ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Ingeniería de Sistemas y Automática

Área de Conocimiento de Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

NAVEGACIÓN REACTIVA CON UN ROBOT MÓVIL ACKERMANN. SIMULACIÓN EN COPPELIA ROBOTICS.

Grado en

Ingeniería Electrónica, Robótica y Mecatrónica

Autor: KEVIN ADRIAN RIVERO CHAKAL

Tutor: CIPRIANO GALINDO ANDRADES

Cotutor: JAVIER GONZALEZ MONROY

MÁLAGA, Octubre de 2.024

Agradecimientos

A mis padres y familiares, por su incondicional apoyo a lo largo de mi carrera y, especialmente, durante la realización de este trabajo.

A mis compañeros de clase y amigos, que han formado parte de mi experiencia en la universidad, y con los que he compartido todos nuestros logros.

Al Prof. Cipriano Galindo Andrades y al Prof. Javier González Monroy, por su guía, apoyo y consejo para el desarrollo de mi Trabajo de Fin de Grado.

A Andrea, quien creyó en mi desde el inicio, y que sin ella no hubiese llegado hasta donde estoy ahora.

Resumen:

El presente trabajo de fin de grado aborda la programación en C++ y Python de un robot Hunter 2.0 de modelo Ackermann, con el objetivo de desarrollar capacidades de navegación autónoma y reactiva en diversos entornos exteriores. El sistema se integra con un sensor hokuyo para la adquisición de datos de su entorno.

Dentro de los objetivos de este proyecto se encuentra el diseño e implementación de un modelo virtual que simule el movimiento del robot real a través del programa gráfico CoppeliaRobotics. Durante el desarrollo de este proyecto, este modelo ha sido usado para realizar pruebas con algoritmos diseñados en el ámbito de la navegación autónoma. Estos algoritmos fueron programados en ROS2.

Entre estos se encuentra el uso del stack de navegación Nav2, el cual cumple como un sistema global de planificación y seguimiento de trayectorias que se actualizan en tiempo real. Adicionalmente, se diseñó un algoritmo de aparcamiento en línea, el cual incluye un análisis del entorno con el fin de encontrar un puesto libre y los movimientos relacionados con la maniobra de aparcamiento.

Finalmente, con el fin de comprobar la eficiencia de estos códigos, se han realizado diferentes experiencias en distintos ambientes que pongan a prueba los algoritmos y las trayectorias diseñadas por nav2. Para esto, se han creado escenarios de simulación y usado habitaciones y pasillos en la E.T.S.I Informática para crear mapas con distintos obstáculos o dimensiones que simulen aparcamientos, en los que se ha estudiado la navegación y autonomía del robot.

Palabras Clave: Ackerman, ROS2, CoppeliaRobotics, navegación reactiva, evasión de obstáculos, aparcamiento, Nav2.

Abstract:

This bachelor’s thesis focuses on programming a Hunter 2.0 Ackermann model robot using C++ and Python, with the goal of developing autonomous and reactive navigation capabilities in various outdoor environments. The system is integrated with a Hokuyo sensor for environmental data acquisition.

One of the objectives of this project is the design and implementation of a virtual model that simulates the movement of the real robot through the graphical program CoppeliaRobotics. This model has been utilized during the project's development to test algorithms designed for autonomous navigation. These algorithms were programmed in ROS2.

Among the implemented features is the use of the Nav2 navigation stack, which serves as a global system that plans and follows trajectories updated in real-time. Moreover, a parallel parking algorithm was designed, which includes an analysis of the environment to identify available spaces, and the movements related to the parking maneuver.

Finally, to verify the efficiency of these codes, various tests were conducted in different environments to challenge the algorithms and the trajectories designed by Nav2. For this purpose, simulation scenarios were created, and rooms and hallways in the E.T.S.I Informática were used to create maps with various obstacles and dimensions simulating parking areas, where the robot's navigation and autonomy were studied.

Keywords: Ackermann, ROS2, CoppeliaRobotics, reactive navigation, obstacles avoidance, parking, Nav2.

Índice de contenido

Índice de Figuras

Índice de Tablas

1. Introducción

Este trabajo se motiva por la necesidad de avanzar en el diseño e implementación de sistemas de navegación autónomos y robustos en robots con cinemática Ackermann, la cual presenta desafíos específicos en su movilidad y limitaciones, como la incapacidad de girar sobre su centro geométrico, en comparación con otros modelos que permiten una mayor variedad de trayectorias [14].

La elección de este modelo de dirección se debe a su gran estabilidad ante terrenos desafiantes. Debido a su morfología, presenta una reducción del deslizamiento lateral de las ruedas [16], lo cual mejora la tracción y presentan mayor eficiencia con radios de giro reducidos en comparación a otros modelos, lo cual es relevante en maniobras precisas en espacios reducidos [2]. Además, presentan una gran similitud a los automóviles comunes, lo cual permite una fácil integración de la navegación autónoma en este tipo de vehículos.

Con este proyecto también se busca abrir nuevas oportunidades en la robótica móvil creando aplicaciones en áreas urbanas y rurales, ya sea de reconocimiento y exploración como de transporte de materiales, permitiendo a estas tecnologías resolver problemas cotidianos de manera eficiente y segura. Abordando los diferentes desafíos y limitaciones, el proyecto no solo contribuye al campo de la robótica, sino que también prepara el terreno para futuras innovaciones y trabajos de investigación relacionados con este modelo de robots.

1.1. Objetivos

El objetivo principal de este trabajo de fin de grado consiste en brindar a un robot con cinemática Ackermann de la capacidad de navegar de manera autónoma utilizando el sistema operativo ROS2 con el apoyo de un sensor láser.

Objetivos Específicos:

- Realización de un modelo virtual en la plataforma de simulación CoppeliaRobotics, el cual refleje con precisión las características físicas y cinemáticas del robot real.
- Integración de un sensor hokuyo en el modelo simulado y el robot real asegurando la captación correcta de datos y su correspondiente procesado de información del entorno de manera eficiente.
- Desarrollo de una navegación reactiva a través del stack de navegación Nav2, el cual permitirá la creación y seguimiento de trayectorias.
- Desarrollo de un algoritmo para la realización de maniobras de aparcamiento en paralelo incluyendo la localización del espacio libre en un recorrido autónomo.

1.2. Metodología de Trabajo

Para llevar a cabo el desarrollo del proyecto se seguirá el siguiente plan de trabajo:

- **Estudio del modelo cinemático Ackermann:** revisión extensa de la documentación existente sobre este modelo de dirección incluyendo principios básicos, aplicaciones y ventajas.
- **Conexión física del Robot:** configuración de la conexión a través del puerto CAN del robot con el sistema de control integrando todos los componentes de hardware. Creación de un control básico por teclado para corroborar el correcto procesado de información.
- **Integración del modelo URDF e implementación en CoppeliaRobotics:** construcción de un modelo virtual y detallado del robot y ajustar los parámetros correspondientes para que el comportamiento simulado sea lo más cercano posible al real. Su utilidad se basa en las diferentes pruebas en simulación que se pueden hacer de algoritmos de navegación antes de usarlo en el robot real evitando accidentes.
- **Instalación de un sensor láser:** integración un sensor hokuyo en el modelo real y el virtual garantizando datos precisos y una correcta comunicación con ROS2 desde ambos modelos.
- **Uso del Stack de navegación Nav2:** utilización del stack para generar trayectorias óptimas basadas en la información proporcionada por el sensor y el mapa del entorno, así como el seguimiento de dicha trayectoria a partir de parámetros internos de Nav2.
- **Realización de algoritmos de aparcamiento:** diseño e implementación de algoritmo para maniobras de aparcamiento autónomo validando su eficacia a través del entorno simulado en diferentes condiciones.
- **Creación de mapas en escenas simuladas:** diseño de diversas escenas en CoppeliaRobotics y creación de sus correspondientes mapas con la herramienta SLAM toolbox simulando entornos exteriores con diversos obstáculos y un ambiente de aparcamiento.
- **Pruebas finales y resultados:** pruebas en el modelo virtual evaluando el rendimiento y precisión de todos los algoritmos, ajustando diferentes parámetros y mejorar su consistencia y desempeño. Finalmente, documentación de resultados obtenidos y proporcionar un análisis general del sistema.

1.3. Estructura de la Memoria

Este trabajo presenta una estructura de 5 capítulos, así como una bibliografía:

Capítulo 2. Herramientas utilizadas: listado de las herramientas y plataformas usadas en el desarrollo del trabajo, así como una breve información de cada una y su utilidad dentro del contexto de este TFG.

Capítulo 3. Marco Teórico: toda la información necesaria para la correcta compresión del modelo cinemático Ackermann.

Capítulo 4. Desarrollo y Resolución de los objetivos: descripción exhaustiva del desarrollo de los diferentes algoritmos, así como la creación del sistema simulado y su implementación.

Capítulo 5. Resultados: recopilación de resultados de las pruebas finales, así como un análisis y sugerencia para futuros proyectos de investigación relacionados con el robot.

2. Herramientas utilizadas

La mayoría de las pruebas y resultados finales del proyecto han sido obtenidos del modelo simulado, por los que las herramientas virtuales son de gran relevancia para la experiencia del trabajo. En este capítulo se detallarán estas herramientas, así como sus diferentes usos y sus estructuras internas con la finalidad de entrar en contexto con el desarrollo de los objetivos.

Estas herramientas son ROS2 y CoppeliaRobotics. El primero representa la estructura de comunicación entre los diferentes componentes del sistema como son el propio robot, Nav2 o Coppelia y herramientas tales como RViz2 para la visualización de datos. Por otro lado, el segundo es la plataforma de simulación en la que se creará el ambiente virtual del robot y diferentes escenarios de trabajo.

Adicionalmente, se explicarán algunos algoritmos en ROS2 que han permitido el desarrollo del proyecto, los cuales son Nav2, stack de navegación adaptable a cualquier modelo robótico, y SLAM, herramienta usada para la lectura de mapas e imprescindible para el uso de la herramienta anterior.

2.1. ROS2

El sistema operativo para robots (ROS) comenzó como una vía de unificación universal en el área de la robótica ganando atención de parte de los desarrolladores gracias a su alta flexibilidad para cualquier utilidad. Su última generación, ROS2, llega con la finalidad de mejorar deficiencias en términos de eficiencia, rendimiento y seguridad. Ambas versiones tienen la misma estructura, la cual es visible en la Figura 1 y se compone de una comunicación basada en nodos, los cuales se conectan a través de canales denominados *topics* por los que se envía información. A través de estos nodos, el usuario puede “publicar” datos, o se puede “suscribir” a estos para recibirlos [5]. Este sistema ha servido como plataforma para la conexión entre dispositivos, así como interpretación de datos ya sea desde la información recibida por el sensor de rango, la localización estimada del robot o la información devuelta del entorno virtual.

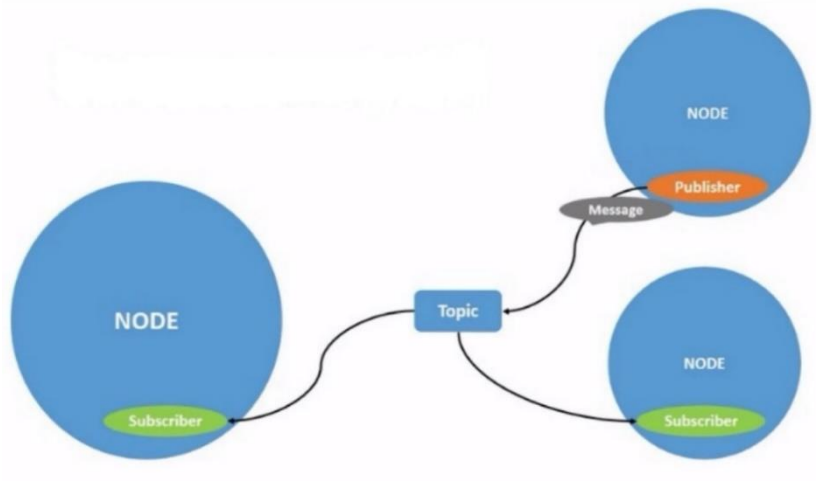

Figura 1. Estructura de Nodos en ROS2 [6].

2.2. RViz

RViz es una herramienta de simulación usada mayormente para visualizar los datos entregados, ya sea de parte de los sensores como de los topics, siendo RViz2 la versión compatible con ROS2. Este programa es capaz de interpretar la información recibida y transformarla en mapas, de dos dimensiones en el caso de este proyecto, los cuales son superpuestos sobre el mapa original permitiendo comparar informaciones con el mundo real [10]. Su uso se centra en la interpretación visual de la información del sensor de rango usado en cada escenario, así como la comprobación en tiempo real de la localización del robot y correcto funcionamiento de los datos publicados en los topics.

2.3. Coppelia Robotics

CoppeliaRobotics es un entorno virtual usado para la simulación en el marco de la robótica siendo útil por su variedad de modelos como robots, vehículos y objetos de diferentes ambientes, así como elementos geométricos sencillos. Esto lo convierte en la plataforma perfecta para la representación de escenarios realistas en ambiente simulado copiando características físicas del mundo real. Adicionalmente, es capaz de procesar información en tiempo real, ya sea de información proveniente del mundo real como de algoritmos externos, como los creados en ROS2 [8][9]. Esta herramienta fue aprovechada para recrear diferentes escenarios donde poder comprobar la efectividad de los algoritmos escritos en ROS2, así como plataforma para el desarrollo del modelo virtual del robot Hunter 2.0, el cual se aprecia en la Figura 2.

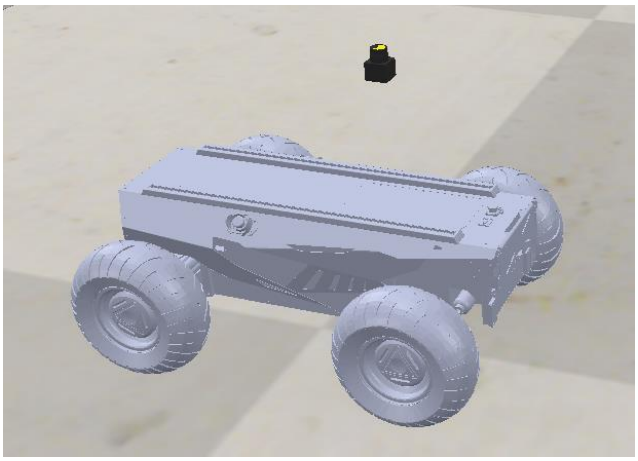

Figura 2. Modelo Virtual en CoppeliaRobotics.

2.4. Nav2

La navegación autónoma de cualquier tipo de robot siempre ha implicado el uso de algoritmos altamente complejos que usan bases de movimiento y cálculo de trayectorias que se adapten apropiadamente a cualquier arquitectura del modelo robótico usado. Es por esto por lo que se han creado stacks de navegación como es Nav2, los cuales poseen estructuras complejas que aseguran una navegación adecuada y segura en una gran variedad de situaciones. Como se aprecia en la Figura 3, tiene una estructura basada en comunicaciones entre subsistemas encargados de diferentes tareas más sencillas [24].

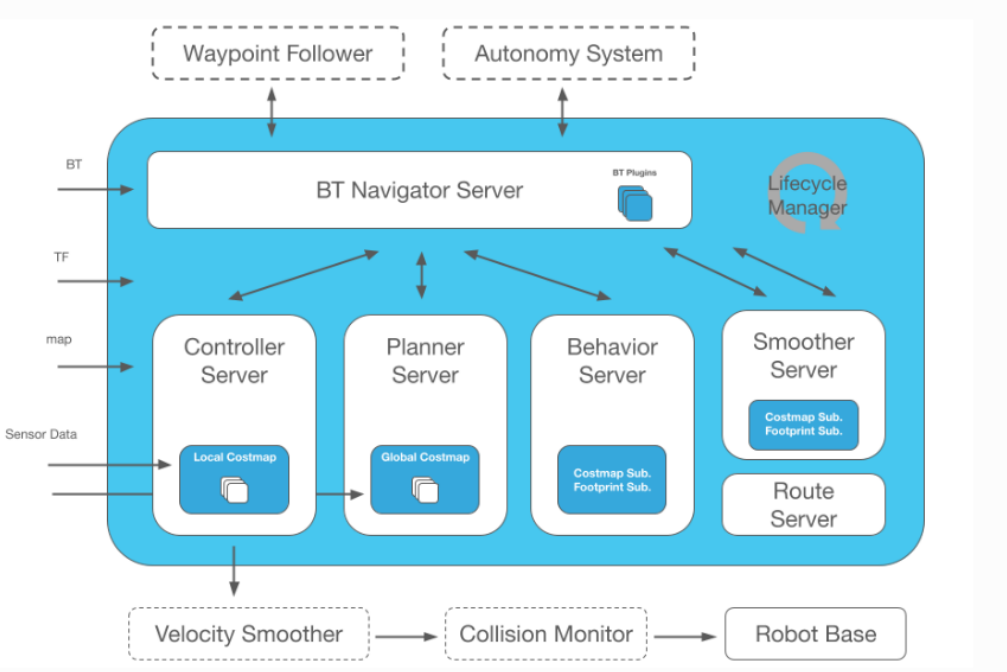

Figura 3. Estructura del Stack de Navegación Nav2 [24].

Entre sus aplicaciones se encuentra el cálculo de trayectorias basado en el modelo de robot usado, el cual toma en cuenta todas las características y limitaciones para diseñar un camino realista que el sistema puede seguir apropiadamente. Una vez hecho esto, el algoritmo de control se encargará de navegar al robot con las velocidades apropiadas hasta la meta preestablecida. Durante todo este proceso, el sistema debe recibir información de su entorno a través de un sensor dado que tiene la capacidad de recalcular su trayectoria en caso de que haya cambios en su entorno o si el robot no sigue correctamente el camino [24].

Su estructura se compone de la siguiente forma:

- La estructura general está basada en el uso de un ***Behavior Tree***, es decir, uso de subsistemas o nodos encargados de realizar tareas específicas facilitando el cálculo de acciones a tomar en cada ciclo y reduciendo el tiempo de cálculo. En este caso, las diferentes ramificaciones están basadas en necesidades básicas de la navegación como seguimiento del robot, cálculo de caminos o detectar obstáculos [25].
- Debido a la existencia de una gran cantidad de nodos trabajando al mismo tiempo es necesario el uso del ***Lifecycle Manager***, el cual se encarga de organizar por orden de llegada o importancia a los nodos que se encuentren activos y quita de la lista aquellos que no respondan o generen fallos evitando retrasos en el sistema [25].
- Para la navegación es necesario conocer donde se encuentra el robot, así como información de su entorno. Para esto se usa el ***Map Server***, el cual se encarga de cargar y filtrar el mapa del entorno a través de un archivo .yaml para futuros cálculos y tener una correcta referencia de la posición del robot [27].
- Una vez recibido el mapa es necesario establecer la posición del robot, así como estimar su localización durante el movimiento. Para ello existe ***AMCL***, siglas del método adaptativo de localización Monte Carlo en español, el cual consiste en la creación de un filtro de partículas en la que cada punto está relacionado con una posible posición en cada iteración permitiendo estimar la posición más probable en la que se encontrará el robot [28]. Para su uso es necesario de mapas creados con archivos .yaml preferiblemente creados con SLAM toolbox [25].
- Para la creación de la trayectoria se usa un ***Planner Server***, el cual usa información global tanto del mapa como del sensor para definir un camino entre el punto inicial y la meta. Estos caminos pueden ser elegidos en base a la distancia recorrida y maniobras necesarias, siempre teniendo en cuenta las capacidades del modelo y puede elegirse el trayecto más corto o un recorrido a través de una serie de localizaciones intermedias. Además, es posible definir la distancia mínima entre la trayectoria y los obstáculos a través de un mapa llamado *global costmap* el cual define las zonas de riesgo de colisión de todo el mapa [26].

- En muchos casos, los cálculos de optimización de rutas pueden generar giros bruscos que no cualquier robot es capaz de hacer y que en la realidad le tomaría al robot más tiempo para recorrerlo. Es por esto por lo que se usa el ***Smoother Server***, el cual se encarga de suavizar estos giros y hacerlos alcanzables para cualquier tipo de robot. El hecho de tenerlos en servidores diferentes puede resultar ventajoso ya que permite el uso de variedad de planificadores o la selección de tramos en la ruta para suavizarlo [25].
- Una vez construido el plan global es necesario poner en marcha el robot, para lo cual se usa el ***Control Server***. Este, como su nombre indica, controla el movimiento del robot y se asegura que se siga correctamente la ruta calculada. Además, se encarga de modificar localmente el trayecto en caso de obstáculos imprevistos o un mal seguimiento de esta. Para esto se usa el *local costmap*, que calcula las zonas de riesgo en el área cercana al robot [25].

2.5. SLAM toolbox

La creación de mapas en diferentes entornos representa una alta complejidad debido a todos los componentes y recursos utilizados para este fin. Debido a esto se ha creado SLAM toolbox, una herramienta de ROS2 que integra todos los recursos informáticos para facilitar esta tarea. Este paquete integra información recibida por sensores láser y transformaciones TF de la localización del robot para crear mapas 2D [29], como el que se aprecia en la Figura 4.

Entre su gran variedad de usos, se destacan:

- Uso sencillo del paquete para iniciar, mapear y guardar datos.
- Modificación de mapas ya creados.
- Mapeo continuo del mapa sobre datos ya existentes.
- Conexión con la plataforma RViz2 para la visualización del mapa y movimiento del robot en un ambiente virtual.

Su principal funcionamiento está basado en la comunicación de nodos de ROS2, en la que se suscribe al topic del mensaje LaserScan del sensor y la posición odométrica del robot. Este paquete se encarga de usar esta pose para crear una correlación con la información recibida del sensor y estimar la distancia entre los objetos leídos con el sistema local.

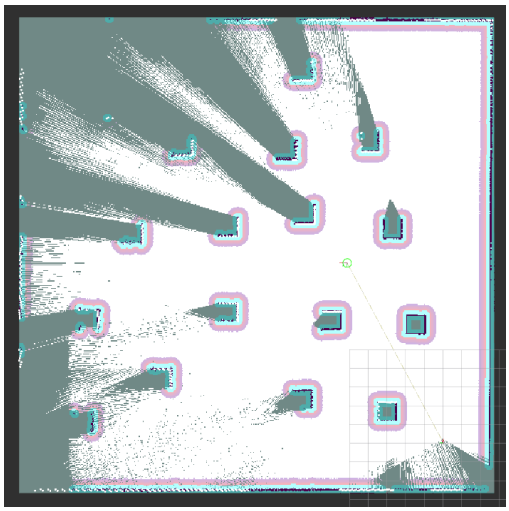

Figura 4. Mapeo de una escena de CoppeliaRobotics con el paquete SLAM.

3. Marco Teórico

Muchos de los resultados de las experiencias se deben sobre todo a conocimientos teóricos necesarios para alcanzar los diferentes objetivos del trabajo. El principal ejemplo es la base teórica del modelo cinemático de Ackermann, la cual es relevante para la comprensión de su estructura. En este capítulo se comienza detallando el conocido modelo de la bicicleta, una versión simplificada de la configuración Ackermann que permite comprender el funcionamiento básico de las ruedas del robot y visualizar los movimientos posibles por este. Luego, se pasará a comprender el modelo real del robot basado en la versión anterior y se comprenderá la complejidad y limitaciones de estos vehículos. Finalmente se pasa a explicar las ecuaciones más relevantes para el trabajo y que serán usadas más adelante.

3.1. Cinemática Ackermann

El modelo cinemático Ackermann representa una base sólida para el diseño de vehículos que deben operar en entornos restringidos y seguir trayectorias curvas con alta precisión. Esta configuración parte del modelo ilustrado en la Figura 5 el cual se caracteriza por su similitud a la estructura de una bicicleta y que parte de la simetría de los vehículos de cuatro ruedas. En esta simplificación, la rueda trasera funciona como fuerza motriz del sistema impulsando el vehículo en dirección lineal sin ser capaz de rotar en el eje Z. La rueda delantera, o directriz, es la que se encarga de orientar el movimiento girando en función del radio de giro deseado y es potenciada por el movimiento de la rueda motriz. De esta forma se observa que el sistema parte de una velocidad *v*, con un ángulo de giro θ y centro de giro P.

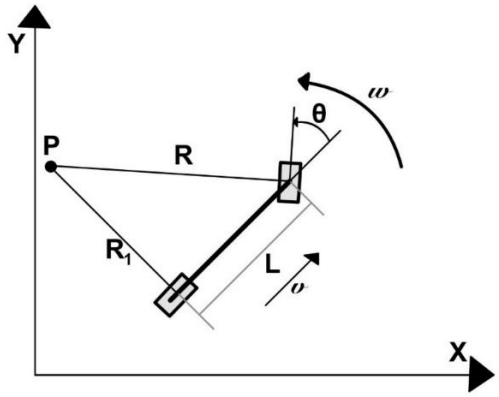

Figura 5. Modelo Simplificado de la cinemática Ackermann [14].

El modelo real es la versión completa y más compleja del anterior. Su morfología es apreciable en la Figura 6 y consiste en un vehículo de cuatro ruedas similar a un automóvil común. Por un lado, dos se encuentran en la sección delantera del sistema equivalente a

la rueda directriz del modelo simplificado, cuya orientación es modificada por la dirección de movimiento de tal forma que, a bajas velocidades, todas las ruedas cumplen con la condición de rodamiento puro sin deslizamiento lateral. Por otro lado, posee dos ruedas traseras no orientables cuyos ejes de rotación son paralelos al de la rueda motriz del modelo anterior. Este eje resultante entre las ruedas es perpendicular al eje del movimiento lineal del sistema.

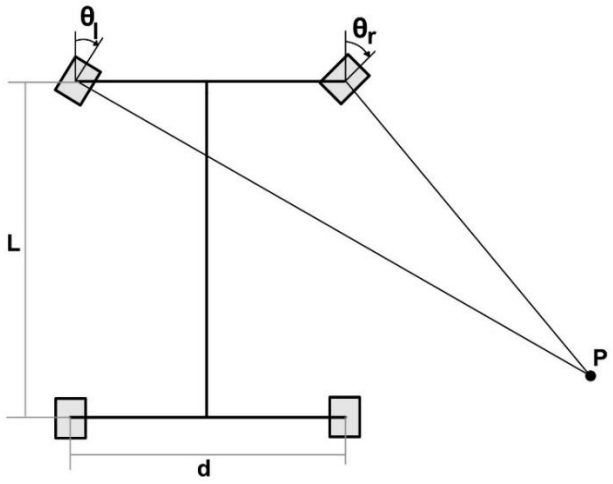

Figura 6. Modelo Cinemático de Ackermann [11].

Un problema intuitivo que se puede presentar en el diseño de esta configuración es que, si las ruedas directrices (delanteras) giraran un mismo ángulo, estas presentarían mismo radio, pero un centro de giro distinto con una distancia igual a la de las ruedas. Esto puede ocasionar deslizamiento en las ruedas, el cual aumentaría en función de la velocidad, lo cual puede ocasionar un accidente, así como presentar un aumento en el desgaste de las ruedas. Para solucionar este riesgo, ambas ruedas deben presentar el mismo centro de giro y, por ende, tener diferentes ángulos. Esto se consigue con el denominado mecanismo de dirección de Ackermann [12], el cual es ilustrado en la Figura 7, en la que un sistema trapezoidal está compuesto por cuatro barras conectadas a las ruedas directrices asegurando un centro común cuyo radio de giro es medido entre este punto y el centro del eje de las ruedas motrices (traseras).

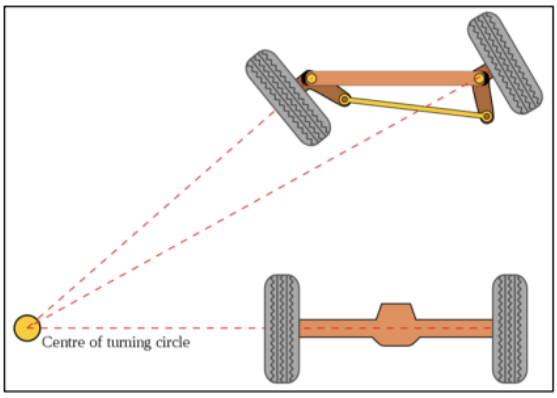

Figura 7. Giro de ruedas delanteras con mismo centro de giro [12].

Una consecuencia directa de este modelo es su naturaleza no holonómica, es decir, que presenta restricciones de movimiento dependientes a las velocidades limitando las trayectorias que puede seguir [15]. En el caso de vehículos tipo Ackermann, estos no pueden realizar movimientos laterales sin cambiar su orientación y posición, así como imposibilitar giros sobre su propio eje. Adicionalmente, estos sistemas incluyen ángulos máximos de giro menores a los noventa grados debido a la estructura física de estos, por lo que dificulta las maniobras de corta distancia [14].

3.1.1. Ecuaciones cinemáticas

Tanto el modelo simplificado como el de cuatro ruedas deben cumplir con ecuaciones cinemáticas de las que no solo se define su comportamiento sino sus limitaciones. En el caso del modelo simplificado, las ecuaciones son más intuitivas y de estas parten las ecuaciones del modelo real. Partiendo de la Figura 2, se entiende *v* como la velocidad lineal del sistema, mientras que *w* es la velocidad angular. Considerando R como el radio de giro, obtenemos la primera ecuación:

$$\omega = \frac{v}{R} \tag{1}$$

Adicionalmente, existe una relación entre este radio de giro y el ángulo de giro (θ) que forma la rueda delantera. Para ello se usa L, la distancia entre los ejes, lo cual devuelve la siguiente ecuación:

$$\tan\theta = \frac{L}{R} \tag{2}$$

A partir de ambas ecuaciones se crea una relación entre las velocidades del sistema y este ángulo de dirección. Despejando R de la ecuación (1) y sustituyendo en la ecuación (2) se obtiene lo siguiente:

$$\theta = \tan^{-1}(\frac{L\omega}{v}) \tag{3}$$

Con estas ecuaciones es posible detallar el modelo real del sistema. Usando la Figura 3, se entiende que el radio de giro de cada rueda directriz es diferente, por lo que se deben calcular a partir de este valor. Sabiendo que R_1 es el radio de uno de los lados y d es la distancia entre ambas ruedas directrices, la relación directa es la siguiente:

$$R_1 = R \pm \sin\beta * \frac{d}{2} \tag{4}$$

Donde β es la diferencia de ángulos entre ambos puntos. Sin embargo, considerando que la mitad de la distancia entre las ruedas (d) es un valor considerablemente pequeño, el

seno de este ángulo puede ser considerado igual a uno, por lo que ambas relaciones quedarían de la siguiente forma:

$$R_{left} = R - \frac{d}{2} \tag{5}$$

$$R_{right} = R + \frac{d}{2} \tag{6}$$

Teniendo en consideración ambas ecuaciones, se pueden crear analogías para la ecuación (2), quedando de la siguiente manera:

$$\tan\theta_{left} = \frac{L}{R_{left}} \tag{7}$$

$$\tan\theta_{right} = \frac{L}{R_{right}} \tag{8}$$

Teniendo en cuenta estos datos, es posible obtener los ángulos de giro de cada rueda. Despejando cada ángulo de las ecuaciones (7) y (8), sí como sustituir R_{left} y R_{right} respectivamente:

$$\theta_{left} = \tan^{-1}\left(\frac{L}{R - \frac{d}{2}}\right) \tag{9}$$

$$\theta_{right} = \tan^{-1}\left(\frac{L}{R + \frac{d}{2}}\right) \tag{10}$$

Finalmente, sustituyendo R de la ecuación, se obtienen las ecuaciones finales:

$$\theta_{left} = \tan^{-1}\left(\frac{L}{\frac{L}{\tan\theta} - \frac{d}{2}}\right) \tag{11}$$

$$\theta_{right} = \tan^{-1}\left(\frac{L}{\frac{L}{\tan\theta} + \frac{d}{2}}\right) \tag{12}$$

Con respecto a la ecuación (1), esta relación únicamente es aplicable cuando ambas ruedas directrices tienen el mismo ángulo de giro, es decir, cuando la velocidad angular es nula. En caso contrario, se obtendría la siguiente ecuación:

$$v_1 = \frac{\omega * (R \pm \frac{d}{2})}{r_r} \quad (13)$$

Donde r_r es el radio de las ruedas del robot y la v_1 la velocidad lineal de cada rueda. Para el contexto del trabajo puede resultar relevante mencionar las ecuaciones en coordenadas polares. En la Figura 8 podemos destacar el vector distancia δ que une el centro de la rueda motriz con el centro del marco global con su respectiva orientación ϕ con respecto al original y un ángulo α con respecto al eje principal del vehículo. De esta forma obtenemos:

$$\dot{\delta} = -v * \cos\alpha\,, \qquad -\theta_m \leq \theta \leq \theta_m \quad (14)$$

$$\dot{\alpha} = -\frac{v}{L}\tan\alpha + \frac{v}{\delta}\sin\alpha, \qquad -\pi \leq \alpha \leq \pi \quad (15)$$

$$\dot{\phi} = \frac{v}{\delta}\sin\alpha\,, |v| \leq v_m, \qquad -\pi < \phi \leq \pi \quad (16)$$

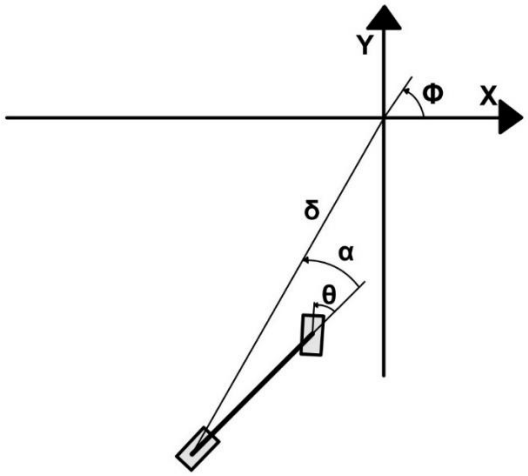

Figura 8. Giro en Coordenadas Polares del Modelo Simplificado [14].

Debido a las restricciones de giro del modelo, existen zonas no alcanzables derivadas de las limitaciones de movimiento y dependientes del ángulo de giro máximo. Asumiendo que este valor es igual en ambas direcciones, en la Figura 9 podemos visualizar los círculos no alcanzables. Las circunferencias externas corresponden a la trayectoria de la rueda directriz. El cálculo de estos radios parte de las ecuaciones (7) y (8).

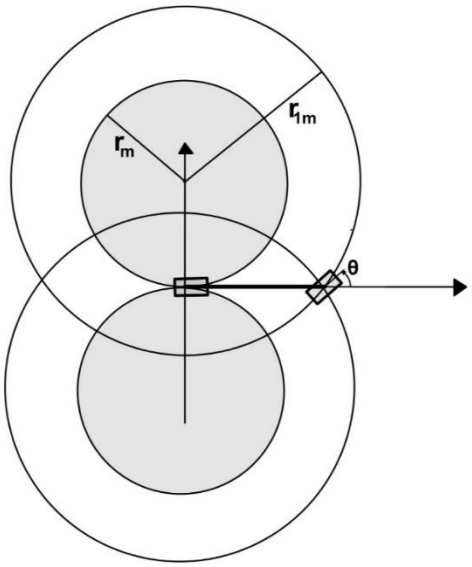

Figura 9. Región no Alcanzable por el Modelo.

$$r_m = \frac{L}{\tan\theta_m}, \qquad r_{1m} = \frac{L}{\sin\theta_m} \tag{17}$$

4. Desarrollo y resolución de los objetivos

A continuación, se detallará el proceso de resolución de objetivos para este proyecto. Como se comentó anteriormente, la meta general es la de brindar a un robot de cuatro ruedas de navegación autónoma. Esto incluye: obtención de datos del entorno, planificación y seguimiento de una ruta en función de esta información externa, capacidad para recalcular la trayectoria en caso de imprevistos y capacidad para aparcar en ambientes preestablecidos.

Adicionalmente, se enlistarán todos los problemas relevantes que se presentaron durante la realización de todas estas actividades, así como sus correspondientes soluciones, con la finalidad de apoyar a futuros usuarios que deseen continuar con el trabajo expuesto.

4.1. Exploración del Hunter 2.0

El modelo usado en este proyecto es un Hunter 2.0 (Figura 10), un robot con configuración Ackermann producido por la compañía Agilex Robotics. Este modelo tiene un diseño especial para entornos exteriores de altas dificultades ya que posee unas ruedas de tamaño y robustez considerables. En su manual de usuario podemos encontrar la siguiente información relevante [19]:

Figura 10. Robot Hunter 2.0.

Con respecto a la seguridad del robot, este no posee integrado ningún sensor de rango ni ninguna fuente de información de su entorno, por lo que su uso debe ser limitado a entornos abiertos, preferiblemente con una distancia de seguridad de dos metros con los obstáculos. Por su cuenta, el robot tiene un peso de entre los 65 y los 70Kg, mientras que su carga no debe superar los 150Kg. Además, se indica que el robot no tiene como propósito el transporte de pasajeros, por lo que de ninguna forma se deben montar personas sobre el vehículo. Adicionalmente, debe ser usado a una temperatura de entre los -10ºC y los 45ºC excepto durante el tiempo de carga, en el cual la temperatura debe estar por encima de los 0ºC.

Existe un puerto de carga, ilustrado en la Figura 11, el cual está localizado en la zona trasera del robot y es etiquetado como puerto Q6. Por otro lado, en la misma imagen encontramos el puerto Q5, cuyos detalles se muestran en la Figura 12. Este posee cuatro pines, uno como salida VCC (pin 1), uno como conexión a tierra GND (pin 2) y dos pines dedicados para conexión CAN (pines 3 y 4) la cual también es posible a través de este mismo puerto. Sin embargo, en la zona superior del robot se encuentra un puerto con las mismas características. El rango de la tensión es de 21-26.8V, mientras que la corriente debe ser inferior a 15 amperios.

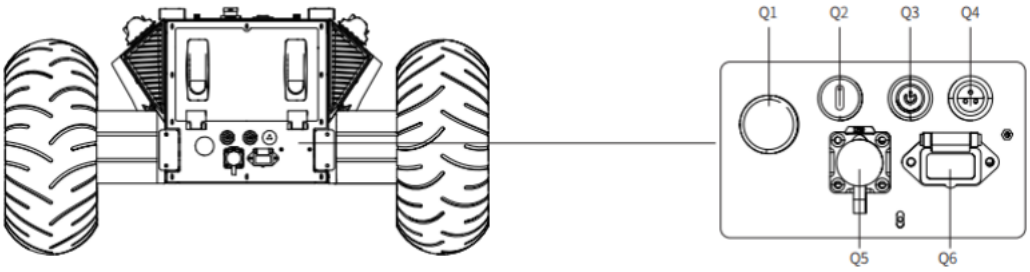

Figura 11. Módulo de Puertos traseros del robot Hunter 2.0 [19].

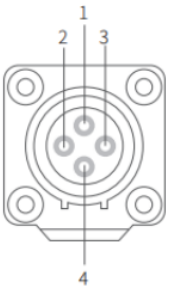

Figura 12. Puerto Q5 de Comunicación CAN y Fuente de Alimentación [19].

El robot tiene unas dimensiones considerables (Figura 13): de largo posee unos 980mm, mientras que su ancho es de 745mm incluyendo las ruedas. Un dato relevante durante el proyecto es la distancia entre los ejes de las ruedas, la cual es de 760mm. Igualmente

importante es el diámetro de las ruedas de 330mm y una distancia entre las ruedas izquierdas y derechas de 605mm.

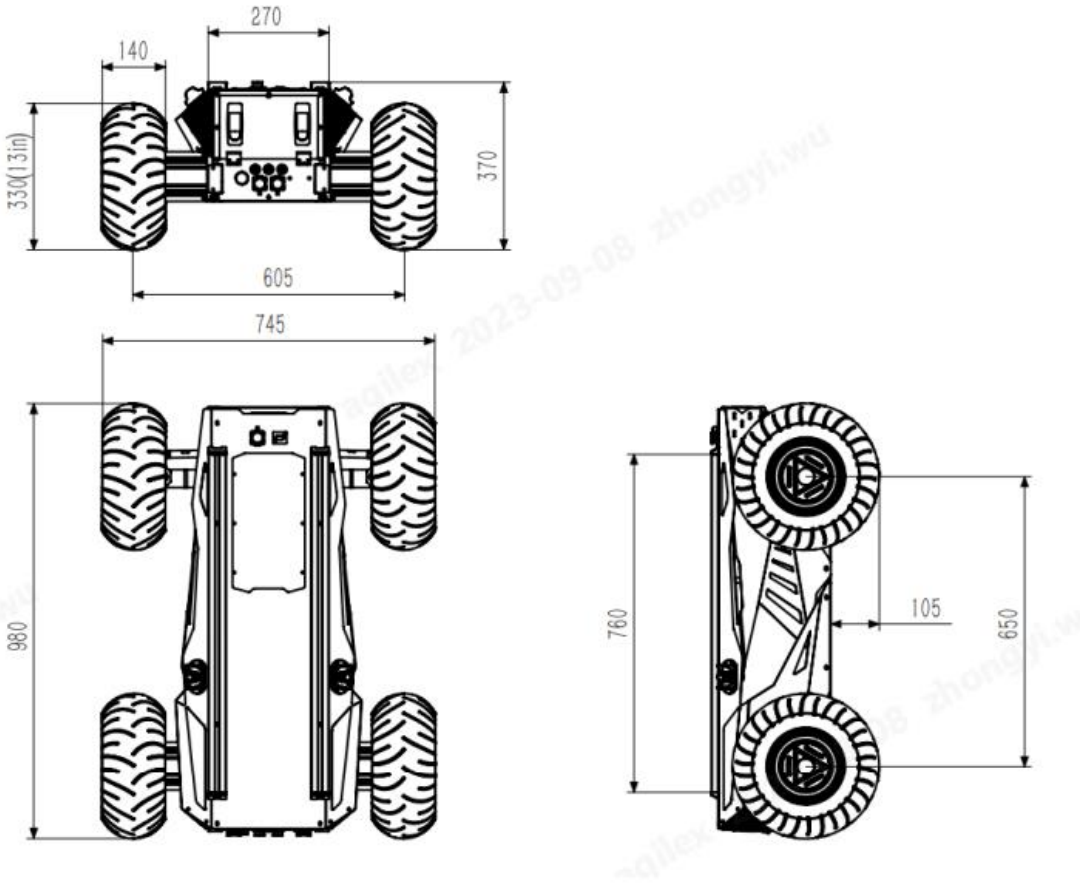

Figura 13. Dimensiones del Robot Hunter 2.0 [19].

Para el control del robot existen dos opciones: a través de comandos por el puerto CAN y con una conexión inalámbrica del mando del robot (Figura 14). Una vez iniciado el robot se deben presionar ambos botones POWER para iniciar el mando. Por razones de seguridad, todos los interruptores superiores deben encontrarse en su punto neutro para que el mando inicialice correctamente. La palanca S2 se encarga del control de las ruedas directrices, las cuales dictaminan el ángulo de giro del robot. Si no se usa el robot se moverá en línea recta. La palanca S1 se usa para determinar la velocidad de las ruedas motrices. Para usar estas palancas el interruptor SWB debe estar fuera de su punto neutro, ya sea colocándolo en el punto superior o inferior. Si este interruptor se encuentra en el punto neutro el robot no se moverá y seguirá las instrucciones enviadas por comando por el puerto CAN. Otros interruptores y botones incluidos no serán usados durante el proyecto.

Figura 14. Mando a Distancia del Robot Hunter 2.0 [19].

La empresa Agilex Robotics ofrece todos los paquetes necesarios para el uso por comando del robot. Para ello se requieren de dos paquetes: "ugv_sdk" [20] y "hunter_ros2" [21], ambos compatibles con ROS2. Por un lado, el primer paquete se encarga de la conexión entre el puerto CAN del robot con el USB del ordenador, conexión física ilustrada en la Figura 15. Esta conexión sirve para recibir información del estado general del robot. El segundo paquete sirve como puente entre las aplicaciones de ROS2 y la conexión con este puerto CAN, es decir, transforma la información de los topics en mensajes codificados en C++, los cuales el paquete "ugv_sdk" es capaz de interpretar y convertirlos en un formato apropiado para el puerto CAN. Esta estructura de comunicación se puede visualizar en la Figura 16.

Figura 15. Conexión al bus CAN y fuente de alimentación del Robot Hunter 2.0 [19].

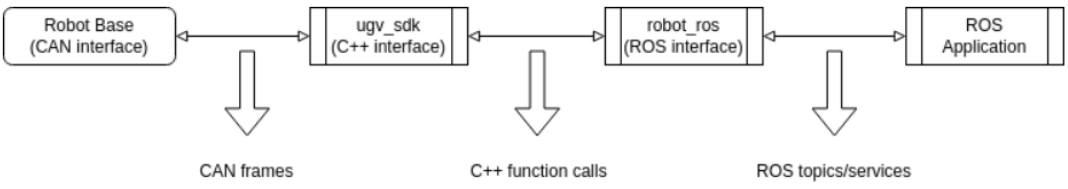

Figura 16. Estructura de Comunicación entre Paquetes de ROS2 con el puerto CAN [20].

Para la comunicación a través de ROS2 se debe de enviar un mensaje de tipo *Twist* a través del topic "cmd_vel". La estructura de este mensaje es fácilmente explicable y análogo a la forma en la que se controla el robot a través del mando. Como se comentó anteriormente, la palanca S2 controla el ángulo de las ruedas directrices, el cual se puede controlar en función de la velocidad angular del robot: a mayor velocidad angular, mayor

ángulo de inclinación y, por ende, mayor radio. Por otro lado, la palanca S1 es responsable de la velocidad de las ruedas motrices, la cual, se puede decir, es la velocidad lineal del sistema. Dentro de la estructura del mensaje encontramos una subestructura llamada *position*, la cual posee tres valores *float*: *x, y, z,* los cuales se pueden interpretar como velocidades lineales en los diferentes ejes, ilustrados en la Figura 17. La velocidad lineal usada se corresponde al componente *x*. Por otro lado, se tiene la subestructura *orientation*, el cual posee los mismos valores *x, y, z* representando velocidades angulares en el mismo sistema de referencia. La velocidad angular usada corresponde al eje *z*. Los otros cuatro valores en ambas subestructuras del mensaje no son utilizados y, por ende, ignoradas por el robot. Las ecuaciones y una explicación más detalladas de estas relaciones se pueden encontrar en el apartado Planificación en el que se replica en el entorno virtual.

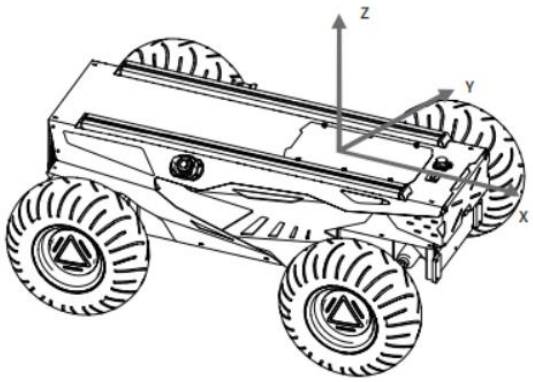

Figura 17. Sistema de Referencia Local del robot Hunter 2.0 [19].

Para el primer contacto y experiencia con el robot real se ha creado un script en ROS2 basado en un control con las flechas de un teclado. El comportamiento es intuitivo: la flecha superior suma 0.1m/s a la velocidad lineal, mientras que la inferior resta el mismo valor. La flecha izquierda suma el valor anterior a la velocidad angular y la derecha lo resta. El ángulo máximo de giro en cada rueda del robot es de 0.573577rad. Para este contacto inicial se ha usado el robot dentro del laboratorio 2.3.6, E.T.S.I Informática, Universidad de Málaga (mapa disponible en el apartado Creación), por lo que se ha ajustado una velocidad lineal máxima de 0,6m/s.

4.2. Instalación del Sensor Hokuyo

Los sensores de rango láser son altamente usados en la navegación y generación de mapas debido a su alta precisión y flexibilidad. Estos emiten un haz de luz enfocado en línea recta a través de un transmisor y es recibido por un elemento receptor de luz. En muchos casos, ambos elementos pueden estar incluidos dentro de un mismo sensor (modelo reflectivo), mientras que en otros estos están físicamente separados en dos componentes (modelo de barrera) [13]. Para el caso de sensores de rango es común encontrar estos en el mismo componente ya que el emisor rota dentro del sensor y emite luz en un intervalo.

Esta tecnología de escaneo es bastante sencilla al ser capaz de detectar únicamente la distancia que hay entre un objeto y el emisor, pero ser bastante útil para el cálculo de superficies y mapeo de espacios en 2D y 3D. La distancia es medida a través del tiempo que el láser infrarrojo tarda en regresar al receptor, cuya velocidad es conocida por el sensor [18].

El sensor disponible para el proyecto es un modelo hokuyo UTM-30LX, disponible en el laboratorio mencionado en el apartado anterior. Este es un sensor de rango con un alcance de 270º con una distancia de entre los 0.1m y los 30m. Es, según la página oficial de Hokuyo Automatics, apropiado para robots de alta velocidad de movimiento debido a su gran rango de alcance y su rapidez de respuesta. Esto se comprueba al analizar sus especificaciones, las cuales indican que es capaz de tomar datos cada veinticinco milisegundos. Debido a su baja potencia, sin embargo, es altamente recomendado únicamente usarlo en interiores, y representa el impedimento de que no devuelve información de la zona trasera, lo cual puede llegar a ser información valiosa a la hora de maniobrar con un robot de altas dimensiones y versatilidad de giro limitado como es el caso del Hunter 2.0 [22].

Para su utilización es necesario de tres paquetes de ROS2 creados por la empresa productora de este sensor, los cuales son: “urg_node”, “urg_c” y “urg_node_msgs”. Estos paquetes forman las bases para la comunicación con el sensor a través de scripts y funciones preestablecidas. Para su conexión física, el sensor tiene conectados dos cables: uno funciona como fuente de alimentación, el cual va conectado a los pines uno y dos del puerto Q5, los cuales se muestran en la Figura 20 (Entre estas conexiones se usa un regulador de tensión para convertir los 24V en 12V, apropiados para el componente), el otro cable es USB y va conectado directamente al ordenador. Para establecer la conexión apropiadamente se deben establecer los parámetros, los cuales han sido elegidos en función de los datos por defecto del robot. Los más relevantes son:

- El frame_id con el que se identificará el sensor es “laser_link”
- El topic por el que publica se llama “scan”
- El identificador del puerto serie es “/dev/ttyACM0”

Una vez terminado de definir los parámetros se pasa a crear un archivo launch.py que permite ejecutar fácilmente el nodo del sensor. Entre otras cosas, se encarga de establecer una configuración para los nodos del programa. En este caso, servirá para organizar el topic del sensor “/hunter/scan”, el cual contiene un mensaje de tipo “LaserScan” con la siguiente estructura:

- header
    - time stamp
        - sec
        - nanosec

    - string frame_id
- float32[] angle_min
- float32[] angle_max
- float32[] angle_increment
- float32[] time_increment
- float32[] scan_time
- float32[] range_min
- float32[] range_max
- float32[] ranges
- float32[] intensities

Para comprender estos datos se observa la Figura 18, en la que se ilustra el rango de alcance del sensor con respecto a la disposición del robot. Del lado derecho se encuentra el valor mínimo del ángulo encontrado en el mensaje del sensor, el cual tiene un valor de -2.094rad. En la zona superior central se encuentra el ángulo 0rad, el cual corresponde con el punto central del sensor. En el lado izquierdo, el valor máximo de 2.094rad, y entre cada valor se tiene un incremento de 0.006132rad.

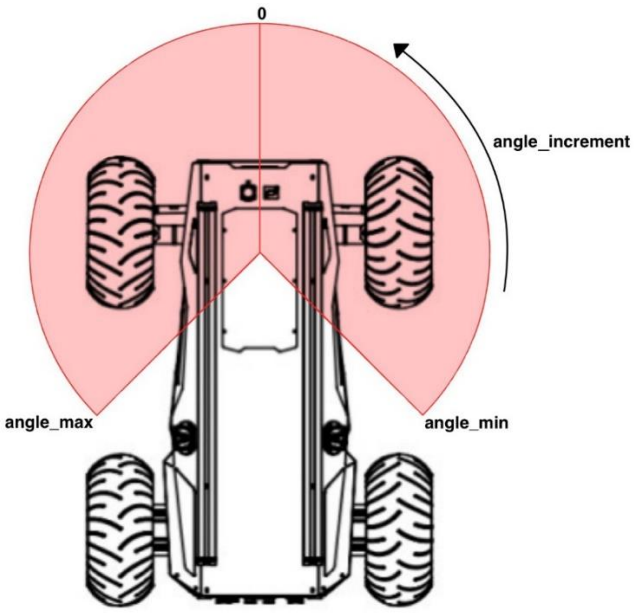

Figura 18. Estructura Espacial de Datos tomados por el Sensor Láser.

Para la implementación sobre el robot se ha colocado, como se puede comprobar en la Figura 10, una plataforma en la que se usó como base para el sensor. Esto no solo permite evitar que el robot aparezca dentro de los datos, sino que también permite sujetarlo con tornillos a la base metálica colocando el componente por encima del sistema de referencia del robot. Interesa sobre todo las coordenadas con respecto a este punto, ya que serán

relevantes para la creación del modelo virtual. Siguiendo el sistema de referencia vista en la Figura 17, las coordenadas locales son: (0.216, 0.00, 0.305)m.

La comprobación de que la lectura de datos es correcta se realiza a través de la plataforma RViz2. Se explicará más adelante la configuración general usada en el proyecto, pero para el sensor laser únicamente se debe seleccionar el topic "/hunter/scan". Para esto, se presiona el botón "add" en la parte inferior izquierda del programa, se selecciona la pestaña "By topic" y se elige el topic antes mencionado. De esta forma, se puede visualizar los resultados. En la Figura 19 se observa la imagen resultante:

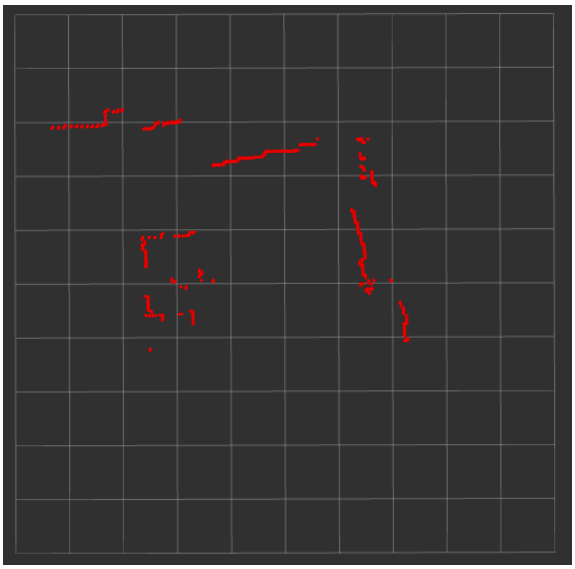

Figura 19. Datos tomados por el Sensor RViz2.

Estos puntos corresponden con las distancias de los objetos que se encuentran alrededor del robot, por lo que se toman estos valores como válidos.

4.3. Desarrollo del Modelo Virtual

Una vez comprobado el comportamiento correcto del sistema real, llega el momento de duplicarlo en un ambiente simulado. Este apartado es, si no, el más relevante del trabajo, ya que servirá como base de experimentación, no solo para el proyecto sino para todos los que vengan en relación con este robot. Se comienza como siempre con una investigación de los materiales ofrecidos por Agilex Robotics. En este caso, los archivos .stl, los cuales corresponden con los modelos 3D de todos los componentes del robot y se localizan en la página de GitHub de la empresa [23].

En la Figura 20 se observa el modelo completo del robot, el cual se puede observar como un mersh en CoppeliaRobotics. A este modelo se le deben agregar las configuraciones físicas apropiadas y los correspondientes scripts para la comunicación con ROS2.

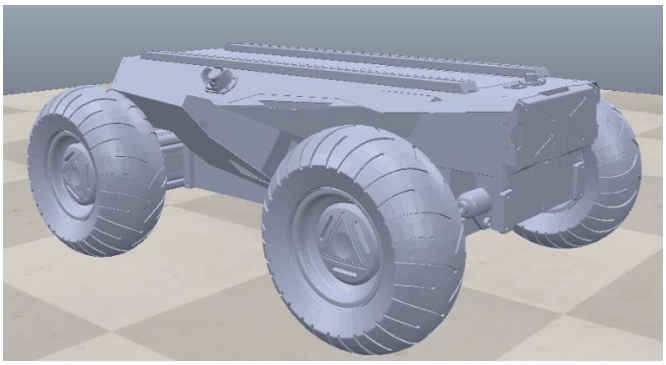

Figura 20. Modelo URDF del robot Hunter 2.0.

Para facilitar la comprensión de las diferentes modificaciones al modelo, se ha agregado la Figura 21, en la que se puede ver su estructura y la relación entre los *frames* y *joints.* Por un lado, Los joints "front_steer_left_joint" y "front_steer_right_joint" corresponden a los puntos de conexión entre el eje y las ruedas delanteras, y se encuentran en el centro de giro de estas, mientras que "front_left_wheel_joint" y "front_right_wheel_joint" son, respectivamente, las ruedas delanteras izquierda y derecha, y se encuentran en la misma posición que sus *parent frames* anteriormente mencionados. Esta separación de ruedas es relevante para el movimiento del vehículo, ya que las primeras dos tendrán control sobre la orientación de las ruedas, mientras que las segundas dos únicamente deben girar libremente en función del movimiento de las ruedas traseras. Por otro lado, se tienen los joints "left_rear_joint" y "right_rear_joint", los cuales corresponden con las ruedas traseras. Dado que estas no necesitan girar en el eje Z, solo es necesario un joint para cada una. Finalmente, están "front_steer_joint" y "rear_wheel_joint" que representan los ejes de giro de las ruedas delanteras y traseras respectivamente.

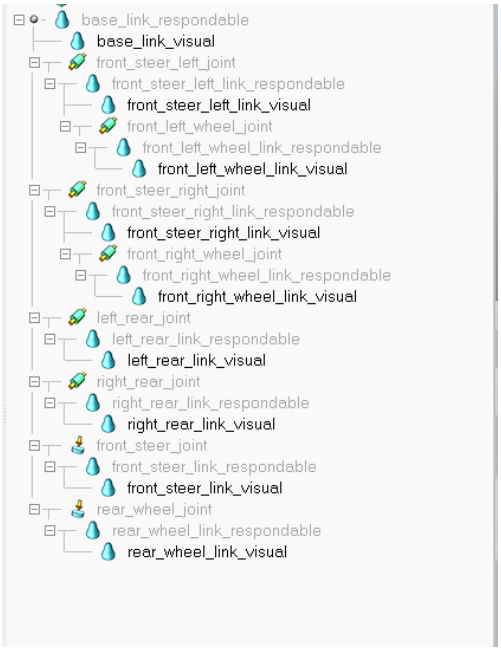

Figura 21. Estructura del modelo URDF del robot, CoppeliaRobotics.

El resto de los elementos de la estructura son únicamente visuales y, por ende, solo representan aspectos visuales del modelo y no afectan las características físicas del modelo. Sin embargo, vale la pena comentar que el archivo .stl del componente "front_steer_left_joint" viene con un fallo en la imagen visual, como se ve en la Figura 22, el cual posee el centro de giro de la rueda trasera derecha por error. Esto no ha sido corregido, ya que no afecta negativamente al modelo y únicamente es apreciable cuando el robot realiza giros.

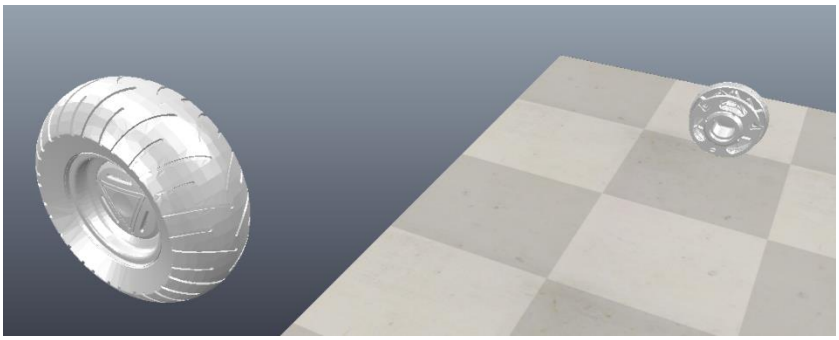

Figura 22. Archivo .STL de la Rueda Delantera Izquierda, CoppeliaRobotics.

Estos modelos son exclusivamente visuales, y aunque ya exista una relación entre los joints, es necesario definir su comportamiento en base a leyes físicas con el fin de facilitar el control del sistema y que el control por los topics de ROS2 sean el mismo al modelo real. En Coppelia, es posible controlar el estado de los joints a través de diferentes características como la fuerza aplicada, la velocidad o la posición, a través de la pestaña "Joint Dinamic Properties" en la ventana de propiedades de cada objeto. En el caso de los joints de las ruedas traseras, estas originalmente se encuentran controladas a través de la posición. Esto implica, por tanto, que se le debe enviar una distancia que debe recorrer en cada iteración. Sin embargo, el control de la cinemática Ackermann y, por ende, la información que se recibe por los topics corresponde con la velocidad lineal del sistema. Esto implica que en cada iteración se debe calcular la posición final de cada joint usando la velocidad recibida y la diferencia de tiempo. Esto naturalmente consume demasiados recursos computacionales y es innecesario. Es por esto por lo que es necesario cambiar este control apropiadamente. En la Figura 23 y Figura 24 se puede visualizar los cambios realizados en ambas ruedas.

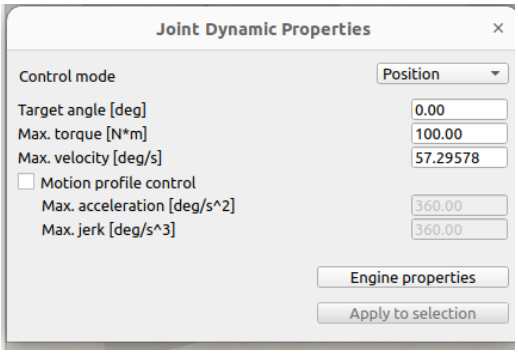

Figura 23. Propiedades Dinámicas Originales de las Ruedas Motrices, CoppeliaRobotics.

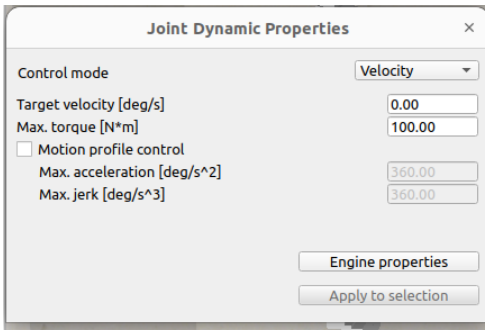

Figura 24. Propiedades Dinámicas Finales de las Ruedas Motrices, CoppeliaRobotics.

Las ruedas delanteras requieren un análisis más complejo, ya que tienen mayor libertad de movimiento. El comportamiento buscado es el siguiente: los joints "front_steer_left_joint" y "front_steer_right_joint" deben rotar las ruedas en el eje Z en función de la orientación que se recibe por el topic. El giro de las ruedas debe ser libre y obedecer el empuje de las ruedas motrices y se verá reflejado por los joints "front_left_wheel_joint" y "front_right_wheel_joint". En el estado original del modelo URDF esto no es alcanzable y se requirieron cambios. Para empezar, la orientación de los primeros dos joints mencionados no son apropiados. La conexión se encuentra horizontal con respecto al cuerpo del robot, lo cual genera que las ruedas giren en el eje Y, pero no que cambien su orientación en el eje Z, por lo que es necesario rotarlos para que realicen giros correctamente.

Para el control de las ruedas directrices es necesario que se haga a través de la posición, dado que la velocidad angular del sistema no se referencia con la velocidad con la que el sistema deba girar, sino con el ángulo de giro que se debe tomar en cada momento. De esta forma, las propiedades dinámicas de ambos joints no deben de cambiar. Sin embargo, es necesario cambiar su configuración, ya que originalmente su posición no era considerada cíclica al corresponder con el giro de las ruedas. Ahora que corresponde con la orientación se debe activar este parámetro, el cual es apreciable en la Figura 25.

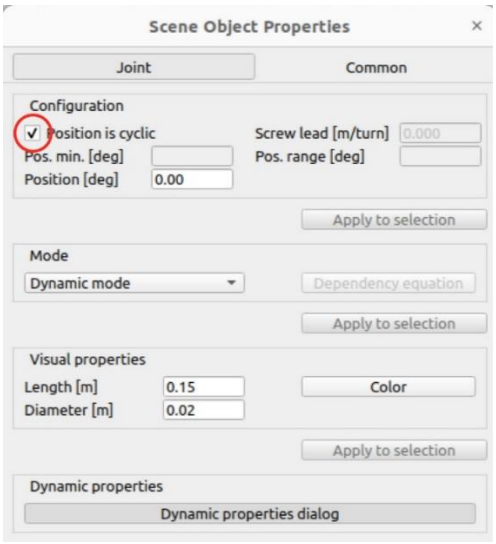

Figura 25. Configuración de los Joints de las Ruedas Delanteras.

Con respecto a los joints representantes del giro de las ruedas en el eje Y, es decir, "front_left_wheel_joint" y "front_right_wheel_joint", estos deben ser libres de fricción y no necesitan ser controlados por la información proveniente de ROS2, sino responder al movimiento potenciado por las ruedas traseras. Por esto, sus propiedades serán cambiadas a ser *free joints*. Esto hace que su pestaña de propiedades quede de la siguiente forma:

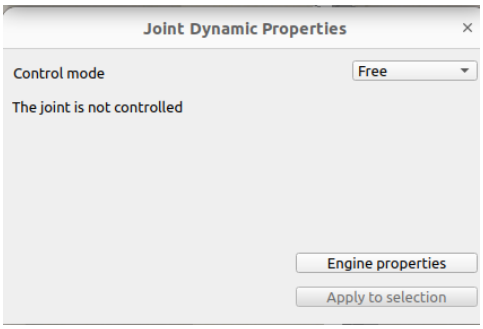

Figura 26. Propiedades Dinámicas Finales de las Ruedas Directrices, CoppeliaRobotics.

Con la finalidad de evadir contraposición entre los modelos 3D del sistema fue requerido que los joints "front_left_wheel_joint" y "front_right_wheel_joint" sean movidos 0.001 metros hacia afuera del modelo, lo que representa distancia completamente imperceptible para el usuario y para la relación del sistema, pero evitando roces con sus *parent frames*. Adicionalmente, ha subido el modelo completo 0.034 metros, siendo igual que en el punto anterior un cambio imperceptible, pero evitando conflictos con la superficie de la escena.

Para la incorporación del sensor no se ha usado el modelo exacto del sensor UTM-30LX, el cual no existe en el catálogo de modelos de Coppelia Robotics. Dentro de las opciones ofrecidas se encuentra el modelo "Hokuyo URG 04LX UG01_Fast ROS" (Figura 27), el cual encaja como modelo genérico para la mayoría de los sensores de rango de esta empresa.

Figura 27. Modelo Genérico de un Sensor Hokuyo, Coppelia Robotics.

Como se detalló en el apartado anterior, el sensor real fue posicionado sobre una plataforma, la cual se encuentra montada sobre el robot. Esta debe ser tomada en cuenta para la colocación del sensor virtual con respecto al sistema local del robot. Primero se debe hacer una relación entre el robot y el sensor, asegurando que se muevan en conjunto. Para esto se asigna al robot como el *parent frame* del sensor. Luego, con respecto a las

coordenadas, se asume que el sensor se encuentra alineado al eje X del robot, es decir, en el medio del eje de las ruedas, dando una distancia nula en el eje Y. Además, esta tendrá una altura sobre el robot de 30.5 centímetros desde el centro del modelo virtual y con una distancia de 21.6 centímetros en el eje X. Se ilustra en la Figura 28 la posición final del modelo, así como su posición relativa en la Figura 29.

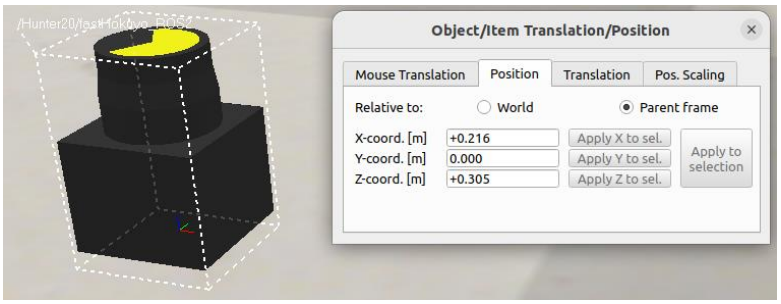

Figura 28. Posición Relativa del Sensor Hokuyo con el Modelo Virtual del Robot.

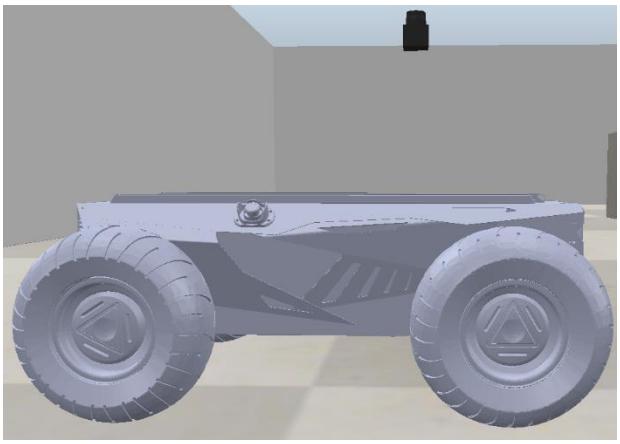

Figura 29. Conjunto Virtual Completo del Robot Hunter 2.0 y el Sensor Hokuyo.

Para poder crear una relación directa entre las señales recibidas por los topics y estas características físicas del modelo se deben cumplir las ecuaciones cinemáticas de la configuración Ackermann. El topic de ROS2 publica una señal en la que se incluye la velocidad angular y la lineal de todo el sistema, las cuales deben ser transformadas en las velocidades y posiciones de cada rueda. En el caso de las ruedas directrices, estas serán controladas en función de la velocidad angular mientras que las motrices dependerán de ambas velocidades.

Se parte convirtiendo la velocidad angular en la orientación de las correspondientes ruedas directrices. Se usarán las ecuaciones (11) y (12) para el cálculo de la orientación de ambas ruedas a partir de la distancia entre los ejes, el ángulo de orientación del sistema y la distancia entre las ruedas delanteras.

Con respecto a las ruedas motrices, se sabe que la velocidad angular es nula cuando estas ruedas tengan velocidades iguales a la velocidad lineal del sistema, mientras que si existe velocidad angular sus velocidades serán diferentes, cumpliendo la ecuación (13).

Para la correcta interpretación de las coordenadas en los diferentes sistemas de referencia se necesita un sistema de transformación (TF). En la Figura 30 se observa cómo quedaría el de este proyecto.

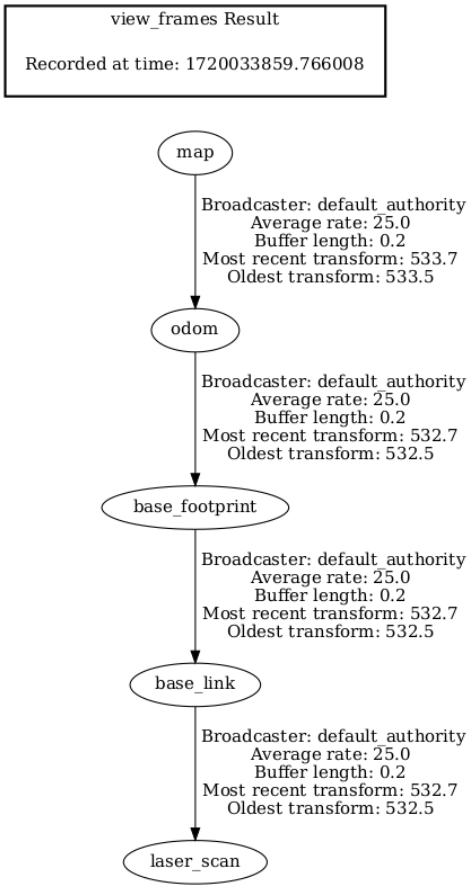

Figura 30. Sistema de Transformación de Coordenadas.

4.4. Planificación y Seguimiento de Trayectorias

Para que el robot tenga un movimiento considerado como autónomo es necesario que este cumpla con la planificación y seguimiento de trayectorias, para lo cual se usará inicialmente el stack de navegación Nav2, como se comentó anteriormente, basándonos en las diferentes limitaciones de la configuración de Ackermann.

Con anterioridad se han mencionado todas las subramas involucradas en el sistema del stack, por lo que primero se mencionarán aquellas que no están directamente relacionadas con el modelo de robot usado. El primero es el *Map Server,* el cual se encarga de cargar y proveer de información proveniente del mapa. En este se incluye el nombre del archivo .yaml del mapa, así como el topic por el que se publica la información relevante y el nombre del *frame_id,* el cual, en ambos casos, los nombres serán:

- topic_name: “/map”
- frame_id: “map”

El siguiente es el denominado *Behavior Tree*, en el cual se especifica la estructura y los diferentes plugins que se encargarán de definir el comportamiento y sincronización entre los nodos del sistema. Los frames y topics usados son:

- global_frame: “map”
- robot_base_frame: “base_link”
- odom_topic: “/odom”

Todos estos permitirán una correcta localización entre el mapa y el robot. Finalmente se menciona el *Waypoint Follower*, el cual ha sido anteriormente mencionado y cuenta con un único plugin capaz de cumplir la función de seguimiento de puntos intermedios hasta llegar a la meta.

Se pasará a hablar de los nodos más complejos del sistema, los cuales, al mismo tiempo, dependen del modelo usado. El primero es el localizador *AMCL Server*, el cual usa los siguientes topics y frames:

- base_frame_id: “base_footprint”
- global_frame_id: “map”
- odom_frame_id: “odom”
- scan_topic: “/laser_scan”
- map_topic: “/map”

Es importante destacar que se ha elegido la pose inicial en el origen del sistema de referencia global con el fin de tener una referencia inicial, por lo que cada vez que se inicie será necesario introducir la pose inicial real del modelo. Por otro lado, el localizador requiere de la especificación del modelo de robot usado. A pesar de no contar con el modelo Ackermann como posible entrada se cuenta con el modelo Omnidireccional, el cual es un modelo sin restricciones que es capaz de moverse libremente en cualquier dirección. Aunque es un modelo muy genérico, no crea ningún conflicto con el modelo usado. La segunda opción es el modelo Diferencial, el cual se basa en un movimiento con dos ruedas, el cual cuenta con características de movimiento que entran en conflicto con la configuración Ackermann.

El *Planner Server* utilizado usa la información de los *costmaps* para crear un camino a la meta. Este servidor cuenta con diferentes plugins que usan bases teóricas diferentes para la creación de trayectos, así como tener en cuenta diferentes características que pueden ser relevantes en función del modelo. Según la documentación del Nav2 [24], las opciones para el modelo Ackermann son:

- SmacPlannerHybrid

- SmacPlannerLatrice

En este proyecto se usará la primera opción, ya que toma en cuenta la diferencia entre los conceptos de robots móviles Dubin o Reeds-Shepp [31], lo cual es relevante ya que son métodos de planificación de rutas a partir de la velocidad y radio mínimo de giro. La principal diferencia es que el segundo permite el movimiento en reversa indispensable para robots de cuatro ruedas. En este plugin se tiene en cuenta factores como el cambio de dirección, curvatura de la trayectoria, retrocesos y el radio de giro mínimo.

Con el *Controller Server* ocurre la misma situación, en el cual existen diferentes plugins que encajan con el modelo usado. Estos pueden ser:

- TEB Controller
- MPPI Controller
- Regulated Pure Pursuit

La primera opción es una versión anterior de la segunda, y por ende no está disponible para las versiones recientes de ROS2. El MPPI Controller parece ser la opción óptima para la cinemática del proyecto, ya que posee parámetros orientados a Ackermann que permitirían dar resultados más personalizados al robot. Sin embargo, existen problemas de incompatibilidad con ROS2 Humble y los diferentes plugins internos del controlador, por lo que esta opción queda descartada temporalmente hasta que sea solucionado por la empresa Open Navigation. Finalmente se ha optado por el Regulated Pure Pursuit, el cual es la versión más reciente y testeada con modelos Ackermann. Este algoritmo está basado en el seguidor de trayectorias del mismo nombre, el cual usa una velocidad lineal constante y calcula una velocidad angular adecuada para alcanzar un punto de la trayectoria que se encuentre cercano al robot. Se usará una velocidad lineal de 0.6 m/s y se seguirá un punto de 0.6 metros de distancia.

Durante las pruebas realizadas al final del proyecto se comprobó que el seguimiento de puntos basado en una velocidad angular variable genera movimientos oscilatorios que no solo ralentiza, sino que entorpece el movimiento del robot. Este percance, sin embargo, es tomado en cuenta en el desarrollo de los algoritmos del proyecto, pero es considerado como un fallo en el uso del stack Nav2.

Dentro del sistema se encuentran dos *Costmaps,* los cuales son el local y el global. A cada uno le corresponden los siguientes frames y topics:

- Local:
    - Global_frame: “odom”
    - Robot_base_frame: “base_link”
    - Scan topic: “/laser_scan”
- Global:
    - Global_frame: “map”

- Robot_base_frame: “base_link”
- Scan topic: “/laser_scan”

Dentro de estos mapas se encuentra el parámetro *inflation radius*, el cual determina la distancia de seguridad que debe existir entre los objetos y el robot para evitar colisiones. Inicialmente, con un valor de 0.2 se comprobó que las oscilaciones causadas por el Controller server causaban que el robot se acercara más de lo adecuado a los obstáculos. Debido a esto, es requerido mantener este valor a 0.5 como mínimo, reduciendo la posibilidad de choques. Sin embargo, en ambientes con poco espacio entre los obstáculos muchas de las trayectorias quedan bloqueadas debido a la falta de espacio entre las distancias de seguridad. Debido a esto, las pruebas planteadas al final del proyecto se realizarán en ambientes exteriores.

Por último, es importante mencionar que estas ramas del sistema tienen en cuenta las dimensiones del robot, las cuales pueden ser modeladas como un modelo circular o rectangular. En este caso, lo ideal es un modelo rectangular cuyas dimensiones pueden apreciarse en la Figura 31.

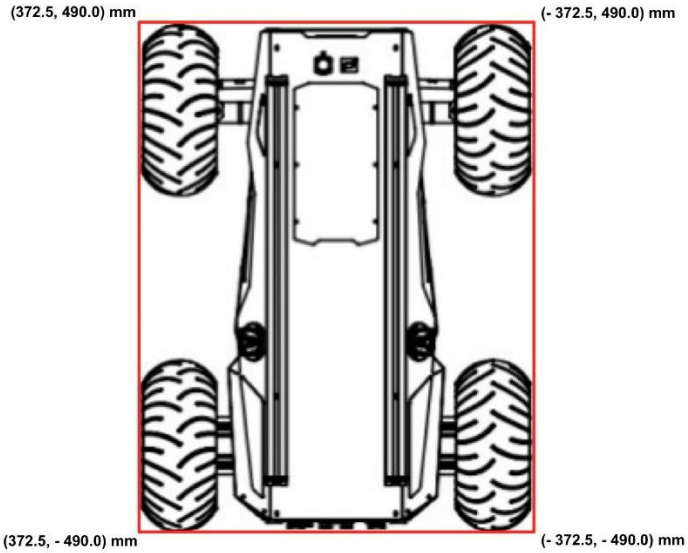

Figura 31. Dimensiones Externas del Robot.

Finalmente, se mencionará el *Recovery Behaviors*, el cual es el encargado de tomar el control del robot una vez el *Controller Server* sea incapaz de alcanzar el objetivo. Este servidor ha demostrado ser altamente limitado, debido a la falta de plugins que se adapten a las limitaciones del modelo Ackermann. Esta rama cuenta con cuatro plugins internos:

- Wait: encargado de esperar un tiempo prudencial hasta que el obstáculo desaparezca (en caso de ser móvil) o hasta que se actualice la información del sensor.
- Spin: realiza el movimiento de giro del robot sobre su propio eje hasta que el objeto este fuera de la trayectoria.

- BackUp: permite que el robot retroceda una distancia prudencial hasta que el obstáculo se encuentre fuera de rango.

Como es lógico, estas acciones son altamente limitadas ya que el robot no puede girar sobre su propio eje y los plugins de giro y retroceso no actúan simultáneamente, por lo que únicamente causan que el robot gire las ruedas directrices con una velocidad angular máxima y luego retroceda con una velocidad lineal de -0.05m/s, dando pie a que únicamente evite ciertos obstáculos que se encuentren en zonas laterales y alejadas del eje central del robot. Teniendo en cuenta estos problemas, se han ajustado todos los parámetros con la finalidad de prevenir el uso de estas acciones de recuperación.

Vale la pena resaltar que, cuando Nav2 tiene el control del movimiento de robot este evitará colisiones con los obstáculos deteniendo el movimiento una vez estos se encuentran muy cerca del robot, como podemos ver en la Figura 32 en la que siempre mantendrá una distancia de seguridad.

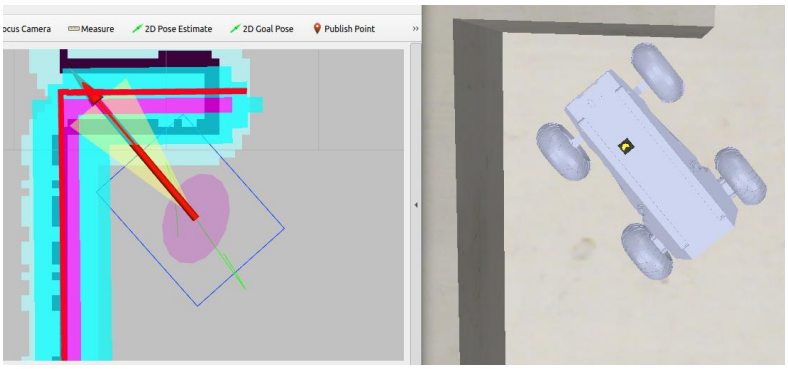

Figura 32. Distancia de Seguridad con obstáculos.

Se aprecia en la Figura 33 una trayectoria creada por Nav2, en la que claramente toma distancias de seguridad con los obstáculos, mientras que en la Figura 34 se observa una colisión con un obstáculo con esa misma trayectoria, pero con el parámetro de seguridad reducido a 0.2.

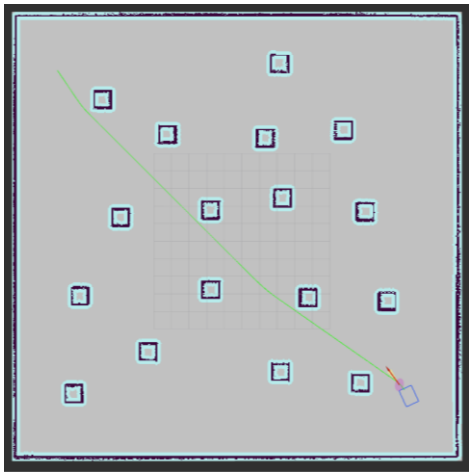

Figura 33. Trayectoria generada por Nav2.

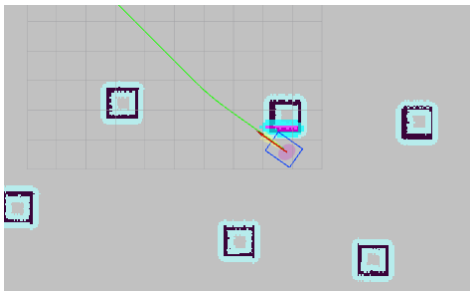

Figura 34. Colisión debido al control por Nav2.

4.5. Configuración en RViz2

Una vez que se tiene el stack preparado es necesario una plataforma en la que visualizar las trayectorias creadas por Nav2 y la información relevante pasada por los topics. Para esto, se ha elegido la siguiente configuración en RViz2:

- El frame usado como sistema de referencia local será el "map", el cual será estático. Toda la información se ilustrará referenciado a este frame.
- Con el topic "/map" se ilustra el mapa introducido en los archivos de Nav2.
- Se selecciona el topic "/amcl_pose" para visualizar la ubicación y orientación estimadas del robot con una flecha.
- "/laser_scan" sirve para enseñar la nube de puntos generada a partir de la información recibida por el sensor.
- A través de los topics "local_costmap/costmap" y "global_costmap/costmap" se puede observar ambos costmaps a través de mapas de colores azul y morado, tal y como se puede apreciar en la Figura 33.
- El topic "/plan" nos mostrará la trayectoria completa que el robot debe seguir hasta la meta, mientras que el topic "/local_plan" mostrará la trayectoria más próxima.
- Finalmente, el topic "/global_costmap/published_footprint" nos permitirá observar las dimensiones del robot, visible como un rectángulo de color azul.

Estos datos se podrán corroborar en la Figura 35. Los resultados se ilustran en la Figura 36.

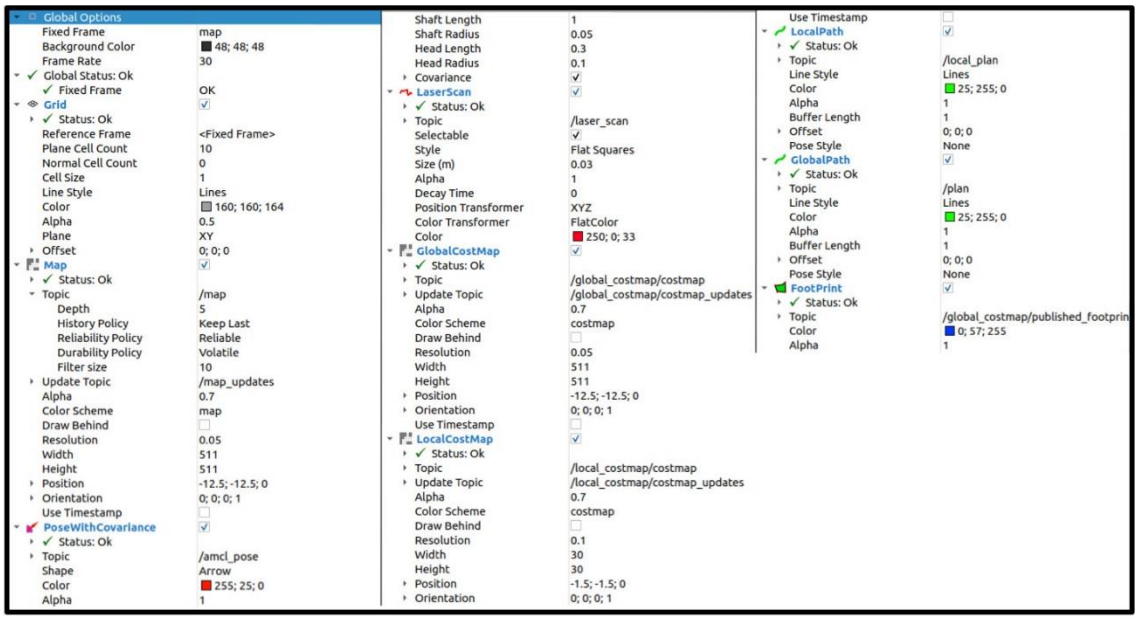

Figura 35. Configuración en RViz2.

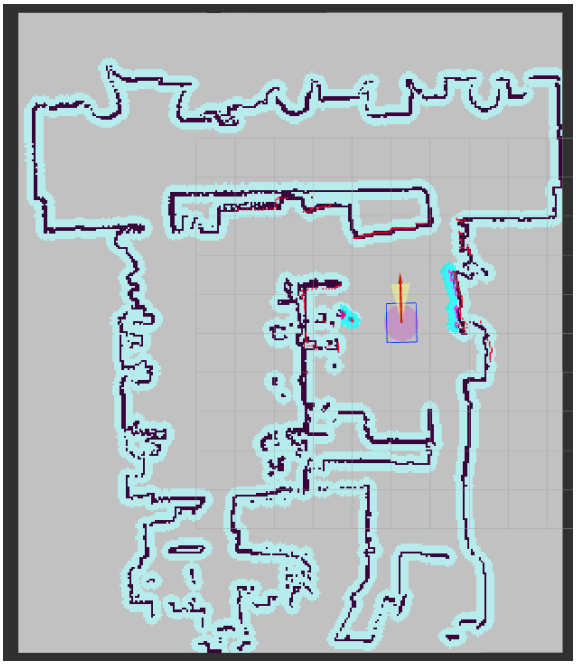

Figura 36. Ilustración 2D de los datos, RViz2.

4.6. Maniobra de Aparcamiento

El algoritmo de aparcamiento es imprescindible para el movimiento autónomo de cualquier robot. Como se ha mencionado en apartados anteriores, la morfología del modelo es idéntica a la de un coche común y, por tanto, requiere de las mismas maniobras para aparcar. En este proyecto se presentará el aparcamiento en línea (Figura 37), detallando las bases teóricas y detalles del código creado.

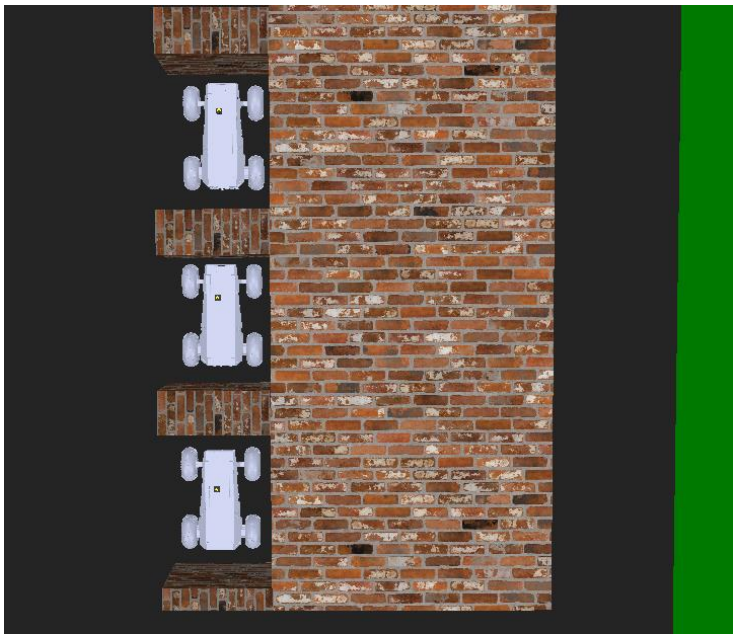

Figura 37. Aparcamiento en línea.

Aprovechado el paralelismo con los vehículos comunes y considerando los consejos comunes para un aparcamiento en paralelo [32], se seguirán los siguientes pasos:

1) Se aprovecha el movimiento autónomo de Nav2 y la evasión de obstáculos para recorrer los diferentes puestos disponibles.
2) Se usan datos del sensor para detectar un espacio de aparcamiento válido.
3) Una vez detectado, el robot se posicionará pasado el espacio vacío, alineando el eje de las ruedas traseras con el borde final de este.
4) Se girarán las ruedas delanteras en dirección al aparcamiento, y se comenzará a retroceder hasta que la parte trasera del robot este dentro del aparcamiento.
5) Continua con el retroceso, girando las ruedas delanteras al lado opuesto.
6) En la mayoría de los casos puede ser necesario alinear apropiadamente el robot, para lo cual se hará un movimiento hacia adelante con las ruedas delanteras apuntando en dirección al aparcamiento.

Para facilitar el desarrollo del algoritmo se asumirán los siguientes datos:

- Todos los puestos de estacionamiento estarán alineados a ambos lados del robot, por lo que el movimiento previo solo tiene que ser en línea recta.
- Un puesto de estacionamiento válido se entiende como un espacio rectangular que tenga como mínimo un largo de 98 centímetros y una profundidad de 74.5 centímetros.

- Si se detectan distancias mayores a cuatro metros entre el robot y un objeto se considerará que este no forma parte del parking y no se tomará en cuenta en la búsqueda.
- Como se comprobó en apartados anteriores, Nav2 tiene grandes complicaciones con espacios limitados o con movimientos precisos, por lo que las maniobras de aparcamiento se realizarán manualmente a través del código de control implementado en este trabajo. Nav2 perderá el control del movimiento una vez se encuentre el puesto adecuado.
- Dado que el movimiento es lineal asumiremos que la distancia entre el robot y los puestos es constante y que es suficiente para inicializar las maniobras desde ese punto.

El escenario resultante en CoppeliaRobotics es diseñado basado en estos aspectos. Los puestos de aparcamiento tienen dimensiones de 1.0 x 2.125 metros con un pasillo intermedio de 3 metros de ancho. En cada lado del escenario existen seis espacios libres.

El principal reto recae en la localización del puesto válido. Se ha buscado una solución que no use los datos del mapa y permitiendo que el robot sea capaz de buscar aparcamientos incluso en ambientes desconocidos usando únicamente los datos del sensor. De esta forma se ha llegado a la siguiente lógica:

Usamos los datos del sensor correspondientes a los ángulos de +/- 90 grados (siguiendo la referencia de la Figura 17) correspondientes a la dirección perpendicular al eje de movimiento del robot a ambos lados. Estos datos darán la distancia de los objetos a ambos lados del robot, lo cual resuelve la dimensión de profundidad del puesto libre. Para el largo del puesto se debe avanzar en línea recta y medir la distancia entre el primer punto con profundidad válida hasta que tenga el largo mínimo. Para simplificar la explicación se ilustrará una situación genérica del lado derecho del robot.

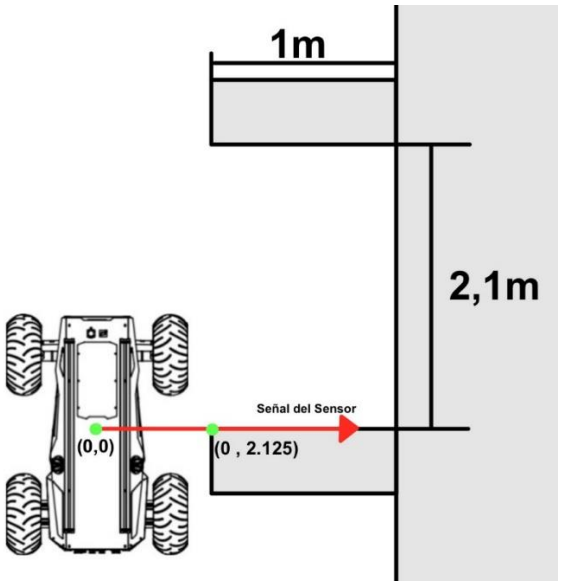

Figura 38. Análisis Inicial de un Puesto de Aparcamiento.

En la Figura 38 se observa al robot en el inicio de un puesto de aparcamiento vacío. Por facilidad se tomará las coordenadas del sensor como el origen del sistema de referencia. De esta forma, la esquina inferior izquierda del aparcamiento se encuentra en el punto (0, 2.125). En los siguientes ciclos de ejecución, ilustrado en la Figura 39, el sensor detectará un cambio en la profundidad del objeto detectado, cuya diferencia será de 74.5 centímetros como mínimo, por lo que considerará este espacio como un posible puesto de aparcamiento.

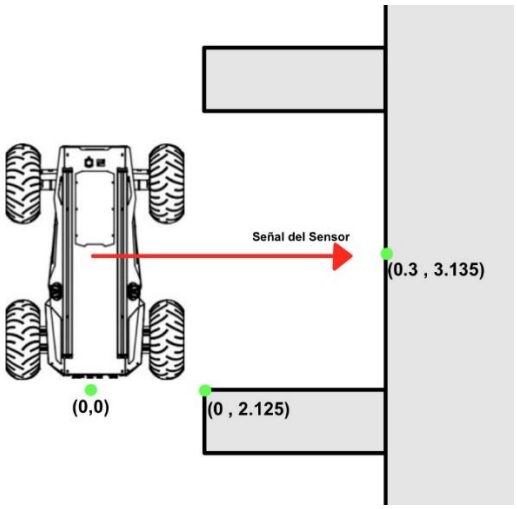

Figura 39. Análisis Intermedio de un Puesto de Aparcamiento.

Es importante destacar que para la realización de este código se ha usado una conversión de ciclos de ejecución y espacio recorrido en cada uno. Para comprobar la profundidad del espacio se debe comparar constantemente la distancia del punto actual con la altura de la esquina inferior. Para esto, el sistema debe guardar el punto más cercano que haya detectado hasta el momento con la finalidad de usarlo como referencia. Este valor, para este ejemplo, será de 2.125 metros. En caso de que la profundidad sea superior a dos metros de diferencia o se detecte un objeto a más de cuatro metros de distancia, se considerará que el parking ha terminado y no se consideraría como válido, mientras que si es una profundidad menor a 74.5 centímetros se considera que el puesto ha terminado y se considera la longitud recorrida por el robot. Como se ve en la Figura 40, es posible que se alcance la distancia requerida antes de llegar al final del puesto. Se lleva un conteo de la distancia recorrida, por lo que en el momento que se hayan recorrido los 98 centímetros y la profundidad siga siendo la correcta se considerará que el espacio es válido y se comenzará a realizar la maniobra de aparcamiento. Si se interrumpe la profundidad del puesto el conteo se reinicia y se espera a encontrar un nuevo espacio libre.

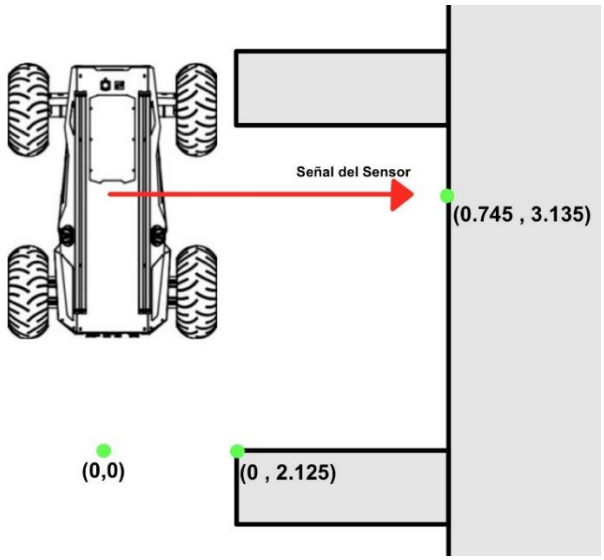

Figura 40. Detección Final de un Puesto de Aparcamiento.

El principal inconveniente con este mecanismo es que debido al uso del plugin Regulated Pure Pursuit el movimiento del robot será oscilatorio sobre la trayectoria, como se mencionó anteriormente, por lo que esto podría introducir datos incorrectos del sensor al no leer distancias de la dirección correcta. Esta situación se ilustra en la Figura 41. En la práctica, se ha demostrado que esto no presenta ningún inconveniente en casi ningún escenario. Sin embargo, es posible que detecte espacios más pequeños de lo permitido y causando que el robot choque con la pared trasera del puesto en el momento del retroceso.

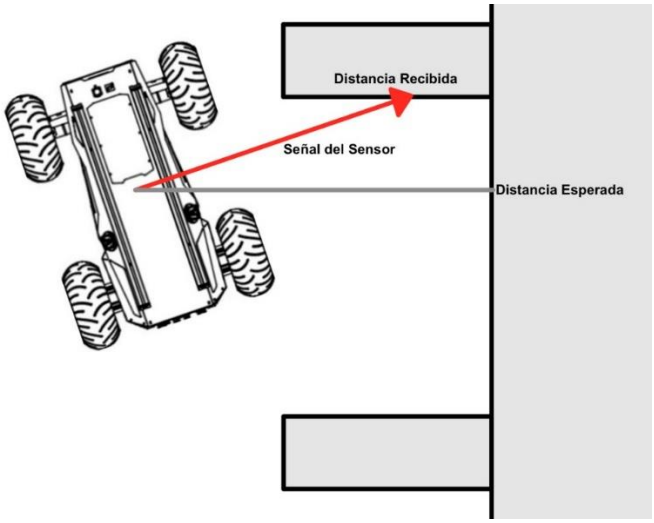

Figura 41. Puesto no Detectado Incorrectamente por Oscilación del Robot.

Para solucionar esto se ha estipulado un mínimo de ciclos de ejecución del código para considerar que la distancia leída es la correcta. En la mayoría de los casos esto causa que el algoritmo considere el puesto como válido con una longitud mayor a la mínima, pero esto solo introduce mayor espacio entre el robot y la pared trasera reduciendo la posibilidad de choques.

En el caso opuesto en el que las oscilaciones causen que el robot no detecte correctamente un puesto válido el modelo simplemente seguiría avanzando sin parar y detectaría el siguiente puesto válido, lo cual no introduce un riesgo de seguridad para la integridad del robot.

Una vez encontrado el puesto disponible se inician las maniobras de aparcamiento. Se describirán los pasos a seguir para un puesto en el lado derecho:

- Se inicia tomando el control del movimiento del robot al enviar la posición actual del robot como meta a Nav2 permitiendo que detecte que ha llegado a la meta.
- Luego, se realiza un movimiento hacia adelante con velocidad lineal de 0.1 m/s y angular de 0.45 m/s hasta alinear el robot con el puesto.
- Una vez con la orientación correcta se pasa a retroceder en línea recta a -0.3 m/s hasta que toda la zona trasera del robot se encuentre dentro del espacio de aparcamiento.
- Se continua con el retroceso, ahora con una velocidad lineal de -0.2 m/s, pero girando en sentido horario con velocidad angular de -0.3 m/s hasta que la orientación del robot se aproxime a la correcta.
- Ahora se comenzará con el proceso de alineación evitando que el robot toque la zona trasera. Se inicia con un movimiento hacia adelante con velocidad lineal de 0.2 m/s y angular de -0.3 m/s.
- Se retoma el movimiento de retroceso con los mismos -0.2 m/s de velocidad lineal y -0.3 m/s de velocidad angular.
- Finalmente, se realiza un movimiento delantero de 0.2 m/s de velocidad lineal y manteniendo los -0.3 m/s de velocidad angular.

Finalizado este proceso se asegura que se ha logrado el aparcamiento en cualquiera de los posibles puestos que haya en el escenario. Para los que se encuentran en el lado izquierdo, bastaría con invertir el signo de todas las velocidades angulares.

4.7. Creación de Mapas

Durante la realización de las pruebas se han requerido de una variedad de mapas que permita realizar diversas experiencias con el modelo real y sobre todo con el virtual. Inicialmente se usó el Laboratorio 2.3.6, mencionado varias veces en este proyecto, en el que se encuentra el robot real. Dentro de estos contactos iniciales se demostró que es ventajoso usar el modelo en ambientes abiertos, los cuales no estaban disponibles físicamente, por lo que muchas de las pruebas se realizaron únicamente en el modelo virtual de CoppeliaRobotics.

Para la creación de los mapas de mundo abierto se ha usado la página *OpenStreetMap*, la cual permite al usuario crear mapas en formato .osm de áreas concretas. Estos formatos no son compatibles con CoppeliaRobotics, por lo que se ha usado el apoyo del software OSM2World, que los convierte a formato .obj, el cual puede ser introducido en el escenario como un *mersh*. Cabe destacar que estos mapas son introducidos en una escala a 2:1 debido a sus grandes dimensiones permitiendo facilitar la creación de todos los mapas.

Con esta página se han extraído cuatro áreas diferentes, las cuales son:

- Facultad de Filosofía y Letras, Universidad de Málaga: se ha elegido este espacio gracias a sus dimensiones rectangulares y sus áreas verdes. Se usará con la finalidad de realizar trayectorias a través de los edificios corroborando el comportamiento de la evasión de obstáculos de grandes dimensiones. Las coordenadas son las siguientes:
    - Mínima Latitud: 36.714566
    - Máxima Latitud: 36.716278
    - Mínima Longitud: -4.469537
    - Máxima Longitud: -4.471253

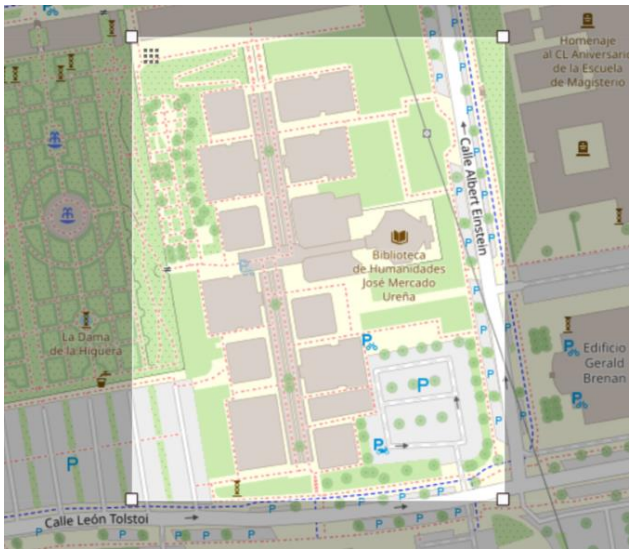

Figura 42. Mapa Facultad de Filosofía y Letras, OpenStreetMap.

- Aparcamiento de la Facultad de Turismo, Universidad de Málaga: la distribución aleatoria de los árboles en el mapa permite la creación de una variedad de obstáculos en una misma trayectoria, así como observar el comportamiento del robot cuando transita entre obstáculos relativamente cercanos. Las coordenadas son las siguientes:
    - Mínima Latitud: 36.713323
    - Máxima Latitud: 36.714716

- Mínima Longitud: -4.466903
- Máxima Longitud: -4.469365

Figura 43. Mapa Aparcamiento de la Facultad de Turismo, OpenStreetMap.

- Parque Infantil, Ciudad de la Justicia: similar al área anterior, es una zona natural con árboles dispuestos en todo el espacio con un edificio en el borde, lo cual permite realizar trayectos en zonas verdes y con edificaciones. La diferencia radica en que este mapa tiene menos obstáculos y un mayor espacio abierto. Las coordenadas son las siguientes:
  - Mínima Latitud: 36.714770
  - Máxima Latitud: 36.715748
  - Mínima Longitud: -4.463367
  - Máxima Longitud: -4.465084

Figura 44. Mapa Parque Infantil, OpenStreetMap.

- Aparcamiento Público, Ciudad de la Justicia: este espacio del mapa se encuentra inicialmente vacío con algunas hileras de árboles que será usado para el diseño de

un aparcamiento virtual para el modelo. El diseño final de este escenario será mostrado más adelante. Las coordenadas son las siguientes:

- Mínima Latitud: 36.713090
- Máxima Latitud: 36. 714974
- Mínima Longitud: -4.462768
- Máxima Longitud: -4.466785

Figura 45. Mapa Aparcamiento Público, OpenStreetMap.

En la Figura 46 se observan las cuatro áreas mencionadas en *Google Maps*, donde se aprecia que forman parte de la zona universitaria de málaga, razón principal por la que fueron elegidas para realizar las experiencias.

Figura 46. Mapa de la Zona Universitaria de Málaga, Google Maps.

Finalmente, para las pruebas de aparcamiento se ha logrado usar un pasillo interior en la E.T.S.I Informática. Este pasillo posee pilares circulares con suficiente espacio entre ellos para simular un puesto de aparcamiento. Esto permite simular un espacio de aparcamiento

para el modelo real en el que se realizarán diferentes experimentos para la maniobra de aparcamiento.

Figura 47. Hunter 2.0 Navegando en el Pasillo Interior, E.T.S.I Informática.

Como se ha mencionado con anterioridad, la herramienta usada para el escaneo de los escenarios usados en el proyecto es SLAM toolbox, el cual usa información obtenida a través del sensor para crear un mapa 2D del entorno. Para ello, se ha controlado el robot por teclado permitiendo que recorra todo el terreno alcanzable. Los mapas creados para el proyecto son los siguientes:

Figura 48. Escenario del Laboratorio 2.3.6, E.T.S.I Informática.

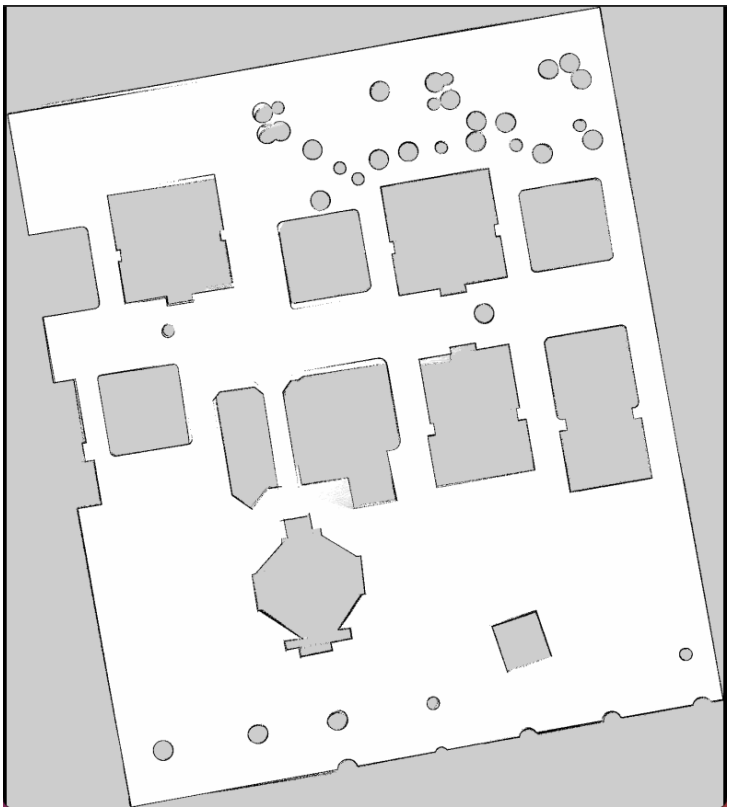

Figura 49. Escenario del Facultad de Filosofía y Letras.

Figura 50. Escenario del Aparcamiento de la Facultad de Turismo.

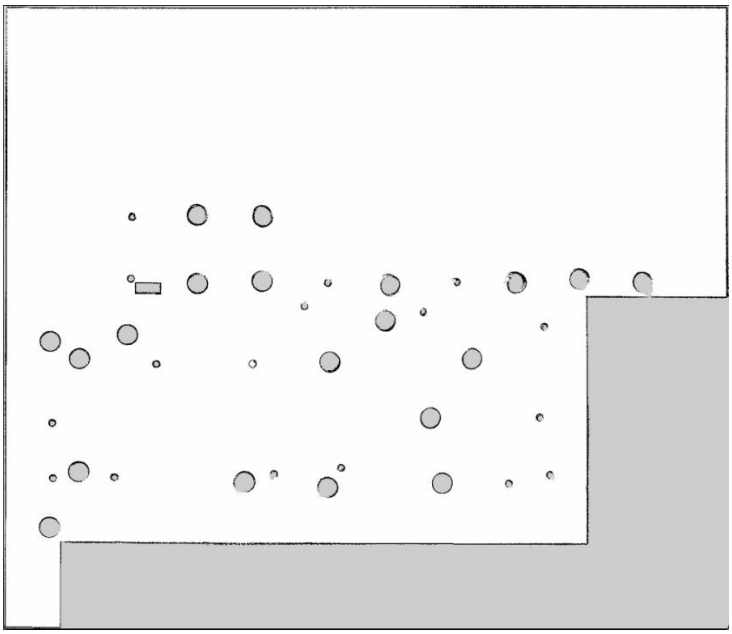

Figura 51. Escenario del Parque Infantil.

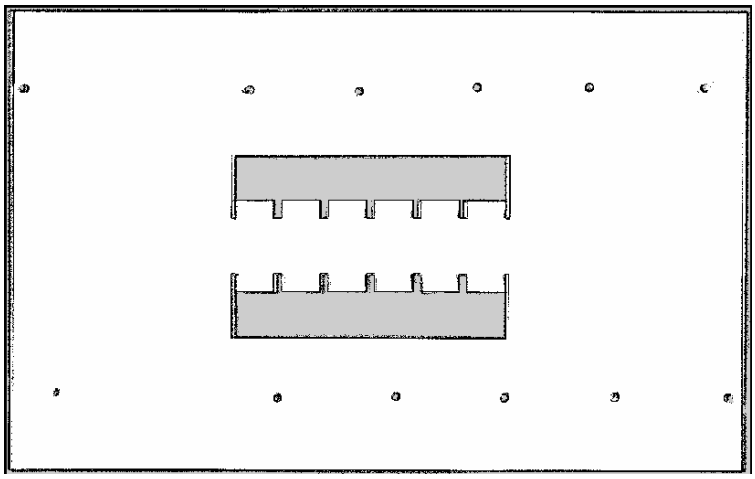

Figura 52. Escenario del Aparcamiento Público.

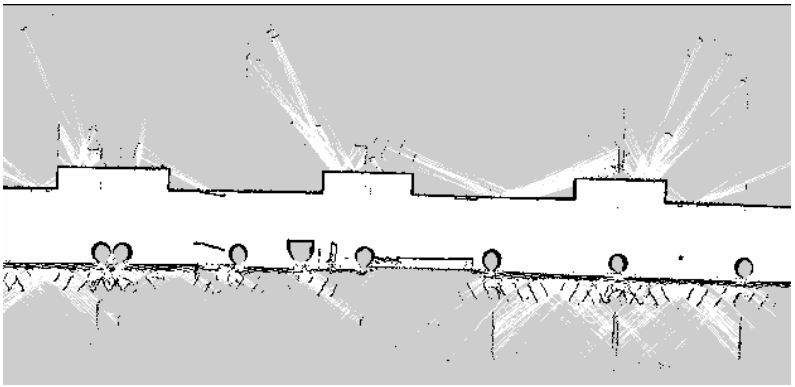

Figura 53. Mapa de un Pasillo Interior, E.T.S.I Informática.

5. Resultados

En este capítulo se detallarán todas las pruebas y experiencias obtenidas durante el desarrollo del proyecto. Desde el inicio se han comprobado el correcto comportamiento de todos los algoritmos de control sobre el robot, ya sea en el modelo real o el virtual. De esta forma, se ilustrarán los resultados y se demostrará que todos los objetivos fueron alcanzados exitosamente. A continuación, se pasarán a desarrollar las siguientes pruebas:

- **Movilidad con Espacio Reducido**: este experimento incluye las primeras experiencias con el robot, tanto en el modelo real como el virtual y se desarrollará en el Laboratorio 2.3.6, E.T.S.I Informática. Debido a las dimensiones de la habitación, Nav2 requiere realizar movimientos bastante precisos que el robot no es capaz de realizar, por lo que las trayectorias posibles en este ambiente son cortas, pero útiles para estudiar los movimientos simples.
- **Trayectoria en Exteriores**: se han creado escenarios en CoppeliaRobotics que permitirá ampliar las experiencias con el movimiento del modelo virtual. De la misma forma, se usarán estos escenarios para probar el algoritmo de evasión de obstáculos incluido en Nav2.
- **Aparcamiento en el Modelo Virtual:** las pruebas iniciales con las maniobras de aparcamiento se realizarán en un escenario simulado debido a la precisión requerida en los movimientos para evitar choques con el modelo real. Para ello se ha creado un parking en CoppeliaRobotics en el que se puedan incluir puestos tanto vacíos como ocupados.
- **Aparcamiento en el Mundo Real**: se ha usado un pasillo interior del E.T.S.I Informática, gracias a su geometría similar a espacios rectangulares que simularán puestos de aparcamiento. En este se centrarán las experiencias a comprobar la efectividad del algoritmo al detectar puestos libres.

5.1. Espacio limitado

Las pruebas realizadas dentro del laboratorio comenzaron con una comparación entre los modelos real y virtual a través de instrucciones manuales idénticas en cada modelo con la finalidad de comprobar que el movimiento del modelo simulado es realista. Dentro de estos contactos iniciales con el robot en el ambiente real fue evidente las limitaciones físicas que presentaba el laboratorio, ya que se necesitaba de mucha precisión con el mando para navegar por los espacios disponibles, especialmente a través de las puertas de las instalaciones, las cuales tenían el espacio justo para que el robot navegase. Este aspecto está más presente en el modelo virtual del laboratorio, apreciable en la Figura 54,

en la que se aprecia como los espacios de las puertas al pasillo exterior no es suficiente para el robot.

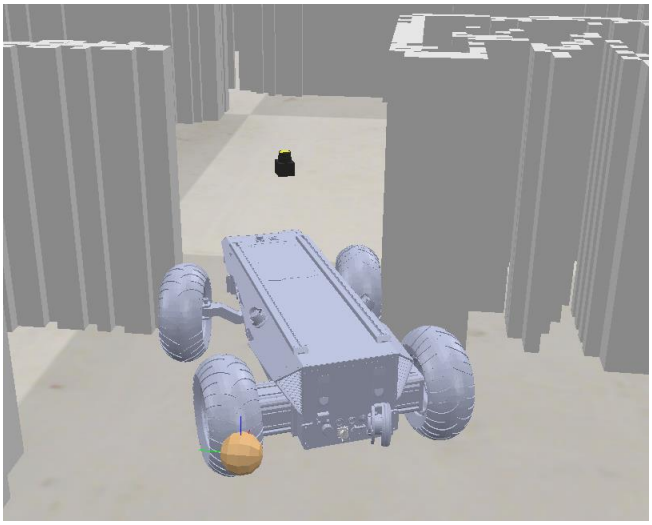

Figura 54. Choque entre el Robot Virtual y una Puerta del Laboratorio.

Debido a esto, las pruebas iniciales se concentraron en movimientos de corta distancia y comparación del giro de los modelos. Estas experiencias fueron útiles para mejorar la precisión del modelo virtual, principalmente con los giros de las ruedas delanteras, los cuales presentaban giros mucho más pronunciados al inicio de las experiencias.

Una vez introducido Nav2 en las experiencias, se pudo comprobar que el comportamiento de evasión de obstáculos generaba altas complicaciones para la navegación dentro del laboratorio. Principalmente en los giros, el robot quedaba atascado con esquinas sin ser capaz de esquivarlas en la mayoría de las experiencias. Este aspecto motivó, además, la modificación del parámetro inflation radius, recordando que define la distancia con los obstáculos. Sin embargo, este cambio resultó en la imposibilidad para el robot de navegar por la habitación dentro del laboratorio.

Debido a que en el mapa generado el espacio de las puertas era bastante justo y aun usando una distancia de seguridad de 0.2 creada por los costmaps, Nav2 no creaba trayectorias desde el laboratorio al pasillo y viceversa. Sin embargo, dentro de las habitaciones del laboratorio, como se aprecia en la Figura 55, las trayectorias eran simples y no requería ninguna evasión de posibles obstáculos en el camino. Aunque ineficiente, esto permitió comprobar la mayoría de los parámetros elegidos para el Stack.

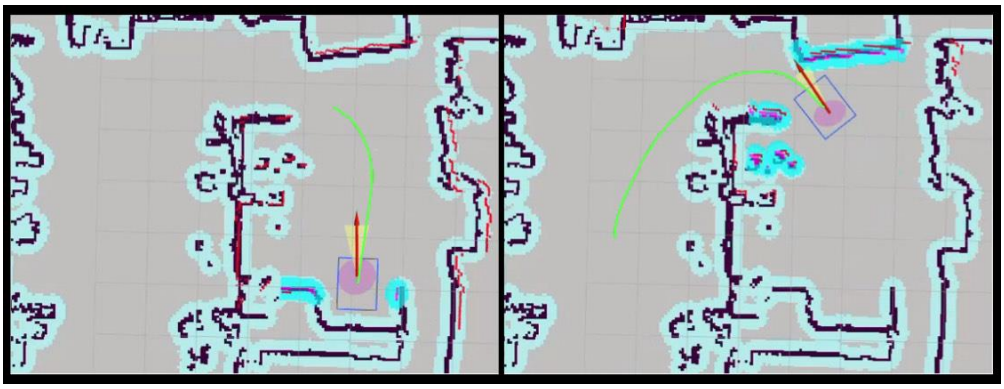

Figura 55. Trayectoria generada por Nav2 dentro del Laboratorio.

Finalmente, tomando en cuenta la problemática relacionada con Nav2 en este ambiente, se creó un conjunto de comandos de velocidades sin este stack que permitiera realizar la trayectoria aproximada de la Figura 56.

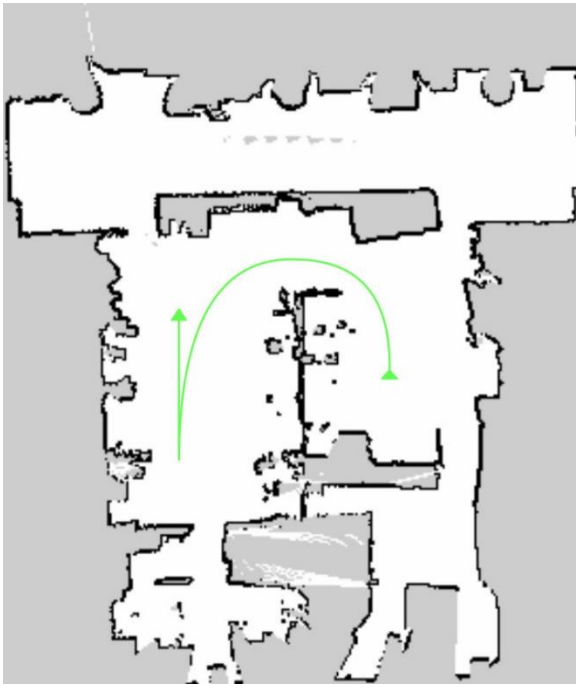

Figura 56. Trayectoria Manual dentro del Laboratorio.

Esto permitiría comparar el movimiento lineal y curvo del modelo real con el virtual, los cuales resultaron satisfactorios. Es evidente que existen datos físicos tales como fricción y deslizamiento de las llantas que crean pequeñas diferencias entre ambos modelos. Sin embargo, en la Figura 57 se comprueba que las posiciones finales en ambos casos son altamente similares, por lo que se considera estas diferencias como despreciables y los resultados del modelo virtual como satisfactorios.

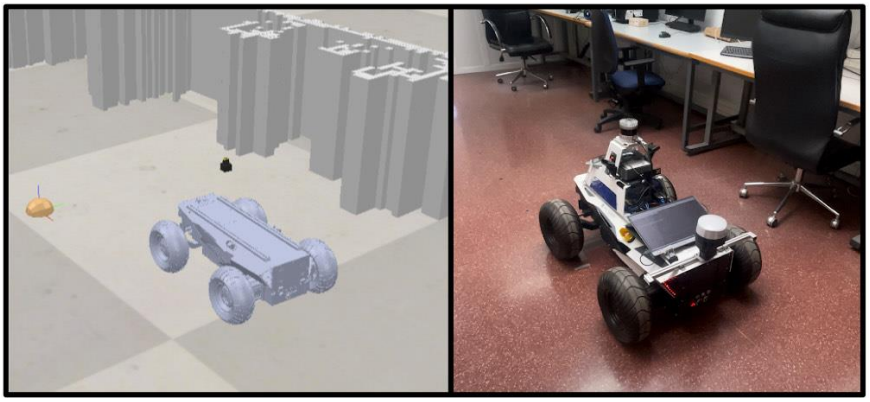

Figura 57. Posiciones Finales de Ambos Modelos.

5.2. Ambiente Exterior con obstáculos

Los experimentos realizados en ámbitos exteriores fueron desarrollados en el modelo virtual, por lo que se ha procedido a crear tres escenarios que simulan un terreno exterior con diferentes objetos espaciados entre ellos con la finalidad de demostrar el correcto funcionamiento de la navegación autónoma. Los escenarios deben poseer espacio pleno entre los objetos, preferiblemente dispersos de manera aleatoria simulando zonas con vegetación y árboles. De la misma forma, se busca una zona urbana con diferentes edificios con esquinas que puedan entorpecer los giros del robot. Los escenarios, ilustrados en *OSM2World*, son los mencionados en el apartado Creación:

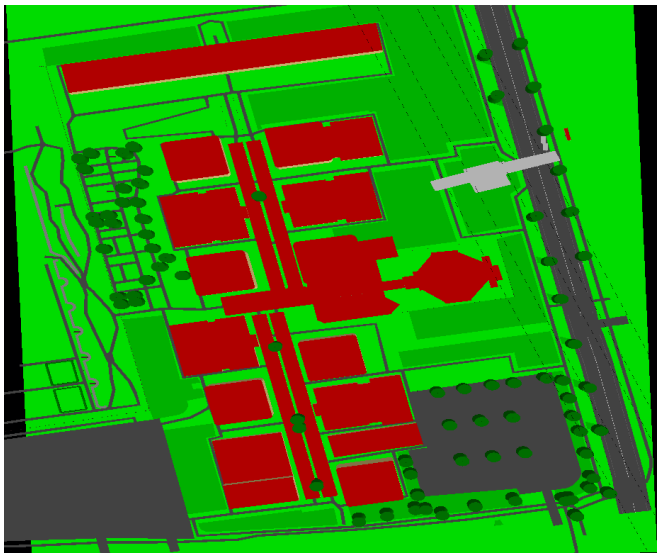

Figura 58. Escenario de Obstáculos #1: Facultad de Filosofía y Letras.

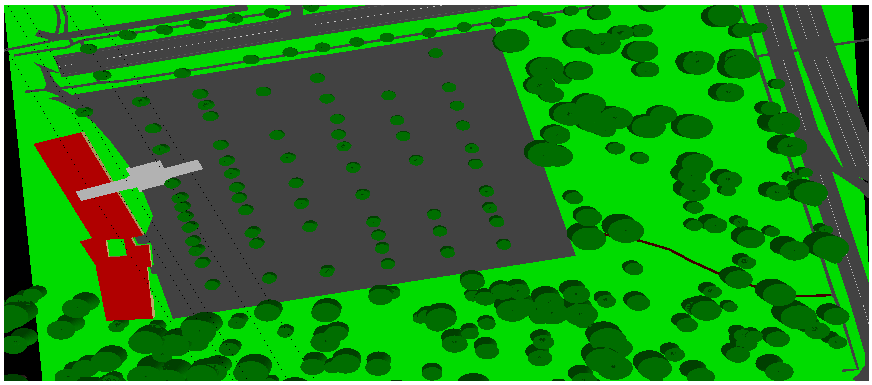

Figura 59. Escenario de Obstáculos #2: Aparcamiento de Facultad de Turismo.

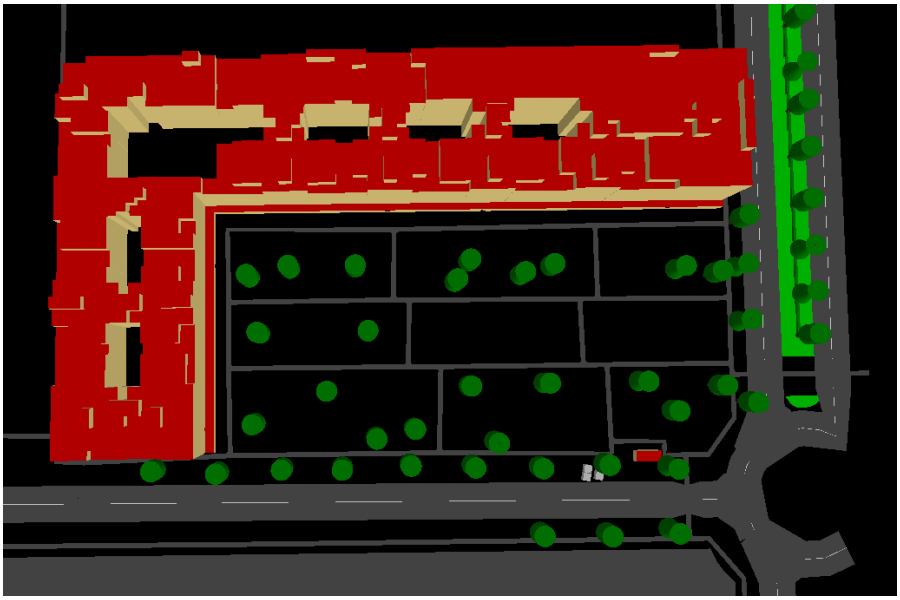

Figura 60. Escenario de Obstáculos #3: Parque Infantil.

Las pruebas han comenzado con movimientos simples alrededor de los espacios abiertos, comprobando el seguimiento de las trayectorias. Usando el Escenario #1 se crea la primera trayectoria, ilustrada en la Figura 61. Se han obtenido los siguientes resultados en diez iteraciones en las que se pone a prueba el comportamiento de Nav2:

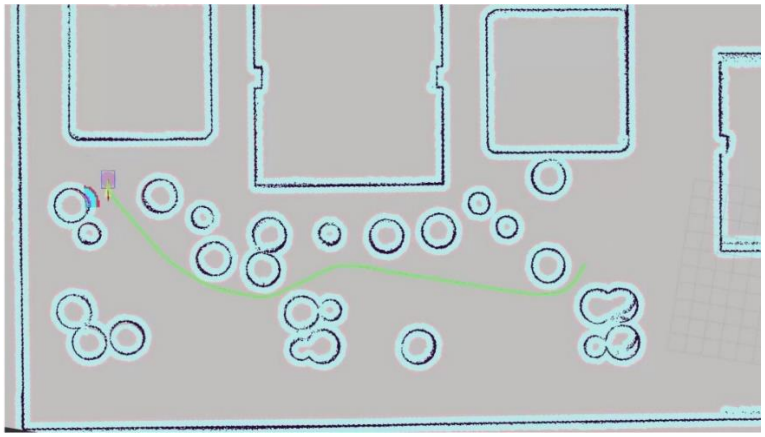

Figura 61. Escenario de Obstáculos #1: Trayectoria de Evasión de Obstáculos #1.

Nº de Iteración	Resultado
1	Logró llegar a la meta usando la acción de evasión de obstáculos.
2	Logró llegar a la meta usando la acción de evasión de obstáculos.
3	Logró llegar a la meta usando la acción de evasión de obstáculos.
4	No logró llegar a la meta debido a un obstáculo en el camino.
5	No logró llegar a la meta debido a un obstáculo en el camino.
6	Logró llegar a la meta sin detenerse por ningún obstáculo.
7	Logró llegar a la meta sin detenerse por ningún obstáculo.
8	Logró llegar a la meta sin detenerse por ningún obstáculo.
9	Logró llegar a la meta sin detenerse por ningún obstáculo.
10	Logró llegar a la meta sin detenerse por ningún obstáculo.

Tabla 1. Evasión de Obstáculos: Resultados del Experimento #1.

Se han obtenido nueve iteraciones en las que el robot logró llegar a la meta, de las cuales solo en tres requirió del Recovery Behavior para desviar al robot del encuentro con un obstáculo. En estos tres casos, esto ocurrió en el punto central en la que dos árboles estaban bastante cerca el uno del otro. Sin embargo, nunca fue requerido esta acción con el árbol al final del trayecto a pesar de pasar bastante cerca de este en todos los casos. En la iteración en la que no logró alcanzar la meta, el robot quedó atrapado en este punto intermedio mencionado anteriormente. En la Figura 62 se han ilustrado dos gráficas, una en la que se muestra el porcentaje de iteraciones en las que el robot alcanza la meta y la segunda en la que se indica el porcentaje de iteraciones que se hizo uso de la subrama de evasión de obstáculos.

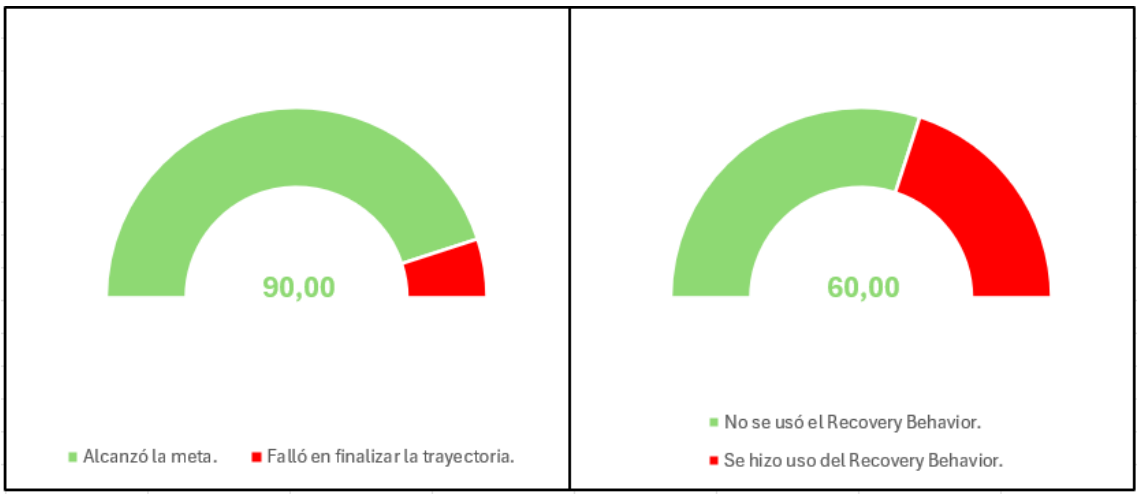

Figura 62. Gráfica de Resultados: Experimento #1 de Evasión de Obstáculos.

Siguiendo con el mismo escenario, se ha creado la segunda trayectoria mostrada en la Figura 63. Esta trayectoria representa un mayor reto para el robot ya que es más larga,

presenta más obstáculos y tiene curvas más pronunciadas. Repitiendo el caso anterior, se realizaron diez iteraciones en las que se comprueba la efectividad del movimiento:

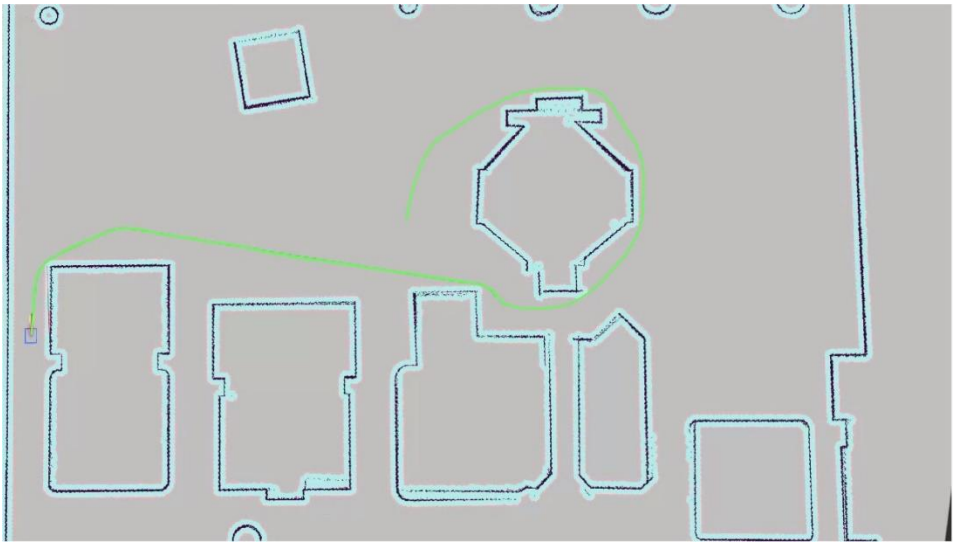

Figura 63. Escenario de Obstáculos #1: Trayectoria de Evasión de Obstáculos #2.

Nº de Iteración	Resultado
1	Logró llegar a la meta sin detenerse por ningún obstáculo.
2	Logró llegar a la meta usando la acción de evasión de obstáculos.
3	Logró llegar a la meta usando la acción de evasión de obstáculos.
4	Logró llegar a la meta sin detenerse por ningún obstáculo.
5	Logró llegar a la meta sin detenerse por ningún obstáculo.
6	Logró llegar a la meta sin detenerse por ningún obstáculo.
7	Logró llegar a la meta usando la acción de evasión de obstáculos.
8	Logró llegar a la meta sin detenerse por ningún obstáculo.
9	Logró llegar a la meta usando la acción de evasión de obstáculos.
10	Logró llegar a la meta usando la acción de evasión de obstáculos.

Tabla 2. Evasión de Obstáculos: Resultados del Experimento #2.

Como se esperaba, al tener un mayor número de curvas, esta trayectoria ha aumentado el porcentaje de iteraciones en la que se ha requerido que el Recovery Behavior tome el control en diferentes puntos del recorrido. Aun así, dado que los obstáculos tenían pleno espacio entre ellos el robot logró completar el trayecto en todas las iteraciones. Adicionalmente, con esta experiencia se comprobó que el planificador busca reducir la distancia de recorrido al acercar la trayectoria a los obstáculos sin importar cuanto espacio libre haya alrededor. Este comportamiento puede ser contraproducente, pero en esta experiencia no ha sido negativo.

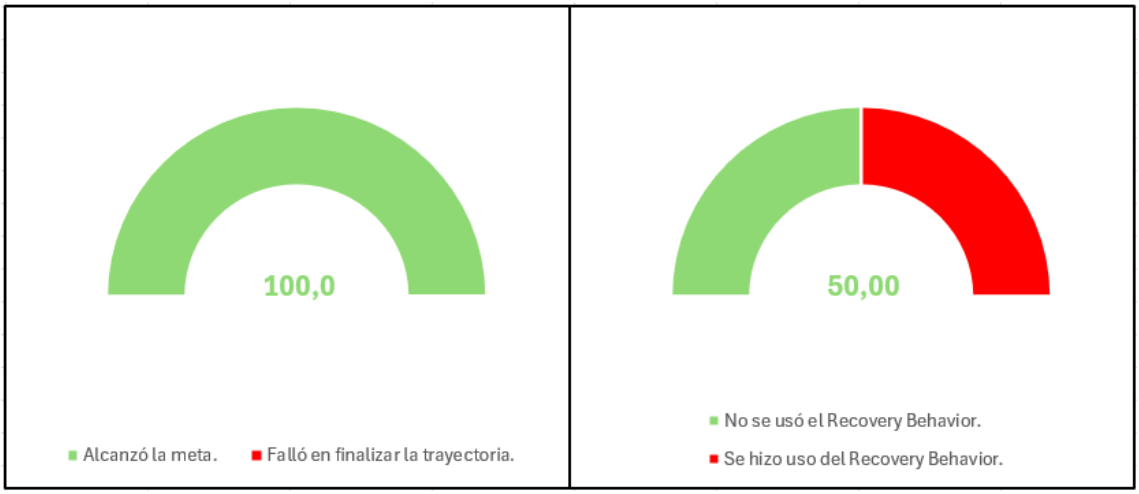

Figura 64. Gráfica de Resultados: Experimento #2 de Evasión de Obstáculos.

La tercera experiencia se realizará en el escenario #2, el cual representará una versión extendida de la primera experiencia. La trayectoria inicial se aprecia en la Figura 65, en la que se ve como el robot deberá atravesar zonas de espacio reducido entre hileras de árboles. Nuevamente se realizan diez iteraciones para comprobar el comportamiento independiente de Nav2.

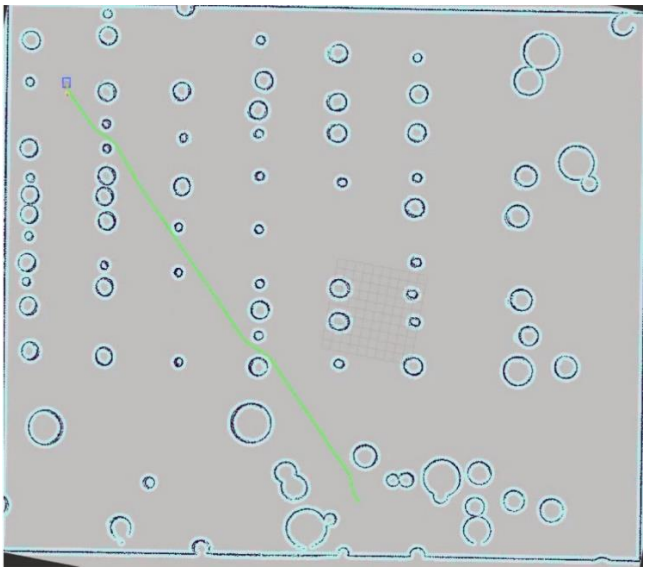

Figura 65. Escenario de Obstáculos #2: Trayectoria de Evasión de Obstáculos #3.

Nº de Iteración	Resultado
1	Logró llegar a la meta sin detenerse por ningún obstáculo.
2	Logró llegar a la meta sin detenerse por ningún obstáculo.
3	No logró llegar a la meta debido a un obstáculo en el camino.
4	No logró llegar a la meta debido a un obstáculo en el camino.
5	No logró llegar a la meta debido a un obstáculo en el camino.

6	Logró llegar a la meta sin detenerse por ningún obstáculo.
7	Logró llegar a la meta sin detenerse por ningún obstáculo.
8	Logró llegar a la meta sin detenerse por ningún obstáculo.
9	Logró llegar a la meta sin detenerse por ningún obstáculo.
10	Logró llegar a la meta sin detenerse por ningún obstáculo.

Tabla 3. Evasión de Obstáculos: Resultados del Experimento #3.

Se aprecia en esta experiencia, e ilustrado en la Figura 66, que en todos los casos en los que se ha usado el Recovery Behavior, este no ha logrado sobrepasar el obstáculo y retomar el recorrido. Sin embargo, en todos los casos al pasar entre las hileras de árboles el robot evadía el árbol apropiadamente, pero la zona trasera con las oscilaciones se acercaba bastante al obstáculo y activaba el sistema de recuperación. Cabe destacar que en estos casos no ocurría ningún choque y el robot hubiese podido avanzar apropiadamente hasta la meta, pero como el algoritmo no encontraba forma de salir de la situación ya que intentaba evitar el obstáculo con movimientos de retroceso que en este caso solo lo acercaba más al obstáculo.

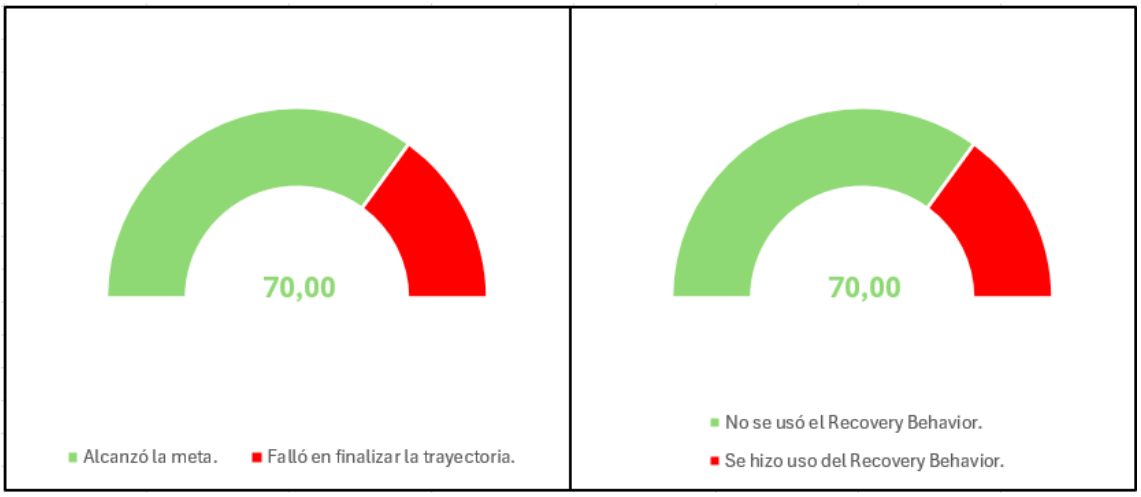

Figura 66. Gráfica de Resultados: Experimento #3 de Evasión de Obstáculos.

La siguiente trayectoria ocurre en la escena #3, en la que se aprovecha la edificación en el borde del mapa para comprobar nuevamente el comportamiento del robot en curvas cercanas a obstáculos de dimensiones considerables. Como se mencionó anteriormente, en este mapa existe un mayor espacio libre, por lo que se prestará atención a como el stack aprovecha este espacio. Los resultados, así como la trayectoria, se muestran a continuación:

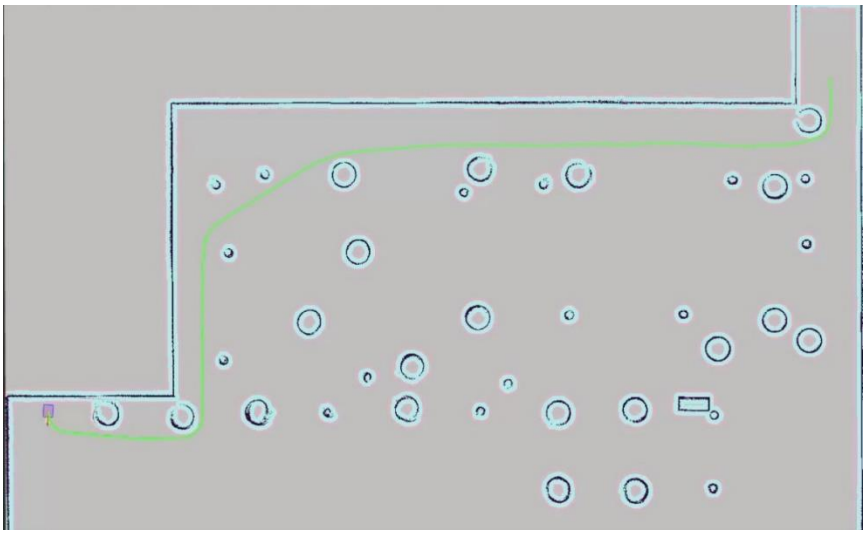

Figura 67. Escenario de Obstáculos #3: Trayectoria de Evasión de Obstáculos #4.

Nº de Iteración	Resultado
1	Logró llegar a la meta sin detenerse por ningún obstáculo.
2	Logró llegar a la meta sin detenerse por ningún obstáculo.
3	Logró llegar a la meta sin detenerse por ningún obstáculo.
4	Logró llegar a la meta sin detenerse por ningún obstáculo.
5	Logró llegar a la meta sin detenerse por ningún obstáculo.
6	No logró llegar a la meta debido a un obstáculo en el camino.
7	Logró llegar a la meta sin detenerse por ningún obstáculo.
8	Logró llegar a la meta sin detenerse por ningún obstáculo.
9	Logró llegar a la meta sin detenerse por ningún obstáculo.
10	Logró llegar a la meta sin detenerse por ningún obstáculo.

Tabla 4. Evasión de Obstáculos: Resultados del Experimento #4.

Alcanzando nueve iteraciones en las que el robot logra alcanzar la meta, se han confirmado aspectos observables en experiencias anteriores. Nuevamente, el robot ha elegido giros en los bordes de los obstáculos en vez de mantener distancias más seguras. Se entiende que esto es debido a parámetros elegidos, pero claramente no se están diseñando las trayectorias óptimas para alcanzar las metas. Adicionalmente, se observa que el Recovery Behavior nuevamente ha fallado, aunque esto solo ocurrió en una iteración por lo que se requeriría más datos para sacar conclusiones claras.

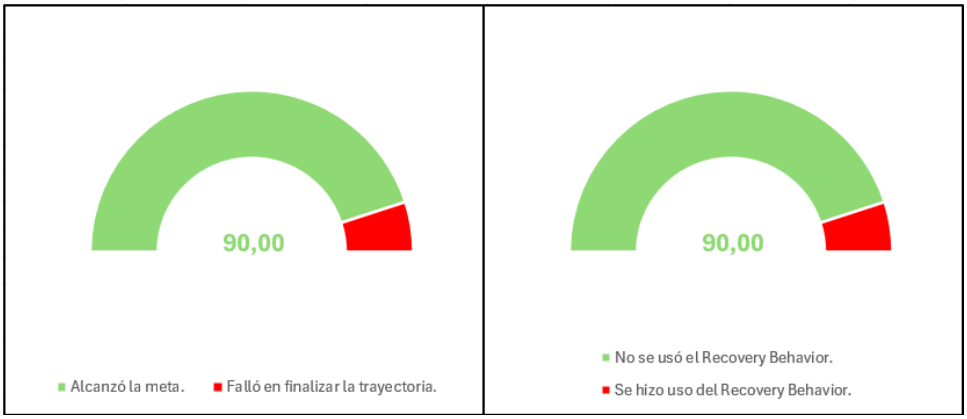

Figura 68. Gráfica de Resultados: Experimento #4 de Evasión de Obstáculos.

Teniendo un total de cuarenta iteraciones, es posible pasar a realizar una comparación global del uso de Nav2 para un robot con modelo Ackermann. Observamos en la Figura 69 un 87,5% de aciertos globales, lo cual es considerablemente alto. Adicionalmente, existe un 20% de iteraciones en las que la subrama de recuperación logró retomar el recorrido hacia la meta, lo cual, comparado con el 12,5% en el que no lo logró, se ve que en la mayoría de los casos la subrama logra evadir el objeto. Considerando todos los casos en los que se usó el Recovery Behavior, se ha tenido un 61.54% de iteraciones en las que finalmente se ha alcanzado la meta.

Figura 69. Gráfica de Resultados: Todos los Experimentos de Evasión de Obstáculos.

5.3. Aparcamiento con el Modelo Virtual

Las pruebas realizadas para el algoritmo de aparcamiento se han dividido en dos partes. La parte inicial es en un escenario de CoppeliaRobotics, el cual simula un aparcamiento con doce puestos disponibles que ha permitido realizar experimentos durante el desarrollo del algoritmo. Como se mencionó anteriormente, se ha usado el escenario del aparcamiento público en la Ciudad de la Justicia como base para la creación de un aparcamiento de dimensiones apropiadas. El resultado final es el siguiente:

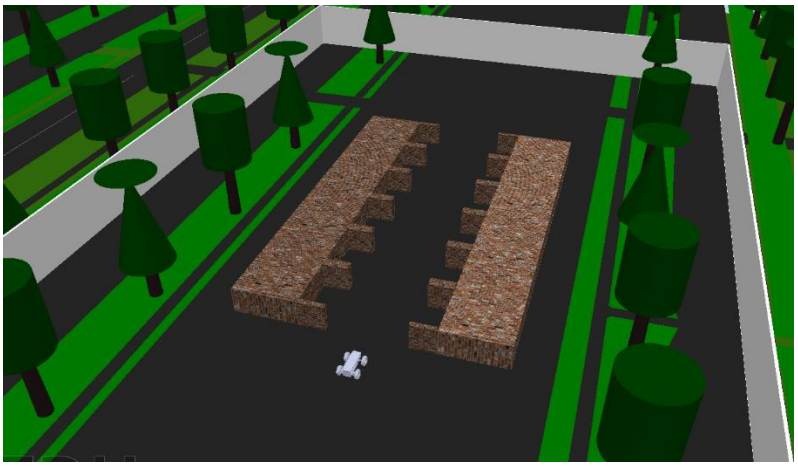

Figura 70. Escenario de Aparcamiento Virtual, CoppeliaRobotics.

Se ha iniciado con una prueba inicial en la que todo el aparcamiento está totalmente vacío. Las diez primeras iteraciones en las que el robot elige el lado izquierdo han dado los siguientes resultados:

Nº de Iteración	Resultado
1	Tocó la pared lateral, por lo que no se completó la maniobra.
2	Tocó la pared lateral, por lo que no se completó la maniobra.
3	Tocó la pared frontal, por lo que no se completó la maniobra.
4	Completó la maniobra, con una posición final ligeramente desviada.
5	Completó la maniobra, con una posición final correcta.
6	Completó la maniobra, quedando parcialmente fuera del puesto.
7	Completó la maniobra, con una posición final correcta.
8	Completó la maniobra, con una posición final correcta.
9	Completó la maniobra, quedando parcialmente fuera del puesto.
10	Completó la maniobra, quedando parcialmente fuera del puesto.

Tabla 5. Aparcamiento Virtual: Resultados del Experimento #1.

Con tres iteraciones en las que el algoritmo fue capaz de finalizar la maniobra, se toman los resultados como satisfactorios con un 70% de iteraciones positivas. Cabe destacar que,

en este apartado, el índice de fallos por no detectar el puesto es mínimo, por lo que no se diferenciará este caso de los choques con la pared. Se ilustran en la Figura 71 los resultados de esta experiencia.

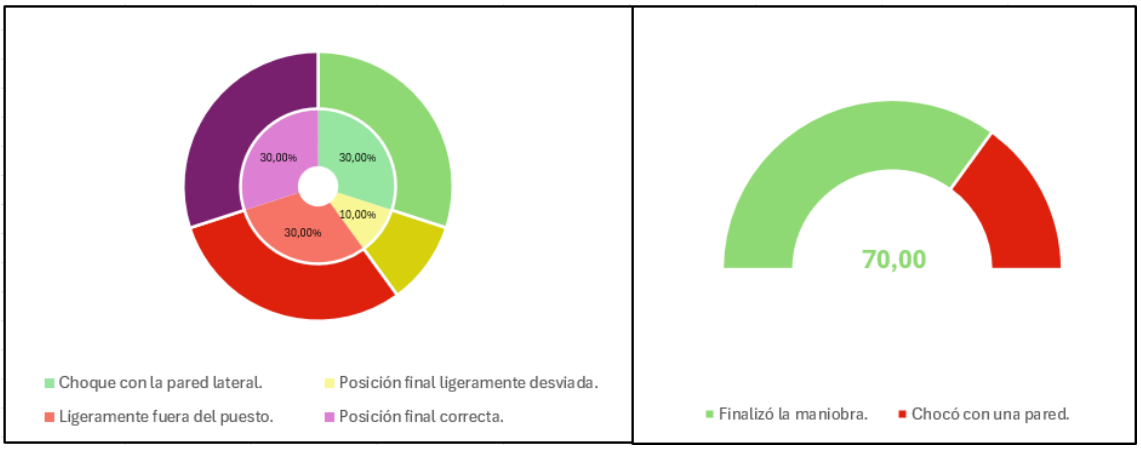

Figura 71. Gráfica de Resultados: Experimento #1 de Aparcamiento Virtual.

Se han realizado otras diez iteraciones en las que el robot elige el lado derecho. Los resultados son los siguientes:

Nº de Iteración	Resultado
1	Completó la maniobra, quedando parcialmente fuera del puesto.
2	Completó la maniobra, quedando parcialmente fuera del puesto.
3	Completó la maniobra, con una posición final ligeramente desviada.
4	El algoritmo no detectó el puesto debido a las oscilaciones del robot.
5	Tocó la pared trasera, por lo que no se completó la maniobra.
6	Tocó la pared trasera, por lo que no se completó la maniobra.
7	Completó la maniobra, con una posición final correcta.
8	Tocó la pared lateral, por lo que no se completó la maniobra.
9	Completó la maniobra, con una posición final correcta.
10	Completó la maniobra, con una posición final correcta.

Tabla 6. Aparcamiento Virtual: Resultados del Experimento #2.

Nuevamente se aprecia una mayoría de resultados satisfactorios, con un total del 60% de aciertos. En este caso, se ha obtenido una iteración en la que el algoritmo falló en detectar el puesto vacante. En esta iteración se percibió un fallo inicial en la que la información del láser estaba desfasada del mapa, así como movimientos oscilatorios más notorios de lo usual que introducen interferencia con los datos del sensor. Los resultados generales se aprecian en la Figura 72.

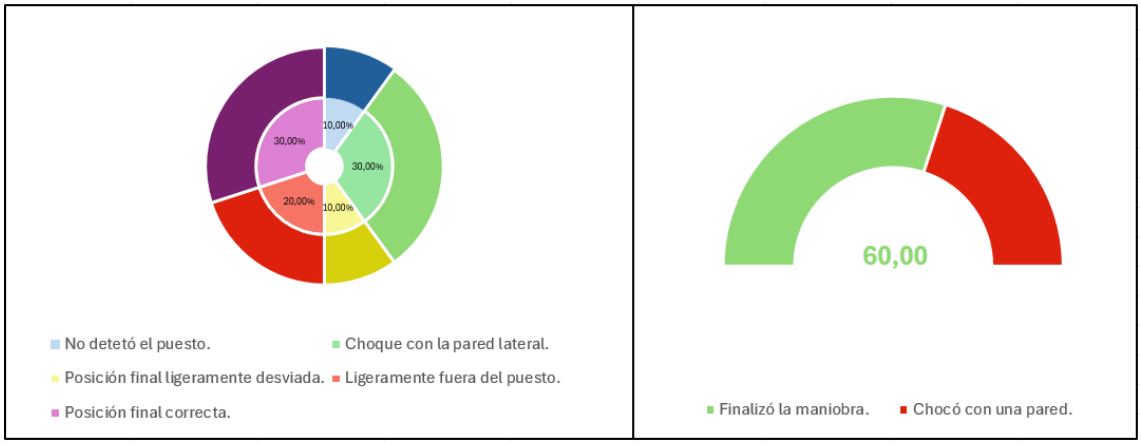

Figura 72. Gráfica de Resultados: Experimento #2 de Aparcamiento Virtual.

En la siguiente experiencia se han ocupado ambos puestos al inicio del recorrido comprobando que la detección de puestos libres funcione correctamente. Se han realizado tantas iteraciones hasta que se tengan diez experiencias de cada lado. Los resultados para el puesto del lado izquierdo se encuentran en la siguiente tabla:

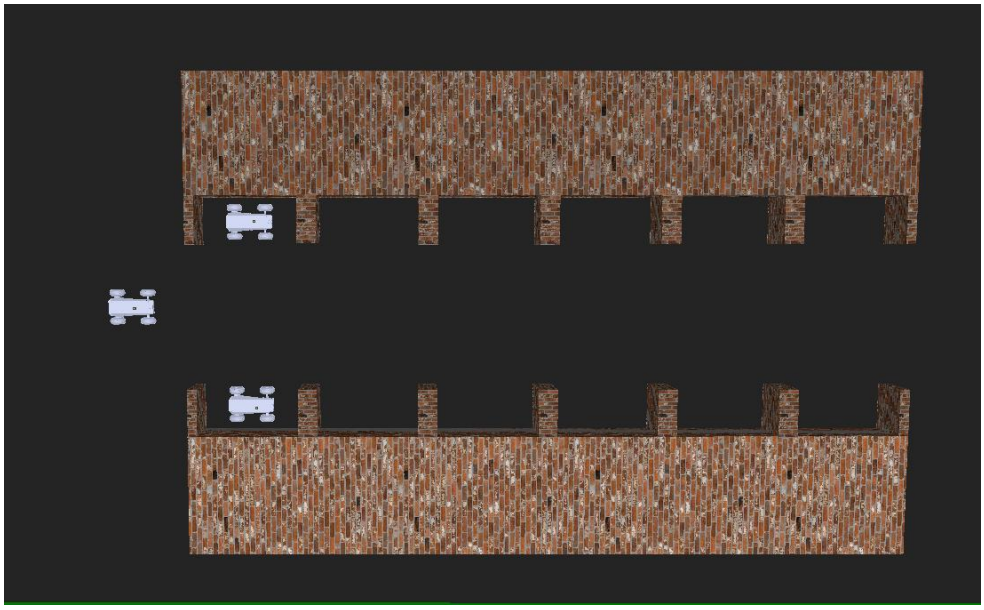

Figura 73. Aparcamiento Virtual Experiencia #3: Primeros Puestos Ocupados.

Nº de Iteración	**Resultado**
1	Tocó la pared lateral, por lo que no se completó la maniobra.
2	Completó la maniobra, quedando parcialmente fuera del puesto.
3	Tocó la pared lateral, por lo que no se completó la maniobra.
4	Completó la maniobra, con una posición final correcta.
5	Completó la maniobra, quedando parcialmente fuera del puesto.
6	Completó la maniobra, con una posición final ligeramente desviada.

7	Completó la maniobra, con una posición final correcta.
8	Completó la maniobra, quedando parcialmente fuera del puesto.
9	Completó la maniobra, con una posición final correcta.
10	Completó la maniobra, con una posición final ligeramente desviada.

Tabla 7. Aparcamiento Virtual: Resultados del Experimento #3.

Nuevamente se destacan resultados satisfactorios con un 80% de aciertos. Se ha detectado el puesto en todos los casos, y se han obtenido fallos chocando con paredes, en ambos casos con el movimiento de retroceso al entrar en el puesto. Los resultados se ilustran en la Figura 74.

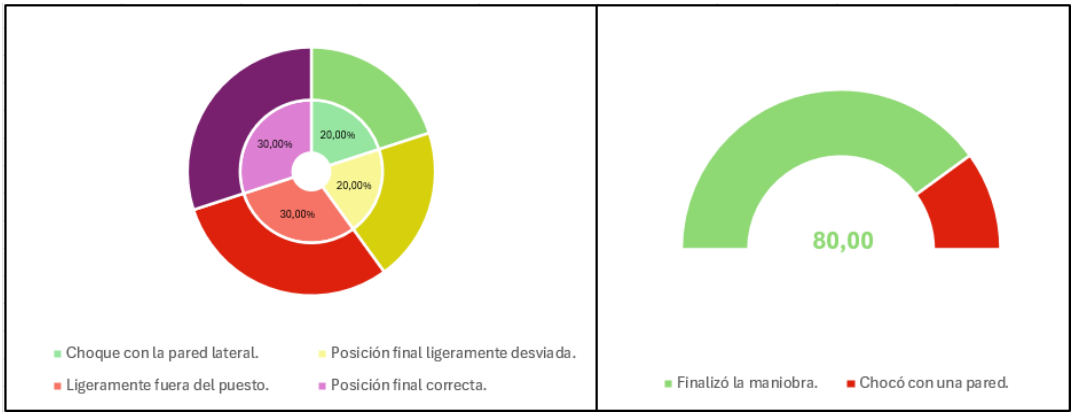

Figura 74. Gráfica de Resultados: Experimento #3 de Aparcamiento Virtual.

La siguiente tabla muestra los resultados para el puesto del lado derecho.

Nº de Iteración	**Resultado**
1	Tocó la pared lateral, por lo que no se completó la maniobra.
2	Tocó la pared lateral, por lo que no se completó la maniobra.
3	Tocó la pared lateral, por lo que no se completó la maniobra.
4	Completó la maniobra, con una posición final correcta.
5	Completó la maniobra, quedando parcialmente fuera del puesto.
6	Tocó la pared lateral, por lo que no se completó la maniobra.
7	Completó la maniobra, quedando parcialmente fuera del puesto.
8	Completó la maniobra, con una posición final ligeramente desviada.
9	Completó la maniobra, con una posición final correcta.
10	Completó la maniobra, con una posición final ligeramente desviada.

Tabla 8. Aparcamiento Virtual: Resultados del Experimento #4.

Nuevamente se han obtenido un 60% en las maniobras con el lado derecho. En este caso, el 40% de fallos ha sido en su totalidad por choques con el entorno. A parte de esto, no

se presentan datos nuevos en estas iteraciones, en comparación con las experiencias anteriores. En la Figura 75 se aprecian las gráficas de los resultados.

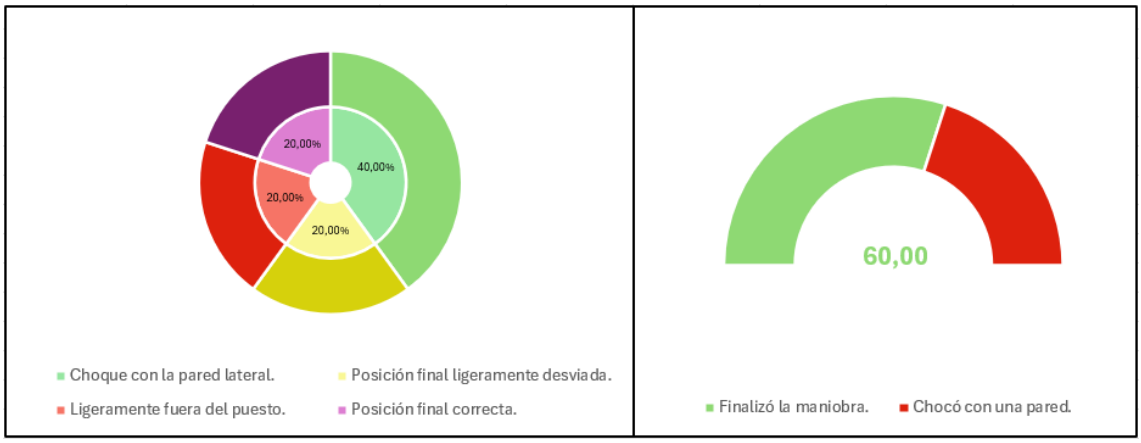

Figura 75. Gráfica de Resultados: Experimento #4 de Aparcamiento Virtual.

Se pasan a realizar ahora dos experiencias con distribuciones de puestos ocupados al azar. En ambos casos, se ha dejado un puesto libre en cada lado a alturas diferentes con el fin de ver el comportamiento en caso de que falle en detectar el primer puesto que encuentre. Este fallo nunca ocurrió en esta experiencia y hubo una detección del puesto del 100%. Ambos mapas y sus correspondientes resultados se encuentran a continuación.

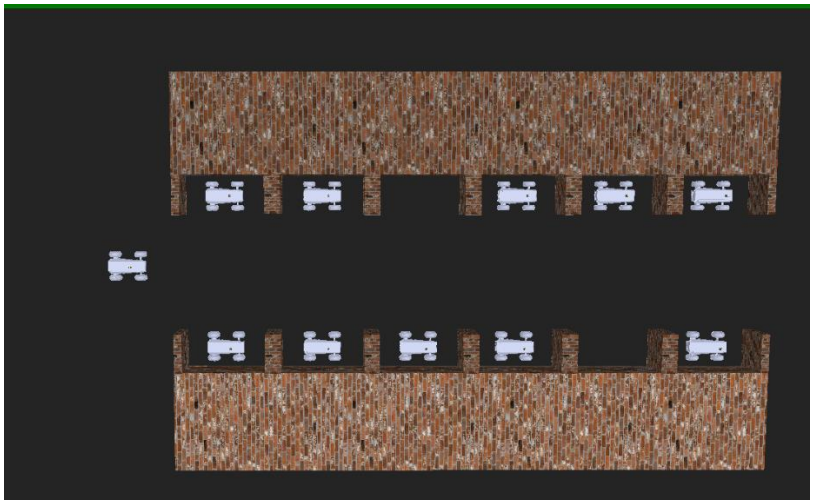

Figura 76. Aparcamiento Virtual Experiencia #5: Distribución Aleatoria #1.

Nº de Iteración	Resultado
1	Completó la maniobra, con una posición final correcta.
2	Completó la maniobra, con una posición final ligeramente desviada.
3	Completó la maniobra, con una posición final ligeramente desviada.
4	Tocó la pared lateral, por lo que no se completó la maniobra.
5	Tocó la pared lateral, por lo que no se completó la maniobra.

6	Completó la maniobra, con una posición final correcta.
7	Completó la maniobra, con una posición final correcta.
8	Tocó la pared lateral, por lo que no se completó la maniobra.
9	Tocó la pared lateral, por lo que no se completó la maniobra.
10	Completó la maniobra, con una posición final ligeramente desviada.

Tabla 9. Aparcamiento Virtual: Resultados del Experimento #5.

En esta experiencia se ha obtenido un 60% de aciertos en el lado izquierdo. Sin embargo, en todas estas iteraciones el robot ha quedado totalmente dentro del espacio de aparcamiento, lo cual se acerca más al objetivo. Se ilustran en la Figura 77 los resultados mencionados.

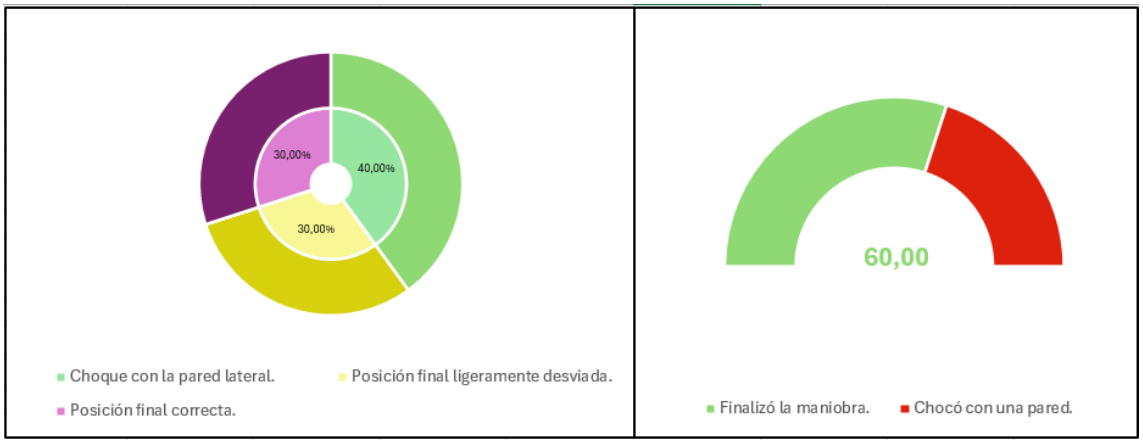

Figura 77. Gráfica de Resultados: Experimento #5 de Aparcamiento Virtual.

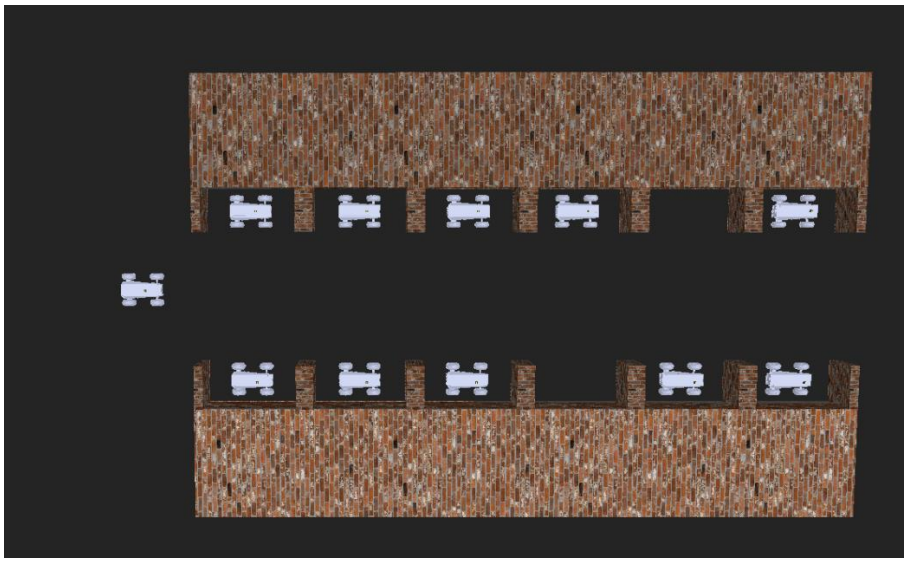

Figura 78. Aparcamiento Virtual Experiencia #6: Distribución Aleatoria #2.

Nº de Iteración	Resultado
1	Completó la maniobra, quedando parcialmente fuera del puesto.
2	Completó la maniobra, quedando parcialmente fuera del puesto.
3	Completó la maniobra, con una posición final ligeramente desviada.
4	Tocó la pared lateral, por lo que no se completó la maniobra.
5	Completó la maniobra, con una posición final ligeramente desviada.
6	Completó la maniobra, con una posición final correcta.
7	Completó la maniobra, quedando parcialmente fuera del puesto.
8	Completó la maniobra, con una posición final ligeramente desviada.
9	Completó la maniobra, quedando parcialmente fuera del puesto.
10	Completó la maniobra, con una posición final ligeramente desviada.

Tabla 10. Aparcamiento Virtual: Resultados del Experimento #6.

Se ha alcanzado unos resultados altamente satisfactorios del 90%, incluyendo un 50% de iteraciones en las que el robot se encuentra completamente dentro del espacio válido.

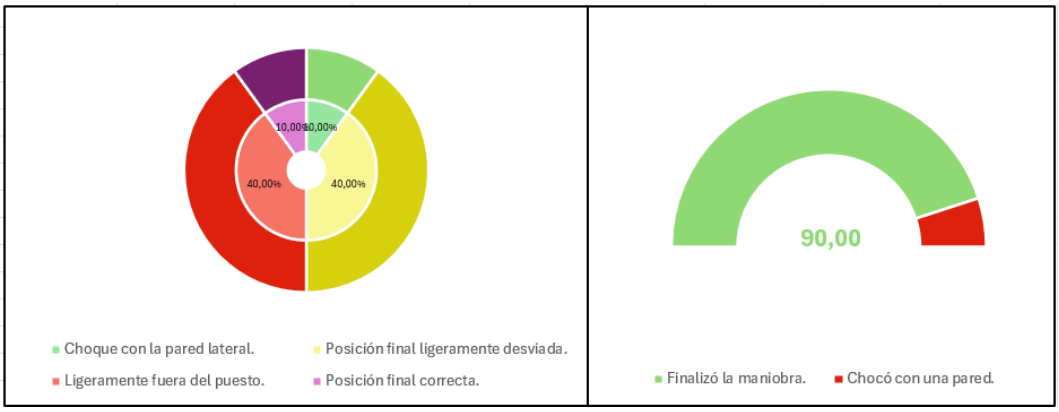

Figura 79. Gráfica de Resultados: Experimento #6 de Aparcamiento Virtual.

Pasaremos ahora a recopilar las sesenta iteraciones para obtener los resultados globales. Al no distinguir entre lado izquierdo o derecho, los resultados son los siguientes:

Resultados	Nº de Iteraciones
No detectó el puesto.	1
Choque con la pared lateral.	17
Detectó el puesto, con posición final ligeramente desviada.	13
Detectó el puesto, con posición final ligeramente fuera del espacio de aparcamiento.	14
Detectó el puesto, con posición final correcta.	15
Total	60

Tabla 11. Aparcamiento Virtual: Resultados Globales.

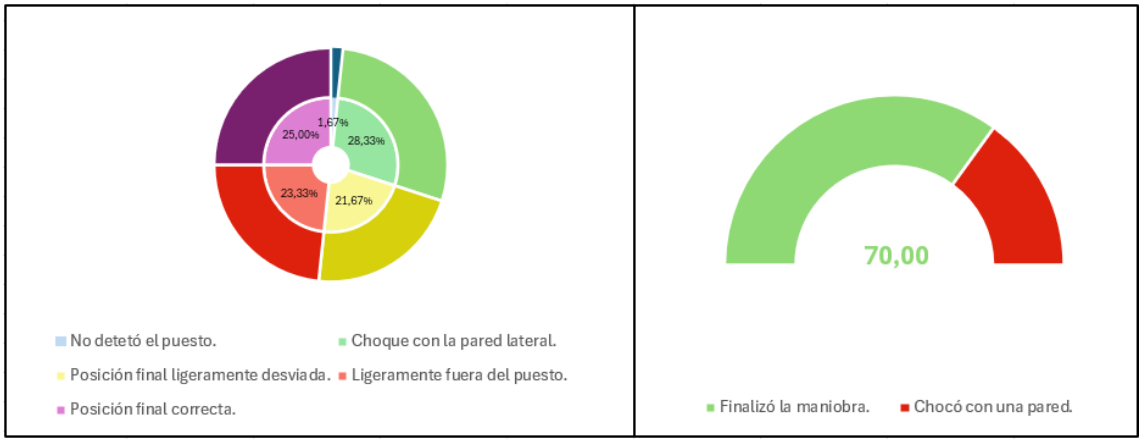

Figura 80. Gráfica de Resultados: Todos los Experimentos del Aparcamiento.

Finalmente se obtiene un 70% de aciertos globales, lo cual es suficiente para considerar las pruebas como superadas. Con relación a este apartado, estos experimentos se han tomado en cuenta como una etapa de seguridad antes de probar el algoritmo con el robot real. Como se ha comprobado dentro de las estadísticas anteriores el algoritmo ha devuelto un porcentaje de éxitos suficientemente alto para poder pasar a la siguiente etapa, aparcamiento en un ámbito real.

5.4. Aparcamiento con el Modelo Real

Como se ha mencionado anteriormente en este documento los experimentos de aparcamiento se han realizado en un pasillo interior en la E.T.S.I informática debido a sus características físicas. En la Figura 81 se encuentra una ilustración simplificada de la sección del pasillo usada para los experimentos. Con Nav2 el robot tiene programado moverse hasta el punto verde, siendo los rectángulos rojos los posibles espacios de aparcamiento que se tomarán en cuenta.

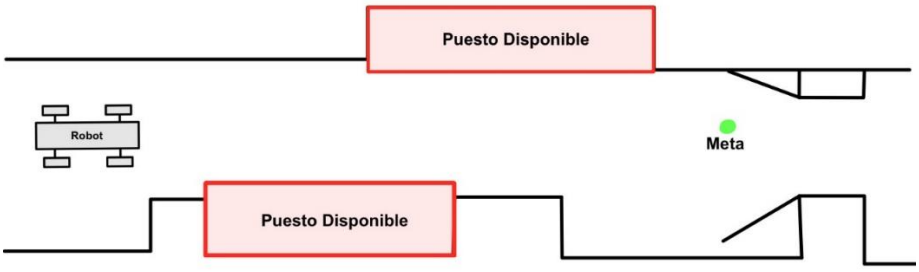

Figura 81. Diseño Simplificado del Pasillo.

Se ha iniciado con el puesto inferior del lado derecho del robot. En total se han realizado trece iteraciones del mismo experimento, los resultados son los siguientes:

Nº de Iteración	Resultado
1	Detecta el puesto, pero toca la pared lateral.
2	Detecta el puesto, posición final correcta.
3	Detecta el puesto, posición final correcta.
4	Detecta el puesto, posición final correcta.
5	No detectó el puesto, siguió de largo.
6	Detecta el puesto, pero toca la pared lateral.
7	Detecta el puesto, posición final ligeramente desviada.
8	Detecta el puesto, posición final ligeramente desviada.
9	No detectó el puesto, pero detectó el siguiente, del lado izquierdo.
10	Detecta el puesto, posición final ligeramente fuera del espacio de aparcamiento.
11	Detecta el puesto, posición final muy cerca de la pared.
12	Detecta el puesto, posición final correcta.
13	Detecta el puesto, posición final correcta.

Tabla 12. Aparcamiento Real: Resultados del Experimento #1.

Dentro de estas pruebas se puede considerar dos tipos de resultados positivos: en los que el algoritmo detecta el puesto vacío e inicia la maniobra de aparcamiento y en los que la maniobra de aparcamiento finaliza sin choques con el entorno. En la Figura 82 podemos observar uno de estos resultados, en los que el fallo ocurrió al entrar en contacto con la pared.

Figura 82. Robot en Contacto con Pared Lateral.

Considerando esto, en once iteraciones logró detectar correctamente el puesto de los cuales nueve lograron terminar las maniobras, uno de ellos bastante cerca de la pared. Dentro de los resultados positivos se destacan cuando el robot tiene posiciones finales correctas y cuando tienen desvíos. Se ilustra en la Figura 83 la posición final deseada

mientras que en la Figura 84 se aprecia el resultado de la iteración número diez en la que el robot queda ligeramente fuera del espacio de estacionamiento.

Figura 83. Posición Final Correcta.

Figura 84. Posición Final Fuera del Aparcamiento.

De los nueve resultados completamente exitosos se tiene cinco posiciones ligeramente desviadas de las cuales una se encuentra ligeramente fuera del aparcamiento y otra está bastante cerca de la pared. De esta forma, se obtienen cuatro iteraciones en las que la posición final es plenamente correcta. En la Figura 85 se aprecia un gráfico recopilando estos datos:

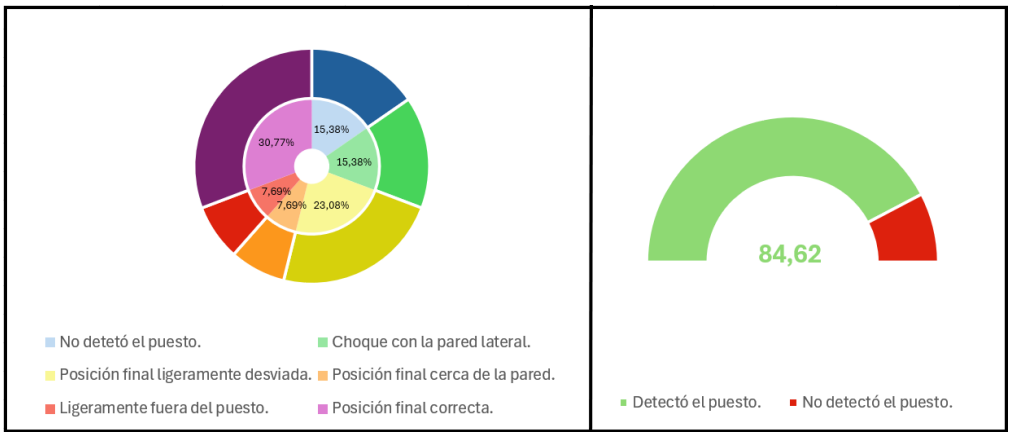

Figura 85. Gráfica de Resultados: Experimento #1 de Aparcamiento.

Finalmente, con un porcentaje de éxito global del 84.6% en detección del puesto, se interpretan los resultados como satisfactorios. Debido a que los puestos son bastante más largos que el robot, es posible colocar objetos en el medio que permitan reiniciar el contador y aún tener espacio suficiente para aparcar. Para este fin, se ha colocado un objeto a un metro de distancia de la esquina inicial y se han realizado diez iteraciones de la segunda prueba con los siguientes resultados:

No de Iteración	Resultado
1	Detecta el puesto, posición final ligeramente fuera del espacio de aparcamiento.
2	No detectó el puesto, siguió de largo.
3	Detecta el puesto, posición final ligeramente desviada.
4	Detecta el puesto, posición final ligeramente desviada.
5	Detecta el puesto, posición final correcta.
6	Detecta el puesto, posición final correcta.
7	Detecta el puesto, posición final ligeramente desviada.
8	Detecta el puesto, posición final ligeramente desviada.
9	Detecta el puesto, posición final correcta.
10	Detecta el puesto, posición final ligeramente desviada.

Tabla 13. Aparcamiento Real: Resultados del Experimento #2.

Siguiendo la lógica de la prueba anterior, se obtuvieron nueve iteraciones en las que se detecta el puesto sin ninguna iteración con choques. De estos resultados positivos, uno de ellos quedó fuera del puesto, cinco iteraciones con posiciones ligeramente desviadas y tres con posiciones finales correctas. Estos resultados se aprecian en la Figura 86.

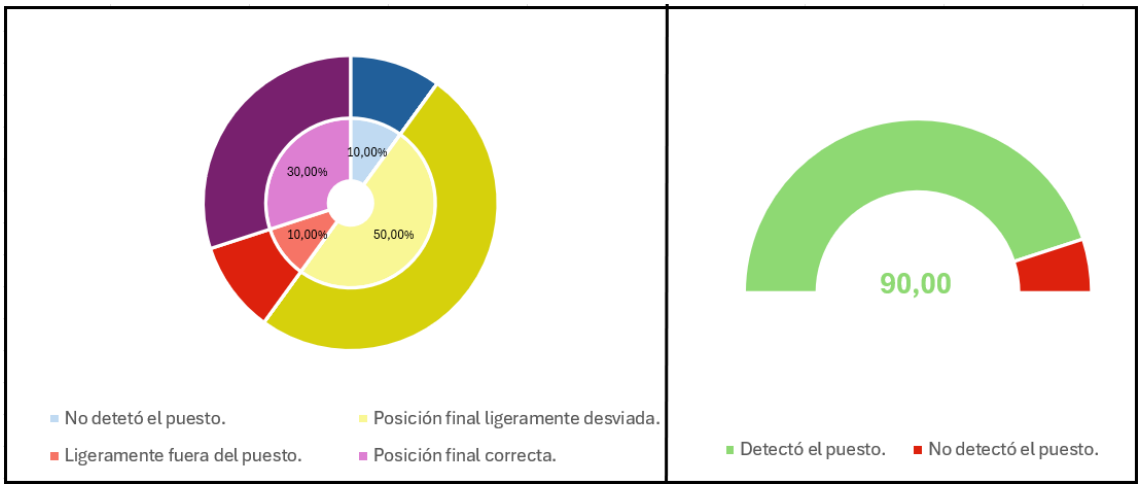

Figura 86. Gráfica de Resultados: Experimento #2 de Aparcamiento.

Con un 90% de éxito de detección del puesto con obstáculos de por medio y un 86.9% de éxito en ambas pruebas queda claro que los resultados para puestos del lado derecho son satisfactorios, por lo que se pasará al puesto del lado izquierdo. Inicialmente, con el puesto totalmente libre, se realizaron diez iteraciones:

Nº de Iteración	Resultado
1	Detecta el puesto, posición final ligeramente desviada.
2	No detectó el puesto, siguió de largo.
3	Detecta el puesto, pero toca la pared lateral.
4	Detecta el puesto, posición final ligeramente desviada.
5	Detecta el puesto, posición final ligeramente desviada.
6	Detecta el puesto, posición final correcta.
7	Detecta el puesto, posición final ligeramente desviada.
8	Detecta el puesto, posición final correcta.
9	Detecta el puesto, posición final muy cerca de la pared.
10	Detecta el puesto, posición final ligeramente desviada.

Tabla 14. Aparcamiento Real: Resultados del Experimento #3.

Nuevamente se ha obtenido una mayoría de resultados positivos, con un total de nueve. De estos, ocho han logrado terminar el aparcamiento, obteniendo seis iteraciones con posiciones desviadas, una de ellas bastante cerca de la pared y solo con dos iteraciones con posiciones finales correctas. Se ilustran estos resultados en la Figura 87.

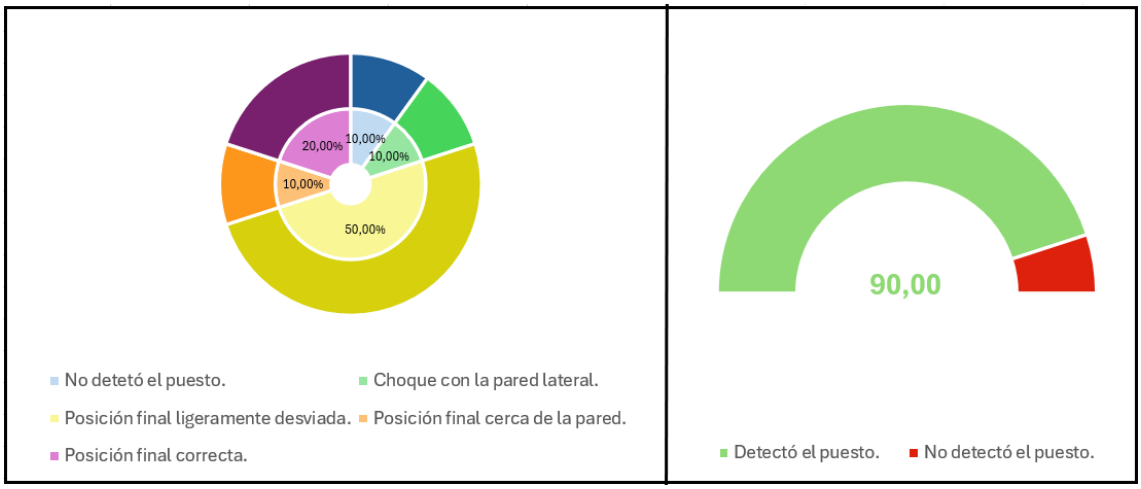

Figura 87. Gráfica de Resultados: Experimento #3 de Aparcamiento.

En esta prueba se ha obtenido un 90% de aciertos globales, lo cual sigue siendo satisfactorio. Para la prueba con un objeto de por medio se realizaron las siguientes seis iteraciones:

Nº de Iteración	Resultado
1	Detecta el puesto, pero toca la pared lateral.
2	Detecta el puesto, posición final ligeramente desviada.
3	Detecta el puesto, pero toca la pared lateral.
4	Detecta el puesto, posición final correcta.
5	Detecta el puesto, posición final correcta.
6	Detecta el puesto, posición final ligeramente fuera del espacio de aparcamiento.

Tabla 15. Aparcamiento Real: Resultados del Experimento #4.

A pesar de que en los seis casos el algoritmo logró detectar el puesto correctamente, en dos ocasiones el robot chocó con la pared, obteniendo cuatro resultados con posiciones finales válidas. De estos, uno tiene un desvío ligero, uno sale del puesto y dos tienen posiciones correctas. Veamos la gráfica en la Figura 88:

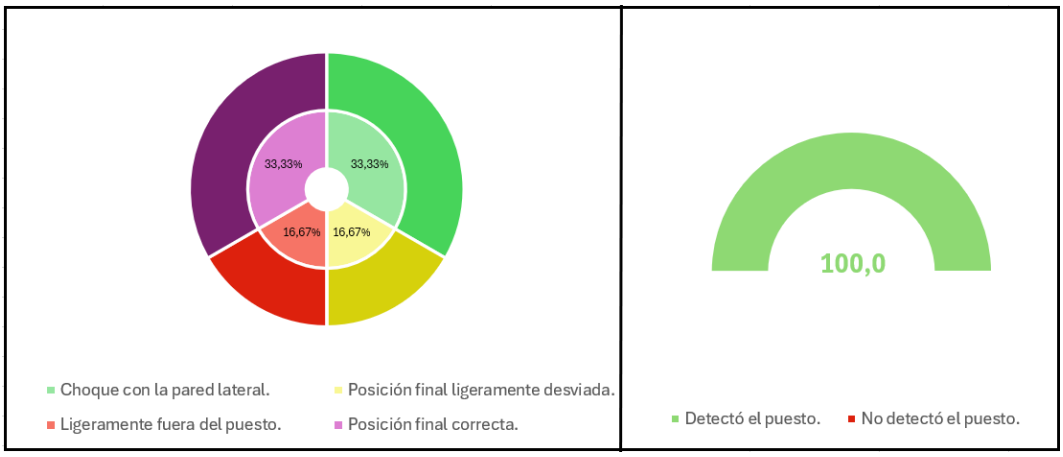

Figura 88. Gráfica de Resultados: Experimento #4 de Aparcamiento.

Finalmente se han compilado los resultados de todas las pruebas de la siguiente forma:

Resultados	Nº de Iteraciones
No detectó el puesto.	4
Choque con la pared lateral.	5
Detectó el puesto, con posición final ligeramente desviada.	13
Detectó el puesto, con posición final muy cerca de la pared.	2
Detectó el puesto, con posición final ligeramente fuera del espacio de aparcamiento.	3
Detectó el puesto, con posición final correcta.	12
Total	39

Tabla 16. Aparcamiento Real: Resultados Finales.

En la Figura 89 se han ilustrado estos datos de la siguiente forma:

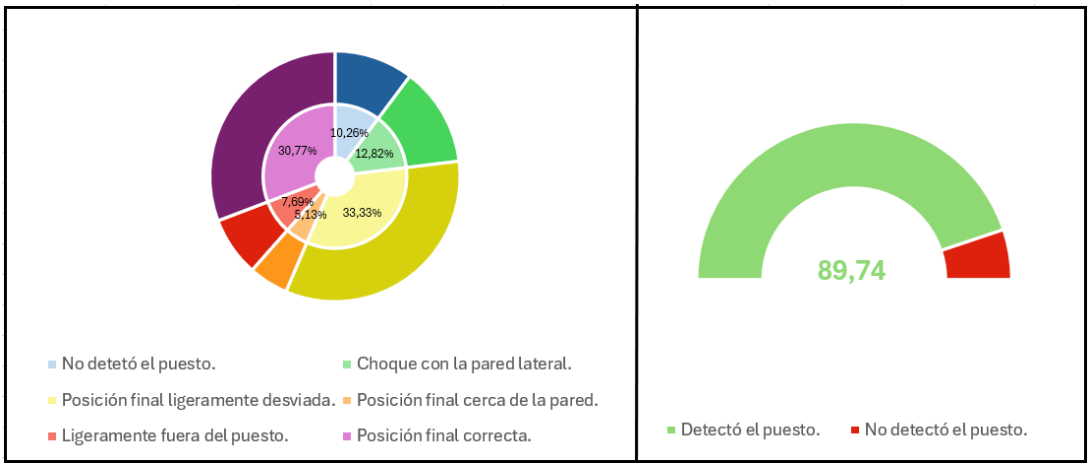

Figura 89.1 Gráfica de Resultados: Todos los Experimentos del Aparcamiento Real.

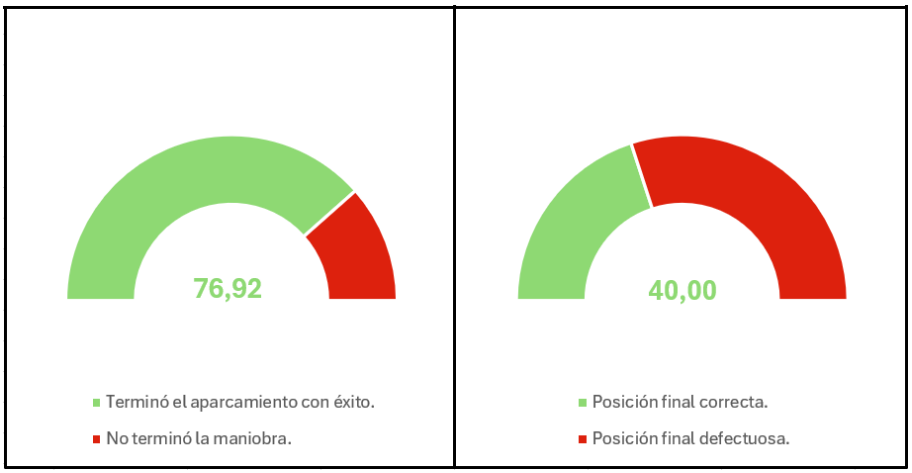

Figura 90.2 Gráfica de Resultados: Todos los Experimentos del Aparcamiento Real.

Se destaca que se ha conseguido un 89,74% de resultados positivos entre los cuatro experimentos, lo cual es un valor bastante alto. Sin embargo, de los treinta resultados positivos en los que se terminó la maniobra, el 40% tienen posiciones finales completamente alineados. Este dato se debe a que la posición inicial del robot es totalmente aleatoria debido a las oscilaciones que realiza el robot cuando Nav2 toma el control.

6. Conclusiones

Con estos resultados altamente satisfactorios se puede considerar que los objetivos del trabajo han sido cumplidos. Inicialmente, se ha logrado diseñar un modelo virtual del robot Hunter 2.0 a través de la plataforma CoppeliaRobotics que realiza movimientos altamente realistas en comparación al modelo real. Seguidamente, se ha brindado al robot con una navegación autónoma capaz de seguir trayectorias sin choques con ningún obstáculo de su entorno y finalmente se ha logrado brindar al robot de la acción de aparcarse en línea.

Dentro de las experiencias con el modelo real, el robot ha demostrado ser ideal para un ambiente de experimentación y desarrollo innovativo con alumnos e investigadores. No solo es bastante amigable y fácil de usar y programar, sino que se adapta a ambientes de cualquier origen y tiene movimientos fluidos que facilitan la programación de navegación. Con respecto a este proyecto, todo contacto con el robot fue positivo y nunca presentó un problema para el desarrollo de los objetivos.

Adicionalmente, se ha visto indicios de que Nav2 presenta complicaciones con el modelo cinemático Ackermann debido a las grandes limitaciones de navegación que estos robots presentan. Se intuye que los desarrolladores de este Stack plantean futuras soluciones para este robot, pero en lo que estas llegan, los futuros alumnos que tengan contacto con este robot deben tomar la libertad de diseñar algoritmos de navegación totalmente nuevos e innovadores para este robot con la finalidad de alcanzar el rendimiento óptimo de estos modelos.

Finalmente, cabe destacar que dentro del proyecto quedó en evidencia el gran desafío que representa la navegación autónoma actualmente. A pesar de conseguir los resultados obtenidos, la robustez del sistema es limitada y se presta a grandes mejoras y cambios. Desde el inicio del trabajo se dejó claro que este serviría como un inicio, los primeros pasos con un robot novedoso que abrirá las puertas a futuros proyectos innovadores. Dentro de las diferentes experiencias se ha construido un camino que permitirá comenzar a encontrar mejoras y aplicaciones a este robot con la esperanza de que se convierta en un proyecto de innovación para la universidad de Málaga.

6.1. Futuras líneas de trabajo

Este proyecto representa una propuesta inicial que permite comprender el funcionamiento del robot Hunter 2.0 y las aportaciones que este puede dar para el área de la navegación autónoma. Debido a esto, existen diferentes mejoras que se pueden ofrecer para complementar el trabajo realizado. Durante el desarrollo de los objetivos han destacado diversos aspectos que requieren mejorías con el fin de implementar un sistema más completo y funcional. Estos aspectos son:

- **Sustituir el controller server del Nav2:** el controlador elegido, Regulated Pure Pursuit, ha demostrado ser poco compatible con el modelo cinemático del robot resultando en movimientos oscilatorios que entorpecen la navegación. Debido a esto, se propone sustituirlo por el controlador MPPI, el cual promete ser una opción óptima para el robot. Al momento de realizar este proyecto este controlador estaba dando fallos de compatibilidad con Ubuntu, por lo que se prevé que esto sea solucionado eventualmente.
- **Diseñar plugins propios**: en caso de que el MPPI no cumpla con las necesidades de navegación del modelo se propone desarrollar plugins externos a Nav2 que permitan dar resultados personalizados para el sistema y una navegación más fluida y eficaz.
- **Realizar más pruebas con el modelo real**: debido a las limitaciones externas al proyecto, el stack de navegación no pudo ser probado con el modelo real en ambientes exteriores. Es beneficioso para el proyecto encontrar una zona de espacio abierto que permita estudiar el comportamiento del modelo real en comparación con el virtual.
- **Aumentar la robustez del sistema**: los algoritmos diseñados presentan diferentes limitaciones que pueden ser tomados en cuenta para diseñar versiones más robustas. Un ejemplo de esto puede ser la evasión de obstáculos que se encuentren en movimiento, lo cual no es tomado en cuenta en ningún momento, o evitar que el robot frene por culpa del Nav2 al encontrar un objeto. De la misma forma, se puede mejorar la maniobra de aparcamiento para que tomen en cuenta la posición inicial del robot al momento de detectar un puesto libre o detecte la dirección del robot al final del movimiento para determinar si su posición es correcta.
- **Experimentar con otros sensores láser**: a pesar de que el sensor Hokuyo encaja con los objetivos de este proyecto, puede ser beneficioso considerar otros tipos de sensores que puedan devolver más datos del entorno, como puede ser un sensor 3D, uno que tome datos en 360 grados o cámaras que permitan detectar diferentes objetos o materiales.

Adicionalmente, se propone a los futuros investigadores encontrar aplicaciones en las que se pueda aprovechar al completo los algoritmos realizados en este trabajo y los que sean propuestos en un futuro. Estas aplicaciones deben ser realizadas en ambientes exteriores, y pueden ser aplicaciones de rescate o transporte de materiales en terrenos desafiantes, por ejemplo. Otra propuesta es la de enlazar este proyecto con otros vehículos con la misma cinemática como base para el desarrollo de trabajos en otros ámbitos. Este proyecto únicamente desarrolla las bases de la navegación autónoma, la cual es un área bastante amplia, por lo que se espera que este trabajo sea complementado con proyectos futuros.

7. Bibliografía

[1] Barrientos Sotelo, V. R., García Sánchez, J. R., & Silva Ortigoza, R. (2007). Robots Móviles: Evolución y Estado del Arte. Polibits, (35), 12-17. URL: https://www.redalyc.org/articulo.oa?id=402640448003

[2] López Valverde, Alberto. "Sistema Locomotor y de Localización de un Microrobot (Eurobot 2009)". Trabajo de Fin de Grado. Escuela Politécnica Superior (Universidad Carlos III de Madrid), 2009. URL: https://e-archivo.uc3m.es/rest/api/core/bitstreams/32d3bd5b-ee51-49a7-9b52-d699d66dff46/content

[3] Molina Villa, Manuel Alejandro; Rodríguez Vásquez, Edgar Leonardo. "Flotilla de Robots para Trabajos en Robótica Cooperativa". Trabajo de Fin de Grado. Facultad de Ingeniería (Universidad Militar Nueva Granada), 2014. URL: https://www.academia.edu/91604475/Flotilla_de_robots_para_trabajos_en_rob%C3%B3tica_cooperativa

[4] ScaleX Innovation. "From ROS to ROS2: A Comprehensive Guide to the Next Generation Robotics". Nov 2023. URL: https://scalexi.medium.com/from-ros-to-ros-2-a-comprehensive-guide-to-the-next-generation-robotics-f93a4e2e5793

[5] Horelican T., Utilizability of Navigation2/ROS2 in Highly Automated and Distributed Multi-Robotic Systems for Industrial Facilities, IFAC-PapersOnLine, Volume 55, Issue 4, 2022, Pages 109-114, ISSN 2405-8963, URL: https://doi.org/10.1016/j.ifacol.2022.06.018

[6] Understanding topics — ROS 2 Documentation: Humble documentation. Ros.org. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

[7] Questions - ROS Answers: Open-Source Q&A Forum. URL: https://answers.ros.org/questions/

[8] Kaiser, J., Norbert Becker, S., Wurster, M., Stricker, N., Lanza, G., "Framework for simulation-based Trajectory Planning and Execution of Robots equipped with a Laser Scanner for Measurement and Inspection", Procedia CIRP, Volume 103, 2021, Pages 292-297, ISSN 2212-8271, URL: https://www.sciencedirect.com/science/article/pii/S221282712100888X

[9] Montenegro, G., Chacón, R., Fabregas, E., Garcia, G., Schröder, K., Marroquín, A., Dormido-Canto, S., Farias, G. (2022). "Modeling and control of a Spherical robot in the CoppeliaSim simulator" (Basel, Switzerland), 22(16), 6020. URL: https://www.mdpi.com/1424-8220/22/16/6020

[10] R. K. Megalingam, A. R, D. HemaTejaAnirudhBabu, G. Sriram and V. S. YashwanthAvvari, "Implementation of a Person Following Robot in ROS-gazebo platform," 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 2022, pp. 1-5, URL: https://ieeexplore.ieee.org/abstract/document/9726010

[11] Torres, C. J., Archila, J. F., Tronco, M. L., Becker, M., Porto, A. J. V., & Tiberti, A. J. (2013). Estudio cinemático de una plataforma robótica para agricultura. Revista Colombiana De tecnologías De Avanzada (RCTA), 2(22), 131-137. URL: https://revistas.unipamplona.edu.co/ojs_viceinves/index.php/RCTA/article/view/421

[12] Johannesen, S. (2022). Diseño, implementación y control de un prototipo de vehículo de Ackermann. URL: https://riunet.upv.es/handle/10251/185078

[13] Keyence, "What is a sensor? Sensor Basics". Guía Técnica. URL: https://www.keyence.com.mx/landing/sensor/en_sensor_16-03-08.jsp

[14] Sosa, L. J. T., & Carelli, R. Control Hibrido para Posicionamiento de un Robot tipo Ackerman. URL: https://www.academia.edu/download/32105940/paper_26.pdf

[15] Chaos, D.; Moreno Salinas, D.; Muñoz, R.; Aranda, J. (2013). Control no lineal de un aerodeslizador no holonómica con acciones de control limitadas. Revista Iberoamericana de Automática e Informática industrial. 10(4):402-412. URL: https://riunet.upv.es/handle/10251/143908

[16] Figueroa, J., Montalvo, W., Baya, M., (2023). "Cinemática y Dinámica de Robots Móviles con Ruedas". Primera Edición, ISBN: 978-9942-7078-3-3. URL: https://ciladi.org/wp-content/uploads/Libro-Robots-VF3.pdf

[17] Marín, L., Soriano, Á., Mayans, V., Vallés, M., Valera, Á., & Albertos, P. Localización Asistida por GPS para Robots Móviles en Configuración Ackermann de Recursos Limitados. URL: http://wks.gii.upv.es/cobami/files/MV_JornadasAutomatica2013LEGO-GPS.pdf

[18] UAVLatam, "¿Qué es un sensor LiDAR y cómo funciona?". 2022. URL: https://uavlatam.com/que-es-un-sensor-lidar-como-funciona/

[19] Agilex Robotics Team. "Hunter2.0 User Manual". Version 2.0.2, 2023. URL: https://global.agilex.ai/pages/download-manual

[20] Agilex Robotics Team. "ugv_sdk Package: C++ SDK for Mobile Robot Platforms". Version 0.8.0, 2022. URL: https://github.com/westonrobot/ugv_sdk

[21] Agilex Robotics Team. "hunter_ros2 Package". 2024. URL: https://github.com/agilexrobotics/hunter_ros2

[22] Hokuyo Autonomic Co., UTN-30LX Sensor Specifications. 2021. URL: https://www.hokuyo-aut.jp/search/single.php?serial=169#spec

[23] Agilex Robotics Team. "ugv_gazebo_sim Package". Version 0.0.1 alpha. 2021. URL: https://github.com/agilexrobotics/ugv_gazebo_sim/tree/master/hunter/hunter2_base/meshes

[24] Open Navigation, "Nav2 1.0.0. Documentation". URL: https://docs.nav2.org/

[25] Open Navigation, "Nav2 1.0.0. Documentation: Navigation Concepts". URL: https://docs.nav2.org/concepts/index.html#behavior-trees

[26] Open Navigation, "Nav2 1.0.0. Documentation: Planner Server". URL: https://docs.nav2.org/configuration/packages/configuring-planner-server.html

[27] Open Navigation, "Nav2 1.0.0. Documentation: Map Server". URL: https://docs.nav2.org/configuration/packages/configuring-map-server.html

[28] Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999, May). Monte Carlo localization for mobile robots. In Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C) (Vol. 2, pp. 1322-1328). IEEE. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=772544

[29] Open Robotics, "slam_toolbox", ROS2 Documentation, 2021. URL: https://wiki.ros.org/slam_toolbox

[30] Steve Macenski. "slam_toolbox Package.". Version 2.8.1. 2024. URL: https://github.com/SteveMacenski/slam_toolbox?tab=readme-ov-file

[31] Takei, R., & Tsai, R. (2013). Optimal trajectories of curvature constrained motion in the hamilton–jacobi formulation. *Journal of Scientific Computing*, *54*, 622-644. URL: https://link.springer.com/article/10.1007/s10915-012-9671-y

[32] DGT, "Como estacionar en línea." 2015. URL: https://revista.dgt.es/es/multimedia/infografia-animada/2015/1030Estacionamiento-en-linea.shtml

[33] Ortigoza, R. S., Sánchez, J. G., Sotelo, V. B., Vilchis, M. A. M., Guzmán, V. H., & Ortigoza, G. S. (2007). Una panorámica de los robots móviles. *Télématique: Revista Electrónica de Estudios Telemáticos*, *6*(3), 1-14. URL: https://dialnet.unirioja.es/servlet/articulo?codigo=2961668

[34] Espitia Cuchango, H. E., & Sofrony Esmeral, J. I. (2012). Algoritmo para planear trayectorias de robots móviles, empleando campos potenciales y enjambres de partículas activas brownianas. *Ciencia e Ingeniería Neogranadina*, *22*(2), 75-96. URL: http://www.scielo.org.co/scielo.php?pid=S0124-81702012000200005&script=sci_arttext

[35] García Sánchez, J. R. (2008). Diseño y construcción de un robot móvil aplicando el método de campos potenciales en la evasión de obstáculos. URL: https://tesis.ipn.mx/bitstream/handle/123456789/26194/Dise%C3%B1o%20y%20construcci%C3%B3n%20de%20un%20robot%20m%C3%B3vil%20aplicando%20el%20m%C3%A9todo%20de%20campos%20potenciales%20en%20la%20evasi%C3%B3n%20de%20obst%C3%A1culos.pdf?sequence=1&isAllowed=y