UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Estudio e integración de un sistema de localización *indoor* en un prototipo de vehículo autónomo

Autor:

D. Óscar Martín Casares

Tutor:

Dr. Juan Carlos Aguado Manzano

D. Adrián Mazaira Hernández

Valladolid, julio de 2024

Título	**Estudio e integración de un sistema de localización *indoor* en un prototipo de vehículo autónomo**
Autor	**D. Óscar Martín Casares**
Tutor	**Dr. Juan Carlos Aguado Manzano** **D. Adrián Mazaira Hernández**
Departamento	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

Presidente	**Ignacio de Miguel Jiménez**
Vocal	**Juan Pablo Casaseca de la Higuera**
Secretario	**Alfonso Bahillo Martínez**

Fecha
Calificación

Resumen

El trabajo realizado persigue como objetivo utilizar una herramienta para la localización en circunstancias *indoor* de un vehículo Renault Twizy, así como el desarrollo de un entorno que permite la interoperabilidad entre diferentes sistemas, como pueden ser diferentes programas o estructuras físicas.

Tras realizar un análisis exhaustivo del escenario por el cual se va a mover el vehículo y a través de la utilización de balizas de ultrasonidos, se localizará el vehículo en dicha superficie. Para ello, se han desarrollo distintos programas que nos han permitido conocer cuál es el rendimiento y precisión en las distintas pruebas realizadas, en la cuales veremos diferentes disposiciones de las balizas con el objetivo de obtener los mejores resultados de localización posibles. Asimismo, se explicará cuál es el algoritmo de control del prototipo de vehículo autónomo, la información de control necesaria, la arquitectura desplegada y cómo se realiza la comunicación entre los distintos elementos que la componen.

En el presente trabajo se podrán encontrar explicaciones de todos los procesos llevados a cabo, con la intención de que cualquier persona pudiera entenderlo y desplegarlo si fuera necesario. Finalmente podremos encontrar una explicación de futuras implementaciones que se podrían hacer para llegar más lejos aún en la autonomización del Renault Twizy en interiores.

Abstract

The work carried out aims to use a tool for indoor localization of a Renault Twizy vehicle, as well as the development of an environment that allows interoperability between different systems, such as various programs or physical structures.

After a thorough analysis of the scenario in which the vehicle will move and using ultrasonic beacons, the vehicle will be localized on that surface. To achieve this, different programs have been developed that have allowed us to know the performance and accuracy of the different tests performed, in which we will see different arrangements of the beacons with the goal of obtaining the best possible localization results. Additionally, the control algorithm of the autonomous vehicle prototype, the necessary control information, the deployed architecture, and how communication is carried out between the different elements that comprise it will be explained.

In this work, you will find explanations of all the processes carried out, with the intention that anyone could understand and deploy it if necessary. Finally, we will provide an explanation of future implementations that could be made to further advance the indoor automation of the Renault Twizy.

Índice de contenido

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Motivación

Durante la última década, los avances en inteligencia artificial, sensores y sistemas de procesamiento han impulsado el desarrollo de los vehículos autónomos, que se espera que revolucionen la forma en que viajamos e interactuamos con el entorno. A estos vehículos se les presupone unas capacidades cognitivas y de toma de decisiones, tienen el potencial de reducir significativamente los accidentes de tráfico, optimizar el uso de la calzada y mejorar la eficiencia del transporte.

Los vehículos autónomos prometen ofrecer una variedad de beneficios tanto para la sociedad como para la propia industria. En primer lugar, tiene el potencial de reducir drásticamente los accidentes de tráfico causados por errores humanos, que actualmente es una de las principales causas de muerte y lesiones en todo el mundo, donde se estima que se pierden 1.35 millones de vidas y más de 50 millones de lesiones cada año [1]. Además, los vehículos autónomos reducirán la congestión del tráfico y las emisiones de gases de efecto invernadero al optimizar la eficiencia del transporte, contribuyendo así a la protección del clima.

En la industria automotriz, la introducción de vehículos autónomos tiene el potencial de transformar los modelos comerciales tradicionales e impulsar la innovación en áreas como la movilidad como servicio (MaaS) o la entrega de productos sin conductor. Además, la creciente demanda de tecnología de conducción autónoma está creando nuevas oportunidades de colaboración entre fabricantes de automóviles, impulsando la creación de empleo y crecimiento económico.

La universidad no es ajena a esta tendencia, y particularmente en la ETSIT de Valladolid se ha venido desarrollando un prototipo de vehículo autónomo. El presente Trabajo Fin de Máster bebe de los progresos realizados hasta el momento y de las carencias que todavía tiene. El prototipo surge de un proyecto realizado por cuatro estudiantes de la Escuela de Ingenieros de Telecomunicación de Valladolid para el concurso TwizyContest celebrado en 2020. Este proyecto se denominó TwizyLine y su objetivo era desarrollar un prototipo de vehículo autónomo, destinado específicamente a automatizar la tarea de aparcamiento de vehículos conectados y autónomos. Debido a la propia naturaleza de las bases del concurso, el proyecto se llevó a cabo en un vehículo eléctrico Renault Twizy.

Figura 1. Logotipo del proyecto TwizyLine.

Entre las distintas tareas llevadas a cabo por los estudiantes nos encontramos con: realizar un análisis de mercado sobre el estado de los estacionamientos autónomos, el desarrollo de una aplicación móvil que permitiera a los usuarios acceder al servicio, el despliegue de un servidor

encargado de controlar el sistema y del aparcamiento autónomo, y el diseño de un sistema que permita añadir capacidades de autonomía al coche [2] [3] [4] [5].

Más adelante se ha continuado realizando mejoras, como la incorporación de medidas de ciberseguridad en el servidor, la reorganización de la estructura eléctrica y de comunicaciones dentro del vehículo o el intento de integrar un LiDAR al vehículo para una mejor localización del mismo.

Es en este punto donde se inserta este Trabajo Fin de Máster, dada la dificultad de integrar la tecnología LiDAR que se vio en trabajos fin de máster anteriores y que es necesario mejorar la localización del Twizy en interiores para que pueda realizar trayectos más seguros y rápidos. Es evidente que la la tecnología GPS no es apta para interiores, por lo que se ha decidido integrar una tecnología de localización en interiores que nos permita sustituir la información del GPS. Con esta información sería más fácil integrar la tecnología LiDAR, pues en muchos casos las librerías LiDAR requieren entre otras informaciones las del GPS.

En concreto se ha decidido utilizar la tecnología de localización indoor desarrollada por la empresa Marvelmind, que se basa principalmente en la localización de móviles a través de balizas que utilizan ultrasonidos. Esta tecnología ya ha sido probada dentro del Grupo de Comunicaciones Ópticas en otro proyecto de investigación, en particular en el desarrollo de una maqueta de simulación de vehículo conectado. El trabajo fin de máster que se presenta aquí utiliza como base el trabajo fin de máster desarrollado por D. Iván Viloria Vázquez [6], pero mientras aquel buscaba el uso de la tecnología en un entorno muy pequeño y controlado, en el caso del presente trabajo fin de máster se quiere utilizar en un entorno real que añade gran dificultad al uso de la tecnología.

1.1 Objetivos

El objetivo principal de este Trabajo Fin de Máster consiste en lograr la localización de un prototipo de vehículo autómono realizado sobre un Renault Twizy mediante la utilización de balizas de ultrasonidos.

Obtenida la localización, el segundo objetivo fundamental es el procesamiento de dicha información para poder facilitársela al ordenador a bordo del vehículo de tal manera que pueda utilizar dicha información para guiarse a través de un mapa ya cargado en el mismo.

Este objetivo general se puede subdividir en varios objetivos específicos:

- Estudiar la herramienta de localización *indoor* de Marvelmind para su aplicación en el Twizy.
- Adaptar la herramienta al escenario de prueba para poder obtener el mayor rendimiento posible.
- Obtención y transformación de los datos de localización de las balizas para su consumo por parte del vehículo en las mejores condiciones posibles, tasa de refresco, precisión…
- Comunicación entre el módulo de fusión y el módulo de control lateral del vehículo de la información necesaria.

1.2 Fases y métodos

Durante el desarrollo de este proyecto se han seguido los siguientes pasos:

1. **Comprensión del trabajo ya realizado.** Dado que este trabajo es continuación de otro proyecto, ha sido necesario comprender las operaciones que ya había realizado D. Iván Viloria Vázquez [6].
2. **Formación y ampliación de conocimientos.** De cara a afrontar los retos de este proyecto, fue necesario revisar algunos conceptos de programación junto con una ampliación de los mismos, además de aprender a manejar diferentes herramientas *software*.
3. **Analizar el escenario de trabajo.** Antes de comenzar a trabajar y realizar pruebas, fue necesario analizar el escenario donde se iban a desplegar las distintas herramientas para así poder adaptarse a él.
4. **Implementación de funcionalidades.** A la localización del vehículo se le han añadido diferentes funcionalidades de gran utilidad para analizar su rendimiento y mejorarlo.
5. **Realización de pruebas.** A lo largo de todo el proyecto se han realizado múltiples pruebas, todas las mostradas en el presente trabajo y otras a pequeña escala que han sido de utilidad para ver cómo funcionaban las distintas herramientas.

1.3 Recursos disponibles

Para el desarrollo de este proyecto se han utilizado diferentes recursos, tanto *hardware* como *software*.

Entre los **recursos *hardware*** nos encontramos el ordenador facilitado por el Grupo de Investigación de Comunicaciones Ópticas, un Zotac ZBOX Magnus EN072080S con sistema operativo Linux cuyas características son las siguientes:

- Procesador Intel Core i7 de seis núcleos a 2,6 GHz.
- 2 memorias RAM de 16 GB DDR4 SO-DIMM.
- Tarjeta gráfica NVIDIA GeForce RTX 2080 Super 8 GB GDDR6.

Asimismo, se ha utilizado un ordenador personal con sistema operativo MacOS, cuyas características técnicas son:

- Procesador Intel Core i5 de doble núcleo a 2,3 GHz.
- Memoria RAM de 8 GB LPDDR3 integrada a 2.133 MHz.
- Tarjeta gráfica Intel Iris Plus Graphics 640.

Finalmente, también se han utilizado las balizas de ultrasonidos de Marvelmind que se detallarán en la sección 3.2.1.

Entre los **recursos *software*** nos encontramos con AutoCAD y la aplicación Dashboard de Marvelmind.

1.4 Estructura de la memoria

Después de este primer capítulo de introducción y motivación, donde se han descrito también las fases por las que ha pasado el proyecto, la memoria se ha estructurado de la siguiente manera:

El **segundo capítulo** lleva a cabo una revisión del estado del proyecto TwizyLine y en base a este estado se contextualiza el momento actual por el que pasan las tecnologías que son utilizadas en este proyecto para avanzar en los objetivos de TwizyLine.

En el **tercer capítulo** podremos encontrar una explicación de las distintas herramientas utilizadas y cómo ponerlas en marcha, también un análisis del escenario de trabajo del vehículo y algunas de las distintas pruebas realizadas. De este modo, podremos comprender el funcionamiento de la herramienta y el rendimiento que proporcionan.

A lo largo del **cuarto capítulo** encontraremos las explicaciones y desarrollos llevados a cabo para procesar y enviar la información de localización al vehículo, así como una futura posibilidad de despliegue del sistema.

Asimismo, podremos encontrar un **quinto capítulo** destinado a exponer las conclusiones a las que se ha llegado, junto con la propuesta de algunas líneas de desarrollo futuras para la herramienta.

De forma adicional, al final del presente trabajo se encuentran los **anexos**, en los cuales podremos encontrar los códigos Python desarrollados para realizar las distintas pruebas e implementación de funcionalidades, así como la librería utilizada.

Capítulo 2. Situación inicial y estado del arte

2.1 Introducción

En este capítulo se detallará en qué estado se encontraba el proyecto que este trabajo quiere continuar. Se hará un recorrido por las distintas tecnologías existentes en la actualidad para lograr los objetivos que se buscan en este Trabajo Fin de Máster y por los métodos utilizados para desarrollar un vehículo autónomo en términos generales. Asimismo, se verá que hay que modificar sustancialmente el control de ruta que ahora mismo se encuentra implementado en el Twizy. En concreto, al incluir el posicionamiento *indoor* que es equivalente a un GPS, se puede predefinir rutas a seguir, para lo cual hay que dotar al vehículo de mapas predefinidos y rutas. Por ello se detallará las distintas formas definir mapas de manera útil para la conducción autónoma y sus formas de obtención.

2.2 Estado previo del proyecto

El presente trabajo y los que le preceden se enmarcan dentro del proyecto TwizyLine iniciado en el año 2020 por alumnos de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid. Dicho proyecto consistió en el desarrollo de un prototipo de vehículo autónomo para el concurso TwizyContest, cuyo fundamento era conseguir movilizar de manera autónoma a las personas que atendieran a los Juegos Olímpicos de París del año 2024 [2] [3] [4] [5].

Uno de los problemas fundamentales a los que se enfrenta el proyecto TwizyLine es que todas las pruebas se realizan en interiores, de tal manera que es imposible utilizar el sensor GPS que tiene incorporado en el sistema desarrollado. Además, este sensor GPS únicamente se utilizaba para monitorizar el vehículo pensando en el control de flotas en un servicio de control de flotas de *carsharing*, no para definir una posible trayectoria óptima. Este hecho es importante, dado que la mayoría de vehículos autónomos utilizan una serie de sensores de localización para definir la ruta que van a seguir, y tras fusionar estas señales en una única señal se calcula la ruta del vehículo a largo plazo y para calcular la trayectoria a corto plazo. Por ello, veremos qué procedimientos sigue el vehículo en la actualidad para obtener su control y localización.

El sistema actual que incorpora el vehículo Twizy para su control de trayectoria no está descrito en ningún trabajo fin de grado o máster relacionado con el proyecto TwizyLine, sino que debido a que se incorporó para presentarlo en el concurso TwizyContest 2020, se aprovecho un algoritmo ya desarrollado para una maqueta de un proyecto totalmente distinto. Concretamente, el trabajo realizado por D. Ignacio Royuela González en el Trabajo Fin de Máster 'Diseño e implementación de un testbed de Edge computing para el soporte de vehículos conectados' describe el algoritmo de control autónomo del Renault Twizy [3]. Este algoritmo está dividido en tres etapas diferentes: percepción, planificación y control, siguiendo de manera muy cercana las fases que se necesitan en un vehículo autónomo real como veremos en la sección 2.5. En la etapa de percepción se interpreta el entorno del vehículo reconociendo las posibles trayectorias que puede seguir, para la cual realiza la integración de todos los datos recibidos de los sensores. En este caso se utiliza la información de un único sensor, que es una cámara de baja resolución, cuya imagen es procesada mediante filtros para identificar las posibles trayectorias definidas a través de líneas azules pintadas o sobrepuestas en el suelo. En la Figura 2 se puede ver el efecto de aplicar un filtro sobre una imagen con tres posibles

caminos, donde cada camino se ve en blanco sobre fondo negro. En la segunda etapa de planificación se decide qué trayectoria de las disponibles debe seguir el vehículo en las bifurcaciones según una ruta programada. Para ello, segmentará la imagen en tres partes, tal y como vemos en la Figura 2, y en cada subimagen se detectan áreas blancas inconexas, asociándole a cada área un punto central (representado en esta imagen con un punto rojo) que servirá para poder definir así las posibles trayectorias a seguir por el vehículo. El algortimo funcionaría de la seguiente manera: Si en las tres subimágenes hay un único punto por subimagen, eso quiere decir que hay una única trayectoria. Se calcularía la línea recta que une los dos puntos más cercanos a la cámara y se proyectaría hacia el punto de referencia del vehículo, de tal manera que se pueden calcular los dos parámetros necesarios para el control a corto plazo de la trayectoria de los que hablaremos a continuación. Si por el contrario, existe más de un punto por subimagen, eso quiere decir que hay varias trayectorias alternativas, y por lo tanto se debe tomar una decisión sobre qué trayectoria tomar. Por ejemplo, en la Figura 2 en la subfigura inferior hay un único punto y en la del medio hay tres, por lo tanto hay tres caminos alternativos a seguir. En este punto, lo ideal sería tener una referencia absoluta del lugar dónde está el vehículo, de tal manera que sabemos en qué punto del mapa estamos y de una planificación anterior podemos saber qué trayectoria tomar, pero este no es el caso. No tenemos referencia absoluta y, por lo tanto, lo único que podemos decir para guiar el vehículo es cuántas bifurcaciones de este tipo hemos pasado. Por lo tanto, el control de la trayectoria sería algo así como "en la primera bifurcación sigue recto, en la segunda gira a la derecha...". De esta forma, el vehículo puede hacer un cálculo de rutas planificado diciendole cómo debe actuar en cada bifurcación que se encuentra. Sin embargo, cuando no detecta bien las bifurcaciones comete un error y lo arrastrá continuamente debido a que no poseé un conocimiento absoluto de la localización.

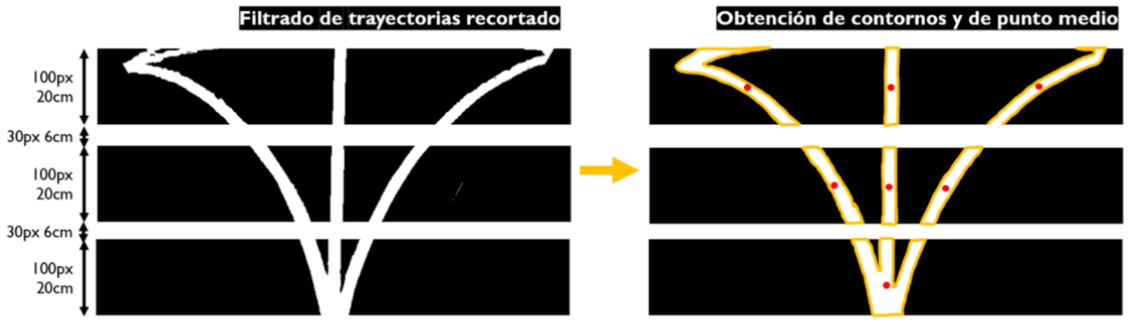

Figura 2. Segmentación de las líneas detectas para definir posibles trayectorias [3].

La tercera etapa de control se utiliza para ajustar los parámetros de control del vehículo, para que no se produzcan ni oscilaciones ni salidas de la trayectoria. Para ello, es clave obtener dos parámetros, el ángulo de *yaw* o guiñada del vehículo con respecto a la trayectoria y la distancia a la que se encuentra el vehículo con respecto a esta desde el punto de referencia del vehículo, como podemos ver en la Figura 3. Estos dos parámetros junto con el algoritmo conocido como Stanley [3], permite definir un ángulo de giro de las ruedas a corto plazo, con el objetivo de que el tiempo de convergencia del vehículo a la trayectoria sea independiente de la velocidad del vehículo. Dado que no es un objetivo de este proyecto, no se describe aquí el algoritmo, aunque sí que es importante mantener en mente los datos de entrada que necesita dado que los utilizaremos en el posterior desarrollo de nuestro proyecto.

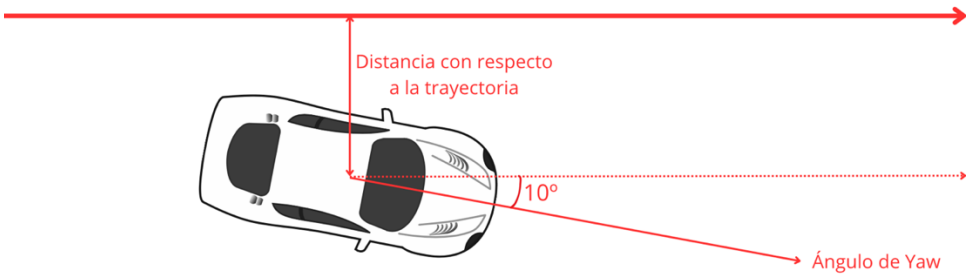

Figura 3. Distancia y ángulo necesarios por el Renault Twizy.

Dado que el vehículo no poseía ningún sistema que le permita el cálculo de la trayectoria o la localización absoluta sobre el mapa, como ya se ha visto, al poder integrar más información con respecto a la trayectoria se obtendrá un mejor control del vehículo. A lo largo del proyecto TwizyLine se han realizado varios trabajos fin de grado y máster orientados a mejorar la percepción del vehículo.

En primera instancia, a través del proyecto 'Guiado de un vehículo autónomo mediante tecnología LiDAR' desarrollado por Dª Carlota Gómez Diego, se intentó integrar un sensor LiDAR y utilizar técnicas de SLAM (Simultaneous Localization and Mapping). En concreto, se implentó la técnica LOAM (Lidar Odometry and Mapping) debido a su no dependencia de datos GPS, la cual permite estimar la posición del vehículo y generar un mapeado del entorno. Sin embargo, esta técnica no proporcionaba una localización lo suficiente precisa para trabajar con vehículos autónomos en tiempo real [7].

Su objetivo era llevar a cabo la estimación de la posición del vehículo y calcular así su trayectoria. Por ello, dado que quería mejorar la precisión, utilizó el método AMCL3D, un algoritmo que requiere la información obtenida del mapa a través de la técnica LOAM y la información de localización lograda a través de una balizas de ultrasonidos. Sin embargo, pese a dejar avanzado el diseño de la arquitectura a desplegar, no se llegó a implementar por completo debido a la envergadura del proyecto y a las dificultades de integración e interoperabilidad de información como la nube de puntos, distancia de balizas, creación del mapa y odometría. A pesar de que resultó imposible conocer la localización asboluta sobre el mapa del vehículo, Dª Carlota plantea cómo se podría resolver el problema a través de la coordinación de todos los sistemas de referencias y cómo obtener la información de control ya mencionada necesaria para el vehículo [7].

Asimismo, conociendo las dificultades vistas en el uso del LiDAR, y fruto de otro proyecto de investigación que el Grupo de Comunicaciones Ópticas estaba desarrollando en paralelo a los esfuerzos explicados, se pensó en un sistema alternativo de localización en interiores. El proyecto referido es ARTEMIS (Arquitectura Edge para el Soporte de Vehículos Conectados en Entornos Metropolitanos). En uno de los paquetes de trabajo de dicho proyecto se debía desarrollar una maqueta que permitiera probar escenarios de vehículo conectado. Dentro del desarrollo de dicha maqueta que incluía un vehículo controlado remotamente, se realizaron pruebas para posicionar dicho vehículo dentro de una pista. Dicha investigación fue recogida en el trabajo fin de máster desarrollado por D. Iván Viloria Vázquez, cuyo título es 'Integración de sistemas de posicionamiento *indoor* para un testbed de Edge Computing para el soporte de vehículos conectados'. Sus objetivos fundamentales fueron el desarrollo de un sistema de

odometría para un vehículo AWS Deepracer, realizar un estudio de diferentes sistemas *indoor* y su integración en el vehículo [6].

El vehículo Deepracer desarrollado por Amazon posee una IMU que permite la obtención de distinta información como la aceleración instantánea o velocidad, pero éste no es muy fiable. Por ello, D. Iván tuvo que implementar un odómetro en dicho vehículo. En segundo lugar, D. Iván tenía que lograr obtener la localización absoluta del vehículo dentro de una maqueta diseñada para dicha tarea, así que estudió diferentes sistemas de posicionamiento *indoor*. Trabajó con dos sistemas, una primera solución más económica que utilizaba los módulos UWB M5Stack desarrollados por la empresa china M5Stack, y una segunda solución de mayor presupuesto que utilizaba los módulos Super Beacon desarrollados por la empresa finlandesa Marvelmind. Finalmente, implementó ambos sistemas en el vehículo y realizó diferentes pruebas y ensayos, concluyendo que la mejor solución era la desarrollada por Marvelmind debido a sus mejores resultados y su más sencilla instalación e integración [6].

Entonces, dado el estado actual del proyecto y conociendo todos los pasos que se han seguido hasta el momento, en el presente trabajo trataremos de resolver los problemas que acometen en la actualidad al vehículo, como la obtención de la localización absoluta, no sobre una maqueta sino sobre un escenario real, y la obtención de los parámetros de control que necesita el vehículo para operar y poder seguir la trayectoria que definiremos en el Capítulo 4. Asimismo, gracias a la labor desarrollada por D. Iván podremos partir con ventaja en este trabajo y utilizaremos las balizas de posicionamiento de Marvelmind, las cuales iremos referenciando a lo largo del presente documento.

2.3 Funcionamiento de vehículos autónomos en interiores

Vistos el problema planteado en la sección anterior sobre la movilidad del vehículo en interiores tiene especial interés hacer una revisión de cómo se ha resuelto este problema anteriormente, porque aunque ya se ha decidido qué tecnología se va a utilizar, la revisión de estado del arte nos puede servir para saber qué resultados serían esperables.

Antes de comenzar con los vehículos autónomos en interior, quizás merece la pena hacer una breve reflexión sobre lo qué es un vehículo autónomo y su historia. Los vehículos autónomos son aquellos capaces de moverse sin necesitar de intervención humana, el avance tecnológico al cual los vehículos de hoy aspiran alcanzar. El desarrollo de vehículos autónomos se remonta a principios del siglo pasado, al año 1921 donde nos encontramos con el primer vehículo radio control, un coche desarrollado por un ingeniero de la armada estadounidense de pequeñas dimensiones controlado por tecnología radio [8]. Sin embargo, los avances desde entonces han sido enormes y se espera que sigan incrementándose. De hecho, es un mercado que se espera que para 2050 repercuta en un ahorro de 800.000 millones de dólares anuales al reducir la congestión, los accidentes y el consumo de energía.

Centrándonos en los vehículos autónomos en interiores, podemos discernir diferentes tipos de vehículos:

1. **Vehículos Guiados Automáticamente (AGVs):** siguen rutas predefinidas mediante sensores o marcas en el suelo. Se utilizan para realizar tareas repetitivas y en entornos planificados.
2. **Vehículos Autónomos (AVs):** navegan por el escenario siendo capaces de adaptarse de manera flexible a los cambios del entorno.

3. **Drones:** son capaces de volar y acceder a lugares para los cuales se requiere de una mayor agilidad.

Todo este tipo de vehículos no serían capaces de moverse si no fuera por las diversas tecnologías y las técnicas de navegación que integran, las cuales les permiten navegar, percibir y movilizarse por el entorno. Gran parte de las tecnologías las detallaremos en la siguiente sección. Las técnicas y algoritmos de navegación en interiores existentes en la actualidad son las siguientes [9]:

1. **Técnicas de *Position-fixing*:** basan su funcionamiento en el cálculo geométrico de distancias y ángulos. La técnica basada en distancias o *range-based* funciona mediante multilateración, es decir, obtiene la posición a través de la estimación de la distancia de tres o más estaciones base. Existe también la técnica basada en ángulos o *angle-based* que calcula los ángulos respecto a las estaciones base y triangula para obtener la posición. Asimismo, existe una mezcla de ambas técnicas denominada *range-and-angle-based*. Podemos observar un esquema sencillo de estas técnicas en la Figura 4.

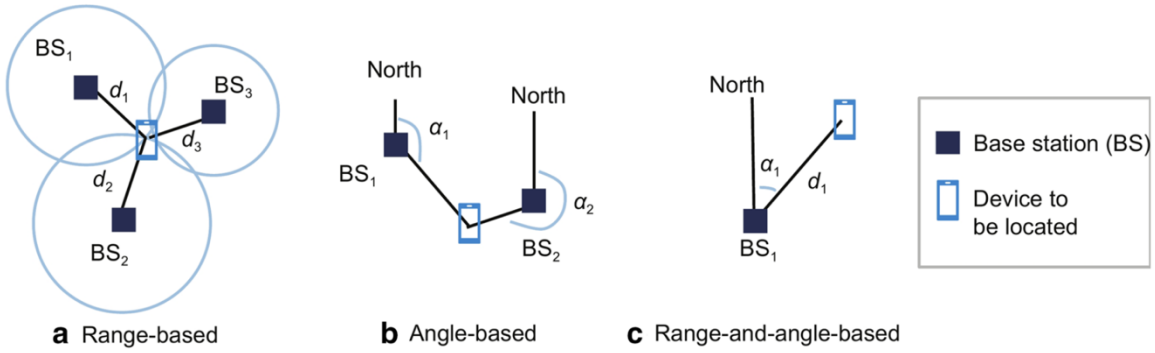

Figura 4. Distintas técnicas de position-fixing [9].

2. **Técnica de *Dead-Rocking:*** también conocida como navegación por estimación de la posición, permite obtener la posición y orientación de un vehículo sin necesidad de depender de sistemas como GPS o balizas. Basa su funcionamiento en la estimación constante de la posición a partir de su movimiento anterior, utilizando la información de sensores como giroscopios, acelerómetros y odómetros. Sin embargo, estas técnicas son sensibles a la acumulación de errores.

3. **Técnica de *Database Matching:*** basan su funcionamiento en la comparación entre los datos obtenidos de los diferentes sensores como cámaras o LiDAR con los datos almacenados en una base de datos, para así obtener la posición y orientación de un vehículo. Requiere de poseer un registro del mapa con antelación y permite mejorar la precisión complementando en aquellos vehículos que poseen otros sensores de localización.

4. **Técnica de *Multi-Sensor Fusion:*** funcionan integrando la información de múltiples sensores como balizas, acelerómetros, odómetros… permitiendo así una mayor precisión respecto a que si utilizásemos los sensores individualmente. Permite además reducir la dependencia de un solo sensor, sin embargo, los algoritmos de fusión de sensores pueden ser muy complejos y requieren de un conocimiento profundo de sus características.

5. **Técnica de *Motion Constraints:*** se basa en el conocimiento previo del entorno y del propio vehículo para acotar las posibles ubicaciones. A partir de datos como la geometría del entorno, el rango de giro del vehículo y otros elementos, limitan el movimiento para descartas ubicaciones imposibles. Permiten ajustarse a situaciones donde el presupuesto sea limitado.

Una vez conocidas las distintas técnicas que utilizan los vehículos para ser localizados, cabe mencionar las diferentes formas de comunicación, puede darse entre los propios vehículos (V2V) o entre vehículos e infraestructura (V2I). La comunicación entre vehículos permite que compartan entre sí información importante como su posición, velocidad o intenciones de maniobra [10]. La comunicación entre vehículo e infraestructura permite intercambiar información como la posición, instrucciones de navegación o alertas. Esta información se debe intercambiar en tiempo real y se suelen utilizar tecnologías como WiFi, Bluetooth, UWB o 4G/5G.

2.4 Sistemas de posicionamiento *indoor*

Nos encontramos con dos modalidades en cuanto a lo que posicionamiento se refiere, posicionamiento en interiores y en exteriores. En exteriores nos encontramos con tecnologías satelitales como GPS (*Global Positioning System*) o GLONASS (*Global Navigation Satellite System*), pero perderán gran precisión en el momento que nos adentramos en edificaciones, por ello existen otras tecnologías destinadas a esta función.

Dado que nuestro proyecto se desarrollará en interiores, se desecharon por completo las tecnologías mencionadas. En lugar de los satélites, los sistemas de posicionamiento en interiores o IPS (*Indoor Positioning System*) hacen referencia al conjunto de dispositivos que se utilizan para localizar de manera inalámbrica a personas u objetos en el interior de edificaciones. Para ello, se utilizan diversas tecnologías como la óptica, radio o acústica [10].

Entre las muchas tecnologías que existen comentaremos aquellas más utilizadas e importantes.

- **Wi-Fi:** tecnología de redes inalámbricas basada en el estándar IEEE 802.11 que permite a los dispositivos conectarse entre sí mediante frecuencias radio. El mecanismo de posicionamiento está basado en las mediciones que los puntos de acceso de la red hacen de la potencia de las emisiones de los dispositivos.

  Tiene la ventaja de utilizar frecuencias libres que no necesitan licencia, requiere poca infraestructura, es decir, bajos costes, y la red es sencilla y rápida de configurar. Por el contrario, solo pueden ser localizados aquellos elementos que tengan la capacidad Wireless y son bastante vulnerables a interferencias de otros objetos o dispositivos. Además, la precisión de localización es aún peor que en sistemas GPS, limitada en la mayoría de los casos entre 5 y 10 metros [11].

- **Bluetooth:** tecnología de radio de corto alcance que opera una banda sin licencia que abarca desde de los 2.402 GHz a los 2.480 GHz, permite a los dispositivos transferir datos de punto a punto. La localización de objetos a través de esta tecnología se obtiene midiendo la potencia de las señales, a las cuales se les aplica algoritmos de trilateración o multilateración.

Está especializado en la transferencia entre distancias cortas, con bajo consumo y bajo coste. Posee una precisión de entre 1 y 3 metros, y un alcance de hasta 25 metros en la mayoría de aplicaciones. Sin embargo, la velocidad de transferencia de la información no es muy elevada, en torno a 2 Mbps [12].

- **UWB (*Ultra-WideBand*):** tecnología radio que utilizan un ancho de banda mayor a 500 MHz, funciona emitiendo pulsos cortos de menos de 1ns de duración para medir la distancia entre dispositivos.

  Es capaz de ofrecer velocidad de transmisión entre 400 y 500 Mbps con una precisión de entre 10 cm y 1 metros. La propia naturaleza de estas señales las hace menos susceptibles a interferencias de otros dispositivos [13]. Además, en el caso particular del material que tenemos para el desarrollo de este proyecto, el Grupo de Comunicaciones Ópticas dispone de varios módulos UWB, pero atendiendo al análisis realizado por D. Iván Viloria Vazquez en su trabajo fin de máster de estos módulos, su presión era bastante baja y tenía una frecuencia de refresco también baja, por lo que, aunque quizás existan otras productos dentro de esta tecnología que sí cubran nuestras necesidades, no la vamos a considerar [6].

- **LiDAR (*Laser Imaging Detection and Ranging*):** tecnología que permite conocer la distancia de un elemento utilizando un láser. Funciona emitiendo pulsos de luz cortos del orden de nanosegunos, donde el receptor recoge los fotones reflejados en las diferentes superficies. A través de diferentes filtros y análisis ópticos trata de mejorar la calidad de la señal recibida para calcular dos variables principalmente, el tiempo de ida y vuelta de la señal, así como la intensidad de dicha reflexión. La primera variable permite calcular características como las dimensiones o distancias y la segunda puede ser utilizada para determinar de qué tipo de material se trata.

  Tal y como afirma Dª Carlota en su Trabajo Fin de Grado, esta solución permite alcanzar una precisión entre milímetros y pocos centímetros y es muy robusto frente a la mayoría de variables ambientales [7].

- **Ultrasonidos:** tecnología que basa su funcionamiento en la transmisión de frecuencias de ultrasonidos, normalmente en torno a los 40 KHz. El dispositivo transmite un ultrasonido y mide el tiempo que tarda en regresar la señal reflejada. Estos ultrasonidos no interfieren con el rango auditivo que poseemos los seres humanos.

  Son capaces de detectar objetos que van desde distancias de pocos centímetros a metros. Pudiendo llegar a tener un precio elevado determinados productos, por norma general son más económicos de lo que podrían llegar a tener tecnologías como LiDAR o UWB. Los sensores además son pequeños y sencillos de integrar en diferentes dispositivos y proporcionan una precisión en torno a centímetros. Sin embargo, pueden ser sensibles a múltiples factores como la temperatura y humedad, ya que la velocidad del sonido varía con ellas, la presencia de obstáculos, etc., y poseen un alcance limitado [6].

Realizado el estudio de las diferentes tecnologías existentes en el mercado, y vistos los resultados en el pasado trabajo fin de máster [6], la tecnología de posicionamiento *indoor* que se decidió utilizar para la localización del vehículo en sustitución del GPS fue la de ultrasonidos. Se apostó por los productos de una empresa reconocida en dicho sector,

Marvelmind, debido al compromiso entre precio y precisión que proporcionaban. Más adelante detallaremos más acerca de estos productos.

Destacar que para llevar a cabo la localización en interiores de un determinado objeto podrían no solo utilizarse una de estas tecnologías, sino que se podrían fusionar para conseguir así una mayor precisión y robustez. Este tipo de tecnologías están en constante evolución, por lo que es interesante estar al día sobre ellas.

2.5 Mapas en vehículos autónomos

Durante muchos años, los mapas nos han ayudado a los conductores a tomar mejores decisiones durante la conducción y en el futuro seguirán teniendo aún más importancia para aumentar la seguridad y el éxito de los vehículos autónomos. Sin embargo, aún existen muchos desafíos por resolver en la construcción de mapas, como puede ser el desarrollo de estructuras de almacenamiento eficientes, definir los requisitos mínimos o establecer un estándar internacional.

Un mapa es una representación de las características del mundo real, de los objetos que posee y de su ubicación en un espacio 2D o 3D, es una representación tanto de sus características físicas, como pueden ser edificios o carreteras, y de conceptos no físicos, como la presión del aire o el flujo de tráfico. Su función es ayudar a los vehículos autónomos a comprender la relación entre el vehículo y el entorno, apoyando tanto en la navegación como en la toma de decisiones. Los mapas son capaces de hacer cosas que otros sensores no son capaces, como no fallar en condiciones ambientales desfavorables, se llega a considerar incluso como un sensor adicional. Sin embargo, no son infalibles y pueden presentar problemas de inexactitud o estar desactualizados, además presentan el gran desafío de su transmisión en tiempo real debido al gran tamaño que poseen.

Nos encontramos con tres formas de categorizar la información de mapeado en un sistema de vehículo autónomo, información topológica, geométrica o semántica. Los mapas topológicos proveen la información sobre la conectividad entre las diferentes características geométricas, es decir, la red de carreteras. Los mapas geométricos componen la información del entorno, que pueden ser elementos estáticos como edificios o señales de tráfico, elementos temporales como vehículos estacionados u obras en la carretera, y elementos dinámicos como el movimiento de personas. Los mapas semánticos proporcionan información que permiten interpretar los elementos del entorno como los límites de velocidad o semáforos [14].

Los mapas para máquinas difieren de los diseñados para humanos, ya que contienen información adicional para la toma de decisiones. Cuando hablamos de máquinas necesitamos que cumplan tres requisitos: precisión de centímetros, eficiencia de almacenamiento y usabilidad, es decir, medidas de precisión de ± 10 cm mejorada a través de actualizaciones en tiempo real y una compresión de la información para que pueda ser descargada y actualizada a través de redes inalámbricas.

Los sistemas de almacenamiento de mapas deben ser capaces de manejar volúmenes enormes de datos entrantes y salientes (sobre todo si son especialmente detallados o se basan en nubes de puntos 3D), y ser capaces de manejar múltiples formatos simultáneamente. Aunque los métodos de almacenamiento de datos tradicionales pueden soportar esto hasta cierto punto, actualmente no pueden cumplir con todos los requisitos de la conducción autónoma. Asimismo,

suponiendo que somos capaces de contener todos los datos de manera comprimida, deben estar organizados de manera lógica para que sean rápidos de acceder [14].

La tendencia actual se aleja de los mapas almacenados localmente en los vehículos hacia los mapas almacenados en la nube, lo que permite reducir sustancialmente los requisitos de almacenamiento a bordo y poder obtener la información actualizada y precisa sobre la marcha. De esta forma se evita la necesidad de actualización *off-line* de los mapas y garantiza la consistencia de los datos entre todos los vehículos. Además, se espera que la comunicación vehículo a vehículo e infraestructura a vehículo puedan ayudar a entregar el contenido en un entorno cooperativo donde los datos más populares se almacenen temporalmente y se compartan entre todos los usuarios de la carretera.

La creación de estos mapas puede realizarse a través de los datos de fuentes satelitales, aéreas o terrestres. Para la conducción autónoma el enfoque está en operar a nivel terrestre, donde aparecen los Sistemas de Mapeo Móvil (MMS), sistemas que se componen de sensores como LiDARs, cámaras estéreo o navegación por satélite (Global Navigation Satellite System, GNS) que permiten generar nubes de puntos 3D de alta calidad del entorno. Además, en los últimos años la perspectiva de trabajo ha cambiado hacia un modo cooperativo, donde múltiples vehículos con estos sensores pueden trabajar de manera simultánea para crear los mapas locales y fusionarlos después en un mapa global. Este enfoque colaborativo ha sido adoptado por compañías como HERE o lvl5 [14].

Los algoritmos de mapeado varían en función de las categorías mencionadas de datos. La información topológica se puede obtener de manera sencilla de fuentes de datos de mapas abiertas. Sin embargo, la información geométrica y semántica tienen más complicación.

La información geométrica expresa la forma y posición de los objetos. La técnica más popular es SLAM, la cual permite construir mapas de entornos desconocidos mientras se realiza un seguimiento de la ubicación del agente en el mapa. Esta técnica ha progresado mucho durante las últimas dos décadas, destacando sus variantes con Filtro de Kalman extendido y con Filtro de partículas. Asimismo, también destaca la variante GraphSLAM, que permite manejar grafos avanzados en velocidad y precisión, e incorporar además GPS.

Finalmente, los algoritmos para extraer la información semántica se centran en la detección y reconocimiento de objetos y aunque se han logrado avances significativos en los últimos años con la visión artificial y el aprendizaje profundo, siguen existiendo complicaciones. Se utilizan algoritmos como CNN (Convolutional Neural Network) para detección de señales de tráfico, FCN (Fully Convolutional Network) para la segmentación de carriles o PSPNet (Pyramid Scene Parsing Network) para la segmentación de escenas [14].

Los algoritmos semánticos nos llevan a la introducción del concepto de ontología en el vehículo autónomo. Como veremos a continuación la ontología ayudará al vehículo autónomo a tomar decisiones. La ontología es una forma de organizar y describir conceptos y relaciones importantes en un área específica. Define las palabras y términos necesarios para hablar de estos conceptos y relaciones, y esta información puede ser usada por programas para entender y procesar datos. Las ontologías se basan en dos partes funcionales, la parte terminológica (Terminological Box, TBox) y la parte declarativa (Assertional Box, ABox) [15].

La parte terminológica contiene la definición de todos los conceptos que la ontología busca describir. Se puede comparar con el conocimiento que los seres humanos adquieren a lo largo

de su vida y que utilizan para entender e interpretar el mundo. La TBox de una ontología representa este conocimiento previo y se define mediante la especificación de conceptos, roles y relaciones.

La parte declarativa contiene la definición de las instancias de las clases previamente definidas en la Caja Terminológica (TBox). Estas instancias, comúnmente llamadas individuos, representan datos de la vida real que la ontología busca interpretar.

Un ejemplo del uso de la ontología en vehículos autónomos lo podemos encontrar en Figura 5. Tenemos una serie de objetos que habría que definir, como son "carretera", "cruce", "vehículo", "señal". Una vez que se han definido los objetos, se pueden establecer relaciones entre ellos que son útiles para que el vehículo entienda el contexto de lo que percibe. Por ejemplo se definen conceptos como "estar a la derecha", "estar conectado", "aproximándose a" y así sucesivamente. Este tipo de modelo se ha implementado en la literatura y ha demostrado ser capaz de resolver problemas de coordinación de tráfico entre vehículos autónomos, gestionando conflictos entre vehículos que llegan a la misma intersección.

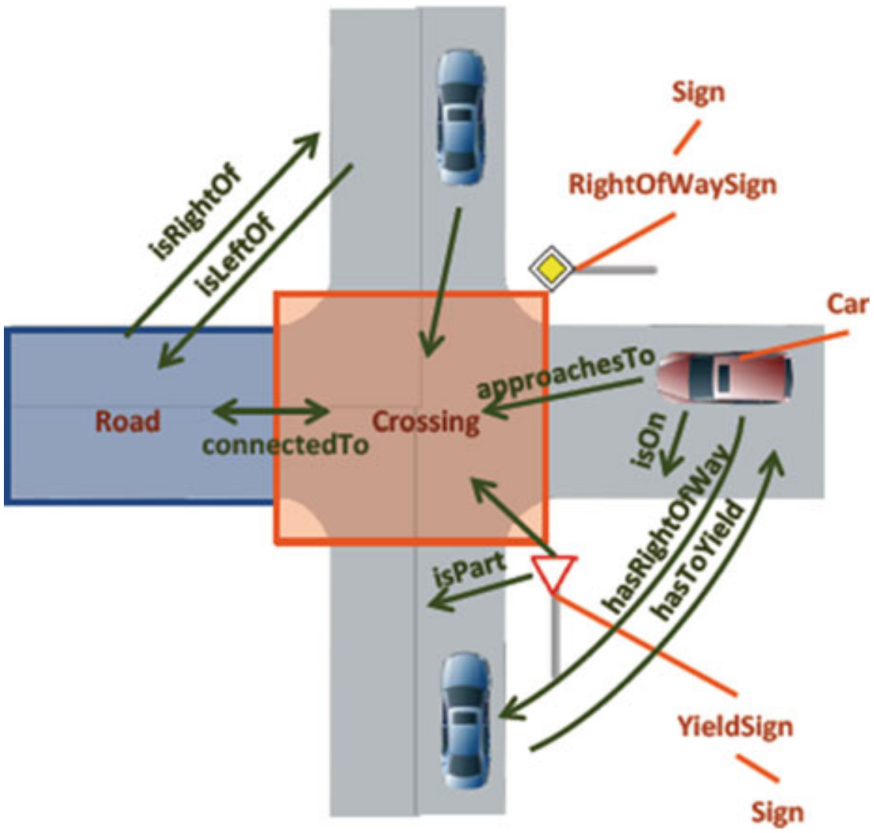

Figura 5. Ejemplo de representación semántica de una intersección. [15]

Para entender como toda la información de los mapas es aprovechada por los vehículos autónomos hay que comprender mejor cómo sus sistemas se organizan. A continuación en la Figura 6 podemos ver los componentes básicos para implementar la conducción autónoma, los cuales se dividen en tres capas: control del vehículo, percepción del entorno y procesamiento con toma de decisiones. Estos componentes deben aparecer en cualquier sistema de conducción autónoma, aunque su implementación requiere un despliegue en *pipeline* [15].

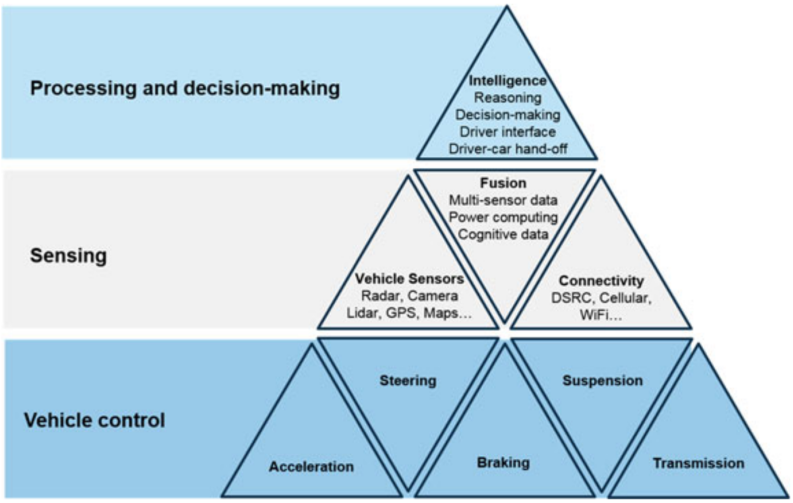

Figura 6. Componentes básicos para la conducción automatizada [15].

Así, como podemos ver en la arquitectura modular de un vehículo autónomo de la Figura 7, los vehículos autónomos son agentes controlados que integran la percepción y el modelado del entorno, la localización y la generación de mapas, la planificación de rutas y la toma de decisiones. A través de sensores como cámaras y radares logran la percepción del entorno, creando modelos detallados que incluyen desde los límites de carril hasta los tipos de vehículos. La localización se logra mediante la combinación de mapas locales y globales, permitiendo al vehículo conocer su posición exacta y los detalles del entorno.

La planificación de rutas y toma de decisiones asegura que el vehículo opere de manera segura y eficiente. Incluye la planificación estratégica de rutas globales, la selección táctica de carriles y la reacción inmediata a obstáculos mediante planificación de rutas locales. La toma de decisiones abarca el razonamiento adaptativo y la planificación de misiones basada en nuevas observaciones. Finalmente, el control del movimiento del vehículo gestiona tanto la dirección lateral como la velocidad longitudinal, asegurando así que se cumplan los objetivos de manera precisa durante toda la operación [15].

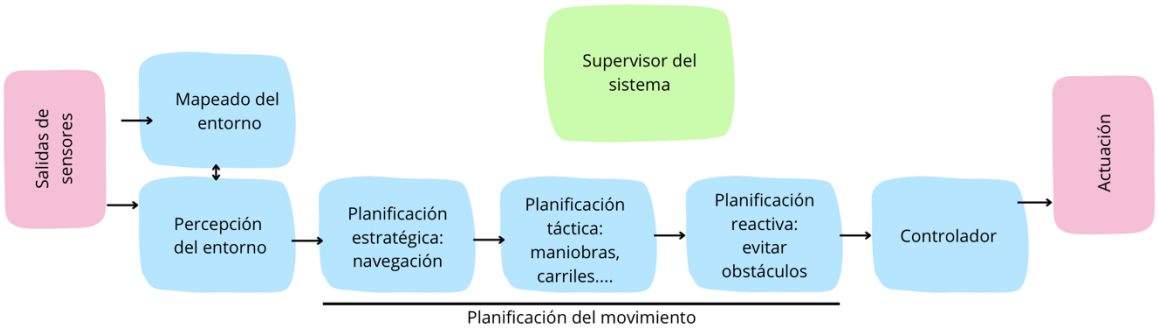

Figura 7. Arquitectura modular de un sistema de vehículo autónomo.

De todas estas funciones, actualmente el proyecto TwizyLine apenas ha comenzado a desarrollar la planificación de movimiento y algunas tomas de decisión en bifurcaciones, como se pudo ver en la sección 2.2. Dado que no es el objetivo de este proyecto resolver el nivel más alto de abstracción en el control de vehículo, sino asegurar el control de su trayectoria dada una ruta concreta, a nivel de trabajo fin de máster nos interesa saber cómo representar esta

información de la manera más sencilla posible. En la literatura podemos encontrar dos aproximaciones a este problema concreto: lanelets y mapas de ocupación.

Un lanelet describe un segmento de carril que se caracteriza por sus límites izquierdo y derecho. Estos límites son polilíneas y permite obtener una aproximación precisa de las geometrías del carril. Los límites izquierdo y derecho definen un área amplia dentro de la cuál el vehículo puede moverse libremente. Podríamos preguntarnos por qué no definir únicamente una lína central, pero conocer solamente la línea central del carril no sería suficiente, ya que la trayectoria debería poder ajustarse según la velocidad y además no determina de manera unívoca el área en el cuál puede moverse libremente el vehículo, dado que un error admisible a cada lado de la línea es un parámetro que puede variar de punto a punto. Por contra, la definición final de la trayectoria que ha de seguir el vehículo requiere mayor inteligencia y puede ser más difícil de gestionar.

Nos puede interesar en un momento determinado que hay incorporaciones o bifurcaciones en un camino. Para ello se debe definir el concepto de lanelets adyadentes. Dos lanelets A y B se consideran adyacentes si los puntos de inicio o fin son idénticos, y si la curvatura de las polilíneas determinada por los bordes derechos e izquierdo está por debajo de un umbral determinado. Podemos ver un ejemplo de adyaciencia en la Figura 8 [16].

Asimismo, la longitud de un lanelet puede variar desde pocos metros a cientros de metros y se pueden concatener para formar corredores de conducción. Vale la pena mencionar que muchos de los investigadores que trabajan con lanelets utilizan librerías y recursos relacionados con OpenStreetMap (OMS), un proyecto colaborativo y una base de datos geoespacial de código abierto que permite crear, editar y utilizar mapas libres y gratuitos.

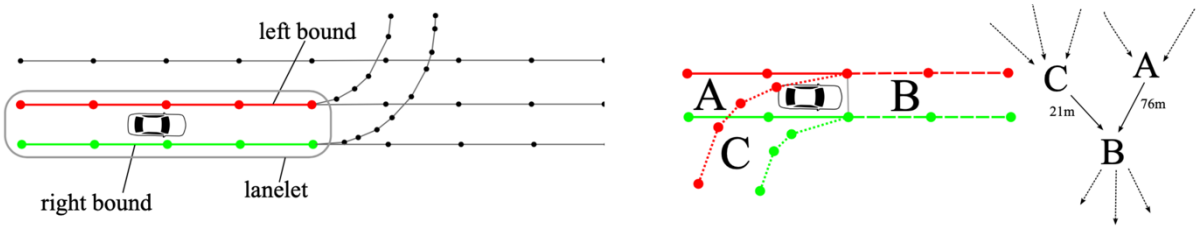

Figura 8. Mapa compuesto de lanelets (izquierda) y ejemplo de adyacencia (derecha) [16].

Los mapas de ocupación representan una idea similar a los lanelets en el sentido de que definen un área dentro de la cual los vehículos pueden moverse libremente. Si acaso, lo mapas de ocupación tienen como ventaja sobre los lanelets que permiten localizar los obstáculos de manera más precisa, ya sean fijos o móviles, permitiendo una mejor respuesta por parte del vehículo a los mismos.

Los mapas de ocupación por rejilla requieren una estructura de datos adecuada para almacenar las celdas de la rejilla, ya que el objetivo no es crear un mapa completo y consistente de todo el entorno, sino más bien crear una representación precisa de los alrededores inmediatos del vehículo para la toma de decisiones y planificación del movimiento. Asimismo, cuando se vuelve a visitar un área, es importante recrear la parte correspondiente del mapa porque el entorno puede haber cambiado, permitiendo tener unicamente un mapa detallado de unos pocos cientos de metros y mover esta sección junto con el vehículo [17].

El principal problema de los mapas de rejilla es encontrar un enfoque para manejar los mapas debido a su alto consumo de recursos. Por otra parte, dado que este algoritmo está más centrado en representar el área en el cual un vehículo puede moverse, no es tan sencillo incorporar la información de reglas de tráfico.

En el caso particular de este trabajo, en vista de cómo el vehículo decide su movimiento, nos interesará una solución más cercana a los lanelet, cuya implementación particular se verá en el Capítulo 4.

Capítulo 3. Localización *indoor*: Marvelmind

3.1 Introducción

En el presente capítulo podremos encontrar una explicación completa de las características y funcionalidades de las herramientas utilizadas para la localización del vehículo, así como unas primeras pautas para aprender a trabajar con ellas y las complejidades que las conciernen.

En segundo lugar, nos encontraremos con un análisis del escenario utilizado para desplegar el sistema de localización *indoor*, conociendo sus límites y posibilidades. Asimismo, se explicará qué métodos se han utilizado para poder adaptar el sistema a dicho escenario.

Finalmente, atenderemos a cómo se ha preparado el escenario y el sistema para poder realizar tres pruebas diferentes, en las cuales se simuló la localización de un vehículo a través de un carro de pruebas, debido a que el vehículo no se encuentra aún operativo. Además, lleva a cabo un análisis de distintas estadísticas extraídas de las pruebas realizadas que nos permitirán conocer el rendimiento del sistema.

3.2 Descripción del sistema de Marvelmind

Marvelmind es una compañía que se dedica al desarrollo de productos de localización *indoor*. Actualmente utiliza dos tecnologías: ultrasonidos y Banda Ultra Ancha (UWB). Estas tecnologías no son compatibles entre sí, y por lo tanto hay que elegir una de ellas.

Los sistemas elegidos para este proyecto de Marvelmind funcionan a través de la tecnología de ultrasonidos y un sistema de balizas, que a nivel comercial se denominan 'beacons'. Los *beacons* son balizas fijas cuya localización en el mapa es conocida a priori. El sistema funciona básicamente de la siguiente manera: a través de la medida del retardo de señales de ultrasonidos emitidas y recibidas por balizas fijas o móviles se puede calcular la posición de los objetos por un proceso de trilateración (no confundir con triangulación). La información obtenida de los ultrasonidos es transmitida por radiofrecuencia a un módem que comunica las balizas con el *software* desarrollado por Marvelmind a través de la frecuencia pública 868 MHz, permitiéndole así conocer su posición mediante un cálculo de trilateración usando el retardo de dichas señales de ultrasonidos.

Por lo tanto, para desplegar un sistema de posicionamiento *indoor* de Marvelmind basado en ultrasonidos necesitamos: un conjunto de balizas fijas ubicadas en diversos puntos del área dentro de la cuál queremos localizar el móvil, al menos una baliza por cada móvil que tengamos dentro del área designada, un módem para poder configurar las balizas y mantenerlas en contacto unas con otras, el *software* que permite gestionar la configuración y otras funciones asociadas con la localización como establecimiento de mapas, situación dentro del mapa de las balizas que hacen de *beacons*, etc.

De entre su catálogo de productos se escogieron las balizas dedicadas a la localización en interiores Super Beacon Indoor. Se adquirieron las balizas Super Beacon versión 2 y versión 3, además del módem que las presta servicio, módulos que pasaremos a explicar en detalle a continuación.

3.2.1 Descripción balizas Super Beacon Indoor

Las balizas Super Beacon utilizan señales de ultrasonidos para calcular la posición de un dispositivo, como pudiera ser una persona, un drone o un vehículo. Las balizas de tipo Super Beacon son capaces tanto de transmitir las señales de ultrasonidos, como de recibirlas.

Disponemos de dos modelos Super Beacon, la versión 2 y la versión 3, entre ellos poseen diferencias. demás de tener un procesador diferente, la segunda versión utiliza un micrófono analógico, mientras que el micrófono de la tercera versión es digital. Soportan distintas bandas radio para transmitir y recibir los resultados de las mediciones de las señales ultrasónicas, la segunda versión soporta las bandas de 433 MHz, 868 MHz y 915 MHz, frente a la tercera versión, que únicamente soporta las dos últimas bandas mencionadas. Sin embargo, dichas diferencias no van a suponer ningún problema de cara a su interoperabilidad [18], dado que hay que elegir aquella banda que es de uso público, que en la UE son las bandas de 433 MHz y 868 MHz.

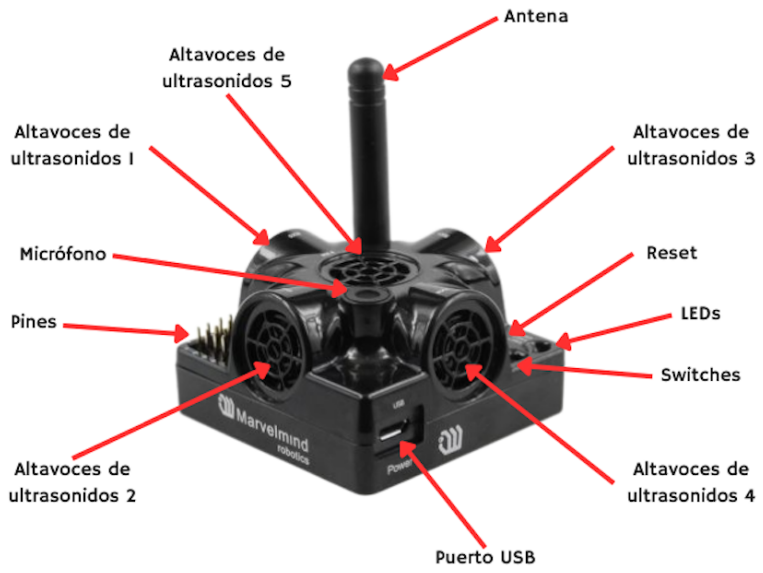

Figura 9. Partes de una Baliza Super Beacon de Marvelmind

Las balizas Super Beacon están compuestas por diferentes elementos, tal y como podemos observar en la Figura 9, detallados a continuación:

1. **Altavoces de ultrasonidos (un total de 5)**: utilizados para transmitir las señales ultrasónicas necesarias para obtener la localización de los receptores.
2. **Micrófono**: utilizado para recibir señales ultrasónicas.
3. **Antena**: utilizada para transmitir y recibir señales de ultrasonidos.
4. **Diferentes pines**: pueden ser utilizados para conectar dispositivos externos como un micrófono adicional o un ordenador.
5. **Puerto USB**: utilizado para cargar la baliza y para conectar ésta al ordenador.
6. **LEDs**: indicaciones básicas.
    a. LED 1: luce mientras se está cargando la baliza.

b. LED 2: parpadea con la frecuencia de actualización durante la realización del seguimiento, y parpadea cada un determinado tiempo para indicar que la baliza se encuentra encendida.
c. LED 3: se encuentra reservado para futuras funcionalidades.

7. **Switches (un total de 2)**: para encender o apagar las balizas, activar o desactivar el modo de ahorro de energía, para cargar las balizas e instalar *firmware*.
8. **Botón de Reset**: restablecer todos los valores de la baliza a los predeterminados de fábrica.

Destacar que, para un correcto funcionamiento de las balizas, se recomienda que se encuentren cargadas adecuadamente con un valor entre 3.85 y 4.25V.

Además de todos estos elementos, visibles a simple vista, en su interior cuenta con el procesador, batería, memoria y una IMU, la cual está compuesta por un acelerómetro, un magnetómetro y un giroscopio de 3 ejes, combinando estos dos últimos elementos es capaz de calcular la orientación de la baliza.

Parte fundamental de estas balizas consiste en comprender cuál es su diagrama de emisión de ultrasonidos, puesto que habrá que tenerlo en cuenta a la hora de diseñar el escenario de seguimiento del vehículo. Es necesario conocerlo para poder garantizar la comunicación entre las balizas y optimizar su colocación. Dado que las balizas pueden actuar tanto como balizas transmisoras o receptoras, tendremos diferentes diagramas de emisión y recepción, disponibles en la Figura 10 y Figura 11 respectivamente.

En el diagrama de emisión de las balizas podemos observar que los lóbulos son debidos a los cinco altavoces de ultrasonidos. Dichos altavoces emiten ondas en todas direcciones, pero su intensidad disminuye conforme aumenta el ángulo. La presencia de lóbulos máximos en la dirección que apuntan los altavoces, indica que la baliza emite la señal con una intensidad superior en dicha dirección.

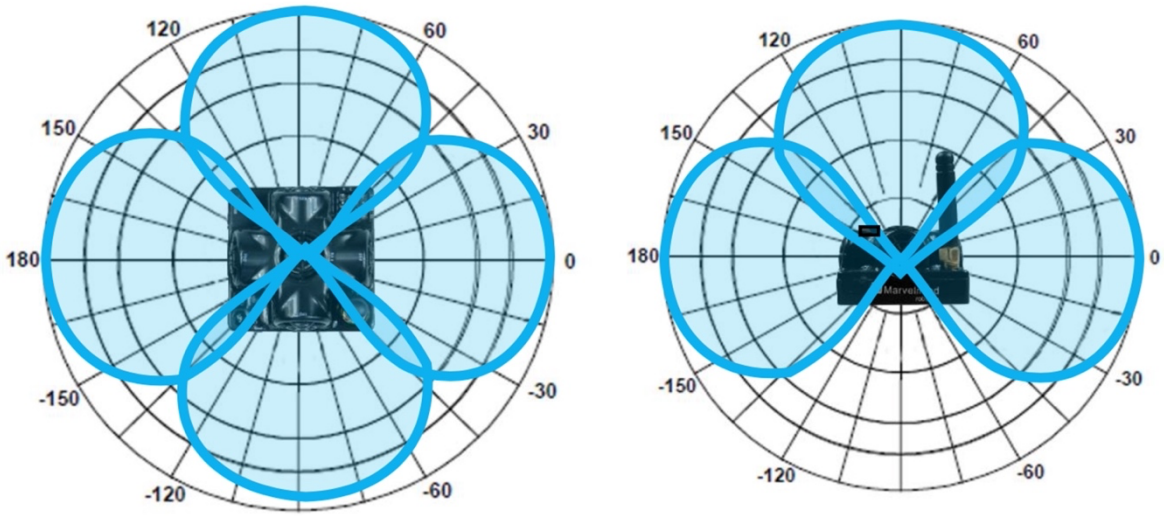

Figura 10. Diagrama de emisión de las balizas Super Beacon.

El diagrama de recepción de las balizas difiere del ya mencionado, dado que no viene determinado por los altavoces de ultrasonidos, sino por el micrófono. En este observamos que

el diagrama de radiación presenta un patrón completamente omnidireccional en el plano XY, esto es, si es visto desde la parte superior de la baliza, lo que facilita la localización del vehículo desde cualquier dirección. Sin embargo, cuando lo observamos desde su perfil, como podría ser un plano XZ o YZ, muestra un patrón direccional, lo cual quiere decir que cuando la baliza actúa como receptora, es más sensible a las señales recibidas desde la perpendicular al eje formado por el micrófono, es decir, las señales las recibirán de mejor forma cuando vengan desde su parte superior [19].

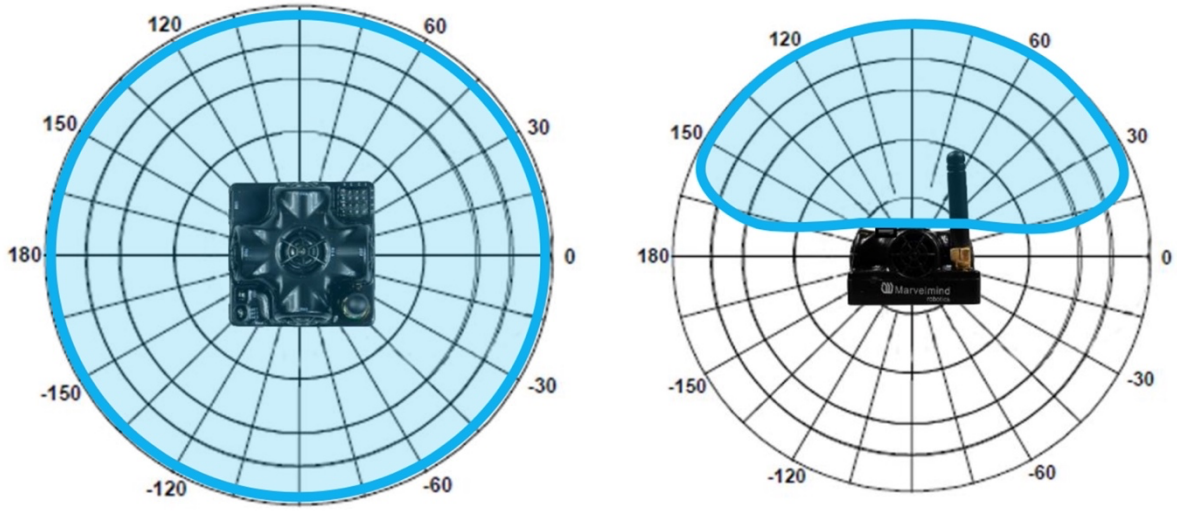

Figura 11. Diagrama de radiación de las balizas Super Beacon cuando actúan como receptoras.

Cabe destacar que las balizas SuperBeacon también incorporan una IMU (Inertial Measurement Unit) que puede ser utilizada para mejorar la localización del móvil. Sin embargo, actualmente la empresa no ha desarrollado ningún *software* o *firmware* para realizar la fusión de los datos, por lo que será una característica que no se utilizará en nuestro sistema.

3.2.2 Descripción Modem

El modem del que disponemos es la versión HW v5.1, y tiene una doble función. Por un lado, es el responsable en enviar y recibir datos y mensajes desde las balizas al Dashboard de Marvelmind para coordinar la puesta en marcha del sistema y luego mantenerlo activo. Por otro lado, éste es el elemento encargado de recibir vía radio las medidas realizadas de las distintas señales de ultrasonidos recibidas por las balizas y convertirlas en datos que puedan ser procesados y mostrados en el Dashboard.

Por lo tanto, se comunica con las balizas desplegadas para obtener distintos datos, como la distancia entre balizas, esencial para poder crear el mapa en primera instancia. También se utiliza para calcular los tiempos de llegada de cada señal, para así obtener la ubicación del objeto móvil.

Tal y como podemos observar en la Figura 12, el modem HW v5.1 está compuesto por diferentes elementos:

1. **Antena**: utilizada para transmitir y recibir señales de ultrasonidos.

2. **Pines**: pueden ser utilizados para conectar dispositivos externos, los hay de alimentación y de datos.
3. **Modo de funcionamiento (también llamado SW2)**: se utiliza para cambiar el modo de funcionamiento, entre modo balizas o modo rastreador.
    a. Modo Balizas: se utiliza para transmitir los datos de ubicación y orientación del propio modem de manera periódica.
    b. Modo Rastreador: se utiliza para seguir la señal de una baliza.
4. **Puerto USB**: utilizado para conectarlo al ordenador.
5. **LEDs**: para indicaciones básicas.
    a. LED verde: se utiliza para indicar el estado del modem. Si parpadea muy rápido indica que se encuentra transmitiendo datos, si parpadea lentamente indica que el modem está recibiendo datos, y si parpadea a un ritmo normal indica que el modem está encendido y listo para funcionar.
    b. LED rojo: se utiliza para indicar que se ha producido algún error.
6. **Botón de Reset**: restablecer todos los valores de la baliza a los predeterminados de fábrica, para lo cual es necesario pulsarlo durante 10 segundos.

Además, a través de la interfaz USB, así como a través de los pines, puede enviar los datos al ordenador o al dispositivo al que se encuentre conectado, y aparte de todos los elementos ya mencionados, posee en su interior el procesador y la memoria.

Figura 12. Partes de un Modem HW v5.1

3.3 Puesta en marcha de las balizas de ultrasonidos

Antes de poder desplegar las balizas es necesario conocer varios elementos, en primer lugar, se requiere instalar un *software* desarrollado por la propia empresa y conocer su funcionamiento. Asimismo, cada baliza posee un *firmware* que controla su funcionamiento, el cual es necesario instalar/actualizar, donde además habrá que tener en cuenta qué arquitectura de comunicación queremos que utilicen las balizas.

3.3.1 Instalación del *software* de Marvelmind

Para poder poner en funcionamiento las balizas, debemos conocer primero las herramientas disponibles para su control y puesta en marcha. La compañía Marvelmind nos facilita la posibilidad de poder instalar su *software* de control en dos plataformas distintas, Windows y

Linux. Dado que en la escuela disponemos de un ordenador Linux, como ya se mencionó en la sección referente a los recursos disponibles, procederemos con la instalación en dicha plataforma.

Al ser una tecnología en continuo desarrollo, la empresa actualiza de manera frecuente su *software*, disponible en su propia dirección web de descargas https://marvelmind.com/download/. En el quinto apartado de dicha página nos encontraremos con la sección '*Install the latest software pack'*, en la cual podremos estar informados de todas las versiones de actualización, su fecha de lanzamiento y las innovaciones que traen. Independientemente de nuestro sistema operativo nos debemos descargar el paquete *software*, ya que en su interior se encuentra el soporte para ambas plataformas mencionadas.

En nuestro caso, queremos instalar la aplicación Dashboard referente a '*Common Indoor Positioning*'. El propio paquete *software* pone a nuestra disposición una sencilla guía de instalación en la carpeta de documentación, consultando ésta nos daremos cuenta de que existen tres versiones diferentes para instalar en Linux. Nos encontramos con:

1. **ARM32**: versión para ordenadores *single-board*, como una Raspberry Pi, con arquitectura de procesador de 32 bits.
2. **ARM64**: versión para ordenadores *single-board*, como una Raspberry Pi, con arquitectura de procesador de 64 bits.
3. **X86**: versión para el resto de los ordenadores con procesador Intel o AMD.

Siguiendo los pasos de instalación, en nuestro caso debemos copiar la información de la carpeta '*X86*' en el directorio en el que nos interese instalar Marvelmind. Entre dicha información nos encontraremos con dos directorios denominados '*defaults'* y '*profiles'*, junto con 3 ficheros llamados '`dashboard_x86`', '`libdashapi.so`' y '`versions_table.ini`'.

Seguidamente deberemos copiar la librería de la aplicación en el directorio protegido con permisos de superusuario '*usr/local/lib*'. La razón por la que dicha carpeta se encuentra protegida es porque contiene todas las bibliotecas compartidas de cualquier programa, por ello podemos utilizar el comando `sudo cp libdashapi.so /usr/local/lib` para copiar la información. Sin embargo, para que el enlazador sea capaz de encontrar la biblioteca debemos actualizar la caché de enlace, que se puede hacer mediante `sudo ldconfig`.

Además, es necesario proveer al usuario de los permisos necesarios para poder acceder a los puertos serie del ordenador, para poder interactuar así con las balizas conectadas a través del puerto USB. Podemos hacerlo mediante el comando `sudo adduser <usuario> dialout`, donde en caso de no conocer cuál es el nuestro nombre usuario para el sistema, podemos conocerlo a través del comando `whoami`.

Finalmente, especificaremos los permisos disponibles para el puerto USB serie '`ttyACM0`'. Para ello es necesario crear un fichero llamado '`99-tty.rules`' en el directorio '*/etc/udev/rules.d*' con el siguiente contenido:

```
# Marvelmind serial port rules
KERNEL=="ttyACM0",GROUP="dialout",MODE="666"
```

Lograremos así especificar los permisos necesarios. El campo `KERNEL` especifica el nombre del puerto con el que se van a asociar las reglas, el campo `GROUP` especifica el grupo al que se

le asignarán dichos permisos, y el campo `MODE` es donde se establecen los permisos, en este caso permisos de lectura y escritura para todos los usuarios del grupo. En caso de no determinar estas reglas, solamente tendría acceso el usuario *root*.

Finalmente establecemos los permisos necesarios para el fichero ejecutable, a través de `sudo chmod 0777 ./dashboard_x86` y podremos lanzar la aplicación mediante `./dashboard_x86`, donde abriremos una pestaña con el aspecto que vemos en la Figura 13.

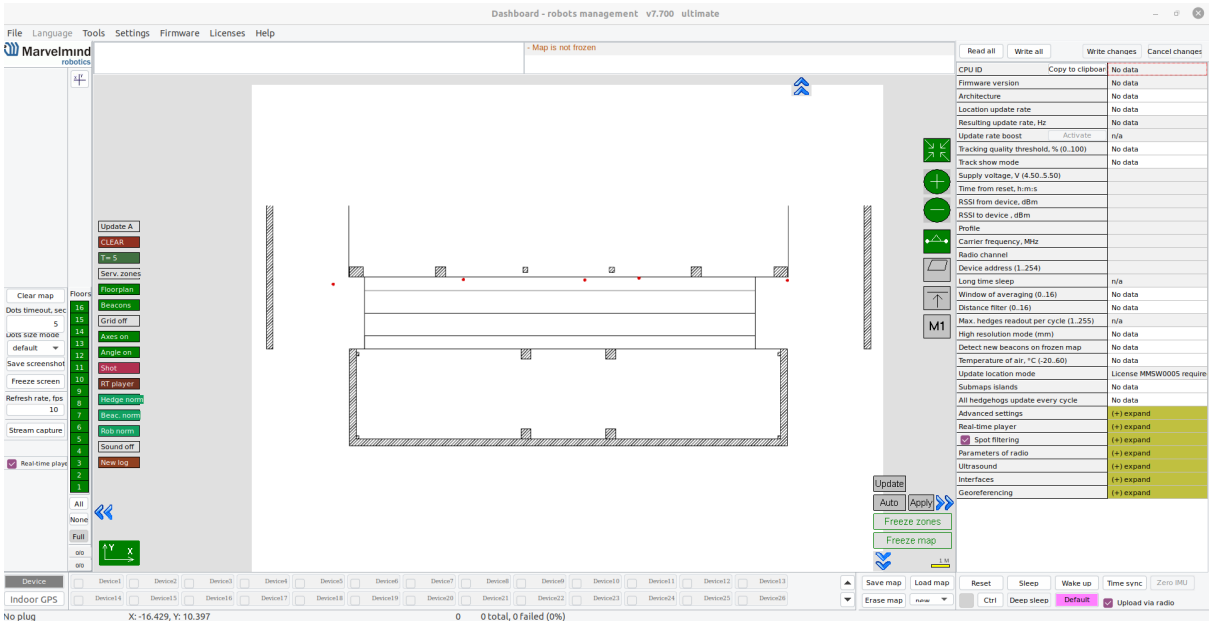

Figura 13. Ejemplo de la aplicación Dashboard de Marvelmind.

Si quisiéramos ahorrarnos este tedioso proceso para abrir la aplicación, podríamos simplemente modificar sus permisos desde el panel de propiedades del fichero ejecutable, activando la casilla 'Permitir y ejecutar el archivo como un programa' para abrirlo como una aplicación normal.

3.3.2 Arquitecturas disponibles

Existen diferentes tecnologías de comunicación y formas de medir la localización de los objetos a través de las balizas, también llamadas arquitecturas, es decir, existen distintas formas de despliegue o arquitecturas. Nos encontramos con tres opciones:

1. ***Non-Inverse Architecture*** **(NIA)**: basa su funcionamiento en que las balizas móviles son las que emiten las señales de ultrasonidos y las balizas fijas las que reciben las señales para calcular el tiempo que tarda la señal en llegar a la baliza. Se transmite en una única frecuencia.
2. ***Inverse Architecture*** **(IA)**: basa su funcionamiento en que las balizas fijas son las que emiten las señales de ultrasonidos y las balizas móviles las que reciben dichas señales para calcular el tiempo que tarda la señal en llegar a la baliza. Se transmite en múltiples frecuencias.
3. ***Multi-Frequency NIA*** **(MF-NIA)**: su funcionamiento es el mismo que el de NIA, pero pudiendo transmitir en múltiples frecuencias.

En función de nuestras necesidades, deberemos seleccionar una de ellas, ya que de la arquitectura seleccionada dependerá el *firmware* que instalaremos en las balizas. Para poder entender de manera más clara las diferencias entre éstas, detallaremos las distintas características que proporciona la empresa de cada una de ellas, tal y como se ve en la Tabla 1.

	NIA	**IA**	**MF-NIA**
Uso típico	Localización de 1 a 4 objetos móviles. Cuando la baliza móvil se instale en un lugar ruidoso, pero las balizas fijas estén en lugares más silenciosos.	Localización de muchos objetos móviles (típicamente personas). Cuando las balizas móviles se encuentran en lugares silenciosos.	Localización de 5 a 16 objetos móviles. Cuando la baliza móvil se instale en un lugar ruidoso, pero las balizas fijas estén en lugares más silenciosos.
No se recomienda	En aplicaciones donde la emisión de ultrasonidos por parte del móvil no es deseable.	En aplicaciones donde el móvil sea muy ruidoso, como un dron.	En aplicaciones donde la emisión de ultrasonidos por parte del móvil no es deseable.
Precisión	± 2cm	± 2cm	± 2cm
Tasa de refresco	Depende del número de balizas móviles, del tamaño del submapa y del perfil radio.	No depende del número de balizas móviles, depende del perfil radio y del tamaño del submapa.	Depende del número de balizas móviles sin son más de 8, del tamaño del submapa y del perfil radio.
Rango	De hasta 30 metros.	De hasta 30 metros o menos en situaciones concretas.	De hasta 30 metros.
	Se pueden cubrir territorios grandes mediante la utilización de submapas.		
Creación de mapas	Se pueden crear submapas de manera automática y manual.		

Tabla 1. Comparación entre las arquitecturas disponibles para las balizas.

Analizando las distintas alternativas y planteando el problema a resolver en el proyecto, la localización del Renault Twizy en una situación *indoor*, podemos darnos cuenta de que la arquitectura más conveniente es NIA. Esta arquitectura es idónea cuando se quiere realizar el seguimiento de pocos objetos, como es nuestro caso. Además, aunque el vehículo no sea especialmente ruidoso, tampoco es despreciable el poco ruido que éste pueda tener.

Dado que llegado el momento necesitaremos calcular el ángulo de *yaw* del vehículo, necesitaríamos incluir un micrófono externo o una baliza nueva. Esto repercutiría de manera negativa en el alcance si utilizásemos la arquitectura IA.

Las balizas son sistemas complejos, donde además de administrarnos la ubicación de un determinado objeto, llegado el momento podríamos llegar a obtener otros datos del vehículo como pueden ser los ángulos de *yaw*, *pitch* y *roll*, también conocidos como guiñada, cabeceo y balanceo, tal y como podemos observar en la Figura 14.

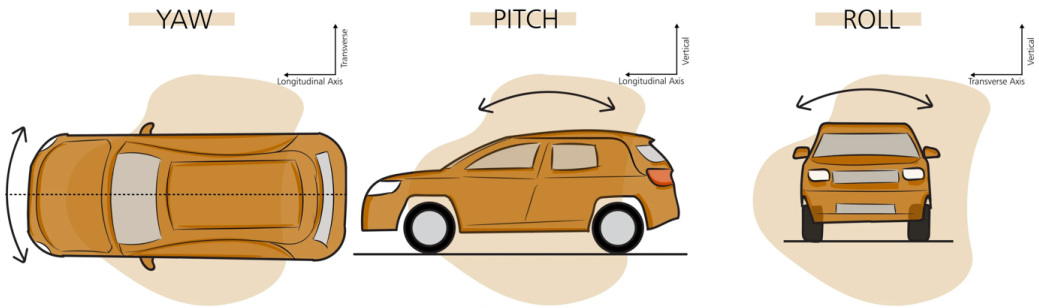

Figura 14. Distintas orientaciones en un vehículo.

Para poder obtener estos datos adicionales se requiere hacer uso de otros elementos además de las balizas, sería necesario conectar a la baliza ubicada en el vehículo dos micrófonos externos. Por lo que debido a la propia naturaleza de funcionamiento de la arquitectura IA, la cual se basa en la transmisión de ultrasonidos por parte de las balizas fijas y recepción de las balizas móviles, se necesitarían tres canales independientes en la baliza. Esto reduciría la distancia máxima en una razón de tres, disminuyendo así la distancia máxima de 30 metros a 10 metros.

Si únicamente quisiéramos obtener el ángulo de *yaw* o guiñada, la característica más útil en nuestra aplicación se podría lograr mediante el emparejamiento de balizas. La empresa finlandesa permite emparejar hasta dos Super Beacon con la finalidad de obtener la guiñada, es decir, el ángulo de rotación del vehículo con respecto a su eje vertical [20].

Entonces, dada la naturaleza del proyecto, donde solamente debemos localizar un único objeto cuyo sonido se estima suficientemente bajo y que vamos a trabajar en un área considerablemente grande, nos decantaremos por la arquitectura NIA. Además, debido que los datos de *pitch* y *roll* no nos interesan en esta aplicación, obtendremos el ángulo de *yaw* a través del emparejamiento de dos balizas en vez de incorporar elementos externos.

3.3.3 Instalación/Actualización del *firmware* de las balizas

En el momento de recibir las balizas, ya sean procedentes de fábrica o de otra persona que las haya manipulado, debemos instalar el *firmware* adecuado en las balizas y modem. Existen dos formas de actualizar cada elemento, una es a través del puerto USB y otra vía radio. Sin embargo, nos centraremos en la forma de hacerlo mediante USB, ya que es considerablemente más veloz y sencilla.

En primer lugar, darnos cuenta de que las balizas poseen dos *switches*, uno dedicado al encendido y apagado, y otro dedicado a activar o desactivar el modo de ahorro de energía. Podemos visualizar dichos botones en la Figura 15. Para poder llevar a cabo la instalación debemos posicionarlos de la forma indicada en la Figura 16, entonces el LED3 comenzará a parpadear en verde.

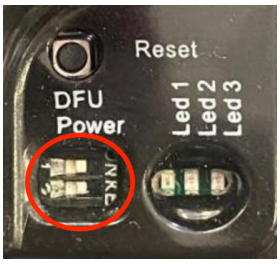

Figura 15. Switches que permiten cambiar entre modos de funcionamiento en las balizas.

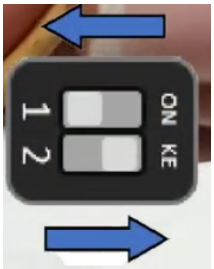

Figura 16. Switches en modo instalación de software.

Estaremos así en disposición de poder actualizar el *firmware*, para ello procedemos de la siguiente manera:

1. Lanzamos la aplicación Dashboard de Marvelmind.
2. Conectamos la baliza o modem a través del cable USB a Micro USB.
3. Seleccionamos 'Firmware' > 'Upload Firmware'.
4. Elegimos el archivo '.hex' correspondiente.
   En nuestro caso, para las balizas Super Beacon éste se encuentra en '*01_Common_Indoor_positioning_SW*' > '*04_Super_Beacon*' > '*NIA_2023_09_05_Super_Beacon_915.hex*'.
   Para el modem, el fichero se encuentra en '*01_Common_Indoor_positioning_SW*' > '*02_Modem_hw51*' > '*NIA_2023_10_31_Modem_hw51_915.hex*'.
   Prestar especial atención en no equivocarnos en seleccionar un fichero de la arquitectura equivocada, ya que se llaman de forma muy parecida.
5. Seleccionar 'Next' y esperar a que la instalación termine.
6. Terminada la instalación, presionar el botón default que observamos en la Figura 17, teniendo seleccionada la baliza en el menú.
7. Repetir de nuevo con cada elemento.

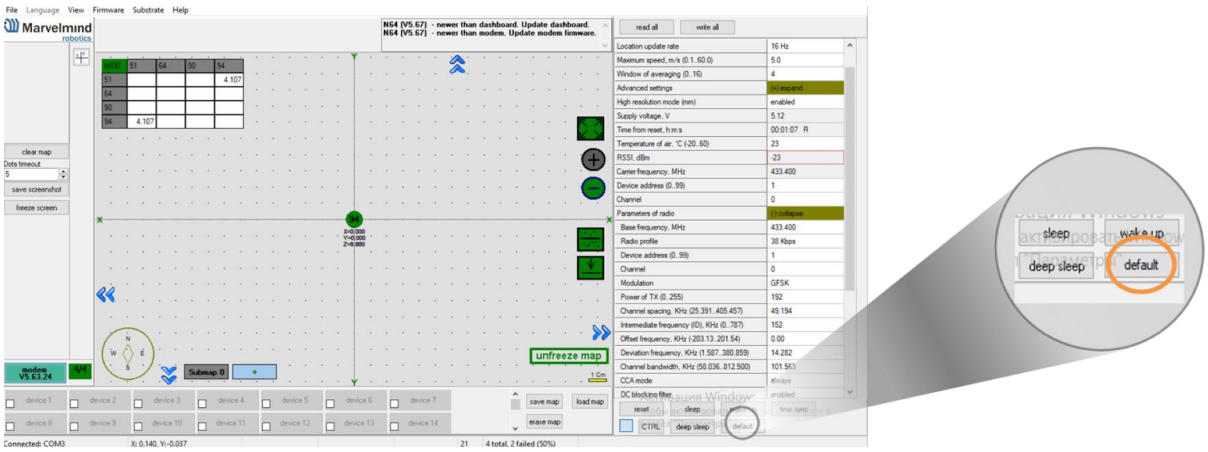

Figura 17. Botón 'Default' para poner los valores por defecto a las balizas.

3.4 Despliegue de las balizas de ultrasonidos

En esta sección se examinará el escenario en el cual se ha llevado a cabo el proyecto, centrado en un aparcamiento de coches considerablemente grande, implicando además el levantamiento de planos a través de AutoCAD para poder abordar la disposición de elementos y la configuración del sistema.

Se abordará la disposición de las balizas con el objetivo de maximizar la cobertura, por ello en primera instancia, se hará un análisis de una situación ideal con el objetivo de comprender mejor el escenario en el que nos encontramos. Posteriormente se realizará un estudio real de las sombras producidas por el entorno del aparcamiento, adaptándonos así a las limitaciones del escenario.

Asimismo, se detallará la configuración de las balizas junto los parámetros escogidos para lograr un funcionamiento óptimo. Analizaremos los *floorplans* y submapas, viendo así su gran

utilidad, y exploraremos en detalle los ficheros ‘log’ generados por el sistema de Marvelmind con la localización de las balizas.

3.4.1 Escenario

El escenario disponible para llevar a la práctica el proyecto es el aparcamiento de nuestra propia facultad, la Escuela Técnica Superior de Ingeniero de Telecomunicación de la Universidad de Valladolid. Se trata de un aparcamiento subterráneo de una única planta con una superficie de más de 4.900m^2, pero debido a su elevadísimo tamaño, se limitó el área a una zona concreta del aparcamiento con 1.280 m^2.

Previo a realizar el despliegue de las balizas, se requiere realizar un estudio del escenario para ver cómo optimizar la disposición de los elementos. Luego, para poder llevar a cabo cálculos teóricos acerca de la superficie, fue necesario realizar el levantamiento del plano.

El levantamiento del plano se diseñó en dos fases, una primera en la cual se hizo un esbozo a mano alzada del perímetro, y una segunda fase en la cual se tomaron las diferentes medidas con la ayuda de un metro laser y una cinta métrica. Posteriormente se dispusieron todas las columnas del lugar, con sus diferentes tamaños, y todas las plazas de aparcamiento disponibles, para delimitar así el terreno que se podrá utilizar para las pruebas con el Renault Twizy [21].

El levantamiento del plano fue un poco laborioso, debido a que se encontraban vehículos aparcados en todo momento. Además, las medidas tomadas eran dispares y asimétricas, debido a que en general las construcciones de edificios de esta clase no se realizan con técnicas de gran precisión, sino que se admiten desvíos de varios centímetros con respecto al plano original. Por lo tanto, para realizar el plano se han aproximado las medidas para que todos los elementos en el mismo estuvieran perfectamente alineados y rectos, haciéndolo más fácil de interpretar, despreciando por lo tanto los errores de alineamiento que normalmente eran de ± 2cm.

Para trasladar finalmente todas las anotaciones a un plano digital, fue necesario adquirir conocimientos básicos de la popular herramienta de diseño asistido de planos 2D y 3D AutoCAD. Finalmente, el plano resultado es el que podemos encontrar en la Figura 19. En éste se ha delimitado una zona en rojo, un rectángulo por el cual se espera poder llegar a mover el vehículo libremente. Además, podemos observar cómo se ve el aparcamiento en tres dimensiones en la Figura 18, donde también se ha tenido en cuenta la altura al techo.

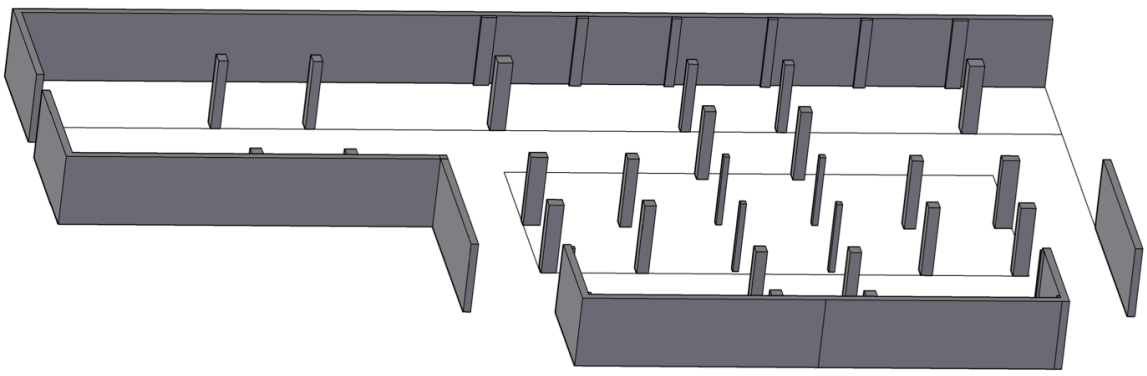

Figura 18. Diseño en 3D con AutoCAD del aparcamiento.

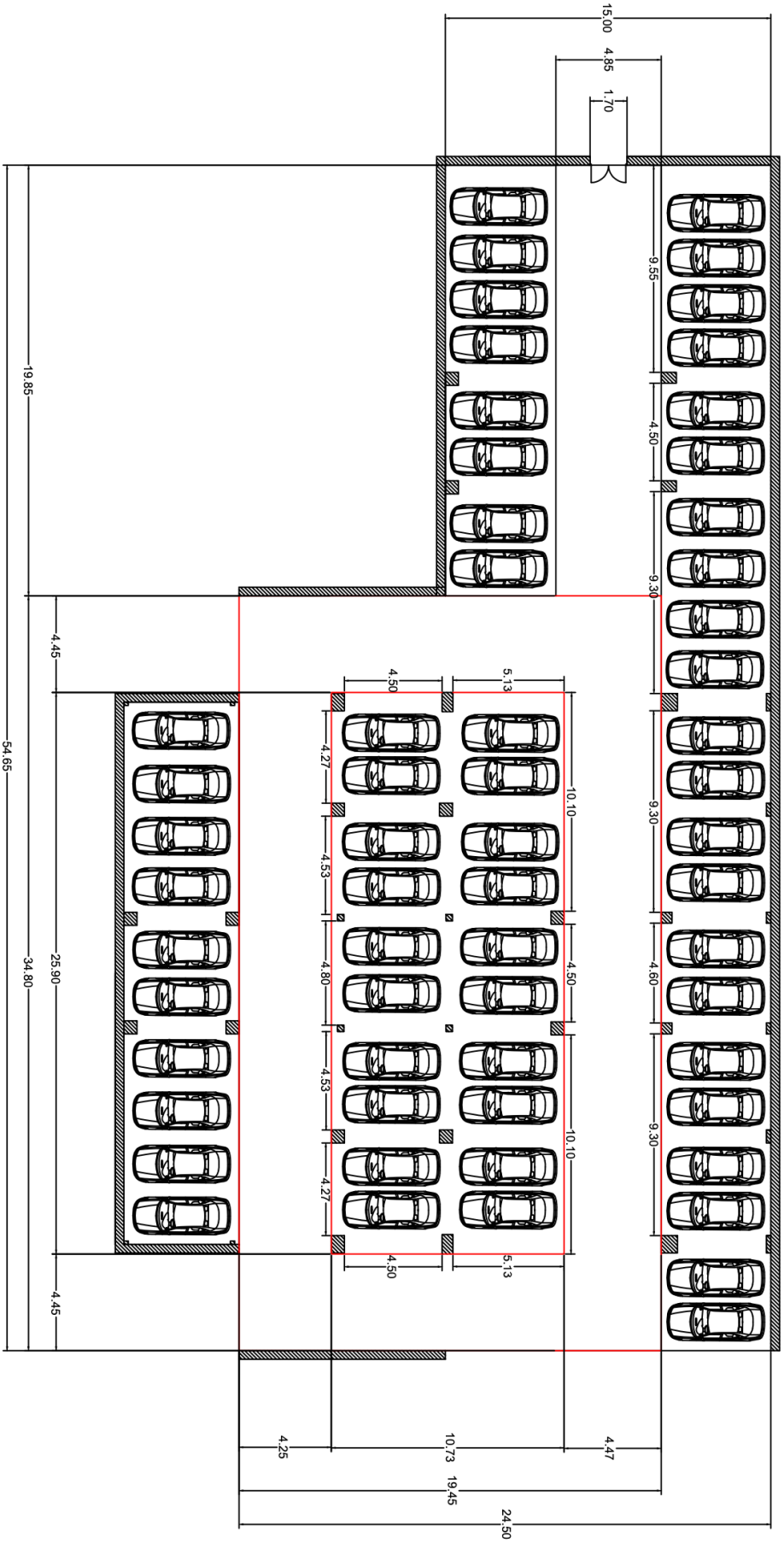

Figura 19. Diseño en AutoCAD del aparcamiento, escenario para el vehículo.

3.4.2 Despliegue de balizas en un escenario ideal

El escenario ideal sería aquel en el cual podríamos maximizar el área cubierta bajo el supuesto de máxima cobertura ofrecida por las balizas. Para calcular dicha situación deberemos tener además varias cosas en cuenta, como la cantidad de balizas disponibles o cuántas son necesarias para obtener la ubicación del vehículo.

En primer lugar, para poder localizar el vehículo de manera correcta, éste debe de estar al alcance de al menos tres balizas fijas simultáneamente, en caso de que queramos conocer su posición y altura, o al alcance de dos balizas si únicamente queremos conocer la posición. Por lo que el sistema es capaz de aplicar algoritmos de bilateración y trilateración. Asimismo, cuando operamos solamente con dos balizas fijas y se aplica el algoritmo de bilateración, el sistema es capaz de despreciar la solución errónea indicándole a mano en qué lado se encuentra el objeto móvil con respecto al eje formado por las balizas fijas. Si no indicásemos en qué lado se encuentra podríamos obtener dos posibles soluciones para el mismo móvil y solo una de ellas sería correcta, por ello el sistema implementa esta solución.

Además, únicamente disponemos de siete balizas Super Beacon, de las cuales hay que tener en cuenta que una de ellas va a funcionar como baliza móvil y otra va a estar reservada para poder calcular el ángulo de *yaw* llegado el momento. Por lo tanto, solamente podemos crear un mapa utilizando cinco balizas fijas. En caso de que queramos obtener todos los datos del vehículo, posición y altura, la mejor disposición para obtener la mayor área posible sería la que encontramos en la Figura 20. Utilizando los valores de alcance ofrecidos por el fabricante, 30 metros de rango, podríamos llegar a cubrir un área total de 1576,66m^2.

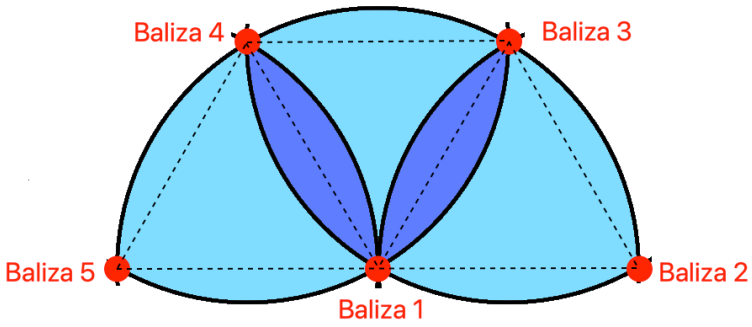

Figura 20. Disposición ideal de las balizas.

Como se puede ver, la solución ideal en un espacio en el que no hubiera obstáculos a la propagación del sonido y que permitiera situar las balizas en cualquier punto correspondería con una celosía hexagonal, donde las balizas se situarían en los vértices de los hexágonos y en el centro de los mismos. Esta es una solución habitual en el despliegue de estaciones radio en general. La distancia entre balizas sería exactamente el radio de cobertura de las mismas, que en nuestro caso serían 30m. Si se deseara tener un margen de confianza, entonces este radio se puede reducir en la medida deseada. En la Figura 20 se muestra en azul pálido las regiones en la configuración mostrada con cinco balizas que tienen cobertura de tres balizas, y en violeta las regiones que reciben cobertura de más de tres balizas.

Sin embargo, dicha situación no se adapta al escenario del que disponemos en el aparcamiento, por lo que habría que situarlas de diferente manera para conseguir la mejor situación posible. Como ya se ha mencionado, de todo el aparcamiento se ha decidido limitar el movimiento del vehículo al rectángulo rojo, como pudimos observar en la Figura 16. Por lo tanto, se debe buscar una configuración diferente, pero garantizando que se cubre toda el área, que además en la zona en la que se va a mover el vehículo siempre cuenta con al menos con la cobertura de tres balizas fijas y que hay un lugar de apoyo en el que las balizas puedan ser fijadas. Teniendo todo esto en cuenta obtenemos que la mejor disposición sería la que se puede ver en la Figura 21.

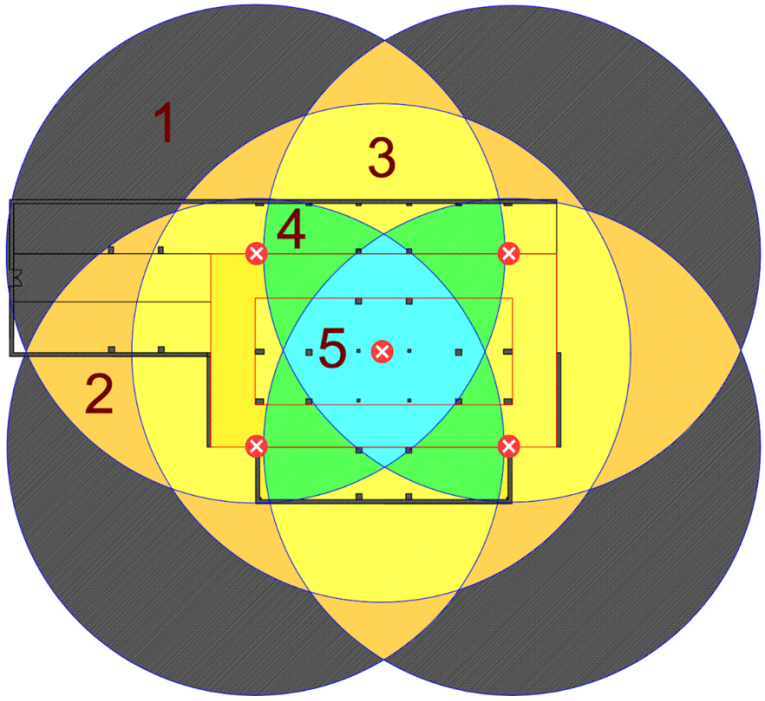

Figura 21. Área máxima que se puede abarcar en una situación ideal dentro del garaje.

En dicha figura se representa la localización de las balizas, marcadas con aspas blancas rodeadas de un círculo rojo. Desde ese punto se traza su zona de cobertura, suponiendo unos 30m de cobertura. A partir de esos datos se construye el mapa de cobertura que vería la baliza móvil, indicando mediante un código de colores cuántas balizas vería desde cada punto. Las zonas en gris solo verían una baliza fija, en naranja dos balizas, en amarillo tres balizas, en verde cuatro balizas y en cian cinco balizas. Dado que es el escenario ideal, se ha despreciado el efecto de columnas y paredes.

3.4.3 Análisis del escenario real

En el escenario real, aparte de tener en cuenta las condiciones establecidas en el apartado anterior (fundamentalmente cobertura en el área designada y puntos de fijación de las balizas), nos vamos a encontrar con obstáculos a la propagación de las ondas sonoras. Dentro de las especificaciones y recomendaciones que se dan en el manual de Marvelmind [15], se recomienda que para tener una buena localización es necesario que haya visión directa entre la baliza móvil y la fija. En el garaje nos encontramos con diversos obstáculos, fundamentalmente columnas si las balizas se fijan en las paredes y tuberías y columnas si se fijan en el techo. Por lo tanto, es conveniente analizar cómo sería si tuviéramos en cuenta todas las sombras

producidas por muros y columnas, suponiendo que las ondas sonoras no van a difractar ni rebotar. De esta manera, a partir del análisis de la situación ideal, obtuvimos todas las sombras producidas por cada una de las cinco balizas con cada elemento del terreno, obteniendo el escenario que se puede observar en la Figura 22.

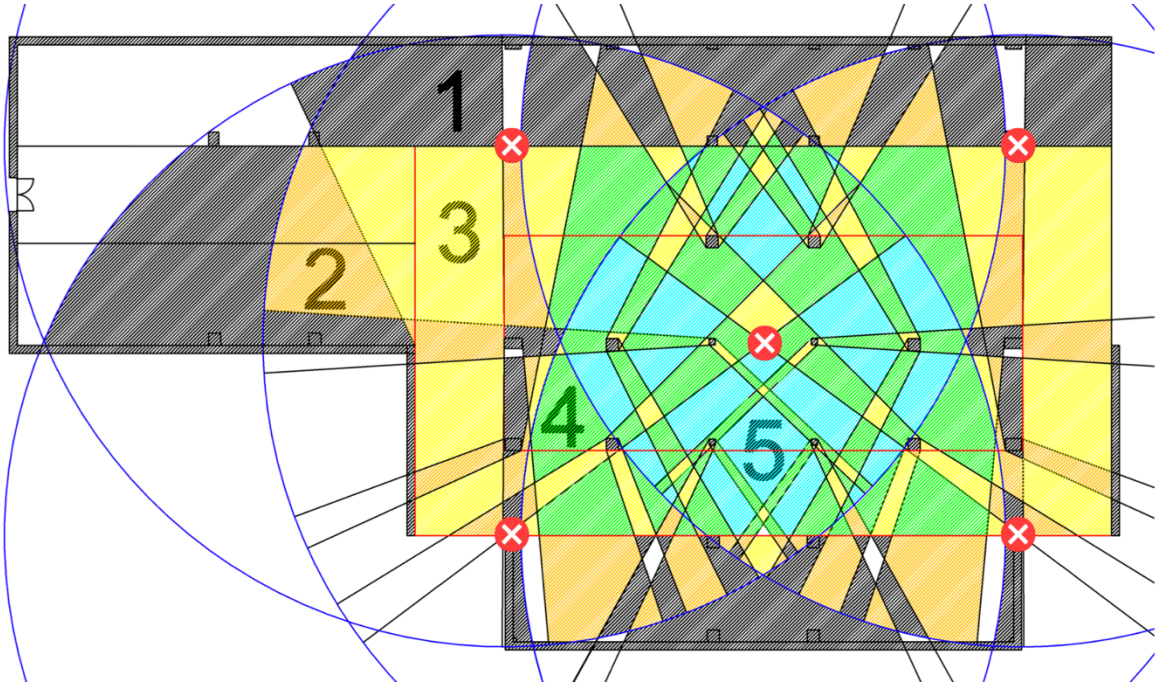

Figura 22. Área real cubierta con las balizas teniendo en cuenta las zonas de sombra.

En dicha figura podemos ver como la situación no es tan favorable y nos encontramos varias zonas donde el número de balizas visibles está por debajo del óptimo marcado en el apartado anterior. Existirán ocasiones, por lo tanto, en las que la localización podría ser errónea o mala, sobre todo en todas aquellas zonas que poseen un color naranja o negro.

Merece la pena señalar que para realizar este mapa se ha supuesto que no hay vehículos estacionados, aunque, excepto que los vehículos estacionados sean muy altos o las balizas estén colocadas muy bajas, no deberían producir sombras adicionales.

3.4.4 Configuración de las balizas

Parte fundamental para hacer que todo funcione correctamente es configurar el sistema de manera correcta. Las balizas de Marvelmind poseen numerosos parámetros para modificar, sin embargo, vamos a detallar aquellos más importantes y que hemos modificado, ya que el resto se han dejado en su valor por defecto de fábrica.

El primero de ellos sería la Frecuencia Base de Radio, la frecuencia a la que operan las balizas. En nuestro caso, seleccionamos 868 MHz ya que es la frecuencia sin licencia que se utiliza para este tipo de dispositivos en Europa. En este caso habría que tener en cuenta cuáles son las frecuencias que son capaces de soportar tanto los modelos Super Beacon, como el modem y ver que sean compatibles. Este tipo de compatibilidad es de especial importancia en el caso de que se quieran comprar más balizas para probar un escenario más amplio.

El segundo sería el Perfil Radio, parámetro que define el rendimiento de transmisión de las balizas. Para nuestro proyecto se ha seleccionado 100Kbps, ya que logra un equilibrio entre el rendimiento y el consumo de energía. Es necesario estudiar la compatibilidad entre los distintos dispositivos, ya que no todos soportan mismos perfiles radio. La segunda versión de la baliza Super Beacon es capaz de soportar todos los perfiles radio, así como el modem, sin embargo, la tercera versión de la baliza solamente soporta 38Kbps y 100Kbps [22]. Es evidente que esta tasa de transmisión es importante para el refresco de la posición, pero es mejor ponerlo en relación con las tramas que deben transmitir las balizas al modem. Conociendo que el tamaño de las tramas recibidas por el modem es de 608 bits, tendremos la posibilidad de transmitir 164 tramas por segundo. Por ejemplo, en el caso de que tengamos cinco balizas operando, para poder conocer el refresco de la posición de las balizas, cada una de ellas será capaz de transmitir 32 tramas por segundo.

La Frecuencia de Ultrasonidos utilizada, frecuencia que se encuentra por encima del rango de audición humana y que las balizas utilizan para la localización del vehículo. Existen numerosas frecuencias a seleccionar compatibles con las balizas, desde los 19kHz hasta los 45kHz, y cada baliza tiene su propia frecuencia establecida de fábrica. Destacar que cada baliza debe ser configurada exclusivamente con aquella frecuencia con la que viene diseñada de fábrica, a pesar de que podamos modificar este parámetro desde el Dashboard. La frecuencia la podemos encontrar indicada en la parte inferior de la baliza. Modificar esta frecuencia, aunque el *software* lo permite, hará que la presición de las medidas que utilicen datos de esa baliza sea sensiblemente peor.

Por último, la Tasa de Refresco, parámetro que establece la frecuencia con la que se transmiten las señales. Dada la naturaleza del proyecto, se busca el seguimiento de un vehículo en tiempo real, por lo que es crítico que la posición se actualice lo más rápido posible. Por ello, se ha seleccionado la mayor tasa de refresco permitida, 16 Hz.

3.4.5 Planos de planta o *Floorplans*

Un plano de planta o *floorplan* representa el plano digital de una superficie, el cual puede abarcar desde una habitación a un edificio entero. Estos planos se pueden cargar en el Dashboard y pueden ser de gran utilidad para la creación de los mapas de localización, ya que nos pueden ayudar a visualizar de manera más clara la ubicación de las balizas a lo largo de la superficie.

A lo largo del proyecto se han desarrollado varios planos a través de AutoCAD, los cuales han sido vitales para trabajar con el Dashboard. A modo de ejemplo, en la Figura 23 podemos encontrar unos de los planos desarrollados para una de las pruebas que se han realizado, una prueba diferente de la ideal descrita en el apartado anterior. En esta podemos ver como se dispusieron las posibles ubicaciones de las balizas en una zona concreta y más pequeña del aparcamiento. De esta forma, posteriormente podemos asegurarnos de que la ubicación de las balizas durante la creación del mapa en el Dashboard sería la correcta. Como podemos observar en la Figura 24, efectivamente la posición de las Super Beacon coinciden en la creación del mapa con los puntos rojos mencionados.

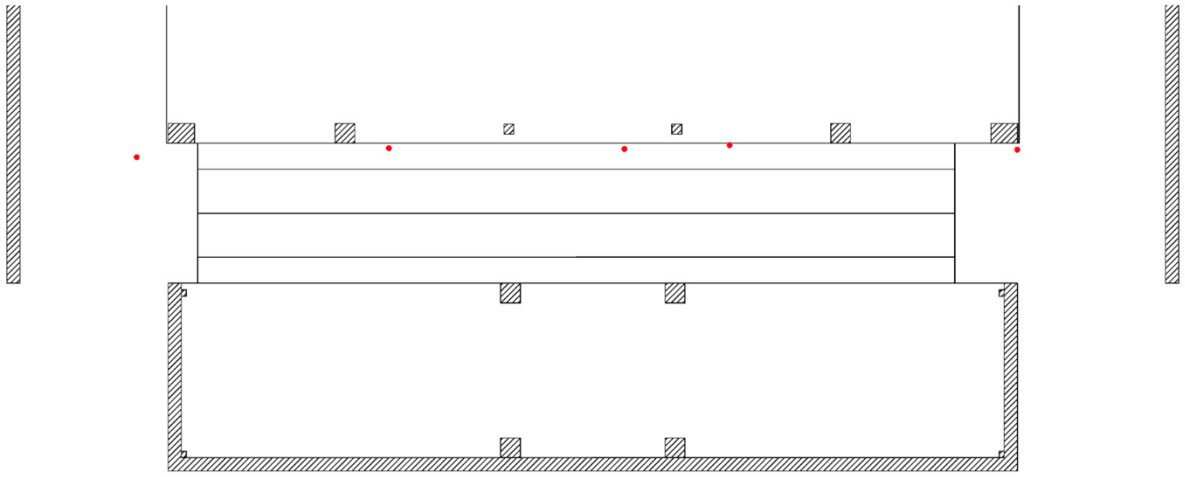

Figura 23. Plano de AutoCAD con las posibles posiciones de las balizas.

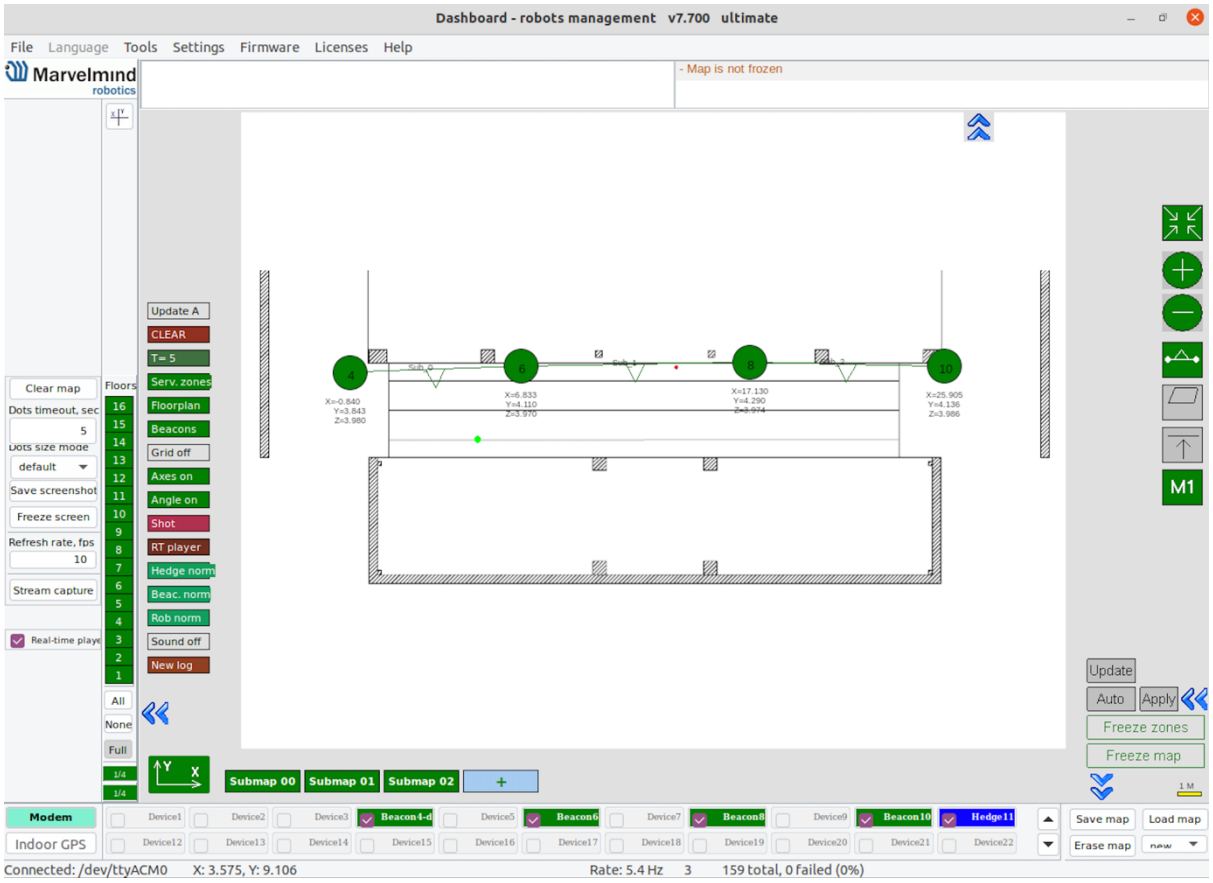

Figura 24. Dashboard con las balizas situadas en el plano.

3.4.6 Submapas

Los submapas son necesarios cuando trabajamos en entornos grandes o complejos, debido a que la precisión empeora a medida que el móvil se aleja de las balizas o cuando nos encontramos en escenarios donde no hay línea de visión directa, ya sea debido a muros u otros obstáculos. La creación de submapas consiste en dividir el escenario en varios mapas más pequeños, donde cada zona estará cubierta por entre 1 y 4 balizas máximo, pudiendo éstas formar parte de varias zonas de manera simultánea. A través de los submapas podremos obtener una mejor precisión en nuestro escenario real, como será detallado más adelante.

Una posible situación de creación de submapas podría ser la ejemplificada en la Figura 25. En ésta nos encontramos un escenario con dos submapas, cada uno de ellos formado por dos balizas fijas, donde el primer submapa está formado por las balizas 10 y 11, y el segundo submapa por las balizas 11 y 12. Cuando la baliza móvil se encuentre en el primer submapa, la localización del móvil será calculada a través de las balizas 10 y 11, y de forma análoga funcionará el segundo. Además, para cada uno de ellos se ha definido su zona de servicio, es decir, la zona de las que cada submapa debe hacerse responsable.

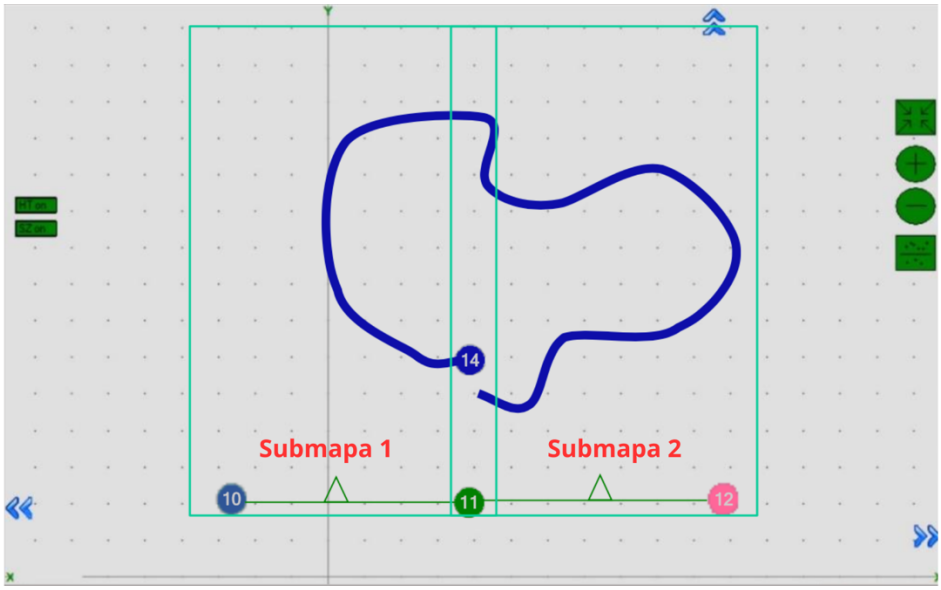

Figura 25. Ejemplo de submapas.

Este ejemplo tiene sentido cuando las balizas fijas que más distan entre sí no estén dentro su rango máximo, es decir, disten más de 30 metros. Se logra así, que a pesar de que las balizas no puedan comunicarse entre sí directamente, sí podrán llevar a cabo la localización del móvil de manera exitosa.

Entre ambos submapas nos encontramos una zona de superposición en la que cabe preguntarse qué es lo que sucede, esta zona se denomina *handover* o área de transición, podemos observarla coloreada en azul en la Figura 26. Este área es necesaria para poder moverse de un submapa a otro, donde la baliza móvil será localizada a través de ambos submapas. Dicha zona debe tener un tamaño concreto, éste dependerá de la velocidad a la que se mueva la baliza móvil y de la tasa de refresco del sistema. Se recomienda que el tamaño de *handover* sea suficiente como para poder obtener 4 o 5 actualizaciones de la posición del objeto en su interior.

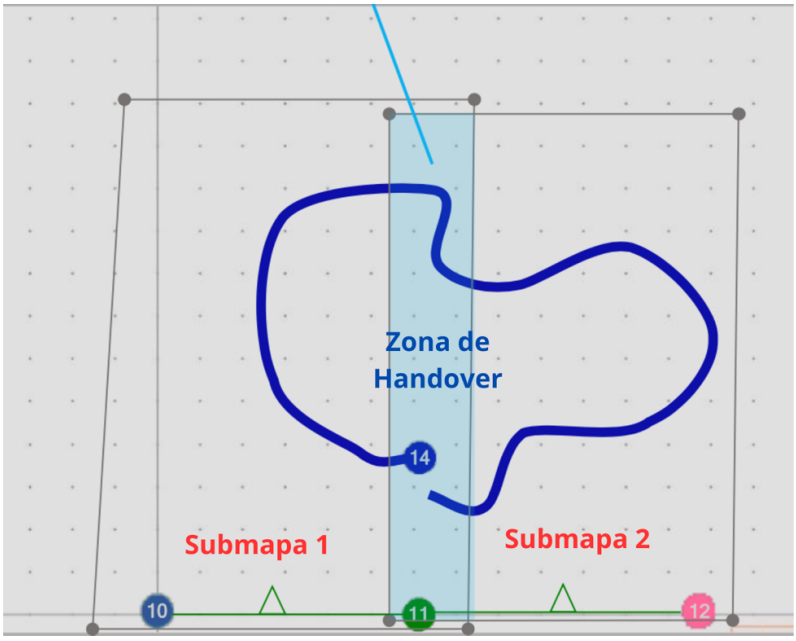

Figura 26. Ejemplo zona de handover.

Además, destacar que se ha detectado un mejor funcionamiento del sistema cuando únicamente utilizamos submapas formados por dos balizas, donde además hay que prestar especial importancia a la frecuencia de funcionamiento de las balizas en el caso de que se utilice arquitectura IA. Es importante no poner en un mismo submapa dos balizas con la misma frecuencia de funcionamiento en arquitectura IA, porque obtendríamos resultados considerablemente erróneos.

Asimismo, cuando creamos submapas formados por dos balizas con una base muy amplia, es decir, separadas al menos cinco metros, cuanto más se acerca la baliza móvil a la línea formada por las balizas fijas, mayor deriva encontraremos en el eje formado por dichas balizas. Podríamos solventar este problema acercando entre sí las balizas fijas, logrando así mayor precisión en el eje formado por las mismas.

3.5 Interpretación de los ficheros de registro (*logs*)

Como ya hemos visto el propio Dashboard de Marvelmind nos muestra a través de su interfaz gráfica la localización y movimiento realizado por las balizas móviles. Sin embargo, puede ser útil no únicamente visualizar su movimiento, sino obtener los datos reales de localización.

En la arquitectura seleccionada para el proyecto, NIA, son las balizas móviles las encargadas de emitir las señales de ultrasonidos y las balizas fijas las encargadas de transmitir los datos al ordenador. En esta situación las balizas móviles transmitirán a las fijas diferentes datos como su ID, la potencia de la señal y la hora de transmisión, para así las balizas fijas almacenarlos en su memoria interna junto con la potencia de señal recibida y la hora de recepción. Entonces es el ordenador, que está conectado a las balizas fijas a través del modem, el que descarga los datos de su memoria interna y los utiliza para calcular la posición del vehículo.

Para cada instante de toma de datos, determinado por la tasa de refresco, el ordenador obtiene la localización, la cual almacena en un fichero con formato CSV. Dentro del directorio de la

aplicación Dashboard nos encontramos una carpeta denominada 'logs', en la cual cada vez que congelamos el mapa a través de la aplicación, se creará un fichero con los datos mencionados.

En el interior de dichos ficheros nos encontraremos con diferente información, la cual dependerá del modelo de balizas y de modem que estemos utilizando. El nombre del fichero nos indicará la fecha y hora de creación de la forma '`2023_12_11__123722__Marvelmind_log.csv`'. Para los dispositivos de los que disponemos, podremos encontrarnos dos modelos diferentes de tramas:

```
T2023_12_11__123737_436,user,41,129, 7, -3.631, 2.499, 0.5, 2, na, 40
T2023_12_11__123737_541,user,41,18, 10, 1.5, -0.07, 0, 0
```

En éstas, los primeros elementos son comunes y quieren decir:

1. `T2023_12_11__123737_436`: hace referencia a la fecha en la que se produce, 11/12/2023, y la hora, 12h : 37min : 37.436secs.
2. `user`: por defecto vendrá *user* en todas las tramas. Se estima que en versiones futuras los usuarios puedan *loggearse* en la aplicación de Marvelmind y así poder diferenciar entre usuarios.
3. `41`: ID del tipo de trama.

Sin embargo, el siguiente elemento ya difiere, `129` o `18`, que indican si la información de la trama pertenece a una baliza móvil, primer caso, o una baliza fija, segundo caso. Por ello, la explicación la información de las balizas móviles es la siguiente [23]:

1 `129`: especifica que la trama pertenece a una baliza *hedgehog* o móvil, y que además la transmisión de la posición se está haciendo en *streaming*, es decir, en tiempo real.
2 `7`: especifica el identificador de la baliza que actúa como móvil.
3 `-3.631, 2.499, 0.5`: coordenadas X, Y, Z dadas en metros, destacar que no será la localización exacta del centro de la baliza, sino del centro del micrófono.
4 `2`: flags.
   - Bit 0: 1 indica que no se ha logrado obtener las coordenadas X, Y, Z correctamente.
   - Bit 1…6: reservados para versiones futura.
   - Bit 7: 1 indica que se encuentra fuera de la zona de servicio.
5 `na`: ángulo *yaw* dado en decigrados (0…3600), es decir, ángulo respecto a la trayectoria. Aparecerá como 'na' cuando no sea calculado.
6 `40`: Tiempo transcurrido desde la emisión del ultrasonido hasta el cálculo de la ubicación en dicha trama, dado en ms.

Por el contrario, si la información pertenece a una baliza fija:

1 `18`: especifica que la trama es de una baliza con posición estacionaria, es decir, una baliza fija.
2 `10`: especifica la dirección de la baliza que actúa como fija.
3 `1.5, -0.07, 0`: coordenadas X, Y, Z dadas en metros.
4 `0`: campo reservado para aplicaciones futuras.

3.6 Pruebas realizadas con las balizas de ultrasonidos

En este apartado se muestran las pruebas que se han realizado, así como un análisis de los resultados obtenidos de éstas. Dado que los resultados de las pruebas en alguna ocasión podían no ser los esperados, se decidió implementar un algoritmo de discriminación de resultados erróneos. Asimismo, para poder llevar a cabo dichas pruebas fue necesario diseñar un carro de pruebas y emular así el Renault Twizy.

3.6.1 Carro de pruebas

En el momento de la realización de este Trabajo Fin de Máster el vehículo Renault Twizy no estaba en situación operativa, debido a que diversos trabajos realizados por otros estudiantes habían ido incluyendo modificaciones pero, por diversas razones, sin poner el vehículo otra vez en marcha. De hecho, en paralelo a este trabajo, otro estudiante estaba trabajando para definitivamente dejar operativo el vehículo. Mi Trabajo Fin de Máster se finalizó sin embargo antes de que se llegara a la situación deseada y por lo tanto no se podía realizar prueba alguna en el vehículo. Por ello se optó por montar un carro con todos los elementos necesarios para la realización de las pruebas, que incluía el ordenador llamado 'PC fusión', dado que el futuro se espera realizar la fusión de datos en el mismo, y la baliza móvil. En el ordenador de fusión también se encontraba instalado el *software* de Marvelmind.

Pretendiendo ser lo más fieles a las condiciones del coche, se tuvo en cuanto la altura de éste para disponer la baliza móvil. Tal y como podemos observar en la Figura 27 se utilizó una varilla metálica, para la cual se creó una base donde poder ubicar la baliza, logrando así adaptarnos a la situación del Twizy. Accediendo a la ficha técnica podemos averiguar que la altura del vehículo es de 1.47 metros, altura a la que se ubicó la varilla.

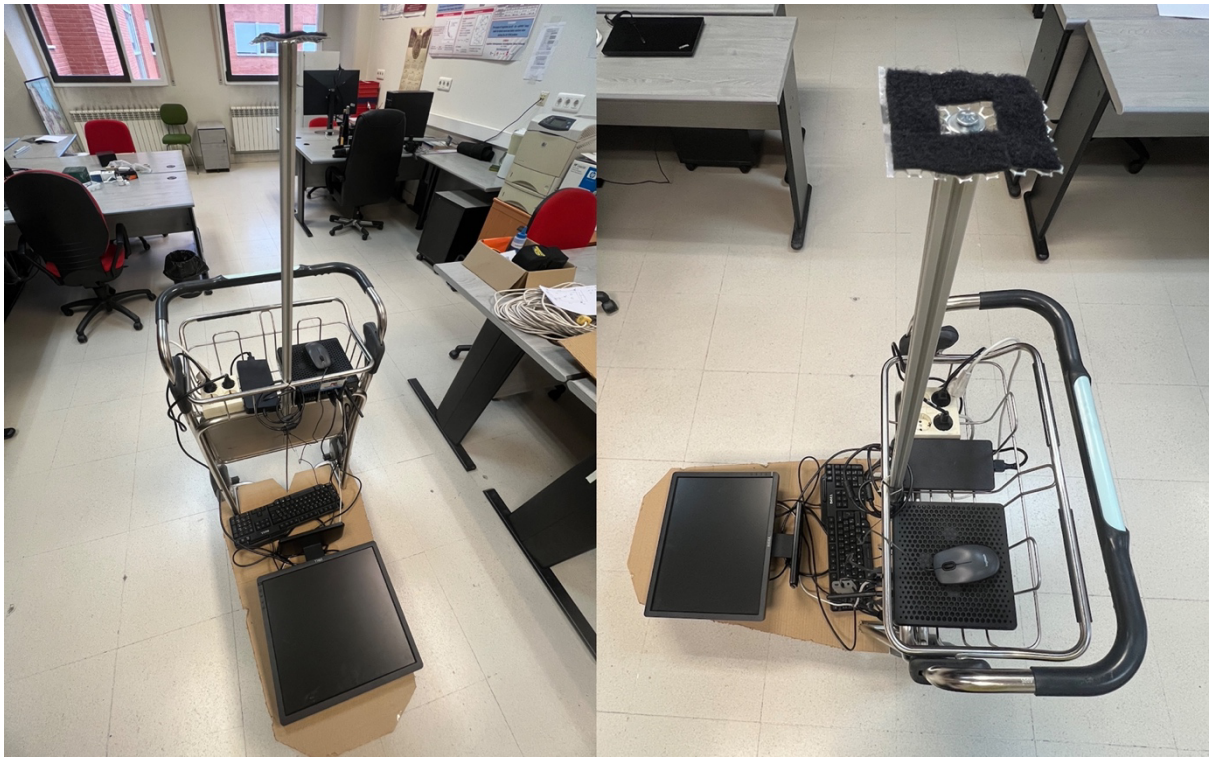

Figura 27. Montaje del carro de pruebas.

Asimismo, se dibujó una cuadrícula en el suelo del aparcamiento de la facultad con el objetivo de poder medir y comprobar el rendimiento del sistema con exactitud. Fue diseñada de tal forma que permitiera recoger la información en tres carriles diferentes, uno central que es el que supuestamente seguiría el vehículo, y dos laterales que nos mostrarían hasta qué punto el vehículo podría obtener una localización precisa cuando se acerque a zonas que entrañen riesgo de colisión. Además, para poder certificar la precisión del sistema de localización se dividió la longitud total de los tres carriles de 23 m en secciones de 1 m, creando así la cuadrícula que se puede observar en la Figura 28.

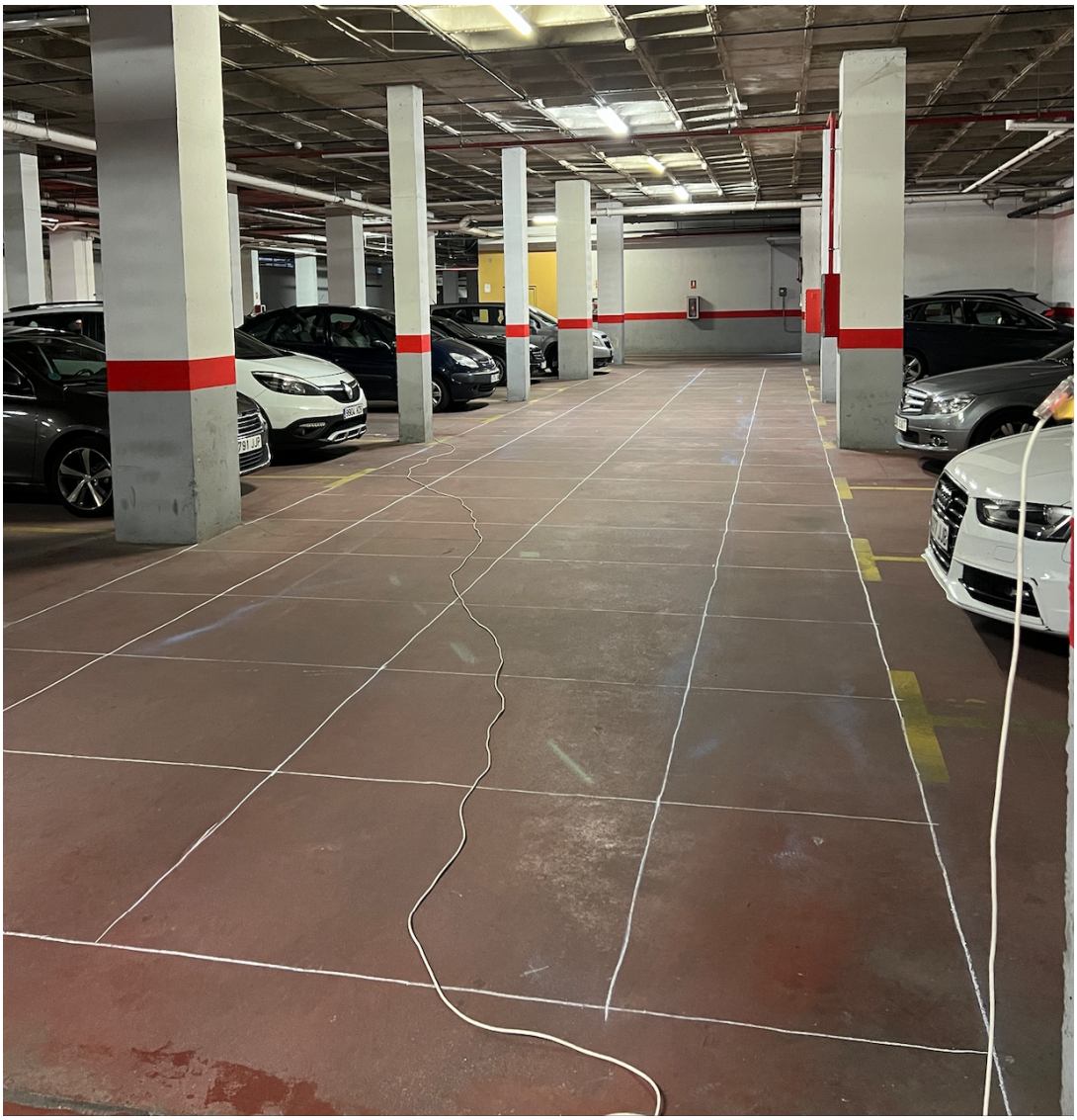

Figura 28. Cuadrícula desplegada en el aparcamiento para las pruebas.

Las medidas llevadas a cabo en toda la cuadrícula implicarán ubicar el carro en cada punto de la misma. En otras palabras, se alineará la baliza con cada punto donde las líneas verticales se cruzan con las horizontales, por cada uno de los tres carriles. Esto permitirá obtener 72 medidas distintas para cada uno de los escenarios que plantearemos más adelante. En la imagen de la Figura 29 podemos ver un ejemplo de cómo se procedió con la toma de medidas, donde se observa que con un alargador de la varilla metálica que sostiene la baliza móvil es posible situar la baliza con gran precisión sobre los puntos de la cuadrícula.

Figura 29. Toma de medidas en la cuadrícula de pruebas con el carro.

3.6.2 Discriminación de *outlayers*

A pesar de que explicaremos más en detalle posteriormente los resultados obtenidos, es necesario comentar que el rendimiento obtenido no concuerda con el mencionado por el fabricante. Por ello, para las diferentes pruebas que se han realizado se ha tenido que establecer un radio para poder eliminar *outlayers*, es decir, despreciar medidas de localización erróneas, o incluso que pudieran engañarnos por aproximarse demasiado a un punto de medida contiguo.

Para calcular qué puede ser considerado *outlayer* durante el funcionamiento normal del vehículo, hemos considerado dos parámetros, la velocidad del vehículo y la tasa de refresco de la posición del vehículo.

Como ya se mencionó en el apartado referente a la configuración de las balizas la tasa de refresco establecida es de 16 Hz, sin embargo, este valor varía con el obtenido en la realidad. Para las pruebas realizadas se procederá al análisis de los ficheros '`log.csv`', los cuales se crean con la información obtenida de las balizas por parte del modem. Si bien la tasa de refresco efectiva fuera de 16 Hz se esperaría obtener una actualización de la posición del vehículo cada 62 ms. Sin embargo, si nos fijamos en los *timestamps* que reflejan la localización de la baliza, realmente obtenemos actualizaciones cada 185 ms. Por lo que podemos afirmar que el uso efectivo del canal es menor al ancho de banda disponible, esto puede deberse al sistema de comunicación que utilizan las balizas.

```
T2023_12_11__121704_482,user,41,129,11,1.015,2.100,1.470,2,na,115,0,na,
na,na
T2023_12_11__121704_667,user,41,129,11,1.017,2.113,1.470,2,na,115,0,na,
na,na
```

Luego, conociendo la tasa de refresco real y suponiendo una velocidad máxima del Twizy de 10 km/h durante su operación en modo autónomo, podemos conocer cuánta distancia máxima podrá recorrer el coche durante el periodo de tiempo mencionado. De esta manera es posible hacer un cálculo de un radio respecto de la última muestra válida tomada fuera del cual, cualquier localización encontrada puede ser considerada un *outlayer* y eliminada. El cálculo es de esta manera muy sencillo:

$$\frac{10km}{1h} \times \frac{1.000m}{1km} \times \frac{1h}{3.600.000ms} = \frac{0{,}0027m}{ms}$$

$$\frac{0{,}0027m}{ms} \times 185ms = 0{,}51m \approx 60\ cm$$

Luego para cada medida realizada en cada uno de los puntos de la cuadrilla se le aplicará una discriminación de todas aquellas medidas que se excedan más de 60 cm, puesto que consideramos que durante la operación normal del vehículo dichas localizaciones pueden ser eliminadas no contribuyendo al error de posicionamiento.

3.6.3 Primera prueba

Como ya se mencionó en el apartado referente al análisis del escenario, la primera intención es abarcar el mayor área posible que permita el sistema. Por ello, en conocimiento del rango máximo de las balizas, se dispusieron las balizas de la forma que podemos ver en la Figura 30, a sabiendas de que habría zonas en las que sería muy complicado la localización del vehículo por culpa de las zonas de sombra. De esta forma se pretende abarcar la superficie delimitada en rojo, es decir, alrededor de 676 m^2. Dada la envergadura de la superficie es necesaria la utilización de submapas, por ello para esta prueba se utilizaron ocho submapas, cuatro de ellos formando el rectángulo exterior y otros cuatro que unieron cada baliza exterior con la central.

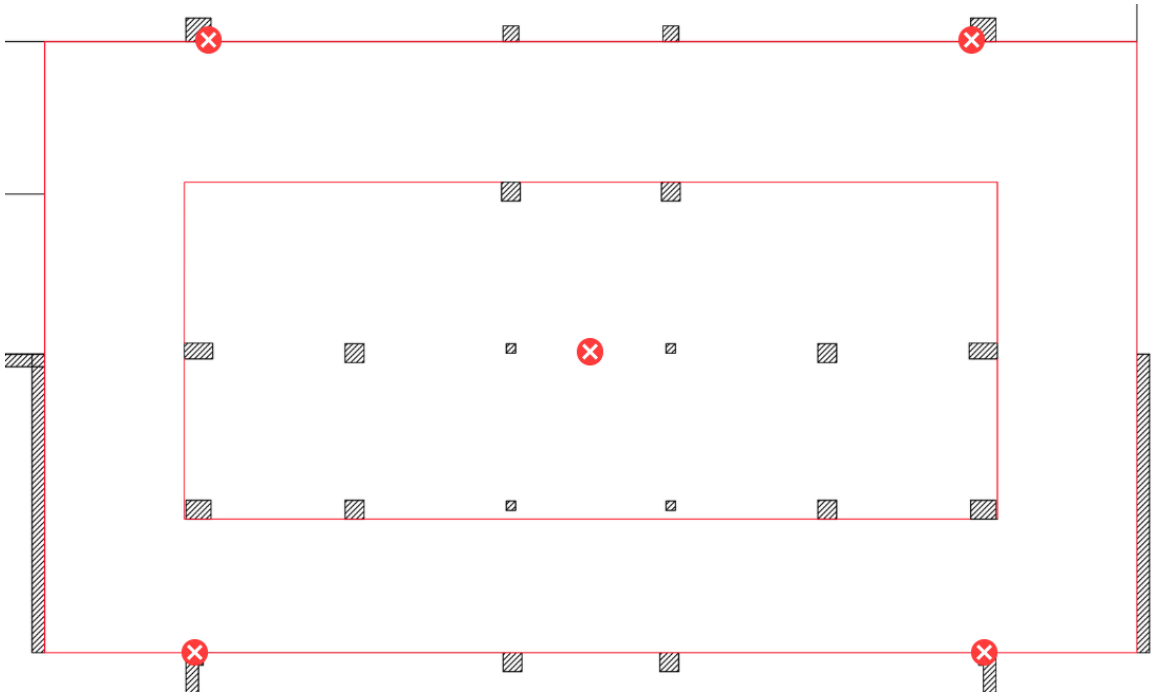

Figura 30. Delimitación de la zona donde se quieren realizar las pruebas en el garaje de la ETSIT.

Para realizar esta prueba se dispusieron las balizas a una altura de 2 metros y se montó un trípode en el centro del aparcamiento para poder ubicar la baliza a dicha altura. En las dos imágenes de la Figura 31 podemos observar la baliza central ubicada a la altura indicada, junto con dos de las balizas exteriores que se encuentran en las columnas.

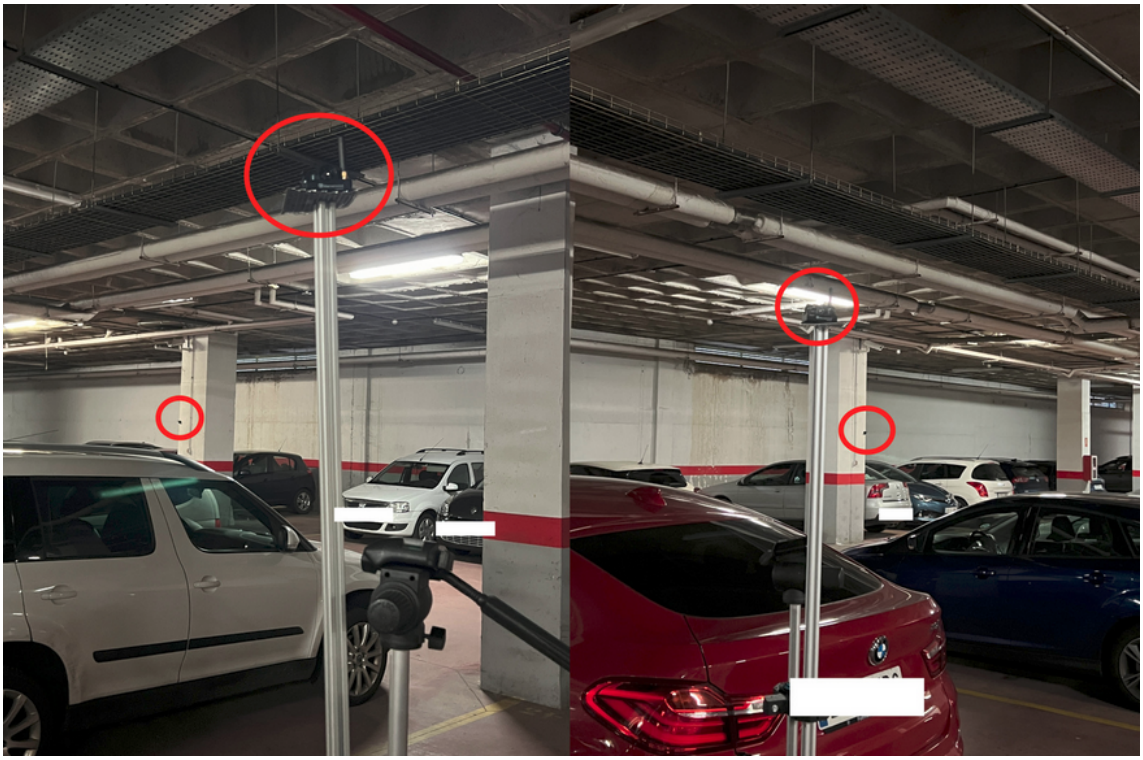

Figura 31. Ejemplo de despliegue de las balizas sobre el escenario.

Durante la propia realización de la prueba ya se detectó que había ciertas zonas donde el sistema no era capaz de localizar el vehículo. Sin embargo, con el objetivo de verificar cómo se comportaba realmente, se desarrolló un código en Python para crear un gráfico ejemplificativo de la cuadrícula creada para pruebas. Para ello, se diseñó la rejilla y sobre ésta se situaron todos los puntos de localización obtenidos a través del modem. Se recorrieron todos y cada uno de los distintos ficheros log.csv creados para cada una de las 72 medidas realizadas, y sobre éstos se aplicó el radio de eliminación de *outlayers*, obteniendo así la situación de la Figura 32. Los códigos python referentes a la creación de estas figuras se encuentran en el Anexo I, así como una explicación de cómo proceder con ellos.

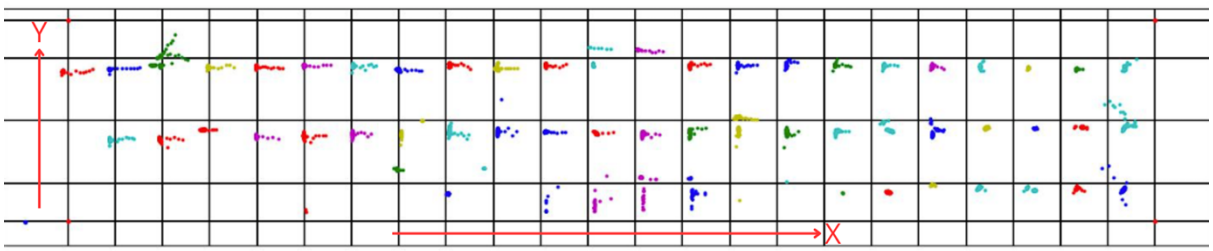

Figura 32. Resultados de localización de la prueba inicial.

En la Figura 32 se representan los 23 metros de cuadrícula y los puntos de localización con diferentes colores para cada medida. En primer lugar, observamos una deriva en la medida

hacia el frente, mientras que durante la toma de medidas el carro de pruebas se mantuvo estático en cada punto. Esto se debe a que a pesar de cortar la toma de medidas desde el propio *software* de Marvelmind, el sistema continúa transmitiendo algunas tramas de localización, por lo que detectaba el movimiento realizado para hacer la siguiente medida. Observando este fenómeno se optó desechar las ultimas tramas captadas, logrando así deshacernos de la deriva y mejorar los resultados obtenidos, podemos observar el resultado en la Figura 33.

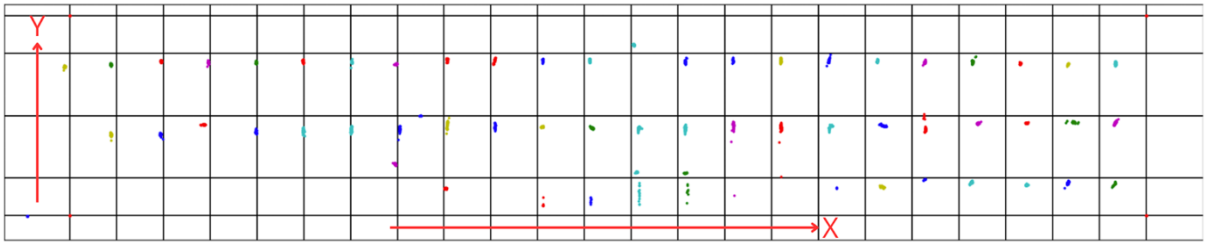

Figura 33. Resultados de localización de la primera prueba tras eliminar deriva.

En segundo lugar, vemos como hay zonas donde ha resultado imposible la localización del vehículo, de las cuales incluso algunas no coinciden con una zona crítica donde únicamente fuese visto por 1 o 2 balizas simultáneamente. Por supuesto, se puede observar que el error obtenido es considerablemente mayor a los ± 2cm que asegura el fabricante.

3.6.4 Segunda prueba

Con intención de deshacernos de las zonas de sombra y así mitigar el efecto observado en la primera prueba, decidimos cambiar la disposición de las balizas, a pesar de que así fuéramos capaces de abarcar un área menor. Por ello, se movieron las balizas y se ubicaron a una altura de 2 metros sobre columnas con la disposición que vemos en la Figura 34. En esta ocasión decidimos centrarnos únicamente en la región delimitada por la cuadrícula. Para esta prueba se utilizaron cuatro submapas con la finalidad de cerrar el rectángulo formado por las balizas, en cuyo interior se iba a mover el vehículo.

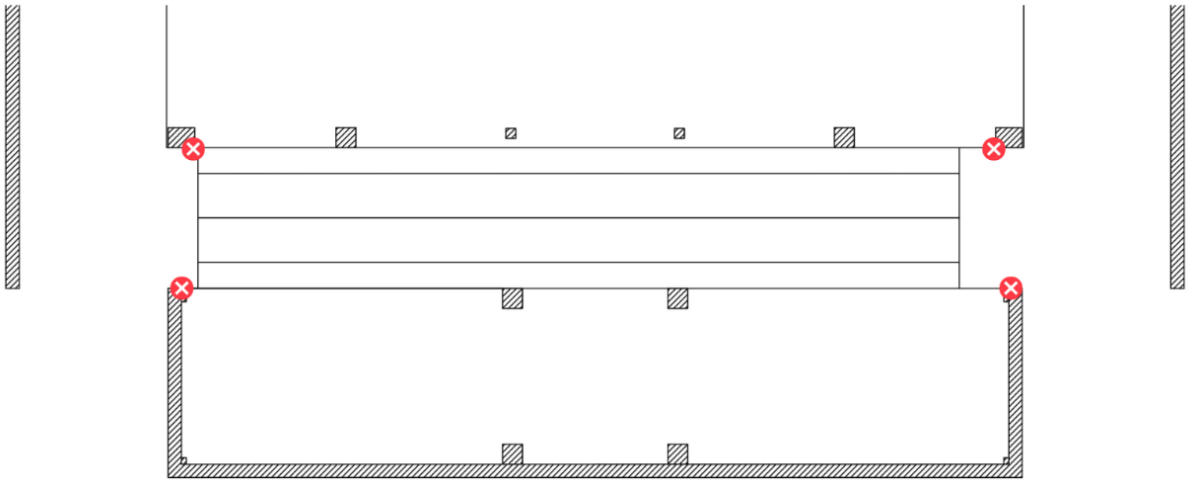

Figura 34. Disposición de balizas en la segunda prueba.

Procesando de igual forma los datos de las medidas obtenidas, los datos de localización plasmados sobre la cuadrícula serían los mostrados en la Figura 35:

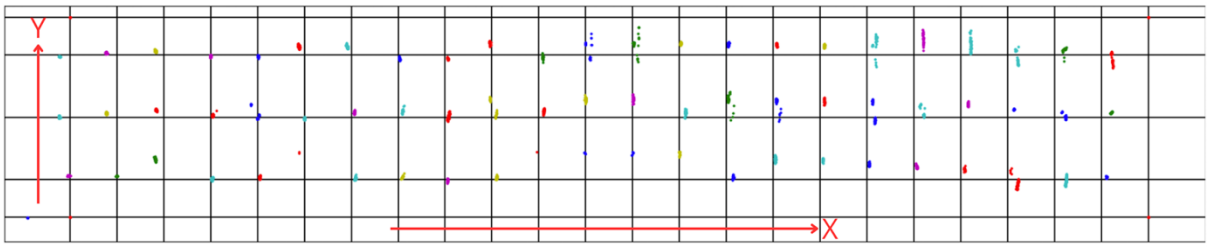

Figura 35. Resultados de localización de la segunda prueba.

En estos observamos una notoria mejoría ya que somos capaces de localizar el coche en todas las posiciones, eliminando así el efecto provocado por las sombras de las columnas en la primera prueba. Sin embargo, podemos ver como en los carriles exteriores en sus zonas centrales de la cuadrícula la localización es bastante inexacta, esto se puede explicar atendiendo a los diagramas de emisión y recepción descritos en la sección 3.2.1. El diagrama de emisión que sería el de la baliza móvil cubre 360º, por lo tanto, en principio no debería dar problemas. Sin embargo, el diagrama de recepción en los planos XZ e YZ muestra que tiene mala recepción para ángulos inferiores a 15º. Dada la disposición de las balizas fijas, tal y como se puede ver en la Figura 36 (antena y micrófono paralelos al suelo), este hecho es importante.

Para mayor facilidad en la interpretación de los datos, vamos a asignar ejes a la cuadrícula, tal y como se ve en la misma gráfica, y vamos a representar conjuntamente el diagrama de recepción y la cuadrícula sobrepuestos, tal y como se puede ver en la Figura 37, de tal manera que es fácil observar que todo el área que queda por debajo de la flecha roja punteada tendría mala recepción. En consecuencia es de esperar que la localización en todo ese área sea incorrecta. De esta forma es fácil calcular una tabla en la que se muestre cómo de cerca puede estar el móvil al eje Y de la cuadrícula antes de que la baliza no pueda determinar la distancia al móvil. Estos datos se pueden encontrar en la Tabla 2, donde se ve que incluso a una distancia de 7 metros de la baliza, ésta ya no sería capaz de recibir la señal del móvil en el carril lateral más cercano a la baliza (que está a una distancia de solo 1 m).

Figura 36. Orientación de las balizas sobre las paredes del aparcamiento.

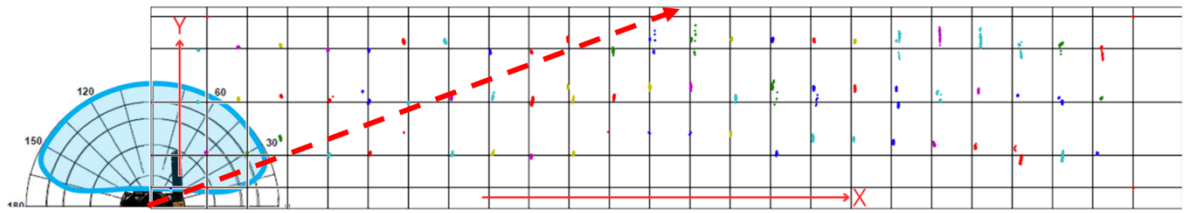

Figura 37. Diagrama de radiación de las balizas Super Beacon sobrepuesto al plano de rejilla. Diagrama no desplazado

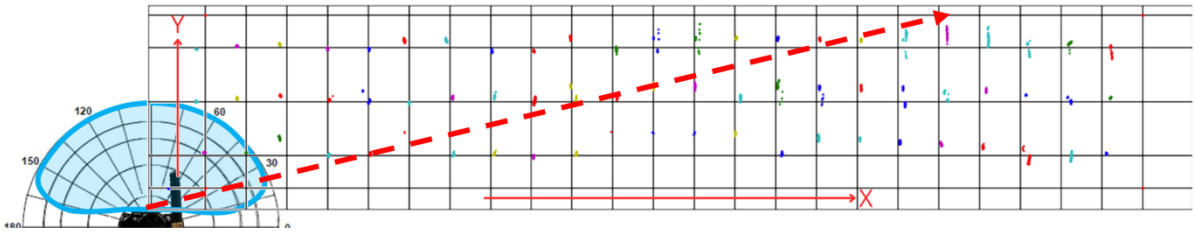

Figura 38. Diagrama de radiación de las balizas Super Beacon sobre puesto al plano de rejilla. Diagrama desplazado.

Distancia X [m]	*1*	*3*	*5*	*7*	*9*	*11*	*13*	*15*	*17*	*19*	*21*	*23*
Distancia Y [m] (15º)	*0.3*	*0.8*	*1.3*	*1.9*	*2.4*	*3.0*	*3.5*	*4.0*	*4.6*	*5.1*	*5.6*	*6.2*
Distancia Y [m] (5º)	*0.1*	*0.3*	*0.4*	*0.6*	*0.8*	*1.0*	*1.1*	*1.3*	*1.5*	*1.7*	*1.8*	*2.0*

Tabla 2. Distancia mínima de la baliza móvil con respecto a las balizas fijas en su plano XY bajo dos supuestos del ángulo de recepción del micrófono.

El diagrama de recepción que ofrece la empresa puede reinterpretarse, dado que parece que en realidad el micrófono es capaz de recibir a ángulos más pequeños, suponiendo que el diagrama de recepción está despalazada sobre el eje. De esta forma se puede reconstruir la figura que sobrepone el diagrama de recepción y la rejilla tal y como se puede ver en la Figura 38, y ver que en este caso el área en el cuál no se puede determinar bien la localización es menor y parece coincidir mejor con el área encontrada experimentalmente. En ese caso, podríamos suponer que el micrófono es capaz de recibir para ángulos superiores a 5º. En ese caso, dado que los carriles laterales están a un metro en el eje Y respecto de las balizas, podrían dar resultados fiables hasta los 10 m de distancia en el eje X, siendo el tercio central de la cuadrícula donde peores resultados se deberían obtener, lo cual coincide con los datos observados.

3.6.5 Tercera prueba

Con el objetivo de eliminar la limitación observada en la segunda prueba debida al alcance de los diagramas de radiación, en esta ocasión se dispusieron las balizas en el techo del aparcamiento a una altura de 4 metros según vemos en la Figura 39. Para esta prueba se formaron tres submapas, uniendo cada baliza con la que se encuentra más próxima a ella.

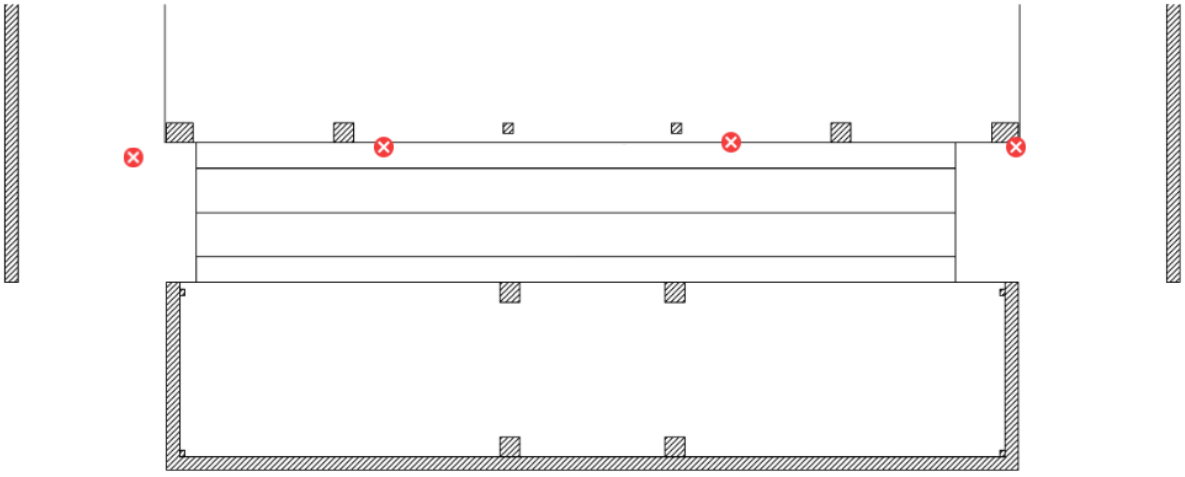

Figura 39. Disposición balizas en la tercera prueba.

Figura 40. Andamio desplegado para poder alcanzar a situar las balizas en el techo.

Destacar que para poder desplegar las balizas en el techo, dada la elevada altura fue necesario montar un andamio disponible en la Universidad, tal y como podemos ver en la Figura 40.

Previamente a realizar la toma de medidas se realizó un análisis de la situación para ver si de esta forma se conseguiría mejorar la situación. Para ello, se recreó el escenario a través de Matlab, representando la sección de corte de cada uno de los tres carriles de la cuadrilla para los diagramas de radiación de las balizas. El objetivo de esta recreación es asegurarse si durante toda la pista de pruebas la baliza situada sobre el vehículo sería vista por tres balizas simultáneamente.

En la Figura 41 nos encontramos las secciones de corte comenzando por el carril más alejado de las balizas, el carril central y el carril más próximo a ellas respectivamente. Podemos ver

representado con la línea negra la altura del Twizy y con los diferentes colores referenciados la las secciones de corte que muestran la cobertura de cada receptor en cada uno de los carriles. Limitándonos al tamaño de la pista de pruebas de 23 metros podemos observar que la situación es prácticamente idéntica para los tres carriles. En sendos carriles nos encontramos que en su comienzo y final hay un tramo donde únicamente son vistos por 2 balizas, abarcando aproximadamente desde los 5.4 a los 7 metros y desde los 25.8 a los 28.4 metros. Por lo que, podemos afirmar que si probásemos dicha situación podríamos encontrarnos con situaciones donde hubiera localizaciones erróneas. El código utilizado para generar la Figura 41 se encuentra disponible en el Anexo II.

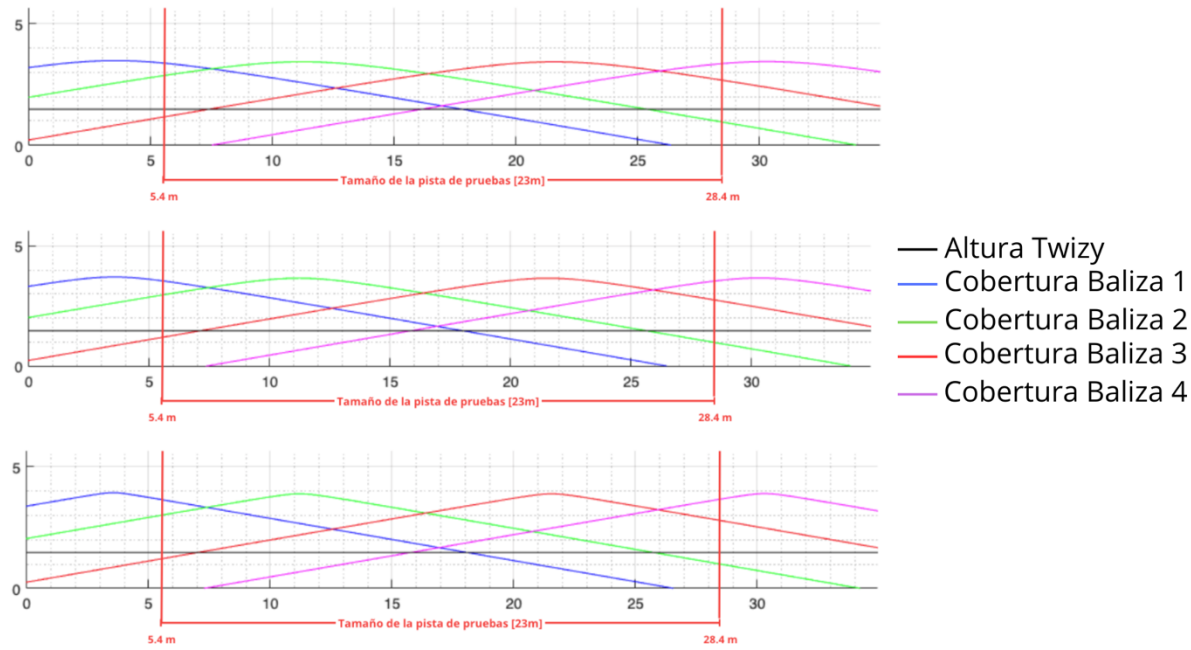

Figura 41. Perfil de línea de visión entre el coche y las balizas desplegadas referentes al carril derecho, central e izquierdo respectivamente.

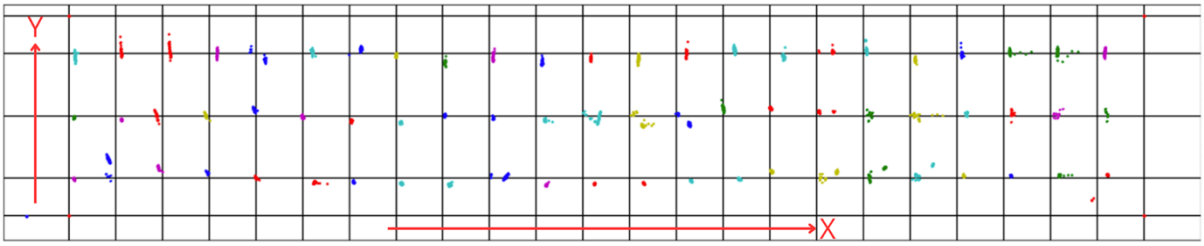

Figura 42. Resultados de localización obtenidos de la tercera prueba.

Para cerciorarnos del verdadero comportamiento en esta nueva tesitura se llevó a cabo la prueba. De igual forma a las pruebas anteriores, se tomaron medidas de los 72 puntos y se representaron a través del código en Python, obteniéndose el resultado mostrado en la Figura 42. Podemos ver una clara mejoría con respecto a la prueba anterior, la zona central donde antes había un claro patrón de divergencia sobre el eje Y mejora. Sin embargo, seguimos observando que los resultados no son los deseables y no son los prometidos por el fabricante. En algunas de las posiciones podemos observar una gran exactitud en la localización, pero en la mayoría hay un error que es superior a los ± 2cm.

3.6.6 Estadísticas

En vista de que los mejores resultados obtenidos son los de la tercera prueba, pasaremos a analizar diferentes estadísticas de los datos de localización obtenidos en dicha prueba. Todas las estadísticas que hemos desarrollado se pueden obtener a través de los códigos presentes en el Anexo III.

En primer lugar, despreciaremos la coordenada Z de los datos (altura de la baliza móvil) dado que no cambiará nunca, ya que nos encontramos en una superficie llana, por ello, prestaremos atención a las coordenadas X e Y. Comenzaremos analizando la desviación estándar y la varianza, las cuales están íntimamente relacionadas. La desviación estándar nos indica la dispersión de los puntos en relación con su media, es decir, consiste en medir la variación de los puntos con respecto a la media [24]. La desviación estándar al cuadrado nos da la varianza, la cual nos proporciona una medida cuadrática de la dispersión de los datos alrededor de la media [25]. La manera más fácil de interpretar la desviación estándar es que si la distribución de la variable aleatoria es normal, una vez la desviación estándar concentraría el 68% de las muestras, dos veces la desviación estándar concentraría el 95% de las muestras y tres veces la desviación estándar el 99.8%.

Atendiendo a la distribución de carriles la Figura 43, para cada uno de ellos tendremos en cuenta la multitud de medidas que se han tomado, de las cuales extraeremos los valores medios de la desviación y de la varianza. De manera individual para las medidas en el eje X y en el eje Y, los resultados son los mostrados en la Tabla 3 y la Tabla 4. En éstos observamos que en el eje Y nos encontramos con unos valores muy similares para los tres carriles, es decir, la variabilidad de las mediciones es más consistente. Sin embargo, en el eje X, nos encontramos que existe una mayor variación entre los tres carriles, por lo que podemos considerar que el escenario se comporta peor en el cálculo de la coordenada X.

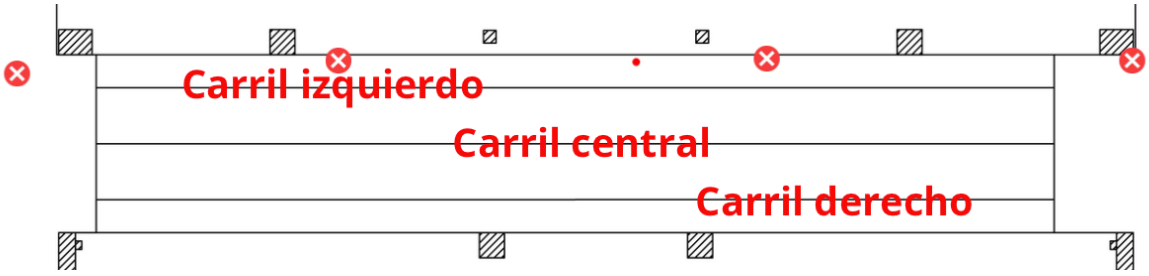

Figura 43. Referencia de los carriles.

Coordenada X	Varianza [m^2]	Desviación estándar [m]
X en carril izquierdo	0.03167	0.1612
X en carril central	0.02608	0.1348
X en carril derecho	0.05688	0.2060

Tabla 3. Varianza y desviación estándar para la coordenada X en la tercera prueba.

Coordenada Y	Varianza [m^2]	Desviación estándar [m]
Y en carril izquierdo	0.009824	0.0914
Y en carril central	0.01782	0.1146
Y en carril derecho	0.02105	0.1174

Tabla 4. Varianza y desviación estándar para la coordenada Y en la tercera prueba.

De igual forma podemos ver reflejado este comportamiento en la Figura 44, en la que se muestra representada la desviación estándar en los dos ejes para cada una de las posiciones de medida. La desviación se encuentra representada sobre el punto medio obtenido. Efectivamente podemos comprobar que en la mayoría de los puntos tenemos mayor deriva en el eje X. Señalar que, como dijimos antes, esta desviación solo concentraría el 68% de las muestras.

Asimismo, también se ha realizado un análisis de la prueba T de Student, una herramienta estadística que se utiliza para determinar si la media de un conjunto de muestras se puede considerar igual a un valor específico, en nuestro caso entre las coordenadas reales medidas y las coordenadas ideales deseadas (cruces de la cuadrícula). Para analizar la prueba obtendremos dos parámetros relacionados entre sí, el valor T y P. El valor T indica la magnitud de la diferencia entre los datos obtenidos y los ideales, para lo cual además se requiere especificar el nivel de riesgo (α) y los grados de libertad (el número de muestras más uno). Por ejemplo para un nivel de riesgo de 0.05 y 30 grados de libertad el valor T debe ser inferior a 2.045. El valor P indica la probabilidad de que la diferencia observada entre los datos ocurra por azar, si el valor obtenido es mayor que el nivel de riesgo (α) sugiere que no hay una diferencia significativa entre los datos ideales esperados y los obtenidos realmente, mientras que si es menor sugiere que existe diferencia.

Para evaluar la prueba T de Student de una muestra se ha utilizado un nivel de riesgo de 0.05 [26]. Dado el número de muestras que tenemos por carril que es superior a mil, podemos considerar que los grados de libertad son infinitos, y en este caso el valor T de referencia es 1.96. Los valores mostrados en la Tabla 5 y la Tabla 6 son una media de todos los valores obtenidos T para dicho carril:

Eje X	T de Student	Valor P [α=0.05]	T límite para obtener α=0.05
Carril izquierdo	97.1381	0.002287	2.0064
Carril central	66.3469	0.001042	2.0114
Carril derecho	98.0218	0.004627	2.0073

Tabla 5. Valores obtenidos de la prueba T de Student para la coordenada X en la tercera prueba.

Eje Y	T de Student	Valor P [α=0.05]	T límite para obtener α=0.05
Carril izquierdo	4.9815	0.008324	2.0064
Carril central	6.1677	0.006077	2.0114
Carril derecho	5.0855	0.002320	2.0073

Tabla 6. Valores obtenidos de la prueba T de Student para la coordenada Y en la tercera prueba.

De igual forma al análisis de la varianza y la desviación, de nuevo podemos ver reflejado que el comportamiento en el eje X es peor al obtenido en el eje Y. Cuanto más alejado T del valor 0, mayor nos indica que es la diferencia entre las medias de los valores ideales y los obtenidos. Asimismo, podemos observar como para sendas coordenadas y carriles, todos los valores P son menores a 0,05, lo que nos indica también que hay una diferencia significativa entre los valores mencionados, y que además los resultados no serían validos o lo suficientemente precisos. Para lograr unos valores precisos para nuestra aplicación, los resultados obtenidos para T deberían haber estado por debajo de los valores existentes en la columna 'T límite para obtener $\alpha = 0.05$'. Esta columna indica el valor límite que podría llegar a alcanzar el valor T, es decir, para ese valor de T se obtendría un valor P igual a 0.05.

Además, se ha representado en la Figura 44 a modo de bandas coloreadas la prueba T de Student. Nos encontramos en color morado la representación del valor T de Student para la coordenada X para cada posición de medida, e igual para la coordenada en Y en color azul. La manera de interpretarla consiste en prestar atención al ancho de la banda, ya que todas poseen igual longitud. Cuanto más ancha sea la banda mayor nos indica que es la diferencia de la coordenada real obtenida frente a la ideal para dicha posición de medida, es decir, peor resultado. Analizando esta figura podemos confirmar que las diferencias entre los valores medidos y los reales son peores en el eje X.

Posteriormente, se ha procedido a analizar las mismas estadísticas para los valores de distancia de cada punto medido con respecto a su posición ideal. En otras palabras, con centro en la posición ideal (cruces de la cuadrícula) formaremos circunferencias siendo el radio la distancia obtenida de cada medición. Los resultados obtenidos son los mostrados en la Tabla 7. En éstos observamos que tanto los radios calculados para el carril izquierdo y central muestran menos variabilidad, es decir, son más consistentes en comparación con el carril derecho. Éste último es el que posee una mayor dispersión en los radios calculados, es decir, pudiera ser el que menos precisión en las mediciones nos proporcionase. En la Figura 45 podemos comprobar estos resultados, en la cual hemos representado el radio medio obtenido para cada una de las posiciones de medida. Se puede observar como en el carril derecho es donde existen mayores variaciones.

Radio	Varianza [m^2]	Desviación estándar [m]
Carril izquierdo	0.04149	0.1939
Carril central	0.04390	0.1898
Carril derecho	0.07794	0.2534

Tabla 7. Varianza y desviación estándar para el radio en la tercera prueba.

Radio	T de Student	Valor P [α=0.05]	T límite para obtener α=0.05
Carril izquierdo	58.5177	2.8824 e-19	2.0064
Carril central	89.4098	6.0836 e-13	2.0114
Carril derecho	117.8626	3.5374 e-12	2.0073

Tabla 8. Valores obtenidos de la prueba T de Student para el radio en la tercera prueba.

Finalmente, también se ha evaluado la prueba T de Student para una muestra de los radios mencionados y los valores obtenidos son los mostrados en la Tabla 8. Obtenemos unos valores más grandes para el carril derecho, los cuales indican que los radios que se han obtenidos tienden a ser significativamente diferentes de los valores ideales. De igual forma, podemos ver reflejados estos resultados en la Figura 45, donde podemos observar unos recuadros proporcionales a los valores de T de Student obtenidos para cada posición de medida. Cuanto mayor sea el recuadro, mayor es la diferencia entre el valor de radio ideal y el obtenido en la prueba, es decir, peor resultado. Pudiendo afirmar entonces que los resultados en carril derecho son menos precisos. Asimismo, se puede observar que los resultados obtenidos para el valor P son considerablemente más pequeños que el nivel de riesgo, lo que nos indica que la diferencia observada no es debida al azar, es altamente significativa e indica que los resultados no son precisos. De igual forma a como se mencionó antes, si quisiéramos lograr unos resultados precisos, los valores obtenidos para T deberían ser iguales o menores al valor mostrado en la columna 'T límite para obtener $\alpha = 0.05$'.

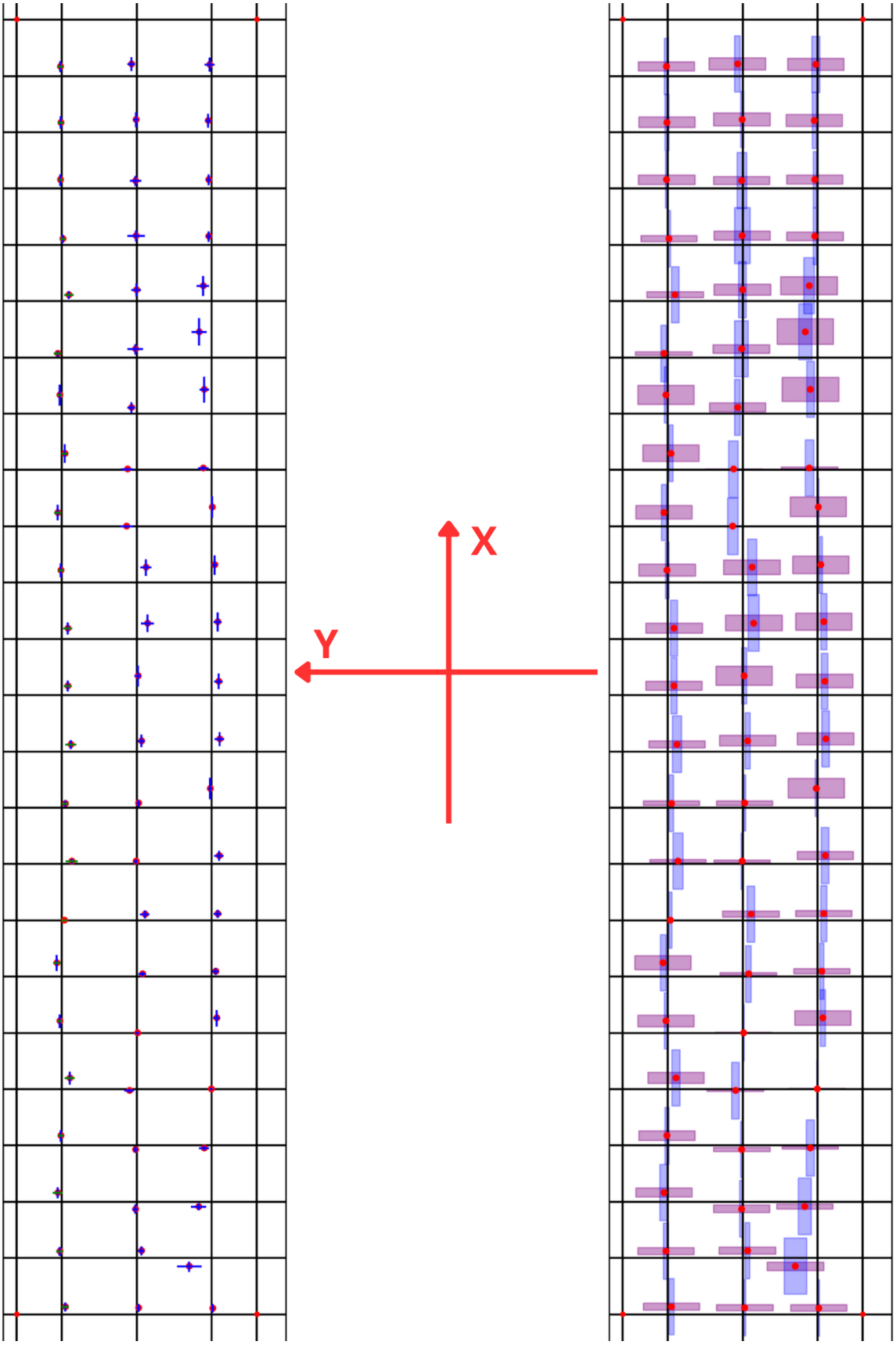

Figura 44. Estadísticas de desviación estándar y puntos medios de localización (izquierda). Estadísticas de la prueba T de Student para cada eje y punto de medida (derecha).

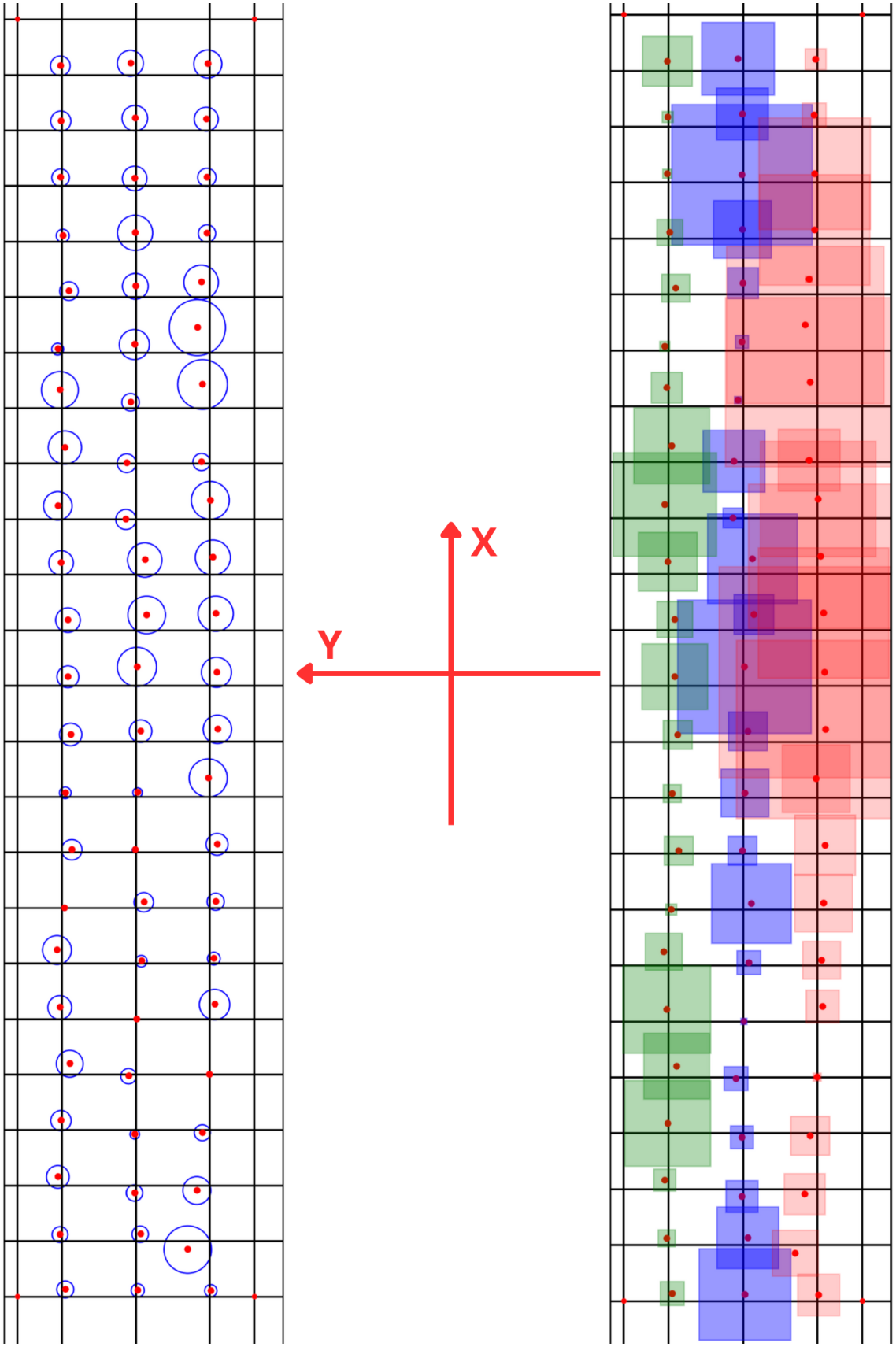

Figura 45. Estadísticas de puntos y radios medios para cada punto de localización (izquierda). Estadísticas de T de Student de los radios para cada punto de medida (derecha).

3.6.7 Discusión de los resultados

Tras todas las pruebas realizadas en el aparcamiento como las realizadas a pequeña escala en el laboratorio, junto con el análisis de los resultados del apartado anterior, ha resultado imposible lograr los resultados prometidos por el fabricante, donde aseguraba una precisión de ± 2cm. Destacar que en las pruebas realizadas en el laboratorio los resultados han sido más satisfactorios que los del aparcamiento, pero puede que no sean útiles para nuestro objetivo, dado que tenemos que asegurar que el vehículo se mueve por el recinto sin riesgos de accidente.

El propio fabricante Marvelmind asegura que la curva de aprendizaje de la herramienta Dashboard es bastante lenta y compleja, afirmación que considero acertada, y que la configuración de parámetros para la obtención de los mejores resultados puede ser complicada. Sin embargo, con los medios disponibles se ha tratado de obtener la mayor precisión posible. Los resultados obtenidos podrían ser de gran utilidad localizar al vehículo dentro de un área concreta, pero no cumpliría con los requisitos necesarios para asegurar la conducción autónoma por parte de un vehículo. Es posible que un futuro contacto con el fabricante pudiera ayudar a conseguir mejorar las estadísticas de localización.

Las causas que pueden hacer que los resultados no sean los esperados pueden ser varias, entre ellas se bajara la posibilidad de que los parámetros de configuración no sean los mejores, la posible existencia de ecos en las señales de ultrasonidos a lo largo del escenario que afecten al filtrado de las señales, así como la problemática que pueden suponer los distintos existentes obstáculos a lo largo de la superficie, como pueden ser columnas, tuberías a lo largo del techo, vehículos estacionados, etc. Por ello, insistimos que quizás para un futuro sería recomendable contar con el asesoramiento del fabricante.

Capítulo 4. Integración con el vehículo

4.1 Introducción

En el presente capítulo se presenta una explicación detallada de cómo acceder a la información de localización directamente a través del propio USB de las balizas, así como el tipo de información que se puede conseguir.

Según se vio en la sección 2.2 sobre el estado previo del proyecto, el vehículo detecta ahora mismo una trayectoria representada por una línea azul en el suelo, y la sigue calculando dos parámetros, la distancia del punto de referencia del vehículo a la trayectoria y la guiñada del vehículo con respecto a la tangete de la trayectoria, también llamado ángulo *yaw*. La distancia con respecto a la trayectoria se puede calcular a partir de la localización que nos da el sistema de balizas, pero hasta el momento no hemos determinado como calcular el ángulo del vehículo. Dada la importancia de este parámetro en nuestro sistema, veremos distintas formas de calcularlo utilizando el sistema de Marvelmind.

Asimismo, veremos cómo nos comunicaremos con el ordenador de control del Renault Twizy, la tecnología y protocolo utilizado para poder proveer la ubicación al vehículo. Finalmente, se propondrá un escenario hipotético en el cual se podrían llegar a obtener unos mejores resultados que los obtenidos durante las pruebas.

4.2 Requisistos del sistema para el control del vehículo

Aunque el vehículo no se encuentra actualmente operativo, se ha querido dejar un módulo *software* que permita utilizar todo el conocimiento adquirido en este trabajo fin de máster para un control del vehículo a través del posicionamiento *indoor*.

Para desarrollar este módulo, se decidió que no se debía modificar el *software* de control del vehículo, dado que no podíamos hacer tests para comprobar que los cambios eran correctos y sin fallos. Desde un punto de vista práctico, esto significa que el objetivo de nuestro programa era sustituir el programa que daba la información de localización del vehículo, llamado '`artemis_autonomous_car.py`'. Este programa analizaba una línea azul pintada en el suelo como ya se explicó en la sección 2.2 sobre el estado del proyecto, y calculaba la distancia del vehículo con respecto a esta línea azul y la guiñada, datos que envíaba al módulo *software* '`mod_con.py`' para el control de la trayectoria. En consecuencia, nuestro *software* simplemente sustituye las operaciones realizadas para el cálculo de la ruta por parte del módulo '`artemis_autonomous_car.py`', enviando directamente a '`mod_con.py`' la información necesaria para el control del vehículo.

Una vez decidido que se iban a mantener las mismas entradas y salidas, se necestiba determinar las necesidades de nuestro nuevo módulo *software*, que se resumen de la siguiente manera:

- El programa se ejecutará en un nuevo ordenador denominado pc-fusión, para evitar la sobrecarga que sufre actualmente el módulo de control.
- Hay que determinar por lo tanto un nuevo modo de comunicación entre los módulos *software*, dado que la nueva versión se encuentra en programas y computadores diferentes, mientras que la anterior se ejecutaba los programas en el mismo ordenador.

- Hay que definir una trayectoria a seguir (que sería equivalente a la línea azul original) y habrá que guardarlo como un mapa que tiene que utilizar nuestro nuevo módulo *software*. Por lo tanto hay que decidir en qué formato se guarda el mapa, relacionado con los conceptos que estudiamos en la sección 2.5.
- Finalmente hay que calcular la distancia a la trayectoria y la guiñada, para lo cual tenemos que decidir qué puntos del mapa son los más cercanos al punto de referencia del vehículo.

Una cuestión que no se ha realizado en este programa es desplazar la localización que da la baliza al punto de referencia del vehículo, que se debería hacer para conseguir un comportamiento exactamente igual que el que se tenía con la cámara. En cualquier caso no es más que una simple traslación de coordenadas que se puede implementar una vez se tenga el vehículo operativo y se quiera utilizar el posicionamiento *indoor* como método de guiado.

Explicados los puntos que se querían completar con el nuevo programa, paso a describir como se implementaron.

4.3 Construcción del mapa

Según lo visto en la sección destinada a la descripción de los mapas y diferentes formas de crearlos (sección 2.5), explicaremos dos planteamientos diferentes sobre cómo definir los mapas de los que se extaerá la trayectoria a seguir por el vehículo. Es conveniente recordar que en dicha sección nos decantamos por el uso de la tecnoloía de lanetets para el almacenaje de mapas. En primera instancia, presentaremos cómo sería la situación ideal, que implicaría la utilización de esto slanelets, para después explicar cómo realmente se ha llevado a cabo la definición de la trayectoria y por qué.

4.3.1 Situación ideal

En un escenario ideal nos interesaría desplegar un mapa definido a través de lanelets, en el cual podamos definir los límites izquierdo y derecho de la trayectoria del vehículo. Por ejemplo, si inicialmente quisiéramos hacer ensayos sobre la cuadrícula de pruebas desplegada en el aparcamiento, la idea sería recorrer toda ella con un recorrido de ida y vuelta, tal y como se puede observar en la Figura 46. De esta forma no solo definiríamos la trayectoria del vehículo, sino que definiríamos el área por el cual podría moverse libremente, es decir, lo abarcado entre los lanelets A y B.

Es intersante observar que los lanelets son solo segmentos. Por lo tanto las zonas curvas requieren de un número relativamente alto de segmentos si se quiere que tengan un aspecto suave. Si se utilizara este sistema en el vehículo sería conveniente tener en cuenta el radio de la curva y la tasa de actualización de la localización del vehículo (ya sea con la localización *indoor* o mediante fusión de sensores con una IMU, que suele ser más rápida) para calcular el número de segmentos necesarios en la descripción de la curva.

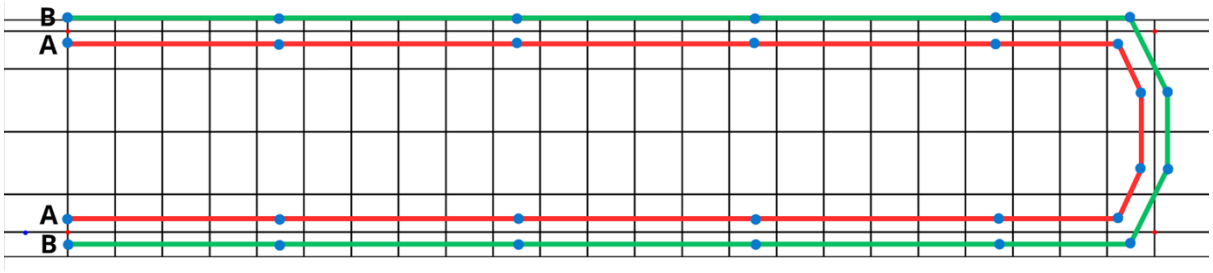

Figura 46. Trayectoria definida a través de lanelets (situación ideal).

4.3.2 Situación real

Debido a que el seguimiento de la trayectoria por parte del vehículo en este momento está basado en el seguimiento de una línea azul sobre el suelo, se ha llegado a la conclusión que sería interesante mantener este tipo de guiado también el mapa. Por lo tanto lo que se plantea es defnir una trayectoria a través de puntos unidos por segmentos. Frente al seguimiento de la línea azul a través de una cámara de detección, eliminaríamos dicho sensor para realizar los cálculos necesarios para el controlar el vehículo a través de las balizas y el ordenador de fusión. De esta forma no sería necesaria la existencia de una línea sobre el suelo, ya que el propio vehículo tendría el conocimiento previo de la trayectoria, definida a través de un vector de puntos. La trayectoria definida es la que nos encontramos en la Figura 47. En el ordenador de fusión se encontrará la trayectoria definida a través de un vector de puntos de puntos, los cuales son unidos a través de segmentos en el momento del cálculo de los parámetros necesarios para el control del vehículo. Por lo que de manera muy sencilla podríamos modificar estos puntos y automáticamente se definirían los segmentos que componen la trayectoria.

De igual forma a como se ha mencionado en la descripción de la situación ideal, en un futuro sería conveniente definir unos puntos que permitan tener una curva más suave y acorde a las características técnicas y de localización del vehículo. Asimismo, dado el estado actual del proyecto, el vehículo no posee ningún algoritmo de cálculo de rutas, sino que ésta se define de manera premeditada, por ello sería de gran utilidad la futura implementación de un algoritmo de planificación del movimiento.

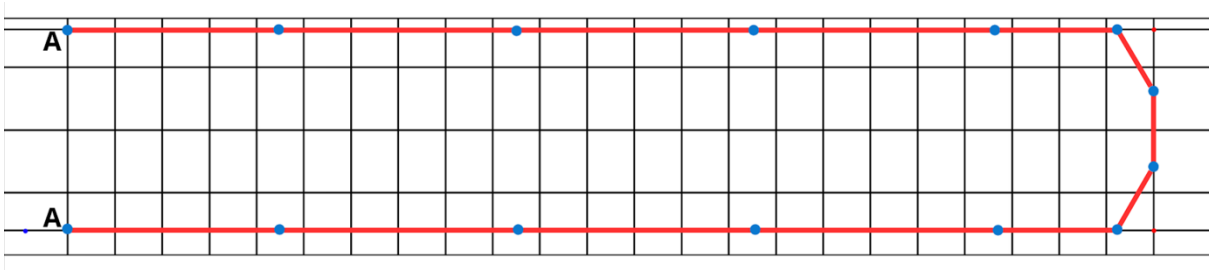

Figura 47. Trayectoria definida a través de puntos (situación real).

Otra cuestión importante en este mapa es que no hemos incoporado bifurcaciones. La incorporación de bifurcaciones e incorporaciones requiería un tratamiento matemático diferente. Se debería establecer un orden entre diferentes segmentos, de tal manera que ya no se podría representar mediante un único vector, sino que se debería presentar como una lista de objetos enlazados. Un objeto puede estar enlazado a dos o más de salida y a dos o más entradas, de tal manera que nos permitiera hacer una planificación de la ruta. Nuestro programa debería

en este caso recibir la ruta planificada, tal y como se realiza en este momento, y construir un vector final con la ruta a seguir, sin tener en cuenta las posibles bifurcaciones.

4.4 Cálculo de parámetros de seguimiento de trayectoria

En todas las pruebas que se han realizado hasta el momento, el ordenador que recogía los datos tenía instalado el software de control de Marvelmind y tenía una pantalla para acceder a dichos datos. Este no va a ser el caso de nuestro vehículo, dado que el software de control va a estar situado en un ordenador fuera del vehículo, y el vehículo no incorpora una pantalla. Por lo tanto, para acceder a la información de localización de la baliza móvil que necesitamos para el cálculo de los parámetros de control de la trayectoria, hay que utilizar un método distinto, que va a ser el acceso a dicha información a través del puerto USB de la propia baliza.

Una vez resuelto este punto, se continua con la explicación de cómo obtener los parámetros y el control final del vehículo.

Figura 48. Escenario de disposición de elementos sobre el vehículo.

4.4.1 Lectura de datos a través de la interfaz USB

Uno de los objetivos de este Trabajo Fin de Máster es proporcionar al vehículo un sistema a través del cual pueda moverse de manera autónoma en un espacio cerrado, esto es, donde no haya presencia de obstáculos móviles y exista un amplio control de los riesgos. En otras palabras, integrar el sistema de localización estudiado en el propio vehículo, pero teniendo en cuenta que no dispondremos de una pantalla para monitorizar su movimiento. Así pues, tras destinar gran parte del esfuerzo a trabajar con las balizas a través de la interfaz gráfica de Marvelmind, el objetivo último es deshacernos de ésta y no requerir de un monitor para poder operar.

Para abandonar nuestra dependencia de la pantalla, habría que leer la información directamente de las balizas móviles en vez de a través del Dashboard y del módem. Sin embargo, hay que tener en cuenta que el módem siempre será necesario, ya que el sistema de Marvelmind requiere obligatoriamente la presencia del módem durante su ejecución. Dada esta necesidad, se podría utilizar un ordenador externo para crear el mapa y permitir la operabilidad de las balizas.

Leer la información directamente de las balizas se puede hacer a través del puerto USB, por lo que conectando las balizas a través de USB al ordenador del vehículo bastaría. De esta forma recaería la responsabilidad de obtener la localización en un ordenador a bordo del vehículo, el ordenador de fusión. En la Figura 48 podemos ver el escenario planteado, donde nos encontramos el ordenador externo ejecutando el Dashboard junto con el módem y el ordenador de fusión conectado a las balizas en el interior del Twizy.

Como ya se mencionó en la sección 3.2.1, las balizas Super Beacon incluyen en su interior una IMU, un dispositivo que permite medir la velocidad, orientación y fuerzas gravitacionales a través de la fusión de sensores giroscópicos, acelerómetros y magnetómetro [27]. Las funciones de cada uno de estos son [28]:

- **Acelerómetro**: mide la aceleración (cambio de velocidad) en una sola dirección. En reposo, mide la fuerza de la gravedad.
- **Giroscopio**: mide la velocidad angular alrededor de sus tres ejes. Mide el ángulo de *yaw*, el cabeceo y el balanceo en un momento dado.
- **Magnetómetro**: mide los campos magnéticos, puede detectar fluctuaciones en el campo magnético de la Tierra. A través de esas fluctuaciones, encuentra el vector hacia el Norte magnético, lo que le da un rumbo absoluto.

Estos sensores generan datos en bruto y tendría que ser la propia IMU la encargada de fusionar la información para proporcionárnosla. Sin embargo, para los modelos de los que disponemos en este momento no es posible obtener los datos fusionados, ya que no está implementado en la actualidad, aunque se espera que en un futuro se implemente esta función.

Hay que destacar que puede ser conveniente calibrar el acelerómetro antes de prestar atención a la información de la IMU, para así mejorar la precisión, reducir la deriva o compensar las interferencias ambientales. Cuando se calibra se miden los datos de caída libre, es decir, la gravedad de la tierra correspondiente a cada eje X, Y, Z, para así poder establecer diferentes valores de corrección.

La propia empresa Marvelmind pone a nuestra disposición un conjunto de ficheros que facilitan la tarea de lectura de la información a través del puerto USB, éstos los podemos encontrar en el mismo paquete *software* descargado para instalar el Dashboard que indicamos en la sección 3.3.1. En el directorio '*19_Examples'* nos encontramos con ficheros preparados en diferentes lenguajes de programación, como C, Java, Arduino o Python, en nuestro caso haremos uso del último. Dado que el ordenador que monta el vehículo trabaja con un sistema Linux, nos dedicaremos a leer la información proveniente de los puertos `ttyACM*`.

El escenario diseñado utiliza idealmente una tasa de refresco ideal de 16Hz, pero nos dimos cuenta de que en función del escenario de trabajo podía variar. Para la situación planteada en el aparcamiento de la universidad ya explicamos que la tasa real obtenida era de 5.4 Hz, es decir, 185 milisegundos cuando la información era leída a través del módem. Sin embargo, realizando la lectura a través de los puertos USB se puede mejorar esta tasa de refresco llegando a obtener una tasa de 23 milisegundos en dicho escenario. En conocimiento de que la tasa de refresco parece depender del número de elementos que operan de manera simultánea, tal y como se dijo en la sección 3.4.4, y que a medida que aumente el número de elementos la tasa de refresco disminuye, no se entiende por qué cuando se realiza la lectura a través del USB se reduce tanto. Suponemos que el sistema está realizando alguna interpretación de los datos que ya tiene disponibles para llegar a obtener dicha tasa de refresco.

Para poder leer la información del puerto USB hay que tener en cuenta que se requerirán unos permisos especiales por motivos de seguridad y de control de acceso. Estos permisos son los siguientes:

```
crw-rw-rw- 1 root dialout 166, 0 Feb 12 09:21 ttyACM0
```

Sin embargo, hay que tener en cuenta que por defecto estos no son los permisos que establece el sistema. Por ello, es necesario modificarlos a través de la siguiente instrucción `sudo chmod 666 ttyACM0`.

A través de la interfaz USB podremos obtener la localización de la baliza en cuestión, las distancias desde dicha baliza a cada una de las balizas fijas que conforman el mapa, junto con todos los datos proporcionados por la IMU. Asimismo, podremos obtener una medida porcentual de la calidad de la localización junto con las marcas de tiempo de cada uno de estos datos. Los códigos python adaptados para la lectura de esta información los podemos encontrar en el Anexo IV.

A continuación, podemos ver un ejemplo:

```
Hedge 11: X: -0.517 m, Y: -0.831 m, Z: 0.000 m, Angle: 494 at time T: 2024-01-24 12:12:47-723

Distances: From:H11 to B4:1.072, B6:1.583, B8:2.539, B0:0.000   at time T: 2024-01-24 12:12:47-703

Quality: Address: 11, Quality: 99%

Raw IMU: AX:-105, AY:-20, AZ:1027,    GX:144, GY:-312, GZ:-77,    MX:0, MY:0, MZ:0,   at time T: 2024-01-24 12:12:47-826

IMU fusion: X:0.000, Y:0.000, Z:0.000, QW:0.914, QX:-0.030, QY:0.044, QZ:0.425,   VX:0.000, VY:0.000, VZ:0.000,   AX:0.000, AY:0.000, AZ:0.000, at time T: 2024-01-24 12:12:47-826
```

Efectivamente como ya hemos mencionado podemos ver como los datos fusionados provenientes de la IMU son nulos a excepción de los cuaterniones. Estos datos obtenidos son los que manipularemos para posteriormente facilitárselos al vehículo.

4.4.2 Cálculo del ángulo de *yaw* y distancia a trayectoria

Explicado en Capítulo 3 cómo obtenter la información de localización de las balizas, pasaremos a explicar cómo a patir de estos datos se puede obtener los parámetros de control del vehículo. Una de las claves en nuestro proyecto es la obtención del ángulo de *yaw* y la distancia con respecto a la trayectoria que se quiere seguir. Sobre la distancia es fácil deducir que es un simple procedimiento matemático a partir de la información de la localización del vehículo y la trayectoria que se desea seguir. Sin embargo, la obtención de la guiñada solo con la información de localización del vehículo no es posible. Por lo tanto veremos dos métodos que se pueden utilizar con el equipamiento de Marvelmind para obtener la guiñada.

Con el equipamiento de Marvelmind hay tres posibles métodos para calcular la guiñada:

- El primer método consiste en acoplar un segundo micrófono a la baliza móvil que se separa mediente un cable unos 50 cm. Esto supone que estaríamos utilizando arquitectura IA, al contrario de la que usamos actualmente que es la NIA. Por lo tanto este método no se llegó a probar.
- El segundo método es utilizar el propio *software* de Marvelmind y dos balizas simultáneas que se emparejan.
- El tecer método es semenjante al anterior, pero en lugar de utilizar el *software* de Marvelmind para emparejar las balizas, se utilizan directamente los datos de las balizas par ael cálculo de la guiñada. Este es el método que finalmente se utilizó porque pse pueden acceder directamente a la información utilizando los puertos USB, tal y como se describió en la sección anterior.

Ángulo de guiñada a través del Dashboard

Como ya se ha mencionado, podemos utilizar el Dashboard para obtener la guiñada del objeto móvil. Para ilustrar cómo obtener éste se ha diseñado un escenario a modo de ejemplo en el laboratorio, tal y como podemos observar en la Figura 49, en el cual, emparejando dos Super Beacon, obtendremos la guiñada.

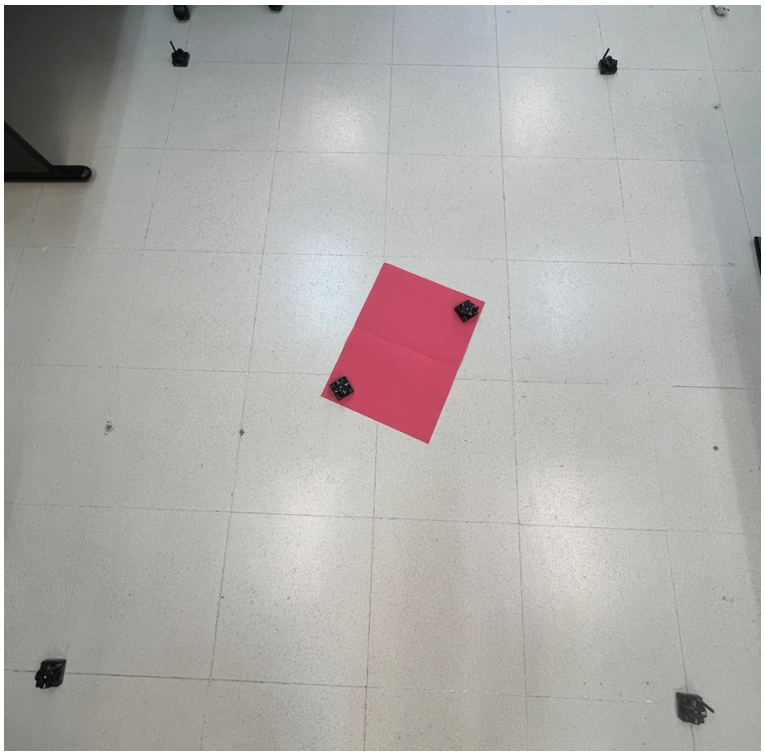

Figura 49. Escenario desplegado en el laboratorio para calcular el ángulo de yaw.

En primer lugar, previo a emparejar las balizas, hay que comprobar que para el mapa diseñado se puede detectar el movimiento de cada *hedgehog* de manera individual, logrado esto podremos entonces emparejarlos. Para ello, a modo de ejemplo podemos disponerlas de una manera similar a la que tenemos en la Figura 50, es decir, debemos ser conscientes de la distancia que las separa y de en qué posición respecto al centro que las une se encuentran. En el caso de la figura tenemos la baliza con dirección número 7 a la derecha y la número 11 a la izquierda, separadas una distancia de 39,5 cm. Destacar que la distancia que las separa debe ser siempre superior a 20cm.

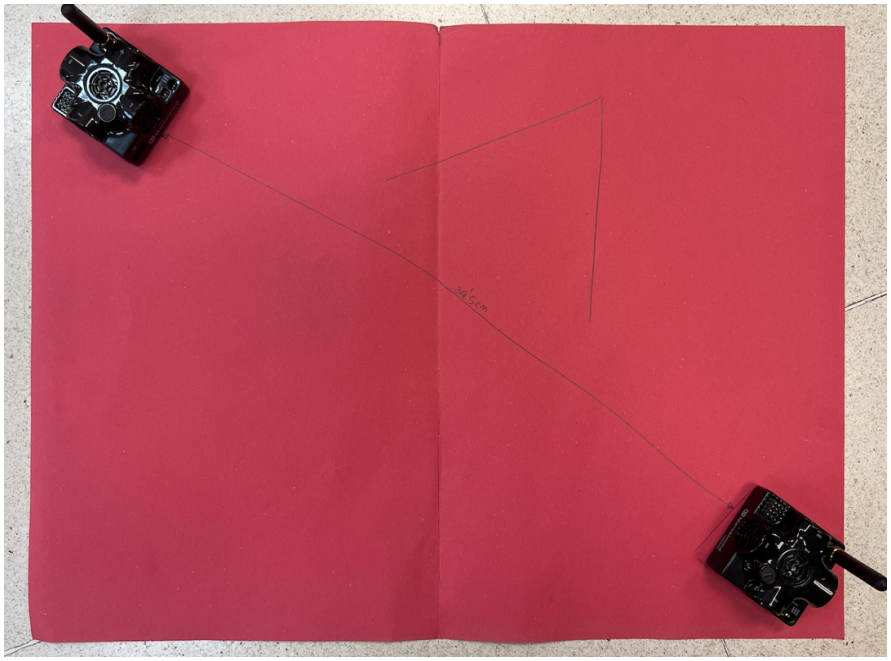

Figura 50. Disposición de balizas para calcular el ángulo de yaw.

Entonces, teniendo ambas balizas móviles operativas, podremos emparejarlas a través del Dashboard con la opción '*Hedgehogs pairing*' como podemos observar en la Figura 51, indicando cuál es el par, la distancia entre ellas y dónde se encuentran.

Hedgehogs pairing	(-) collapse
Pairing mode	pair
Address of paired hedge (1..255)	7
Location againist center	left
Base of the pair, cm (1..255)	39
Heading shift, deg (-90..90)	0
IMU fusion for angle	enabled
Send location of center	enabled
Communication in pair	n/a

Figura 51. Configuración del Dashboard para calcular el ángulo de yaw.

Lograremos así crear la situación que vemos a continuación en la Figura 52, pasando de tener dos balizas individuales sobre al mapa, a crear una estructura en la que se moverán al unísono indicándonos la dirección que siguen.

De igual forma, no solamente podremos obtener visualmente el resultado, sino que obtendremos el resultado a través de su correspondiente fichero '`log.csv`' que registra las tramas de movimiento. Conociendo los campos descritos en la sección 3.5, dos ejemplos de tramas obtenidas serían los siguientes, encontrándonos el *yaw* en el noveno campo expresado entre 0 y 3600 decigrados:

```
T2024_01_17__110847_186,user,41,129,7,0.826,0.878,0.000,2,3006,119
T2024_01_17__110847_371,user,41,129,11,0.837,0.872,0.000,2,3001,121
```

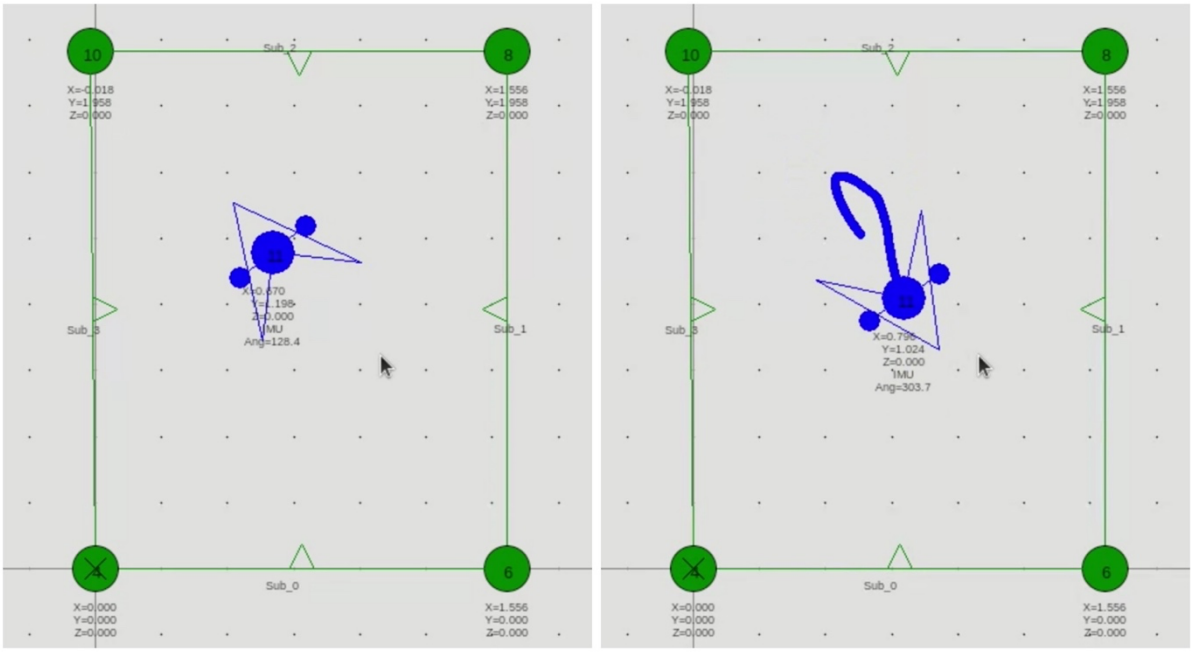

Figura 52. Ejemplo de yaw en el Dashboard.

Ángulo de guiñada a través del puerto USB.

Dado el futuro requerimiento de deshacernos de la utilización del Dashboard en el propio vehículo, nace la necesidad de obtener el ángulo de *yaw* sin utilización de dicho *software*. Por ello, a partir de los códigos proporcionados por la empresa mencionados en la sección 4.4.1, se ha trabajado para poder realizar las tareas deseadas.

En primer lugar, destacar que se ha de preparar el puerto USB para poder ser leído adecuadamente, tal y como se ha mencionado al comienzo de esta sección. Una vez seamos capaces de leer la información cabría esperar la periodicidad que nombramos anteriormente de alrededor de 23 milisegundos. Sin embargo, nos damos cuenta de que esto no siempre es así, analizando las tramas recibidas de los puertos USB vemos que las recibimos de manera algo dispar.

Cuando trabajamos con una sola baliza móvil y leemos los datos de localización a través del puerto USB apenas notamos un cambio en los *timestamps*, rara vez recibimos una trama con una periodicidad mayor. Sin embargo, en el momento en el que realizamos la lectura de datos de dos balizas de manera simultánea, el caso que nos concierne, vemos cómo cambia la periodicidad más a menudo, disminuyendo la frecuencia media.

Para poder calcular el ángulo de *yaw* del vehículo se requiere conocer la posición de las dos balizas y disponer de un plano de referencia. En nuestro caso, se ha definido el mapa mediante coordenadas y en base a dichas coordenadas cuál queremos que sea la dirección 0º. Las balizas se dispondrían sobre el vehículo de la forma que podemos observar en la Figura 53.

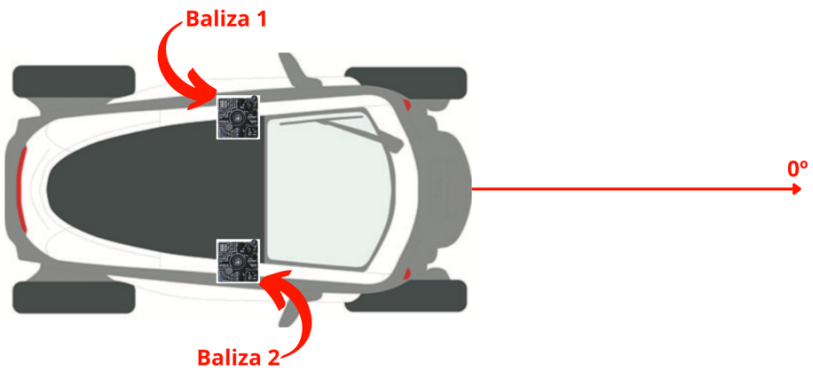

Figura 53. Situación de balizas sobre el Twizy.

Entonces, mediante la utilización de los algoritmos dispuestos de Marvelmind junto con la modificación que se ha realizado para poder leer la información de dos balizas de manera simultánea, algoritmo que podemos encontrar en el Anexo V, obtendremos la siguiente información:

```
Hedge 7: X: -0.517 m, Y: -0.831 m, Z: 0.000 m, Angle: 494 at time T:
2024-01-24 14:22:47-698
Hedge 11: X: -0.517 m, Y: -0.831 m, Z: 0.000 m, Angle: 494 at time T:
2024-01-24 14:22:47-723
```

4.4.3 Cálculo de parámetros de control

Conocidas las localizaciones de ambas balizas móviles, debemos proceder al cálculo de la guiñada y de la distancia con respecto a la trayectoria. Utilizamos como punto de referencia del vehículo para realizar los cálculos el punto medio del segmento que conecta las dos balizas móviles. Hay que recordar, que este no es el punto de referencia del vehículo, y que en el futuro habría que realizar una traslación de coordenadas para referir los cálculos a este punto de referenica.

Como ya se ha comentado en la sección 4.3.2, para realizar las pruebas se ha definido una trayectoria en forma de 'U' que recorra toda la cuadrícula de pruebas, como podemos ver en la Figura 54. Asimismo, sobre esta figura podemos observar cómo hemos definido la referencia de ángulos en el algoritmo de cálculo.

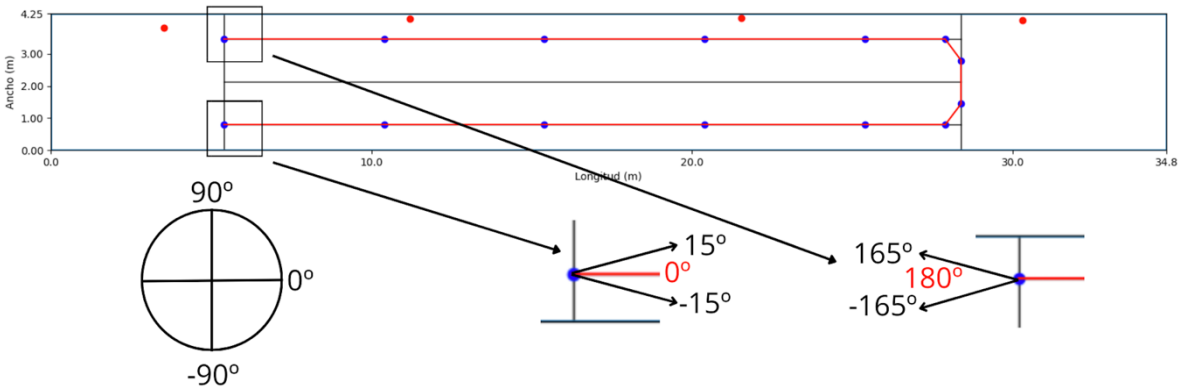

Figura 54. Trayectoria seguida por el vehículo y ángulos de referencia.

El algoritmo se encargará de calcular el punto medio entre las balizas y relativizarlo sobre el mapa. Conocido este punto se calculará qué segmento es el más cercano de todos los que componen la trayectoria, para conocer a qué distancia se encuentra el vehículo de la trayectoria y la guiñada.

El procedimiento es el siguiente; en primer lugar se calculará la distancia más corta desde el punto medio formado por las balizas a cada uno de los segmentos definidos por los vectores de puntos, para así quedarse con la menor de todas las distancias y seleccionar el segmento más cercano. En conocimiento de cuál es el segmento más cercano de manera, se puede conocer cuál es el ángulo formado por los puntos que lo unen, a fin de utilizar relaciones trigonométricas para calcular el formador ángulo del vehículo respecto al segmento y en base a los ángulos de referencia definidos en la Figura 54. Este algoritmo de procesado en tiempo real de la trayectoria del vehículo lo podemos encontrar en el Anexo V.

Se realizaron algunas pruebas a pequeña escala en el laboratorio a la vez que se iba desarrollando el algoritmo, pero para verificar su correcto funcionamiento se realizó la prueba final en el garaje. Se montó el carro de pruebas de la forma que podemos observar en la Figura 55 y se recorrió la trayectoria aplicando el algoritmo de cálculo mencionado.

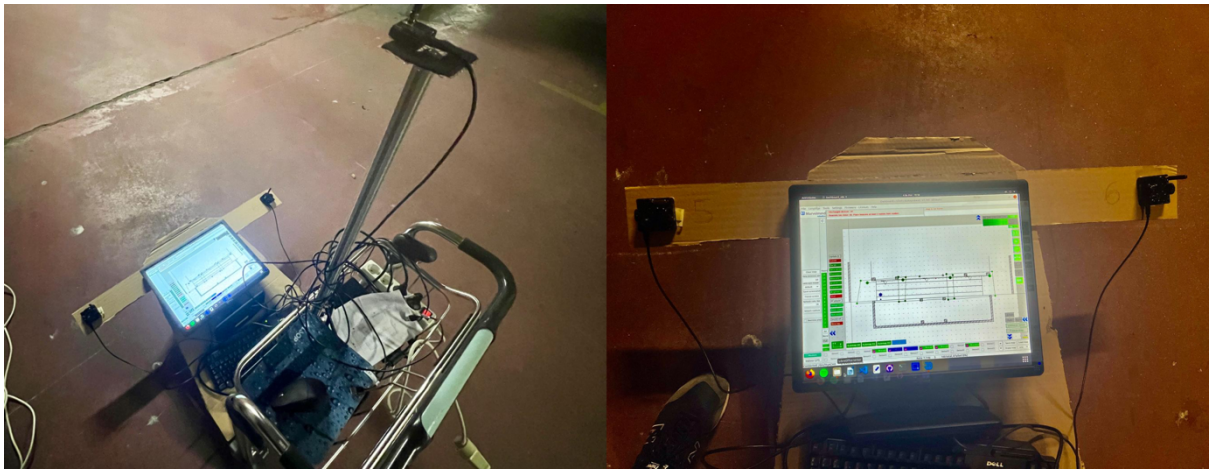

Figura 55. Prueba realizada de seguimiento de trayectoria.

Un ejemplo de los resultados que se obtienen durante la ejecución y que se podrían facilitar al vehículo para autonomizarlo serían los siguientes:

```
El segmento más cercano es el 0 y está a 0.1405 metros del coche, ángulo
del segmento (trayectoria):  0.0 grados

Ángulo coche:  0.90183710704582, en posición: 8.931, 0.6495
```

Asimismo, aplicando un procesado posterior para ejemplificar los resultados obtenidos con la información recogida durante la prueba, podríamos plasmar sobre un mapa el recorrido y resultado total del experimento. En la Figura 56 podemos encontrar el resultado del algoritmo de post procesado utilizado para la creación del mapa, obtenido a través del programa que se encuentra en el Anexo VIAnexo VI.

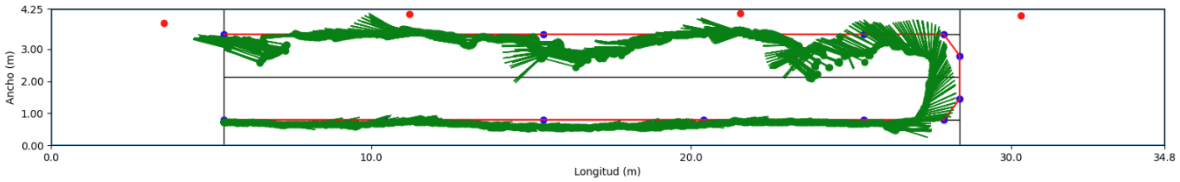

Figura 56. Resultado del seguimiento de la trayectoria del vehículo.

Con puntos verdes nos encontramos la localización proporcionada por el sistema, y en forma de flecha la dirección que sigue el vehículo para esa posición. Analizando los resultados podemos sacar varias conclusiones.

En primer lugar, observamos como claramente el comportamiento en el carril izquierdo es considerablemente peor, mientras que en las anteriores pruebas realizadas se detectó que el comportamiento podría ser peor en el carril derecho. Destacar que durante el desarrollo del proyecto se han ido modificando parámetros de funcionamiento de las balizas, siendo los más acertados los implementados en esta prueba. Enfatizando sobre todo en la modificación de las frecuencias de trabajo de las diferentes balizas, donde en diferentes pruebas a menor escala detectamos una mejoría clara.

Entonces, este empeoramiento puede deberse a diferentes causas, como el ángulo de detección o las interferencias. Además, dado que se encuentra muy próximo a las columnas y encontrarse justamente debajo de las balizas, las reflexiones pueden ser más críticas. Dada la imposibilidad de recrear en este caso la situación de altura ideal del vehículo, al estar las balizas mucho más próximas al suelo, también pueden afectar las reflexiones de las ondas de ultrasonidos sobre éste.

Finalmente, el desarrollo que se ha hecho de este método de control de la trayectoria tiene múltiples cuestiones que pueden ser mejoradas en el futuro. Algunos de los problemas que se han encontrado y que deberían resolverse en el futuro son los siguientes:

- En este momento únicamente estamos calculando el ángulo de *yaw* y la distancia a la trayectoria suponiendo que se va seguir la trayectoria mencionada. Pero en el ejemplo que hemos desarrollado, el vehículo podría saltar de un extremo de la trayectoría en forma de U al otro símplemente porque no se asegura que el vehículo pasa por todos los segmentos, sino que busca el segmento más cercano. Por lo tanto hay que incorporar un seguimiendo de la trayectoria, no solo una búsqueda del segmento más cercano.
- Por otro lado, en el caso anterior, no solo se saltaría una parte de la trayectoria a seguir, sino que además cambiaría de sentido. Por lo tanto habría que incorporar un sentido de movimiento esperado en la trayectoria (hay que recordar que los lanelets resolvían esto de manera adecuada).
- El ángulo de guiñada se calcula a partir de la posición de dos balizas, pero dicha posición nunca es obtenida de manera simultánea, por lo que hay un error debido a el desplazamiento de una baliza con respecto de la otra. En este caso la solución consistiría en incorporar información de la IMU, cuando esté disponible, para calcular la posición simultánea más problable.

4.5 Comunicación con el vehículo

En última instancia, es necesario transmitirle la información de localización procesada de las balizas al ordenador principal a bordo del vehículo, denominado módulo de control. Para poder facilitarle la información se puede hacer a través de varias maneras, por ejemplo, a través de un cable USB o a través de un cable Ethernet. Debido a que el ordenador principal en la actualidad posee todos los puertos USB ocupados, nos decantamos por utilizar Ethernet.

Para mandar la información de un ordenador a otro a través de un cable de red, se pueden utilizar diferentes protocolos y tecnologías, como puede ser MQTT, SSH o sockets. En nuestro caso, dado que previamente en el proyecto ya se habían llevado a cabo comunicaciones a través de sockets, decidimos utilizar esta misma herramienta.

Un socket es una interfaz de comunicación que permite comunicar dos procesos, que pueden estar en la misma máquina o en diferentes, y la comunicación puede ser unidireccional o bidireccional. En un modelo basado en sockets, uno de los procesos debe actuar como servidor y el otro como cliente. El servidor crea un socket y espera la conexión entrante del cliente, el cliente crea también un socket y se conecta al del servidor. En otras palabras, para que la comunicación a través de sockets sea posible, uno de los procesos debe estar esperando conexiones entrantes, es decir, el servidor, mientras que el cliente debe iniciar la conexión. Una vez conectados, pueden intercambiar la información [29].

Las direcciones IP destinadas a redes locales y privadas han de tener la estructura 192.168.X.X. De esta manera se configuró el servidor con la dirección 192.168.0.3 y el cliente con 192.169.0.7, ambas con máscaras 255.255.255.0. En la Figura 57 podemos ver reflejado el esquema general que se ha dispuesto.

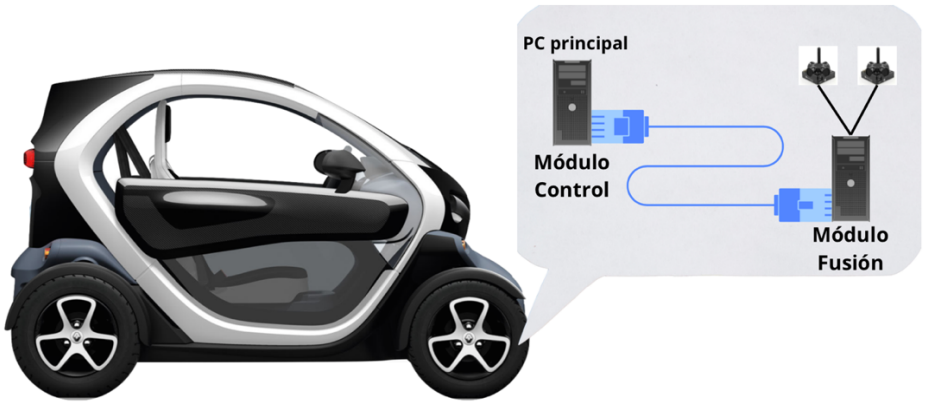

Figura 57. Esquema de conexión entre ordenadores del vehículo.

En este esquema podemos ver un ordenador en donde se encuentran todos los servicios de percepción y que se ha denominado en los diferentes esquemas de TwizyLine Módulo de Fusión. Por otro lado este servidor se conecta con otro ordenador que contendrá al cliente que recibe los datos procesados de la fase de percepción y que en términos de TwizyLine recibe el nombre de Módulo de Control, el cual, como su nombre indica, se encarga de controlar todos los actuadores que permiten controlar el movimiento del vehículo de manera autónoma.

En el Anexo VII se puede encontrar el código desarrollado e implementado para realizar la conexión e intercambio de información, pero este sería un ejemplo de la información que están

intercambiando ambos ordenadores. En el lado del servidor, quien gestiona las balizas, nos encontramos:

```
twizyline@twizyline-ZBOX-QCM7T3000-EN072080S-EN072070S-
EN052060C:~/Desktop/TFM       Oscar/TFM_Balizas_Lidar/Anexos/Anexo_VII$
python3 envioInfoSocketTiempoReal.py
Trying open serial port: /dev/ttyACM1
Trying open serial port: /dev/ttyACM2
Serial port opened
Esperando por una conexión...
Serial port opened
Conexión establecida desde: ('192.168.0.7', 60554)
[SERVER] Envío: Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.710704958511784 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.710704958511784 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.631805862824393 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.631805862824393 y
Distancia a trayectoria: 1.413
[SERVER] Envío: Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[SERVER] Envío: Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[SERVER] Envío: Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[SERVER] Envío: Angulo respecto a trayectoria: 31.534791905188285 y
Distancia a trayectoria: 1.415
[SERVER] Envío: Angulo respecto a trayectoria: 31.534791905188285 y
Distancia a trayectoria: 1.415
[SERVER] Envío: Angulo respecto a trayectoria: 31.565181851223727 y
Distancia a trayectoria: 1.4145
```

Por parte del cliente, la información recibida es:

```
artemis@artemis-NUC12DCMv9:~/Desktop/socket$                    python3
recepcionInfoSocketTiempoReal.py
Conexión establecida con el servidor
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.868101096467953 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.710704958511784 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.710704958511784 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.631805862824393 y
Distancia a trayectoria: 1.413
```

```
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.631805862824393 y
Distancia a trayectoria: 1.413
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.61355632267535 y
Distancia a trayectoria: 1.4140000000000001
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.534791905188285 y
Distancia a trayectoria: 1.415
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.534791905188285 y
Distancia a trayectoria: 1.415
[CLIENT] Recibo:  Angulo respecto a trayectoria: 31.565181851223727 y
Distancia a trayectoria: 1.4145
```

4.6 Diseño del despliegue

En un principio, en base a las características de funcionamiento proporcionadas por el fabricante, se pensaba desplegar el sistema en todo el aparcamiento, tal y como ya se ha mencionado. Sin embargo, a pesar de que se ha visto que la precisión real del sistema es bastante peor a la que la empresa Marvelmind declara, se ha decidido estudiar cómo debería ser un despliegue para poder dar localización *indoor* al vehículo con suficiente precisión. Por ello, tras todo lo aprendido durante el presente proyecto, se pasará a presentar un diseño final de despliegue que haría funcionar al sistema de la manera adecuada.

En primer lugar, se ha visto que las condiciones de trabajo han de ser demasiado ideales para que se pueda soportar el alcance máximo de las balizas. Algunas de estas condiciones ideales pueden ser tener una línea de visión directa, evitar zonas de reflexión, obstáculos que provoquen zonas de sombra o vehículos metálicos que interfieran en las señales. Estas condiciones son sumamente complicadas de conseguir en nuestro escenario de trabajo y dado que se requiere de una precisión muy elevada debido a la propia naturaleza del proyecto, pensamos que se deberían desplegar muchas más balizas por toda la superficie.

En el mejor escenario que hemos simulado desplegamos las balizas a unas distancias dispares, pero entre los 7 y 10 metros en ellas, y aun así los resultados obtenidos no fueron suficientemente precisos. Por ello, proponemos desplegar una baliza fija cada 4 metros, donde cada *array* de balizas se dedica a cubrir un único carril, podemos observar la propuesta en la Figura 58:

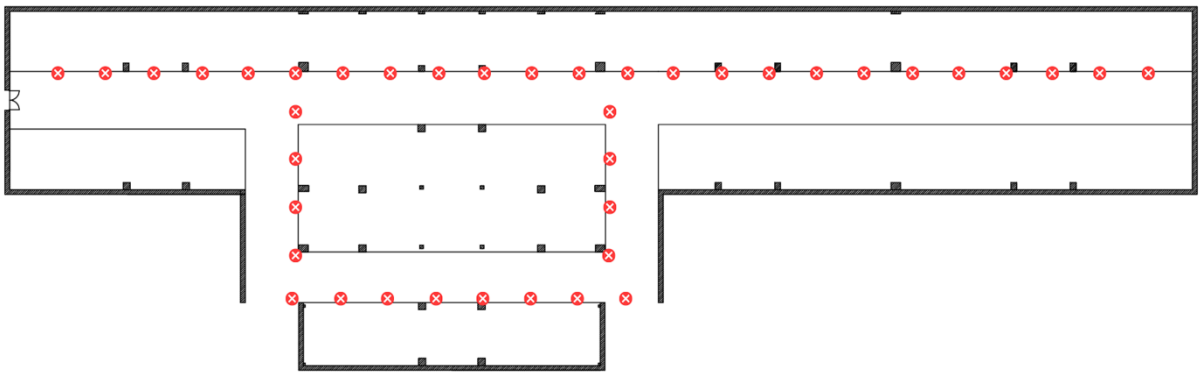

Figura 58. Disposición de las balizas en situación ideal en el mayor escenario posible.

Asimismo, en la actualidad las balizas se han fijado directamente al techo, lo cual puede producir reflexiones, y encima se encuentra con obstáculos de visión debido a las numerosas tuberías y diferentes elementos que recorren el aparcamiento a dicha altura. Por ello, se propone bajar la altura de éstas con un trípode o similares, tal y como vemos en la Figura 59.

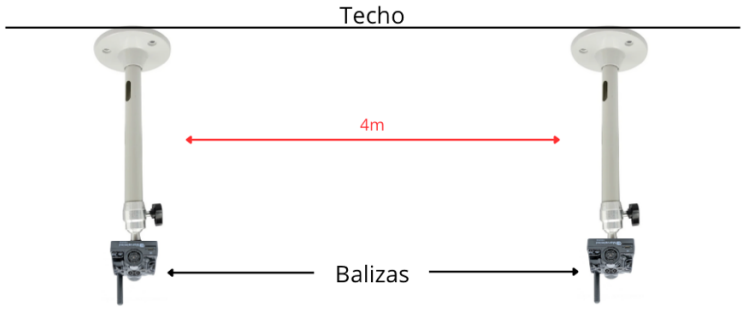

Figura 59. Ejemplo de trípode recomendado para las balizas.

No obstante, esta disposición aumentaría de manera muy considerable el presupuesto del proyecto, por lo que sería conveniente analizar su viabilidad.

Para poder abarcar toda la superficie del aparcamiento con la distancia entre balizas mencionadas, harían falta 42 balizas, junto las 2 que irían sobre la superficie del vehículo, 44 balizas. Tal y como podemos observar en la Tabla 9, esto supondría un coste muy elevado.

	Precio por unidad	**Unidades**	**Coste**
Balizas Super Beacon Indoor	119 €	44	5236 €
Trípodes	4,36 €	44	191, 84 €
			5427, 84 €

Tabla 9. Costes asociados al despliegue ideal con balizas.

Dado el elevado presupuesto convendría buscar alternativa que permitiera conseguir unos buenos resultados y que permitiera reducir el coste, aunque hubiera que incluir nuevas tecnologías.

El departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática tiene en su inventario un LiDAR RoboSense RS-LiDAR-16, por lo que consideraré éste como la opción para cubrir nuestras necesidades. El problema que tiene el LiDAR es que funciona con GPS, técnica no recomendable en nuestro caso al trabajar en un entorno subterráneo. Dada las propias limitaciones de GPS y de nuestro escenario, sería imposible para el LiDAR conocer su localización, por lo que puede ser interesante fusionar las balizas y la tecnología láser.

Puesto que hablamos de manejo de vehículos destinados a transportar personas, la seguridad es un pilar fundamental, por lo que incorporar el LiDAR podría ayudarnos de manera efectiva a detectar los obstáculos de manera precisa y reducir así la precisión que necesitamos para hacer funcionar el sistema. Por ello, proponemos reducir el número de balizas a pesar de perder precisión en la localización e incorporar el LiDAR. En vista a los resultados obtenidos en las pruebas, una distancia entre balizas de 8 metros junto con su trípode adjunto podrían ser suficientes. Disminuir el número de balizas a la mitad supondría una reducción en el

presupuesto de 1.579, 92 €, por lo que sería una mejor opción. A continuación, podemos encontrar las cifras en la Tabla 10.

	Precio por unidad	**Unidades**	**Coste**
Balizas Super Beacon Indoor	119 €	22	2618 €
Trípodes	4,36 €	22	95, 92 €
RoboSense RS-LiDAR-16	1.134 €	1	1.134 €
			3847, 92 €

Tabla 10. Coste asociado al despliegue ideal con balizas y LiDAR.

Capítulo 5. Conclusiones y líneas futuras

A continuación, se detallarán aquellas conclusiones que se han sacado en clave durante el transcurso del trabajo y se plantearán diferentes implementaciones futuras que pueden ayudar a mejorar el proyecto.

5.1 Conclusiones

El principal objetivo de este Trabajo Fin de Máster consistía en lograr la localización precisa de un Renault Twizy para mejorar su seguimiento de una trayectoria. Para ello, se investigó una de las tecnologías más prometedoras para el escenario del que disponemos: las balizas de ultrasonidos. A pesar de que las especificaciones de estas balizas establecen una preción de ±2cm, precisión muy superior a la necesaria, los resultados obtenidos no cumplieron con las expectativas.

Durante el desarrollo del proyecto, nos enfrentamos al hecho de que las balizas de ultrasonidos eran extremadamente sensibles a factores externos, como puede ser la presencia de obstáculos que impidan la visión directa, zonas de sombra y reflexiones. A pesar de estos desafíos, se exploraron diversas alternativas para maximizar los resultados con los recursos disponibles, como distintas formas de despliegue de las balizas o configuraciones que nos permitieran obtener una mejor tasa de refresco, objetivos que fueron alcanzados.

Inicialmente, nos propusimos cubrir una superficie demasiado extensa, para la cual se suponía que las balizas serían adecuadas según sus especificaciones. Sin embargo, a lo largo del proyecto, tuvimos que reducir dicha superficie hasta en dos ocasiones debido a las condiciones desfavorables del escenario, que impedían alcanzar una precisión satisfactoria con el requisito final, la seguridad del pasajero.

Dada la incompatibilidad entre los datos recopilados por las balizas y la información requerida por el vehículo, fue necesario realizar una transformación y optimización de los datos para lograr la interoperabilidad. Además, se desarrollaron diversos programas y métodos para evaluar el rendimiento del sistema, incluyendo análisis estadísticos y la generación de gráficos, que se espera sean de gran utilidad en futuras etapas del proyecto.

Ante la discrepancia entre los resultados obtenidos y los objetivos iniciales, también se investigó la posibilidad de mejorar la localización del vehículo autónomo, así como un análisis de viabilidad económica. Se concluyó que, para alcanzar los objetivos de cobertura inicialmente planteados, sería necesario emplear un mayor número de dispositivos de localización o combinar las balizas con otras tecnologías, como el LiDAR.

Finalmente, en cuanto a la integración con el *software* ya desarrollado del vehículo, debido a que el Renault Twizy no se encuentra operativo en el momento del desarrollo del proyecto y lo desarrollado no se podría probar, se ha preparado para que su integración con los módulos del vehículo sea muy sencilla. Dado que partimos con la ventaja de que la información de localización obtenida de las balizas y el procesamiento de dicha información para facilitársela al vehículo se realizan en un ordenador externo, bastaría únicamente con insertar las líneas preparadas en el lugar adecuado para lograr así la conexión mediante sockets entre los módulos de control y fusión, para así asignar los valores recibidos a las variables de guiñada y distancia a trayectoria.

5.2 Líneas futuras

Sería muy emocionante llevar a cabo nuevos desarrollos en el proyecto con tal de mejorar el sistema. Por ello, vamos a plantear algunas ideas de desarrollo futuras que podrían ayudar a optimizar los resultados.

- Al hablar de vehículos autónomos es necesario tener en cuenta que la seguridad es lo más importante, por ello sería elemental mejorar los resultados de localización del vehículo. Dado que se ha visto que estos resultados dependen de la tecnología utilizada, las balizas, y del escenario, el aparcamiento, sería conveniente considerar aplicar mejoras en ambas. Una mejoría en las balizas vendría dada por un aumento del número de dispositivos que se utilizan, ya que se ha visto el número de dispositivos empleados en el proyecto son escasos. En cuanto al aparcamiento, dado que la viabilidad de situar las balizas sobre el techo no es la óptima, sería recomendable utilizar una serie de trípodes, tal y como se propone en la Figura 58, que ayudasen a saltar obstáculos y posibles reflexiones de las señales.
- Dado que la curva de aprendizaje con las herramientas de Marvelmind es bastante complicada, sería recomendable para futuras mejoras ponerse en contacto con la compañía para obtener los mejores parámetros de configuración posibles en el escenario por el que se va a mover el Renault Twizy.
- En caso de que se quisieran utilizar tecnologías complementarias, se propone la incorporación de un LiDAR que permitiera aumentar la seguridad en la detección de obstáculos, y poder así combinar su información con la obtenida de las balizas de ultrasonidos.
- Implementación de un algoritmo de planificación y cálculo de rutas que permita definir la trayectoria de manera automática, indicándole al ordenador a bordo del vehículo cuál es el destino al que quiere llegar. Asimismo, realizar mejoras sobre el algoritmo de seguimiento de la trayectoria para tener en cuenta posibles errores que actualmente no se están teniendo en cuenta, como por ejemplo, incorporar un control del sentido que se espera que recorra el vehículo sobre la trayectoria.
- El ángulo de guiñada se calcula a partir de la posición de dos balizas, pero dicha posición nunca es obtenida de manera simultánea, por lo que hay un error por el desplazamiento de una baliza con respecto de la otra. En este caso la solución consistiría en incorporar información de la IMU, en el momento que esté disponible por parte de la empresa, para calcular la posición simultánea más problable.
- Finalmente, se esperaba que el vehículo pudiera estar funcionando en etapas finales del proyecto que permitieran realizar las pruebas sobre el propio Renault Twizy. Sin embargo, a pesar del trabajo realizado en paralelo para poner en marcha el vehículo, no se ha conseguido dado la envergadura del proyecto. Por ello, se encomienda la futura realización de las pruebas sobre el propio vehículo.

Capítulo 6. Bibliografía

[1] «Global Road Safety Statistics,» *Brake the road safety charity,* 2023.

[2] A. M. Hernández, «Front-end implementation for an automatized car parking,» Universidad de Valladolid, 2020.

[3] I. Royuela, «Four level autonomous vehicle for an automatized parking,» Universidad de Valladolid, 2020.

[4] M. Martín, «Plan de Comunicación de TwizyLine plataforma de carshring con aparcamiento autónomo,» Universidad de Valladolid, 2020.

[5] S. P. Arnanz, «Back-end implementation for an automatized car parking,» Universidad de Valladolid, 2020.

[6] I. V. Vázquez, «Integración de sistemas de posicionamiento indoor para un testbed de Edge Computing para el soporte de vehículos conectados,» Universidad de Valladolid, 2022.

[7] C. G. Diego, «Guiado de vehículo autónomo mediante tecnología LiDAR,» Universidad de Valladolid, 2022.

[8] C. Engelking, «The 'Driverless' Car Era Began More Than 90 Years Ago,» *Discover,* 2017.

[9] N. E. a. Y. Li, «Indoor navigation: state of the art and future trends,» *Satellite Navigation,* 2021.

[10] M. V. Capilla, «Sistema de posicionamiento en interiores,» Universidad Politécnica de Madrid, 2018.

[11] A. C. Sanz, «Sistemas de posicionamiento basados en WiFi,» Universitat Politècnica de Catalunya, 2006.

[12] Inpixon, «Bluetooth RTLS, Location Tracking, & Positioning,» https://www.inpixon.com/technology/standards/bluetooth-low-energy, 2022. [En línea]. Available: Inpixon. [Último acceso: Mayo 2024].

[13] L. M. Calero, «Sistema de posicionamiento en interiores con tecnología UWB,» Universidad de Jaén, 2019.

[14] Y. G. S. K. Kelving Wong, «Mapping for Autonomous Driving,» 2019.

[15] J. I.-G. y. C. Z. Alexandre Armand, Digital Maps for Driving Assistance Systems and Autonomous Driving, 2016.

[16] J. Z. y. C. S. Philipp Bender, «Lanelets: Efficient map representation for autonomous driving,» 2014.

[17] J. C. y. K. S. Constantin Wellhausen, «Efficient Grid Map Data Structures for Autonomous Driving in Large-Scale Environments,» 2021.

[18] M. Robotics, «Super Beacon,» Marvelmind, 2024. [En línea]. Available: https://marvelmind.com/product/super-beacon/. [Último acceso: Mayo 2024].

[19] M. Robotics, «Starter Set Super MP-3D,» Marvelmind, 2024. [En línea]. Available: https://marvelmind.com/product/starter-set-super-mp-3d/. [Último acceso: Mayo 2024].

[20] Wikipedia, «Aircraft principal axes,» 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Aircraft_principal_axes. [Último acceso: Mayo 2024].

[21] M. Á. Á. González, «Levantamiento de planos arquitectónicos,» Universitat Politècnica de València, 2022.

[22] Marvelmind, «Marvelmind Indoor Navigation System Operating Manual,» 2024.

[23] Marvelmind, «Hardware interfaces and protocols of data exchange with Marvelmind devices,» 2023.

[24] Cuemath, «Standard Deviation,» 2023. [En línea]. Available: https://www.cuemath.com/data/standard-deviation/. [Último acceso: Abril 2024].

[25] Cuemath, «Variance,» 2023. [En línea]. Available: https://www.cuemath.com/data/variance/. [Último acceso: Abril 2024].

[26] P. d. f. e. JMP, «La prueba T de una muestra,» 2023. [En línea]. Available: https://www.jmp.com/es_es/statistics-knowledge-portal/t-test/one-sample-t-test.html. [Último acceso: Mayo 2024].

[27] A. Navigation, «Inertial Measurement Unit (IMU) – An Introduction,» 2024. [En línea]. Available: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/. [Último acceso: Abril 2024].

[28] A. Kalnoskas, «What is Sensor Fusion?,» Sensor Tips An EE World Online Resource, 2021. [En línea]. Available: https://www.sensortips.com/featured/what-is-sensor-fusion-faq/. [Último acceso: Abril 2024].

[29] T. J. Tutorials, «What is a Socket?,» Oracle, [En línea]. Available: https://www.sensortips.com/featured/what-is-sensor-fusion-faq/. [Último acceso: Abril 2024].

Anexo I

En primer lugar, para poder crear los diferentes mapas con la localización del vehículo, es necesario tomar todas las capturas de localización a través del sistema de ficheros CSV implementado en el Dashboard de Marvelmind.

Todos los códigos están adaptados a la cuadrilla de pruebas diseñada en el aparcamiento de la facultad, por lo que en la mayoría de los códigos habrá una distinción entre los tres carriles existentes, diferenciados como '`dcha`', '`ctro`' e '`izq`'. Nos encontraremos un directorio por cada carril con los 23 ficheros CSV correspondientes a cada punto de la cuadrilla, tal y como podemos observar en la Figura 60.

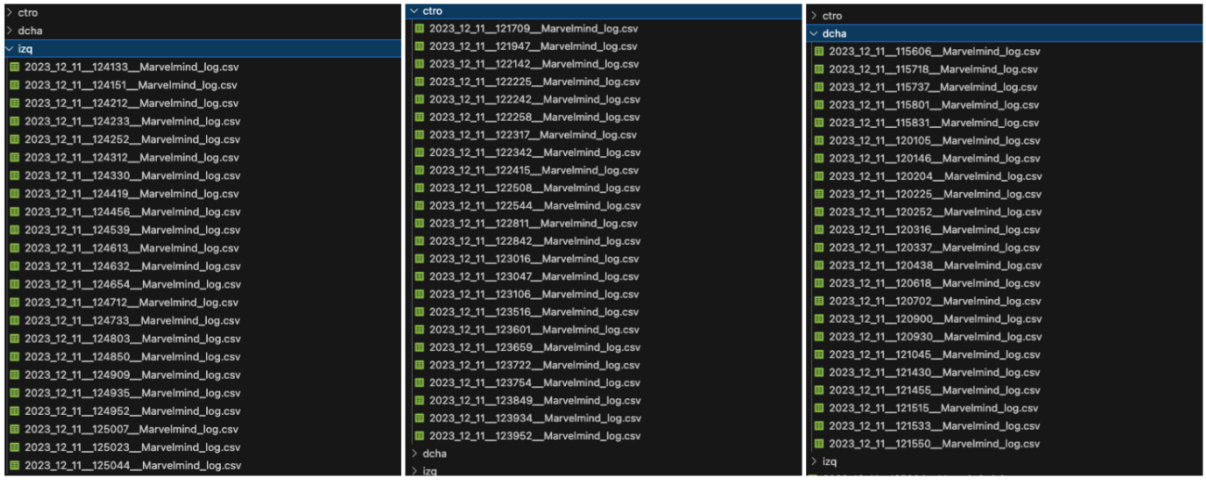

Figura 60. Estructura de ficheros del Anexo I.

En caso de detectar en dichos ficheros que se ha producido una deriva hacia el frente como se mencionó en la memoria, existe un fichero denominado '`eliminarDeriva.py`' encargado de eliminarla. Si es necesario, tendremos que hacer uso de él antes de pasar a procesar los ficheros. El código es el siguiente:

```
import os

# Directorios a analizar
directorios = ["ctro", "dcha", "izq"]

# Función para eliminar contenido a partir de la línea 51
def eliminar_contenido(fichero):
    with open(fichero, "r") as f_in:
        lineas = f_in.readlines()

    # Lista para almacenar las primeras 50 líneas
    lineas_a_escribir = lineas[:60]

    with open(fichero, "w") as f_out:
        f_out.writelines(lineas_a_escribir)

# Recorrer directorios y ficheros
for directorio in directorios:
    for fichero in os.listdir(directorio):
        if fichero.endswith(".csv"):
            ruta_fichero = os.path.join(directorio, fichero)
            eliminar_contenido(ruta_fichero)
```

```
print("Proceso finalizado")
```

En los ficheros CSV nos encontramos con muchas tramas erróneas y tramas que no necesitamos, por ello, deberemos procesarlos para quedarnos con la información que nos interesa. Para poder llevar a cabo esta operación utilizaremos el código del fichero '`procesarFicheros.py`'. Su código es el siguiente:

```
import os
import csv
import pandas as pd
import math
from pathlib import Path

carriles = ['dcha/','izq/', 'ctro/']

for carril in carriles:

    # ficheros = os.listdir('ctro/')
    ficheros = os.listdir(carril)
    print("El número de ficheros encontrados: ", len(ficheros))



    #### ------------------------------ ####
    #### ------ Ordenar ficheros ------ ####
    #### ------------------------------ ####
    horas = []
    for nombre in ficheros:
        aux = nombre.split(sep='_')
        horas.append(aux[4])
    horas = sorted(horas) # CSV ordenados

    # print("Horas de las medidas ordenadas: ", horas)

    ficherosOrdenados = []
    for hora in horas:
        for fichero in ficheros:
            if hora in fichero:
                ficherosOrdenados.append(fichero)

    # print("Ficheros ordenados para extracción: ", ficherosOrdenados)

    #### ------------------------------ ####
    #### ----- Seleccionar trama ----- ####
    #### ------------------------------ ####

    def calcularRadio(x2, y2, coordenadasBase, distanciaABase):
        # print("Los puntos medidos son: ", x2, ', ', y2)
        centroTeorico = [coordenadasBase[0] + distanciaABase, coordenadasBase[1]]
        x1, y1 = centroTeorico[0], centroTeorico[1]
        # print("El centro teórico es:", centroTeorico, '[metros]')

        radioDelPuntoMedido = math.sqrt((float(x2) - float(x1))**2 + (float(y2) -
float(y1))**2)
        # print(f"El radio entre los puntos es: {radioDelPuntoMedido}")
        # if radioDelPuntoMedido < 0.6:
        #     return radioDelPuntoMedido
        # else: return None
        return radioDelPuntoMedido

    def comprobarSiInteresaLaTrama(trama):
        if trama[2] == '41' and trama[3] == '129' and trama[4] == '11': # Trama que
corresponde al hedgehog
```

```
            if trama[5] != 'na' and trama[6] != 'na' and trama[7] != 'na':
                return True
        else: return False

    def extraerInfoTrama(trama):
        X, Y, Z = trama[5], trama[6], trama[7]
        return X, Y, Z

    # El carril derecho está desplazado a 90 cm en el eje X de la cuadrícula (0,0 de
nuestra cuadrícula dibujada, (OJO, no es el mismo que el (0,0)))
    if 'ctro' in carril:
        coordenadasBase = [0.9, 2.13 + 0.01, 1.47] # X e Y en metros (Distancia al
0,0)
    if 'dcha' in carril:
        coordenadasBase = [0.9, 0.80 + 0.01, 1.47] # X e Y en metros (Distancia al
0,0)
    if 'izq' in carril:
        coordenadasBase = [0.9, 3.46 + 0.01, 1.47] # X e Y en metros (Distancia al
0,0)

    distanciaABase = 0 # metros que avanzo en eje X

    indice = 0
    encabezados = ['Índice', 'Radio', 'Coordenada X', 'Coordenada Y', 'Coordenada
Z', 'X exacto', 'Y exacto', 'Z exacto']

    # Ruta del directorio a crear
    directorio_nuevo = carril + 'procesados/'
    # Crear el directorio usando pathlib
    path_directorio_nuevo = Path(directorio_nuevo)
    path_directorio_nuevo.mkdir(parents=True)

    for fichero in ficherosOrdenados:

        ruta = carril + fichero
        df = pd.DataFrame(columns=encabezados)

        with open(ruta, 'r') as csvFile:
            lector_csv = csv.reader(csvFile)

            # Usamos enumerate para obtener tanto el índice como la trama
            for num, trama in enumerate(lector_csv, start=1):
                indice = indice + 1
                if comprobarSiInteresaLaTrama(trama):
                    X, Y, Z = extraerInfoTrama(trama)
                    radio = calcularRadio(X, Y, coordenadasBase, distanciaABase)
                    # print(radio)
                    if radio != None:
                        info  =  [indice,  radio,  X,  Y,  Z,  coordenadasBase[0]  +
distanciaABase, coordenadasBase[1], coordenadasBase[2]]
                        df       =       pd.concat([df,       pd.DataFrame([info],
columns=encabezados)], ignore_index=True)
                    else:
                        pass

        distanciaABase = distanciaABase + 1 # Aumento en un metro la coordenada X
        ficheroDestino = carril + 'procesados/Processed_' + fichero
        # Escribir en un archivo CSV
        df.to_csv(ficheroDestino, index=False)
        indice = 0
```

A través de este código habremos generado dentro de los directorios de los carriles una carpeta denominada ‘procesados’, en la cual nos encontramos un fichero por cada posición medida con la información relevante, como la localización y el radio que proporciona dicha localización.

Finalmente podríamos pasar a generar el mapa con las localizaciones representadas en colores para conocer cómo se ha comportado el sistema. Sin embargo, debido a que nos podemos encontrar *outlayers* en las localizaciones, en el fichero denominado '`creacionMapa.py`' se ha tenido en cuenta una variable destinada a eliminarlos si quisiéramos. Manipulando la variable `representarOutlayers` podremos controlarlo y cambiando el valor `radioEliminarOutlayers` podremos definir el radio límite. Si quisiéramos no eliminar los *outlayers* para ver el comportamiento real, obtendríamos algo similar a la Figura 61.

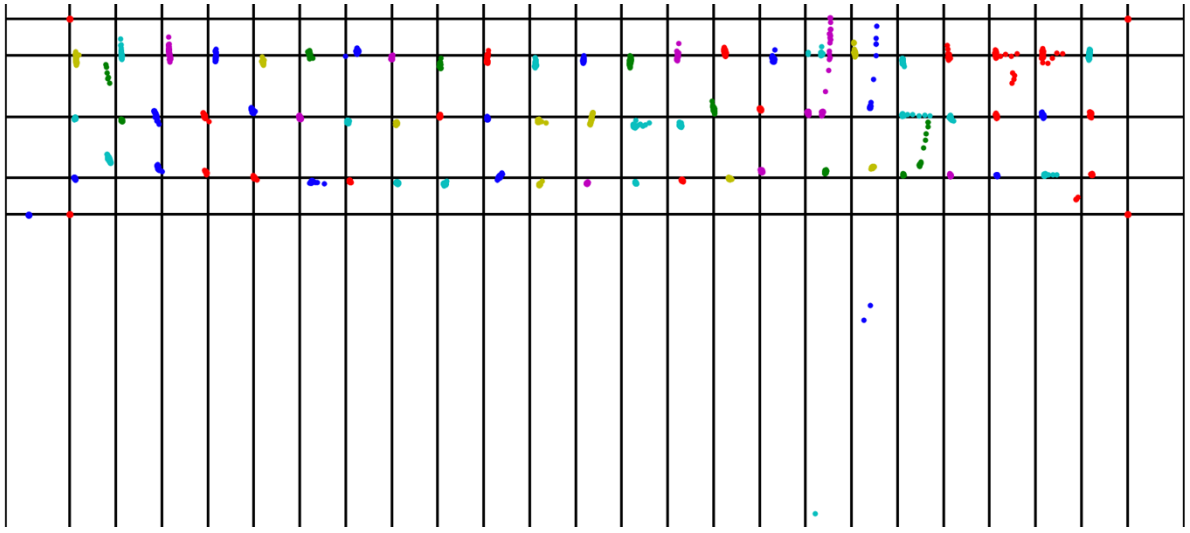

Figura 61. Ejemplo de outlayers del Anexo I.

El código es el siguiente:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import csv
import math

representarOutlayers = True
radioEliminarOutlayers = 0.6 # En metros

def calcularRadio(x2, y2, exactoX, exactoY):
    # print("Los puntos medidos son: ", x2, ', ', y2)
    centroTeorico = [exactoX, exactoY]
    x1, y1 = centroTeorico[0], centroTeorico[1]
    # print("El centro teórico es:", centroTeorico, '[metros]')

    radioDelPuntoMedido = math.sqrt((float(x2) - float(x1))**2 + (float(y2) -
float(y1))**2)
    # print(f"El radio entre los puntos es: {radioDelPuntoMedido}")

    if radioDelPuntoMedido < radioEliminarOutlayers:
        return radioDelPuntoMedido
    else: return None


#### ----------------------------- ####
#### ------- Creación Mapa -------- ####
#### ----------------------------- ####

# Tamaño de cada celda en metros
cell_size = 1
```

```
# Tamaño del mapa en metros
x_len = 23
y_len = 4.26

# Número de celdas en cada dimensión
x_cells = int(x_len / cell_size)
y_cells = int(y_len / cell_size)

# Crear una matriz de ceros
grid = np.zeros((y_cells, x_cells))

# Crear el mapa de cuadrícula usando matplotlib
fig, ax = plt.subplots(figsize=(x_len, y_len))
ax.imshow(grid, cmap='binary')

# Dibujar solo el contorno de la cuadrícula
for edge, spine in ax.spines.items():
    spine.set_visible(True)
    spine.set_linewidth(1)

# Dibujar líneas verticales cada 1 metro, comenzando desde x=0.9
for x in np.arange(0.9, x_cells + 1, 1):
    ax.axvline(x, color='black')

# Dibujar los tres carriles horizontales
ax.axhline(0.01, color='black')
ax.axhline(0.80 + 0.01, color='black')
ax.axhline(2.13 + 0.01, color='black')
ax.axhline(3.46 + 0.01, color='black')
ax.axhline(4.26 + 0.01, color='black')

# Marcar los puntos (0.9,0.01), (0.9,4.26+0.01), (23+0.9,0.01) y (23+0.9,4.26+0.01)
con un punto rojo
points = [(0.9,0.01), (0.9,4.26+0.01), (23+0.9,0.01), (23+0.9,4.26+0.01)]
for point in points:
    ax.plot(point[0], point[1], 'ro', markersize = 3)

ax.plot(0, 0, 'bo', markersize = 3)
# ax.plot(4.866,1.771, 'go', markersize = 2)
# ax.plot(4.872,1.798, 'go', markersize = 2)


# Invertir el eje Y para mover el origen a la esquina inferior izquierda
ax.invert_yaxis()


#### ----------------------------- ####
#### ------- Puntos Centro -------- ####
#### ----------------------------- ####
colores                                                                    =
['mo','ro','bo','co','go','yo','ro','bo','co','yo','co','ro','bo','co','yo','mo','r
o','bo','co','go','mo','ro','bo','co','go']
i = 0

ficherosProcesados = []
for fichero in os.listdir('ctro/procesados/'):
    if fichero.startswith('Processed'):
        ficherosProcesados.append(fichero)

for fichero in ficherosProcesados:
    ruta = 'ctro/procesados/' + fichero
    with open(ruta, 'r') as csvFile:
        lector = csv.reader(csvFile)
        next(lector)
        for linea in lector:
```

```
            coordenadaX, coordenadaY = linea[2], linea[3]
            exactoX, exactoY = linea[5], linea[6]
            if representarOutlayers == False:
                radio = calcularRadio(coordenadaX, coordenadaY, exactoX, exactoY)
                if radio != None:
                    ax.plot(float(coordenadaX),   float(coordenadaY),   colores[i],
markersize = 2)
                else:
                    pass
            else:
                ax.plot(float(coordenadaX),     float(coordenadaY),     colores[i],
markersize = 2)
    i = i + 1


#### ------------------------------ ####
#### ------- Puntos Derecha ------- ####
#### ------------------------------ ####
colores                                                                          =
['mo','ro','bo','co','go','yo','ro','bo','co','yo','co','ro','bo','co','yo','mo','r
o','bo','co','go','mo','ro','bo','co','go']
i = 0

ficherosProcesados = []
for fichero in os.listdir('dcha/procesados/'):
    if fichero.startswith('Processed'):
        ficherosProcesados.append(fichero)

for fichero in ficherosProcesados:
    ruta = 'dcha/procesados/' + fichero
    with open(ruta, 'r') as csvFile:
        lector = csv.reader(csvFile)
        next(lector)
        for linea in lector:
            coordenadaX, coordenadaY = linea[2], linea[3]
            exactoX, exactoY = linea[5], linea[6]
            if representarOutlayers == False:
                radio = calcularRadio(coordenadaX, coordenadaY, exactoX, exactoY)
                if radio != None:
                    ax.plot(float(coordenadaX),   float(coordenadaY),   colores[i],
markersize = 2)
                else:
                    pass
            else:
                ax.plot(float(coordenadaX),     float(coordenadaY),     colores[i],
markersize = 2)
    i = i + 1

#### ------------------------------ ####
#### ------ Puntos Izquierda ------ ####
#### ------------------------------ ####
colores                                                                          =
['mo','ro','bo','co','go','yo','ro','bo','co','yo','co','ro','bo','co','yo','mo','r
o','go','co','go','mo','ro','bo','co','go']
i = 0

ficherosProcesados = []
for fichero in os.listdir('izq/procesados/'):
    if fichero.startswith('Processed'):
        ficherosProcesados.append(fichero)

for fichero in ficherosProcesados:
    ruta = 'izq/procesados/' + fichero
    with open(ruta, 'r') as csvFile:
        lector = csv.reader(csvFile)
        next(lector)
```

```
        for linea in lector:
            coordenadaX, coordenadaY = linea[2], linea[3]
            exactoX, exactoY = linea[5], linea[6]
            if representarOutlayers == False:
                radio = calcularRadio(coordenadaX, coordenadaY, exactoX, exactoY)
                if radio != None:
                    ax.plot(float(coordenadaX),   float(coordenadaY),   colores[i],
markersize = 2)
                else:
                    pass
            else:
                ax.plot(float(coordenadaX),     float(coordenadaY),     colores[i],
markersize = 2)
    i                  =                  i                  +                  1

# Eliminar las marcas de las divisiones en los ejes x e y
ax.set_xticks([])
ax.set_yticks([])

# Ajustar los márgenes de la figura
plt.subplots_adjust(left=0, right=1, bottom=0, top=1)
plt.show()
```

Anexo II

Código desarrollado a través de Matlab para conocer las secciones de corte de los diagramas de recepción de las balizas situadas en el techo del aparcamiento. Para desplazarse y conocer la información de cada uno de los carriles bastaría con cambiar el valor de la variable '`desp`'.

```
h = 4; % Altura del techo
n = 4; % Número de balizas
alpha = 80; % Ángulo de recepción
altura_Twizy = 1.47; % Altura coche
altura_min_recep = 0;
desp = 3.45; % Desplazamiento lateral con respecto a la vertical de los receptores
x0 = [3.54 11.2 21.55 30.3]; % Posiciones en X donde se encuentran las balizas
y0 = [3.82 4.1 4.11 4.05]; % Posiciones en Y donde se encuentran las balizas
alpha_rad = alpha / 180*pi;

figure
hold on;

for i=1:n
x_max=sqrt((h-altura_min_recep)^2/cos(alpha_rad)^2-(y0(i)-desp)^2);
x=-x_max:0.05:x_max;
z=h-cos(alpha_rad)*sqrt(x.^2+(y0(i)-desp)^2);
    if i==1
        plot(x+x0(i),z,'b');
    elseif i==2
        plot(x+x0(i),z,'g');
    elseif i==3
        plot(x+x0(i),z,'r');
    elseif i==4
        plot(x+x0(i),z,'m');
    end
end

axis([0 35 0 h]);
plot ([0 35],altura_Twizy*[1 1],'k');
grid
grid MINOR
axis equal
```

Anexo III

Para poder generar las estadísticas de localización necesitamos acudir a los ficheros generados en el Anexo I referentes a las características básicas de localización y de radio. A partir de dichos datos podríamos obtener estadísticas como la varianza, la desviación estándar, el valor medio o la T de Student de las coordenadas X, las coordenadas Y y los radios obtenidos.

El algoritmo creará un directorio denominado 'estadisticas' donde se almacenarán todas las estadísticas. Se crearán tres ficheros, uno por cada carril. El código es el siguiente:

```
import warnings
warnings.filterwarnings("ignore")
import csv
import numpy as np
import math
import os
from scipy.stats import ttest_1samp
import pandas as pd
from pathlib import Path

# Ruta del directorio a crear
directorioNuevo = 'estadisticas/'
# Crear el directorio usando pathlib
path_directorioNuevo = Path(directorioNuevo)
path_directorioNuevo.mkdir(parents=True)

carriles = ['ctro','dcha', 'izq']

for carril in carriles:

    #### ------------------------------ ####
    #### ------ Ordenar ficheros ------ ####
    #### ------------------------------ ####

    ficherosProcesados = []
    horas = []
    ruta = carril + '/procesados/'
    for fichero in os.listdir(ruta):
        if fichero.startswith('Processed'):
            aux = fichero.split(sep='_')
            horas.append(aux[5])
    horas = sorted(horas)

    for hora in horas:
        for fichero in os.listdir(ruta):
            if hora in fichero:
                ficherosProcesados.append(fichero)


    #### ------------------------------ ####
    #### -------- Estadísticas -------- ####
    #### ------------------------------ ####

    # Trataremos como X1 el valor de X exacto/objetivo y como X2 el valor obtenido.
Equivalente para Y y Radio.
    contador = 0
    for fichero in ficherosProcesados:
        contador = contador + 1
        ruta = carril + '/procesados/'
        ruta = ruta + fichero
        x1, x2 = np.array([]), np.array([])
        y1, y2 = np.array([]), np.array([])
```

```
        radio1, radio2 = np.array([]), np.array([])

        datosExistentes = []

        # print(ruta)
        with open(ruta, 'r') as csvFile:
            lector_csv = csv.reader(csvFile)
            encabezados = next(lector_csv)

            for num, trama in enumerate(lector_csv, start=1):
                datosExistentes.append(trama)
                x1,    x2    =    np.append(x1,    float(trama[5])),    np.append(x2,
float(trama[2]))
                y1,    y2    =    np.append(y1,    float(trama[6])),    np.append(y2,
float(trama[3]))
                radio1,    radio2    =    np.append(radio1,    0),    np.append(radio2,
float(trama[1])) # El radio ideal sería 0

        #### --------- MEDIA de X --------- ####
        mediaX = np.mean(x2)
        # print("La media de los valores de X es: ", mediaX)
        #### --------- VARIANZA y DESVIACIÓN ESTÁNDAR de X --------- ####
        diferenciaX = x1 - x2
        diferenciaXCuadrado = diferenciaX ** 2
        varianzaX = np.mean(diferenciaXCuadrado)
        desviacionX = math.sqrt(varianzaX)
        # print("La varianza de las X es: ", varianzaX, " y su Desviación estándar:
", desviacionX)
        #### --------- T de STUDENT de X --------- ####
        TdeStudentX, probabilidadX = ttest_1samp(diferenciaX, 0)
        # print("El valor de T de Student es: ", TdeStudentX, " y la probabilidad de
que los puntos medidos y reales sean iguales: ", probabilidadX)


        #### --------- MEDIA de Y --------- ####
        mediaY = np.mean(y2)
        # print("La media de los valores de Y es: ", mediaY)
        #### --------- VARIANZA y DESVIACIÓN ESTÁNDAR de Y --------- ####
        diferenciaY = y1 - y2
        diferenciaYCuadrado = diferenciaY ** 2
        varianzaY = np.mean(diferenciaYCuadrado)
        desviacionY = math.sqrt(varianzaY)
        # print("La varianza de las Y es: ", varianzaY, " y su Desviación estándar:
", desviacionY)
        #### --------- T de STUDENT de Y --------- ####
        TdeStudentY, probabilidadY = ttest_1samp(diferenciaY, 0)
        # print("El valor de T de Student es: ", TdeStudentY, " y la probabilidad de
que los puntos medidos y reales sean iguales: ", probabilidadY)


        #### --------- MEDIA del Radio --------- ####
        mediaRadio = np.mean(radio2)
        # print("La media de los valores de Y es: ", mediaRadio)
        #### --------- VARIANZA y DESVIACIÓN ESTÁNDAR de Y --------- ####
        diferenciaRadio = radio1 - radio2
        diferenciaRadioCuadrado = diferenciaRadio ** 2
        varianzaRadio = np.mean(diferenciaRadioCuadrado)
        desviacionRadio = math.sqrt(varianzaRadio)
        #  print("La  varianza  de  las  Y  es:  ",  varianzaRadio,  "  y  su  Desviación
estándar: ", desviacionRadio)
        #### --------- T de STUDENT del Radio --------- ####
        TdeStudentRadio, probabilidadRadio = ttest_1samp(diferenciaRadio, 0)
        # print("El valor de T de Student es: ", TdeStudentY, " y la probabilidad de
que los puntos medidos y reales sean iguales: ", probabilidadY)
```

```
        encabezados    =    ['PuntoMetro',    'VarianzaX',    'DesviacionEstandarX',
'ValorMedioX',  'TstudentX',  'ProbabilidadX',  'VarianzaY',  'DesviacionEstandarY',
'ValorMedioY',
                    'TstudentY',          'ProbabilidadY',          'VarianzaRadio',
'DesviacionEstandarRadio', 'ValorMedioRadio', 'TstudentRadio', 'ProbabilidadRadio',
'Fichero']

        df = pd.DataFrame(columns=encabezados)

        info = [contador, varianzaX, desviacionX, mediaX, TdeStudentX, probabilidadX,
                varianzaY, desviacionY, mediaY, TdeStudentY, probabilidadY,
                varianzaRadio,    desviacionRadio,    mediaRadio,    TdeStudentRadio,
probabilidadRadio, fichero]

        df    =    pd.concat([df,    pd.DataFrame([info],    columns=encabezados)],
ignore_index=True)

        if carril == 'ctro':
            ficheroDestino = 'estadisticas/estadisticasCtro.csv'
        elif carril == 'dcha':
            ficheroDestino = 'estadisticas/estadisticasDcha.csv'
        elif carril == 'izq':
            ficheroDestino = 'estadisticas/estadisticasIzq.csv'

        # Comprueba si el archivo ya existe
        if os.path.exists(ficheroDestino):
            # Si el archivo ya existe, solo añade nuevas filas sin encabezados
            df.to_csv(ficheroDestino, mode='a', header=False, index=False)
        else:
            # Si el archivo no existe, añade nuevas filas con encabezados
            df.to_csv(ficheroDestino, mode='a', header=True, index=False)
```

Tras ejecutar este código se nos habrá generado un fichero con las estadísticas para cada carril y posición en dicho carril. Sin embargo, para analizarlo de manera más general y obtener los datos que nosotros hemos utilizado para analizar los resultados, haremos uso de 4 ficheros. Estos ficheros serán los encargados de extraer la varianza, desviación media y T de Student para cada carril de manera general. A modo de ejemplo se muestra el fichero dedicado a obtener la varianza y desviación media para las variables X e Y, y su representación sobre el mapa, su nombre es `'XY_mapa_Desviacion_PuntoMedio.py'`. El resto los encontramos en el interior del directorio y funcionan de manera análoga. Su contenido es el siguiente:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import csv

#### ------------------------------ ####
#### ------- Creación Mapa -------- ####
#### ------------------------------ ####

# Tamaño de cada celda en metros
cell_size = 1

# Tamaño del mapa en metros
x_len = 23
y_len = 4.26

# Número de celdas en cada dimensión
x_cells = int(x_len / cell_size)
y_cells = int(y_len / cell_size)

# Crear una matriz de ceros
```

```
grid = np.zeros((y_cells, x_cells))

# Crear el mapa de cuadrícula usando matplotlib
fig, ax = plt.subplots(figsize=(x_len, y_len))
ax.imshow(grid, cmap='binary')

# Dibujar solo el contorno de la cuadrícula
for edge, spine in ax.spines.items():
    spine.set_visible(True)
    spine.set_linewidth(1)

# Dibujar líneas verticales cada 1 metro, comenzando desde x=0.9
for x in np.arange(0.9, x_cells + 1, 1):
    ax.axvline(x, color='black')

# Dibujar los tres carriles horizontales
ax.axhline(0.01, color='black')
ax.axhline(0.80 + 0.01, color='black')
ax.axhline(2.13 + 0.01, color='black')
ax.axhline(3.46 + 0.01, color='black')
ax.axhline(4.26 + 0.01, color='black')

# Marcar los puntos (0.9,0.01), (0.9,4.26+0.01), (23+0.9,0.01) y (23+0.9,4.26+0.01)
con un punto rojo
points = [(0.9,0.01), (0.9,4.26+0.01), (23+0.9,0.01), (23+0.9,4.26+0.01)]
for point in points:
    ax.plot(point[0], point[1], 'ro', markersize = 3)

ax.plot(0, 0, 'bo', markersize = 3)
# ax.plot(4.866,1.771, 'go', markersize = 2)
# ax.plot(4.872,1.798, 'go', markersize = 2)


# Invertir el eje Y para mover el origen a la esquina inferior izquierda
ax.invert_yaxis()


umbral_t_x = 0.05
umbral_t_y = 0.05

#### ----------------------------- ####
#### ------- Puntos Centro -------- ####
#### ----------------------------- ####

# Abrir el archivo CSV
with open('estadisticasCtro.csv', 'r') as csvFile:
    lector = csv.reader(csvFile)
    next(lector)  # Saltar la primera fila de encabezados
    for linea in lector:
        # Leer los valores del CSV
        desviacionX,    desviacionY,    coordenadaXMedia,    coordenadaYMedia    =
float(linea[2]), float(linea[7]), float(linea[3]), float(linea[8])
        tStudentX,  ProbTStudentX,  tStudentY,  ProbTStudentY  =  float(linea[4]),
float(linea[5]), float(linea[9]), float(linea[10])

        # Marcar el punto medio en el mapa
        ax.plot(coordenadaXMedia, coordenadaYMedia, 'ro', markersize=4)

        # Calcular los puntos extremos de la línea perpendicular al eje X
        punto1_x = coordenadaXMedia - desviacionX / 2
        punto2_x = coordenadaXMedia + desviacionX / 2

        # Dibujar la línea perpendicular al eje X
        ax.plot([punto1_x, punto2_x], [coordenadaYMedia, coordenadaYMedia], 'b-')

        # Calcular los puntos extremos de la recta perpendicular al eje Y
```

```
        punto1_y = coordenadaYMedia - desviacionY / 2
        punto2_y = coordenadaYMedia + desviacionY / 2

        # Dibujar la línea perpendicular al eje Y
        ax.plot([coordenadaXMedia, coordenadaXMedia], [punto1_y, punto2_y], 'g-')


# #### ------------------------------ ####
# #### ------- Puntos Derecha ------- ####
# #### ------------------------------ ####

with open('estadisticasDcha.csv', 'r') as csvFile:
    lector = csv.reader(csvFile)
    next(lector)  # Saltar la primera fila de encabezados
    for linea in lector:
        # Leer los valores del CSV
        desviacionX,    desviacionY,    coordenadaXMedia,    coordenadaYMedia    =
float(linea[2]), float(linea[7]), float(linea[3]), float(linea[8])
        tStudentX,  ProbTStudentX,  tStudentY,  ProbTStudentY  =  float(linea[4]),
float(linea[5]), float(linea[9]), float(linea[10])

        # Marcar el punto medio en el mapa
        ax.plot(coordenadaXMedia, coordenadaYMedia, 'ro', markersize=4)

        # Calcular los puntos extremos de la línea perpendicular al eje X
        punto1_x = coordenadaXMedia - desviacionX / 2
        punto2_x = coordenadaXMedia + desviacionX / 2

        # Dibujar la línea perpendicular al eje X
        ax.plot([punto1_x, punto2_x], [coordenadaYMedia, coordenadaYMedia], 'b-')

        # Calcular los puntos extremos de la recta perpendicular al eje Y
        punto1_y = coordenadaYMedia - desviacionY / 2
        punto2_y = coordenadaYMedia + desviacionY / 2

        # Dibujar la línea perpendicular al eje Y
        ax.plot([coordenadaXMedia, coordenadaXMedia], [punto1_y, punto2_y], 'g-')

# #### ------------------------------ ####
# #### ------ Puntos Izquierda ------ ####
# #### ------------------------------ ####

with open('estadisticasIzq.csv', 'r') as csvFile:
    lector = csv.reader(csvFile)
    next(lector)  # Saltar la primera fila de encabezados
    for linea in lector:
        # Leer los valores del CSV
        desviacionX,    desviacionY,    coordenadaXMedia,    coordenadaYMedia    =
float(linea[2]), float(linea[7]), float(linea[3]), float(linea[8])
        tStudentX,  ProbTStudentX,  tStudentY,  ProbTStudentY  =  float(linea[4]),
float(linea[5]), float(linea[9]), float(linea[10])

        # Marcar el punto medio en el mapa
        ax.plot(coordenadaXMedia, coordenadaYMedia, 'ro', markersize=4)

        # Calcular los puntos extremos de la línea perpendicular al eje X
        punto1_x = coordenadaXMedia - desviacionX / 2
        punto2_x = coordenadaXMedia + desviacionX / 2

        # Dibujar la línea perpendicular al eje X
        ax.plot([punto1_x, punto2_x], [coordenadaYMedia, coordenadaYMedia], 'b-')

        # Calcular los puntos extremos de la recta perpendicular al eje Y
        punto1_y = coordenadaYMedia - desviacionY / 2
        punto2_y = coordenadaYMedia + desviacionY / 2
```

```
        # Dibujar la línea perpendicular al eje Y
        ax.plot([coordenadaXMedia, coordenadaXMedia], [punto1_y, punto2_y], 'g-')



# Eliminar las marcas de las divisiones en los ejes x e y
ax.set_xticks([])
ax.set_yticks([])

# Ajustar los márgenes de la figura
plt.subplots_adjust(left=0, right=1, bottom=0, top=1)
plt.show()
```

Anexo IV

Para poder leer la información de la IMU a través del puerto USB deberemos haber preparado el puerto USB como ya se ha explicado en el informe. Una vez hecho esto haremos uso del siguiente script denominado '`lecturaUSB.py`'. Para hacer uso de él debemos tener en cuenta cuál es el puerto USB que nos ha asignado el sistema y establecerlo en la variable `tty` que servirá para crear el hilo referente a la recepción de información de la baliza. Se hará a su vez uso de la librería de Marvelmind que se encuentra en el Anexo VIII.

```
from marvelmindLib import MarvelmindHedge
from time import sleep
import sys

def main():
    hedge = MarvelmindHedge(tty = "/dev/ttyACM1", adr=None, debug=False) # create
MarvelmindHedge thread

    if (len(sys.argv)>1):
        hedge.tty= sys.argv[1]

    hedge.start() # start thread
    while True:
        try:
            hedge.dataEvent.wait(1)
            hedge.dataEvent.clear()

            if (hedge.positionUpdated):
                hedge.print_position()

            if (hedge.distancesUpdated):
                hedge.print_distances()

            if (hedge.rawImuUpdated):
                hedge.print_raw_imu()

            if (hedge.fusionImuUpdated):
                hedge.print_imu_fusion()

            if (hedge.telemetryUpdated):
                hedge.print_telemetry()

            if (hedge.qualityUpdated):
                hedge.print_quality()

            if (hedge.waypointsUpdated):
                hedge.print_waypoint()

            if (hedge.userDataUpdated):
                hedge.print_user_data()
        except KeyboardInterrupt:
            hedge.stop()  # stop and close serial port
            sys.exit()
main()
```

Anexo V

El procesado en tiempo real requerirá leer los datos entrantes de las balizas a través de dos puertos USB, que deberemos manipular para establecer los dos puertos asignados por el sistema a través de las variables `tty`. Este algoritmo implementa un buffer para la correcta recepción de las tramas en tiempo y sincronización, siendo capaz de adaptarse además ante posibles pérdidas de tramas. El nombre del fichero es '`procesamientoTiempoRealConBuffer.py`' y hará uso de la librería de Marvelmind del Anexo VIII.

```
from marvelmindLib import MarvelmindHedge
from time import sleep
from math import sqrt, atan, pi, cos, sin, atan2, degrees
import sys
import collections

def recta(x1, y1, x2, y2):
    """
    Calcula la ecuación de una recta a partir de dos puntos.

    Args:
        x1: La coordenada X del primer punto.
        y1: La coordenada Y del primer punto.
        x2: La coordenada X del segundo punto.
        y2: La coordenada Y del segundo punto.

    Returns:
        La ecuación de la recta.
    """
    # print(x1,y1,x2,y2)

    if x1 == x2:
        if y1 < y2:
            m = 1
        else:
            m = -1
    else:
        m = (y2 - y1) / (x2 - x1)

    b = y1 - m * x1

    return m, b

def distanciaPuntoRecta(x, y, m, b):
    """
    Calcula la distancia desde un punto a una recta.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        recta: La ecuación de la recta.

    Returns:
        La distancia desde el punto a la recta.
    """
    x = float(x); y = float(y)
    return abs((m * x + b - y) / sqrt(m**2 + 1))

def segmentoMasCercano(x, y, trayectoria):
    """
    Encuentra el segmento de la trayectoria que está más cerca de un punto dado.
```

```
    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        trayectoria: La trayectoria a analizar.

    Returns:
        El índice del segmento más cercano, la distancia más corta al segmento más
cercano y el ángulo del segmento más cercano.
    """

    distancias = []

    # El coche está en:
    # plt.plot([x], [y], "go")

    for i in range(len(trayectoria) - 1):
        m, b = recta(trayectoria[i][0], trayectoria[i][1], trayectoria[i+1][0],
trayectoria[i+1][1])
        distancia = distanciaPuntoRecta(x, y, m, b)
        distancias.append((i, distancia, atan(m) * 180 / pi))

    indiceMasCercano, distanciaMasCercano, anguloTrayecoria = min(distancias,
key=lambda x: x[1])

    return indiceMasCercano, distanciaMasCercano, anguloTrayecoria

def calcularYaw(posicionX1, posicionY1, posicionX2, posicionY2):
    """ La baliza X1,Y1 corresponde a la baliza situada en la derecha
        y la baliza X2,Y2 a la baliza situada a la izquierda """
    # Calcula la diferencia en las coordenadas x y y entre las balizas móviles
    diffX = float(posicionX2) - float(posicionX1)
    diffY = posicionY2 - posicionY1

    # Calcula el ángulo de yaw en radianes usando atan2
    yaw_radianes = atan2(diffY, diffX)

    # Convierte el ángulo de radianes a grados
    yaw_grados = degrees(yaw_radianes) - 90

    # print("El ángulo de yaw es:", yaw_grados, "grados")

    return yaw_grados

def anguloRespectoTrayectoria(buffer1, buffer2):

    xBaliza1, yBaliza1, xBaliza2, yBaliza2 = buffer1[1], buffer1[2], buffer2[1],
buffer2[2]

    xCoche = (xBaliza1 + xBaliza2) / 2
    yCoche = (yBaliza1 + yBaliza2) / 2

    indice, distancia, anguloTrayecoria = segmentoMasCercano(xCoche, yCoche,
trayectoria)
    anguloCoche = calcularYaw(xBaliza1, yBaliza1, xBaliza2, yBaliza2)
    print("El segmento más cercano es el ", indice, "y está a", distancia,"metros
del coche, ángulo del segmento (trayectoria):", anguloTrayecoria)
    print("\nÁngulo coche: ", anguloCoche)
    print("")

    with open("localizacionCoche.txt","a") as file:
        aEscribirCoche = str(xCoche) + ";" + str(yCoche) + ";" + str(anguloCoche) +
"\n"
        file.write(aEscribirCoche)

    return anguloCoche
```

```
def procesar_trayectoria(cadena):
    # Eliminar paréntesis y dividir la cadena en cada coma
    valores = cadena.replace('(', '').replace(')', '').split(',')
    # Convertir cada par de valores en una tupla de flotantes
    trayectoria = [(float(valores[i]), float(valores[i+1])) for i in range(0,
len(valores), 2)]
    return trayectoria


with open("trayectoria.txt", "r") as file:
    trayectoria = file.readline()
trayectoria = procesar_trayectoria(trayectoria)

def main():
    hedge = MarvelmindHedge(tty = "/dev/ttyACM1", adr=None, debug=False) # create
MarvelmindHedge thread
    hedge2 = MarvelmindHedge(tty = "/dev/ttyACM2", adr=None, debug=False) # create
MarvelmindHedge thread

    if (len(sys.argv)>1):
        hedge.tty= sys.argv[1]
        hedge2.tty= sys.argv[1]

    hedge.start() # start thread
    hedge2.start() # start thread

    posicionBaliza1 = None
    posicionBaliza2 = None

    bufferBaliza1 = collections.deque()
    bufferBaliza2 = collections.deque()

    valido = False

    timestampAnterior1 = 0
    timestampAnterior2 = 0

    tasaMilisegundos = 250
    angulo = None

    while True:
        try:
            hedge.dataEvent.wait(1)
            hedge.dataEvent.clear()
            hedge2.dataEvent.wait(1)
            hedge2.dataEvent.clear()

            if (hedge.positionUpdated):
                posicionBaliza1 = hedge.position()
                nBaliza, xBaliza1, yBaliza1, timeStamp1 = float(posicionBaliza1[0]),
float(posicionBaliza1[1]), float(posicionBaliza1[2]), float(posicionBaliza1[5])
                datosBuffer = [nBaliza, xBaliza1, yBaliza1, timeStamp1]
                bufferBaliza1.append(datosBuffer)

                #  print(posicionBaliza1[0],"Time:", timeStamp1, "Dif-anterior:",
timeStamp1 - timestampAnterior1)
                # timestampAnterior1 = timeStamp1

                with open("localizacionBalizas.txt","a") as file:
                    aEscribir1 = str(nBaliza) + "," + str(xBaliza1) + "," +
str(yBaliza1) + "," + str(timeStamp1) + "\n"
                    file.write(aEscribir1)

            if (hedge2.positionUpdated):
                posicionBaliza2 = hedge2.position()
```

```
                nBaliza2,       xBaliza2,       yBaliza2,       timeStamp2       =
float(posicionBaliza2[0]),   float(posicionBaliza2[1]),   float(posicionBaliza2[2]),
float(posicionBaliza2[5])
                datosBuffer = [nBaliza2, xBaliza2, yBaliza2, timeStamp2]
                bufferBaliza2.append(datosBuffer)

                #  print(posicionBaliza2[0],"Time:",  timeStamp2,  "Dif-anterior:",
timeStamp2 - timestampAnterior2)
                # timestampAnterior2 = timeStamp2

                with open("localizacionBalizas.txt","a") as file:
                    aEscribir2  =  str(nBaliza2)  +  ","  +  str(xBaliza2)  +  ","  +
str(yBaliza2) + "," + str(timeStamp2) + "\n"
                    file.write(aEscribir2)


            if len(bufferBaliza1) > 0 and len(bufferBaliza2) > 0:

                print(bufferBaliza1)
                print(bufferBaliza2)

                if len(bufferBaliza1) == len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[-1]
                    lecturaBuffer2 = bufferBaliza2[-1]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True

                        angulo     =     anguloRespectoTrayectoria(lecturaBuffer1,
lecturaBuffer2)
                        bufferBaliza1.clear()
                        bufferBaliza2.clear()

                elif len(bufferBaliza1) > len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[0]
                    lecturaBuffer2 = bufferBaliza2[-1]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True

                        angulo     =     anguloRespectoTrayectoria(lecturaBuffer1,
lecturaBuffer2)
                        bufferBaliza1.popleft()
                        bufferBaliza2.pop()
                    if not valido:
                        lecturaBuffer1 = bufferBaliza1[-1]
                        lecturaBuffer2 = bufferBaliza2[-1]
                        if    abs(lecturaBuffer1[3]    -    lecturaBuffer2[3])    <
tasaMilisegundos:
                            #     print("Restaré",     lecturaBuffer1[0],     "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                            valido = True

                            angulo    =    anguloRespectoTrayectoria(lecturaBuffer1,
lecturaBuffer2)
                            bufferBaliza1.clear()
                            bufferBaliza2.clear()
```

```
                elif len(bufferBaliza1) < len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[-1]
                    lecturaBuffer2 = bufferBaliza2[0]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True
                        angulo      =      anguloRespectoTrayectoria(lecturaBuffer1,
lecturaBuffer2)
                        bufferBaliza1.pop()
                        bufferBaliza2.popleft()
                    if not valido:
                        lecturaBuffer1 = bufferBaliza1[-1]
                        lecturaBuffer2 = bufferBaliza2[-1]
                        if    abs(lecturaBuffer1[3]    -    lecturaBuffer2[3])    <
tasaMilisegundos:
                            #     print("Restaré",      lecturaBuffer1[0],      "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                            valido = True
                            angulo    =    anguloRespectoTrayectoria(lecturaBuffer1,
lecturaBuffer2)
                            bufferBaliza1.clear()
                            bufferBaliza2.clear()

                valido = False



        except KeyboardInterrupt:
            hedge.stop()  # stop and close serial port
            hedge2.stop()  # stop and close serial port
            sys.exit()
main()
```

Anexo VI

La creación del mapa a posteriori se hará utilizando la localización del coche extraída durante el ejercicio de procesamiento en tiempo real del Anexo V. A través de la información que se encuentra en el fichero '`localizacionCoche.txt`' creará un mapa indicando la posición y la trayectoria de cada trama, su contenido será del estilo de:

```
5.395;0.735;-2.495573283384303
5.3955;0.7345;-2.5555342134604047
5.3955;0.734;-2.558616508318707
5.3955;0.735;-2.5648034428412387
5.395;0.735;-2.495573283384303
5.3955;0.7354999999999999;-2.42927332349997 …
```

La trayectoria a seguir por el vehículo está definida en el fichero '`trayectoria.txt`', donde se define cada punto que delimita los segmentos de la siguiente forma:

```
(5.4,    0.79),(10.4,    0.79),(15.4,    0.79),(20.4,    0.79),
(25.4,0.79),(27.9, 0.79),(28.4, 1.45),(28.4, 2.79),
(27.9, 3.46),(25.4, 3.46),(20.4, 3.46),(15.4, 3.46),
(10.4, 3.46),(5.4, 3.46)
```

La posición en la que se encuentran las balizas que hemos utilizado con respecto al (0,0) que hemos definido las debemos indicar en el fichero '`posicionBalizas.txt`':

```
(3.54, 3.82),(11.20, 4.1),(21.55, 4.11),(30.30, 4.05)
```

Asimismo, también definiremos la cuadrícula definida para realizar las pruebas en un fichero aparte, denominado '`cuadricula.txt`':

```
5.4, 0, 5.4, 4.25
28.4, 0, 28.4, 4.25
5.4, 0.79, 28.4, 0.79
5.4, 2.125, 28.4, 2.125
5.4, 3.46, 28.4, 3.46
```

Finalmente, el código encargado den ensamblar todo lo mencionado y crear el mapa será el denominado '`creacionMapaConTrayectoria.py`'. Su contenido es el siguiente:

```
import matplotlib.pyplot as plt
from math import sqrt, atan, pi, cos, sin, atan2, degrees
import numpy as np

def contorno(rectangulo):
    """
    Representa un rectángulo en 2D con matplotlib.

    Args:
        rectangulo: El rectángulo a representar.
    """

    plt.plot([punto[0] for punto in rectangulo], [punto[1] for punto in rectangulo],
linewidth=2)
```

```
    plt.axis([0, 34.8, 0, 4.25])
    plt.xticks([0, 10, 20, 30, 34.8])
    plt.yticks([0, 1, 2, 3, 4.25])
    plt.xlabel("Longitud (m)")
    plt.ylabel("Ancho (m)")

def situarBalizas(puntos):
    """
    Dibuja 4 puntos en un plano.

    Args:
        puntos: Los puntos a dibujar.
    """

    for punto in puntos:
        plt.plot([punto[0]], [punto[1]], "ro")

def trazarTrayectoria(puntos):
    """
    Traza una trayectoria a partir de una lista de puntos.

    Args:
        puntos: Una lista de puntos con coordenadas (x, y).
    """

    # Representamos los puntos.

    for punto in puntos:
        plt.plot([punto[0]], [punto[1]], "bo")

    # Trazamos la trayectoria.

    for i in range(len(puntos) - 1):
        plt.plot([puntos[i][0], puntos[i + 1][0]], [puntos[i][1], puntos[i + 1][1]],
"r")

def recta(x1, y1, x2, y2):
    """
    Calcula la ecuación de una recta a partir de dos puntos.

    Args:
        x1: La coordenada X del primer punto.
        y1: La coordenada Y del primer punto.
        x2: La coordenada X del segundo punto.
        y2: La coordenada Y del segundo punto.

    Returns:
        La ecuación de la recta.
    """
    # print(x1,y1,x2,y2)

    if x1 == x2:
        m = 1
    else:
        m = (y2 - y1) / (x2 - x1)
    b = y1 - m * x1

    return m, b

def distanciaPuntoRecta(x, y, m, b):
    """
    Calcula la distancia desde un punto a una recta.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
```

```
        recta: La ecuación de la recta.

    Returns:
        La distancia desde el punto a la recta.
    """
    x = float(x); y = float(y)
    return abs((m * x + b - y) / sqrt(m**2 + 1))

def segmentoMasCercano(x, y, trayectoria):
    """
    Encuentra el segmento de la trayectoria que está más cerca de un punto dado.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        trayectoria: La trayectoria a analizar.

    Returns:
        El índice del segmento más cercano, la distancia más corta al segmento más
cercano y el ángulo del segmento más cercano.
    """

    distancias = []

    # El coche está en:
    plt.plot([x], [y], "go")

    for i in range(len(trayectoria) - 1):
        m, b = recta(trayectoria[i][0], trayectoria[i][1], trayectoria[i+1][0],
trayectoria[i+1][1])
        distancia = distanciaPuntoRecta(x, y, m, b)
        distancias.append((i, distancia, atan(m) * 180 / pi))

    indiceMasCercano, distanciaMasCercano, anguloTrayecoria = min(distancias,
key=lambda x: x[1])

    return indiceMasCercano, distanciaMasCercano, anguloTrayecoria

def direccionCoche(x, y, longitud, anchura, anguloCoche):
    """
    Dibuja una flecha en un punto del plano.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        longitud: La longitud de la flecha.
        anchura: La anchura de la flecha.
        color: El color de la flecha.
    """

    # Convertir el ángulo a radianes
    anguloRadianes = np.radians(anguloCoche)

    # Calcular las coordenadas del punto extremo de la flecha
    x2 = x + longitud * cos(anguloRadianes)
    y2 = y + longitud * sin(anguloRadianes)

    # Dibujamos la flecha.

    plt.plot([x, x2], [y, y2], "green", linewidth=1)
    plt.arrow(x, y, x2 - x, y2 - y, width=anchura, head_width=anchura * 1,
head_length=longitud * 0.1, fc="green", ec="green")

def calcularYaw(posicionX1, posicionY1, posicionX2, posicionY2):

    # Calcula la diferencia en las coordenadas x y y entre las balizas móviles
```

```
    diffX = float(posicionX2) - float(posicionX1)
    diffY = posicionY2 - posicionY1

    # Calcula el ángulo de yaw en radianes usando atan2
    yaw_radianes = atan2(diffY, diffX)

    # Convierte el ángulo de radianes a grados
    yaw_grados = degrees(yaw_radianes) - 90

    # print("El ángulo de yaw es:", yaw_grados, "grados")

    return yaw_grados

def procesarCadena(cadena):
    # Eliminar paréntesis y dividir la cadena en cada coma
    valores = cadena.replace('(', '').replace(')', '').split(',')
    # Convertir cada par de valores en una tupla de flotantes
    trayectoria = [(float(valores[i]), float(valores[i+1])) for i in range(0,
len(valores), 2)]
    return trayectoria

def cuadricula(x1, y1, x2, y2):
    """
    Dibuja una línea en un plano.

    Args:
        x1: La coordenada X del primer punto de la línea.
        y1: La coordenada Y del primer punto de la línea.
        x2: La coordenada X del segundo punto de la línea.
        y2: La coordenada Y del segundo punto de la línea.
    """

    plt.plot([float(x1), float(x2)], [float(y1), float(y2)], linewidth=1,
color="black")

# Crear axis proporcional
fig, ax = plt.subplots()
ax.set_aspect('equal')

# Dibujar cuadrícula suelo
with open ("cuadricula.txt", "r") as file:
    lineas = file.readlines()
    for linea in lineas:
        valores = linea.strip().split(", ")  # Dividir la línea en valores separados
por comas
        cuadricula(*valores)

# Contorno del mapa
rectangulo = ([0, 0], [34.8, 0], [34.8, 4.25], [0, 4.25], [0, 0])
contorno(rectangulo)

with open("posicionBalizas.txt", "r") as file:
    posicionBalizas = file.readline()
posicionBalizas = procesarCadena(posicionBalizas)
situarBalizas(posicionBalizas)

with open("trayectoria.txt", "r") as file:
    trayectoria = file.readline()
trayectoria = procesarCadena(trayectoria)
trazarTrayectoria(trayectoria)

longitudFlecha = 1
anchuraFlecha = 0.002

# Posición coche
with open ("localizacionCoche.txt", "r") as file:
```

```
    lineas = file.readlines()

for linea in lineas:
    linea = linea.split(";")

    xCoche, yCoche, anguloCoche = float(linea[0]), float(linea[1]), float(linea[2])
    direccionCoche(xCoche, yCoche, longitudFlecha, anchuraFlecha, anguloCoche)

    indice, distancia, anguloTrayecoria = segmentoMasCercano(xCoche, yCoche,
trayectoria)

    # print("El segmento más cercano es el", indice, "y está a", distancia, " metros
del coche")
    # print("El ángulo del segmento (trayectoria) es: ", anguloTrayecoria)
    # print("Ángulo coche: ", anguloCoche, "con posición: ",xCoche,",",yCoche)

plt.show()
```

Anexo VII

A través de una manipulación del código del Anexo VI incluiremos la creación del servidor, así como el envío de la información necesaria. El nombre del fichero desarrollado es '`envioInfoSocketTiempoReal.py`' y su código es el siguiente:

```
from marvelmindLib import MarvelmindHedge
from time import sleep
from math import sqrt, atan, pi, cos, sin, atan2, degrees
import sys
import collections
import socket

def recta(x1, y1, x2, y2):
    """
    Calcula la ecuación de una recta a partir de dos puntos.

    Args:
        x1: La coordenada X del primer punto.
        y1: La coordenada Y del primer punto.
        x2: La coordenada X del segundo punto.
        y2: La coordenada Y del segundo punto.

    Returns:
        La ecuación de la recta.
    """
    # print(x1,y1,x2,y2)

    if x1 == x2:
        if y1 < y2:
            m = 1
        else:
            m = -1
    else:
        m = (y2 - y1) / (x2 - x1)

    b = y1 - m * x1

    return m, b

def distanciaPuntoRecta(x, y, m, b):
    """
    Calcula la distancia desde un punto a una recta.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        recta: La ecuación de la recta.

    Returns:
        La distancia desde el punto a la recta.
    """
    x = float(x); y = float(y)
    return abs((m * x + b - y) / sqrt(m**2 + 1))

def segmentoMasCercano(x, y, trayectoria):
    """
    Encuentra el segmento de la trayectoria que está más cerca de un punto dado.

    Args:
        x: La coordenada X del punto.
        y: La coordenada Y del punto.
        trayectoria: La trayectoria a analizar.
```

```
    Returns:
        El índice del segmento más cercano, la distancia más corta al segmento más
cercano y el ángulo del segmento más cercano.
    """

    distancias = []

    # El coche está en:
    # plt.plot([x], [y], "go")

    for i in range(len(trayectoria) - 1):
        m,  b  =  recta(trayectoria[i][0],  trayectoria[i][1],  trayectoria[i+1][0],
trayectoria[i+1][1])
        distancia = distanciaPuntoRecta(x, y, m, b)
        distancias.append((i, distancia, atan(m) * 180 / pi))

    indiceMasCercano,  distanciaMasCercano,  anguloTrayecoria  =  min(distancias,
key=lambda x: x[1])

    return indiceMasCercano, distanciaMasCercano, anguloTrayecoria

def calcularYaw(posicionX1, posicionY1, posicionX2, posicionY2):
    """ La baliza X1,Y1 corresponde a la baliza situada en la derecha
        y la baliza X2,Y2 a la baliza situada a la izquierda """
    # Calcula la diferencia en las coordenadas x y y entre las balizas móviles
    diffX = float(posicionX2) - float(posicionX1)
    diffY = posicionY2 - posicionY1

    # Calcula el ángulo de yaw en radianes usando atan2
    yaw_radianes = atan2(diffY, diffX)

    # Convierte el ángulo de radianes a grados
    yaw_grados = degrees(yaw_radianes) - 90

    # print("El ángulo de yaw es:", yaw_grados, "grados")

    return yaw_grados

def anguloRespectoTrayectoria(buffer1, buffer2):

    xBaliza1, yBaliza1, xBaliza2, yBaliza2 = buffer1[1], buffer1[2], buffer2[1],
buffer2[2]

    xCoche = (xBaliza1 + xBaliza2) / 2
    yCoche = (yBaliza1 + yBaliza2) / 2

    indice,  distancia,  anguloTrayecoria  =  segmentoMasCercano(xCoche,  yCoche,
trayectoria)
    anguloCoche = calcularYaw(xBaliza1, yBaliza1, xBaliza2, yBaliza2)
    # print("El segmento más cercano es el ", indice,  "y está a", distancia,"metros
del coche, ángulo del segmento (trayectoria):", anguloTrayecoria)
    # print("\nÁngulo coche: ", anguloCoche)
    # print("")

    with open("localizacionCoche.txt","a") as file:
        aEscribirCoche = str(xCoche) + ";" + str(yCoche) + ";" + str(anguloCoche) +
"\n"
        file.write(aEscribirCoche)

    return anguloCoche, distancia

def procesar_trayectoria(cadena):
    # Eliminar paréntesis y dividir la cadena en cada coma
    valores = cadena.replace('(', '').replace(')', '').split(',')
    # Convertir cada par de valores en una tupla de flotantes
```

```
    trayectoria = [(float(valores[i]), float(valores[i+1])) for i in range(0,
len(valores), 2)]
    return trayectoria


with open("trayectoria.txt", "r") as file:
    trayectoria = file.readline()
trayectoria = procesar_trayectoria(trayectoria)

def main():
    hedge = MarvelmindHedge(tty = "/dev/ttyACM1", adr=None, debug=False) # create
MarvelmindHedge thread
    hedge2 = MarvelmindHedge(tty = "/dev/ttyACM2", adr=None, debug=False) # create
MarvelmindHedge thread

    if (len(sys.argv)>1):
        hedge.tty= sys.argv[1]
        hedge2.tty= sys.argv[1]

    hedge.start() # start thread
    hedge2.start() # start thread

    posicionBaliza1 = None
    posicionBaliza2 = None

    bufferBaliza1 = collections.deque()
    bufferBaliza2 = collections.deque()

    valido = False

    timestampAnterior1 = 0
    timestampAnterior2 = 0

    tasaMilisegundos = 250
    anguloRespectoTrayec = None
    distanciaATrayec = None

    # Definir la dirección IP y el puerto en el que el servidor estará escuchando
    host = '192.168.0.3'  # Dirección IP del servidor
    puerto = 2048


    # Crear un objeto de socket para el servidor
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as servidor:
        # Vincular el socket al host y puerto especificados
        servidor.bind((host, puerto))
        # Esperar conexiones entrantes (máximo 1)
        servidor.listen(1)
        print('Esperando por una conexión...')
        # Aceptar la conexión entrante
        conexion, direccion = servidor.accept()
        print('Conexión establecida desde:', direccion)

    while True:
        try:
            hedge.dataEvent.wait(1)
            hedge.dataEvent.clear()
            hedge2.dataEvent.wait(1)
            hedge2.dataEvent.clear()

            if (hedge.positionUpdated):
                posicionBaliza1 = hedge.position()
                nBaliza, xBaliza1, yBaliza1, timeStamp1 = float(posicionBaliza1[0]),
float(posicionBaliza1[1]), float(posicionBaliza1[2]), float(posicionBaliza1[5])
                datosBuffer = [nBaliza, xBaliza1, yBaliza1, timeStamp1]
                bufferBaliza1.append(datosBuffer)
```

```
                #  print(posicionBaliza1[0],"Time:",  timeStamp1,  "Dif-anterior:",
timeStamp1 - timestampAnterior1)
                # timestampAnterior1 = timeStamp1

                with open("localizacionBalizas.txt","a") as file:
                    aEscribir1  =  str(nBaliza)  +  ","  +  str(xBaliza1)  +  ","  +
str(yBaliza1) + "," + str(timeStamp1) + "\n"
                    file.write(aEscribir1)

            if (hedge2.positionUpdated):
                posicionBaliza2 = hedge2.position()
                nBaliza2,       xBaliza2,       yBaliza2,       timeStamp2       =
float(posicionBaliza2[0]),   float(posicionBaliza2[1]),   float(posicionBaliza2[2]),
float(posicionBaliza2[5])
                datosBuffer = [nBaliza2, xBaliza2, yBaliza2, timeStamp2]
                bufferBaliza2.append(datosBuffer)

                #  print(posicionBaliza2[0],"Time:",  timeStamp2,  "Dif-anterior:",
timeStamp2 - timestampAnterior2)
                # timestampAnterior2 = timeStamp2

                with open("localizacionBalizas.txt","a") as file:
                    aEscribir2  =  str(nBaliza2)  +  ","  +  str(xBaliza2)  +  ","  +
str(yBaliza2) + "," + str(timeStamp2) + "\n"
                    file.write(aEscribir2)


            if len(bufferBaliza1) > 0 and len(bufferBaliza2) > 0:

                if len(bufferBaliza1) == len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[-1]
                    lecturaBuffer2 = bufferBaliza2[-1]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True

                        anguloRespectoTrayec,           distanciaATrayec           =
anguloRespectoTrayectoria(lecturaBuffer1, lecturaBuffer2)
                        mensaje   =   'Angulo   respecto   a   trayectoria:   '   +
str(anguloRespectoTrayec) + ' y Distancia a trayectoria: ' + str(distanciaATrayec)
                        bufferBaliza1.clear()
                        bufferBaliza2.clear()

                elif len(bufferBaliza1) > len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[0]
                    lecturaBuffer2 = bufferBaliza2[-1]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True

                        anguloRespectoTrayec,           distanciaATrayec           =
anguloRespectoTrayectoria(lecturaBuffer1, lecturaBuffer2)
                        mensaje   =   'Angulo   respecto   a   trayectoria:   '   +
str(anguloRespectoTrayec) + ' y Distancia a trayectoria: ' + str(distanciaATrayec)
                        bufferBaliza1.popleft()
                        bufferBaliza2.pop()
                    if not valido:
```

```
                        lecturaBuffer1 = bufferBaliza1[-1]
                        lecturaBuffer2 = bufferBaliza2[-1]
                        if    abs(lecturaBuffer1[3]    -    lecturaBuffer2[3])    <
tasaMilisegundos:
                            #     print("Restaré",     lecturaBuffer1[0],     "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                            valido = True

                            anguloRespectoTrayec,         distanciaATrayec         =
anguloRespectoTrayectoria(lecturaBuffer1, lecturaBuffer2)
                            mensaje   =   'Angulo   respecto   a   trayectoria:   '   +
str(anguloRespectoTrayec) + ' y Distancia a trayectoria: ' + str(distanciaATrayec)
                            bufferBaliza1.clear()
                            bufferBaliza2.clear()

                elif len(bufferBaliza1) < len(bufferBaliza2):

                    lecturaBuffer1 = bufferBaliza1[-1]
                    lecturaBuffer2 = bufferBaliza2[0]
                    if     abs(lecturaBuffer1[3]     -     lecturaBuffer2[3])     <
tasaMilisegundos:
                        #       print("Restaré",       lecturaBuffer1[0],       "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                        valido = True
                        anguloRespectoTrayec,           distanciaATrayec           =
anguloRespectoTrayectoria(lecturaBuffer1, lecturaBuffer2)
                        mensaje   =   'Angulo   respecto   a   trayectoria:   '   +
str(anguloRespectoTrayec) + ' y Distancia a trayectoria: ' + str(distanciaATrayec)
                        bufferBaliza1.pop()
                        bufferBaliza2.popleft()
                    if not valido:
                        lecturaBuffer1 = bufferBaliza1[-1]
                        lecturaBuffer2 = bufferBaliza2[-1]
                        if    abs(lecturaBuffer1[3]    -    lecturaBuffer2[3])    <
tasaMilisegundos:
                            #     print("Restaré",     lecturaBuffer1[0],     "y",
lecturaBuffer2[0],     "->",     lecturaBuffer1[3],     "-",     lecturaBuffer2[3],
":",lecturaBuffer1[3] - lecturaBuffer2[3])
                            valido = True
                            anguloRespectoTrayec,         distanciaATrayec         =
anguloRespectoTrayectoria(lecturaBuffer1, lecturaBuffer2)
                            mensaje   =   'Angulo   respecto   a   trayectoria:   '   +
str(anguloRespectoTrayec) + ' y Distancia a trayectoria: ' + str(distanciaATrayec)
                            bufferBaliza1.clear()
                            bufferBaliza2.clear()

                valido = False


            conexion.sendall(mensaje.encode())
            print('[SERVER] Envío:', mensaje)
            anguloRespectoTrayec = None
            distanciaATrayec = None



        except KeyboardInterrupt:
            print("El servidor ha sido detenido por el usuario.")
            hedge.stop()  # stop and close serial port
            hedge2.stop()  # stop and close serial port
            sys.exit()
main()
```

Para las pruebas realizadas en el laboratorio emulando la situación real del vehículo, se desarrolló un código destinado a la recepción de la información por parte del cliente. El código desarrollado se denomina ‘`recepcionInfoSocketTiempoReal.py`’ y su contenido es:

```
import socket

# Definir el host y el puerto al que el cliente se conectará
host = '192.168.0.3'  # Dirección IP del servidor
puerto = 2048

# Crear un objeto de socket para el cliente
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as cliente:
    # Conectar al servidor
    cliente.connect((host, puerto))
    print("Conexión establecida con el servidor")

    try:
        while True:
            # Recibir datos del servidor
            datos = cliente.recv(1024)
            if not datos:
                break
            print('[CLIENT] Recibo: ', datos.decode())

    except KeyboardInterrupt:
        print('El cliente ha sido detenido por el usuario')
```

Anexo VIII

Parte de los ficheros desarrollados hacen uso de la siguiente librería '`marvelmindLib.py`' desarrollada por Marvelmind, cuyo código a fecha de presentación del trabajo es el que se mostrará a continuación. No es descartable que en un futuro la empresa lo actualice y cambie el código, por ello, considero relevante incluir el código.

```
#!/usr/bin/env python
#
# marvelmind.py - small class for recieve and parse coordinates from Marvelmind
mobile beacon by USB/serial port
# Written by Alexander Rudykh. Support: info@marvelmind.com
#
### Attributes:
#
#    adr - address of mobile beacon (from Dashboard) for data filtering. If it is
None, every read data will be appended to buffer.
#       default: None
#
#   tty - serial port device name (physical or USB/virtual). It should be provided
as an argument:
#       /dev/ttyACM0 - typical for Linux / Raspberry Pi
#       /dev/tty.usbmodem1451 - typical for Mac OS X
#
#   baud - baudrate. Should be match to baudrate of hedgehog-beacon
#       default: 9600
#
#   maxvaluescount - maximum count of measurements of coordinates stored in buffer
#       default: 3
#
#   valuesUltrasoundPosition - buffer of measurements
#
#   debug - debug flag which activate console output
#       default: False
#
#   pause - pause flag. If True, class would not read serial data
#
#   terminationRequired - If True, thread would exit from main loop and stop
#
#
### Methods:
#
#   __init__ (self, tty="/dev/ttyACM0", baud=9600, maxvaluescount=3, debug=False)
#       constructor
#
#   print_position(self)
#       print last measured data in default format
#
#   position(self)
#       return last measured data in array [x, y, z, timestamp]
#
#   distances(self)
#      return raw distances in array [hedge, beacon0, distance0, beacon1, distance1,
beacon2, distance2, beacon3, distance3, timestamp]
#
#   raw_imu(self)
#       return raw IMU (accelerometer, gyro, magnetometer) data in array [AX,AY,AZ,
GX,GY,GZ, MX,MY,MZ, timestamp]
#
#   imu_fusion(self)
#        return IMU fusion (position, quaternion, velocity, acceleration) data in
array [X,Y,Z, QW,QX,QY,QZ, VX,VY,VZ, AX,AY,AZ, timestamp]
#
```

```
#   telemetry(self)
#       return telemetry (battery voltage, RSSI) data in array [vbat, RSSI]
#
#   quality(self)
#       return location quality data in array [address, quality]
#
#   waypoint(self)
#        return last received waypoint in array [type, index, total_number, param1,
param2, param3]
#
#   stop(self)
#       stop infinite loop and close port
#
#
###

###
# Changes:
# lastValues -> valuesUltrasoundPosition
# recieveLinearDataCallback -> recieveUltrasoundPositionCallback
# lastImuValues -> valuesImuRawData
# recieveAccelerometerDataCallback -> recieveImuRawDataCallback
# mm and cm -> m
###

import serial
import struct
import collections
import time
from threading import Thread
from threading import Event
import math
import datetime

CRC16_TABLE = (
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
```

```
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 )

def _crc16(arr, offset, size, table):
    if (len(arr)<offset+size):
        return -1
    n= 0
    wCRCWord = 0xFFFF
    while (n<size):
        nTemp = arr[n+offset] ^ wCRCWord
        wCRCWord = wCRCWord >> 8
        wCRCWord  = wCRCWord ^ table[nTemp & 0xff]
        n= n+1
    return wCRCWord

# Modbus RTU CRC-16
def crc16_mb(arr, offset, size):
    return _crc16(arr, offset, size, CRC16_TABLE)


class MarvelmindHedge (Thread):
    def __init__ (self, adr=None, tty="/dev/ttyACM0", baud=9600, maxvaluescount=3,
debug=False,                                    recieveUltrasoundPositionCallback=None,
recieveImuRawDataCallback=None,                          recieveImuDataCallback=None,
recieveUltrasoundRawDataCallback=None):
        self.tty = tty  # serial
        self.baud = baud  # baudrate
        self.debug = debug  # debug flag
        self._bufferSerialDeque = collections.deque(maxlen=255)  # serial buffer
        self._bufferSerialReply= bytearray(6)
        self._userDataArray= bytearray(255)

        self.valuesUltrasoundPosition  =  collections.deque([[0]*6]*maxvaluescount,
maxlen=maxvaluescount) # ultrasound position buffer
        self.recieveUltrasoundPositionCallback = recieveUltrasoundPositionCallback

        self.valuesImuRawData      =      collections.deque([[0]*10]*maxvaluescount,
maxlen=maxvaluescount) # raw imu data buffer
        self.recieveImuRawDataCallback = recieveImuRawDataCallback

        self.valuesImuData       =       collections.deque([[0]*14]*maxvaluescount,
maxlen=maxvaluescount) # processed imu data buffer
        self.recieveImuDataCallback = recieveImuDataCallback

        self.valuesUltrasoundRawData   =   collections.deque([[0]*5]*maxvaluescount,
maxlen=maxvaluescount)
        self.recieveUltrasoundRawDataCallback = recieveUltrasoundRawDataCallback

        self.valuesTelemetryData     =     collections.deque([[0]*5]*maxvaluescount,
maxlen=maxvaluescount)
        self.valuesQualityData      =      collections.deque([[0]*5]*maxvaluescount,
maxlen=maxvaluescount)
        self.valuesWaypointData     =     collections.deque([[0]*16]*maxvaluescount,
maxlen=maxvaluescount)
        self.valuesUserData       =       collections.deque([[0]*5]*maxvaluescount,
maxlen=maxvaluescount)


        self.pause = False
        self.terminationRequired = False

        self.positionUpdated= False;
        self.distancesUpdated= False;
        self.rawImuUpdated= False;
        self.fusionImuUpdated= False;
        self.telemetryUpdated= False
        self.qualityUpdated= False
```

```
        self.waypointsUpdated= False
        self.userDataUpdated= False

        self.adr = adr
        self.serialPort = None

        self.dataEvent= Event()
        Thread.__init__(self)

    def print_position(self):
        if (self.position()[6]):
            tsec= int(math.trunc(self.position()[5]/1000.0))
            tmsec= int(self.position()[5] % 1000)
            dt = datetime.datetime.utcfromtimestamp(tsec)
            print ("Hedge {:d}: X: {:.3f} m, Y: {:.3f} m, Z: {:.3f} m, Angle: {:d}
at     time     T:     {:%Y-%m-%d     %H:%M:%S}-{:03d}".format(self.position()[0],
self.position()[1], self.position()[2], self.position()[3], self.position()[4], dt,
tmsec))
        elif (isinstance(self.position()[1], int)):
            print ("Hedge {:d}: X: {:.3f} m, Y: {:.3f} m, Z: {:.3f} m, Angle: {:d}
at     time     T:     {:.3f}".format(self.position()[0],     self.position()[1],
self.position()[2],             self.position()[3],             self.position()[4],
self.position()[5]/1000.0))
        else:
            print ("Hedge {:d}: X: {:.3f}, Y: {:.3f}, Z: {:.3f}, Angle: {:d} at time
T:   {:.3f}".format(self.position()[0],   self.position()[1],   self.position()[2],
self.position()[3], self.position()[4], self.position()[5]/1000.0))

    def position(self):
        self.positionUpdated= False
        return list(self.valuesUltrasoundPosition)[-1];

    def print_distances(self):
        self.distancesUpdated= False
        rd= self.distances()
        if (rd[10]):
            tsec= int(math.trunc(rd[9]/1000.0))
            tmsec= int(rd[9] % 1000)
            dt = datetime.datetime.utcfromtimestamp(tsec)
            print   ("Distances:   From:H{:d}   to     B{:d}:{:.3f},   B{:d}:{:.3f},
B{:d}:{:.3f}, B{:d}:{:.3f}   at time T: {:%Y-%m-%d %H:%M:%S}-{:03d}".format(rd[0],
rd[1], rd[2], rd[3], rd[4], rd[5], rd[6], rd[7], rd[8], dt, tmsec))
        else:
            print   ("Distances:   From:H{:d}   to     B{:d}:{:.3f},   B{:d}:{:.3f},
B{:d}:{:.3f}, B{:d}:{:.3f}   at time T: {:.3f}".format(rd[0], rd[1], rd[2], rd[3],
rd[4], rd[5], rd[6], rd[7], rd[8], rd[9]/1000.0))

    def distances(self):
        return list(self.valuesUltrasoundRawData)[-1];

    def print_raw_imu(self):
        self.rawImuUpdated= False
        ri= self.raw_imu()
        if (ri[10]):
            tsec= int(math.trunc(ri[9]/1000.0))
            tmsec= int(ri[9] % 1000)
            dt = datetime.datetime.utcfromtimestamp(tsec)
            print ("Raw IMU: AX:{:d}, AY:{:d}, AZ:{:d},   GX:{:d}, GY:{:d}, GZ:{:d},
MX:{:d}, MY:{:d}, MZ:{:d},   at time T: {:%Y-%m-%d %H:%M:%S}-{:03d}".format(ri[0],
ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], dt, tmsec))
        else:
            print ("Raw IMU: AX:{:d}, AY:{:d}, AZ:{:d},   GX:{:d}, GY:{:d}, GZ:{:d},
MX:{:d}, MY:{:d}, MZ:{:d},   at time T: {:.3f}".format(ri[0], ri[1], ri[2], ri[3],
ri[4], ri[5], ri[6], ri[7], ri[8], ri[9]/1000.0))

    def raw_imu(self):
```

```
        return list(self.valuesImuRawData)[-1];

    def print_imu_fusion(self):
        self.fusionImuUpdated= False
        ifd= self.imu_fusion()
        if (ifd[14]):
            tsec= int(math.trunc(ifd[13]/1000.0))
            tmsec= int(ifd[13] % 1000)
            dt = datetime.datetime.utcfromtimestamp(tsec)
            print ("IMU fusion: X:{:.3f}, Y:{:.3f}, Z:{:.3f},  QW:{:.3f}, QX:{:.3f},
QY:{:.3f}, QZ:{:.3f},    VX:{:.3f}, VY:{:.3f}, VZ:{:.3f},    AX:{:.3f}, AY:{:.3f},
AZ:{:.3f},   at time T: {:%Y-%m-%d %H:%M:%S}-{:03d}".format(ifd[0], ifd[1], ifd[2],
ifd[3], ifd[4], ifd[5], ifd[6], ifd[7], ifd[8], ifd[9], ifd[10], ifd[11], ifd[12],
dt, tmsec))
        else:
            print ("IMU fusion: X:{:.3f}, Y:{:.3f}, Z:{:.3f},  QW:{:.3f}, QX:{:.3f},
QY:{:.3f}, QZ:{:.3f},    VX:{:.3f}, VY:{:.3f}, VZ:{:.3f},    AX:{:.3f}, AY:{:.3f},
AZ:{:.3f},   at time T: {:.3f}".format(ifd[0], ifd[1], ifd[2], ifd[3], ifd[4], ifd[5],
ifd[6], ifd[7], ifd[8], ifd[9], ifd[10], ifd[11], ifd[12], ifd[13]/1000.0))

    def imu_fusion(self):
        return list(self.valuesImuData)[-1];

    def print_telemetry(self):
        self.telemetryUpdated= False
        td= self.telemetry()
        print  ("Telemetry:  Vbat:  {:.3f}V,  RSSI:  {:d}dBm".format(td[0]/1000.0,
td[1]))

    def telemetry(self):
        return list(self.valuesTelemetryData)[-1];

    def print_quality(self):
        self.qualityUpdated= False
        qd= self.quality()
        print ("Quality: Address: {:d}, Quality: {:d}%".format(qd[0], qd[1]))

    def quality(self):
        return list(self.valuesQualityData)[-1];

    def print_waypoint(self):
        self.waypointsUpdated= False
        wd= self.waypoint()
        print ("Movement: Type: {:d}, Index: {:d}, Total: {:d},  Param1: {:d}, Param2:
{:d}, Param3: {:d}".format(wd[0], wd[1], wd[2], wd[3], wd[4], wd[5]))

    def waypoint(self):
        return list(self.valuesWaypointData)[-1];

    def replyWaypointRcvSuccess(self):
        if (self.adr is None):
            return
        self._bufferSerialReply[0]= self.adr
        self._bufferSerialReply[1]= 0x4a
        self._bufferSerialReply[2]= 0x01
        self._bufferSerialReply[3]= 0x02

        CRC_calcReply= crc16_mb(self._bufferSerialReply, 0, 4)
        self._bufferSerialReply[4]= CRC_calcReply & 0xff
        self._bufferSerialReply[5]= (CRC_calcReply>>8) & 0xff
        self.serialPort.write(self._bufferSerialReply)

    def user_data(self):
        return list(self.valuesUserData)[-1];

    def print_user_data(self):
```

```
        self.userDataUpdated= False
        ud= self.user_data()
        tval= ud[0][0]
        tsec= int(math.trunc(tval/1000.0))
        tmsec= int(tval % 1000)
        dt = datetime.datetime.utcfromtimestamp(tsec)
        print("User data    at time T: {:%Y-%m-%d %H:%M:%S}-{:03d} : ".format(dt,
tmsec));
        dsize= ud[1][0]
        s= "    "
        for x in range(0, dsize-8):
            s= s + str(ud[2][x]).zfill(3) + ", "
        print(s)


    def stop(self):
        self.terminationRequired = True
        print ("stopping")

    def run(self):
        while (not self.terminationRequired):
            if (not self.pause):
                try:
                    if (self.serialPort is None):
                        print ("Trying open serial port: {:s}".format(self.tty))
                        self.serialPort    =    serial.Serial(self.tty,    self.baud,
timeout=3)
                        print ("Serial port opened")
                    readChar = self.serialPort.read(1)
                    while (readChar is not None) and (readChar is not '') and (not
self.terminationRequired):
                        self._bufferSerialDeque.append(readChar)
                        readChar = self.serialPort.read(1)
                        bufferList = list(self._bufferSerialDeque)

                        strbuf = (b''.join(bufferList))

                        pktHdrOffset = strbuf.find(b'\xff\x47')
                        if (pktHdrOffset == -1):
                            pktHdrOffset = strbuf.find(b'\xff\x4a')
                        if (pktHdrOffset >= 0 and len(bufferList) > pktHdrOffset + 4
and pktHdrOffset<220):
#                           print(bufferList)
                            isMmMessageDetected = False
                            isCmMessageDetected = False
                            isNTMmMessageDetected = False
                            isRawImuMessageDetected = False
                            isNTRawImuMessageDetected = False
                            isImuMessageDetected = False
                            isNTImuMessageDetected = False
                            isDistancesMessageDetected = False
                            isNTDistancesMessageDetected = False
                            isTelemetryMessageDetected= False
                            isQualityMessageDetected= False
                            isWaypointsMessageDetected= False
                            isUserDataMessageDetected= False
                            anyMsgFound= False
                            isRealtime= False

                            if (not anyMsgFound):
                                pktHdrOffsetCm = strbuf.find(b'\xff\x47\x01\x00')
                                if (pktHdrOffsetCm != -1):
                                    anyMsgFound = True
                                    isCmMessageDetected = True
                                    if  (self.debug):  print  ('Message  with  US-
position(cm) was detected')
```

```
                            if (not anyMsgFound):
                                pktHdrOffsetMm = strbuf.find(b'\xff\x47\x11\x00')
                                if (pktHdrOffsetMm != -1):
                                    anyMsgFound = True
                                    isMmMessageDetected = True
                                    if  (self.debug):  print  ('Message  with  US-
position(mm) was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetRawImu                                  =
strbuf.find(b'\xff\x47\x03\x00')
                                if (pktHdrOffsetRawImu != -1):
                                    anyMsgFound = True
                                    isRawImuMessageDetected = True
                                    if (self.debug): print ('Message with raw IMU
data was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetDistances                               =
strbuf.find(b'\xff\x47\x04\x00')
                                if (pktHdrOffsetDistances != -1):
                                    anyMsgFound = True
                                    isDistancesMessageDetected = True
                                    if (self.debug): print ('Message with distances
was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetImu = strbuf.find(b'\xff\x47\x05\x00')
                                if (pktHdrOffsetImu != -1):
                                    anyMsgFound = True
                                    isImuMessageDetected = True
                                    if (self.debug): print ('Message with processed
IMU data was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetTelemetry                               =
strbuf.find(b'\xff\x47\x06\x00')
                                if (pktHdrOffsetTelemetry != -1):
                                    anyMsgFound = True
                                    isTelemetryMessageDetected= True
                                    if (self.debug): print ('Message with telemetry
data was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetQuality=
strbuf.find(b'\xff\x47\x07\x00')
                                if (pktHdrOffsetQuality != -1):
                                    anyMsgFound = True
                                    isQualityMessageDetected= True
                                    if  (self.debug):  print  ('Message  with  quality
data was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetWaypoints=
strbuf.find(b'\xff\x4a\x01\x02')
                                if (pktHdrOffsetWaypoints != -1):
                                    anyMsgFound = True
                                    isWaypointsMessageDetected= True
                                    if (self.debug): print ('Message with waypoints
data was detected')

                            if (not anyMsgFound):
                                pktHdrOffsetMm_NT                                   =
strbuf.find(b'\xff\x47\x81\x00')
                                if (pktHdrOffsetMm_NT != -1):
```

```
                                    anyMsgFound = True
                                    isRealtime= True
                                    isNTMmMessageDetected = True
                                    if (self.debug): print ('Message with realtime
US-position(mm) was detected')

                                if (not anyMsgFound):
                                    pktHdrOffsetRawImu_NT                                =
strbuf.find(b'\xff\x47\x83\x00')
                                    if (pktHdrOffsetRawImu_NT != -1):
                                        anyMsgFound = True
                                        isRealtime= True
                                        isNTRawImuMessageDetected = True
                                        if (self.debug): print ('Message with realtime
raw IMU data was detected')

                                if (not anyMsgFound):
                                    pktHdrOffsetDistances_NT                             =
strbuf.find(b'\xff\x47\x84\x00')
                                    if (pktHdrOffsetDistances_NT != -1):
                                        anyMsgFound = True
                                        isRealtime= True
                                        isNTDistancesMessageDetected = True
                                        if (self.debug): print ('Message with realtime
raw distances was detected')

                                if (not anyMsgFound):
                                    pktHdrOffsetImu_NT                                   =
strbuf.find(b'\xff\x47\x85\x00')
                                    if (pktHdrOffsetImu_NT != -1):
                                        anyMsgFound = True
                                        isRealtime= True
                                        isNTImuMessageDetected = True
                                        if (self.debug): print ('Message with realtime
processed IMU data was detected')

                                if (not anyMsgFound):
                                    pktHdrUserData = strbuf.find(b'\xff\x4a\x80\x02')
                                    if (pktHdrUserData != -1):
                                        anyMsgFound = True
                                        isRealtime= True
                                        isUserDataMessageDetected = True
                                        if  (self.debug):  print  ('Message  with  user
payload data was detected')

                                msgLen = ord(bufferList[pktHdrOffset + 4])
                                if (self.debug): print ('Message length: ', msgLen)

                                self.dataEvent.set()

                                try:
                                    if (len(bufferList) > pktHdrOffset + 4 + msgLen +
2):
                                        usnCRC16 = 0
                                        if (isCmMessageDetected):
                                            usnTimestamp,  usnX,  usnY,  usnZ,  usnAdr,
usnAngle, usnCRC16 = struct.unpack_from ('<LhhhxBhxxH', strbuf, pktHdrOffset + 5)
                                            usnX = usnX/100.0
                                            usnY = usnY/100.0
                                            usnZ = usnZ/100.0
                                            usnAngle = 0b0000111111111111&usnAngle
                                        elif (isMmMessageDetected):
                                            usnTimestamp,  usnX,  usnY,  usnZ,  usnFlags,
usnAdr, usnAngle, usnCRC16 = struct.unpack_from ('<LlllBBhxxH', strbuf, pktHdrOffset
+ 5)
                                            usnX = usnX/1000.0
```

```
                                        usnY = usnY/1000.0
                                        usnZ = usnZ/1000.0
                                        usnAngle = 0b0000111111111111&usnAngle
                                        if ((usnFlags&0x40)==0):
                                            self.adr= usnAdr
                                    elif (isNTMmMessageDetected):
                                        usnTimestamp, usnX, usnY, usnZ, usnFlags,
usnAdr, usnAngle, usnCRC16 = struct.unpack_from ('<qlllBBhxxH', strbuf, pktHdrOffset
+ 5)
                                        usnX = usnX/1000.0
                                        usnY = usnY/1000.0
                                        usnZ = usnZ/1000.0
                                        usnAngle = 0b0000111111111111&usnAngle
                                        if ((usnFlags&0x40)==0):
                                            self.adr= usnAdr
                                    elif (isRawImuMessageDetected):
                                        ax, ay, az, gx, gy, gz, mx, my, mz,
timestamp, usnCRC16 = struct.unpack_from ('<hhhhhhhhhxxxxxxLxxxxH', strbuf,
pktHdrOffset + 5)
                                    elif (isNTRawImuMessageDetected):
                                        ax, ay, az, gx, gy, gz, mx, my, mz,
timestamp, usnCRC16 = struct.unpack_from ('<hhhhhhhhhxxxxxxqxxxxH', strbuf,
pktHdrOffset + 5)
                                    elif (isImuMessageDetected):
                                        x, y, z, qw, qx, qy, qz, vx, vy, vz, ax, ay,
az, timestamp, usnCRC16 = struct.unpack_from ('<lllhhhhhhhhhhxxLxxxxH', strbuf,
pktHdrOffset + 5)
                                    elif (isNTImuMessageDetected):
                                        x, y, z, qw, qx, qy, qz, vx, vy, vz, ax, ay,
az, timestamp, usnCRC16 = struct.unpack_from ('<lllhhhhhhhhhhxxqxxxxH', strbuf,
pktHdrOffset + 5)
                                    elif (isDistancesMessageDetected):
                                        HedgeAdr, b1, b1d, b2, b2d, b3, b3d, b4, b4d,
timestamp,usnCRC16 = struct.unpack_from ('<BBlxBlxBlxBlxLxxxH', strbuf, pktHdrOffset
+ 5)
                                    elif (isNTDistancesMessageDetected):
                                        HedgeAdr, b1, b1d, b2, b2d, b3, b3d, b4, b4d,
timestamp,usnCRC16 = struct.unpack_from ('<BBlxBlxBlxBlxqxxxH', strbuf, pktHdrOffset
+ 5)
                                    elif (isTelemetryMessageDetected):
                                        vbat, rssi_dbm, usnCRC16 =
struct.unpack_from ('<HbxxxxxxxxxxxxxH', strbuf, pktHdrOffset + 5)
                                    elif (isQualityMessageDetected):
                                        quality_addr, quality_per, usnCRC16 =
struct.unpack_from ('<BBxxxxxxxxxxxxxxH', strbuf, pktHdrOffset + 5)
                                    elif (isWaypointsMessageDetected):
                                        mvmType, mvmIndex, mvmTotal, mvmParam1,
mvmParam2, mvmParam3, usnCRC16 = struct.unpack_from ('<BBBhhhxxxH', strbuf,
pktHdrOffset + 5)
                                    elif isUserDataMessageDetected:
                                        timestamp = struct.unpack_from ('<q',
strbuf, pktHdrOffset + 5)
                                        userDataSize = struct.unpack_from ('<B',
strbuf, pktHdrOffset + 4)
                                        for x in range(0, userDataSize[0]-8):
                                            tmpv= struct.unpack_from ('<B', strbuf,
pktHdrOffset + 5 + 8 + x)[0]
                                            self._userDataArray[x] = tmpv
                                        usnCRC16 = struct.unpack_from ('<H', strbuf,
pktHdrOffset + 5 + userDataSize[0])[0]

f                                           CRC_calc = crc16_mb(bytearray(strbuf),
pktHdrOffset, msgLen+5)

                                    if CRC_calc == usnCRC16:
```

```
                                        if          (isMmMessageDetected          or
isCmMessageDetected or isNTMmMessageDetected):
                                            self.positionUpdated= True
                                            value  =  [usnAdr,  usnX,  usnY,  usnZ,
usnAngle, usnTimestamp, isRealtime]
                                            if (self.adr == usnAdr or self.adr is
None):
self.valuesUltrasoundPosition.append(value)
                                                if
(self.recieveUltrasoundPositionCallback is not None):
self.recieveUltrasoundPositionCallback()
                                        elif      (isRawImuMessageDetected      or
isNTRawImuMessageDetected):
                                            self.rawImuUpdated= True
                                            value = [ax, ay, az, gx, gy, gz, mx, my,
mz, timestamp, isRealtime]
                                            self.valuesImuRawData.append(value)
                                            if  (self.recieveImuRawDataCallback  is
not None):
                                                self.recieveImuRawDataCallback()
                                        elif     (isDistancesMessageDetected     or
isNTDistancesMessageDetected):
                                            value = [HedgeAdr, b1, b1d/1000.0, b2,
b2d/1000.0, b3, b3d/1000.0, b4, b4d/1000.0, timestamp, isRealtime]
self.valuesUltrasoundRawData.append(value)
                                            self.distancesUpdated= True
                                            if
(self.recieveUltrasoundRawDataCallback is not None):
self.recieveUltrasoundRawDataCallback()
                                        elif        (isImuMessageDetected        or
isNTImuMessageDetected):
                                            self.fusionImuUpdated= True
                                            value = [x/1000.0, y/1000.0, z/1000.0,
qw/10000.0, qx/10000.0, qy/10000.0, qz/10000.0, vx/1000.0, vy/1000.0, vz/1000.0,
ax/1000.0,ay/1000.0,az/1000.0, timestamp, isRealtime]
                                            self.valuesImuData.append(value)
                                            if (self.recieveImuDataCallback is not
None):
                                                self.recieveImuDataCallback()
                                        elif (isTelemetryMessageDetected):
                                            self.telemetryUpdated= True
                                            value = [vbat, rssi_dbm]
                                            self.valuesTelemetryData.append(value)
                                        elif (isQualityMessageDetected):
                                            self.qualityUpdated= True
                                            value = [quality_addr, quality_per]
                                            self.valuesQualityData.append(value)
                                        elif (isWaypointsMessageDetected):
                                            self.waypointsUpdated= True
                                            value  =  [mvmType,  mvmIndex,  mvmTotal,
mvmParam1, mvmParam2, mvmParam3]
                                            self.valuesWaypointData.append(value)
                                            self.replyWaypointRcvSuccess()
                                        elif (isUserDataMessageDetected):
                                            self.userDataUpdated= True
                                            value   =   [timestamp,   userDataSize,
self._userDataArray]
                                            self.valuesUserData.append(value)
                                    else:
                                        if self.debug:
                                            print ('\n*** CRC ERROR')
```

```
                                    if pktHdrOffset == -1:
                                        if self.debug:
                                            print ('\n*** ERROR: Marvelmind USNAV
beacon packet header not found (check modem board or radio link)')
                                        continue
                                    elif pktHdrOffset >= 0:
                                        if self.debug:
                                            print ('\n>> Found USNAV beacon packet
header at offset %d' % pktHdrOffset)
                                    for x in range(0, pktHdrOffset + msgLen + 7):
                                        self._bufferSerialDeque.popleft()
                                except struct.error:
                                    print ('smth wrong')
                        except OSError:
                            print ('*** ERROR: OS error (possibly serial port is not
available)')
                            time.sleep(3)
                        except serial.SerialException:
                            print ('*** ERROR: serial port error (possibly beacon is reset,
powered down or in sleep mode). Restarting reading process...')
                            self.serialPort = None
                            time.sleep(3)
                    else:
                        time.sleep(1)

                if (self.serialPort is not None):
                    self.serialPort.close()
```