Trabajo Fin de Máster

Estudio y análisis comparativo de algoritmos de planificación local para robots móviles basado en ROS.

Autor

Pau Jordán Tomás

Directores

Vicent Girbés Juan
Valero Laparra Pérez-Muelas

Máster Universitario en Ingeniería Electrónica
Escola Tècnica Superior d'Enginyeria
Valencia, septiembre 2021

Índice

Índice de figuras

Índice de tablas

1. Introducción

En esta primera sección vamos a hablar en primer lugar sobre la motivación que nos ha llevado a la realización de este proyecto, en base a posibles necesidades de la comunidad frente al avance constante de la robótica autónoma. Una vez explicado el motivo que nos lleva a realizar el trabajo hablaremos sobre una serie de objetivos, los cuales nos ayudarán a mantener el camino claro y poder realizar el proyecto. Por último, se hará un resumen de cada uno de los apartados del documento y así, tener una noción del proyecto sin tener que indagar en él.

1.1. Motivación

El motivo por el cual se ha decidido realizar el siguiente trabajo se debe a la resolución de un problema que hemos percibido en la navegación de la robótica autónoma, ya que, para lograr la navegación autónoma se necesita conocer el estado del robot en tiempo real para poder gobernarlo mientras realiza los movimientos. Esa gestión se puede realizar de varios modos:

- Colocación de una gran cantidad de sensores que nos reporten información del entorno del robot para poder conocer las situaciones en las que se encuentran y cómo actuar respecto a los estímulos registrados.
- Añadir a esta parte de sensores una inteligencia, el cual es nuestro caso, con ello, no harían falta tantos sensores, ya que gracias a los mapas generados por el robot tendríamos una navegación más óptima y eficaz que el caso anterior.

En este caso hemos detectado que la cantidad de planificadores de rutas que son capaces de lograr este modo de comportamiento es muy grande, y a la hora de elegir uno de ellos para una funcionalidad puede ser algo bastante tedioso. Mediante este estudio, lo que se pretende es sacar conclusiones sobre el comportamiento de diferentes algoritmos de navegación, y así poder ayudar a otros usuarios a tomar la decisión correcta a la hora de realizar un sistema de navegación para un robot móvil y una utilidad específica.

Con ello no llegamos a resolver la problemática de la gran cantidad de elecciones disponibles, pero entregamos una herramienta que pueden utilizar para facilitar la elección. Además de ser un buen trabajo para validar la bondad de algoritmos que realicen a nivel personal los usuarios.

Como añadido, también decir que, hoy en día la utilización de planificadores que sean capaces de gobernar la planificación de nuestro robot se está volviendo cada vez algo más importante, desde la aparición de los primeros coches autónomos, aunque para esto aún queda bastante, hasta la robótica doméstica, ya que hoy en día casi todos los hogares cuentan con un robot aspirador que limpia la casa para nosotros [25].

1.2. Objetivos

Para este proyecto, tal y como se ha comentado en el apartado anterior, se va a realizar una comparativa sobre diferentes planificadores locales en un sistema de navegación integrado en un robot móvil. Para llegar a cumplir el que será el objetivo principal de este proyecto, primero debemos de alcanzar unas metas para seguir un orden en el trabajo y poder cumplir con los plazos de entrega. A continuación, se muestran los diferentes objetivos propuestos para la realización del proyecto:

1. Construcción de una base sobre la que poder probar diferentes algoritmos, es decir, tener todas las herramientas a punto, así como un repositorio funcional, donde poder aplicar las modificaciones en el comportamiento a la hora de hacer pruebas.
2. Realización de un estudio de los algoritmos y de los parámetros que los manejan, para así poder hacer que el robot se comporte de formas diferentes y realizar una comparativa lo más completa posible.
3. Recopilación de datos, a partir de una serie de pruebas establecidas, hablaremos de ellas en la Sección 4, en la que también definiremos unas métricas donde escogeremos los datos que consideremos relevantes para nuestro proyecto. Las pruebas primero se realizarán en un entorno virtual y luego en la realidad.

Una vez realizadas las pruebas y obtenidos los datos que necesitamos podremos centrarnos en realizar la comparativa y extraer conclusiones para alcanzar el objetivo principal. Más adelante también hablaremos de objetivos a futuro, ya que a esta comparativa se le podrían añadir planificadores propios o de otros usuarios y comprobar la calidad de estos algoritmos. Pero esta parte la desarrollaremos con más profundidad en la Sección 6.2.

1.3. Estructura

Para la realización del proyecto, en la Sección 2 vamos a realizar una primera toma de contacto con la temática del proyecto, explicando en primer lugar unas nociones básicas de la navegación con robots móviles y los algoritmos que se utilizan para conseguir esta navegación autónoma. En este apartado, también se estudiarán algunos conceptos sobre la utilización y funcionamiento del sistema operativo y algunos programas que necesitaremos para realizar el proyecto, así como el simulador que vamos a utilizar para realizar las simulaciones del trabajo. Finalmente, veremos un estudio donde se verán algunos algoritmos soportados por la plataforma. En este apartado veremos un breve resumen del funcionamiento de cada uno de ellos y seguidamente hablaremos sobre los parámetros que controlan estos algoritmos, para así, conocer bien el funcionamiento de cada uno y posteriormente elegir los parámetros necesarios para cambiar su comportamiento y lograr una comparativa más extensa.

Posteriormente, en la Sección 3 realizaremos una explicación sobre el circuito de pruebas diseñado, contando los diferentes objetivos que va a tener que alcanzar el robot y a qué pruebas se enfrentará en cada uno de estos objetivos. Luego, se explicarán una serie de conceptos necesarios para la puesta en marcha de la experimentación del proyecto.

Siguiendo el índice, pasaremos a la Sección 4 donde seleccionaremos los parámetros más relevantes vistos en la Sección 2.3 y así poder realizar tres modos de conducción, para que la comparativa del proyecto sea más extensa y detallada. Con los parámetros necesarios elegidos pasaremos a definir una serie de métricas, donde explicaremos cómo extraeremos los diferentes valores de las variables definidas para poder realizar el análisis del comportamiento de estos planificadores locales. Finalmente, explicaremos de qué modo vamos a adquirir los datos para poder realizar las métricas comentadas anteriormente.

Una vez extraídos los datos para las métricas definidas, pasaremos a realizar toda la comparativa de los resultados obtenidos en la sección 5.1. Con estos datos en forma de tablas y gráficas, veremos de una forma más visual y cómoda lo que ha sucedido en las simulaciones y podremos analizar y extraer conclusiones del funcionamiento de estos algoritmos estudiados para este proyecto. Seguidamente, en la Sección 5.2 vamos a comentar de una forma más subjetiva el comportamiento de los diferentes algoritmos, y comentar cuál se ha comportado de mejor forma en este circuito de pruebas.

Por último, para concluir con el proyecto, en la Sección 6 veremos las conclusiones que hemos extraído de la realización de este trabajo, donde evaluaremos si el proyecto ha cumplido los objetivos vistos en la Sección 1.2. Además, en la Sección 6.2 veremos algunas de las diferentes líneas de trabajo a futuro que presenta este proyecto.

2. Marco teórico

2.1. Navegación autónoma

2.1.1. Conceptos básicos de la navegación en robots

La navegación en robots es la parte de la robótica que dota de cierta inteligencia al robot y es la que consigue que éste se mueva de un punto a otro de forma segura. La navegación actúa sobre la velocidad y la aceleración para completar estos objetivos. Para conseguir esto, el robot se apoya en un gran número de recursos. En este apartado vamos a introducir un poco la lógica que deben de seguir los robots para lograr un comportamiento correcto. Para entender mejor la lógica de navegación, vamos a apoyarnos en la Figura 1, en la que tenemos un diagrama de flujo explicando de una forma más gráfica la navegación.

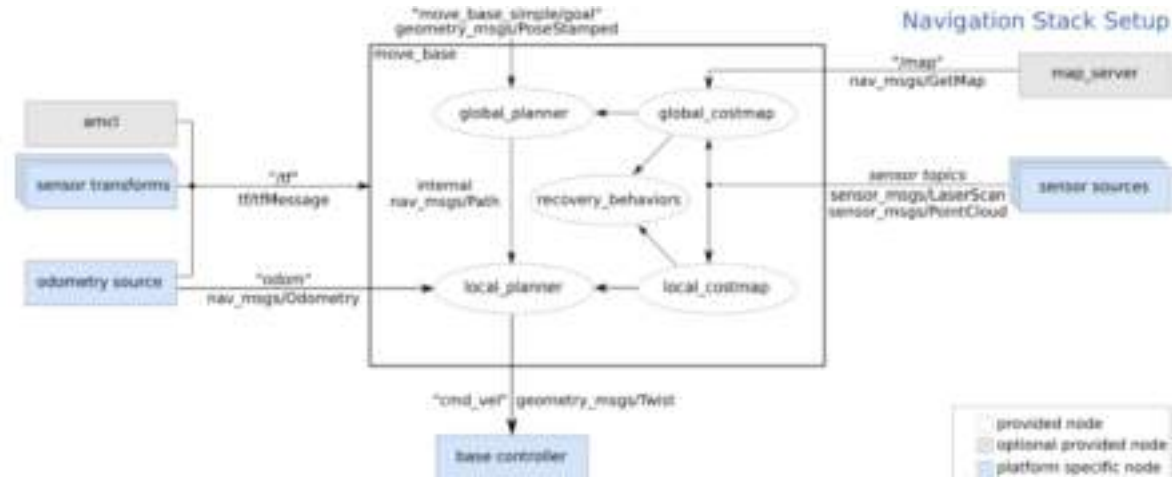


Figura 1: Esquema básico del funcionamiento de la navegación en robots.

Cuando empezamos la navegación en un robot, en primer lugar hay que crear un mapa de costos global, este mapa es una imagen del entorno, marcando con un 255 las zonas donde tenemos un obstáculo, con un 0 las zonas libres y las zonas que aún no han sido mapeadas se marcan con un -1. Esto se consigue mapeando con ayuda de los sensores del propio robot, es por eso que vemos que el bloque de sensores va al mapa de costos global.

También observamos que este bloque se dirige al mapa de costos local. Dicho mapa es el mismo mapa que el explicado anteriormente, pero a diferencia del mapa de costos global, éste es un mapa dinámico, esto quiere decir que este mapa se modifica en función de la respuesta que están obteniendo los sensores en tiempo real y además solo se crea el mapa de costos en el entorno del alcance de los sensores del robot [2].

Cada mapa de costos se dirige a su planificador. En el caso del planificador global, va a calcular una ruta entera desde el punto de inicio hasta el punto objetivo. Por eso este tipo de planificadores necesita conocer el comando que le indique dónde está el objetivo y en qué posición debe el robot alcanzar este objetivo.

El caso del planificador local es un poco más complejo ya que, además de necesitar la trayectoria calculada con antelación del planificador global, necesita conocer la respuesta en tiempo real de la información de los sensores, traducida por el mapa de costos local y necesita tener información sobre la odometría del robot. Con todo ello, este planificador es capaz de calcular una pequeña ruta libre de obstáculos y segura para la navegación [23]. Además estos algoritmos son capaces de recalcular la trayectoria global para evitar posibles obstáculos que puedan surgir en el entorno del robot y que no estén registrados en el mapa global.

2.1.2. Cómo iniciar la navegación

Lo primero que debemos hacer es asegurarnos de que el robot esté listo para la navegación. Para ello, realizaremos 3 comprobaciones:

-Sensores de alcance: si el robot no obtiene información de estos sensores (láser, ultrasonidos, infrarrojos, etc.) el robot no funcionará con la navegación, ya que no obtendrá la información del entorno. Para que esta información se pueda leer correctamente y la velocidad de datos sea la adecuada utilizaremos una herramienta de visualización gráfica que explicaremos en la Sección 2.2.2.1 del proyecto.

-Odometría: es la estimación de la posición de vehículos móviles durante la navegación. Para el estudio, utilizamos información sobre la rotación de las ruedas, para estimar cambios de posición durante el tiempo. Para comprobar que la odometría funciona correctamente, vamos a realizar 2 procesos:

- La primera prueba verifica qué tan razonable es la odometría para la rotación. Para ello, en el caso de este robot, miraríamos el escaneo láser que proporciona el robot, configuramos el tiempo de decaimiento (unos 2 seg) y realizamos una rotación en el lugar. Luego, miramos cómo de cerca coinciden los escaneo entre sí en rotaciones posteriores. Como es muy difícil que el escaneo coincida del todo, vamos a asegurar que la desviación no sea mayor a 1 o 2 grados.

- En la segunda, pondremos el robot a unos metros de la pared. Luego, conduciremos el robot directamente a la pared y veremos el grosor de la pared según lo informado por el escaneo láser agregado en la aplicación. En teoría la pared debería verse como un solo escaneo, peo como en la prueba anterior sólo vamos a asegurarnos de que no tenga un grosor de más de unos pocos centímetros. Pero si conduce un metro hacia una pared y el escaneo muestra más de medio metro es probable que algo esté mal con la odometría.

-Localización: cuando los puntos anteriores funcionen correctamente, hacer un mapa y ajustar AMCL (Adaptive Monte-Carlo Localizer) (es un sistema de localización probabilística para un robot que se mueve en 2D, utiliza un filtro de partículas para rastrear la pose de un robot contra un mapa conocido.) no es tan difícil. Primero, moveremos el robot para generar un mapa. Después usaremos ese mapa con AMCL y nos aseguraremos de que el robot permanezca localizado. Si la odometría del robot no es muy buena, habrá que jugar con los parámetros para la odometría para AMCL. Una buena prueba es asegurarse que el escaneo láser y el mapa se puedan ver en el marco del mapa en nuestro entorno gráfico y que el escaneo coincida bien con el mapa del entorno [11].

Vamos a usar SLAM (Simultaeous Localization And Mapping) para que el robot mapee el circuito de pruebas que vamos a diseñar y aprenda a moverse por ella. Gracias al SLAM el robot podrá estimar trayectorias y re-calcular rutas en caso de que una de ellas no sea viable o tenga algún obstáculo [13].

Podemos utilizar un láser o sensor, ya que para construir el mapa debemos tomar los datos del láser y de la odometría del robot y interpretar la información para rellenar las celdas de ocupación 2D. Este mapa se actualiza mientras el robot se va moviendo. Este mapa solo nos da información local del robot, es decir la información que los sensores son capaces de percibir en un instante concreto. El mapa global estará dividido por celdas, en las cuales tendremos diferentes valores. Siendo 0 cuando la celda esté totalmente libre y 100 si la celda esta totalmente ocupada. Si tenemos un valor de -1 significara que no sabemos cómo está esta celda [24].

2.1.3. Planificadores locales

La comparativa que vamos a realizar se basa en comparar diferentes planificadores locales. En la navegación utilizamos dos tipos de planificadores diferentes para asegurar que el objetivo final se alcanza correctamente, planificadores globales y locales. En este caso, los que nos interesan y vamos a analizar son los locales.

Los planificadores locales utilizan los sensores del robot para planificar una trayectoria segura, es decir, mediante el plan global el robot crea una trayectoria del punto A al punto B, basado en el mapa que anteriormente ha mapeado y ha guardado. Con el planificador local conseguimos que el robot recorra esa trayectoria de forma más segura, ya que, si el robot se acerca mucho a los bordes, o tenemos obstáculos dinámicos que se cruzan por la trayectoria u obstáculos que no había cuando se hizo el mapeo de la zona, sólo con la trayectoria global el robot no sería capaz de evitar esta clase de obstáculos [16].

Los planificadores locales utilizan el modelo de movimiento del robot para encontrar el mejor conjunto de comandos que cumplan con el plan global. Para un robot de dirección deslizante, por ejemplo, "simularíaçada velocidad de avance y rotación que es capaz de realizar y elegir el mejor conjunto de velocidades para lograr el objetivo.

Luego los planificadores locales son capaces de recalcular las trayectorias a tiempo real. En nuestro caso, vamos a estudiar cómo se comportan estos algoritmos y cómo recalculan la ruta global cuando surgen esta clase de problemas.

Con todos estos datos vamos a ser capaces de realizar una búsqueda de algoritmos de navegación que cumplan todas estas especificaciones. Además, vamos a buscar algoritmos los cuales tengan unos parámetros que sean sencillos de manejar para poder realizar las pruebas con distintos comportamientos del robot.

Tras la búsqueda de distintos planificadores locales, en este proyecto nos hemos centrado en la comparación de algunos planificadores locales en los que la plataforma de firmware/software que hemos usado nos da soporte. En este caso se van a comparar 5 planificadores diferentes, los cuales se explicarán más a fondo en la Sección 2.3:

⇒ TR Rollout planner (TR) [19].

⇒ Dynamic Window Approach local planner (DWA) [18].

⇒ Elastic BAND local planner (EBAND) [3].

⇒ Timed Elastic Band local planner (TEB) [26].

⇒ Active Scene Recognition local planner (ASR) [20].

Como se comentará más adelante, este proyecto tiene una proyección a futuro, ya que una vez realizada esta comparativa se pueden añadir otros algoritmos propios y así tener un conocimiento sobre la bondad de los planificadores que realizamos, haciendo así una comparativa mucho más extensa.

En la Figura 2 se muestra un ejemplo de planificación en robots móviles. Como vemos, el robot a partir del mapa del entorno previamente mapeado (la parte gris claro de la figura), encuentra una ruta libre de obstáculos estáticos optimizando el camino para llegar de la forma más corta y rápida que los parámetros permitan (línea verde). Esto es lo que sería el planificador global, ya que como vemos la trayectoria calculada es la trayectoria entera que el robot va a realizar. El planificador local, a partir de estos datos, va calculando pequeñas trayectorias intentando imitar al planificador global y de este modo ir cumpliendo el objetivo marcado.

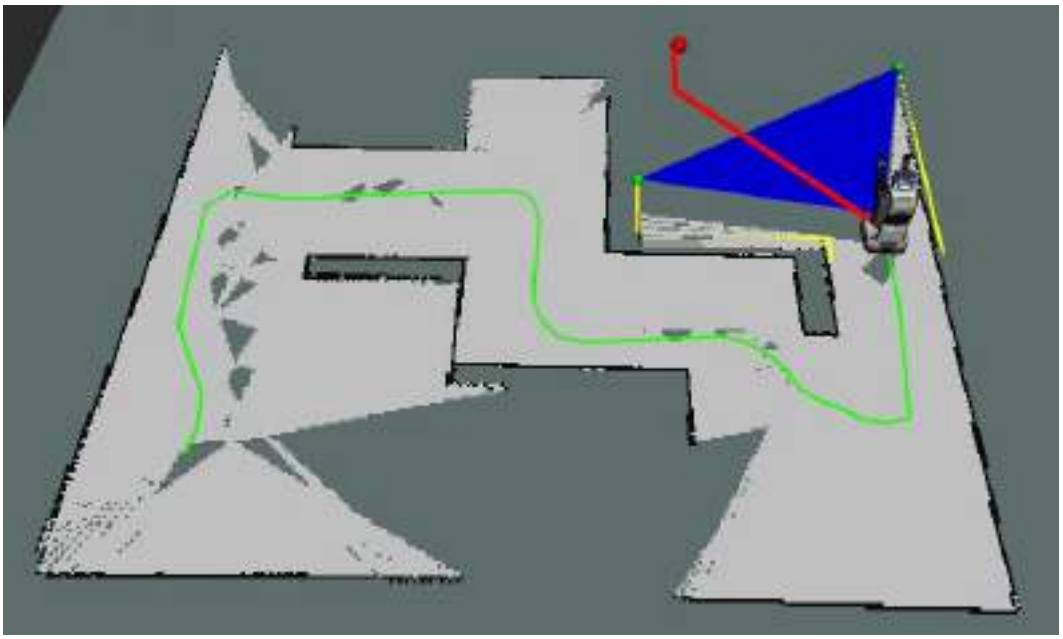


Figura 2: Ejemplo de planificación en robots móviles.

2.2. Robot Operating System (ROS)

En este apartado vamos a hablar sobre el programa que vamos a utilizar para realizar nuestra comparativa. Comentando en primer lugar el contexto sobre cómo surgió la idea de este firmware y el porqué de la necesidad de realizar un conjunto de paquetes software de estas características. También comentaremos una serie de ventajas que esta plataforma nos ofrece a diferencia de otras plataformas sobre robótica que existen hoy en día.

En segundo lugar, pasaremos a una parte más técnica donde explicaremos, por encima, cómo funciona el programa. Es decir una pequeña introducción para en los siguientes puntos poder entender la terminología empleada.

2.2.1. Historia de ROS

ROS (Robot Operating System) es un marco flexible para la escritura de software del robot (su logo se muestra en la Figura 3). En este sistema podemos encontrar una gran cantidad de herramientas, librerías y conversiones con el objetivo de simplificar las tareas complejas en una gran variedad de plataformas robóticas.

Figura 3: Logo de ROS.

La idea de este sistema es la de que los programadores de robótica puedan utilizar código diseñado por otros programadores, con el fin de facilitar el desarrollo de códigos. Se trata de una plataforma muy extendida y los fabricantes cada vez más introducen drivers compatibles con ROS, por ello, se está convirtiendo en un estándar en esta industria. ROS se creó desde cero para incentivar el desarrollo de códigos de robótica con la intención de que distintos grupos de expertos colaboren y se basen en el trabajo realizado por otro grupo, es decir, el programa se basa en el código abierto.

La historia de este programa empieza a mediados del 2000. cuando en la Universidad de Stanford, se incorporó una inteligencia artificial (IA), donde crearon prototipos internos de sistemas de software dinámicos y flexibles, destinados al uso en robótica. En 2007, Willow Garage, una empresa robótica visionaria, proporcionó una gran cantidad de recursos para extender los conceptos anteriores mucho más y también crear implementaciones bien probadas. Este esfuerzo fue impulsado por muchos investigadores los cuales contribuyeron con sus conocimientos tanto a las ideas centrales de ROS como a sus paquetes de software fundamentales. Durante todo el desarrollo el software se hizo de forma abierta utilizando licencias permisivas de código abierto BSD. Poco a poco se ha ido convirtiendo en una plataforma muy utilizada en el campo de la investigación robótica.

ROS se desarrolló en múltiples instituciones y para diversos robots, lo cual es un punto fuerte. Un usuario puede iniciar su propio repositorio de código con sus propios servidores, manteniendo la propiedad y el control. No requiere ninguna clase de permiso, siempre que se elija poner el repositorio a disposición pública. Con ello se puede conseguir el reconocimiento de otra gente, así como beneficiarse de mejoras en el código que otros usuarios puedan aportar al código inicial. Hoy en día ROS consta de una comunidad muy extensa a nivel mundial, donde podemos encontrar desde proyectos para los usuarios más inexpertos hasta grandes proyectos de automatización industrial.

En conclusión, las razones que nos han movido a utilizar el programa son las siguientes. En primer lugar, que sea de código libre y podamos utilizarlo sin necesidad de licencia. En segundo lugar, la gran comunidad de usuarios que tiene, lo que facilita mucho tanto la obtención de código como la resolución de posibles problemas que puedan surgir.

2.2.2. Conceptos básicos de ROS

En este apartado vamos a explicar de forma resumida cómo funciona ROS. El objetivo es comprender el proceso que estamos realizando para lograr las simulaciones de las cuales extraeremos los datos para realizar el estudio comparativo de algoritmos de planificación local, que es el objetivo principal del proyecto.

ROS funciona mediante una serie de grupos llamados nodos, que se comunican unos con otros mediante un sistema de publicación-suscripción anónima. Esto quiere decir que los nodos en ningún momento saben con quién se están comunicando. Esta es la forma con la que podemos comunicarnos dentro del robot y también tener información sobre todo lo que está pasando a nivel interno.

Estos nodos son controlados por un nodo llamado máster. Éste se encarga de gestionar a los demás nodos, es decir, todos los nodos tienen que solicitar la suscripción al máster y posteriormente es éste el que se encarga de que cada nodo se suscriba al topic del cual quiere sacar información. En el caso de las publicaciones no funciona exactamente igual, sino que cuando un nodo quiere publicar alguna información, ya sea velocidad, aceleración, etc. en primer lugar el nodo pide permiso para publicar la información y posteriormente el nodo lo publica en el topic.

Los topics son buses de comunicación utilizados para realizar el intercambio de información entre nodos. Cuando un nodo se suscribe a otro, se suscribe al topic que le interesa donde el nodo ha publicado la información correspondiente en ese topic, por lo que podemos tener más de un publicador y por supuesto más de un suscriptor para la misma información.

Otro sistema de comunicación que nos ofrece ROS son los servicios, que se basan en comunicación unidireccional. Por lo que un servicio se basa en 2 mensajes: uno de solicitud y otro de respuesta. Por ejemplo, un nodo proveedor ofrece un servicio con un nombre determinado, otro nodo cliente llama a este servicio enviando el mensaje de solicitud y se queda esperando la respuesta hasta que el nodo proveedor envía la información a éste.

En la Figura 4 podemos ver de una forma más clara cómo funciona la comunicación en ROS.

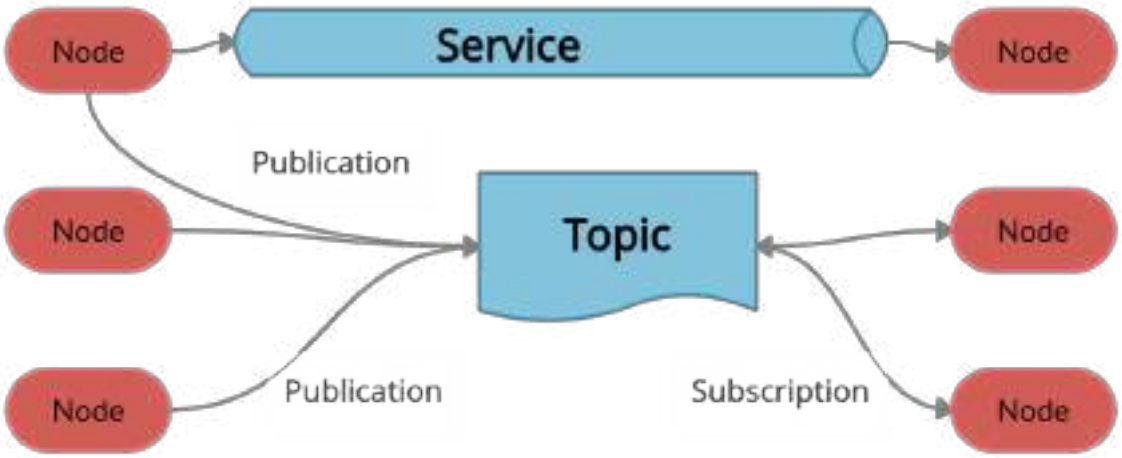


Figura 4: Esquema de comunicación de ROS.

Para crear un proyecto en ROS debemos seguir también un orden en el que tenemos un repositorio, dentro tenemos diferentes stacks y dentro de estos paquetes. A continuación, explicaremos la función que realiza cada nivel.

Los repositorios son solo repositorios de código. Estos repositorios de código suelen estar asociados con organizaciones o una comunidad de desarrolladores.

Los stacks o pilas son la unidad básica para liberar código ROS. Recopilan paquetes de temática similar y también están destinadas a agrupar el código que se desarrolla en conjunto y es mutuamente interdependiente. Por ejemplo, el stack de navegación consta de varios paquetes de planificación, un nodo ROS de alto nivel, un paquete de localización y estructuras de datos de obstáculos.

Por último, los paquetes son los que crean y construyen el código ROS. Se trata del bloque de nivel más bajo de lo que puede construir en ROS. Pueden contener algoritmos, librerías, conjuntos de mensajes u otros componentes básicos en particular.

Todo lo comentado en estos párrafos lo podemos ver de una forma más visual en la Figura 5.

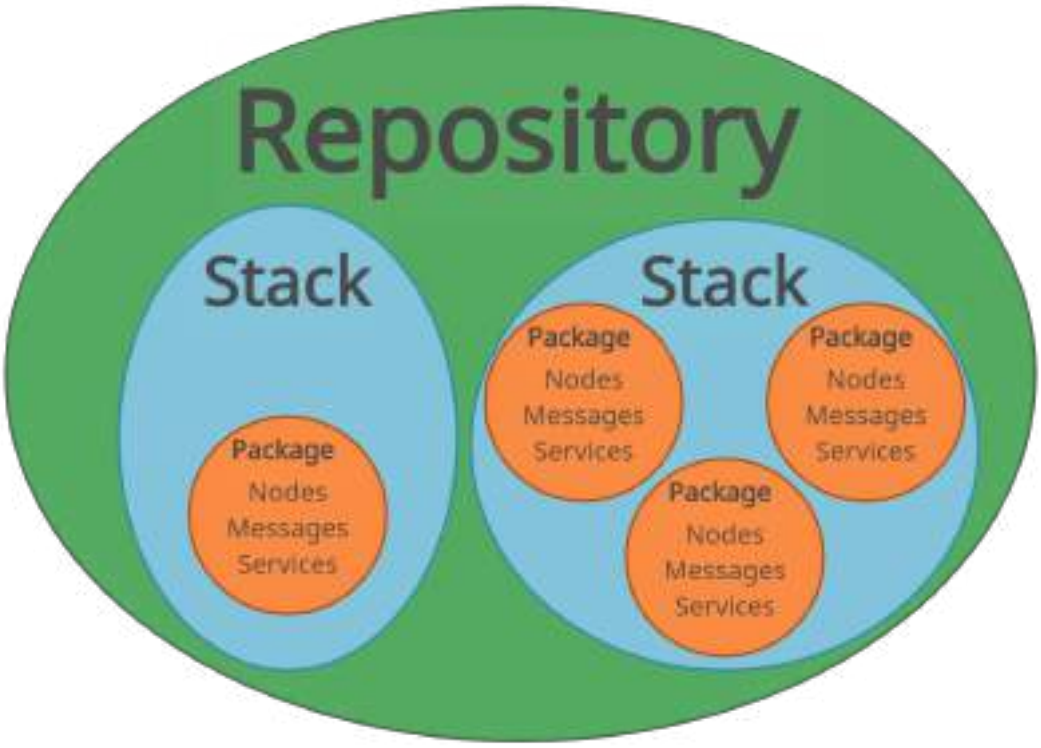


Figura 5: Esquema de paquetes de ROS.

Además de todo esto, dentro del gran sistema de ROS se encuentran una gran cantidad de herramientas donde podremos controlar todos estos conceptos comentados. Éstas nos ayudarán a realizar un control sobre el robot de forma sencilla y visual. En la página de ROS podemos encontrar una gran cantidad de tutoriales que nos guiarán a través del proceso de aprendizaje, tanto del sistema como de las herramientas que nos ofrecen [4].

Como apunte, este sistema con el nodo máster, desaparece en ROS2, lo que da una gran plasticidad al sistema, aunque hay que gestionar manualmente los envíos y las recepciones a los diferentes nodos.

ROS2 es un sistema muy nuevo, ya que surgió a mediados de este año. Por ese motivo hemos decidido seguir en ROS, ya que se había invertido una gran cantidad de tiempo en aprender a utilizar este sistema y al ser nuevo, aun no hay una comunidad lo suficientemente extensa. En consecuencia, los planificadores puede que den problemas a la hora de compilarlos en este nuevo sistema, y tener que reducir el número de algoritmos, quitándole peso al trabajo realizado.

2.2.2.1 Rviz

Esta herramienta utilizada es una de las herramientas principales de nuestro proyecto. Rviz es un visualizador 3D de ROS que nos ayudará a ver el comportamiento del robot y la información de los sensores, todo ello en tiempo real. Con Rviz también podemos comunicarnos con el robot de forma gráfica, en este caso se utiliza para enviar a los tópicos correspondientes el planificador local que vamos a utilizar, así como decir el objetivo que el robot debe alcanzar. Con esta herramienta podemos monitorizar todo ello y observar cómo se comporta el robot durante la trayectoria [12].

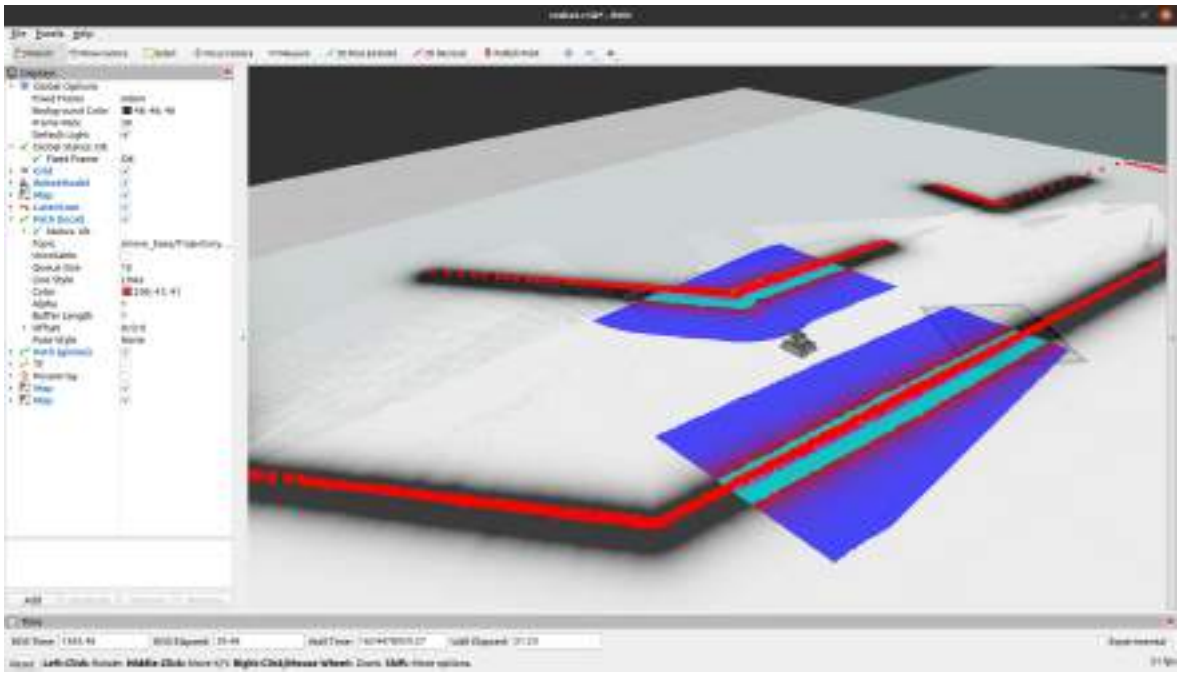


Figura 6: Ejemplo de ventana de Rviz.

2.2.2.2 Dynamic reconfigure

Esta es otra de las herramientas de ROS, la cual nos va a permitir configurar el valor de los parámetros de toda la navegación, pero en nuestro caso vamos a modificar solamente los parámetros de los planificadores locales. Dynamic reconfigure nos permite hacer esto de forma gráfica y sin necesidad de volver a lanzar el nodo. Con esto se podrán modificar los parámetros para la realización de las pruebas, donde se pondrán a prueba diferentes configuraciones de los parámetros para así poder ver como se comporta el planificador, tal y como muestra la Figura 7. Además, esta herramienta también nos va a permitir monitorizar los datos de las simulaciones de una forma precisa, pudiendo realizar gráficas de las diferentes variables a estudiar. Esto nos ayudará a realizar la comparativa y así extraer conclusiones más exactas sobre el comportamiento de los planificadores locales.

Hoy en día, el enfoque de esta herramienta es proporcionar una forma estándar de exponer un conjunto de parámetros de un nodo. En esta herramienta, además de encontrar los parámetros de diferentes nodos, podemos personalizar la ventana de la herramienta con una interfaz propia, donde podemos insertar aquello que sea interesante para nuestro proyecto. Esto resulta muy útil para los controladores hardware, pero tiene una aplicación mucho más extensa [9].

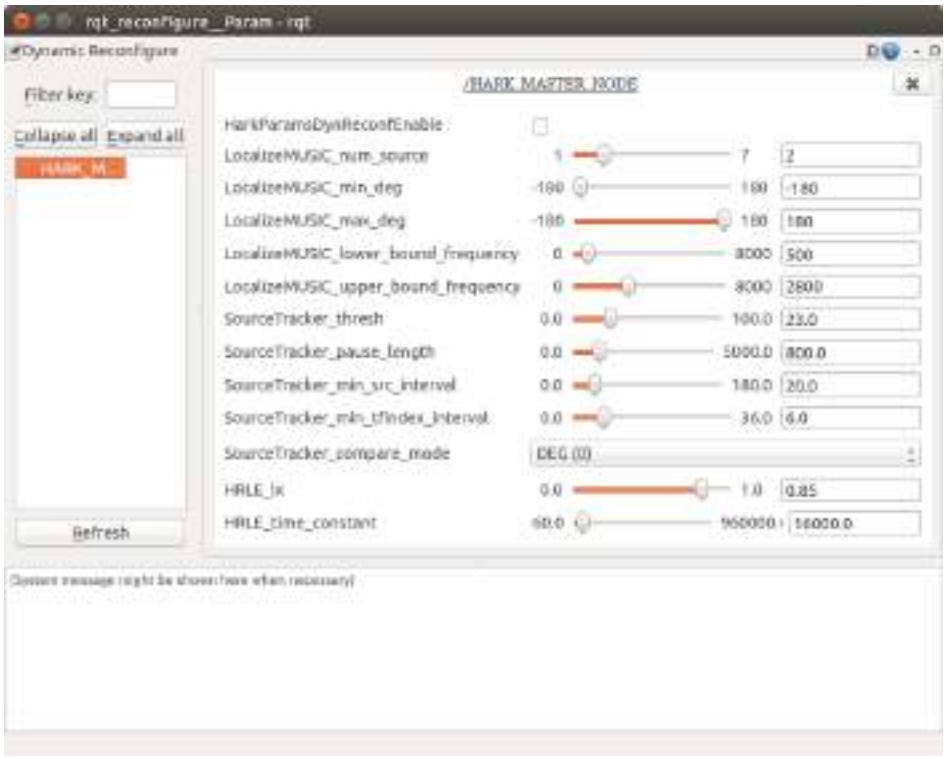


Figura 7: Ejemplo de ventana de dynamic reconfigure.

2.2.3. Simulador Gazebo

Figura 8: Logo de Gazebo.

Gazebo es un simulador de entornos 3D que nos permite de forma sencilla e intuitiva evaluar el comportamiento de un robot en un mundo virtual. El programa nos permite realizar numerosas opciones, entre ellas podemos diseñar robots totalmente personalizados, crear mundos donde estos robots pueden interactuar, además también podemos importar modelos ya diseñados con los que simular estos mundos [17].

Una de las ventajas de este simulador es que podemos sincronizar todo este entorno virtual con ROS de forma que los robots pueden publicar en los nodos correspondientes la información que están teniendo los sensores que estamos emulando. Así resulta muy sencillo implementar lógicas de navegación en diferentes entornos y con diferentes robots.

Gazebo tiene unas herramientas CAD de diseño bastante limitadas, así que los mundos que creemos

no pueden ser muy complejos. A pesar de ello, podemos realizar unos modelados muy próximos a la realidad, ya que podemos incluir gravedad, propiedades físicas de los objetos, momentos de inercia, dotar a los objetos de detección de colisión, incluso la presión, temperatura del mundo, vientos, etc.

A continuación, vamos a profundizar un poco en las opciones principales de diseño que nos ofrece Gazebo:

- En la parte superior de la ventana encontramos una barra de herramientas. Tenemos modelado de figuras geométricas donde se pueden insertar cubos, cilindros y cubos; también tenemos varios modos de direccionamiento de la luz.
- A la izquierda de la ventana principal tenemos las opciones para seleccionar elementos, moverlos o rotarlos.
- A la izquierda de la ventana de simulación encontramos un listado donde aparecen todos los modelos que están en el mundo, así como las configuraciones para hacer la simulación más real.
- Seleccionando un modelo también podemos modificar las propiedades, como la posición, la orientación, el nombre...

En resumen, Gazebo es un simulador creado para probar algoritmos de navegación, diseño de robots, entrenar sistemas de inteligencia artificial, y todo ello de forma precisa y eficiente utilizando escenarios realistas, tanto en interiores como en exteriores. Además, este programa es totalmente gratuito y tiene una comunidad muy extensa, lo que significa que hay una gran cantidad de modelos creados y probados, así como paquetes que incluyen un entorno gráfico ya preparado para ejecutarse en ROS [7].

2.3. Algoritmos de planificación local

En este apartado vamos a introducir resumidamente cómo funciona cada uno de los planificadores locales que hemos escogido en la Sección 2.1.3. En segundo lugar, realizaremos un estudio paramétrico para escoger los valores que harán que el robot se comporte del modo que deseamos.

Los algoritmos que vamos a ver a continuación surgen por la problemática de la planificación de trayectorias geométricas. Existen diversas formas de planificación como la planificación basada en muestreo, los cuales se aplican a un amplio conjunto de problemas y han tenido éxito en el tratamiento de casos difíciles de planificación. Para instancias de planificación específicas, a menudo más sencillas, existen enfoques alternativos que brindan garantías teóricas y, para instancias de planificación simples, superan a los planificadores basados en muestreo [15].

2.3.1. Planificador Trajectory Rollout (TR)

Este algoritmo implementa la lógica para la navegación local de un robot móvil. TR proporciona comandos de navegación local para que sean seguros para el propio robot.

TR utiliza la velocidad actual como referencia para generar un rango de muestreo a una frecuencia razonable, con el cual se determina la siguiente muestra de velocidad. Además, con esta señal de muestreo, genera la ruta de simulación correspondiente. Con ayuda del mapa de costos, el algoritmo evalúa diferentes cosas, entre ellas, los obstáculos, la distancia a la meta y la distancia desde la ruta global que se ha planificado. Con este mapa de costos, al igual que otros algoritmos, se selecciona la ruta local óptima y crea los comandos de velocidad adecuados para ese fragmento de trayectoria[10].

El algoritmo funciona de la siguiente forma: El robot genera la ruta de avance para alcanzar el objetivo, si en esa ruta el robot encuentra un obstáculo, el algoritmo ingresa el modo de escape, entonces este retrocede y gira, hasta que se aleja del obstáculo y vuelve a calcular el comando de avance en función de los datos que hemos comentado anteriormente [19] (Ver Figura 9).

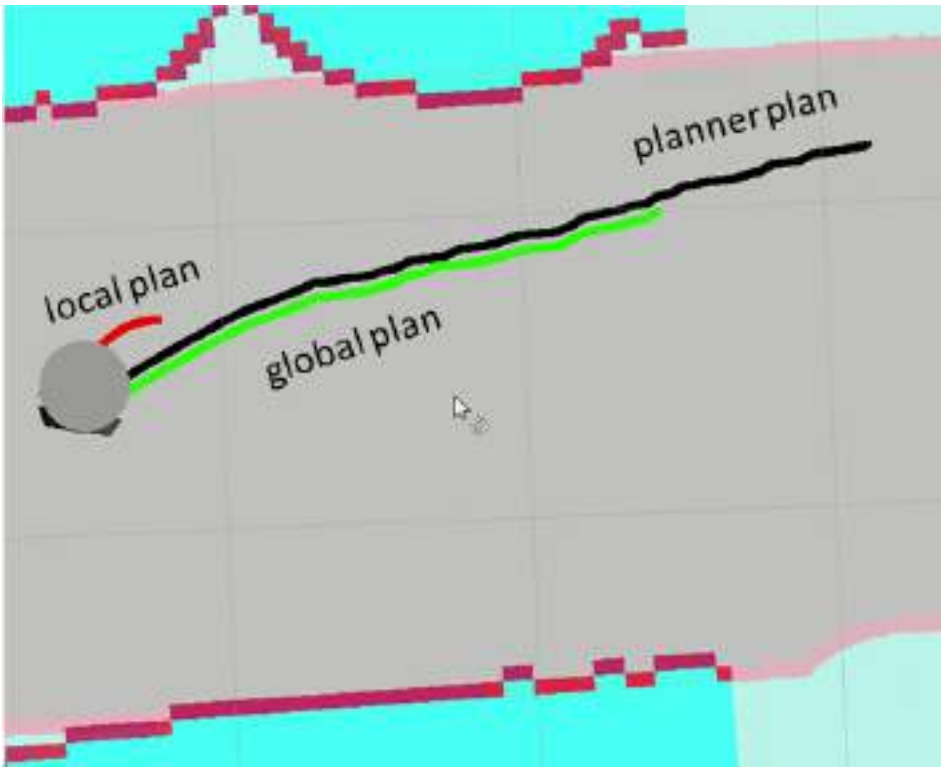


Figura 9: Ejemplo de funcionamiento del planificador TR.

Este planificador tiene 6 grupos de parámetros que nos ayudarán a realizar el control del algoritmo:

1. Primer grupo, los parámetros de configuración del robot, este conjunto de parámetros modifica las velocidades y aceleraciones máximas y mínimas, tanto lineales como angulares.

   - **Aceleración límite en x (a_x^T).**
   - **Aceleración límite en y (a_y^T).**
   - **Aceleración límite angular (α^T).**
   - **Velocidad angular máxima (ω_{max}^T).**
   - **Velocidad angular mínima (ω_{min}^T).**
   - **Velocidad máxima en x (Vx_{max}^T).**
   - **Velocidad mínima en x (Vx_{min}^T).**
   - **Velocidad máxima en y (Vy_{max}^T).**
   - **Velocidad mínima de entrada angular (ω_{in}^T).**
   - **Velocidad de escape (V_{esc}^T):** Velocidad usada para la conducción durante escapes en m/s. Debe ser negativo para que el robot sea capaz de escapar, ya que, si es positivo, el robot tratará de escapar hacia adelante.
   - **Habilitar funciones holonómicas ($Func_{hol}^T$):** Poner en verdadero si el robot que se va a utilizar es holonómico.
   - **Velocidad en y para robots holonómicos (Vy_{hol}^T):** Este parámetro solo es relevante si el parámetro anterior es verdadero.

2. El segundo grupo de parámetros es el de tolerancias y controlan la precisión en el objetivo final.

   - **Precisión en la orientación (Tol_{yaw}^T).**
   - **Precisión en la posición (Tol_{xy}^T).**
   - **Aviso de posición alcanzada (Tol_{latch}^T):** Es un booleano con el que si está en verdadero el robot dará vueltas sobre sí mismo una vez alcanzada la posición final.

3. El tercer grupo de parámetros es el de simulación de avance.

   - **Tiempo de simulación para el cálculo de la trayectoria (Sim^T_{time}):** La cantidad de tiempo que el robot toma para la simulación de trayectorias hacia adelante.
   - **Distancia lineal entre puntos ($Sim^T_{point_lin}$):** La distancia en metros que el planificador toma entre puntos a la hora de calcular una distancia.
   - **Distancia angular entre puntos ($Sim^T_{point_th}$):** La distancia, en radianes, que el planificador toma entre muestras angulares en una trayectoria dada. Por lo que si este valor es muy pequeño el planificador tomará cálculos a menor distancia. Este valor tiene que ser menor que la distancia total a la que el robot se moverá. Por lo que ahorramos ciclos y con ello carga computacional.

   El rango de valores óptimos de los siguientes parámetros para su utilización es entre 8 y 15.

   - **Muestras para la velocidad en x ($vx^T_{samples}$).**
   - **Muestras para la velocidad en y ($vy^T_{samples}$).**
   - **Muestras para la velocidad angular ($\omega^T_{samples}$).**
   - **Frecuencia del controlador (F^T_{cntl}):** La frecuencia con la que llamaremos al controlador en Hz.

4. Ahora pasamos al cuarto grupo de parámetros que controla la puntuación de cada trayectoria. En primer lugar, tenemos una ecuación con unos parámetros que nos calculan el costo, el cual nos dice el umbral de seguridad que tenemos para considerar que una trayectoria es segura para el robot.

$$Cost = t^T_{path} \cdot P + t^T_{goal} \cdot G + t^T_{obs} \cdot C \tag{1}$$

   P = distancia a la ruta desde el punto final de la trayectoria, en metros.

   G = distancia a la meta local desde el punto final de la trayectoria, en metros.

   C = costo máximo de obstáculos a lo largo de la trayectoria en costo de obstáculos (0-254).

   - **Coeficiente de tiempo de seguimiento de la ruta (t^T_{path}):** marca cuánto tiempo debe permanecer el controlador cerca de la ruta asignada.
   - **Coeficiente de tiempo para alcanzar el objetivo (t^T_{goal}):** determina cuanto tiempo debe estar el controlador intentando alcanzar el objetivo local. También controla la velocidad.
   - **Coeficiente de tiempo para alcanzar el objetivo (t^T_{obs}):** dice cuánto debe intentar el controlador evitar un obstáculo.
   - **Cambiar entre metros o celdas ($Swc^T_{measure}$):** con este parámetro decidimos si queremos que los parámetros anteriores los representamos en metro o en celdas. Por defecto esta en celdas.
   - **Distancia para calcular la trayectoria (L^T_{path}):** cómo de lejos tiene que mirar el robot hacia adelante al marcar diferentes trayectorias.
   - **Cambio de puntuación en los cálculos (L^T_{scor}):** decide si puntuar según el rumbo del robot hacia la ruta o su distancia desde la ruta.
   - **Puntuación en el tiempo simulado (t^T_{scor}):** cómo de lejos mira el robot hacia adelante en el tiempo a lo largo de la trayectoria simulada cuando se usa la puntuación de rumbo.
   - **Habilitar el algoritmo DWA (DWA^T_{en}):** permite usar el planificador DWA si el valor es verdadero.
   - **Habilitar publicación de la cuadrícula (En^T_{grid}):** dice si la cuadrícula de costos que el planificador utilizará para planificar será publicada o no. Se publica en el topic cost_cloud.
   - **Marco para poner el cost_cloud ($Frame^T_{id}$):** es el nombre del sistema de referencia al que se asocia la cuadrícula de costos, por defecto tendremos la odometría (odom).

5. El quinto grupo de parámetros controlan y previenen las oscilaciones.

- **Controlador de oscilaciones ($Cntl_{osc}^{T}$):** es la distancia que debe recorrer el robot en metros antes de que se restablezcan los indicadores de oscilación.

6. Por último, parámetros del planificador global, forman el último grupo de parámetros de este algoritmo.

   - **Elimina la ruta global (Del_{global}^{T}):** decide si eliminar la ruta global o no a medida que el robot avanza por el camino. Si es verdadero, cuando el robot se aleje más de 1 metro de la trayectoria calculada por el planificador global éste calculará puntos locales fuera del punto final del plan.

2.3.2. Planificador Dynamic Window Approach (DWA)

El planificador Dynamic Window Approach (DWA) es un algoritmo que podemos encontrar en un paquete dentro de ROS, el cual proporciona un controlador que se encarga de mover una base móvil en el plano. Este controlador sirve para conectar el algoritmo al robot. Usando un mapa, DWA crea una trayectoria cinemática para que el robot llegue desde el principio hasta la ubicación del objetivo. Durante la trayectoria, el planificador crea una función de costos alrededor del robot. Ésta es representada como un mapa de cuadrícula. Esta función de costos codifica si atravesar las celdas de la cuadrícula, en función de estos costos. El trabajo del controlador es usar esta función de costos para determinar las velocidades d_x, d_y, d_{theta} que se van a enviar al robot [1].

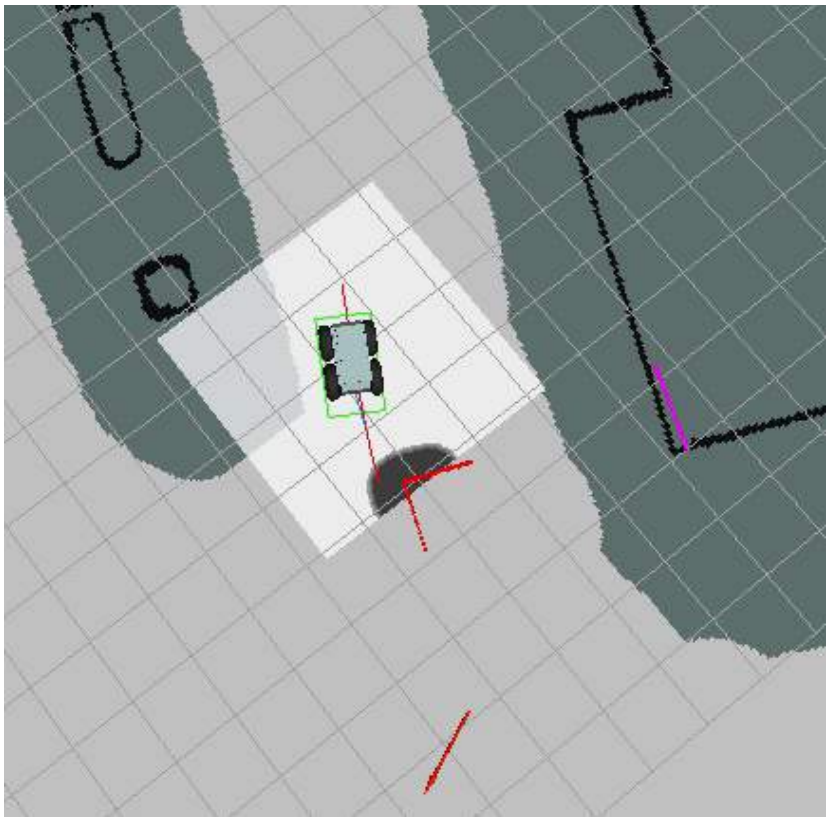


Figura 10: Ejemplo de funcionamiento del planificador DWA.

Resumiendo, ayudándonos de la Figura 10, la idea básica de cómo funciona el planificador local DWA es la siguiente:

⇛ Muestra discretamente del espacio de control del robot.

⇛ Para cada velocidad muestreada, se realizan simulaciones hacia adelante desde el estado actual del robot para predecir lo que sucedería si se aplicara la velocidad muestreada.

⇛ Evalúa cada trayectoria resultante de la simulación hacia adelante.

⇛ Descarta trayectorias ilegales.

⇛ Elije la trayectoria con la puntuación más alta y envía las velocidades asociadas.

⇛ Por último, se limpia todo y se repite el algoritmo.

Como podemos ver en la Figura 10, el robot planifica diferentes trayectorias por las que podría circular, y el algoritmo se encarga de descartar las trayectorias que lleven a la colisión y quedarse con aquellas que cumplan una trayectoria que satisfaga los parámetros con los que está configurado el algoritmo [18] [8].

Este planificador tiene 6 grupos de parámetros que nos ayudarán a realizar el control del algoritmo:

1. El primer grupo está formado por los parámetros de configuración del robot. Este conjunto de parámetros modifica las velocidades y aceleraciones máximas y mínimas, tanto lineales como angulares.

   - **Aceleración límite en x (a_x^{DWA}).**
   - **Aceleración límite en y (a_y^{DWA}).**
   - **Aceleración angular límite (α^{DWA}).**
   - **Velocidad máxima translacional (v_{max}^{DWA}).**
   - **Velocidad mínima translacional (v_{min}^{DWA}).**
   - **Velocidad máxima en x (vx_{max}^{DWA}).**
   - **Velocidad mínima en x (vx_{min}^{DWA}).**
   - **Velocidad máxima en y (vy_{max}^{DWA}).**
   - **Velocidad mínima en y (vy_{min}^{DWA}).**
   - **Velocidad rotacional máxima (ω_{max}^{DWA}).**
   - **Velocidad rotacional mínima (ω_{min}^{DWA}).**

2. El segundo grupo de parámetros es el de tolerancias en el objetivo. Controlan la precisión en el objetivo final.

   - **Precisión en la orientación (Tol_{yaw}^{DWA}).**
   - **Precisión en la posición (Tol_{xy}^{DWA}).**
   - **Aviso de posición alcanzada (Tol_{latch}^{DWA}):** booleano con el que si está en verdadero el robot dará vueltas sobre sí mismo una vez alcanzada la posición final.

3. El tercer grupo de parámetros es el de simulación de avance.

   - **Tiempo de simulación de la trayectoria (Sim_{time}^{DWA}):** la cantidad de tiempo que el robot toma para la simulación de trayectorias hacia adelante.
   - **Distancia entre puntos (Sim_{point}^{DWA}):** la distancia en metros, que el planificador toma entre puntos a la hora de calcular una distancia. Por lo que si este valor es muy pequeño el planificador tomará cálculos a menor distancia. Este valor tiene que ser menor que la distancia total a la que el robot se moverá. Por lo que ahorramos ciclos y con ello carga computacional.

   El rango de valores óptimos para la utilización de estos parámetros es entre 8 y 15, tal y como vimos en la Sección 2.3.1.

   - **Muestras para la velocidad en x ($vx_{samples}^{DWA}$).**
   - **Muestras para la velocidad en y ($vy_{samples}^{DWA}$).**
   - **Muestras para la velocidad angular ($\omega_{samples}^{DWA}$).**
   - **Frecuencia del controlador (F_{cntl}^{DWA}):** la frecuencia con la que llamaremos al controlador en Hz.

4. Ahora pasamos al cuarto grupo el cual controla la puntuación de trayectoria.

   En primer lugar, tenemos una ecuación con unos parámetros que nos calculan el costo, el cual nos dice el umbral de seguridad que tenemos para considerar que una trayectoria es segura para el robot.

$$Cost = t_{path}^{DWA} \cdot P + t_{goal}^{DWA} \cdot G + t_{obs}^{DWA} \cdot C \tag{2}$$

   P = distancia a la ruta desde el punto final de la trayectoria, en metros.

   G = distancia a la meta local desde el punto final de la trayectoria, en metros.

   C = costo máximo de obstáculos a lo largo de la trayectoria en costo de obstáculos (0-254).

   - **Coeficiente de tiempo de seguimiento de la ruta (t_{path}^{DWA}):** marca cuánto tiempo debe permanecer el controlador cerca de la ruta asignada.
   - **Coeficiente de tiempo para alcanzar el objetivo (t_{goal}^{DWA}):** determina cuánto tiempo debe estar el controlador intentando alcanzar el objetivo local. También controla la velocidad.
   - **Coeficiente de tiempo para evitar un obstáculo (t_{obs}^{DWA}):** dice cuánto debe intentar el controlador evitar un obstáculo.
   - **Distancia del robot al sub-objetivo (L_{point}^{DWA}):** es la distancia desde el punto central del robot para colocar un punto de objetivo local, en metros.
   - **Tiempo de paro antes de una colisión ($Stop_{time}^{DWA}$):** es el tiempo en el que el robot debe detenerse antes de una colisión para que la trayectoria sea válida, en segundos.
   - **Valor absoluto de la velocidad (V_{abs}^{DWA}):** a la cual se comienza a escalar la huella del robot, en m/s.
   - **Factor máximo ($scal_{max}^{DWA}$):** factor máximo para escalar la huella del robot.
   - **Habilitar publicación de la cuadrícula (En_{grid}^{DWA}):** dice si la cuadrícula de costos que el planificador utilizará para planificar será publicada o no. Se publica en el topic cost_cloud.

5. El quinto grupo de parámetros controlan y previenen las oscilaciones.

   - **Controlador de oscilaciones ($Cntl_{osc}^{DWA}$):** la distancia que debe recorrer el robot en metros antes de que se restablezcan los indicadores de oscilación.

6. Por último, parámetros del planificador global, forman el último grupo de parámetros de este algoritmo.

   - **Elimina la ruta global (Del_{global}^{DWA}):** decide si eliminar la ruta global o no a medida que el robot avanza por el camino. Si es verdadero, cuando el robot se aleje más de 1 metro de la trayectoria calculada por el planificador global éste calculará puntos locales fuera del punto final del plan.

2.3.3. Planificador Elastic BAND (EBAND)

El planificador EBAND deforma la trayectoria local basada en fuerzas internas y externas. Estas fuerzas son utilizadas para optimizar el camino y repeler obstáculos. Mediante las fuerzas internas, el algoritmo es capaz de contraer el camino, y así consigue una conducción por el camino más corto entre el punto inicial y el objetivo final. En cambio, las fuerzas externas se encargan de evitar que en el camino haya obstáculos.

Este planificador lo que hace es buscar mínimos locales, es decir, si le decimos al robot que vaya a un objetivo y éste tiene un obstáculo en el medio, éste decide qué es más corto, si esquivarlo por la izquierda o por la derecha. Este planificador admite tanto robots de accionamiento diferencial como omnidireccionales [3]. Nos apoyaremos de la Figura 11 para entender el funcionamiento del planificador de mejor manera vamos a explicar la lógica que éste sigue.

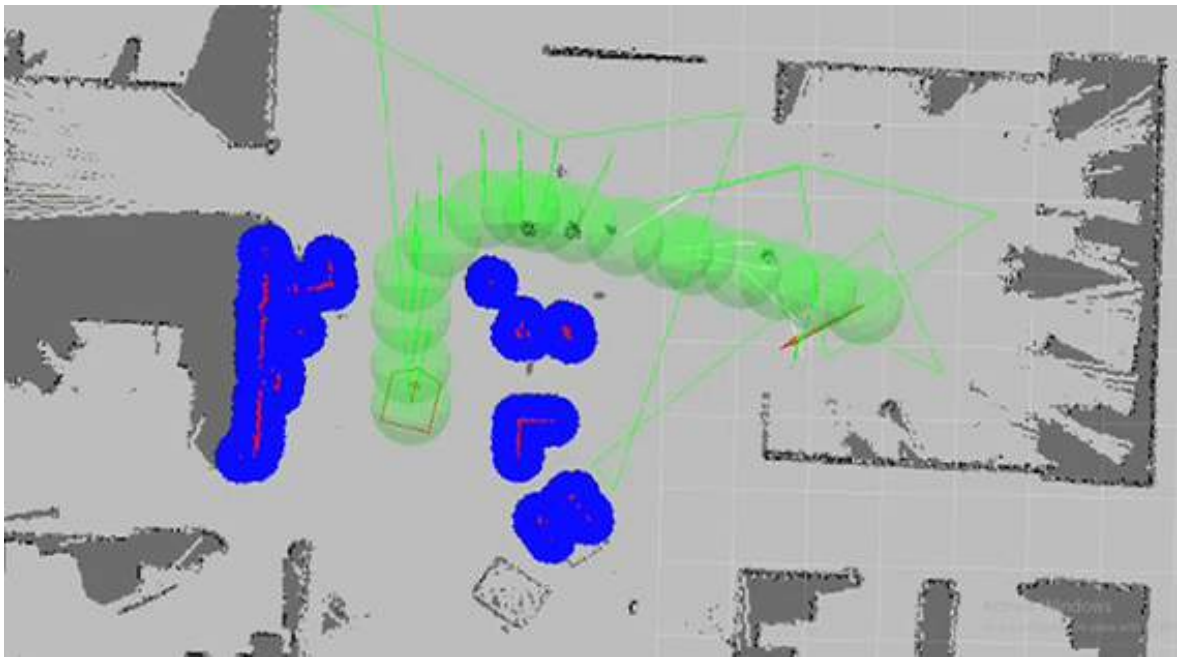


Figura 11: Ejemplo del funcionamiento del planificador EBAND.

El planificador calcula una banda elástica, con esto intenta seguir la ruta generada conectando los puntos centrales de la banda. Para el cálculo de la velocidad, se realiza el siguiente algoritmo:

1. En primer lugar, si el robot está dentro de la tolerancia establecida en la posición, éste se queda dando vueltas sobre sí mismo hasta que alcanza la orientación objetivo.

2. Pero si el robot está fuera de esta tolerancia de posición, por ejemplo, si le decimos que se desplace a un punto, el robot resta la orientación actual menos la orientación del siguiente punto. Ésta se compara con un valor establecido, y si es menor, el robot ejecutará componentes de velocidades con las que forma un arco para ir al siguiente punto y así hasta alcanzar la meta y empieza a realizar el primer paso.

3. Por último, si la resta calculada en el paso anterior es mayor que el valor establecido el robot lo que hace es empezar a girar sobre sí mismo hasta que la diferencia de estas dos orientaciones se vuelve menor que el valor establecido, en ese momento se ejecuta el paso 2 [3].

En cuanto a los parámetros, en este planificador tenemos 4 bloques:

1. La primera parte consiste en parámetros comunes correspondientes a la precisión en el objetivo final del robot y a unas velocidades.

   - **Precisión en la posición (Tol_{xy}^{EBAND}).**
   - **Precisión en la orientación (Tol_{yaw}^{EBAND}).**
   - **Velocidad angular de detección (ω_{stop}^{EBAND}):** velocidad angular mínima que determina si el robot debe detenerse para evitar ciclos límite o bloqueos.
   - **Velocidad lineal de detección (v_{stop}^{EBAND}):** velocidad lineal mínima que determina si el robot debe detenerse para evitar ciclos límite o bloqueos.

2. El segundo grupo son parámetros de visualización. En este grupo sólo tenemos 1 parámetro.

   - **Tiempo de vida del marcador ($Lifetime_{marker}^{EBAND}$):** tiempo de vida que tendrán los marcadores de visualización.

3. Pasamos al tercer grupo, donde veremos los parámetros de la banda elástica.

   - **Límite geométrico de la burbuja (Bub_{lim}^{EBAND}):** limite geométrico de la burbuja con respecto a la distancia de la burbuja pequeña.

- **Distancia mínima entre burbujas (Bub_{dist}^{EBAND}):** distancia de conectividad entre las burbujas.
- **Expansión de la burbuja (Bub_{exp}^{EBAND}).**
- **Ganancia de la fuerza interna (F_{int}^{EBAND}).**
- **Ganancia de la fuerza externa (F_{ext}^{EBAND}).**
- **Optimización del número de iteraciones ($Iter_{opt}^{EBAND}$):** controla el número de iteraciones de la optimización del EBAND.
- **Número de iteraciones para el cálculo de fuerzas ($Iter_{force}^{EBAND}$):** número de iteraciones para buscar el equilibrio entre las fuerzas internas y externas.
- **Impulso de equilibrio máximo (Eq_{max}^{EBAND}).**
- **Fuerza mínima significativa (F_{min}^{EBAND}):** fuerza que consideramos significativa para realizar los cálculos, si se modifica el valor el robot cambiará su comportamiento.
- **Peso del mapa de costos ($W_{costmap}^{EBAND}$):** peso del mapa de costos. A mayor valor el mapa de costos adquiere más peso en el cálculo de la trayectoria.

4. Por último, los parámetros del cuarto grupo son los encargados de controlar la trayectoria. Aquí vemos parámetros como velocidades mínimas y máximas, lineales y angulares, aceleraciones lineales y angulares, etc.

   - **Velocidad máxima lineal (v_{max}^{EBAND}).**
   - **Velocidad máxima angular (ω_{max}^{EBAND}).**
   - **Velocidad mínima lineal (V_{min}^{EBAND}).**
   - **Velocidad mínima angular (ω_{min}^{EBAND}).**
   - **Aceleración máxima (a_{max}^{EBAND}).**
   - **Aceleración máxima lineal (a_{max}^{EBAND}).**
   - **Aceleración máxima angular(α_{max}^{EBAND}).**
   - **Velocidad mínima de entrada angular (ω_{in}^{EBAND}).**
   - **Velocidad mínima de entrada lineal(V_{in}^{EBAND}).**
   - **Constante proporcional (k_{prop}^{EBAND}):** controla el PID del algoritmo.
   - **Constante derivativa (k_{damp}^{EBAND}):** controla el PID del algoritmo.
   - **Tasa de control de trayectoria ($Ctrl_{rate}^{EBAND}$):** controla la tasa de control, con este parámetro controlamos la trayectoria por la que va a circular el robot. Si el valor es bajo el robot va por la trayectoria más cercana, aunque tenga que pasar por sitios muy estrechos o con muchos obstáculos; si es muy grande el robot calcula una ruta más segura aunque el camino sea más largo.
   - **Masa virtual ($mass_{virtual}^{EBAND}$).**
   - **Umbral de corrección de rotación (Th_{corr}^{EBAND}).**
   - **Habilitar la conducción diferencial ($Diff_{drive}^{EBAND}$):** este valor debe ser verdadero si el robot es de conducción diferencial.
   - **Multiplicador del radio de las burbujas ($Mult_{vel}^{EBAND}$).**
   - **Multiplicador del umbral de rotación ($Mult_{rot}^{EBAND}$):** coeficiente que utiliza para multiplicar el umbral de rotación.
   - **Deshabilitar la histéresis ($Hyst_{dis}^{EBAND}$):** este último parámetro determina si intentar acercarse a la meta en caso de superar la tolerancia. Si este valor es muy pequeño, puede provocar que la diferencia nunca sea menor que este valor y por tanto el robot se quede girando sobre sí mismo sin avanzar hacia la meta.

2.3.4. Planificador Timed Elastic Band (TEB)

El siguiente algoritmo a estudiar es el TEB, este planificador funciona mediante la deformación y la optimización de la trayectoria global, es decir, la trayectoria incluye información temporal. Este algoritmo va a minimizar la función de costo en vez de aplicar y generar fuerzas, con esto se consiguen que las trayectorias tengan un tiempo óptimo (ver Figura 12). Esta información temporal proporcionada por el algoritmo permite incorporaciones de restricciones dinámicas imitando a un controlador predictivo.

Este algoritmo admite robots de conducción diferencial, robots con forma de coches y omnidireccionales, ya que este planificador, como más adelante veremos, podemos decir la topología del robot, con esto el algoritmo consigue optimizar trayectorias paralelas con el fin de superar parcialmente el problema de los mínimos locales. Esto sólo afecta al planificador local, ya que por las limitaciones de la CPU seguimos dependiendo del planificador global.

Este algoritmo consta de modo de seguimiento de ruta, lo que minimiza la distancia al plan global, en vez de minimizar el tiempo de transición. El principal problema de este algoritmo es que si se programa para funcionar al máximo de su capacidad, requerirá una gran carga computacional, y por tanto la resolución del mapa de costos es limitada [26].

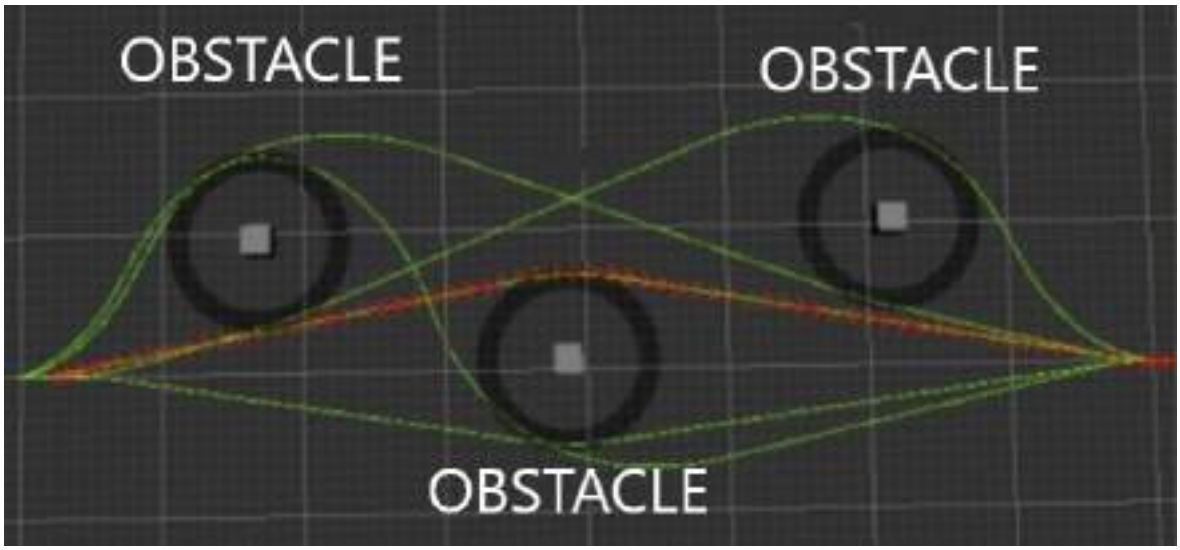


Figura 12: Ejemplo de funcionamiento del planificador TEB.

Este planificador tiene 7 grupos de parámetros, es el planificador que más parámetros tiene, por lo que su estudio es el más extenso.

1. El primer grupo de parámetros son de configuración del robot. En este grupo tenemos todos los límites de velocidades y aceleraciones, tanto lineales como angulares.

   - **Aceleración límite en x (a_x^{TEB}).**
   - **Aceleración límite angular (α^{TEB}).**
   - **Velocidad máxima en x (v_x^{TEB}).**
   - **Velocidad máxima hacia atrás en x ($v_{x,b}^{TEB}$).**
   - **Velocidad máxima angular (ω_{max}^{TEB}).**

   Los siguientes parámetros sólo son relevantes para robots con formas similares a un coche.

   - **Radio mínimo de giro (R_{min}^{TEB}):** radio de giro mínimo de un robot (poner a 0 para un robot diferencial).
   - **Distancia entre ejes (L^{TEB}):** la distancia entre el eje trasero y delantero. El valor puede ser negativo para robots con ruedas traseras, pero para que funcione el parámetro ω^{TEB} debe ser verdadero. Este parámetro sustituye la velocidad de rotación en el mensaje de velocidad ordenada por el ángulo de dirección correspondiente.

- **Comando de cambio de la velocidad de rotación (ω^{TEB}).**

Los siguientes parámetros son relevantes sólo para robots holonómicos. La velocidad en este caso no puede ser 0.

- **Velocidad máxima en y (v_y^{TEB}).**
- **Aceleración límite en y (a_y^{TEB}).**

Los siguientes parámetros son relevantes para el modelo de huella usado para la optimización.

- **Modo de la huella ($mode_{foot}^{TEB}$):** especifica la huella del robot usado para la optimización. Hay 5 modos: punto, circular, linea, dos círculos y polígono. El tipo de modelo influye significativamente en el tiempo de computación requerida.

Si hemos seleccionado la opción circular, hay que indicar el radio del círculo, donde el centro se encuentra en los ejes de rotación del robot.

- **Radio del círculo (R_c^{TEB}).**

Si hemos seleccionado la opción line, hay que indicar el principio y el final del segmento.

- **Inicio de la línea ($line_i^{TEB}$).**
- **Final de la línea ($line_f^{TEB}$).**

Si hemos seleccionado la opción dos círculos, necesitamos conocer el radio del circulo frontal y trasero.

- **Radio círculo frontal y trasero (rad_{front}^{TEB} y rad_{rear}^{TEB}).**
- **Offset frontal y trasero ($Offset_{front}^{TEB}$ y $Offset_{rear}^{TEB}$):** distancia que hay del centro al frente del círculo a lo largo del eje x, y cuánto se desplaza el centro del círculo posterior a lo largo del eje x del robot. En ambos casos se asume que el eje de rotación del robot está ubicado en las coordenadas [0.0].

Si hemos seleccionado la opción polígono, debemos especificar la lista de los vértices que tenga el robot. Como condición siempre hay que cerrar el polígono y no repetir el primer y el último vértice.

- **Vértices del polígono (ver_{pol}^{TEB}).**

- **Comprobador de viabilidad ($Update_{foot}^{TEB}$):** actualiza la huella antes de comprobar la viabilidad de la trayectoria (si es verdadero).

2. El segundo grupo de parámetros son los relacionados con las tolerancias en el objetivo.

   - **Tolerancia en la posición (tol_{xy}^{TEB}).**
   - **Tolerancia en la orientación (tol_{yaw}^{TEB}).**
   - **Restricción de la velocidad en la meta (vel_{free}^{TEB}):** elimina la restricción de la velocidad de la meta, de tal modo que el robot pueda llegar a la meta con la máxima velocidad.

3. El tercer grupo nos va a ayudar a configurar la trayectoria del robot.

   - **Resolución temporal ($resTemp^{TEB}$):** resolución temporal deseada de la trayectoria, la trayectoria no esta fijada en $resTemp^{TEB}$ ya que la resolución temporal es parte de la optimización, pero la trayectoria cambiará de tamaño entre iteraciones si $resTemp^{TEB} \pm resTemp_{hys}^{TEB}$ se viola.
   - **Histéresis de la resolución temporal ($resTemp_{hys}^{TEB}$):** histéresis para realizar el cambio de resolución temporal (se recomienda que su valor sea $0{,}1 \cdot dt\_ref$).
   - **Muestras mínimas ($Samples_{min}^{TEB}$):** Número mínimo de muestras que debe de ser como mínimo mayor de 2.

- **Orientación de los sub-objetivos ($Subgoals_{yaw}^{TEB}$):** añade información sobre la orientación de los sub-objetivos locales proporcionados por el planificador global, si es verdadero, sino, sólo se aporta información sobre la posición.
- **Desviación de los puntos ($viapoint_{weight}^{TEB}$):** si el valor es positivo los puntos se desvían del plan global (modo de seguimiento de ruta). Si es negativo estará desactivado. Para ajustar la intensidad hay que modificar el parámetro.
- **Separación mínima entre puntos ($viapoint_{sep}^{TEB}$):** el valor determina la resolución de la ruta de referencia (separación mínima entre cada 2 puntos intermedios consecutivos a lo largo del plan global).
- **Limitación de la longitud de los planes globales (lim_{global}^{TEB}):** especifica la máxima longitud del subconjunto de los planes globales cogidos en la cuenta para la optimización. La longitud actual que se determina por la conjunción lógica del tamaño mapa de costos local y este es el limite máximo. Si este parámetro está a 0 o negativo desactiva esta limitación.
- **Reset de trayectoria ($reset_{path}^{TEB}$):** reinicia la trayectoria si un objetivo previo es actualizado con una separación de más valor que el especificado.
- **Verificación de viabilidad ($check_{feasibility}^{TEB}$):** especifica hasta qué posición se debe verificar la viabilidad en cada intervalo de muestreo.
- **Publicación de comentarios ($pub_{comments}^{TEB}$):** publica comentarios del planificador que contengan la trayectoria completa y una lista de obstáculos activos. Este parámetro será útil para evaluar la trayectoria calculada.
- **Modificación del horizonte ($Modification_{horizon}^{TEB}$):** si es verdadero, encoge el horizonte temporal (50 %) en caso de detectar daños de forma automática.
- **Inicialización de trayectorias ($path_{init}^{TEB}$):** si es verdadero, las trayectorias deben ser inicializadas con movimientos hacia atrás en caso de que el objetivo este detrás del inicio sin el mapa de costos local (esto sólo se recomienda si el robot está equipado con sensores traseros).
- **Habilitar longitud de arco exacta ($longArc_{exact}^{TEB}$):** si es verdadero, el planificador usa la longitud de arco exacta en velocidad, aceleración y la tasa de giro. Esto incrementa el tiempo de CPU. En caso contrario se utiliza la aproximación euclídea.
- **Duración mínima para la reducción de horizonte ($Hduration_{min}^{TEB}$):** especifica la duración mínima para reducir el horizonte en caso de que una trayectoria inviable sea detectada (para activar este parámetro, es necesario que $Modification_{horizon}^{TEB}$ sea verdadero).

4. Pasamos al cuarto grupo de parámetros del algoritmo TEB, el cual controla los obstáculos, es decir, cómo detectarlos y cómo responder en consecuencia.

   - **Separación mínima a los obstáculos (sep_{min}^{TEB}).**
   - **Habilitar obstáculos costmap ($ObsCM_{enable}^{TEB}$):** especifica si los obstáculos del mapa de costos local deben ser tomados en cuenta. Cada celda está marcada como obstáculo considerado como point-obstacle. Por tanto, se recomienda no escoger una resolución del mapa de costos demasiado pequeña.
   - **Obstáculos detrás del robot Obs_{behind}^{TEB}):** limitar los obstáculos del mapa de costos local ocupados que se tienen en cuenta para la planificación detrás del robot (especifica la distancia en metros).
   - **Trayectoria alrededor de obstáculos Obs_{pose}^{TEB}):** el valor de este parámetro influye ligeramente la trayectoria alrededor de los obstáculos (a mayor longitud de la trayectoria) para conectar todas las poses con cada obstáculo. Cada posición de obstáculo se adjunta a la pose más cercana en la trayectoria para mantener una distancia. Vecinos adicionales también se pueden tener en cuenta.
   - **Distancia de amortiguación ($inflation_{dist}^{TEB}$):** zona de amortiguación alrededor de obstáculos con penalizaciones de costos diferentes a 0 (debe ser mayor que sep_{min}^{TEB} para que tenga efecto).
   - **Incluir obstáculos dinámicos (Obs_{dyn}^{TEB}):** si es verdadero, el movimiento de los obstáculos con velocidad distinta a 0 (a través de obstáculos proporcionados por el usuario en el topic /obstacles u obtenido del costmap_converter) es predicho y considerado durante la optimización vía un modelo de velocidad constante.

- **Estrategia de la trayectoria ($path_{legacy}^{TEB}$):** la estrategia de la trayectoria de posición de conexión con obstáculos para que la optimización sea modificada. Se puede cambiar la estrategia anterior/previa poniendo el parámetro a verdadero. Estrategia anterior: para cada obstáculo, se encuentra la posición del algoritmo más cercana; estrategia previa: para cada posición del algoritmo, se encuentran solo los obstáculos relevantes.
- **Importancia de los obstáculos relevantes (Obs_{rel}^{TEB}):** intenta conectar sólo los obstáculos relevantes con la trayectoria discretizada durante la optimización. Pero todos los obstáculos sin una distancia específica son forzados a ser incluidos (como un múltiplo de sep_{min}^{TEB}). Por ejemplo, si escogemos 2.0 para hacer cumplir los obstáculos de consideración dentro de un radio de $2{,}0 \cdot sep_{min}^{TEB}$ (este parámetro es usado sólo si el parámetro $path_{legacy}^{TEB}$ está desactivado).
- **Ignorar todos los obstáculos (Obs_{cutoff}^{TEB}):** con este parámetro conseguimos que todos los obstáculos sean ignorados durante la optimización. Pero aun así, el parámetro Obs_{rel}^{TEB} se procesa en primer lugar.

Los siguientes parámetros son relevantes sólo si se desean plugins de costmap_converter:

- **Convertir las celdas en puntos ($CM_{converter}^{TEB}$):** define el nombre del plugin para convertir las celdas del mapa de costos en puntos/líneas/polígonos. Para deshabilitar la conversión hay que poner un string vacío, de modo que las celdas son tratadas como puntos-obstáculos.
- **Llamar a la cola de callbacks ($Queue_{call}^{TEB}$):** si es verdadero, el costmap_converter invoca su cola de callbacks en diferentes threads.
- **Frecuencia de refresco del costmap (CM_{rate}^{TEB}):** tasa que define la frecuencia en Hz con la que el plugin del costmap_conerter procesa el mapa de costos actual (el valor no debe ser mucho más alto que la actualización del mapa de costos).

5. Pasamos al quinto grupo de parámetros, el cual controla la optimización del algoritmo. Este conjunto debe configurarse correctamente para que el robot funcione de la mejor forma posible.

   - **Número de iteraciones (num_{it}^{TEB}):** número de iteraciones de solucionador reales llamadas en cada iteración del bucle externo.
   - **Optimizador interno ($Optim_{in}^{TEB}$):** cada iteración automática del bucle externo vuelve a dimensionar la trayectoria acorde a la resolución temporal deseada e invoca al optimizador interno (que realiza num_{it}^{TEB}). El número total de iteraciones del solucionador en cada ciclo de planificación es por lo tanto el producto de ambos valores.
   - **Penalización para aproximaciones ($Pen_{epsilon}^{TEB}$):** añade un pequeño margen seguro para penalizar funciones para aproximaciones de restricción.
   - **Optimizador de velocidad máxima en x (W_{vx}^{TEB}):** el peso de optimización para satisfacer la velocidad máxima permitida de translación.
   - **Optimizador de velocidad máxima angular (W_{ω}^{TEB}):** el peso de optimización para satisfacer la velocidad máxima permitida angular.
   - **Optimizador de aceleración límite en x (W_{ax}^{TEB}):** el peso de optimización para satisfacer la aceleración máxima permitida de translación.
   - **Optimizador de aceleración límite angular (W_{α}^{TEB}):** el peso de optimización para satisfacer la aceleración máxima permitida angular.
   - **Optimizador de la cinemática no holonómica (W_{kin}^{TEB}):** el peso de optimización para satisfacer la cinemática no holonómica (este parámetro debe ser elevado, incluso con un valor de 1000 no implica una mala condición de la matriz debido a los pequeños valores de costo bruto en comparación con otros casos). Si reducimos este valor significativamente, ajustaremos la compensación entre el movimiento longitudinal y el movimiento lateral no compatible.
   - **Optimizador de conducción hacia adelante (W_{Vfw}^{TEB}):** el peso de optimización para forzar al robot a elegir solo direcciones hacia adelante. Con un pequeño peso (valor de 1) aun seguiría conduciendo hacia atrás. Y con un valor por encima de 1000 evita casi la conducción hacia atrás pero no lo garantiza.

- **Optimizador del radio de giro mínimo ($W^{TEB}_{TurnRad}$):** el peso de optimización para hacer cumplir un radio de giro mínimo (solo se utiliza para robots con forma de coche).
- **Optimizador de tiempo (W^{TEB}_{time}):** el peso de optimización para contener la trayectoria w,r,t translación/tiempo de ejecución.
- **Peso para mantener la distancia mínima ($W^{TEB}_{obstacle}$):** el peso de optimización para mantener una distancia mínima de los obstáculos.
- **Optimizador de los puntos viables (W^{TEB}_{points}):** el peso de optimización para minimizar la distancia a los puntos viables.
- **Optimizador de la inflación ($W^{TEB}_{inflation}$):** peso de optimización para la penalización de la inflación (debe ser pequeño).
- **Optimizador de adaptación de valores ($W^{TEB}_{inflation}$):** algunos pesos especiales (actualmente $W^{TEB}_{obstacle}$) se escalan repetidamente por este factor en cada iteración exterior TEB ($weight\_new = weight\_old \cdot factor$). Este parámetro incrementa los pesos de forma iterativa en lugar de establecer un valor enorme que a priori conduce a mejores condiciones numéricas del problema de optimización subyacente.

6. Pasamos al sexto grupo, donde estudiaremos los parámetros de planificación paralela en topologías distintivas.

   - **Planificación paralela ($Plan^{TEB}_{paralel}$):** activa la planificación paralela en topologías distintivas (requiere muchos recursos de CPU).
   - **Habilitar multithreading (En^{TEB}_{mTh}):** activa multitreads para planificar cada trayectoria en un thread diferente. Es decir, dividimos las ramas de compilación que se están ejecutando en el algoritmo.
   - **Número máximo de trayectorias (Num^{TEB}_{path}):** especifica el número máximo de trayectorias distintivas tenidas en cuenta (cargas computacionales).
   - **Costo de las trayectorias ($Cost^{TEB}_{path}$):** especifica cuánto costo de trayectoria debe tener un nuevo candidato w,r,t con una trayectoria previa seleccionada para que sea seleccionada (se selecciona si: $new\_cost < old\_cost \cdot factor$).
   - **Escalado del costo de obstáculos ($Cost^{TEB}_{obs}$):** escalado extra de los términos de costo de los obstáculos sólo para seleccionar el mejor candidato.
   - **Escalado de los puntos viables ($Cost^{TEB}_{point}$):** escalado extra de los términos de costo de los puntos viables sólo para seleccionar el mejor candidato.
   - **Costo de tiempo ($Cost^{TEB}_{time}$):** si es verdadero, el costo de tiempo (suma de los cuadrados de las diferencias de tiempos) es reemplazado por el tiempo de transición total (suma de tiempos de diferencia).
   - **Número de muestras generadas ($Num^{TEB}_{samplesG}$):** especifica el número de muestras generadas para crear el gráfico de mapa vial.
   - **Ancho de una región de cálculo ($Width^{TEB}_{area}$):** puntos clave aleatorios que se muestran en una región rectangular entre el inicio y el objetivo. Especifica el ancho de una región en metros.
   - **Parámetro interno de escala ($Presc^{TEB}_{int}$):** parámetro interno de escala (H-signature) que se utiliza para distinguir entre clases de homotopía. Reducir este parámetro sólo si se observan problemas con muchos obstáculos en el mapa de costos local, no elegir un valor extremadamente bajo, de lo contrario los obstáculos no podrán ser distinguidos entre uno a otro ($0{,}2 < valor <= 1$).
   - **Umbral del parámetro interno de escala ($Presc^{TEB}_{Th}$):** si la diferencia entre las partes reales y complejas están por debajo de este umbral, se asumirá que esos 2 H-signatures son iguales.
   - **Umbral de la exploración (Obs^{TEB}_{Th}):** especifica el valor del producto escalar entre el rumbo del obstáculo y el rumbo del objetivo para tenerlos en cuenta para la exploración.
   - **Visualizar la gráfica de trayectorias ($Graph^{TEB}_{visual}$):** visualiza la gráfica que se crea para explorar las trayectorias distintivas.

- **Tiene en cuenta todas las trayectorias (All^{TEB}_{points}):** si es verdadero, todas las trayectorias de diferentes topologías se adjuntan para poner los puntos viables, de lo contrario, sólo la trayectoria que comparte la misma topología como plan inicial/global será conectada con ello.
- **Tiempo máximo para cambiar de clase ($Timeout^{TEB}_{class}$):** especifica la duración de tiempo en segundos que debe expirar antes de que se permita un cambio a una nueva clase de equivalencia.

7. Por último, para finalizar con este extenso planificador, vamos a pasar a analizar el séptimo grupo de parámetros.

   - **Nombre del topic de la odometría ($Odom^{TEB}_{topic}$):** nombre del topic del mensaje de la odometría, entregado por el driver del robot o por el simulador.
   - **Marco de planificación global (Map^{TEB}_{frame}):** sistema de coordenadas al que se referencia la planificación global. En caso del mapa estático, este parámetro debe ser cambiado a "/map".

2.3.5. Planificador Active Scene Recognition (ASR)

Este planificador local es un algoritmo basado en el algoritmo "Follow the Carrot", es decir, la ruta local intenta seguir lo máximo posible a la ruta global. Con este plan global calcula las velocidades necesarias, y también lo utiliza para evitar obstáculos, por lo que hay que actualizar el plan global de forma continuada.

El planificador seguirá este plan global hasta que se alcanza o bien la distancia o bien el ángulo máximo. El robot avanzará siempre que estemos por debajo del ángulo máximo, si esto no se cumple, el robot retrocederá por el plan global hasta que esta condición se cumpla.

Este planificador además tiene una función de seguridad, en la que el robot nunca se va a dirigir a una zona cerca de un obstáculo. El algoritmo verifica el mapa de costos global y si el robot se encuentra muy cerca de un obstáculo éste se detendrá hasta que le demos un nuevo objetivo [20]. A continuación, se muestra un ejemplo de su funcionamiento en la Figura 13.

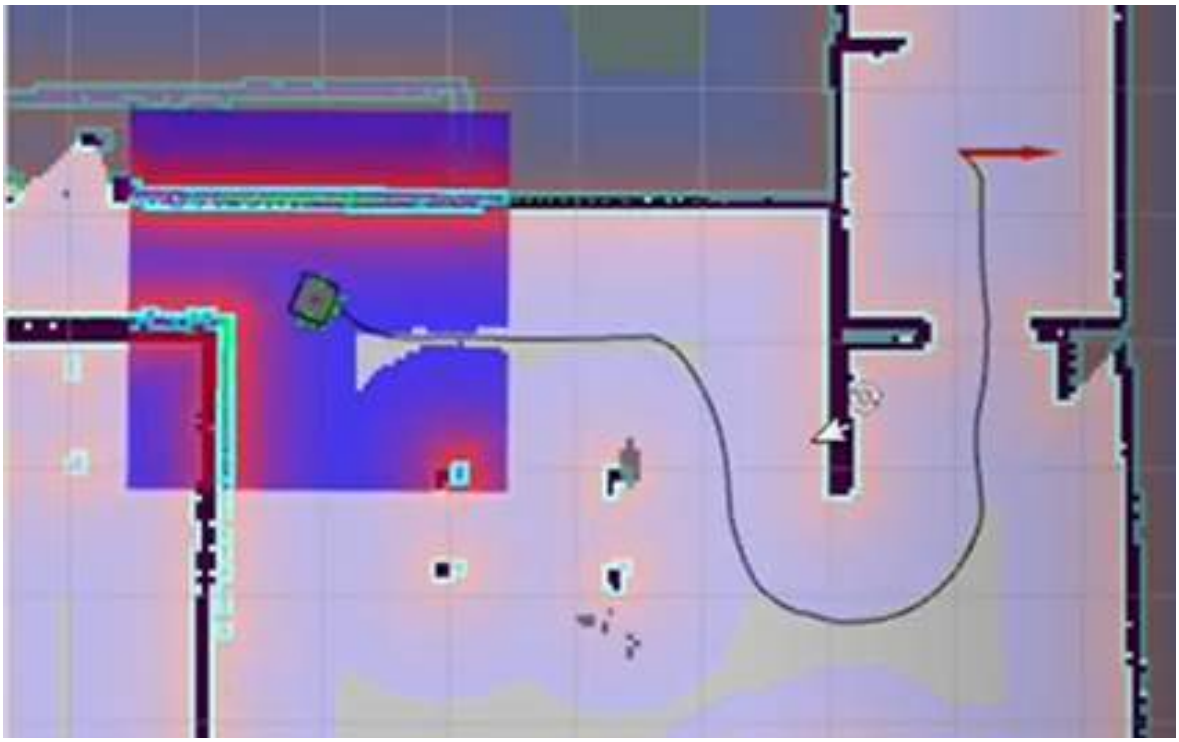


Figura 13: Ejemplo de funcionamiento del planificador ASR.

En resumen, este planificador funciona siguiendo 3 pasos:

1. Girar sobre el terreno a la orientación del plan global.
2. Conducir hacia la meta.
3. Girar en el lugar a la orientación de la meta.

En cuanto a los parámetros, veremos que no consta de demasiados. Vamos a ver cómo se comporta cada uno.

- **Velocidad máxima en x (v_{max}^{ASR}).**
- **Velocidad angular máxima (ω_{max}^{ASR}):** el planificador intenta alcanzar esta velocidad lo más rápido posible.
- **Velocidad angular mínima (ω_{min}^{ASR}):** se utiliza para garantizar una orientación adecuada en el objetivo.
- **Aceleración en x (a_x^{ASR}).**
- **Aceleración en z (a_z^{ASR}).**
- **Precisión en la posición (Tol_{xy}^{ASR}).**
- **Precisión en la orientación (Tol_{yaw}^{ASR}).**
- **Desaceleración de la rotación ($Slow_{down}^{ASR}$):** un factor que influye en la desaceleración de la rotación. Debería compensar imprecisiones en el control del motor. Si el parámetro es mayor que 0. la desaceleración comienza más tarde.
- **Tiempo de simulación para el cálculo de la trayectoria (Sim_{time}^{ASR}):** el tiempo en segundos para mirar a lo largo del plan global con velocidad máxima, para calcular la ruta.
- **Frecuencia controlador ($Freq_{ctrl}^{ASR}$):** debe establecerse en el mismo valor que la frecuencia del planificador de la Sección 2.3.1 el parámetro F_{cntl}^{ASR}.
- **Habilitar la unión de los mapas de costos global y local ($Join_{map}^{ASR}$):** si es verdadero, se unen los mapas de costos locales y globales. Se evitarán los obstáculos locales.

3. Simulaciones preliminares

En esta sección vamos a ver en primer lugar el robot móvil y los sensores que se van a utilizar. A continuación, analizaremos el circuito diseñado para estas pruebas, explicando a su vez la división de los distintos objetivos que el robot va a tener que alcanzar con cada uno de los planificadores explicados en la Sección 2.3. Seguidamente, vamos a ver una serie de simulaciones preliminares, necesarias para realizar las pruebas de navegación autónoma para la extracción de datos.

3.1. Robot móvil

Una vez diseñado el circuito de pruebas que vamos a utilizar en el proyecto, vamos a hablar sobre el robot utilizado durante las pruebas del proyecto. Se trata del robot ROSbot 2.0, de la empresa Husarion. Husarion es una empresa especializada en robótica. Ofrece plataformas de robots autónomos para investigación y educación, y herramientas de software que permiten conectar sus robots a través de Internet. ROSbot es una plataforma de robot móvil autónomo con tracción 4x4 equipada con LIDAR, cámara RGB-D, IMU, codificadores, sensores de distancia. Este robot está disponible en dos versiones: 2.0 y 2.0 PRO. Toda la parte software que incluye el robot está desarrollada por ROS [27]. En las figuras siguientes, podemos ver el robot ROSbot que vamos a utilizar en el proyecto, tanto la parte frontal, mostrada en la Figura 14 como en la parte trasera, mostrada en la Figura 15

Figura 14: Robot ROSbot 2.0 por la parte frontal.

Figura 15: Robot ROSbot 2.0 por la parte trasera.

Siguiendo con la explicación del ROSbot, vamos a explicar las partes por las que se compone el robot. Este tiene una gran cantidad de sensores para poder realizar toda la parte de navegación y evasión de obstáculos. A continuación, vamos a listar los componentes que lleva ROSbot. En la Figura 16 tendremos una imagen donde podemos ver donde tiene el robot situado cada uno de los sensores que se explican a continuación [27].

- **Sensor infrarrojo:** el robot cuenta con 4 sensores de distancia de tecnología ToF con un alcance de hasta 2m. Concretamente el componente utilizado es el VL53L0X.
- **CORE2:** el microcontrolador que tenemos en este robot se trata de un STM32F407, en este circuito embebido es donde se realiza toda la programación del firmware del robot.
- **Motores DC:** los motores que lleva el robot se componen por 4 motores de continua Xinhe Motor XH-25D. Los motores usados son el modelo RF-370, con 6 VDC de tensión nominal, con 5000rpm de velocidad. Cuando no hay una carga en el eje de salida tenemos una velocidad de 165rpm, par de bloqueo de 2.9 $kg \cdot cm$ y con una corriente de bloqueo de 2.2A. La relación de transmisión del giro de las ruedas es de aproximadamente 34 (30613/900 exactamente). Los motores también vienen con unos encoders magnéticos los cuales nos ayudarán a medir la distancia que recorre el robot, estos sensores tienen 48ppr y 12 polos.
- **Sensor Inertial Measurement Unit (IMU):** ROSbot también lleva un sensor IMU el cual nos dará lecturas sobre aceleraciones, orientaciones y la dirección de las señales magnéticas. El componente utilizado en este robot es el MPU-9250. Esta IMU se compone tiene 9 ejes para realizar las medidas comentadas anteriormente. También hay ROSbots que en lugar de integrar el MPU-9250, integran el BNO055, este compoente es bastante similiar al anterior, se trata de un sensor de orientación absoluta inteligente de 9 ejes.
- **Cámara RGBD:** el robot también cuenta con una cámara integrada Orbbec Astra con tamaño de imagen RGB de 640x480 y tamaño de imagen de profundidad de 640x480.
- **Batería:** el robot además cuenta con 3 baterías de Li-Ion 18650 protegidas, baterías recargables, 3500 mAh de capacidad, 3.7 V de voltaje nominal.
- **Antena:** ROSbot tiene una antena conectada directamente al módulo Wi-Fi ASUS Tinker Board. Utiliza un cable RP-SMA (m) <-> I-PEX MHF4 para conectar la antena con SBC.

- **SBC:** el robot tiene también una CPU para el alto nivel del robot. Para ello utiliza una placa ASUS Tinker con 2 GB de RAM, Rockchip RK 3288 con 4x1.80 GHz como CPU y un ARM Mali-T764 MP2 como GPU y 32 GB MicroSD. El SBC se ejecuta en un sistema operativo basado en Ubuntu, personalizado para usar ROS.
- **LIDAR:** por último, el robot tiene un láser con el que será capaz de ver a que distancia se encuentran los objetos del entorno y así poder evitar obstáculos. El láser utilizado es el RpLidar A2, 360grados y un alcance de hasta 8m.

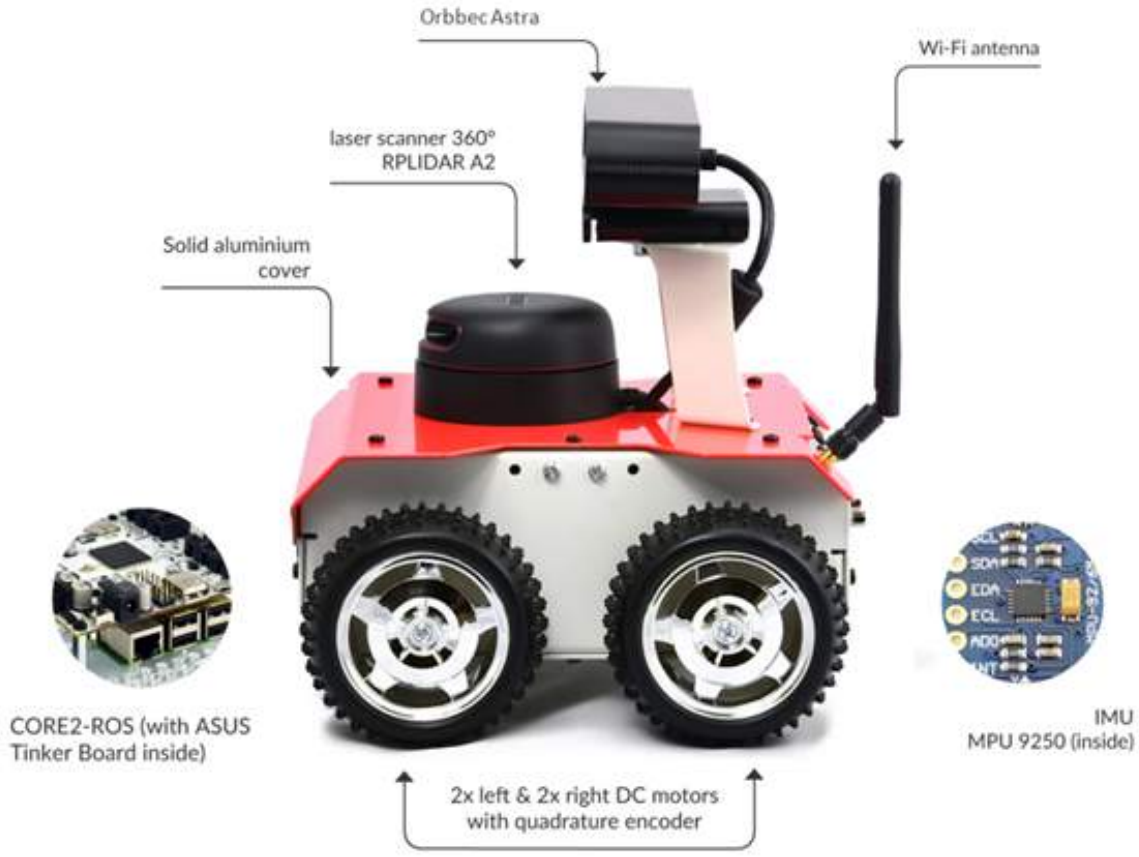


Figura 16: Sensores del robot ROSbot 2.0.

Una vez conocidos los componentes que tiene el robot que vamos a utilizar, debemos digitalizar el modelo para realizar las simulaciones necesarias y así poder realizar la comparativa del proyecto. Realizar la digitalización de un entorno gráfico es bastante costoso y puede resultar bastante complejo, además no es uno de los objetivos para este proyecto. Por lo que, en lugar de realizar toda la infraestructura de simulación, vamos a instalar una serie de paquetes donde la infraestructura ya está montada.

El paquete que hemos utilizado para el proyecto se llama rosbot_description. Es un repositorio de GitHub donde podemos encontrar un entorno preparado para la utilización del robot ROSbot simulado [28]. El enlace del repositorio es el siguiente:

`https://github.com/husarion/rosbot_description/tree/master`.

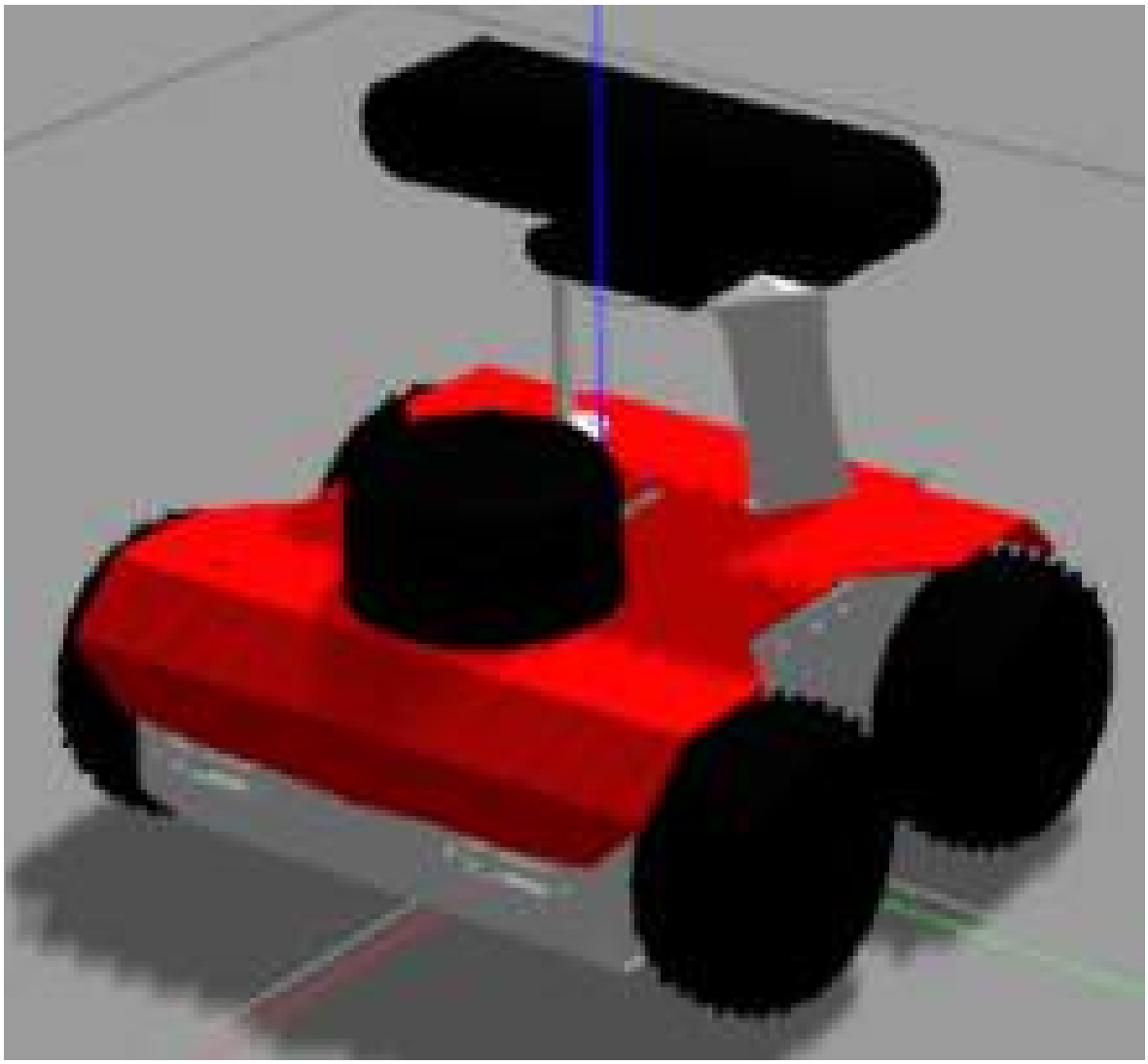


Figura 17: Modelo simulado en Gazebo del robot ROSbot 2.0.

En la Figura 17 podemos ver una imagen del robot ROSbot en el entorno gráfico Gazebo. Con este robot seremos capaces de enlazar este entorno gráfico de Gazebo con ROS para así establecer las comunicaciones necesarias entre ROS y Gazebo para la realización de las simulaciones necesarias.

Pero primero, para que funcione, el repositorio debemos instalar una serie de paquetes de los que rosbot_description depende. Estos paquetes son:

- navigation.
- map_server.
- slam_gmapping.
- tf2.
- eband_local_planner
- teb_local_planner.
- asr_ftc_local_planner.
- navegacion_asr.

3.2. Circuito de pruebas

En este apartado vamos a explicar el circuito de pruebas que hemos escogido para realizar la comparativa de los diferentes algoritmos. Para ello, hemos realizado una serie de pruebas a las cuales el robot deberá enfrentarse. En la Figura 18 vemos una imagen de Gazebo donde se puede ver el circuito al completo.

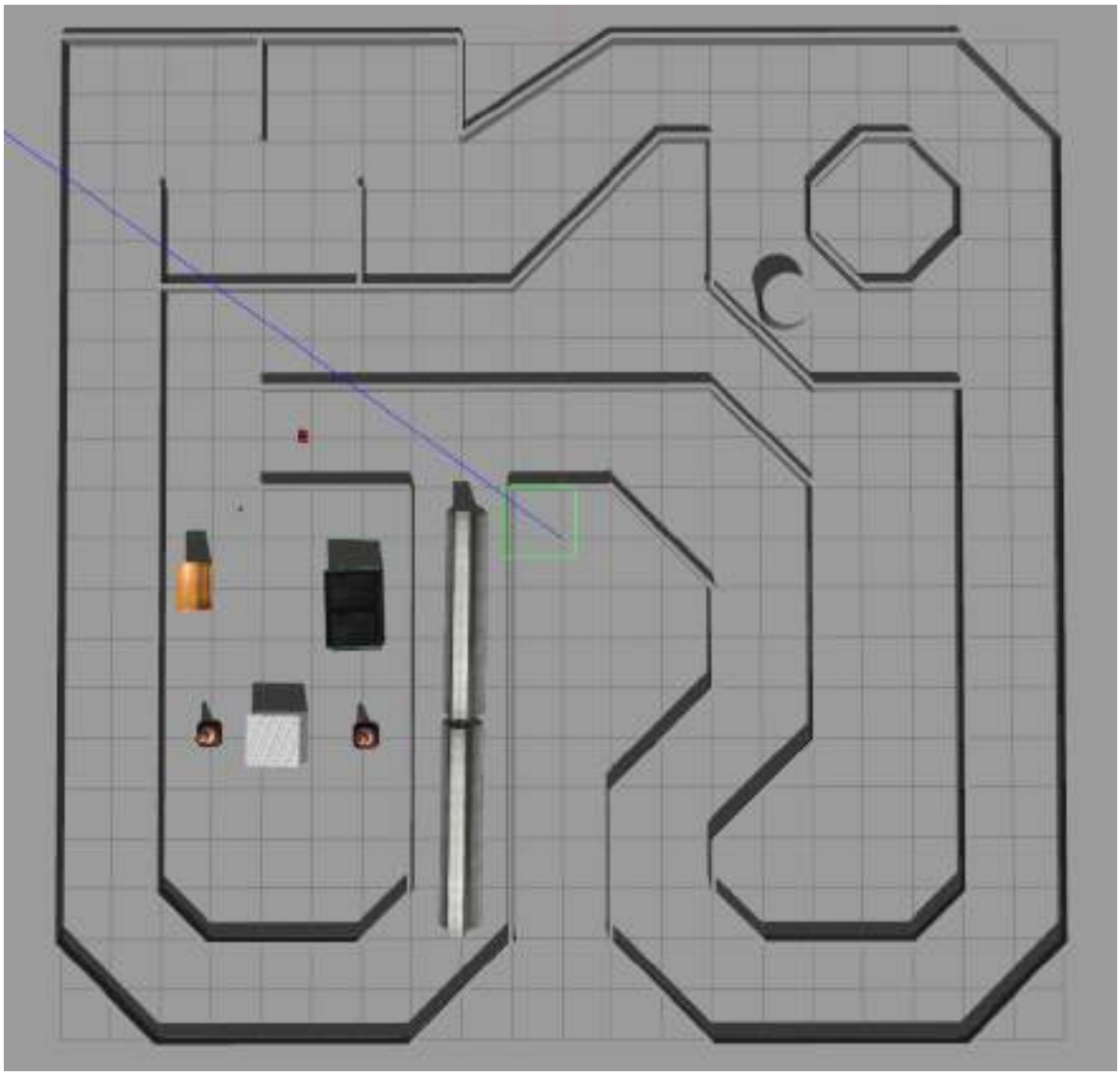


Figura 18: Circuito de pruebas completo.

A continuación, vamos a ver cada fragmento del circuito explicando el comportamiento que en teoría tendrá que realizar el robot en cada uno de los casos.

- ☞ En la Figura 19 se representa el primer tramo donde el robot recorrerá el circuito hasta que realice la curva en forma de U. En este tramo veremos cómo se comporta el robot atravesando pasillos estrechos, prueba explicada a continuación y haciendo curvas en forma de U.

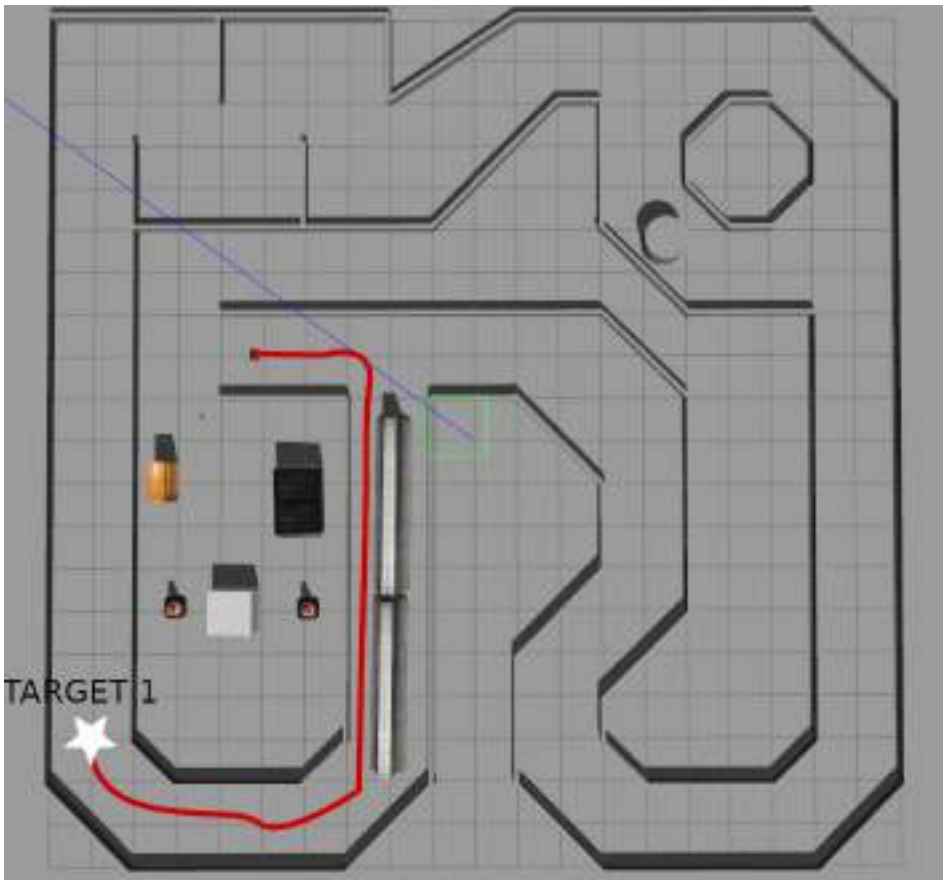


Figura 19: Trayectoria del robot para alcanzar el primer objetivo.

Pasillo: en la Figura 20 el robot deberá atravesar una zona con movilidad reducida. En esta zona el robot deberá calcular una trayectoria bastante estable, ya que si se desvía un poco se verá en una situación de colisión y se bloqueará. En esta zona veremos el comportamiento de los planificadores en situaciones extremas.

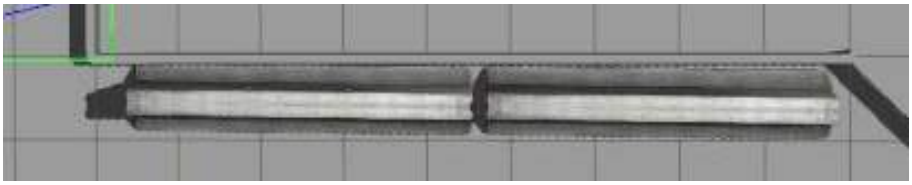


Figura 20: Prueba de movilidad reducida.

☞ En la Figura 21, el robot empezará el segundo tramo en el punto donde acabó el primer tramo y se extenderá hasta donde empieza la curva de arriba a la derecha. En este tramo, como se muestra en la Figura 21, el robot circulará por un pasillo amplio donde el podrá desplazarse a su máxima velocidad, luego realizará una maniobra de zig-zag, y posteriormente llegará al destino preparado para empezar el siguiente tramo.

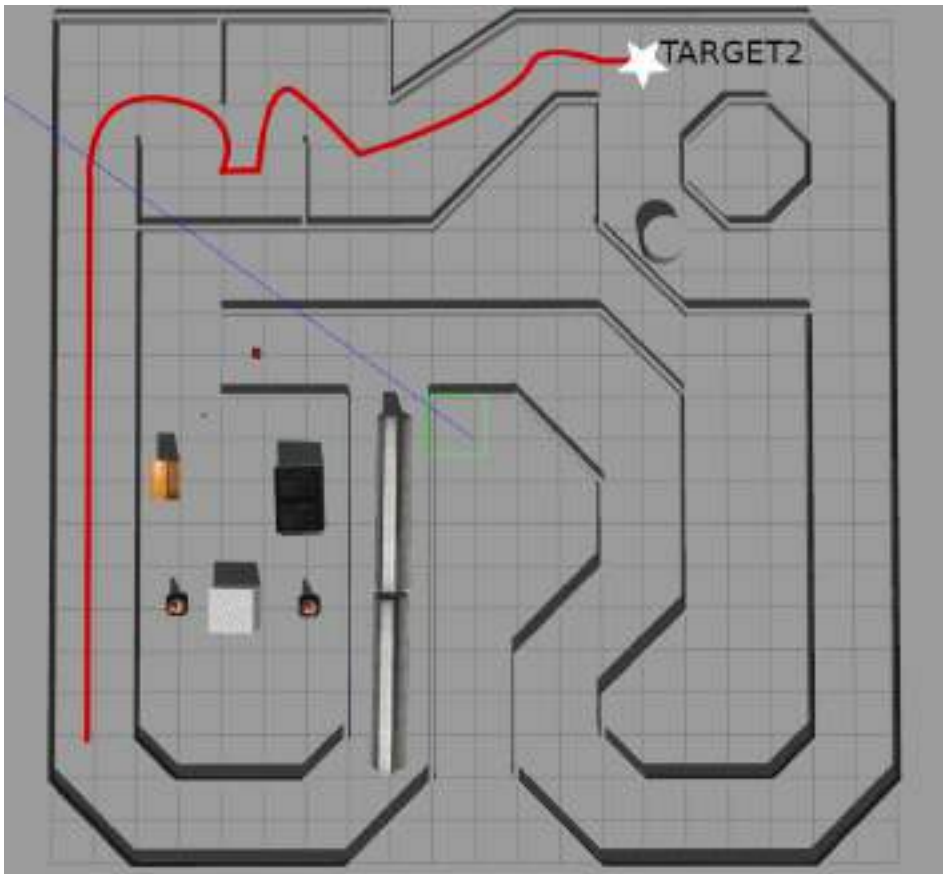


Figura 21: Trayectoria del robot para alcanzar el segundo objetivo.

Zig-zag: en la Figura 22 el robot circulará por una zona con obstáculos obligándolo a realizar una trayectoria en zig-zag. Esta prueba puede parecer muy similar a la prueba de la Figura 26, pero la diferencia entre estas dos pruebas es que esta prueba realiza un curva en S mucho más pronunciada y en un espacio mucho menor. Así los planificadores pueden gestionar en un solo cálculo una curva en forma en S. Esto les permite gestionar la curva de una forma más eficiente. Es por esto que en el zig-zag se realizan más de una curva en forma de S.

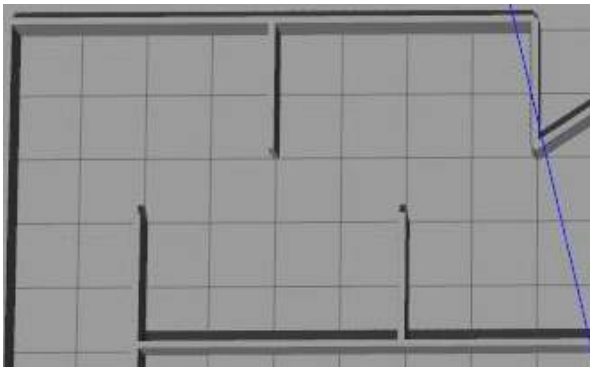


Figura 22: Prueba de zig-zag.

☞ En la Figura 23, alcanzaremos el tercer objetivo, donde el robot circulará por una zona donde tendrá que decidir la ruta que atravesar (ver Figura 24). En función de las prioridades de cada algoritmo veremos diferentes soluciones a un mismo problema. Luego circulará otra vez por un pasillo ancho y libre de obstáculos.

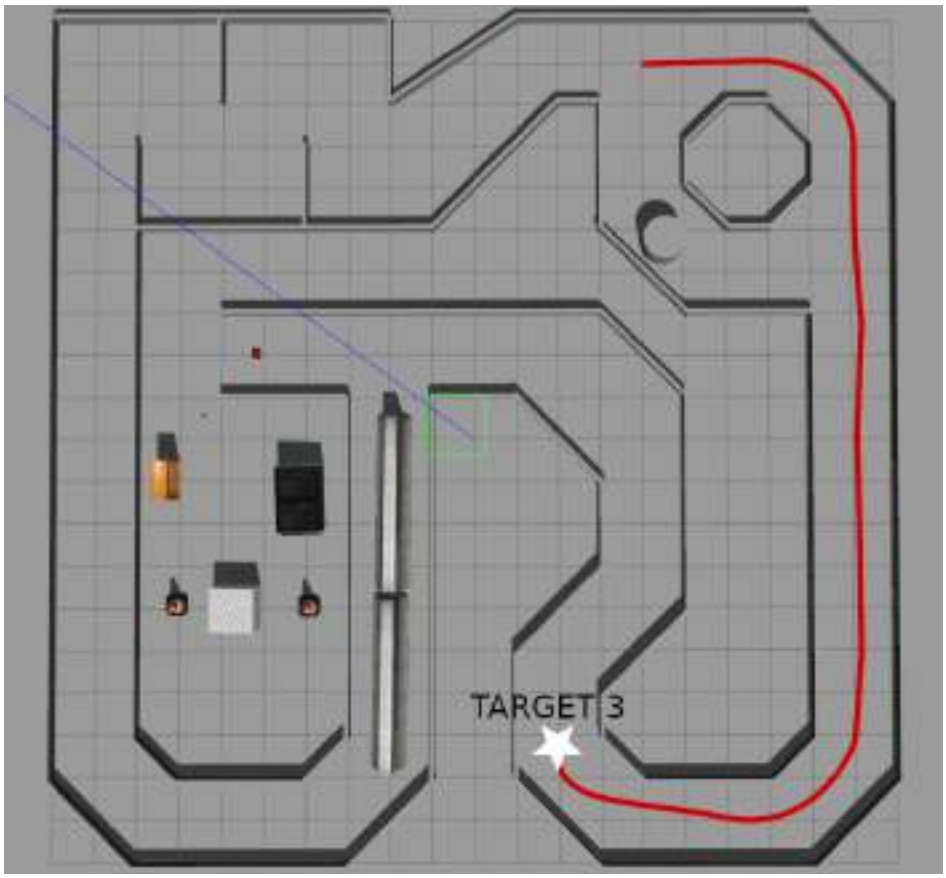


Figura 23: Trayectoria del robot para alcanzar el tercer objetivo.

Prueba de inteligencia: en la Figura 24 al robot se le dará una ruta donde tendrá dos opciones de cumplir el objetivo, una será más larga pero más ancha y por lo tanto más segura, y la otra será más corta pero atravesarla tiene una mayor dificultad. El robot deberá evaluar la situación y elegir la ruta más rápida para alcanzar el objetivo.

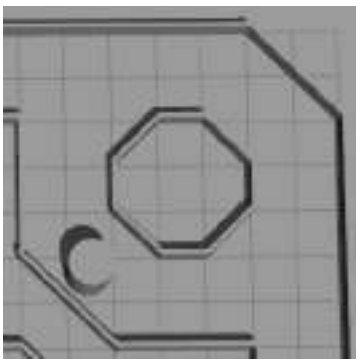


Figura 24: Prueba de inteligencia.

☞ En el cuarto tramo, el robot hará una curva en S llegando a la zona con obstáculos, tal y como muestra la Figura 25.

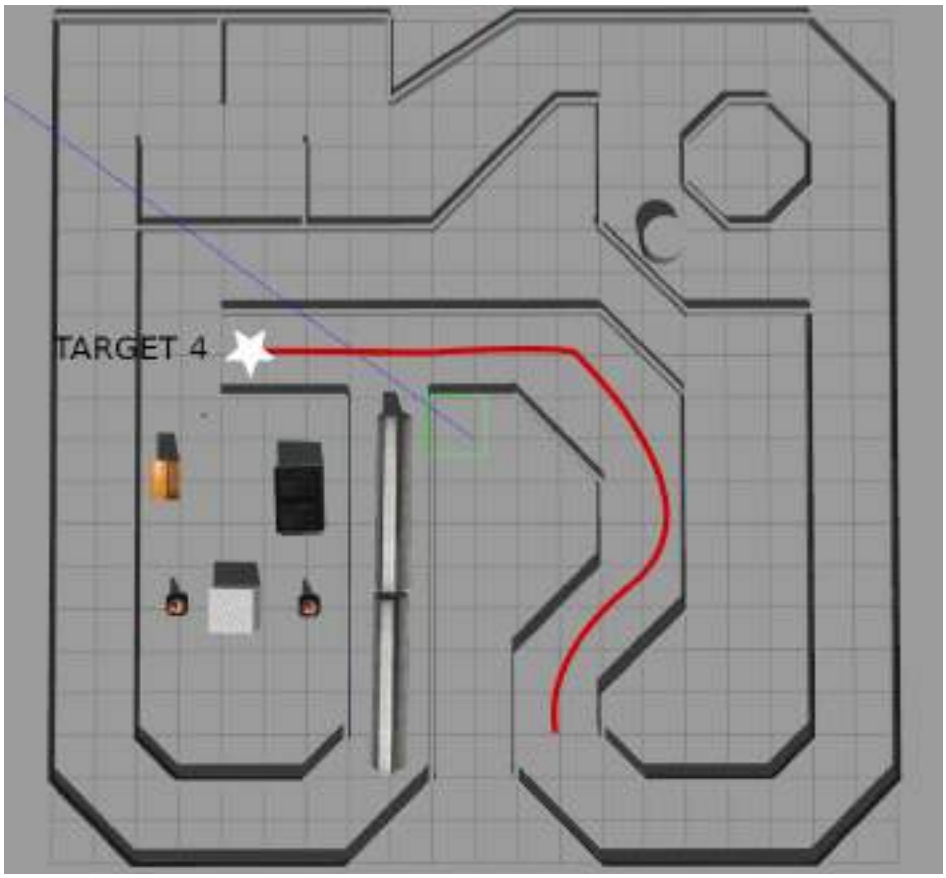


Figura 25: Trayectoria del robot para alcanzar el cuarto objetivo.

Camino en S: en la Figura 26 el robot circulará por una curva en forma de S, en esta prueba podremos ver como los algoritmos planifican las curvas, ya que tienen que hacer una curva a derechas y en un periodo corto de tiempo deben planificar una curva a izquierdas. Si la primera la planifica de forma que se abre demasiado, la segunda le resultará mucho más complicada.

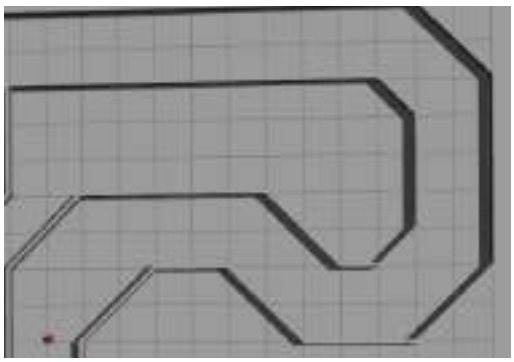


Figura 26: Prueba curva en forma de S.

☞ Finalmente, el úĺtimo tramo acabaría de cerrar el circuito, tal y como se muestra en la Figura 27, llegando después de la zona con obstáculos, ya que una de las pruebas a evaluar en este tramo será la de moverse por una zona amplia pero con obstáculos que el robot deberá evitar.

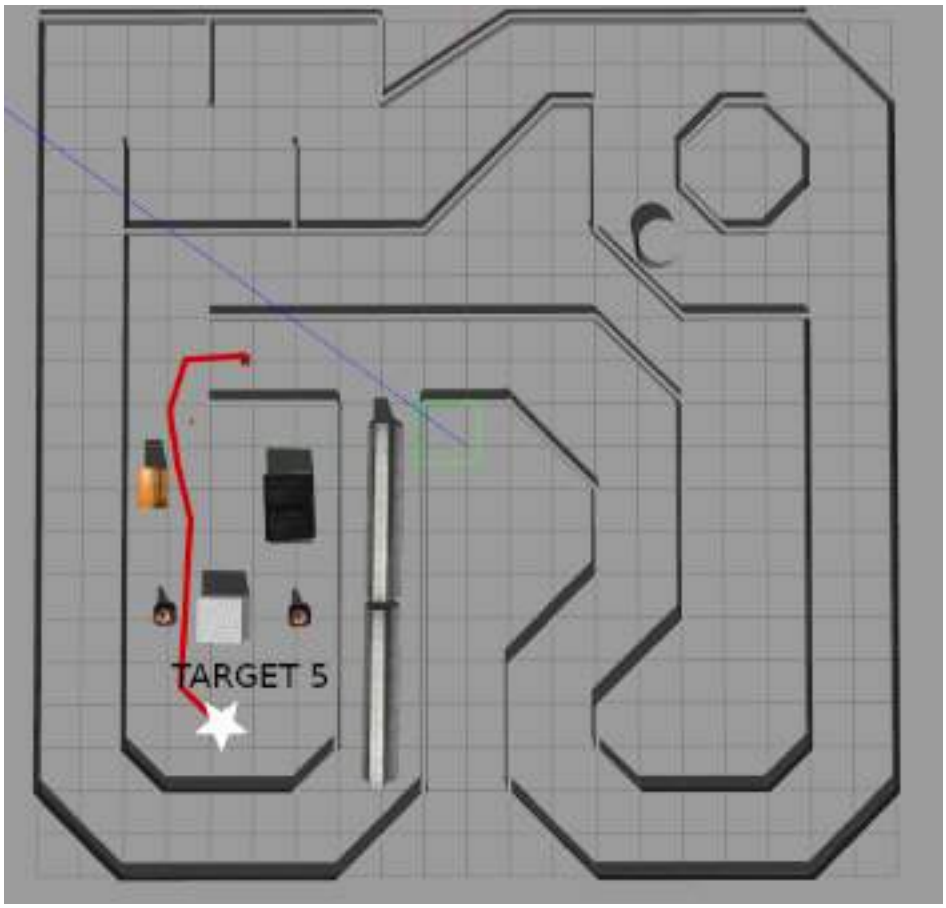


Figura 27: Trayectoria del robot para alcanzar el quinto objetivo.

Campo libre con objetos: en la Figura 28 el robot circulará por una zona abierta pero habrá obstáculos por el medio. Como podemos ver la zona llena de obstáculos tiene varios caminos a elegir para alcanzar el objetivo. Por lo que veremos las diferentes soluciones que planifican los algoritmos, ya que cada uno de ellos puede que calcule una forma distinta para alcanzar el objetivo.

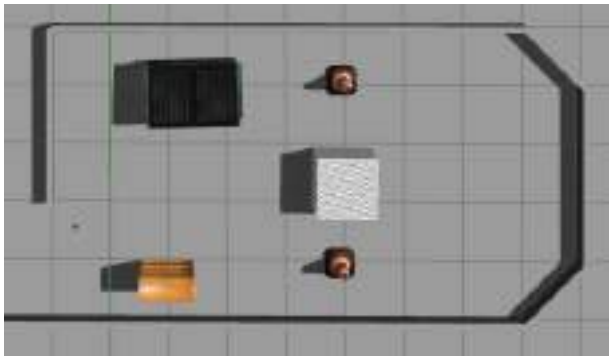


Figura 28: Prueba de zona con obstáculos.

3.3. Mapeado del entorno

Una vez analizados todos los planificadores y descrito el circuito de pruebas, podemos empezar a realizar algunas simulaciones iniciales que nos ayudarán a hacer la puesta en marcha de toda la simulación.

En primer lugar, una vez tenemos todos los paquetes descargados y compilados, podemos pasar a ejecutarlos haciendo un roslaunch, es un comando de ROS que lanza el fichero .launch que nosotros

elijamos. Estos ficheros son los que agrupan a todos los ficheros que necesitamos para crear la simulación, es decir, el fichero que contiene la información del mundo de Gazebo, el fichero que contiene la información del mapa global, los drivers de control del robot y de lose sensores, etc.

En el paquete rosbot_description, nos vienen todos los ficheros preparados para ejecutar, así que cuando nos descargamos el repositorio en nuestro espacio de trabajo, abrimos la terminal y ejecutamos el siguiente comando:

catkin_make

Con esto vamos a compilar todos los paquetes que tengamos en nuestro espacio de trabajo.

Una vez pasemos la compilación sin errores podemos parar a lanzar la simulación. Para ello pondremos los siguiente en la terminal:

roslaunch rosbot_description rosbot_rviz_gmapping.launch

Esto nos lanzará el entorno simulado de Gazebo, el programa de visualización Rviz, el máster de ROS y todos los nodos necesarios para realizar la correcta conducción del robot. Se parecerá a las Figuras 29 y 30.

Figura 29: Entorno cargado para el mapeo de la zona.

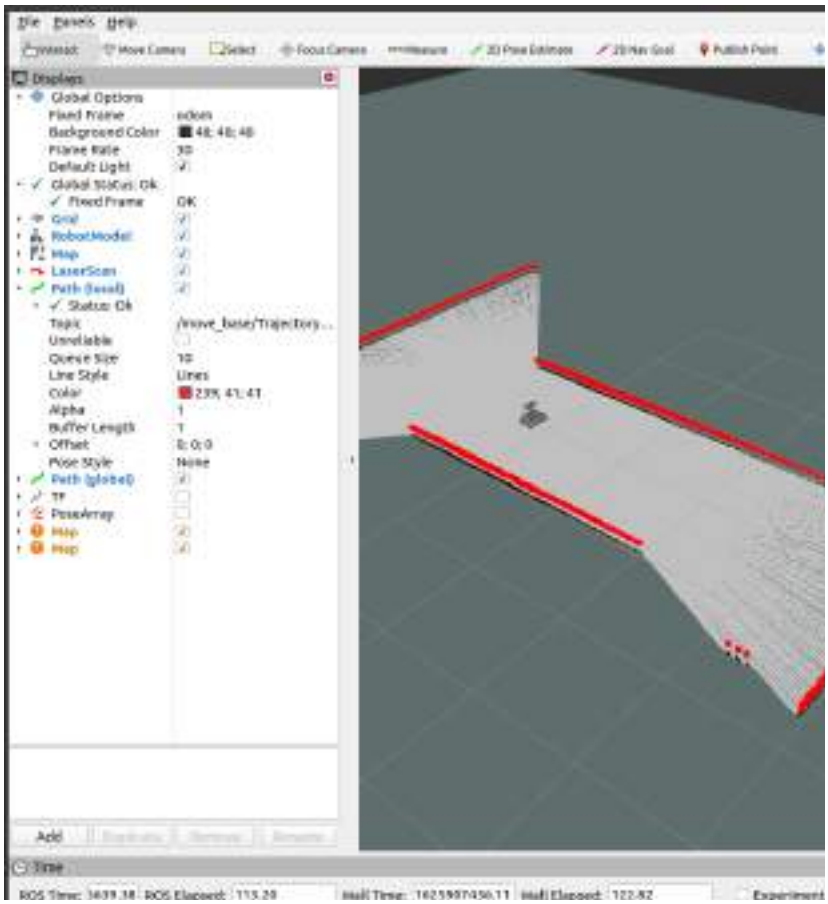


Figura 30: Visualización del mapeo de la zona.

Este comando nos va a permitir hacer un mapeado del circuito de pruebas, para que posteriormente el robot tenga un mapa global sobre el que guiarse cuando hagamos las pruebas. Pero para que el robot se pueda mover necesitamos otro comando que nos va a permitir controlarlo y así hacer el mapa.

Abrimos otro terminal sin cerrar el que teníamos abierto e introducimos el siguiente comando:

roslaunch rosbot_navigation rosbot_teleop.launch

En la terminal nos aparecerá la Figura 31. Vemos una serie de indicaciones para saber cómo funciona el programa y poder mover al robot por todo el mundo simulado.

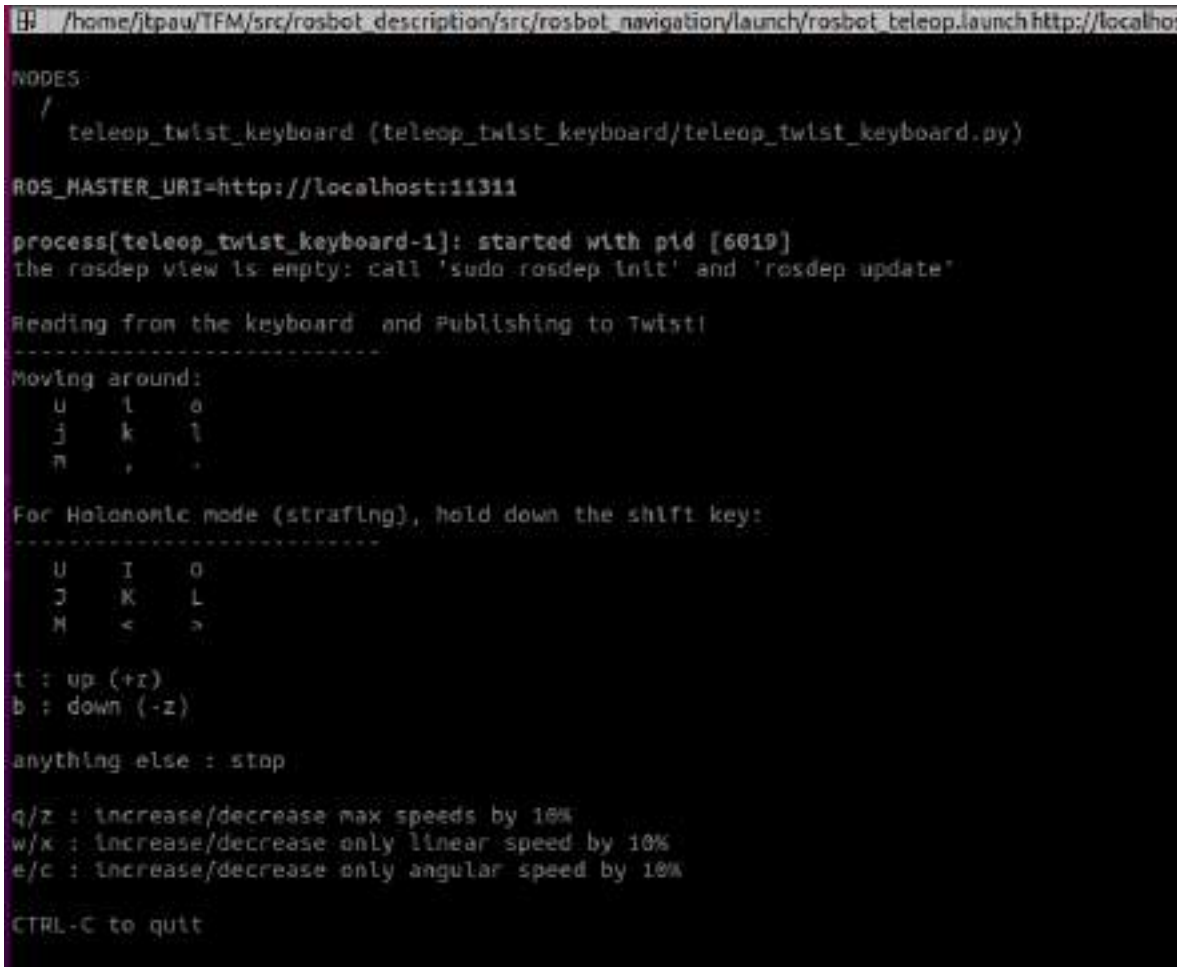


Figura 31: Terminal de comando para mover el robot.

Con esto vamos a ser capaces de mapear todo el circuito, y una vez lo tengamos todo mapeado y estemos satisfechos con el resultado, introducimos el siguiente comando en un tercer terminal para guardar el mapa creado:

rosrun map_server map_saver -f
/ros_workspace/src/rosbot_description/ src/rosbot_navigation/maps/test_map

El comando utilizado ahora es rosrun, bastante parecido a roslaunch, pero la diferencia es que rosrun únicamente lanza el nodo que nosotros le indicamos. La ruta que vemos en el comando corresponde a la ubicación donde guardaremos nuestro mapa, luego en ros_workspace pondremos nuestro espacio de trabajo, y donde pone test_map pondremos el nombre que le queramos dar al mapa. Cuando lancemos este comando se nos generará en esa ubicación un archivo .pgm, el cual utilizaremos para la navegación. Es importante recordar que no debemos cerrar ningún terminal cuando lancemos este último comando, ya que de lo contrario perderemos todo el circuito mapeado.

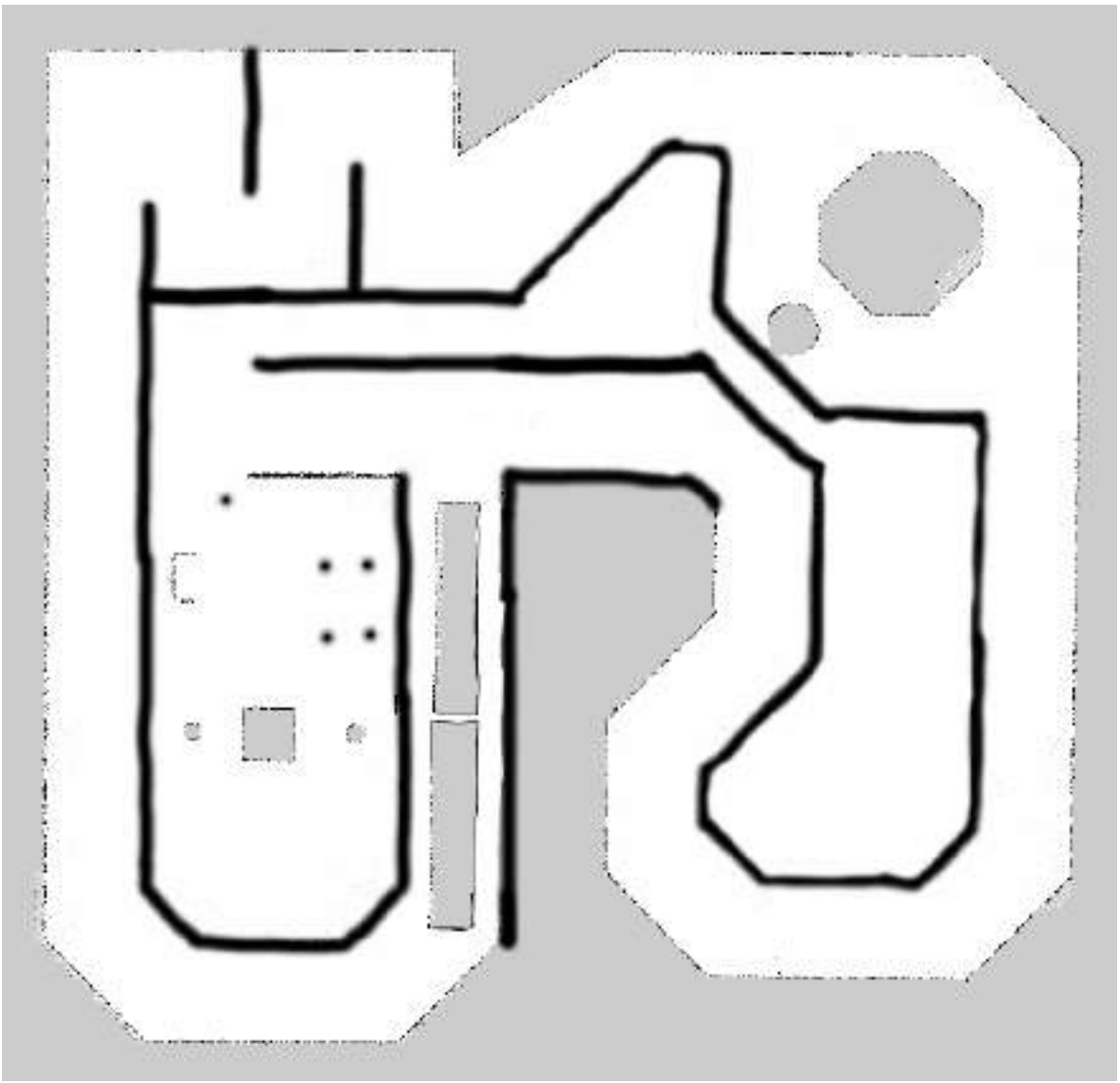


Figura 32: Mapeado del circuito de pruebas.

En la Figura 32, vemos el mapa que nos ha generado ROS una vez guardado. Este mapa es el que van a utilizar todos los planificadores locales para poder realizar cálculos de trayectoria y completar las pruebas. Con todo esto, tenemos las herramientas necesarias para empezar las simulaciones que realizaremos con el objetivo de extraer los datos y realizar el estudio comparativo de los diferentes planificadores locales.

4. Metodología

En esta sección del proyecto, en primer lugar vamos a realizar un estudio de los parámetros de los diferentes algoritmos vistos en la Sección 2.3, con ello determinaremos tres modos de funcionamiento (agresivo, normal y suave) para dichos algoritmos de planificación local. Posteriormente, en la Sección 5 se realizará una experimentación para ver cómo se comportan los algoritmos en diferentes situaciones. Después, en la Sección 5.1 se hará un estudio comparativo para ver las ventajas e inconvenientes de cada algoritmo en función de la situación en la que se encuentre el robot.

4.1. Estudio paramétrico de planificadores locales

En este apartado vamos a ver qué parámetros son los más significativos a la hora de actuar sobre el comportamiento del robot. Después del estudio realizado en el apartado anterior hemos podido ver que hay una gran cantidad de parámetros que son necesarios para un buen funcionamiento a la hora del cálculo de rutas, pero estos no afectan al comportamiento que estamos buscando, es decir, conducción agresiva, suave y normal. Luego, vamos a ver de cada planificador los parámetros que hacen que el robot se comporten de los modos descritos.

4.1.1. Parámetros TR

Parámetros	Variables	Conducción agresiva	Conducción normal	Conducción suave
a_x^T	acc_lim_x	3.0	2.5	2.0
α^T	acc_lim_theta	1.5	1.0	0.5
V_{max}^T	max_vel_x	0.7	0.5	0.3
ω_{max}^T	max_vel_theta	3.0	2.5	2.0
$Sim_{point_lin}^T$	sim_granularity	0.07	0.025	0.01
Sim_{time}^T	sim_time	2.3	1.7	0.7

Tabla 1: Parámetros para la navegación con TR.

En la Tabla 1, tenemos velocidades y aceleraciones máximas tanto lineales como angulares.

El quinto parámetro controla el trozo de trayectoria calculado en cada interacción. Si el valor es pequeño el planificador realizará cálculos de distancias más cortas por lo que el robot se comportará de una forma más segura.

Luego, seguimos con un parámetro cuyo valor corresponde al tiempo de simulación que el robot va a calcular en cada iteración. Por tanto, si el valor es muy elevado, el algoritmo calculará más trozo de la ruta con lo cual, el robot circulará más tiempo con velocidad máxima y el comportamiento que presente será más agresivo, este es el Sim_{time}^T.

4.1.2. Parámetros DWA

Parámetros	Variable	Conducción agresiva	Conducción normal	Conducción suave
a_x^{DWA}	acc_lim_x	3.0	2.5	2.0
α^{DWA}	acc_lim_theta	4.0	3.2	2.8
vt_{max}^{DWA}	max_vel_trans	0.7	0.55	0.22
vt_{xmax}^{DWA}	max_vel_x	0.7	0.55	0.22
ω_{max}^{DWA}	max_rot_vel	1.5	1.0	0.5
Sim_{point}^{DWA}	sim_granularity	0.07	0.025	0.01
Sim_{time}^{DWA}	sim_time	2.3	1.7	0.7
$Stop_{time}^{DWA}$	stop_time_buffer	0.1	0.2	0.4

Tabla 2: Parámetros para la navegación con DWA.

En la Tabla 2, al igual que estamos viendo en todos los planificadores, tenemos velocidades y aceleraciones máximas tanto lineales como angulares.

El sexto parámetro controla el trozo de trayectoria calculado en cada interacción. Si el valor es pequeño el planificador realizará cálculos de distancias más cortas por lo que el robot se comportará de una forma más segura.

Luego, seguimos con el parámetro Sim_{time}^{DWA} que ya hemos visto en la Sección 4.1.1, el cual es bastante común en estos algoritmos para realizar el control, por lo que lo veremos en algunos algoritmos.

Por último, hemos decidido escoger este parámetro, que decide el tiempo en el que el robot debe detenerse antes de una colisión. Esto nos parece interesante ya que si el tiempo es muy pequeño debe realizar grandes cambios de velocidades y aceleraciones, lo que va a suponer un comportamiento agresivo, además de que con este parámetro ponemos a prueba los reflejos del robot.

4.1.3. Parámetros EBAND

Parámetros	Variables	Conducción agresiva	Conducción normal	Conducción suave
F_{min}^{EBAND}	eband_significant_force_lower_bound	0.2	0.15	0.05
$W_{costmap}^{EBAND}$	costmap_weight	15	20	50
$Vmax_{lin}^{EBAND}$	max_vel_lin	1.2	1	0.7
ω_{max}^{EBAND}	max_vel_th	1.6	1.5	1.2
V_{min}^{EBAND}	min_vel_lin	0.2	0.2	0.05
ω_{min}^{EBAND}	min_vel_th	0.5	0.5	0.2
a_{max}^{EBAND}	max_acceleration	0.5	0.4	0.25
a_{max}^{EBAND}	max_traslational_acceleration	0.4	0.4	0.25
α_{max}^{EBAND}	max_rotational_acceleration	1.3	1	0.8
$Ctrl_{rate}^{EBAND}$	ctrl_rate	50	75	100
$Mult_{vel}^{EBAND}$	bubble_velocity_multiplier	2.0	1.0	0.2

Tabla 3: Parámetros para la navegación con EBAND.

En la Tabla 3 vamos a ver los parámetros del planificador EBAND. En primer lugar, hay un parámetro que controla la magnitud mínima de la fuerza que consideramos significativa (F_{min}^{EBAND}), es decir, cuánta fuerza debe ejercer el robot para que el algoritmo la tenga en cuenta. El valor de este parámetro va a aumentar para que su comportamiento sea más agresivo, así le indicaremos al robot que debe hacer fuerzas mayores para el movimiento.

$W_{costmap}^{EBAND}$ es un peso que nos indica la importancia del mapa de costos en el cálculo de la trayectoria que calcula el algoritmo.

Luego pasamos a las velocidades y aceleraciones del algoritmo, que, al igual que en los demás algoritmos, tenemos velocidades y aceleraciones máximas tanto angulares como lineales. Aunque en este caso hemos añadido también las velocidades mínimas, ya que para este algoritmo resulta muy interesante tener el control sobre estos parámetros.

Ahora pasamos a otro de los parámetros escogidos para realizar el control del EBAND, $Ctrl_{rate}^{EBAND}$. Este valor controla la tasa del control, es decir, la trayectoria por la que el robot circulará. Cuando este valor sea pequeño el robot circulará por espacios más amplios aunque esto suponga realizar rutas más largas. Del mismo modo, si el valor es elevado, el planificador calculará rutas lo más cortas posibles a la meta.

Por último, el parámetro $Mult_{vel}^{EBAND}$ que altera el comportamiento del robot variando su valor teniendo un comportamiento agresivo cuando el valor del parámetro ronda por el 2, como vemos en la Tabla 3.

4.1.4. Parámetros TEB

Parámetros	Variable	Conducción agresiva	Conducción normal	Conducción suave
v_x^{TEB}	max_vel_x	0.8	0.4	0.3
v_{xbw}^{TEB}	max_vel_x_backwards	0.3	0.2	0.1
ω^{TEB}	max_vel_theta	0.4	0.3	0.2
a_x^{TEB}	acc_lim_x	0.6	0.5	0.4
α^{TEB}	acc_lim_theta	0.6	0.5	0.4
vel_{free}^{TEB}	free_goal	ON	OFF	OFF
sep_{min}^{TEB}	min_obstacle_dist	0.24	0.25	0.26
$inflation_{dist}^{TEB}$	inflation_dist	0.55	0.6	0.65
$W_{obstacle}^{TEB}$	weight_obstacle	99	100	110

Tabla 4: Parámetros para la navegación con TEB

En la Tabla 4, podemos ver los parámetros escogidos para el planificador local TEB. Este planificador es una modificación del planificador EBAND, es decir, los dos algoritmos se basan en el concepto de las bandas elásticas pero TEB añade la temporalidad al algoritmo. Es por esto que podemos observar que algunos parámetros son parecidos o simplemente son iguales.

Los primeros corresponden a velocidades máximas a las que el robot puede llegar, tanto lineales como angulares. Posteriormente vemos un par de valores que corresponden a las aceleraciones máximas que alcanzará nuestro robot y del mismo modo que con las velocidades, tenemos aceleraciones lineales y angulares.

Una vez vistos estos parámetros, pasamos a analizar los siguientes, los cuales son únicos de cada planificador y, por tanto, vamos a explicarlos con un poco más de detenimiento.

El booleano vel_{free}^{TEB} es un valor que nos indicará si el robot elimina la restricción de la velocidad a la meta, por tanto cuando esté a verdadero, vamos a ver como el robot alcanzará el objetivo con la velocidad máxima, por ello lo vamos a poner en verdadero cuando queramos que éste tenga un comportamiento agresivo.

El siguiente de los parámetros va a interpretar la distancia mínima a la que el robot evitará los obstáculos, es decir, si este valor es muy elevado cuando el robot detecte un obstáculo, éste generará una zona alrededor del obstáculo con un radio igual al valor elegido en este parámetro. Por eso, cuando queramos un comportamiento agresivo este valor será muy bajo, así el planificador calculará rutas acercándose más a los objetos y aumentando así el riesgo de colisión.

El siguiente valor está relacionado con el anterior, ya que este valor crea penalizaciones en la zona de amortiguación que se ha creado con el otro parámetro. Luego, cuando mayor sea este valor más peso tendrá el valor del otro parámetro (sep_{min}^{TEB}). Por eso, cuando el comportamiento sea muy cauteloso el valor de este parámetro será más elevado, y así tendremos más en cuenta el valor anterior y el robot priorizará alejarse de los objetos.

Por último, vamos a ver un parámetro que enlaza también con los dos anteriores, ya que este parámetro de optimización lo usa el algoritmo para que las rutas escogidas sean lo más óptimas posibles. En este caso, el valor se utiliza para la optimización para mantener la distancia mínima entre obstáculos. Por eso, el valor de este parámetro aumenta cuanto más cauteloso queremos que sea el robot.

4.1.5. Parámetros ASR

Parámetros	Variables	Conducción agresiva	Conducción normal	Conducción suave
v_x^{ASR}	max_x_vel	0.8	0.5	0.3
ω_{max}^{ASR}	max_rotation_vel	1.5	1	0.5
a_x^{ASR}	acceleration_x	1	0.5	0.4
a_z^{ASR}	acceleration_z	1	0.5	0.4
Sim_{time}^{ASR}	sim_time	1.5	1	0.5

Tabla 5: Parámetros para la navegación con ASR

Por último, vamos a ver el algoritmo ASR. Este algoritmo es el que menos parámetros tiene para configurarlo, lo que hace que la Tabla 5 sea mucho más corta que el resto.

En este caso tenemos los parámetros de velocidades y aceleraciones máximas que nuestro robot va a alcanzar durante la realización de la ruta. Estas velocidades y aceleraciones son tanto lineales como angulares.

Por último, vamos a modificar el parámetro Sim_{time}^{ASR} un valor que corresponde al tiempo de simulación que el robot va a calcular en cada iteración. Por tanto, si el valor es muy elevado, el algoritmo calculará más trozo de la ruta de modo que el robot circulará más tiempo con velocidad máxima y el comportamiento que presente será más agresivo.

4.2. Métricas

Una vez conocidos los diferentes parámetros que vamos a utilizar para la definición del comportamiento del robot, así como el diseño y la puesta en marcha del simulador, nos falta definir una serie de métricas las cuales nos servirán de guía en la comparativa posterior. En este apartado definiremos las variables y valores que nos parecen interesantes para la realización de la comparativa. Además, se va a estudiar el cálculo de las diferentes métricas obtenidas. De este modo será más fácil de entender cómo se obtienen los diferentes valores y ayudará a la comprensión de los resultados que veremos en la Sección 5.1.

En primer lugar vamos a definir las variables que vamos a evaluar en las métricas:

- Velocidad lineal.
- Velocidad angular.
- Aceleración lineal.
- Tiempos del recorrido.
- Distancia a objetos.
- Precisión para alcanzar el destino.

Una vez conocidas las variables a evaluar podemos pasar a explicar las métricas con las que extraeremos conclusiones:

- **Tiempo total t_{end}:** Tiempo invertido por el robot en completar una vuelta al circuito de pruebas entero.

$$t_{end} = \sum_{n=1}^{5} t_{target}(n) \tag{3}$$

- **Tiempo de cada objetivo t_{target}:** Tiempo invertido por el robot en completar cada uno de los objetivos definidos en la Sección 3.2.

- **Tiempo medio por objetivo $\overline{t_{target}}$:** Valor medio de los tiempos registrados por el robot en cada uno de los puntos.

$$\overline{t_{target}} = \frac{\sum_{n=1}^{5} t_{target}(n)}{5} \tag{4}$$

- **Precisión en posición e_p:** Distancia entre la posición del robot y la de la configuración destino ($q_T = [x_T\ y_T\ \theta_T]^T$), calculada como el porcentaje del error cuadrático entre las coordenadas x e y de ambos.

$$e_p(\,\%) = \sqrt{(x_T - x_R(t_{target})^2 + (y_T - y_R(t_{target}))^2} \cdot 100 \tag{5}$$

- **Precisión media por objetivo $\overline{e_p}$**: Valor medio del error cuadrático calculado para cada objetivo.

$$\overline{e_p}(\,\%) = \frac{\sum_{n=1}^{5} e_p(n)}{5} \tag{6}$$

- **Velocidad lineal media en cada objetivo $\bar{v}$:** El valor medio de la velocidad durante el recorrido de cada objetivo.

$$\bar{v} = \frac{\sum_{n=1}^{v_f} v(n)}{v_f} \tag{7}$$

- **Velocidad lineal media de los valores medios $\bar{\bar{v}}$:** El valor medio de las velocidades medias registradas en cada objetivo.

$$\bar{\bar{v}} = \frac{\sum_{n=1}^{5} \bar{v}(n)}{5} \tag{8}$$

- **Velocidad lineal mínima en cada objetivo v_{min}:** El valor de la velocidad lineal mínimo registrado por el robot en cada objetivo.

- **Velocidad lineal media de los valores mínimos $\overline{v_{min}}$:** El valor medio de los valores mínimos registrados en cada objetivo.

$$\overline{v_{min}} = \frac{\sum_{n=1}^{5} v_{min}(n)}{5} \tag{9}$$

- **Velocidad lineal máxima en cada objetivo v_{max}:** Es la máxima velocidad lineal alcanzada por el robot en cada uno de los objetivos.

- **Velocidad lineal media de los valores máximos $\overline{v_{max}}$:** Es el valor medio de los valores máximos registrados en cada uno de los objetivos.

$$\overline{v_{max}} = \frac{\sum_{n=1}^{5} v_{max}(n)}{5} \tag{10}$$

- **Velocidad angular media en cada objetivo $\bar{\omega}$:** El valor medio de la velocidad angular durante el recorrido de cada objetivo.

$$\bar{\omega} = \frac{\sum_{n=1}^{\omega_f} \omega(n)}{\omega_f} \tag{11}$$

- **Velocidad angular media de los valores medios $\bar{\bar{\omega}}$:** El valor medio de las velocidades medias registradas en cada objetivo.

$$\bar{\bar{\omega}} = \frac{\sum_{n=1}^{5} \bar{\omega}(n)}{5} \tag{12}$$

- **Velocidad angular mínima en cada objetivo ω_{min}:** El valor mínimo de la velocidad angular registrado por el robot en cada objetivo.

- **Velocidad angular media de los valores mínimos $\overline{\omega_{min}}$:** El valor medio de los valores mínimos registrados en cada objetivo.

$$\overline{\omega_{min}} = \frac{\sum_{n=1}^{5} \omega_{min}(n)}{5} \tag{13}$$

- **Velocidad angular máxima en cada objetivo ω_{max}:** Es la máxima velocidad angular alcanzada por el robot en cada uno de los objetivos.

- **Velocidad angular media de los valores máximos $\overline{\omega_{max}}$:** Es el valor medio de los valores máximos registrados en cada uno de los objetivos.

$$\overline{\omega_{max}} = \frac{\sum_{n=1}^{5} \omega_{max}(n)}{5} \tag{14}$$

- **Aceleración lineal media en cada objetivo $\bar{a}$:** El valor medio de la aceleración durante el recorrido de cada objetivo.

$$\bar{a} = \frac{\sum_{n=1}^{a_f} a(n)}{a_f} \tag{15}$$

- **Aceleración lineal media de los valores medios $\bar{\bar{a}}$:** El valor medio de las aceleraciones medias registradas en cada objetivo.

$$\bar{\bar{a}} = \frac{\sum_{n=1}^{5} \bar{a}(n)}{5} \tag{16}$$

- **Aceleración lineal mínima en cada objetivo a_{min}:** El valor de la aceleración lineal mínimo registrado por el robot en cada objetivo.
- **Aceleración lineal media de los valores mínimos $\overline{a_{min}}$:** El valor medio de los valores mínimos registrados en cada objetivo.

$$\overline{a_{min}} = \frac{\sum_{n=1}^{5} a_{min}(n)}{5} \tag{17}$$

- **Aceleración lineal máxima en cada objetivo a_{max}:** Es la máxima aceleración lineal alcanzada por el robot en cada uno de los objetivos.
- **Aceleración lineal media de los valores máximos $\overline{a_{max}}$:** Es el valor medio de los valores máximos registrados en cada uno de los objetivos.

$$\overline{a_{max}} = \frac{\sum_{n=1}^{5} a_{max}(n)}{5} \tag{18}$$

- **Distancia media a objetos en cada objetivo $\bar{d}$:** Es la distancia media detectada por el Lidar a lo largo de cada objetivo.
- **Distancia media de los valores medios $\bar{\bar{d}}$:** Es el valor medio de los valores medios registrados en cada uno de los objetivos.

$$\bar{\bar{d}} = \frac{\sum_{n=1}^{d_f} \bar{d}(n)}{d_f} \tag{19}$$

- **Distancia mínima a objetos en cada objetivo d_{min}:** Es la distancia mínima detectada por el Lidar a lo largo de cada objetivo.
- **Distancia media de los valores mínimos $\overline{d_{min}}$:** Es el valor medio de los valores mínimos registrados en cada uno de los objetivos.

$$\overline{d_{min}} = \frac{\sum_{n=1}^{5} d_{min}(n)}{5} \tag{20}$$

- **Orientación objetos Θ:** Es el valor de la orientación en la que el Lidar detectó el acercamiento al objeto correspondiente. Esta orientación es conocida para cada una de las métricas explicadas anteriormente. Para ello, extraeremos la información del topic /scan [6].

En la Figura 33 podemos ver una de las gráficas que hemos extraído puesta a modo de ejemplo para comprender la condensación que hemos realizado con los datos:

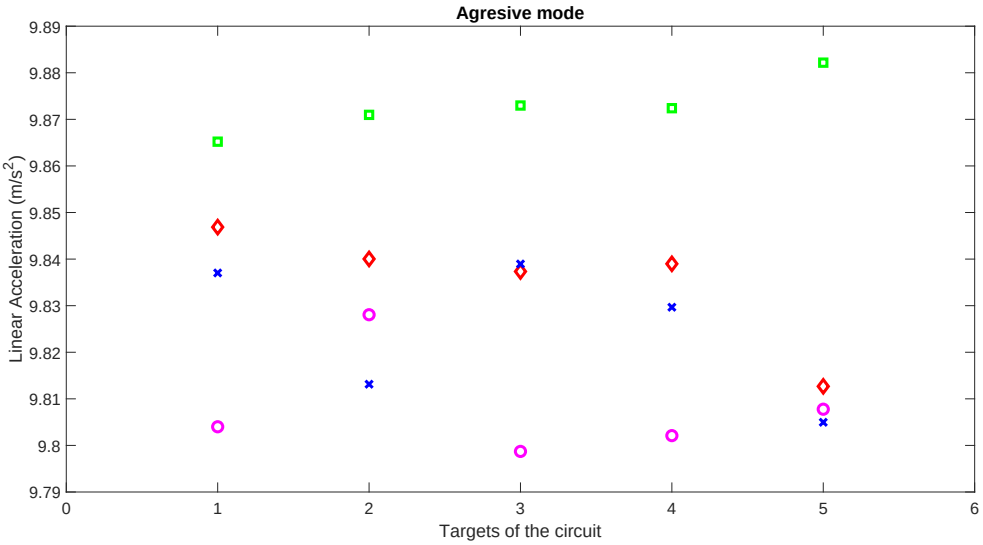


Figura 33: Ejemplo de las gráficas extraídas.

4.3. Simulaciones

En este apartado vamos a ver cómo hemos realizado las simulaciones, las cuales hemos guardado para posteriormente poder extraer las variables y las métricas que explicamos en la Sección 4.2.

En primer lugar, vamos a ver cómo realizar las simulaciones. En la Sección 3 vimos las simulaciones preliminares que eran necesarias para hacer la puesta en marcha. Aunque para las simulaciones finales hemos generado una serie de ficheros que nos van a facilitar el cambio entre algoritmos.

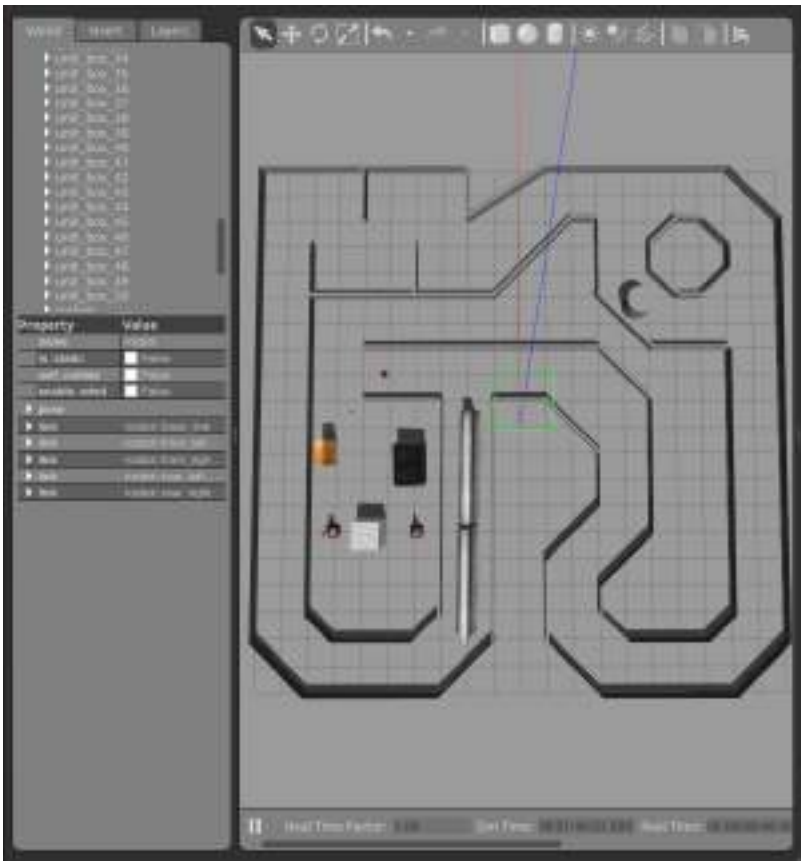
Hemos creado unos ficheros .launch de cada planificador donde podemos encontrar todos los ejecutables necesarios para realizar la simulación. En el .launch específico de cada planificador se lanza el mundo de Gazebo, se lanza el máster de ROS, se lanza el planificador global y el planificador local correspondiente con los parámetros del modo en el que queramos evaluar, y además, se lanza el entorno de Rviz donde podemos ver los cálculos que realiza el robot a tiempo real. Tal y como muestra la Figura 34. Podemos encontrar el código en el Anexo A.1.

Para los parámetros de cada modo, como ya se ha comentado, se lanzan en el .launch pero el fichero que lanzamos en el .launch es un .yaml donde se colocan los parámetros para la configuración de diferentes comportamientos a evaluar. Dentro del fichero < planificador > .launch, podremos modificar el modo de comportamiento cambiando por el fichero .yaml correspondiente en la siguiente linea de código. Podemos encontrar el código entero en el Anexo A.2.

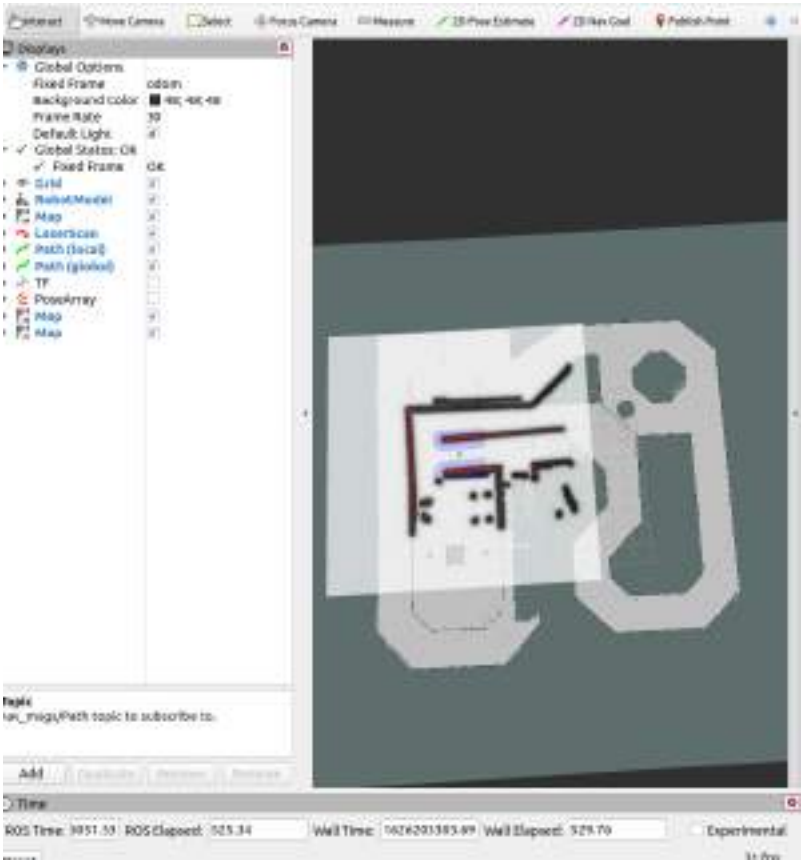
< rosparam file = "$(find rosbot_navigation)/config/../params.yaml"
command= "load" />

Por tanto, si en una terminal lanzamos el siguiente comando:

roslaunch rosbot_description <planificador> .launch



(a) Entorno de simulación Gazebo.



(b) Entorno de visualización RViz.

Figura 34: Punto inicial de la simulación.

El circuito de la Figura 34 se ha dividido en 5 puntos, como se comentó en la Sección 3.2, por lo que en las simulaciones haremos que el robot alcance 5 metas diferentes por las que atravesará todo el circuito y así realizará todas las pruebas. Pero no sería justo lanzar los puntos a ojo con la herramienta del Rviz, ya que el punto final cambiaría y las rutas calculadas podrían variar siendo algunas más difíciles que otras debido a este error de precisión. Por ello, para lanzar los objetivos hemos realizado un fichero .bash en el que escribimos el punto del objetivo al que debe de ir el robot. El código lo podemos encontrar en el Anexo A.3

Para realizar una simulación debemos abrir 3 terminales, en uno lanzaremos toda la simulación y el entorno con el comando ya visto, en una segunda terminal empezaremos a grabar la simulación (ver Sección 4.4) realizada y en tercer lugar, en otra terminal diferente, lanzaremos el fichero .bash con el siguiente comando:

Trayectorias_circuito.bash "tramo donde ir"

4.4. Adquisición de datos

En este punto, vamos a ir en paralelo con el punto de las simulaciones, puesto que, los datos que necesitamos para la comparativa son los mismos datos que da en tiempo real la simulación del robot. Es por eso que la adquisición de los datos se debe de hacer mientras se realiza la simulación. Para ello, vamos a utilizar un comando de ROS que será clave para este trabajo, se trata de rosbag [5]. Este comando, como su nombre indica, crea una bolsa donde se guarda toda la información que se ha publicado y los nodos suscritos que han pasado en la simulación. Por lo que podemos conocer: tiempos, velocidades, aceleraciones, información de sensores... Con toda esta información es con la que vamos a realizar la comparativa. Por tanto, una vez tengamos el robot en el punto inicial, vamos a lanzar el siguiente comando:

rosbag record -a

Este comando empezará a grabar toda la simulación y el comando -a se utiliza para indicar que se debe almacenar la información de todos los nodos que estén en ejecución. Lógicamente, esto supone que las bolsas creadas sean muy pesadas ya que hay mucha información. Esta es la razón por la que el circuito está dividido en 5 fragmentos, para que el posterior procesado de datos sea mucho más ágil. Si los archivos fueran demasiado grandes, ROS nos ofrece comandos con los que comprimir y descomprimir estas bolsas para que sean más fáciles de transportar.

rosbag compress
rosbag decompress

Del mismo modo, existen muchas otras indicaciones para el rosbag. Podemos volver a reproducir la simulación guardada en la bolsa con exactitud. Podemos obtener información sobre la bolsa, chequear que la bolsa está bien creada, y otras funcionalidades [5].

Una vez finalizadas las grabaciones, seguiremos con la adquisición de datos, pero esta vez los datos a adquirir ya serán datos útiles para la realización de la comparativa del proyecto.

Para la extracción de datos vamos a utilizar Matlab. Matlab es una herramienta muy potente utilizada para la realización de cálculos complejos, modelos de control, entrenamiento y realización de modelos de inteligencia artificial, procesado de datos... En nuestro caso vamos a usar un paquete de herramientas de ROS que tenemos disponible en Matlab [21]. Con este paquete el programa va a ser capaz de leer los ficheros .bag los cuales generaremos en las simulaciones, y vamos a poder procesar la información del fichero para así poder extraer información que nos ayude a calificar el comportamiento de cada uno de los planificadores que estamos comparando. Para lograr esto necesitamos realizar una serie de ficheros que al ejecutarse realicen el procesado de estos datos [22]. Una muestra de los ficheros que hemos realizado para la extracción de datos se muestra en el Anexo A.4.

5. Resultados y discusión

Una vez obtenidas todas las simulaciones necesarias y extraídas las variables y métricas definidas, pasaremos a comentar los resultados del proyecto. En esta sección se analizarán y discutirán los resultados de las simulaciones.

Antes de empezar hay que decir que el planificador ASR que habíamos estudiado su funcionamiento y parámetros en las Secciones 2.3.5 y 4.1.5, se validó su funcionamiento con un circuito de pruebas diferente y éste era capaz de realizar la navegación y de alcanzar los objetivos que se le imponían, pero con el circuito de pruebas diseñado para la realización de las pruebas no se ha podido conseguir que el robot realizara ninguna navegación. Por lo que cuando se empezaron las simulaciones al ver que no funcionaba y no se hubieran podido extraer datos relevantes sobre este planificador se decidió extraerlo de las simulaciones y comparar los otros 4 planificadores que sí que funcionaron correctamente. No obstante, el estudio del algoritmo y el estudio paramétrico siguen en el trabajo, ya que puede ayudar a comprender el algoritmo y conocer el funcionamiento de sus parámetros. Una vez dicho esto podemos empezar a analizar los resultados y después se discutirán los mismos para finalmente sacar unas conclusiones.

5.1. Resultados

Una vez extraídos los datos en Matlab, hemos obtenido una serie de gráficas y tablas donde hemos recopilado la información de las simulaciones realizadas. Vamos a mostrar todos esos datos y comentar el comportamiento de los distintos planificadores.

En primer lugar, vamos a ver cómo se han comportado los planificadores en cada tramo del circuito. Se analizará el tiempo que han tardado en recorrer cada tramo, así como el tiempo de media invertido por tramo y el tiempo total que tardaron en dar la vuelta entera al circuito.

TR			
Tiempo (s)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	50.750	49.470	56.600
Target 2	80.270	71.160	83.710
Target 3	69.280	66.380	80.960
Target 4	57.110	41.250	46.880
Target 5	24.430	87.170	35.460
Promedio	56.358	63.086	60.722
Total	281.840	315.430	303.610

Tabla 6: Tiempos del circuito del planificador TR.

DWA			
Tiempo (s)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	50.55	89.32	46.59
Target 2	75.06	64.29	82.97
Target 3	65.12	77.71	69.53
Target 4	54.71	46.51	45.41
Target 5	19.98	27.71	51.87
Promedio	53.084	61.108	59.274
Total	265.42	305.54	296.37

Tabla 7: Tiempos del circuito del planificador DWA.

EBAND			
Tiempo (s)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	71	62.2	101.05
Target 2	64.3	83.35	155.25
Target 3	49.69	82.09	134.95
Target 4	38.6	78.55	94.65
Target 5	20.56	27.99	47.55
Promedio	48.83	66.836	106.69
Total	244.15	334.18	533.45

Tabla 8: Tiempos del circuito del planificador EBAND.

TEB			
Tiempo (s)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	85.86	65.55	92.53
Target 2	191.49	93.42	135
Target 3	89.97	68.2	90.3
Target 4	63.13	54.19	70.87
Target 5	56.15	35.05	52.96
Promedio	97.32	63.282	88.332
Total	486.6	316.41	441.66

Tabla 9: Tiempos del circuito del planificador TEB.

Una vez introducidas las tablas con los tiempos analizamos qué ha sucedido durante las simulaciones. En el modo agresivo el planificador que ha conseguido llevar al robot más rápido ha sido el planificador EBAND (ver Tabla 8), por lo contrario el planificador TEB (ver Tabla 9) fue el más lento. En lo que corresponde al modo normal y al modo suave, el planificador más rápido fue el DWA (ver Tabla 7), mientras que en estos dos casos fue el planificador EBAND quien circuló más lentamente hacia la meta.

Hablando de tiempos en los diferentes tramos vemos que el primer tramo del circuito para los planificadores TR (ver Tabla 6) y DWA (ver Tabla 7) han obtenido el segundo tiempo más rápido de todas sus pruebas, mientras que pata los planificadores EBAND (ver Tabla 8) y TEB (ver Tabla 9) se puede ver que ha sido mucho más costoso. Por ese motivo podemos decir que para la circulación en pasillos estrechos los planificadores TR y DWA se comportan de mejor forma que el EBAND y el TEB.

En el segundo tramo, todos los planificadores han obtenido tiempos muy elevados. Esto se debe a que al ser un pasillo largo el planificador global calculaba el tramo por encima de la pared, por lo que el robot debía detenerse y recalcular la ruta para ir por el camino correcto, por ello vemos estos tiempos tan elevados.

El tercer tramo también tenía una trayectoria extensa. En este tramo la dificultad residía en la planificación larga por una zona amplia y sin obstáculos o una ruta más corta pero con una dificultad mayor. Apuntar que, en esta prueba el planificador TEB (ver Tabla 9) fue el único de los 4 planificadores comparados de los que eligió la ruta larga. En este mismo podemos observar que fue el más rápido para los parámetros óptimos, ya que el resto de planificadores eligió la ruta difícil y tuvo que recalcular su ruta para evitar colisiones. Pero para los modos agresivos y suaves sí que fue más largo ese tiempo, haciendo uno de los peores tiempos.

En el cuarto tramo TR (ver Tabla 6) y DWA (ver Tabla 7) volvieron a ser los más veloces, aunque en el modo agresivo EBAND (ver Tabla 8) consiguió el mejor tiempo en este tramo. Por lo que respecta a TEB (ver Tabla 9) sigue siendo el más lento de los 4 planificadores.

En el quinto tramo, se puede ver que se han obtenido los tiempos más rápidos de todo el circuito. En este caso, otra vez fue DWA (ver Tabla 7) quien hizo los tiempos más rápidos en todos los modos. Luego podemos decir que el mejor planificador en zonas con objetos es el DWA, aunque también decir que EBAND (ver Tabla 8) en modo agresivo solo fue más lento que DWA por tan solo unas décimas de segundo. Una vez vistos los tiempos podemos extraer que el planificador más rápido ha sido DWA, y por el contrario el más lento fue TEB (ver Tabla 9).

A continuación, vamos a seguir analizando el rendimiento de los algoritmos con los datos de precisión para alcanzar el punto objetivo. En esta parte vamos a ver cómo de precisos han sido los planificadores a la hora de acercase a la meta, antes de empezar a analizar los resultados. Se ha analizado el porcentaje de error, así como la media de error en cada vuelta.

TR			
Error(%)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	0.797	2.396	2.250
Target 2	6.687	5.283	3.745
Target 3	1.275	2.380	1.653
Target 4	7.021	7.055	7.575
Target 5	1.528	0.917	1.560
Promedio	3.4618	3.6062	3.3567

Tabla 10: Precisión del robot usando TR.

DWA			
Error(%)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	1.2104	0.8299	1.2871
Target 2	2.8431	1.5533	1.1225
Target 3	0.8855	0.9554	1.6483
Target 4	6.3658	7.107	5.2329
Target 5	2.2721	2.3992	1.1435
Promedio	2.7154	2.569	2.0869

Tabla 11: Precisión del robot usando DWA.

EBAND			
Error(%)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	1.1249	0.835	0.7069
Target 2	1.4762	3.8628	3.7796
Target 3	2.1671	2.3388	0.8159
Target 4	5.7533	7.5582	6.8287
Target 5	1.2424	1.6989	1.8198
Promedio	2.3528	3.2587	2.7902

Tabla 12: Precisión del robot usando EBAND.

TEB			
Error(%)	Conducción agresiva	Conducción normal	Conducción suave
Target 1	2.6025	2.1592	1.3645
Target 2	3.7099	2.289	3.1497
Target 3	2.0828	1.6923	2.3005
Target 4	6.8996	7.1176	6.9386
Target 5	0.8026	1.186	2.7158
Promedio	3.2195	2.8889	3.2938

Tabla 13: Precisión del robot usando TEB.

Al igual que en el caso anterior, hemos colocado las tablas de los errores de precisión y ahora vamos a analizar los datos que hemos extraído en esta métrica. Si nos fijamos en temas de precisión, al realizar las simulaciones se ha visto que el robot llegaba al destino todas las veces y simplemente

en lo mostrado en las simulaciones se apreciaba que el robot se quedaba en el punto y la orientación indicadas.

En todos los planificadores hemos puesto una tolerancia de 0.2 metros en la posición y una tolerancia de 0.1 radianes para la orientación. Con esto hemos recogido las posiciones finales y hemos sacado el porcentaje de error que tenían los planificadores. Lo primero que observamos, es que tal y como habíamos supuesto en las simulaciones, los acercamientos a los objetivos han sido muy buenos, ya que, si nos fijamos tenemos un 3 % de error de media, lo que supone que el robot se quedaba de media a 0.6 cm del punto final. Pero, siendo un poco más específicos, vamos a analizar los datos de cada uno para ver cuál es el que mejor comportamiento tiene. Aunque, como ya hemos dicho, la precisión que presentan todos los planificadores es muy buena, y por tanto podemos decir que todos los planificadores en esta prueba son bastante eficaces.

En conducción agresiva, si nos fijamos en los errores medios vemos que el planificador EBAND (ver Tabla 12) ha sido el mejor en aunque por una diferencia muy pequeña. Aunque fijándonos en el error cometido en cada trayecto, hay planificadores que han realizado unos errores muy pequeños, como el planificador TR (ver Tabla 10) donde se produjo un error del 0.7 % o el planificador TEB (ver Tabla 13) que ha obtenido errores del 0.8 %.

Si nos fijamos en los acercamientos al objetivo 4 podemos ver como el porcentaje sube el doble de la media, esto ocurre para todos los planificadores y para todos los modos de conducción por lo que se puede suponer que la posición y orientación, sumada a la trayectoria que los planificadores han calculado hacen que la aproximación sea peor en todos los casos. Así hemos encontrado una prueba en la que podemos examinar la precisión de los planificadores cuando se acercan al objetivo. En cuanto a los modos de conducción normales y suave el planificador DWA (ver Tabla 11) ha sido el mejor, aunque tal y como hemos comentado líneas arriba, este planificador ha sido el mejor pero con muy poca diferencia con el resto de errores calculados.

En resumen, hablando de la precisión, todos los planificadores tienen un comportamiento muy similar en todas las pruebas, por lo que no se puede afirmar cuál es el mejor planificador en esta prueba. Para ello, necesitaríamos realizar más pruebas sobre los mismos puntos analizados para ver si los datos se mantienen estables y son significativos estadísticamente o si por el contrario varían y conseguimos extraer una conclusión acerca de la precisión.

Una vez vista la precisión que presentan los diferentes algoritmos, vamos a analizar otra de las métricas a estudiar, las velocidades registradas en las simulaciones. Para el análisis de las velocidades se han hecho diferentes gráficas con las que vamos a poder averiguar cómo se ha comportado el robot.

Hemos hecho dos gráficas por cada modo de comportamiento. En una de ellas hemos puesto la velocidad que ha tenido el robot durante tola la vuelta al circuito, mientras que la segunda consiste en las velocidades medias que ha tenido el robot en cada tramo. Antes de empezar vamos a poner la leyenda para que sea más visual la comparación entre planificadores.

Leyenda		
Planificador	Color	Marca
TR	Rojo	◊
DWA	Azul	×
EBAND	Verde	□
TEB	Magenta	○

Tabla 14: Leyenda planificadores.

En primer lugar, vamos a analizar el modo de conducción agresiva. Como podemos ver en la Figura 35, tenemos los 4 planificadores comparados, en los que podemos ver 5 puntos que corresponden a la velocidad que ha obtenido el planificador en cada trayectoria. De este modo podemos ver qué planificador calculó la ruta para alcanzar velocidades más elevadas por el circuito, así como si el planificador ha sido capaz de mantener una regularidad por el paso de las pruebas, es decir, su robustez frente a distintas situaciones.

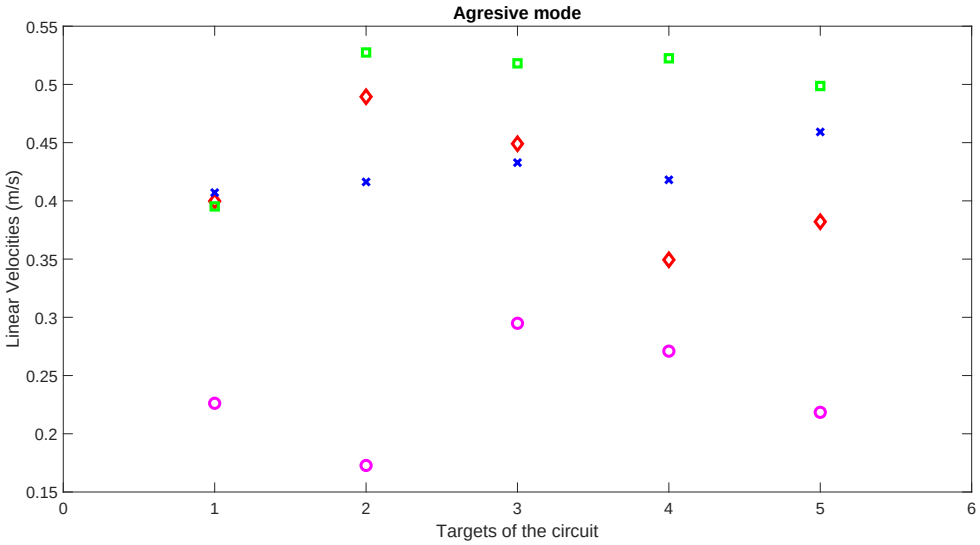


Figura 35: Velocidades medias en modo de conducción agresiva.

El primer planificador es el EBAND, que de media ha sido el más rápido de todos los planificadores. En cuanto a la robustez frente a distintas pruebas, vemos que se ha mantenido bastante constante, aunque durante el primer trayecto podemos ver que la velocidad media fue un poco menor que en las otras partes del circuito. Luego, siguiendo el orden de más veloces a más lentos encontramos a DWA y a TR, los cuales, como podemos ver, son bastante similares. Mientras que TR fue más veloz en las pruebas del principio del circuito, descendió sus velocidades medias al final y DWA tuvo un comportamiento totalmente opuesto al comentado con TR. Analizando la linealidad de estos dos planificadores, podemos observar que en el caso de TR ha sido más irregular y presenta unos altibajos que en el caso de DWA no se aprecian, ya que éste se mantiene bastante estable. Por último, el planificador más lento de los 4 ha sido el planificador TEB, el cual ha estado muy por debajo de las velocidades de los demás planificadores. Además, en cuanto a la regularidad del trayecto en este modo, TEB ha sido también el que más ha variado su velocidad media en el trayecto.

Por tanto, en modo agresivo podemos decir que el planificador EBAND ha obtenido unos resultados mejores que el resto. Con esto, podemos pasar a la siguiente gráfica donde estudiaremos el modo normal o de parámetros óptimos.

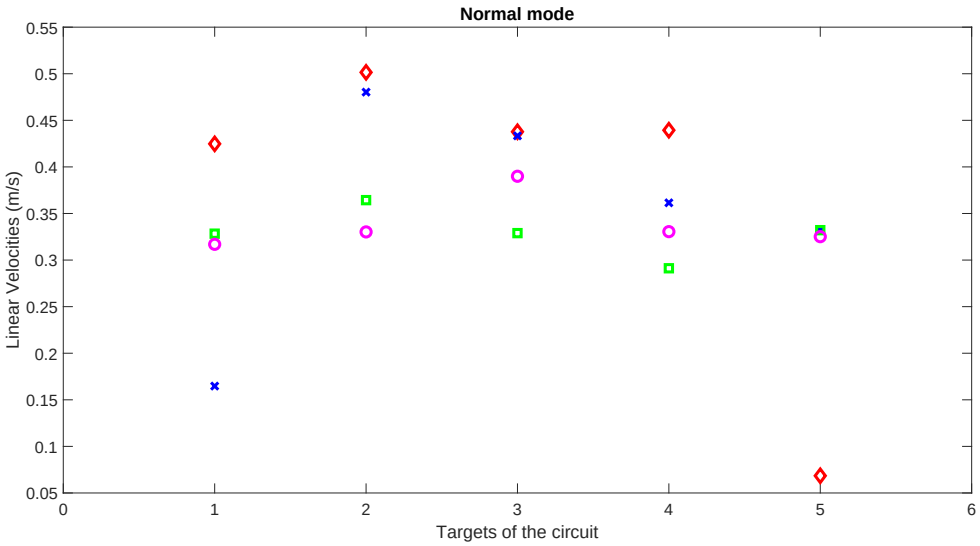


Figura 36: Velocidades medias en modo de conducción normal.

Vamos a analizar la Figura 36, pero a diferencia de la Figura 35, vamos a analizar cada planificador

sin ningún orden, puesto que tal y como se observa las velocidades son más difíciles de ver.

En primer lugar empezaremos por el planificador TR, el cual ha obtenido las velocidades medias más elevadas hasta el último punto donde podemos ver que la velocidad se desploma prácticamente a 0. Por lo que, aunque fue el más rápido, la regularidad en este modo de conducción ha sido la peor de todos los planificadores. En segundo lugar, el planificador DWA ha sido bastante veloz también pero a la par que el planificador TR, DWA también ha sido bastante irregular. En el caso del planificador EBAND, esta vez ha sido mucho más lento, pero al igual que en el modo anterior sigue siendo de los planificadores más estables. Por último, el planificador TEB ha conseguido velocidades más parecidas a los demás planificadores. Además, en este modo de conducción este planificador ha sido mucho más estable. Como añadido, decir que, en las simulaciones, TEB ha calculado rutas en las cuales ha conseguido mantener unas velocidades constantes y suaves, lo que se ve reflejado en la gráfica, pero aún así esto se verá mucho más claro en las siguientes gráficas.

Ahora, vamos a pasar a la siguiente gráfica donde estudiaremos el modo suave. Tal y como en la Figura 36 vamos a comentar los resultados de cada planificador y así finalizar estas medidas. En este caso, en la Figura 37, el planificador DWA fue el planificador más rápido en este modo de conducción cauteloso. Aunque, una vez más el planificador ha sido bastante irregular. Con respecto al planificador TR, las velocidades han sido más lentas que el planificador DWA, pero por el contrario, ha sido más regular que éste, aunque aún así la conducción ha sido bastante irregular. El planificador EBAND en estas simulaciones ha sido el más lento de los 4, aunque por el contrario, si nos fijamos en la Figura 37, podemos ver que el planificador ha realizado una conducción muy estable, tal es así que podemos ver que la línea del planificador EBAND es prácticamente una línea horizontal. Por último, el planificador TEB ha sido más rápido que el planificador EBAND, pero más lento que los otros dos planificadores. Al mismo tiempo, en cuanto a la robustez, también ha sido más irregular que EBAND, pero mucho más estable que DWA y TR. Una vez vistos los 3 modos de conducción, podemos decir que los planificadores DWA y TR por lo general son más veloces, pero por el contrario son bastante irregulares, mientras que EBAND y TEB son bastante más robustos en este aspecto.

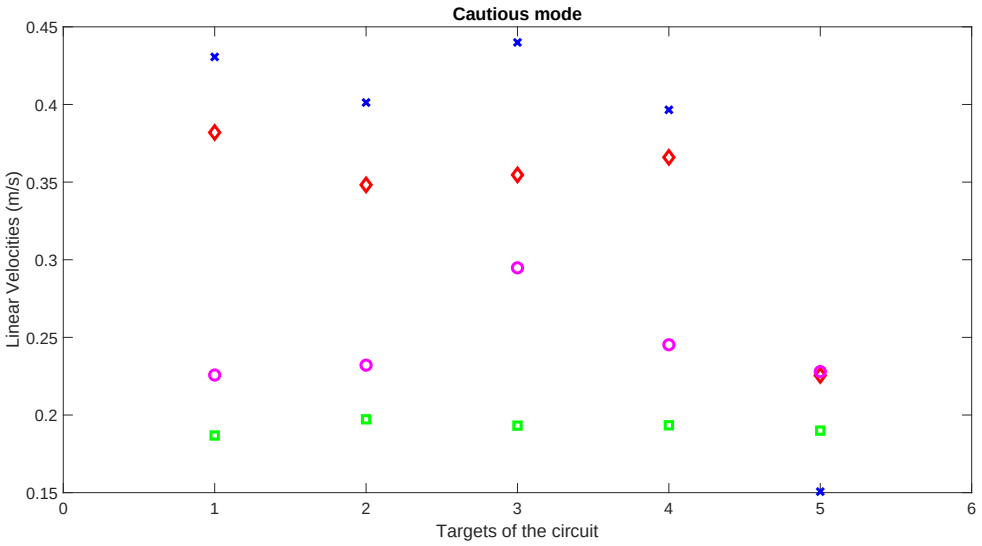


Figura 37: Velocidades medias en modo de conducción suave.

Una vez analizadas las velocidades medias vamos a profundizar en el análisis de las velocidades durante toda la trayectoria, con esto podremos ver cómo se han comportado los planificadores durante todo el trayecto. También nos ayudará a comprender mejor el funcionamiento, como las veces que el robot ha tenido que parar o disminuir la velocidad para intentar seguir la ruta global establecida. Al igual que en el caso anterior, vamos a empezar por el modo de conducción agresiva.

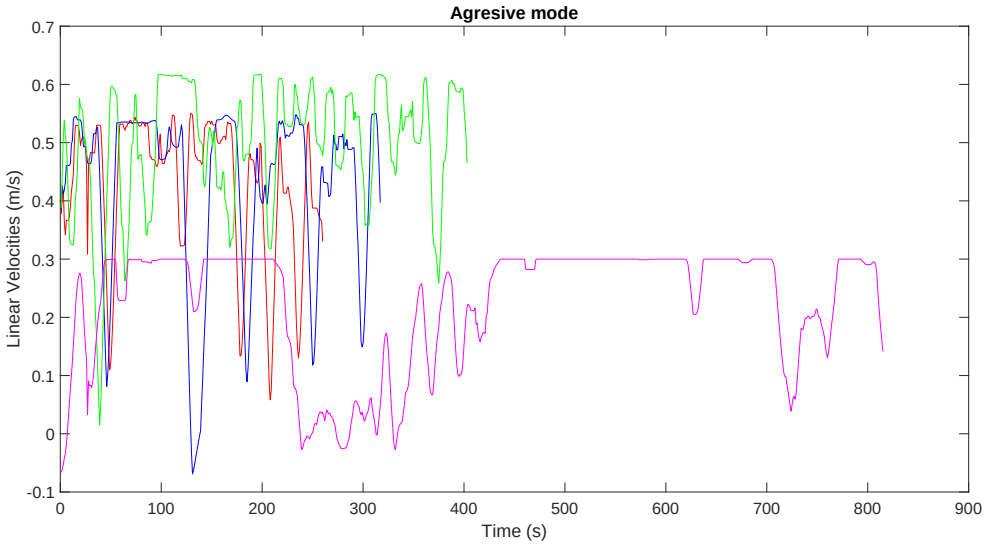


Figura 38: Gráfica de velocidades en modo agresivo.

Para empezar, la Figura 38 nos muestra el mismo color correspondiente con la leyenda de la Tabla 14. El planificador TR (rojo) ha ido cambiando de velocidad bruscamente, esto significa que el robot ha ido acelerando hasta su máxima velocidad pero como ha tenido que recalcular rutas o simplemente evitar algún obstáculo que se encontraba en su trayectoria, se observa que el robot frenaba prácticamente en seco para realizar la trayectoria hasta el final.

Ahora vamos a fijarnos en el planificador DWA (azul), con el que, del mismo modo que el planificador TR, vemos un comportamiento bastante irregular. Además podemos ver un pico donde el planificador hizo que el robot se detuviera para recalcular la ruta.

Para el planificador EBAND (verde), podemos apreciar aparentemente un comportamiento irregular, pero a diferencia de los planificadores vistos anteriormente, vemos que los picos tienen una amplitud mucho más corta, por lo que estas velocidades podrían ser de cambios de velocidades debidos a la ruta calculada.

Por último, vemos el planificador TEB (magenta) que ha mantenido una velocidad constante durante muchos tramos, lo que indica que la ruta calculada por este planificador hace que el robot pueda conducir sin problemas por el circuito y no tenga que recalcular rutas. Por lo tanto, con respecto a la movilidad del robot por el circuito, el planificador TEB, aunque fue el más lento, tuvo el mejor comportamiento de todos los planificadores para este modo de conducción.

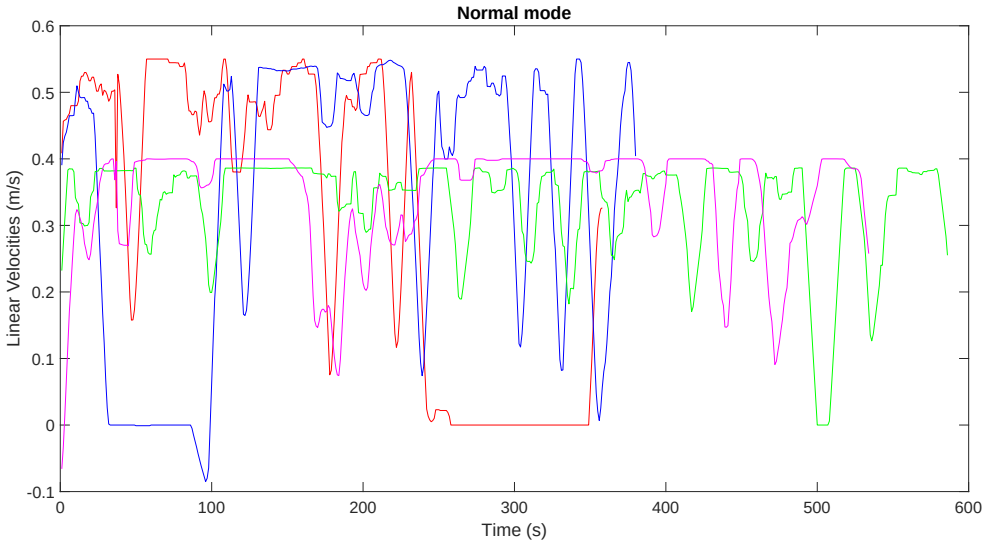


Figura 39: Gráfica de velocidades en modo normal.

Ahora analizaremos la gráfica de las velocidades para el modo de conducción normal. Para este modo de conducción (ver Figura 39), vamos a analizar los planificadores del mismo modo que en la Figura 38, para posteriormente elegir un planificador que consideraremos el mejor en lo que a estas pruebas respecta.

Tal y como ocurrió en la Figura 38, los planificadores TR (rojo) y DWA (azul) son los más irregulares en lo que a la conducción respecta. Es cierto que son los más veloces y que en este modo de conducción normal tienen tramos donde se mantiene la velocidad constante. En consecuencia podemos decir que en este caso se ha ejecutado la ruta de mejor forma que en el modo anterior. También vemos que el planificador DWA al principio estuvo un tiempo detenido, luego vemos una velocidad negativa, lo que quiere decir que el robot tuvo que retroceder para volver a la ruta calculada. En el caso del TR vemos que estuvo durante un tiempo detenido recalculando la nueva ruta y cuando la obtuvo siguió con la nueva ruta, pero a diferencia de DWA, TR no tuvo que retroceder para seguir con la trayectoria calculada.

Por lo que hace al planificador EBAND (verde), podemos ver que en este modo el planificador se comporta de una forma mucho más estable, aunque es la intención de los diferentes modos de conducción. Podemos ver que el planificador ha mantenido la velocidad más o menos constante, aunque podemos ver unos cambios de velocidad, pero si nos fijamos en la Figura 39 podemos ver que las pendientes no son tan bruscas como otros planificadores. De hecho, estos cambios se podrían deber a los cambios de velocidad debidos a la apariencia del circuito de pruebas.

Para finalizar el planificador TEB (magenta), como en el caso anterior, vemos que su gráfica de velocidad es bastante estable y con cambios, al igual que el planificador EBAND. Si nos fijamos bien podemos ver cómo los cambios de velocidades se corresponden en ambos planificadores por lo que deducimos que significa que el robot necesitaba disminuir la velocidad a causa del propio circuito.

En este modo de conducción, volvemos a ver que el planificador TEB es el mejor, pero esta vez comparte protagonismo con el planificador EBAND, ya que, a diferencia que en el caso anterior en este modo ha conseguido una gran conducción.

Finalmente, vamos a pasar a estudiar el modo cauteloso, y con ello habremos finalizado el estudio de velocidad, por lo tanto pasaremos a estudiar las aceleración del robot, que nos aportará más información. Para este último modo de conducción vamos a analizar las gráficas de la Figura 40. Como en los modos mostrados en las Figuras 38 y 39, los planificadores DWA y TR se comportan del mismo modo, por lo que como ya se ha dicho, estos planificadores tienen un comportamiento irregular, ya que como se puede ver tenemos unos cambios muy bruscos de velocidades debido a cálculos incorrectos de trayectoria.

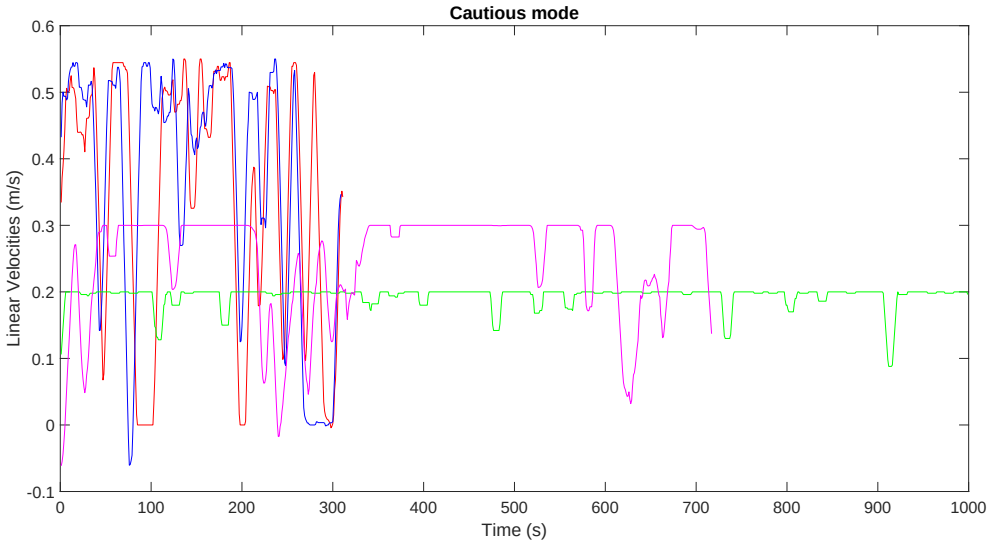


Figura 40: Gráfica de velocidades en modo suave.

Del mismo modo, el planificador TEB (magenta) se ha mantenido constante, teniendo un comportamiento bastante regular excepto en algunas ocasiones donde se ven algunos cambios de velocidad pero muchos menos bruscos que en los otros planificadores. En este modo de conducción, es el planificador EBAND (verde) el que ha realizado la conducción más constante y más regular, teniendo como

resultado una línea prácticamente horizontal. Por lo que la planificación de la ruta con el algoritmo EBAND en este modo ha sido con diferencia la mejor.

Una vez vistas las velocidades que se han obtenido en las simulaciones (ver Figuras 35-40), vamos a analizar las aceleraciones registradas por la IMU integrada en el robot. Al igual que en el caso de las velocidades, tenemos dos tipos de gráficas diferentes, una donde se han juntado las aceleraciones medias de la trayectoria y otra con las aceleraciones registradas por la IMU durante las simulaciones. En estas gráficas lo que se podrá observar será la variación de la aceleración, lo que nos indicará los distintos movimientos que ha realizado el robot y si éste ha hecho movimientos bruscos.

En primer lugar, vamos a ver cómo se han comportado los planificadores para el modo agresivo. Para ello estudiaremos la Figura 41, donde vemos que el planificador EBAND es el que tiene una aceleración más elevada. Por la forma de la gráfica este planificador en este modo de conducción ha tenido unos cambios de velocidades constantes. Para los planificadores TR (rojo) y DWA (azul) vemos que ha tenido unas aceleraciones menos bruscas, pero por el contrario estas han sido más dispares, lo que supone una conducción menos continuada, debido a parones y acelerones causados por el cálculo de rutas locales nuevas al no poder seguir la ruta global debido a obstáculos o giros donde el robot no calculo bien la ruta y no pudo tomar la curva de una forma eficiente. Por último, con respecto al planificador TEB, vemos que tiene una aceleración más baja que los demás lo que significa que el robot ha mantenido la velocidad más constante y los cambios de velocidad que ha realizado han sido menos bruscos causando aceleraciones pequeñas.

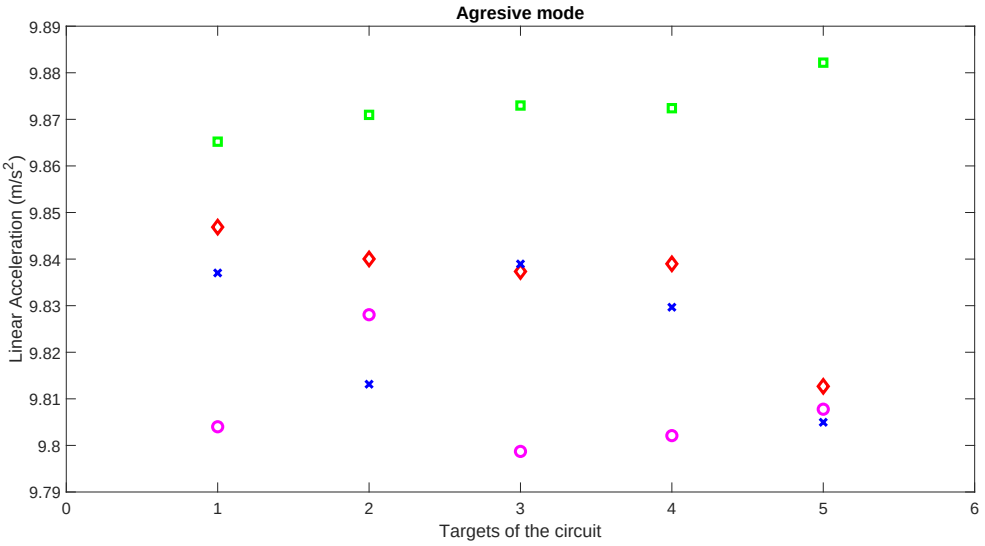


Figura 41: Aceleraciones medias en modo de conducción agresiva.

Cabe recordar que la Figura 41 corresponde con las aceleraciones medias de cada trayectoria, así que las variaciones que vemos de un punto a otro se podrían deber a la situación de distintas pruebas, ya que el robot no se estaba sometiendo a las mismas pruebas. Por ejemplo, en el tercer tramo podemos ver un aumento de la aceleración para todos los planificadores excepto para el planificador TEB. Esto indica que en esta prueba el robot no ha podido circular con velocidad constante y ha habido mucha variación de la velocidad. Cuando estudiemos las siguientes gráficas, veremos en qué momentos el robot ha acelerado o frenado.

Siguiendo con la comparativa de los planificadores vamos a ver cómo se han comportado y qué valores ha registrado la IMU para el modo de conducción normal. En este modo de conducción mostrado en la Figura 42, el comportamiento del planificador EBAND ha cambiado ya que esta vez las aceleraciones medias de cada tramo de conducción han sido más lentas, pero por lo que respecta a la constancia de la aceleración se aprecia que durante toda la simulación las aceleraciones han sido bastantes regulares. El planificador TR durante el primer tramo obtuvo unas aceleraciones bastante elevadas lo que nos puede indicar la irregularidad de este planificador a la hora de planificar una ruta. El caso del planificador DWA es bastante parecido a la Figura 41, aunque en el primer tramo se obtuvieron unas aceleraciones bastante bajas, en el tercer tramo el robot registró aceleraciones muy altas, lo que indica que durante las pruebas de ese tramo, el robot debió de cambiar su rumbo para conseguir el objetivo establecido. Por último, vemos que el planificador TEB fue el más constante

de la circulación teniendo unos cambios de velocidad mucho más suaves, es decir, si el planificador necesitaba frenar o acelerar durante la conducción lo realizó de un forma más progresiva. Es por eso que los valores reflejados en la gráfica son los más bajos de los 4 planificadores, siendo así el algoritmo de navegación que mejor valores ha obtenido para este modo de conducción.

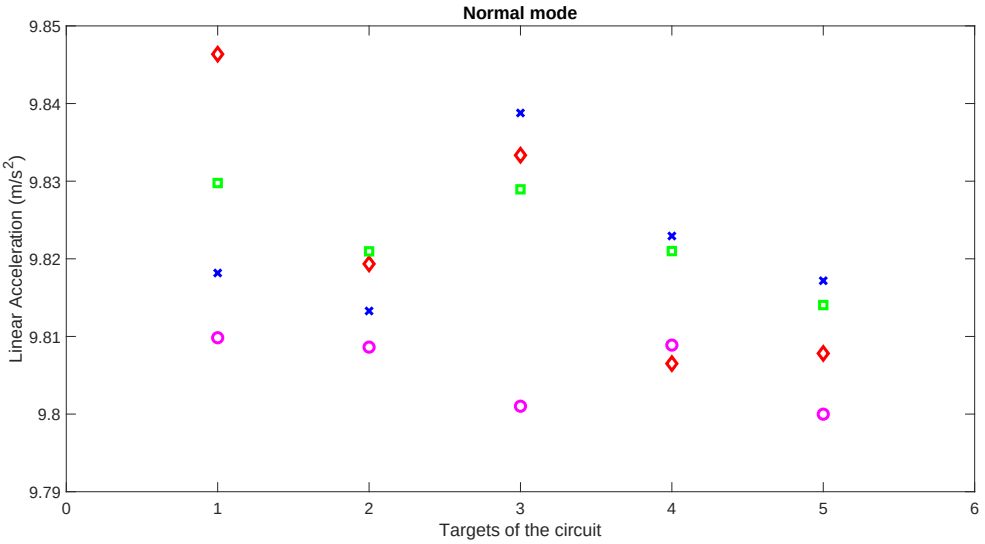


Figura 42: Aceleraciones medias en modo de conducción normal.

Obtener buenas puntuaciones en este modo es bastante significativo, ya que éste es el modo en el que analizamos los planificadores con los valores óptimos del panificador, por lo que estos son los valores con los que el planificador realizará las mejores conducciones, a priori. Dicho esto, vamos a pasar a analizar el último modo de conducción, en el cual veremos aceleraciones más bajas ya que al ser un modo cauteloso de navegación, los valores marcados para la conducción son más bajos y no pueden alcanzar picos tan elevados como en los otros modos de navegación.

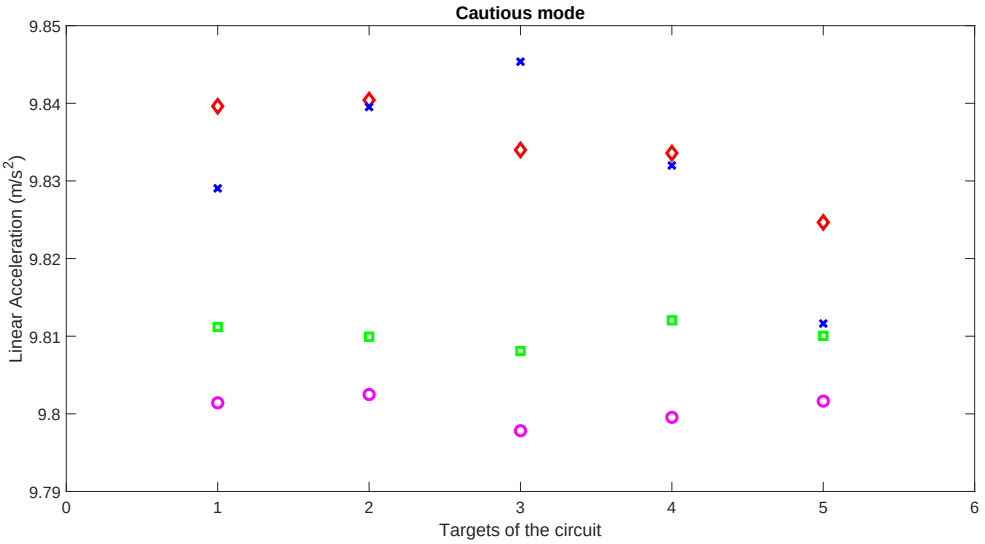


Figura 43: Aceleraciones medias en modo de conducción suave.

En la Figura 43 vemos unas curvas mucho más suaves que en los casos anteriores. Tal y como estamos viendo gráfica tras gráfica, los planificadores TR y DWA están siendo los más irregulares. La Figura 43 muestra perfectamente que las aceleraciones registradas por estos dos planificadores está por encima del resto, esto nos indica unos cambios de velocidad bastante más bruscos que en el resto de casos. Con las Figuras 41, 42 y 43 hemos respaldado las conclusiones que obteníamos de las gráficas

de las velocidades, puesto que se nos mostraban unas gráficas con unas pendientes muy elevadas, que aquí se reflejan. Por el contrario, EBAND y TEB van mejorando con cada modo de conducción que pasamos, llegando así a superar a los planificadores anteriores. En la Figura 43 se pueden ver claramente las aceleraciones con un valor menor, lo que indica suavidad a la hora de la conducción, y además, constancia durante las pruebas. Esto nos indica que dichos planificadores pueden mantener un comportamiento constante en diferentes entornos, lo que se traduce en una robustez del algoritmo superior a los otros dos planificadores.

Vamos a seguir poniendo a prueba estas primeras teorías que vamos extrayendo de los datos, añadiendo unas gráficas donde podremos ver las aceleraciones que ha sufrido el robot durante toda la simulación, al igual que en el caso de las velocidades. Así será posible comprender mejor el funcionamiento de cada uno de estos planificadores. Para seguir con el análisis de las aceleraciones, vamos a realizar el mismo procedimiento que estamos siguiendo hasta ahora, primero analizaremos el modo de conducción agresivo, mostrado a continuación en la Figura 44.

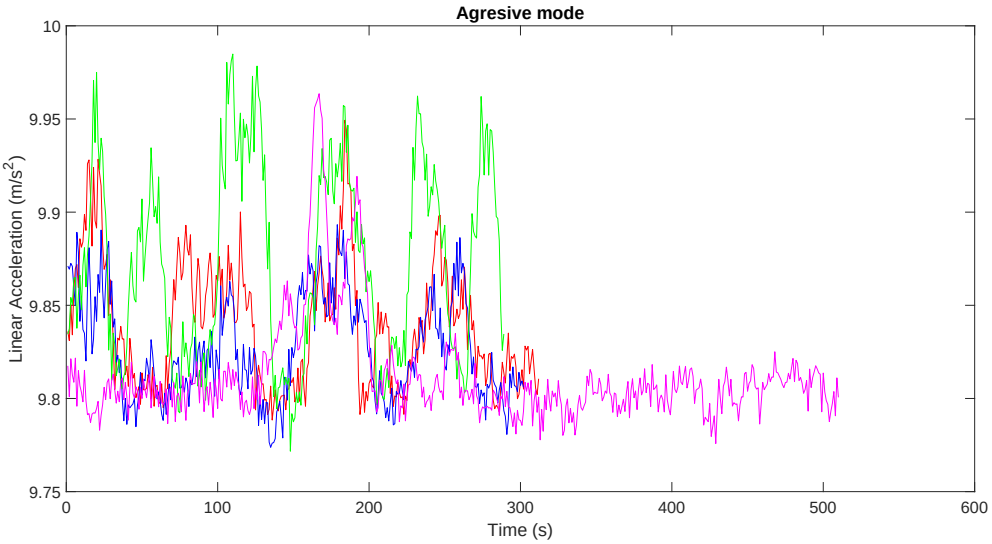


Figura 44: Gráfica aceleraciones en modo agresivo.

En este primer modo de conducción podemos ver que el planificador EBAND es el que se comporta con más nerviosismo, es decir, se aprecian subidas y bajadas de aceleración, lo que puede significar que el robot intenta mantener una velocidad constante, pero debido a la planificación realizada no es capaz de mantenerla por lo que necesita estar en constante cambio para no colisionar y fallar en la obtención del objetivo. Siguiendo esta regularidad, en este modo, TR y DWA serían los siguientes planificadores que muestran estos altibajos, pero en este caso, podemos ver que los planificadores se comportan bastante mejor que EBAND y además son más veloces en realizar toda la vuelta al circuito. Por último, tenemos al planificador TEB que mantiene un nivel bastante constante, podemos ver una variación de los valores al igual que en los otros casos, pero dicha variación es mucho menor que en los otros casos, lo que indica que el robot no ha tenido que modificar la trayectoria ni la velocidad de forma brusca, sino que ha sabido conducir de forma más constante.

Es importante destacar que, en la Figura 44, durante el tiempo de simulación entre 150 y 200 segundos, podemos apreciar que todos los planificadores sufren un gran pico en la medida de la aceleración. Al ser una subida general, podría tratarse de un punto del circuito de pruebas en el que la planificación es muy compleja y, por tanto, todos debieron realizar un cambio brusco de la aceleración para conseguir completar el circuito.

Siguiendo con la comparativa, vamos a pasar a ver las aceleraciones obtenidas por los distintos planificadores estudiados en el modo normal de funcionamiento. En este caso de conducción con los valores óptimos (Figura 45), vemos que el comportamiento de EBAND mejora mucho, siendo aun bastante irregular en su conducción pero mejorando el resultado que ofrecen TR y DWA, que, una vez más tienen picos de acelerones y frenazos causados por la mala gestión de la trayectoria. Además, vemos que en este caso TR es el planificador más lento, esto se debe a que durante el transcurso de la simulación tuvo algún problema y se tuvo que detener por completo hasta que encontró una solución y es por eso que esta vez supera a TEB que es el planificador más lento de los 4. Por lo

que respecta a DWA tiene un comportamiento bastante similar a TR, en cuanto al comportamiento brusco de acelerones y frenazos. Pero podemos apreciar que los picos de aceleración que presenta este planificador ocurren en diferentes puntos de la trayectoria. Por último, en este modo de conducción TEB sigue siendo el que más suave navega, es decir, su conducción, tal y como se muestra en la Figura 45, tiene aceleraciones pero se encuentran dentro de un rango normal debido a la conducción y a los diferentes obstáculos que nos podamos encontrar durante la simulación.

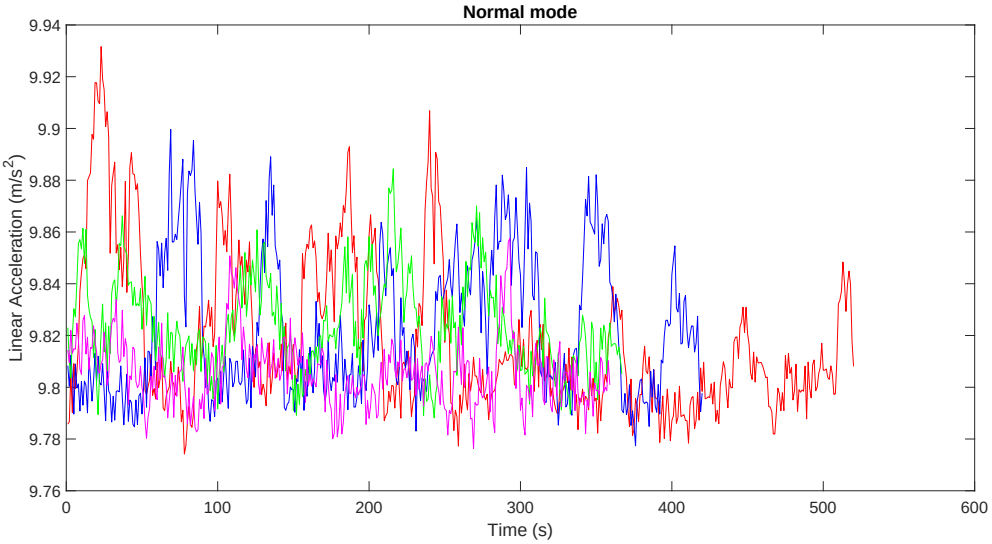


Figura 45: Gráfica aceleraciones en modo normal.

En resumen, en el caso de los valores normales, se puede comprobar que los planificadores EBAND y TEB tienen una mejor conducción por este circuito de pruebas que los planificadores TR y DWA, los cuales tienen una conducción mucho mas irregular. Esto podría causar que el robot volcara debido a una aceleración muy fuerte que superara el peso del robot y, por tanto, éste acabaría tirado en mitad del circuito.

Por último, para concluir con las aceleraciones, vamos a estudiar el caso de la conducción más suave y segura. En este modo, deberíamos encontrar las aceleraciones más estables de todas las simulaciones.

En primer lugar, vamos a comentar los planificadores TR y DWA, los cuales son los que se comportan de forma más brusca, teniendo unos grandes picos en la aceleración, producidos por cambios bastantes buscos en la conducción. Esta vez, la duración del tiempo está dentro de los parámetros normales, por lo que podemos decir que no tuvieron ningún peligro de colisión como en el caso anterior, y el cálculo de la trayectoria fue mejor que en el caso anterior, pero aun así han obtenido unas aceleraciones muy elevadas para este modo de conducción.

En segundo lugar, vamos a hablar de EBAND, el cual se puede apreciar su evolución a medida que pasamos de modo, es el planificador que más se ve afectado por el cambio de los parámetros, esto lo lleva a convertirse en el mejor para este modo. Podemos ver en la Figura 46 un notable descenso de la aceleración, lo que ya indica que sus cambios de velocidad fueron bastante suaves. Por último, y siguiendo la tendencia vista en otros modos, TEB se ha vuelto a comportar de forma bastante suave, en este caso, como ya hemos comentado, EBAND estaría por delante de TEB, pero aun así en este modo de funcionamiento ambos planificadores han tenido una conducción bastante similar. En consecuencia, el ganador en este caso no se puede ver a simple vista al igual que en los otros modos de conducción.

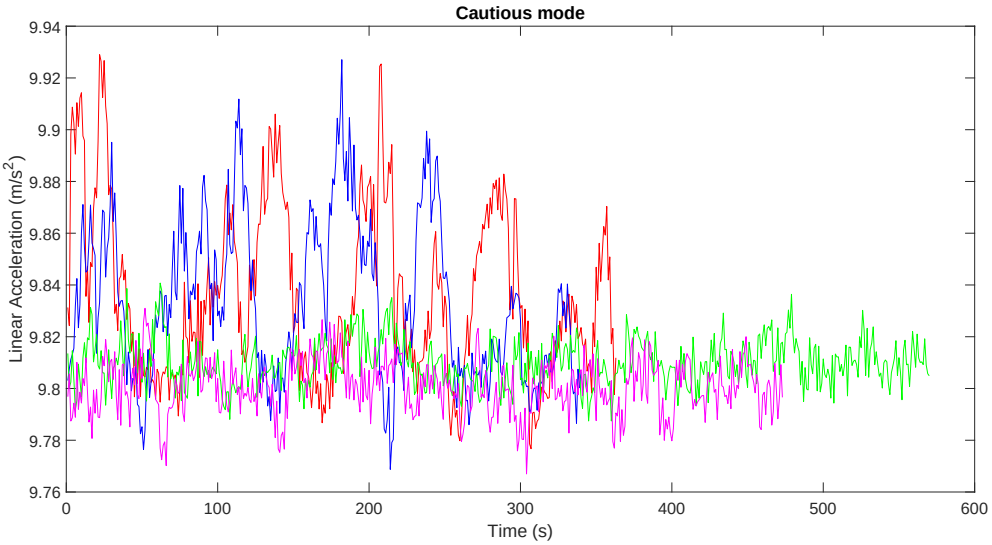


Figura 46: Gráfica aceleraciones en modo suave.

Como ya se ha comentado, los planificadores EBAND y TEB tienen una conducción más fina, es decir, no necesitan estar recalculado la ruta cada vez que surge algún problema. Mientras están en movimiento son capaces de evadir de los obstáculos necesarios para cumplir el objetivo.

Para seguir con la comparativa, vamos a ver otra de las métricas estudiadas, la distancia registrada por el láser durante toda la simulación. El láser de este robot es un láser que capta un rango de 360º por lo que si el robot se acerca a algún obstáculo por algún lado, quedará registrado. Lo que vamos a ver a continuación son las distancias medias y mínimas de cada planificador a la hora de hacer las simulaciones. Con esto seremos capaces de ver a qué distancias conducía nuestro robot. Por ejemplo, en la curva en forma de U algunos planificadores hacían la curva pegados a la pared como si estuvieran siguiéndola, mientras que otros planificaban seguían la ruta por medio del carril, lo que les daba un mayor margen de maniobra a la hora de realizar la curva.

Vamos a empezar en primer lugar con las distancias medias que se han obtenido de las simulaciones en los diferentes planificadores. Como podemos ver en la Tabla 15, todos los planificadores tienen una media que ronda entorno a los 2.5 metros, pero hay uno que ha tenido unos valores bastante más elevados. Se trata del planificador TEB que ha circulado en todos los modos de conducción bastante más alejado de objetos que los demás planificadores. De esta tabla podemos ver como efectivamente el planificador TEB realiza una planificación mucho más cómoda para el robot, tal y como hemos explicado en el ejemplo de la curva en forma de U que hemos comentado anteriormente. Cuando veamos las gráficas de las rutas realizadas por los robots durante las simulaciones podremos ver como la línea del planificador TEB encaja con lo comentado con las tablas.

Distancias medias			
dist media(m)	Conducción agresiva	Conducción normal	Conducción suave
Treajectory	2.371	2.400	2.393
DWA	2.415	2.516	2.475
EBAND	2.320	2.476	2.689
TEB	4.202	4.492	4.663

Tabla 15: Distancias medias del robot a los objetos.

Pasemos ahora a estudiar los valores mínimos que se han registrado en las simulaciones. Con esta prueba podremos descubrir si el robot ha tenido alguna colisión (distancia 0) y ver qué punto del robot es el que ha sufrido una colisión.

Distancias mínimas			
dist min(m)	Conducción agresiva	Conducción normal	Conducción suave
Treajectory	0.642	0.928	0.928
DWA	0.6148	0.6243	0.6243
EBAND	0.6608	0.6608	0.6349
TEB	0.70092	1.1814	1.1814

Tabla 16: Distancias mínimas del robot a los objetos.

Destacar que durante las simulaciones el robot no ha sufrido ninguna colisión, y así lo demuestran los datos de la Tabla 16. Esto no quiere decir que el robot se haya acercado demasiado a algún objeto y éste haya tenido que recalcular su ruta para poder seguir adelante, sin llegar a colisionar con el objeto.

En primer lugar, vamos a ver los datos obtenido con el planificador TR. En el modo agresivo vemos que el robot se acercó más al objeto que en los demás modos, pero este registro se corresponde con la parte izquierda de atrás del robot. Lo que puede indicar que el robot superó un obstáculo y volvió a su ruta pasando cerca del objeto que ya había evitado. En modo normal vemos que la distancia al objeto asciende hasta casi 1 metro de distancia, esta vez el robot se acercó al objeto por la parte delantera derecha, lo que indica que el robot detectó el objeto y pudo evitar la colisión y seguir con la ruta establecida. Para el modo suave, vemos que la distancia mínima se repite, esto posiblemente se deba a los parámetros puestos de acercamiento establecidos, que en el caso agresivo dejan de cumplirse por la velocidad del robot y su conducción más errática. Por lo que respecta a la zona de acercamiento también fue por la parte delantera derecha del robot.

En segundo lugar, siguiendo el orden de la Tabla 16, vamos a ver el comportamiento del planificador DWA. En el modo agresivo, al igual que en el caso anterior, obtuvo el mayor acercamiento de todos. Pero a diferencia del caso anterior, DWA ha obtenido unos valores de entorno a unos 60cm en todos los casos, esto indica que el robot se acercaba más a los obstáculos que TR. En este primer caso se acercó por la parte trasera izquierda, igual que en el caso anterior. En el modo normal y suave, del mismo modo que ocurría con TR, vemos que se han obtenido las mismas distancias de acercamiento a los objetos, pero en este caso, diferenciándose del planificador anterior DWA ha registrado estos acercamientos por la misma zona, la trasera izquierda.

En tercer lugar, veamos el planificador EBAND, el cual ha obtenido también resultados bastante similares en ambos modos como DWA, y del mismo modo estos giran entorno a 60cm, aunque hay unos centímetros de diferencia entre ambos. Para el modo agresivo y para el modo normal se obtuvieron unas distancias mayores que en el caso del modo suave, a diferencia de la tendencia que estábamos viendo en los otros dos planificadores. La constante que mantenemos es la zona por la cual se ha realizado el acercamiento. Esta zona es la trasera izquierda y al igual que en DWA se cumple para los 3 modos de conducción.

Por último, pasemos a analizar los datos del planificador TEB. Como ya se ha comentado, este planificador durante las simulaciones se ha podido ver que realizaba rutas mucho más cómodas para el robot, ya que intentaba circular por el centro del circuito. Viendo la Tabla 16, sus distancias mínimas son las mayores de todos los planificadores, siendo el planificador que menos se ha acercado a los objetos. En el modo agresivo, comprobamos que de media es donde los robots se acercan más a los objetos, y en este planificador se sigue manteniendo la tendencia, aunque el valor es 10cm más alejado que el resto. En cuanto a la zona seguimos observando que corresponde con la parte trasera izquierda. El modo normal y suave, vemos que otra vez han obtenido valores iguales, y como también está pasando, mayores al modo anterior, llegando a mantener una distancia igual al doble que los otros planificadores. Además esta vez, diferenciándose de todos los casos vistos anteriormente el acercamiento fue por la zona delantera izquierda.

Pasemos ahora a ver las rutas realizadas por el robot durante las simulaciones de los diferentes planificadores. Con esto vamos a ver qué rutas eligieron los planificadores, y comprender mejor los valores que hemos estado analizando anteriormente.

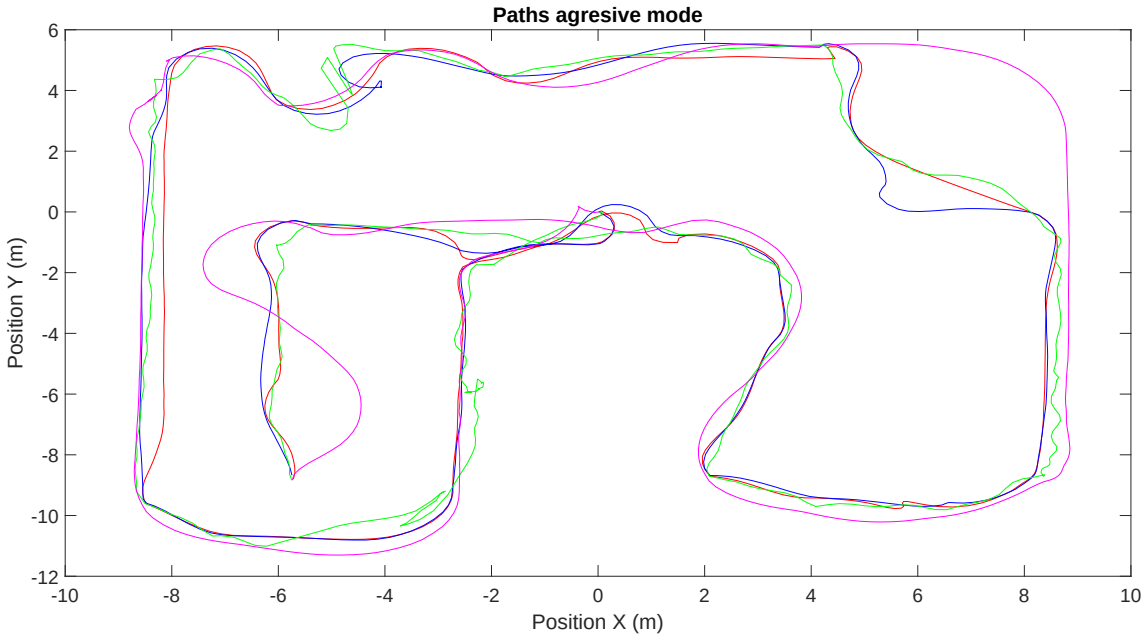


Figura 47: Trayectorias realizadas por el robot en modo agresivo.

De la Figura 47 lo más destacable es la línea de la ruta del planificador EBAND (los colores se corresponden con la Tabla 14), donde podemos ver claramente una conducción muy nerviosa y agresiva, que deja en el mapa un dibujo con movimientos cortos y rápidos. También se aprecian unos recálculos donde parece que el robot fue marcha atrás y trozos de ruta donde se desmarca mucho de la ruta a seguir, debido a ese nerviosismo del robot. Enlazando con las pruebas anteriores, se demuestra que el planificador TEB siempre va un poco más despegado que el resto de planificadores. Además, cosas que remarcar de este planificador en diferentes pruebas como por ejemplo en la toma de decisión de ir por una ruta larga y segura o por el contrario otra más corta pero con una mayor dificultad de paso, éste fue el único que decidió pasar por la zona amplia, y por tanto cuando pasó a la zona recta vemos como la hizo prácticamente recta, a diferencia de los demás que la realizaron acercándose a la pared. Otra prueba donde se diferenció del resto fue en la última prueba donde tenía que llegar al objetivo evitando obstáculos, y mientras que el resto de planificadores eligió una ruta más corta, TEB realizó una ruta bastante más larga y haciendo curvas amplias y suaves evitando los obstáculos a una mayor distancia.

Vamos a ver ahora el modo de conducción normal. En este caso vemos como EBAND deja de comportarse de forma tan agresiva y ya se comporta de una forma más suave, esto se ve reflejado en los datos analizados anteriormente. Podemos ver que en este caso ya no hay recálculo, aunque en la recta de la derecha aún podemos ver algo de esta agresividad del modo anterior. También se aprecia que en la prueba de la elección de trayectoria, aunque EBAND siguió la misma que todos los demás se puede ver que su cálculo fue mucho mejor, ya que hizo una curva suave para pasar por la prueba de forma más cómoda que TR y DWA, los cuales tuvieron que recalcular la nueva ruta. DWA al final del pasillo estrecho realizó una marcha atrás para seguir su funcionamiento. Por lo que respecta al zig-zag, parece que los que mejor han gestionado la prueba fueron DWA y TR.

En la Figura 48 se observa que el planificador TEB realizó una trayectoria bastante similar al caso anterior, aunque se aprecian movimientos más suaves la lógica de conducción fue la misma que comentamos en el caso anterior. Algo que no se ha dicho en la Figura 47, y que ocurre todas las veces, es que cuando empieza la trayectoria hacia un objetivo que no se encuentra delante el robot realiza una marcha hacia atrás para encararse con la trayectoria y es cuando empieza la conducción. Es por ello que podemos ver un triángulo al inicio de cada una de las simulaciones.

Por último, vamos a pasar a analizar el comportamiento en el modo de conducción suave. En este modo veremos un comportamiento en cuanto a la lógica bastante similar a los otros casos, pero nosotros nos fijaremos en el comportamiento del robot.

Vamos a empezar hablando del planificador DWA, el cual ha tenido que realizar un recálculo en la recta de la izquierda, después hizo el zig-zag con gran habilidad al igual que TR, y así llegó a la curva de decisión donde una vez más no fue capaz de calcular una mejor ruta y, como muestra la Figura

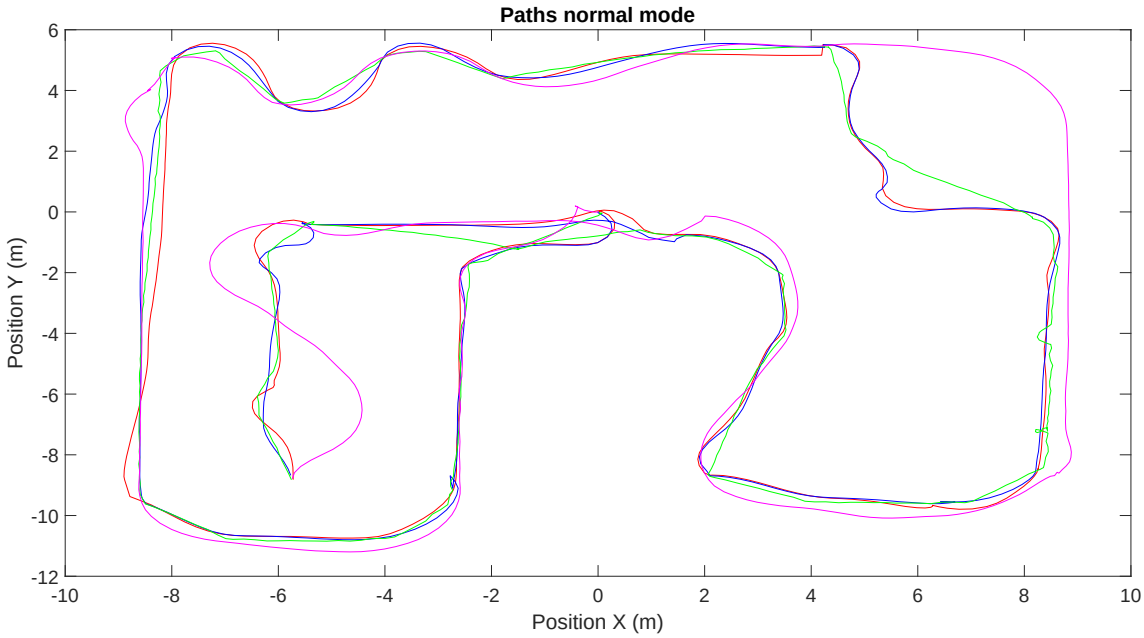


Figura 48: Trayectorias realizadas por el robot en modo normal.

49, se quedó enganchado en la esquina de la curva. Por el contrario, TR sí que fue capaz de realizar una trayectoria mejor que en los modos anteriores, tal y como hizo EBAND, pero más adelante en la curva de la derecha tuvo que recalcular la ruta debido a un posible acercamiento a la pared de la curva. EBAND y TEB tuvieron una conducción bastante similar a los modos anteriores.

Con todo ello acabamos la parte de análisis comparativo de resultados entre los diferentes planificadores estudiados. Hasta el momento nos hemos ajustado a los datos que nos dieron las simulaciones y será en el apartado de discusión donde hagamos un resumen de las distintas pruebas y concluyamos cuáles son los puntos fuertes y los puntos débiles de cada uno de estos planificadores.

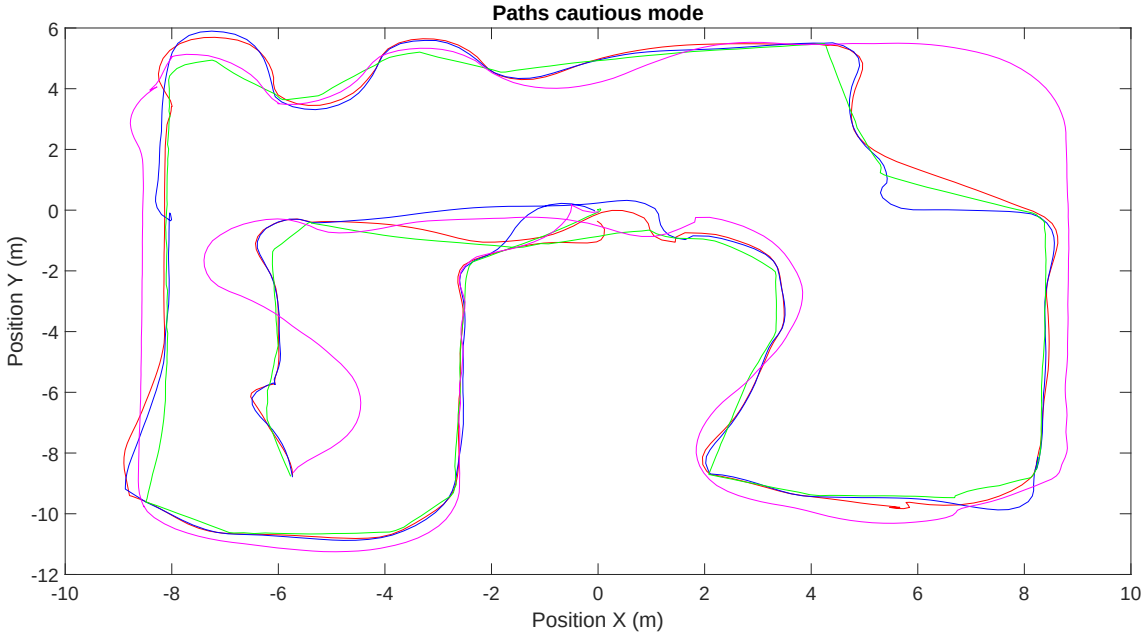


Figura 49: Trayectorias realizadas por el robot en modo suave.

5.2. Discusión

Una vez analizados los datos obtenidos a partir de las simulaciones, podemos realizar una discusión de los mismos para extraer las conclusiones sobre los planificadores y por tanto decidir cuál es el planificador que mejor comportamiento tuvo durante estas pruebas realizadas. A modo más objetivo, se van a evaluar las distintas pruebas con una numeración de 1 al 4 cada uno de los planificadores tendrá una puntuación en cada prueba en función de los resultados obtenidos. La numeración será 4 puntos para el mejor planificador y 1 punto para el peor, lo llamaremos P_i. Luego tendremos unos pesos para ponderar las pruebas al cual llamaremos A_i donde i es el número de la prueba realizada. Por tanto para extraer el resultado de cada uno de los planificadores:

$$Resultado = \frac{\sum_{i=1}^{i} A_i \cdot P_i}{n_{test}} \tag{21}$$

Para la resolución de estos resultados hemos decidido que los pesos de todas las pruebas tengan el mismo valor en este caso $A_1 = A_2 = A_3 = A_4 = A_5 = A_i = 1$. Pero si en estudios posteriores nos pareciera interesante el resultado de alguna prueba en concreto o tuviéramos una funcionalidad que donde alguna prueba no fuera relevante, únicamente cambiando el valor de los pesos obtendríamos los resultados deseados.

En primer lugar, vamos a hablar sobre el planificador TR. Este planificador es uno de los planificadores más comunes en el entorno de ROS, por lo que cabe pensar que será un planificador eficaz y robusto. Pero nada más lejos de la realidad, nos encontramos un planificador que con lo que respecta a los tiempos de simulación (ver Tabla 6), es decir, el tiempo que le costaba realizar todas las pruebas, obtuvo unas muy buenas marcas, donde fue el segundo planificador que más rápido lo consiguió. Aunque fue de los más rápidos, cuando pasamos a observar los datos de velocidades (Figuras 35-40) y aceleraciones (Figuras 41-46) nos dimos cuenta de que este planificador, a pesar de ser muy rápido, realizaba una conducción bastante errática, es decir, iba con acelerones y frenazos bastante bruscos, lo que supuso que en alguna de las simulaciones el robot se tambaleara debido a los cambios bruscos mencionados.

En los datos de precisión (ver Tabla 10, el algoritmo TR, una vez más no pudo destacar frente al resto, aunque es verdad que en esta prueba todos y cada uno de los planificadores obtuvieron unos resultados bastante buenos. Pero por pequeñas diferencias resultó que el planificador más impreciso fue TR.

Pasando a las distancias (ver Tablas 15 y 16), el planificador TR remontó, siendo uno de los que más distancia dejaba frente a objetos, esto nos dice que este planificador puede compensar la brusquedad de su conducción con una seguridad bastante buena frente a los obstáculos. No obstante esta mala conducción sumada a la seguridad hacía que el planificador tuviera que recalcular la trayectoria muchas veces durante la simulación.

Por último, para acabar con este algoritmo, decir que para realizar circuitos en zig-zag este planificador es muy buena opción ya que esta prueba como podemos ver en las gráficas de las trayectorias la realizó todas las veces con una gran eficacia, esto se muestra en las Figuras 47-49.

Puntuaciones para TR			
Pruebas	Conducción agresiva	Conducción normal	Conducción suave
Precisión	1	1	1
Tiempo	2	3	3
Velocidad	2	4	3
Conducción	2	1	1
Trayectoria	2	1	1
Distancia	4	3	2
Resultado final	2.167	2.167	1.83

Tabla 17: Resultados para el planificador TR.

En segundo lugar, haremos un resumen de los datos que hemos extraído del planificador DWA durante estas simulaciones. Con DWA vimos que en lo referente a los tiempos y velocidades fue el más veloz, teniendo unos tiempos menores que TR (ver Tabla 7) pese a tener la misma velocidad durante

el trayecto. Esto se podría deber a los parones que tuvo que realizar TR a causa de los recálculos de la ruta. Pero del mismo modo que TR, el algoritmo DWA también tiene una conducción bastante brusca, por lo que obtuvo unas gráficas de velocidad (ver Figuras 35-40) y aceleración (ver Figuras 41-46) bastante puntiagudas y por tanto provocó, al igual que en el caso comentado anteriormente, un tambaleo del robot que pudo haber supuesto el vuelco del mismo y por tanto el impedimento de llegar al objetivo establecido.

En cuanto a la precisión (ver Tabla 11), pudimos observar que en este caso fue el mejor de los 4 planificadores, aunque remarcando que en esta prueba todos los planificadores obtuvieron unos valores bastante similares y bastante buenos.

En lo referente a la distancia (ver Tablas 15 y 16), una vez más fue el primero, pero esta vez eso no era lo deseado, ya que fue el que más se acercó a los objetos durante las simulaciones. Esto causaba recálculos de rutas y debido a su alta velocidad frenazos bastante bruscos que provocaban el tambaleo del robot.

Con el algoritmo DWA el robot se mueve cerca del plan global, sin bucles, lo que explica estos acercamientos y conducción brusca (ver Figura 48), ya que el cálculo de ruta no es demasiado extenso. Esto podría ser útil si el robot se moviera por habitaciones pequeñas con objetos, pero para nuestro circuito no se tuvo en cuenta esa prueba por lo que no se puede demostrar al 100 %. También vimos que en la prueba de la elección de ruta este algoritmo, a diferencia de todos los demás, fue incapaz de planificar bien la ruta en ninguno de sus modos de conducción (ver Figura 47). Pero, al igual que TR, en la prueba del zig-zag sí que fue capaz de realizarla con gran facilidad y sin tener que realizar recálclos de ningún tipo (ver Figura 49).

Puntuaciones para DWA			
Pruebas	Conducción agresiva	Conducción normal	Conducción suave
Precisión	3	4	4
Tiempo	3	4	4
Velocidad	3	3	4
Conducción	3	3	2
Trayectoria	3	2	2
Distancia	3	1	1
Resultado final	3	2.83	2.83

Tabla 18: Resultados para el planificador DWA.

Siguiendo con el resumen vamos a ver el comportamiento del planificador EBAND. Empezaremos por el tiempo (ver Tabla 8, en el que EBAND obtuvo el tercer puesto por detrás de los dos planificadores vistos anteriormente. Aunque este planificador ha sido un poco especial, ya que la posición de su velocidad ha ido variando mucho en función del modo que se estaba evaluando, pasando de ser el más rápido en el modo agresivo (ver Figura 38) al más lento en el modo suave (ver Figura 40). Si vemos en primer lugar el modo agresivo podemos encontrar un planificador bastante errático, con unas aceleraciones altísimas (ver Figura 44), provocadas por una conducción bastante alocada; es más, si nos fijamos en la Figura 47, en este modo podemos ver claramente la irregularidad de este planificador, ya que la línea que se describe en la gráfica es bastante ruidosa lo que se traduce en cambios de dirección bruscos y aleatorios durante toda la simulación. Pero como ya hemos comentado cuando cambiamos de modo, el planificador cambia completamente, convirtiendo la navegación irregular en una navegación más segura y suave; viendo las Figuras 48 y 49 podemos apreciar claramente el cambio. Además, en estos modos podemos ver como el planificador calcula otras rutas más eficientes y cómodas para el robot, lo que hace mejorar infinitamente las gráficas de velocidad y aceleración respecto a la primera simulación.

Con lo que a la precisión respecta (ver Tabla 12), vemos que este planificador es el que mejores resultados ha obtenido después de DWA. Una vez más hay que recalcar que esta clasificación es totalmente numérica, ya que los resultados son muy parejos y los datos bastante similares por lo que la diferencia no es significativa.

Por último, EBAND ocupa el segundo lugar en lo que a las distancias respecta. Este planificador, siguiendo la tendencia de cambio observada anteriormente, tiene un comportamiento similar en distancia a obstáculos, pero de una forma bastante más sutil puesto que los cambios no son tan bruscos

y el acercamiento sigue siendo de los más pequeños de los 4 (ver Tablas 15 y 16).

En resumen, este planificador es bastante bueno, ya que tiene una gran versatilidad de uso, como hemos podido ver. EBAND es capaz de cambiar su comportamiento totalmente y aún así seguir encontrando las soluciones al problema planteado, lo que le otorga una gran robustez frente a cambios, pudiéndose adaptar a una gran variedad de funcionalidades diferentes.

Puntuaciones para EBAND			
Pruebas	Conducción agresiva	Conducción normal	Conducción suave
Precisión	4	2	3
Tiempo	4	1	1
Velocidad	4	1	1
Conducción	1	2	3
Trayectoria	1	3	3
Distancia	1	2	3
Resultado final	2.5	1.83	2.33

Tabla 19: Resultados para el planificador EBAND.

En último lugar, vamos a hablar sobre el comportamiento del planificador TEB. Este planificador no destacó por su velocidad, puesto que fue el más lento de los 4. Por lo general tardó bastante más que el resto de planificadores y eso que con este planificador el robot no tuvo que detenerse y recalcular rutas, por lo que se hace más evidente la lentitud de este planificador (ver Tabla 9). Relacionado con esta lentitud pudimos ver unas gráficas de velocidades (ver Figuras 35-40) y aceleraciones (ver Figuras 41-46) bastante bajas. Además de valores bajos también se observaron cambios de velocidad suaves, es decir, no había picos de cambios bruscos, por lo que la seguridad del robot nunca se vio afectada a la hora de la conducción con este planificar, más bien todo lo contrario. En cuanto a la precisión de este planificador, vemos que los números indican que fue el peor de los 4 planificadores, obteniendo unos porcentajes unas décimas más elevados que el resto (ver Tabla 13).

Por otro lado, en cuanto a las distancias (ver Tablas 15 y 16) este planificador fue el que más mantuvo la distancia respecto a los objetos y paredes del circuito, el cálculo de su ruta siempre tendía a circular por medio del carril, lo que provocaba que la conducción fuera tan suave y continuada, sin necesidad de recálculos.

Siguiendo con lo comentado, si vemos las Figuras 47-49, sin lugar a dudas fue el planificador que más sorprendió, ya que, como se ha comentado, siempre intentaba despegarse de la pared y circular por una ruta cómoda para el robot. En consecuencia, la línea dibujaba durante gran parte del circuito se aleja de las otras 3 rutas, las cuales son bastante similares.

Otro punto donde sorprendió el planificador TEB y que ya hemos comentado, fue en la prueba donde el planificador debía elegir qué camino escoger, y este fue el único que eligió un camino libre de obstáculos aunque era un camino más largo. Esta elección permitió al planificador coger la curva sin ninguna dificultad y luego encarar la recta posterior de forma más cómoda y centrada en el carril de circulación, y con ello, luego consiguió hacer una curva en forma de U con más facilidad que los demás. Luego, en la última prueba, también se volvió a diferenciar del resto de planificadores, ya que éste calculó una ruta más larga, otra vez, pero gracias a eso consiguió evitar los obstáculos de una forma simple y eficaz, haciendo curvas de amplio diámetro que fueran cómodas para el robot y de esa forma conseguir el objetivo lo más sencillamente posible.

Puntuaciones para TEB			
Pruebas	Conducción agresiva	Conducción normal	Conducción suave
Precisión	2	3	2
Tiempo	1	2	2
Velocidad	1	2	2
Conducción	4	4	4
Trayectoria	4	4	4
Distancia	2	4	4
Resultado final	2.33	3.167	3

Tabla 20: Resultados para el planificador TEB.

En la Tabla 21 podemos ver las puntuaciones obtenidas tras realizar las distintas pruebas realizadas a cada uno de estos planificadores. Estas puntuaciones son totalmente objetivas, ya que estás extraídas de los datos que se han obtenido. Estas puntuaciones nos ayudarán a elegir al mejor planificador, pero tenemos otros factores que influyen en la elección final.

Resultados de los planificadores			
Planificadores	Conducción agresiva	Conducción normal	Conducción suave
TR	2.167	2.167	1.83
DWA	3	2.83	2.83
EBAND	2.5	1.83	2.33
TEB	2.33	3.167	3

Tabla 21: Resultados de los planificadores.

Una vez comentados todos los planificadores, podemos decir que para este circuito de pruebas el planificador que mejor resolvió cada una de las pruebas de una forma sencilla y eficaz fue TEB. Aunque, hay que añadir que después de ver el comportamiento plástico de EBAND podemos decir que dicho planificador puede que tenga un mayor rango de operación y que será el que mejor se adapte a otros circuitos de prueba distintos. Por lo que respecta a TR y DWA, podemos concluir que estos planificadores son una buena opción para realizar simulaciones pequeñas y rápidas, pero si queremos exigirle al robot una funcionalidad y complicar las simulaciones un poco estos planificadores empezarán a quedarse cortos para el proyecto.

6. Conclusiones y Líneas futuras

6.1. Conclusiones

Tras finalizar el proyecto podemos ver si los objetivos que teníamos en la Sección 1.2 se han cumplido o por el contrario no hemos podido ser capaces de alcanzarlos. El primer objetivo era la construcción de un entorno capaz de poder comparar diferentes algoritmos de planificación local. Realizando esta tarea hemos podido aprender sobre un sistema operativo como ROS, el cual se está convirtiendo en un referente en la industria, en primer lugar porque es un software libre y en segundo por la gran comunidad que hay detrás de él, ya que gracias a ella cualquier usuario novel es capaz de aprender y ejecutar proyectos de navegación en robótica. Los otros objetivos también se han cumplido correctamente, ya que se ha podido realizar la comparativa de algoritmos de planificación local, lo cual era el objetivo principal del trabajo.

Por lo que respecta a la comparativa, hemos podido ver diferentes planificadores, con diferentes comportamientos. En el circuito de pruebas diseñado, hemos podido poner a prueba estos algoritmos, y si vemos los datos extraídos podemos ver que hay una gran diferencia entre TR y DWA, y EBAND y TEB. Este resultado concuerda con los algoritmos, ya que el planificador DWA es una modificación de TR, el cual es un planificador bastante simple y es por eso que se utiliza para realizar navegaciones orientadas a pruebas académicas, fuera de este margen es un planificador limitado. Es por esto que tenemos DWA el cual mejora un poco los resultados con TR.

Por lo que respecta a EBND y TEB, están basados en un algoritmo diferente a los otros 2, por ello se aprecia una brecha en el comportamiento. Pero a diferencia de la otra pareja, esta pareja sí se diferencia más entre ellos dos. En este caso es EBAND el algoritmo original y TEB una adaptación de éste. La introducción de la temporización en el algoritmo realiza una mejora bastante evidente en el comportamiento. El planificador TEB ha sido el planificador que más ha sorprendido en las pruebas, puesto que el comportamiento directamente era diferente al resto. En el caso de EBAND, al comportamiento era mejor que DWA y TR pero las lógicas de navegación que realizaba eran bastante similares.

Por todo esto es por lo que el planificador TEB ha sido el planificador que mejor se ha comportado en este trabajo. Aunque en su contra juega la complejidad que presenta. En primer lugar, porque como vimos en la Sección 2.3.4 hay una gran cantidad de variables que controlan el algoritmo, luego su puesta en marcha es especialmente complicada. Además, en el cambio de comportamiento era bastante sensible a modificaciones, ya que al haber tantos parámetros, realizar cambios en alguno de ellos descompensaba de forma exagerada al robot. Pero teniendo los parámetros óptimos de conducción el algoritmo realiza una planificación bastante buena. En cambio, en el planificador EBAND al no tener la temporalidad el número de parámetros es bastante menor y esto lo convierte en un planificador bastante robusto frente a cambios.

En conclusión, a nivel personal ha sido bastante enriquecedor porque he aumentado mucho el conocimiento en lógicas de navegación en robots móviles. Este aprendizaje puede servir para una inversión posterior en el mundo laboral, ya que durante el máster no se explica nada parecido y puedes especializarte en un campo con una competencia laboral bastante baja, actualmente.

6.2. Líneas futuras

En este apartado vamos a comentar algunas de las posibilidades que tiene este proyecto, ya que, a pesar de ser un trabajo bastante completo sobre la comparativa de los planificadores locales que ROS nos ofrece tiene algunas líneas con las que completar y dar importancia a todo este trabajo realizado.

- ✓ Una de las líneas donde poder ampliar el trabajo, sería añadir planificadores a los que ya están comparados, con el objetivo de hacer el estudio más completo y robusto, además de tener unos datos más amplios de los que podemos tener a día de hoy. También se podrían añadir algunos algoritmos más, incluso algoritmos de desarrollo propio para agrandar la comparativa y poder llegar a hacer una publicación más importante y con más reconocimiento.

  Este proyecto se podría utilizar para que otros usuarios de la comunidad robótica, si desarrollan un algoritmo de estas características, puedan hacer pruebas de su algoritmo y comparar con este estudio la bondad de los algoritmos que estos usuarios han desarrollado. Para ello, se ha creado un repositorio en GitHub donde podemos encontrar todos los ficheros necesarios para replicar el proyecto y también añadir estos algoritmos de los usuarios [14].

- ✓ Este estudio se podría preparar para realizar una publicación en una revista científica o en un congreso relacionado con el tema.

  Para ello, se debería adaptar todo el trabajo y resumirlo en un artículo científico que sea publicable. Además, podrían realizarse más pruebas para confirmar los datos que hemos extraído en el proyecto y así aumentar la robustez y fiabilidad de los resultados que hemos concluidos.

- ✓ Otra línea de futuro podría ser realizar más pruebas para ampliar el banco de pruebas que tenemos actualmente y por tanto agrandar la comparativa. Una de las pruebas interesantes que se podrían hacer es comparar el comportamiento de estos planificadores locales con objetos dinámicos, es decir, objetos en movimiento como pudieran ser personas, animales, otros robots... También se podrían realizar pruebas con un robot y un entorno real, aunque los valores no deben cambiar demasiado con respecto a las simulaciones sería interesante certificar los resultados obtenidos con estas pruebas reales.

A modo de conclusión solo decir que el proyecto tiene mucho potencial y camino por delante. Si se amplía alguna de las líneas descritas podremos ayudar de forma más extensa a otros usuarios de robótica a seleccionar qué planificador es mejor en función de la aplicación para la que se quiera utilizar el robot.

A. Anexos

A.1. Código launch

```
<?xml version="1.0"?>
<launch>

  <include file="$(find rosbot_description)/launch/rosbot_rviz.launch" >
  </include>

  <master auto= "start"/>

  <!-- Map server -->
  <arg name="map_file" default="$(find rosbot_navigation)/maps/test_map.yaml"/>

  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!-- Localization -->
  <node pkg="amcl" type="amcl" name="amcl" output="screen">
    <remap from="scan" to="/scan"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="odom_model_type" value="diff-corrected"/>
    <param name="base_frame_id" value="base_link"/>
    <param name="update_min_d" value="0.1"/>
    <param name="update_min_a" value="0.2"/>
    <param name="min_particles" value="500"/>
  </node>

  <!-- Move base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <!--    <rosparam file="$(find rosbot_navigation)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />

    <rosparam file="$(find rosbot_navigation)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />

    <rosparam file=
    "$(find rosbot_navigation)/config/local_costmap_params.yaml"
    command="load" />

    <rosparam file=
    "$(find rosbot_navigation)/config/global_costmap_params.yaml"
    command="load" /> -->
    <rosparam file=
    "$(find teb_local_planner_tutorials)/cfg/diff_drive/costmap_common_params.yaml" command="load" ns="global_costmap" />

    <rosparam file=
    "$(find teb_local_planner_tutorials)/cfg/diff_drive/costmap_common_params.yaml" command="load" ns="local_costmap" />

    <rosparam file=
    "$(find teb_local_planner_tutorials)/cfg/diff_drive/
```

```
    local_costmap_params.yaml" command="load" />
45
46      <rosparam file=
47      "$(find teb_local_planner_tutorials)/cfg/diff_drive/
        global_costmap_params.yaml" command="load" />
48
49
50      <!-- <rosparam file="$(find teb_local_planner_tutorials)/cfg/
        diff_drive/teb_local_planner_params.yaml" command="load" /> -->
51
52      <rosparam file="$(find rosbot_navigation)/config/teb/
        suave_teb_params.yaml" command="load" />
53
54     <!--
55      <rosparam file="$(find rosbot_navigation)/config/teb/
        normal_teb_params.yaml" command="load" />
56
57      <rosparam file="$(find rosbot_navigation)/config/teb/
        agresivo_teb_params.yaml" command="load" />
58      -->
59
60
61      <remap from="cmd_vel" to="cmd_vel"/>
62      <remap from="odom" to="odom"/>
63      <remap from="scan" to="/scan"/>
64
65      <param name="base_local_planner" value= "teb_local_planner/
        TebLocalPlannerROS"/>
66
67      <param name="base_global_planner" value="global_planner/
        GlobalPlanner" />
68
69    </node>
70  </launch>
```

A.2. Código yaml

```yaml
EBandPlannerROS:
 eband_significant_force_lower_bound: 0.15
 costmap_weight: 20
 max_vel_lin: 1
 max_vel_th: 1.5
 min_vel_lin: 0.2
 min_vel_th: 0.5
 max_acceleration: 0.4
 max_translational_acceleration: 0.4
 max_rotational_acceleration: 1
 Ctrl_Rate: 75.0
 bubble_velocity_multiplier: 1.0
 xy_goal_tolerance: 0.2
 yaw_goal_tolerance: 0.1
```

```yaml
DWAPlannerROS:
   acc_lim_x: 2.5
   acc_lim_theta: 1.0
   max_vel_trans: 0.55
   max_vel_x: 0.55
   min_vel_x: -0.1
   max_vel_theta: 2.5
   sim_granularity: 0.025
   angular_sim_granularity: 0.025
   sim_time: 5.0
   stop_time_buffer: 0.2

   xy_goal_tolerance: 0.2
   yaw_goal_tolerance: 0.1
```

A.3. Código bash

```
#!/bin/bash

#POSICION INICIAL.

if [[ "$1" -eq 0 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 0.013
    y: -0.002
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.7
    w: 0.713"

###PRIMER TARGET.

elif [[ "$1" -eq 1 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: -8.56
    y: -9.56
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.77
    w: 0.63"

###SEGUNDO TARGET.

elif [[ "$1" -eq 2 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 4.31
    y: 5.5
    z: 0.0
```

```
  orientation:
    x: 0.0
    y: 0.0
    z: 0.12
    w: 1.0"

##TERCER TARGET.

elif [[ "$1" -eq 3 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 2.02
    y: -8.68
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.83
    w: 0.55"

###CUARTO TARGET.

elif [[ "$1" -eq 4 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: -5.47
    y: -0.41
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 1.0
    w: 0.53"

###QUINTO TARGET.

elif [[ "$1" -eq 5 ]]; then
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: -5.73
```

```
    y: -8.91
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -0.7
    w: 0.716"
fi
```

A.4. Código MatLab

```
clear all
close all
clc
bag(1) = rosbag ('agresivo/Target1_DWA_A.bag');
bag(2) = rosbag ('agresivo/Target2_DWA_A.bag');
bag(3) = rosbag ('agresivo/Target3_DWA_A.bag');
bag(4) = rosbag ('agresivo/Target4_DWA_A.bag');
bag(5) = rosbag ('agresivo/Target5_DWA_A.bag');

bag(6) = rosbag ('normal/Target1_DWA_N.bag');
bag(7) = rosbag ('normal/Target2_DWA_N.bag');
bag(8) = rosbag ('normal/Target3_DWA_N.bag');
bag(9) = rosbag ('normal/Target4_DWA_N.bag');
bag(10) = rosbag ('normal/Target5_DWA_N.bag');

bag(11) = rosbag ('suave/Target1_DWA_S.bag');
bag(12) = rosbag ('suave/Target2_DWA_S.bag');
bag(13) = rosbag ('suave/Target3_DWA_S.bag');
bag(14) = rosbag ('suave/Target4_DWA_S.bag');
bag(15) = rosbag ('suave/Target5_DWA_S.bag');

%% take the time in a file
for i=1:15
    topic_start_goal = select(bag(i),'Topic','/move_base_simple/goal')
   ;
    topic_end_goal = select(bag(i),'Topic','/move_base/result');
    time (i) = abs(topic_start_goal.StartTime - topic_end_goal.EndTime
   );
end
    total_time(1) = sum(time(1:5));
    total_time(2) = sum(time(6:10));
    total_time(3) = sum(time(11:15));

    mean_time(1) = mean(time(1:5));
    mean_time(2) = mean(time(6:10));
    mean_time(3) = mean(time(11:15));

save results_DWA.mat time total_time mean_time -append;

%% Linear Velocity
for i=1:15
    topic_cmd_vel = select(bag(i),'Topic','/cmd_vel');
    vel_x_lin = timeseries(topic_cmd_vel, 'Linear.X');
    max_vel_lin = 0;
    min_vel_lin = 100;
    for j=1:length(vel_x_lin.Data)
        if(vel_x_lin.Data(j) > max_vel_lin)
            max_vel_lin = vel_x_lin.Data(j);
        end
        if (vel_x_lin.Data(j)< min_vel_lin)&&(vel_x_lin.Data(j)>0)
            min_vel_lin=vel_x_lin.Data(j);
        end
    end
    mean_vel_lin(i)= mean(vel_x_lin.Data);
    min_arr_vel_lin(i) = min_vel_lin;
    max_arr_vel_lin(i) = max_vel_lin;
```

```
end

save results_DWA.mat mean_vel_lin min_arr_vel_lin max_arr_vel_lin -
   append;

%% Angular Velocity
for i=1:15
    topic_cmd_vel = select(bag(i),'Topic','/cmd_vel');
    vel_z_ang = timeseries(topic_cmd_vel, 'Angular.Z');
    min_vel_ang=100;
    max_arr_vel_ang(i) = max(abs(vel_z_ang.Data));
    for j=1:length(vel_z_ang.Data)
        if (abs(vel_z_ang.Data(j))< min_vel_ang)...
        && (abs(vel_z_ang.Data(j))>0)
            min_vel_ang=abs(vel_z_ang.Data(j));
        end
    end
    mean_vel_ang(i)= mean(vel_z_ang.Data);
    min_arr_vel_ang(i) = min_vel_ang;
end
save results_DWA.mat mean_vel_ang min_arr_vel_ang max_arr_vel_ang -
   append;

%% Presition
t1=[-8.56 -9.56 0 0 0 0.77 0.63];
t2=[4.31 5.5 0 0 0 0.12 0.99];
t3=[2.02 -8.68 0 0 0 0.83 0.56];
t4=[-5.47 -0.41 0 0 0 1 0.053];
t5=[-5.73 -8.91 0 0 0 -0.7 0.716];

teoric_position=[t1;t2;t3;t4;t5;t1;t2;t3;t4;t5;t1;t2;t3;t4;t5];

for i=1:15
      topic_result= select(bag(i),'Topic','/amcl_pose');

      real_position = topic_result.timeseries.Data

      (topic_result.NumMessages,4:10);

      Error_position(i,:)=abs(teoric_position(i,:)-real_position);

      Error_percentage(i)=
      norm(Error_position(i,:))/norm(teoric_position(i,:)) *100;
end
   Error_porcentaje_medio(1)= mean(Error_percentage(1:5));
   Error_porcentaje_medio(2)= mean(Error_percentage(6:10));
   Error_porcentaje_medio(3)= mean(Error_percentage(11:15));
save results_DWA.mat Error_position Error_percentage
Error_porcentaje_medio -append;

%%
clear all
close all

%% Linear Acceleration
   for i=1:15
    topic_acc = select(bag(i),'Topic','/imu');
    acc_x = timeseries(topic_acc, 'LinearAcceleration.X');
```

```
    acc_y = timeseries(topic_acc, 'LinearAcceleration.Y');
    acc_z = timeseries(topic_acc, 'LinearAcceleration.Z');

    acc_to_norm (1,:) = acc_x.Data;
    acc_to_norm (2,:) = acc_y.Data;
    acc_to_norm (3,:) = acc_z.Data;
    for j=1:acc_z.Length
        acc_norm(j) = sqrt(acc_to_norm(1,j)^2 + ...
        acc_to_norm(2,j)^2 + acc_to_norm(3,j)^2);
    end
    if(i==1) x1=acc_norm; end
    if(i==2) x2=acc_norm; end
    if(i==3) x3=acc_norm; end
    if(i==4) x4=acc_norm; end
    if(i==5) x5=acc_norm; end
    if(i==6) x6=acc_norm; end
    if(i==7) x7=acc_norm; end
    if(i==8) x8=acc_norm; end
    if(i==9) x9=acc_norm; end
    if(i==10) x10=acc_norm; end
    if(i==11) x11=acc_norm; end
    if(i==12) x12=acc_norm; end
    if(i==13) x13=acc_norm; end
    if(i==14) x14=acc_norm; end
    if(i==15) x15=acc_norm; end

    mean_acc(i) = mean(acc_norm);
    min_acc(i) = min(acc_norm);
    max_acc(i) = max(acc_norm);
    clear acc_to_norm
    clear acc_norm
   end
    acc_A =[x1 x2 x3 x4 x5];
    acc_N =[x6 x7 x8 x9 x10];
    acc_S =[x11 x12 x13 x14 x15];

   i=1; fin=0; j=0;
while fin == 0
    if (j<length(acc_A)) && (abs(j-length(acc_A)) >= 100)
    save_acc_A(i) = mean(acc_A(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end

i=1; fin=0; j=0;
while fin == 0
    if (j< length(acc_N)) && (abs(j-length(acc_N)) >= 100)
    save_acc_N(i) = mean(acc_N(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end

j=0; i=1; fin=0;
while  fin == 0
```

```
    if (j<= length(acc_S))&& (abs(j-length(acc_S)) >= 100)
    save_acc_S(i) = mean(acc_S(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end

save results_DWA.mat save_acc_A save_acc_N save_acc_S -append;
save results_DWA.mat mean_acc min_acc max_acc -append;



%% Save velocities
for i=1:5
    topic_cmd_vel = select(bag(i),'Topic','/cmd_vel');
    vel_x_lin_A(i) = timeseries(topic_cmd_vel, 'Linear.X');
end
    vel_A = [vel_x_lin_A(1).Data' vel_x_lin_A(2).Data'...
    vel_x_lin_A(3).Data' vel_x_lin_A(4).Data' vel_x_lin_A(5).Data'];
for i=6:10
    topic_cmd_vel = select(bag(i),'Topic','/cmd_vel');
    vel_x_lin_N(i-5) = timeseries(topic_cmd_vel, 'Linear.X');
end
vel_N = [vel_x_lin_N(1).Data' vel_x_lin_N(2).Data'...
vel_x_lin_N(3).Data' vel_x_lin_N(4).Data' vel_x_lin_N(5).Data'];
for i=11:15
    topic_cmd_vel = select(bag(i),'Topic','/cmd_vel');
    vel_x_lin_S(i-10) = timeseries(topic_cmd_vel, 'Linear.X');
end
vel_S = [vel_x_lin_S(1).Data' vel_x_lin_S(2).Data'...
vel_x_lin_S(3).Data' vel_x_lin_S(4).Data' vel_x_lin_S(5).Data'];

i=1; fin=0; j=0;
while fin == 0
    if (j<length(vel_A)) && (abs(j-length(vel_A)) >= 100)
    save_velocity_A(i) = mean(vel_A(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end
   save_velocity_A(27) = mean(vel_A(2600:2690));

i=1; fin=0; j=0;
while fin == 0
    if (j< length(vel_N)) && (abs(j-length(vel_N)) >= 100)
    save_velocity_N(i) = mean(vel_N(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end
   save_velocity_N(37) = mean(vel_N(3600:3672));

j=0; i=1; fin=0;
while  fin == 0
    if (j<= length(vel_S))&& (abs(j-length(vel_S)) >= 100)
```

```
    save_velocity_S(i) = mean(vel_S(j+1:j+100));
    else
    fin=1;
    end
    j=j+10; i=i+1;
end
save results_DWA.mat save_velocity_A...
save_velocity_N save_velocity_S -append;

%% Distance to a objects
for i=1:15
    bSel=select(bag(i),'Topic','/scan');
    msg=readMessages(bSel,'DataFormat','struct');
    size_msg=size(msg);
    for j=1:size_msg(1)
        min_val=10;
        max_val=0;
        for k=1:720
           if ( min_val > msg{j}.Ranges(k) )
                min_val = msg{j}.Ranges(k);
                pos_min = k;
           end
           if ( max_val < msg{j}.Ranges(k) )
               max_val = msg{j}.Ranges(k);
               pos_max = k;
           end
        end
        min_array(j) = min_val;
        max_array(j) = max_val;
        pos_min_array(j) = pos_min;
        pos_max_array(j) = pos_max;
        mean_array(j) = mean(msg{j}.Ranges);
    end

    for j=1:length(max_array)
        min_val=10;
        max_val=0;
        if ( min_val > min_array(j) )
             min_val = min_array(j);
             pos_min = pos_min_array(j);
        end
        if ( max_val < max_array(j) )
             max_val = max_array(j);
             pos_max = pos_max_array(j);
        end
    end
    min_values(i) = min_val;
    max_values(i) = max_val;
    pose_min_value(i) = pos_min;
    pose_max_value(i) = pos_max;
    mean_values(i) = mean(mean_array);
end

for i=1:15
    bSel=select(bag(i),'Topic','/amcl_pose');
    posex_matrix(i)=timeseries(bSel,'Pose.Pose.Position.X');
    posey_matrix(i)=timeseries(bSel,'Pose.Pose.Position.Y');
end
```

```
posexA= [posex_matrix(1).Data' posex_matrix(2).Data'...
posex_matrix(3).Data' posex_matrix(4).Data' posex_matrix(5).Data'];

poseyA= [posey_matrix(1).Data' posey_matrix(2).Data'...
posey_matrix(3).Data' posey_matrix(4).Data' posey_matrix(5).Data'];

posexN= [posex_matrix(6).Data' posex_matrix(7).Data'...
posex_matrix(8).Data' posex_matrix(9).Data' posex_matrix(10).Data'];

poseyN= [posey_matrix(6).Data' posey_matrix(7).Data'...
posey_matrix(8).Data' posey_matrix(9).Data' posey_matrix(10).Data'];

posexS= [posex_matrix(11).Data' posex_matrix(12).Data'...
posex_matrix(13).Data' posex_matrix(14).Data' posex_matrix(15).Data'];

poseyS= [posey_matrix(11).Data' posey_matrix(12).Data'...
posey_matrix(13).Data' posey_matrix(14).Data' posey_matrix(15).Data'];

save results_DWA.mat posexA poseyA posexN poseyN...
posexS poseyS min_values max_values pose_min_value...
pose_max_value mean_values -append;
```

Referencias

[1] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 341–346 vol.1, 1999.

[2] W. Chung and K. Iagnemma. *Wheeled Robots*, pages 575–594. Springer International Publishing, Cham, 2016.

[3] C. Connette, B. Marthi, and P. Khandelwal. EBAND local planner. `http://wiki.ros.org/eband_local_planner`. Último acceso: 26-08-2021.

[4] R. developers. Tutoriales ROS. `http://wiki.ros.org/navigation/Tutorials/RobotSetup`. Último acceso: 26-08-2021.

[5] T. Field, J. Leibs, J. Bowman, and D. Thomas. Rosbag ROS. `http://wiki.ros.org/rosbag`. Último acceso: 26-08-2021.

[6] T. Foote. Laser scan instruccions. `http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html`. Último acceso: 30-08-2021.

[7] O. S. R. Foundation. Gazebo simulator. `http://gazebosim.org/`. Último acceso: 30-08-2021.

[8] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. 2008.

[9] B. Gassend. Dynamic reconfigure ROS. `http://wiki.ros.org/dynamic_reconfigure`. Último acceso: 25-08-2021.

[10] B. Gerkey. Planning and control in unstructured terrain. 2008.

[11] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and localization with rfid technology. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 1, pages 1015–1020 Vol.1, 2004.

[12] D. Hershberger, D. Gossow, J. Faust, and W. Woodall. Rviz ROS. `http://wiki.ros.org/rviz`. Último acceso: 25-08-2021.

[13] W. Jeong and K. M. Lee. Cv-slam: a new ceiling vision-based slam technique. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3195–3200, 2005.

[14] P. Jordan. Project repository. `https://github.com/jtpau/TFM/tree/master`. Último acceso: 26-08-2021.

[15] L. E. Kavraki and S. M. LaValle. *Motion Planning*, pages 139–162. Springer International Publishing, Cham, 2016.

[16] A. Kelly. An intelligent predictive controller for autonomous vehicles. 1994.

[17] D. Kragic and K. Daniilidis. *3-D Vision for Navigation and Grasping*, pages 811–824. Springer International Publishing, Cham, 2016.

[18] E. Marder-Eppstein. DWA local planner. `http://wiki.ros.org/dwa_local_planner`. Último acceso: 26-08-2021.

[19] E. Marder-Eppstein and E. Perko. Trajectory local planner. `http://wiki.ros.org/base_local_planner`. Último acceso: 26-08-2021.

[20] F. Marek. ASR local planner. `http://wiki.ros.org/asr_ftc_local_planner`. Último acceso: 26-08-2021.

[21] MathWords. Readmessages matlab. `https://es.mathworks.com/help/ros/ref/readmessages.html`.

[22] MathWords. Rosbag matlab. `https://es.mathworks.com/help/ros/ref/rosbag.html`. Último acceso: 30-08-2021.

[23] J. Minguez, F. Lamiraux, and J.-P. Laumond. *Motion Planning and Obstacle Avoidance*, pages 1177–1202. Springer International Publishing, Cham, 2016.

[24] M. Mujahad, D. Fischer, B. Mertsching, and H. Jaddu. Closest gap based (cg) reactive obstacle avoidance navigation for highly cluttered environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1805–1812, 2010.

[25] E. Prassler, M. E. Munich, P. Pirjanian, and K. Kosuge. *Domestic Robotics*, pages 1729–1758. Springer International Publishing, Cham, 2016.

[26] C. Rösmann. TEB local planner. `http://wiki.ros.org/teb_local_planner`. Último acceso: 26-08-2021.

[27] H. sp. Manual ROSbot 2.0. `https://husarion.com/manuals/rosbot/`.

[28] K. Wojciechowski. rosbot_descripton package. `https://github.com/husarion/rosbot_description/tree/master`. Último acceso: 12-09-2021.