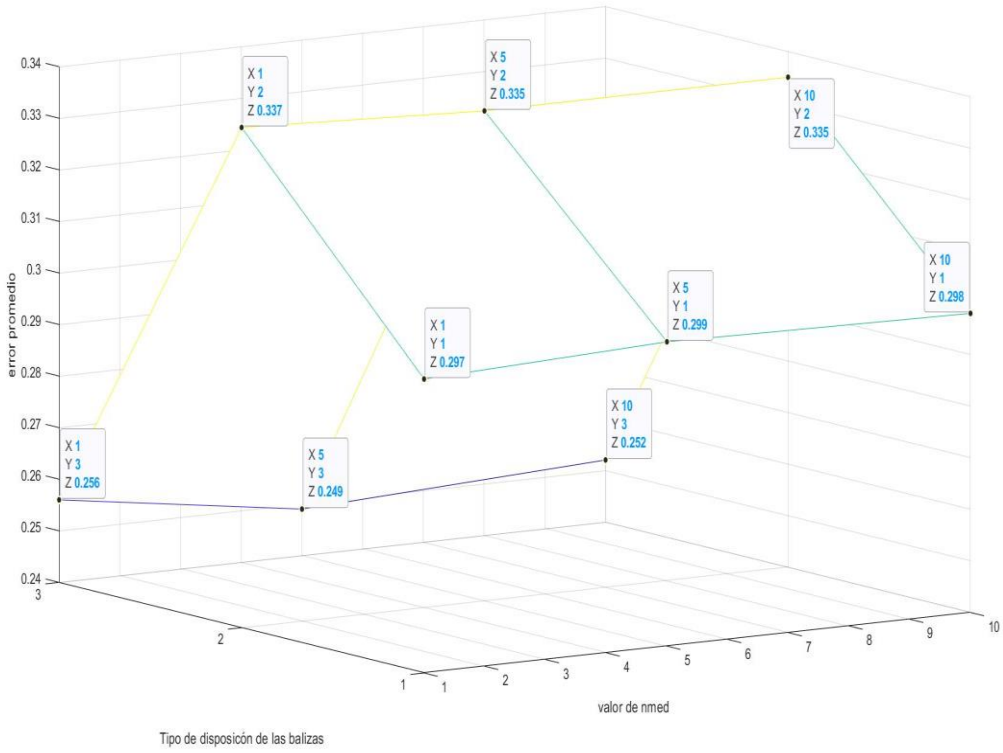


Figura 55: Error promedio en la línea contigua función de la disposición de las balizas y *`nmed`*

En el segundo experimento (posicionamiento en la línea contigua), al igual que en el anterior, los mejores resultados se han obtenido con la tercera disposición, teniendo errores en torno a los 25cm. Con esto se confirma que está es la mejor disposición a la hora de crear una red de este tipo con estas balizas. También se ha comprobado que en estos experimentos el valor del filtro de datos tiene poca influencia en el error cometido.

El último experimento que se realizó fue el de emplear un carro (Figura 56), que se utiliza para atar las plantas por la parte superior del invernadero, como objeto móvil que lleva el *tag* incorporado, el cual se colocó en el centro de este.

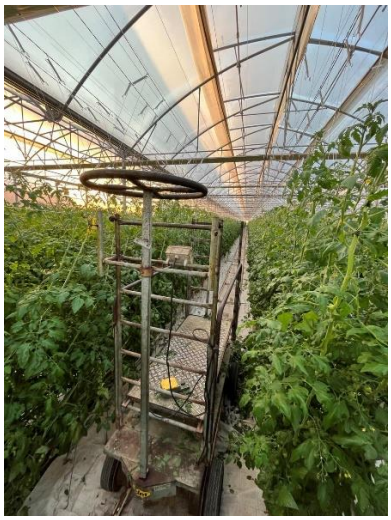


Figura 56: Carro de invernadero

Esta prueba se realizó solamente en la línea central ya que no se tuvo la posibilidad de trasladar este a la siguiente línea, los resultados obtenidos se muestran en Figura 57 fueron los siguientes.

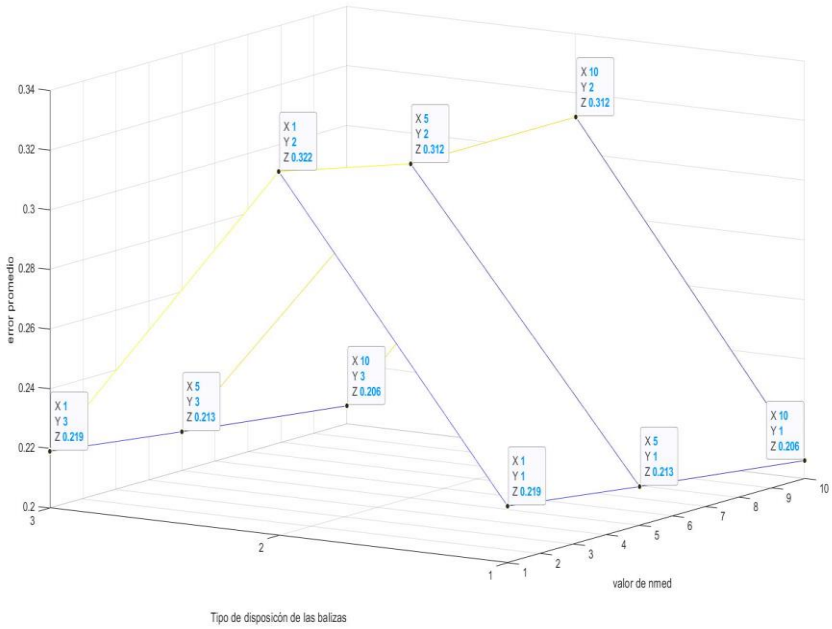


Figura 57: Error promedio cometido en la prueba del carro función de la disposición de las balizas y `nmed`

Se observa que los errores cometidos son ligeramente mayores que los que se cometieron cuando se tomaron los datos a pie y como curiosidad, se observa que esta vez, los resultados para las configuraciones uno y tres son muy similares. También cabe destacar que el carro está completamente construido con metal, esto genera grandes interferencias en las ondas, lo que justifica que los errores sean mayores que cuando las muestras se tomaron a pie.

Como conclusión, a grandes rasgos, los errores que se han cometido son muy superiores a los cometidos en las primeras pruebas que se realizaron en una superficie más controlada. Esto es causa de los errores que se han cometido configurando las posiciones de los *anchors* ya que resultaba bastante difícil tomar una buena medida en esas condiciones, esto genera gran cantidad de errores, sin embargo, otra de las causas que influyen en gran medida en el error cometido, es el hecho de que la estructura completa del invernadero es metálica como ya se ha visto antes, además el metal se emplea para diversos usos dentro del invernadero como por ejemplo para sujetar los hilos que están unidos a las plantas, de hecho el soporte sobre el que se han colocado las balizas también es metálico ya que no había posibilidad de colocarlo en otra ubicación mejor, todos estos elementos metálicos generan un apantallamiento de las ondas electromagnéticas las cuales sufren perturbaciones y generan errores en las medidas, es más, el propio fabricante de las balizas aconseja que no se coloque demasiado cerca de estructuras grandes de metal, lo cual se convierte en una tarea imposible en un invernadero.

5. Conclusiones y trabajos futuros

En este trabajo se ha planteado la posibilidad de emplear un sistema de posicionamiento para interiores de bajo coste, basado en UWB, en un sistema robótico para controlar la posición de este y tener la capacidad de guiarlo por un entorno conocido.

Los resultados que se han ido obteniendo a lo largo del proyecto dejan ver que la posibilidad de llevar a cabo una aplicación de este tipo es viable debido a que los errores que se han cometido no son demasiado grandes en ambientes controlados. Sin embargo, cuando este sistema se traslada a un ambiente con una fuerte presencia de estructuras metálicas, como es el caso de un invernadero, se aprecia un crecimiento muy alto del error que se comete, lo que llevaría a un mal posicionamiento del robot y por lo tanto a la imposibilidad de guiarlo de forma correcta por la superficie elegida, basándose únicamente en el sistema de balizas

Como trabajos futuros, se podría plantear la combinación de varios sistemas como puede ser SLAM u otros, con lo que se conseguirá más información de la posición del robot y se aumentaría la precisión. Además, si la aplicación se aplica en un invernadero, también se debería de considerar la posibilidad de crear ciertos soportes no metálicos para colocar las balizas ya que como se ha visto la influencia del metal sobre estas genera bastantes errores y, en función del tipo de plantación que este tenga, se tendría que crear un soporte en el robot para elevar el *tag* por encima del nivel de la vegetación y que la señal sufra las menos perturbaciones posibles. Otra de las recomendaciones para esta aplicación es la de aprovechar una red eléctrica cercana para alimentar las balizas y evitar que estas dependan de una batería externa. Por último, durante el montaje de este sistema, se deberían emplear sistemas de medición lo más precisos posibles, como pueden ser metros y niveles láser, para la configuración de las coordenadas de los *anchors*. Con la aplicación de todas estas mejoras, el sistema mejoraría lo suficiente como para ser una buena solución en aplicaciones de robot en invernadero.

6. Bibliografía

[1] Craig, J. J. (2006). *Robótica.* Pearson Edución

[2] DECAWAVE. (2017). *MDEK 1001 Kit User Manual.* https://www.decawave.com/sites/default/files/mdek1001_system_user_manual.pdf

[3] iRobot. (2018). *iRobot® Create® 2 Open Interface(OI) Specification based on the iRobot® Roomba® 600.* https://www.irobotweb.com/-/media/MainSite/Files/About/STEM/Create/2018-07-19_iRobot_Roomba_600_Open_Interface_Spec.pdf

[4] Raspberry Pi. (2019). *Raspberry Pi 4 Model B Datasheet.* https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[5][2] Borenstein, J., Everett, H. R., Feng, L., Lee, S. W., Byrne, R. H., Borenstein, J., & Feng, L. (1996). *Where am I? Sensors and Methods for Mobile Robot Positioning Prepared by the University of Michigan For the Oak Ridge National Lab (ORNL) D&D Program and the United States Department of Energy's Robotics Technology Development Program Within the Environmental Restoration, Decontamination and Dismantlement Project*.

[6][3] Subedi, S., & Pyun, J. Y. (2020). A survey of smartphone-based indoor positioning system using RF-based wireless technologies. In *Sensors (Switzerland)* (Vol. 20, Issue 24, pp. 1–32). MDPI AG. https://doi.org/10.3390/s20247230.

[7] Moreno, J. M. (2007). *Informe Técnico: Protocolo ZigBee (IEEE 802.15.4)*.

[8] Goswami, S. (2013). Indoor location technologies. In *Indoor Location Technologies*. Springer New York. https://doi.org/10.1007/978-1-4614-1377-6.

[9] Embleblue. (2014). *iBeacon Series Product specification.*

[10] Koubaa, A. (Ed.). (2016). *Robot Operating System*. Springer.

[11] Newman, W. (n.d.). *A Systematic Approach to Learning Robot Programming with ROS*.

[12] Martínez, A. (2014). *ROS Concepts. ROS.org. http://wiki.ros.org/ROS/Concepts*

[13][Cernuda del Río, A., Gayo Avello, D., Cueva Lovelle, J. M., López Pérez, B., Díaz Fondón, M. Á., Tajes Martínez, L., Álvarez García, F., Carús Villazón, C., Cueva Lobelle, N., del Puerto Paule Ruiz, M., Pérez Díaz, J. A., Álvarez Gutiérrez, D., Luengo Díez, C., García Fernández, N., González Rodríguez, M., Pérez Pérez, J. R., Riesco Albizu, M., Labra Gayo, J. E., & Martínez Prieto, A. B. (2001). *Fundamentos de informática general*. Universidad de Oviedo. Servicio de Publicaciones.

[14] Jorba, J., Suppi, R. (2004). *Administración avanzada de GNU/Linux*. UOC

[15] Fernández Montoro, A. (2012). *Python 3 al descubierto*. RC Libros.

[16] Moore, H. (2007). *Matlab® para ingenieros.* Pearson.

[17] Ollero, A. (2001). *Robótica, manipuladores y robótica móvil.* Marcombo

[18] Hernández Millán, G., Ríos Gonzales, L. H., & Bueno López, M. (2016). Implementación de un controlador de posición y movimiento de un robot móvil diferencial. *Revista Tecnura, 20(48)*, 123-136. doi: https://doi.org/10.14483/udistrital.jour.tecnura.2016.2.a09.

[19] Perron, J. (2017). *Create_autonomy. ROS.org. http://wiki.ros.org/create_autonomy*

[20] Moreno, J. C., Giménez, A., & Rodríguez, F. (2019). Proyecto AGRICOBIOT: Robot Colaborativo para Transporte Inteligente en Interior de Invernaderos con Soporte en IoT. *Jornadas Nacionales de Robótica.* http://jnr2019.ua.es/

[21] Valera Martínez, D. L., Belmonte Ureña, L. J., Molina Aiz, F. D., & López Martínez, A. (2014). *Los invernaderos de Almería Análisis de su tecnología y rentabilidad* (pp. 69–73). Cajamar Caja Rural.

Anexos

A. Presupuesto del sistema

Material	Unidades	Precio unitario (€)	Precio total (€)
Kit MDEK 1001	1	214,92	214,92
iRobot iCreate 2	1	249,99	249,99
Raspberry Pi 4 Model b	1	179	179
Pilas recargables RCR123a	12	1,34	16,08
Power bank HETP	1	29,95	29,95
Total			689,94

Tabla 3: Presupuesto del proyecto

B. Código fuente

B.1 Códigos de Matlab

```
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


N =200;
%Parametros del robot
b = 0.23;
r = 0.035;

%Postura inicial
x0 = 0;
y0 = 0;
fi0 = 0;

% x = zeros(1,N);
% y = zeros(1,N);
% fi = zeros(1,N);


x(1) = x0;
y(1) = y0;
fi(1) = fi0;

%Tiempo de muestreo
T = 0.1;
%Velocidades
w = 0;
v = 1; % [m/s]
% mapa inicial
xc = zeros(N,1);
yc = zeros(N,1);
```

```
%Camino a seguir
d = 20;
%Senoidal
for i = 1:N+d
    xc(i)=sin((i-1)/10);
    yc(i)=20*(i-1)/100;
end

%%%%%%%%%%%%%%%%%%

%Recta
% xc(1) = 0;
% yc(1) = 0;
% for i = 2:N+d
%     xc(i) = i/10;
%     yc(i) =xc(i);
% end

%%%%%%%
hold on
plot(xc,yc,'r');

hold off


% seguimiento de trayectoria con pure pursuit

wd = zeros(N,1);
wi = wd;
Dx = wd;
L = wd;
g = wd;



%Búsqueda del pto objetivo

for i = 1:N-1
    Lmin = 1e5;
    for n=1:length(xc)
        Lk = sqrt((xc(n)-x(i))^2+(yc(n)-y(i))^2);
        if(Lk<Lmin)
            Lmin=Lk;
            nmin=n;
        end
    end
    xob = xc(nmin+d);
    yob = yc(nmin+d);

    L(i) = sqrt((xob-x(i))^2+(yob-y(i))*2);
    Dx(i) = (xob-x(i))*cos(fi(i))+(yob-y(i))*sin(fi(i));
    g(i) = -2*Dx(i)/L(i)^2;

    wi(i)=(v/r)*(1-(b/2)*g(i));
    wd(i)=(v/r)*(1+(b/2)*g(i));

    %cinemática del mecanismo diferencial

    x(i+1)=x(i)-(T/2)*r*sin(fi(i))*(wd(i)+wi(i));
```

```
    y(i+1)=y(i)+(T/2)*r*cos(fi(i))*(wd(i)+wi(i));
    fi(i+1)=fi(i)+T*(r/b)*(wd(i)-wi(i));
    hold on
    plot(x(i+1),y(i+1),'hr');
    hold off

end
hold on
plot(x,y,'ob');
hold off

xlabel('x')
ylabel('y')
```

Pure_pursuit_simulación.py

B.2 Códigos de Python

```
#DWM Serial Parser by Brian H. | Updated 3/6/19
import serial
import time
import datetime
import numpy as np
import math
import matplotlib.pyplot as plt
import os
import sys

cont = 0
stop = 2.0

file =
open("/home/guille/Escritorio/Pruebas_DWM/Dinamico/Superficie_2m2_dina
mico_1.txt","w")
DWM = serial.Serial(port="/dev/ttyACM0", baudrate=115200)
print("Connected to " +DWM.name)
DWM.write("\r\r".encode())
time.sleep(1)
DWM.write("lec\r".encode())
time.sleep(1)
x = []
y = []
while True:
    try:
        line=DWM.readline()
        if(line):
            parse=line.decode().split(",")
            if parse[0] == 'POS' and parse[1] == '0'and len(parse)> 4:
                x1 = float(parse[3])
                y1 = float(parse[4])
                file.write("{:.2f},{:.2f}".format(x1,y1))
                file.write(os.linesep)
                print(x1,y1)
                cont = cont + 1

                if y1 > stop :

                    DWM.write("\r".encode())
                    DWM.close()
```

```
                    file.close()
                    sys.exit()

        else:
            print("Distance not calculated : ",line.decode())
    except Exception as ex:
        print(ex)
        break
```

DWM.py

```
import os
import matplotlib.pyplot as plt
import math
x = []
y = []
x_media = []
y_media = []
xt = []
yt = []
xaux = 0
yaux = 0
cont = 0
erroraux = 0
nmed = 10
nombre_imagen =
'/home/guille/Escritorio/Pruebas_DWM/imagenes/Grafica_DWM_Estatico_Pru
eba_2_filtro_10.png'
with
open("/home/guille/Escritorio/Pruebas_DWM/Estatico/Superficie_2m2_está
tico.txt","r") as archive:
    for linea in archive:
        if ',' in linea:
            if float(linea[(linea.index(',')+1):]) :
                x.append(float(linea[:linea.index(',')]))
                y.append(float(linea[(linea.index(',')+1):]))


#aplicamos el filtro

for i in range(len(x)):
    if cont == nmed:

        x_media.append(xaux/nfiltro)
        y_media.append(yaux/nfiltro)
        cont = 1
        xaux = x[i]
        yaux = y[i]
    else:
        xaux = xaux + x[i]
        yaux = yaux + y[i]
        cont = cont + 1


#Desinimos la trectoria para compararla con los datos tomados por las
balizas
def recta():
    global xt, yt

    for t in range(199):
```

```
            xt.append(1.0)
            yt.append(t/100)


    def punto():
        global xt, yt

        xt.append(1.0)
        yt.append(1.0)

    #Calculamos el error acumulado para cada prueba
    def calculo_error():
        global x_media,y_media,xt,yt,error, erroraux, cont2

        for i in range(len(x_media)):

            Lmin = 10000000000

            for t in range(len(xt)):

                Lk =  math.sqrt(pow((xt[t] - x_media[i]),2) + pow((yt[t] -
    y_media[i]),2))

                if Lk<Lmin:
                    Lmin = Lk

            erroraux = erroraux + Lmin


        error = erroraux/len(x_media)


    #recta()
    punto()
    calculo_error()
    print(error)


    fig, ax = plt.subplots()
    plt.xlabel("$x$[m]", fontsize = 20)
    plt.ylabel("$y$[m]", fontsize = 20)
    plt.grid(True)
    ax.scatter(x_media,y_media,color = 'tab:purple')
    ax.scatter(xt,yt,color = 'tab:green')
    plt.legend(('Trayectoria con UWB','Trayectoria real') , loc = 'upper
    right')
    plt.savefig(nombre_imagen, dpi = 300)
    plt.show()
    archive.close()
```

Analisis_Datos_DWM.py

B.3 Nodos de ROS

```
#!/usr/bin/env python
import rospy
import numpy as np
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import *
import matplotlib.pyplot as plt
import os

#Datos de inicializacion
file =
open("/home/guille/Escritorio/Pruebas_PP_sin_DWM/Curva/Vel_0.05_d_3.tx
t","w")
d = 3
r = 0.23 #23cm distancia entres ruedas
b = 0.035 #3.5cm radio de las ruedas
e = 0.05 #error
vel = Twist() #Velocidad para publicar en el Robot
vel.linear.x = 0.05#Velocidad lineal constante
vel.angular.z = 0 #Velocidad angular inicial
goal = False #Bit de senalizacion de final de trayectoria
#trayectoria
xt = [0.0]
yt = [0.0]
#Objetivo
xob = 0
yob = 0
#Posicion y orientacion actual
yaw = 0
pos = [0.0,0.0]


pub = rospy.Publisher('cmd_vel', Twist, queue_size=1) #con esto
publicaremos los mensajes de la velocidad

def begin(): #Inicializamos el nodo y definimos como subscriber del
topic /odom

    rospy.init_node('pure_pusuit_1', anonymous=True)
    rospy.Rate(5)
    rospy.Subscriber('odom',Odometry,Position)


#Funcion para la obtencion de la posicion y orientacion a traves del
topic /odom

def Position(odom_data):
    global yaw
    global pos
    global xs,ys

    curr_time = odom_data.header.stamp
    orientation = odom_data.pose.pose.orientation # orientation x y z
w
    quaternion = [
```

```
            odom_data.pose.pose.orientation.x,
            odom_data.pose.pose.orientation.y,
            odom_data.pose.pose.orientation.z,
            odom_data.pose.pose.orientation.w]
        euler = tf.transformations.euler_from_quaternion(quaternion)
        pos = [odom_data.pose.pose.position.y * (-1),
               odom_data.pose.pose.position.x]

        yaw = (-1)*euler[2]

        file.write("{:.2f},{:.2f}".format(pos[0],pos[1]))
        file.write(os.linesep)

    #Definicion de la trayectoria (recta)

    def Trayectoria(x0,y0,xf,yf):

        global xt,yt

        xt = [x0]
        yt = [y0]
        pdt = (y0-yf)/(x0-xf)
        org = y0-pdt*x0
        paso= 0.01
        pasofloat = float(paso)
        rng = int((xf-x0)/pasofloat)
        npuntos = list(range(rng))
        del npuntos[0]

        for i in npuntos:
            xt.append(xt[i-1]+pasofloat)
            num = pdt*xt[i] + org
            yt.append(num)


        xt.append(xf)
        yt.append(yf)

    #Definicion de la trayectoria (curva)

    def curva():
        global xt, yt

        for i in range(1,151):
            xt.append(float(xt[i-1] +0.01))
            yt.append(math.sin(xt[i]*(3.1416/2.0)))


    #Definicion de la trayectoria (senoidal)
    def senoide():
        global xt, yt

        for i in range(1,501):
            xt.append(math.sin(float(i)/75)*0.75)
            yt.append((float(i)-1)/175)



    #Obtencion del pto objetivo

    def objetivo():
```

```
    global pos,yaw,xob,yob,d

    Lmin = 10000.0
    n = 0

    for i in range(len(xt)):

        Lk = math.sqrt(pow((xt[i] - pos[0]),2) + pow((yt[i] -
pos[1]),2))
        if Lk<Lmin:
            Lmin = Lk
            n = i
    if (n+d) < len(xt):
        xob = xt[n + d]
        yob = yt[n + d]
    else:
        xob = xt[-1]
        yob = yt[-1]


#Calculo de la velocidad angular para alcanzar el pto objetivo

def calculo_vel():
    global xob,yob,pos,vel,r,b

    L = math.sqrt(pow((xob - pos[0]),2) + pow((yob - pos[1]),2))
    Dx = (xob-pos[0])*math.cos(yaw) - (yob - pos[1]) *math.sin(yaw)
    if L == 0:
        g = 0;
    else:
        g = -2*Dx/(pow(L,2))
    #rr = 1/g #radio de curvatura
    wi = (vel.linear.x/r)*(1-(b/2)*g)
    wd = (vel.linear.x/r)*(1+(b/2)*g)
    vel.angular.z = (r/b)*(wd-wi)



if __name__ == "__main__":


        try:
            begin()
            #Trayectoria(0.0,0.0,2.0,2.0)
            curva()
            #senoide()
            while goal == False:
                objetivo()
                calculo_vel()
                pub.publish(vel)
                if (((pos[0] > (xt[-1]-e)) and (pos[0] < (xt[-1]+e)))
and ((pos[1] > (yt[-1]-e)) and (pos[1] < (yt[-1]+e)))):
                        goal = True


            print('Fin del recorrido')
            file.close()
            sys.exit()
```

```
        except rospy.ROSInterruptException as e:
            print(e)
            sys.exit()
        except KeyboardInterrupt:
            #file.close()
            sys.exit()
```

pure_pursuit_1.py

```
#!/usr/bin/env python
import rospy
import numpy as np
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import *
import matplotlib.pyplot as plt
import os
from position.msg import pos_1_2


#Datos de inicializacion
file =
open("/home/guille/Escritorio/Pruebas_PP_con_DWM/Curva/d_20_nmed_10.tx
t","w")
d = 20
r = 0.23 #23cm distancia entres ruedas
b = 0.035 #3.5cm radio de las ruedas
e = 0.05 #error
vel = Twist() #Velocidad para publicar en el Robot
vel.linear.x = 0.15#Velocidad lineal constante
vel.angular.z = 0 #Velocidad angular inicial
goal = False #Bit de senalizacion de final de trayectoria
#trayectoria
xt = [0.0]
yt = [0.0]
#Objetivo
xob = 0
yob = 0
#Posicion y orientacion actual
yaw = 0
a1 = 0.0
a2 = 0.0
pos = [a1,a2]
#Configuracion adquision datos DWM
nmedia = 10
cont = 0
posauxx = 0
posauxy = 0

#--------------------#
pub = rospy.Publisher('cmd_vel', Twist, queue_size=1) #con esto
publicaremos los mensajes de la velocidad



def begin(): #Inicializamos el nodo y definimos como subscriber del
topic /odom
```

```
    rospy.init_node('pure_pusuit_DWM', anonymous=True)
    rospy.Rate(20)
    rospy.Subscriber('Rposition',pos_1_2,Position)
    rospy.Subscriber('odom',Odometry,Orientation)


def Position(data):
    global pos,posauxx,posauxy, nmedia, cont

    if cont == nmedia:
        pos[0] = posauxx/nmedia
        pos[1] = posauxy/nmedia
        cont = 0
        posauxx = 0
        posauxy = 0

        pos = [data.x1,data.y1]
        file.write("{:.2f},{:.2f}".format(pos[0],pos[1]))
        file.write(os.linesep)

    else:
        posauxx = posauxx + data.x1
        posauxy = posauxy + data.y1
        cont = cont + 1

    pos = [data.x1,data.y1]

def Orientation(odom_data):
    global yaw, pos, a, b


    curr_time = odom_data.header.stamp
    orientation = odom_data.pose.pose.orientation # orientation x y z
w
    quaternion = [
        odom_data.pose.pose.orientation.x,
        odom_data.pose.pose.orientation.y,
        odom_data.pose.pose.orientation.z,
        odom_data.pose.pose.orientation.w]
    euler = tf.transformations.euler_from_quaternion(quaternion)
    #pos = [odom_data.pose.pose.position.y*(-1)+(a1),
odom_data.pose.pose.position.x+(a2)]


    yaw = (-1)*euler[2]



def Trayectoria(x0,y0,xf,yf):

    global xt,yt

    xt = [x0]
    yt = [y0]
    pdt = (y0-yf)/(x0-xf)
    org = y0-pdt*x0
    if x0 > xf:
        paso = -0.01
    else:
        paso = 0.01
```

```
        pasofloat = float(paso)
        rng = abs(int((xf-x0)/pasofloat))
        npuntos = list(range(rng))
        del npuntos[0]

        for i in npuntos:
            xt.append(xt[i-1]+pasofloat)
            num = pdt*xt[i] + org
            yt.append(num)


        xt.append(xf)
        yt.append(yf)

    def curva():
        global xt, yt

        for i in range(1,151):
            xt.append(float(xt[i-1] +0.01))
            yt.append(math.sin(xt[i]*(3.1416/2.0)))


    #Definicion de la trayectoria (senoidal)
    def senoide():
        global xt, yt

        for i in range(1,501):
            xt.append(math.sin(float(i)/75)*0.75)
            yt.append((float(i)-1)/175)



    #Obtencion del pto objetivo

    def objetivo():

        global pos,yaw,xob,yob,d

        Lmin = 10000.0
        n = 0

        for i in range(len(xt)):

            Lk = math.sqrt(pow((xt[i] - pos[0]),2) + pow((yt[i] -
    pos[1]),2))
            if Lk<Lmin:
                Lmin = Lk
                n = i
        if (n+d) < len(xt)  :
            xob = xt[n + d]
            yob = yt[n + d]
        else:
            xob = xt[-1]
            yob = yt[-1]


    #Calculo de la velocidad angular para alcanzar el pto objetivo

    def calculo_vel():
        global xob,yob,pos,vel,r,b
```

```
        L = math.sqrt(pow((xob - pos[0]),2) + pow((yob - pos[1]),2))
        Dx = (xob-pos[0])*math.cos(yaw) - (yob - pos[1]) *math.sin(yaw)
        if L == 0:
            g = 0;
        else:
            g = -2*Dx/(pow(L,2))
        #rr = 1/g #radio de curvatura
        wi = (vel.linear.x/r)*(1-(b/2)*g)
        wd = (vel.linear.x/r)*(1+(b/2)*g)
        vel.angular.z = (r/b)*(wd-wi)

if __name__ == "__main__":


            try:

                begin()
                #Trayectoria(0.1,0.1,1.0,1.0)
                curva()
                #senoide()

                while goal == False:
                    objetivo()
                    calculo_vel()
                    pub.publish(vel)
                    if (((pos[0] > (xt[-1]-e)) and (pos[0] < (xt[-1]+e)))
and ((pos[1] > (yt[-1]-e)) and (pos[1] < (yt[-1]+e)))):
                            goal = True


                print('Fin del recorrido')
                file.close()
                sys.exit()

            except rospy.ROSInterruptException as e:
                print(e)
                sys.exit()
            except KeyboardInterrupt:
                #file.close()
                sys.exit()
```

pure_pursuit_DWM.py

```
    #!/usr/bin/env python

import rospy
import serial
import math
import numpy as np
import time
from position.msg import pos_1_2
```

```
DWM = serial.Serial(port="/dev/ttyACM0", baudrate=115200)
print("Connected to " +DWM.name)
DWM.write("\r\r".encode())
time.sleep(1)
DWM.write("lec\r".encode())
time.sleep(1)



def Posiciones():
    pub = rospy.Publisher('Rposition', pos_1_2, queue_size=10)
    rospy.init_node('Position_DWM', anonymous=True)
    rate = rospy.Rate(20) # 20hz
    pos = pos_1_2()
    time.sleep(1)
    pos.x1 = 0
    pos.y1 = 0
    pos.x2 = 0
    pos.y2 = 0
    while not rospy.is_shutdown():


        line=DWM.readline()
        linestr = str(line)
        if(line):
            parse=line.decode().split(",")
            print(parse)
            if parse[0] == 'POS' and '0C33' in parse[2]  and
len(parse) == 8:
                pos.x1 = float(parse[3])
                pos.y1 = float(parse[4])

            if parse[0]=='POS'and'C424'in parse[2]andlen(parse) == 8:
                pos.x2 = float(parse[3])
                pos.y2 = float(parse[4])


        else:
            print("Position not calculated: ",line.decode())

        pub.publish(pos)
        rate.sleep()


if __name__ == '__main__':
    try:
        Posiciones()
    except rospy.ROSInterruptException:
        pass
```

Position_DWM.py

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

.La robótica se ha convertido en los últimos años en la solución más empleada a la hora de resolver los diferentes problemas que surgen tanto en los procesos industriales como en otros ámbitos. Uno de los campos de estudio dentro de la robótica que está en expansión es el de la robótica móvil, en gran parte por varias marcas de automóviles que están invirtiendo grandes sumas de dinero en estos estudios. En la robótica móvil uno de los problemas recurrentes es el del posicionamiento de los robots, sobre todo cuando se necesita mucha precisión para evitar ciertos obstáculos y más aún si el lugar por donde el robot se debe mover es un recinto cerrado. En aras de plantear una solución a este problema, en este proyecto se trata de buscar una solución para el posicionamiento en interiores de robots móviles investigando primero los tipos de sistemas que existen y están disponibles en el mercado para posteriormente seleccionar el que más convenga para realizar pruebas. Además de esto, se implementará un algoritmo de seguimiento de trayectorias en un robot para probar el funcionamiento del sistema seleccionado en conjunto con un robot móvil.

Currently, robotics has become the most widely used solution to solve the different problems that appear in industrial processes and other areas. One of the fields of study within robotics that is expanding is that of mobile robotics, due to several automobile brands that are investing large sums of money in these studies. In mobile robotics one of the recurring problems is the positioning of robots, especially when high precision is needed to avoid certain obstacles and even more so if the place where the robot must move is an indoor area. In order to propose a solution to this problem, this project aims to find a solution for indoor positioning of mobile robots by first investigating the types of systems that exist and are available on the market and then select the most suitable for testing. In addition to this, a trajectory tracking algorithm will be implemented in a robot to test the performance of the selected system in conjunction with a mobile robot.