Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Modelado, simulación y aplicación de técnicas de navegación autónoma en un automóvil

Autor: Sergio León Doncel

Tutor: Francisco Rodríguez Rubio

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Modelado, simulación y aplicación de técnicas de navegación autónoma en un automóvil

Autor:
Sergio León Doncel

Tutor:
Francisco Rodríguez Rubio
Catedrático de Universidad

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Modelado, simulación y aplicación de técnicas de navegación autónoma en un automóvil

Autor: Sergio León Doncel

Tutor: Francisco Rodríguez Rubio

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha

Agradecimientos

"No hay enseñanza posible sin la bendita amistad, que es el mejor conductor de ideas entre hombre y hombre"

- Benito Pérez Galdós -

En primer lugar, me gustaría agradecer a mi tutor Francisco por la oportunidad brindada que da pie a este TFG y su apoyo durante su desarrollo. Por otro lado, a mis amigos, tanto aquellos que siempre me han estado apoyando en mi vida como aquellos nuevos que han surgido gracias a este grado, por hacer la travesía más agradable, con especial mención a Álvaro García Lora, de quién debo parte de mis logros. A todos los profesores que me han acompañado en mi trayectoria académica, desde los inicios hasta mi etapa actual, que me han permitido construir una base sólida desde la que construir todo. A Richard Haes Ellis y Rafael Rey Arcenegui por transmitirme sus conocimientos a nivel profesional en el ámbito software. A mi familia por su apoyo continuo en el camino. Por último, me gustaría dedicar este trabajo a mis seres queridos fallecidos, Salvador DC, Jaime PR y Sonia JR.

Sergio León Doncel

Sevilla, 2023

Resumen

Este proyecto nace a raíz de un grupo de trabajo coordinado con una meta en común, el desarrollo de una arquitectura software que envuelva la funcionalidad de conducción autónoma para el control de vehículos.

Dentro de este marco común, mi papel consiste en la creación de un modelo digital para un automóvil, que permita realizar simulaciones lo más fieles a la realidad posible, donde analizar el comportamiento de las técnicas y algoritmos desarrollados para la conducción autónoma.

Para ello, se parte de una etapa inicial de diseño de un mecanismo simple para modelar un vehículo. Haciendo uso del formato URDF, se realiza la descripción de los enlaces y articulaciones necesarios, incluyendo tanto la cinemática como la dinámica, así como la representación visual y el modelo de colisión. Inclusive se utilizan algunos plugins propios de Gazebo para la inclusión de distintos sensores.

Tras ello, se continua con la implementación y sintonización de un sistema de control con el uso de controladores PIDs para actuar sobre las transmisiones de las articulaciones.

Partiendo de lo anterior, y gracias al framework de ROS, se dota al sistema con los servicios estándar de un sistema operativo en tiempo real, donde programar y ejecutar la lógica necesaria. Aquí comienza la fase de desarrollo software, integrando diferentes paquetes de la comunidad junto a mis propias soluciones, tanto a bajo nivel como en la capa de interfaz de usuario.

El alcance final del proyecto es dotar al vehículo de capacidad de localización y mapeado simultáneo, así como la planificación y seguimiento de trayectorias dado un objetivo.

Abstract

This project was born as a result of a coordinated work group with a common goal, the development of a software architecture that involves autonomous driving functionality for vehicle.

Within this common framework, my role is the creation of a digital model for a car, which allows simulations as close to reality as possible, where to analyze the behavior of the techniques and algorithms developed for autonomous driving.

To do so, an initial design stage of a simple mechanism to model a vehicle starts. Using the URDF format, the description of the necessary links and joints is made, including both kinematics and dynamics, as well as the visual representation and the collision model. Even some of Gazebo's own plugins are used for the inclusion of different sensors.

After that, work is continued with the implementation and tuning of a control system with the use of PID controllers to act on the joint transmissions.

Based on the above, and thanks to the ROS framework, the system is provided with the standard services of a real-time operating system, where the necessary logic is programmed and executed. Here begins the software development phase, integrating different packages from the community along with my own solutions, both at low level and in the user interface layer.

The final scope of the project is to provide the vehicle with simultaneous localization and mapping capabilities, as well as trajectory planning and tracking given a target.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

NOTACIÓN

ROS	Robot Operative System
SLAM	Simultaneous Localization and mapping
GICP	Generalized-Iterative Closest Point
LiDAR	Laser Imaging Detection and Ranging
URDF	Unified Robot Description Format
SDF	Simulation Description Format
XML	Extensible Markup Language
ODE	Open Dynamics Engine
OGRE	Object-Oriented Graphics Rendering Engine
HMI	Human Machine Interface
IMU	Inertial Motion Unit
GPS	Global Positioning System
SAE	Society of Automotion Engineers

1 INTRODUCCIÓN

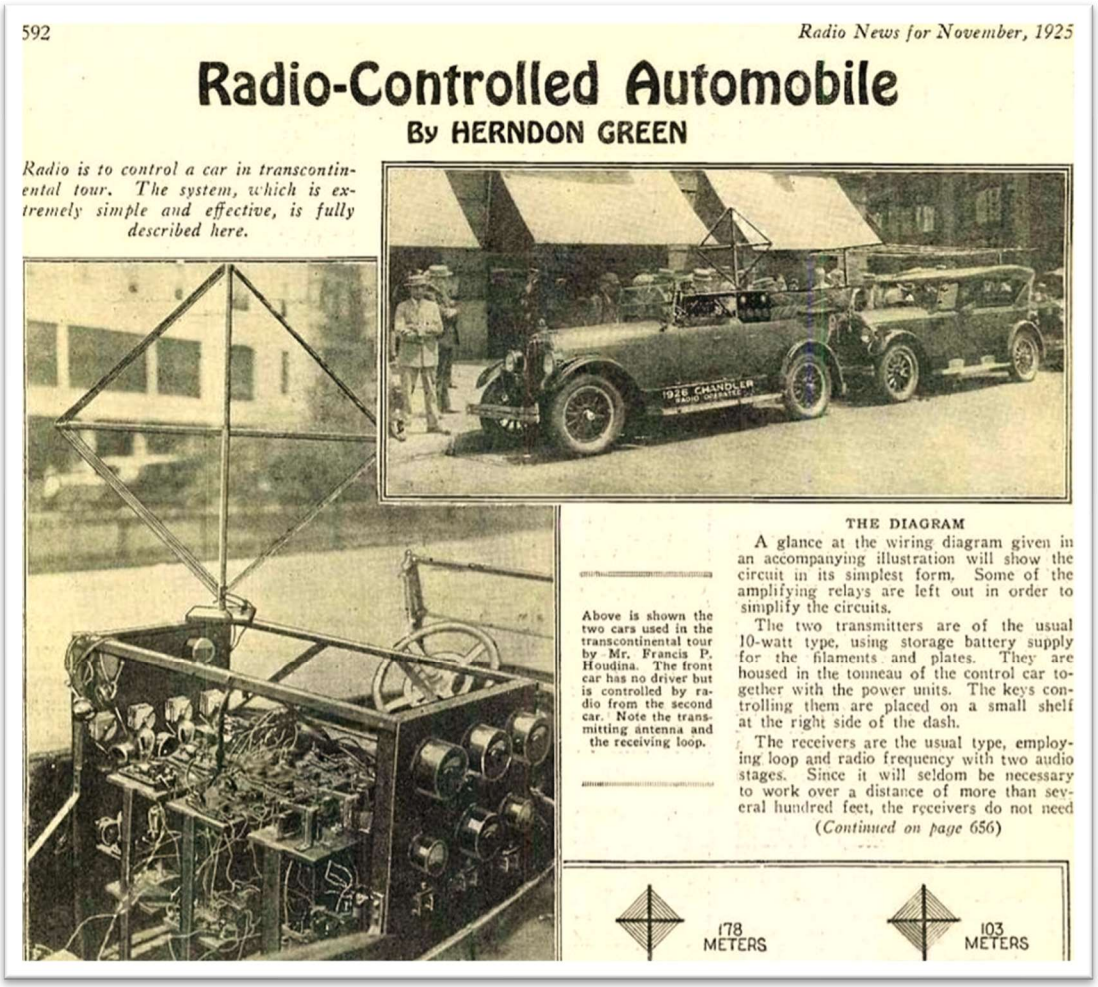
"La 'magia' de un hombre es la ingeniería de otro.
'Sobrenatural' es una palabra nula"

- Robert A. Heinlein -

1.1 Evolución histórica

La historia de la conducción autónoma se remonta un siglo atrás, cuando los ingenieros comenzaron a experimentar con la idea de crear vehículos que pudieran moverse sin la necesidad de un conductor humano.

En la década de 1920, un ingeniero eléctrico estadounidense, conocido como Francis P Houdina, adaptó un 1926 Chandler para dotarlo con un sistema radiocontrol que permitiese manejarlo a distancia, renombrándolo como American Wonder [1]. Posteriormente, hizo una demostración pública recorriendo en torno a 19 kilómetros por las calles de Manhattan que, pese a terminar su travesía con un accidente, causó un gran revuelo y sembró la semilla del interés.



592 *Radio News for November, 1925*

Radio-Controlled Automobile

By HERNDON GREEN

Radio is to control a car in transcontinental tour. The system, which is extremely simple and effective, is fully described here.

1926 CHANDLER

Above is shown the two cars used in the transcontinental tour by Mr. Francis P. Houdina. The front car has no driver but is controlled by radio from the second car. Note the transmitting antenna and the receiving loop.

THE DIAGRAM

A glance at the wiring diagram given in an accompanying illustration will show the circuit in its simplest form. Some of the amplifying relays are left out in order to simplify the circuits.

The two transmitters are of the usual 10-watt type, using storage battery supply for the filaments and plates. They are housed in the tonneau of the control car together with the power units. The keys controlling them are placed on a small shelf at the right side of the dash.

The receivers are the usual type, employing loop and radio frequency with two audio stages. Since it will seldom be necessary to work over a distance of more than several hundred feet, the receivers do not need

(*Continued on page 656*)

178 METERS

103 METERS

Figura 1–1 Portada de periódico informando hito histórico [2]

Ochenta años más tarde, apareció el considerado padre del vehículo autónomo, Ernst Dickmanns. Este investigador alemán logró en 1987 el primer hito histórico de la conducción autónoma tal y como la conocemos hoy día. Gracias a una inversión de la Comisión Europea en el proyecto EUREKA Promethues [3], convirtió una furgoneta en un vehículo guiado por visión sacádica por computador, y logró viajar sin conductor por una autovía en ausencia de tráfico, alcanzando velocidades de hasta 100 km/h.

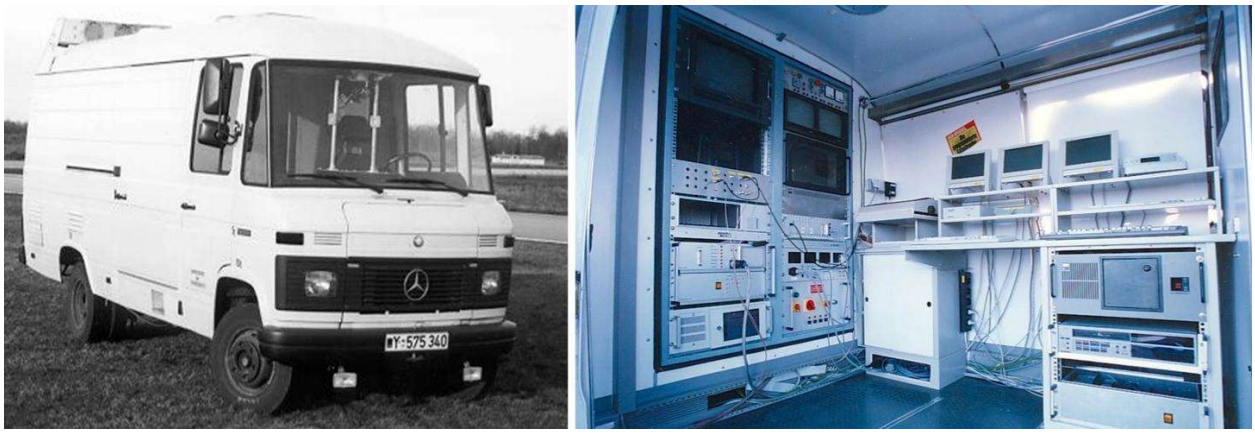


Figura 1–2 Furgoneta Mercedes-Benz 508D dotada de ordenador a bordo para visión por computador [4]

Figura 1–3 Cabina de furgoneta adaptada para el experimento [5]

Poco después, en 1994, un Mercedes W140 S500 recorrió más de 1000 kilómetros por una carretera de París, adelantando otros coches y alcanzando los 130 km/h. Un año después, el récord fue batido por el mismo modelo de coche, robotizado para recorrer 1600 km en un viaje de ida y vuelta entre Múnich y Copenhague [6], llegando a circular a 180 km/h por la Autobahn sin intervención humana.

Figura 1–4 Berlina Mercedes modificada para conducción autónoma en carretera [5] [7]

Esta serie de experimentos terminaron despertando el interés de las empresas tecnológicas, con precursores como Google y Tesla en busca de un modelo comercial, y llegando hoy día a ser un estándar en la rama de investigación de toda marca automovilística.

A partir de entonces, la tecnología de conducción autónoma ha seguido avanzando a pasos agigantados, y actualmente se están realizando pruebas en carreteras y ciudades de todo el mundo. Se espera que, en los próximos años, esta clase de vehículos se convierta en una realidad cada vez más común en nuestras vidas.

1.2 Motivación

A pesar de todo el esfuerzo dedicado a la concienciación en Seguridad Vial en la última década, cada año mueren en torno a 1.35 millones de personas en accidentes de tráfico [8], lo cual equivale a un fallecimiento cada 23 segundos. Además, si se atiende a las causas, el factor humano es responsable en el 90% de los casos [9].

En una sociedad avanzada como la nuestra, estas cifras no son aceptables. A pesar del gran descenso de muertes gracias a las medidas adoptadas: normativa en seguridad, concienciación de conductores… cada vez es más complicado mantener esa tendencia.

En este contexto, la conducción autónoma aparece como la solución ideal al problema. Si eliminamos a los humanos de la ecuación, conseguiríamos reducir notoriamente las cifras, y lo que es más, nos da pie a seguir mejorando la tecnología para alcanzar el tan ansiado número 0.

Inevitablemente, la gran mayoría de nosotros hemos conocido a alguien que ha fallecido viajando en coche. A pesar de que hoy en día esta tecnología aún necesite mejorarse para lograr el objetivo, considero que persigue una causa justa por la que invertir nuestros esfuerzos y recursos, para investigar y desarrollar en la materia.

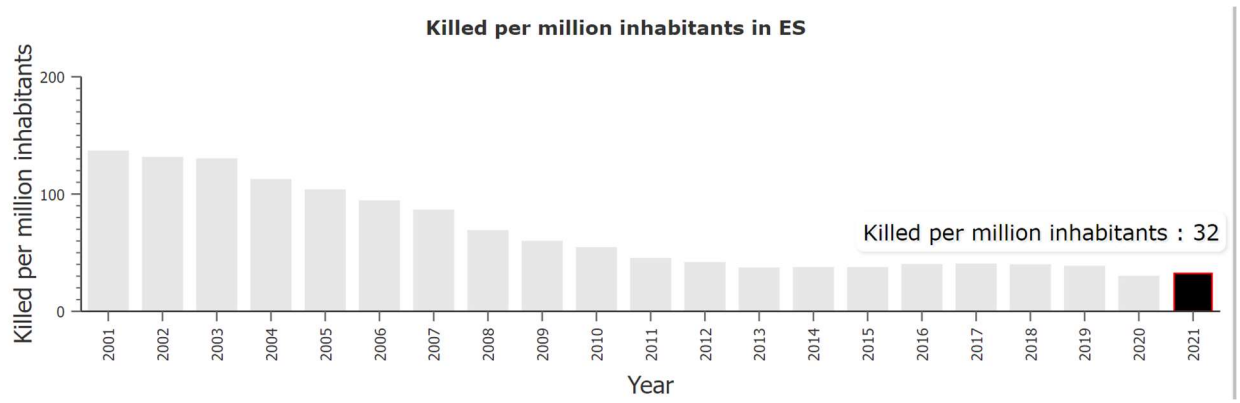


Figura 1–5 Descenso en la curva de muertes por accidentes de tráfico en España (Road Safety Atlas [10])

Sin embargo, esta pretensión de conseguir un vehículo completamente autónomo tiene muchos más beneficios para la sociedad, además del anteriormente mencionado [11]:

- Reducción de la congestión: El hecho de que los vehículos autónomos puedan estar conectados entre sí para compartir información, puede hacer posible evitar atascos o vías cortadas por accidente escogiendo otra ruta con antelación, usar señalización más optimizada con semáforos dinámicos en función de los datos que se reciban de los vehículos.
- Apertura del transporte privado para aquellos que no pueden conducir: Personas mayores, discapacitados y menores de edad. Todos ellos tienen en común la incapacidad para conducir, y por tanto la consecuente limitación de libertad. Es cierto que siempre tendrán la opción del transporte público o los taxis, pero todos sabemos los inconvenientes de estos. Sin embargo, gracias a la conducción autónoma podrán programar una ruta y disfrutar el viaje sin depender de terceros.
- Consumo más optimizado de energía: La preocupación medioambiental es creciente en los últimos años. La conducción autónoma permitiría aplicar una conducción "eco" optimizada para mejorar la eficiencia del combustible (la mayor capacidad de anticipación de estos permite evitar acelerones y frenazos bruscos), que unido a la reducción de atascos por la mejora de congestión, reduciría el combustible gastado de forma inútil y podría llevar hasta el 40% de mejora según las predicciones [12]. A pesar de ello, se debe tener en cuenta que la computación a bordo de los vehículos tiene un incremento de consumo añadido respecto a los vehículos normales, procesar la cantidad de datos ingentes que se requiere procesar tiene un precio.
- Aumento de tiempo libre: Casi todo el mundo debe gastar parte de su tiempo diario en el transporte a la escuela o trabajo, que a lo largo de todo el año es muy considerable. Normalmente, esto se ve como una pérdida de tiempo ya que el único objetivo es llegar de un punto a otro. Con la conducción autónoma, las personas tendrían la oportunidad de aprovechar ese tiempo dedicándolo a otras actividades.

2 ESTADO DEL ARTE

La investigación en esta área lleva afortunadamente dando frutos en los últimos años. Se pueden destacar dos fuentes principales: [13]

- "**DARPA Grand Challenge**" [14]: Gracias a la Agencia de Proyectos de Investigación Avanzados de Defensa de los Estados Unidos, cada pocos años se organizan competiciones dentro del ámbito con recompensa económica que incentivan el esfuerzo para crear avances en esta materia. La primera edición se realizó en 2004, en el desierto de Mojave, retando a los equipos para conseguir recorrer una travesía de 240 km, la cual resulto en fracaso ya que ninguno de ellos logró conseguirlo, e incluso el récord quedó marcado en 11.78 km.

  Sin embargo, gracias a los datos recopilados, un año después se volvió a realizar una nueva edición en la cual se consiguieron grandes resultados. 5 de los equipos que participaron consiguieron completar el recorrido objetivo de 212 km, a pesar de que a priori la ruta tenía mayor dificultad, ya que contaba con mayor número de curvas y las vías eran en general menos amplias.

Figura 2–1 Victoria del equipo Team Tartan en el Urban Grand Challenge [15]

La última edición con especial importancia ocurrió en 2007, con la novedad de que la ruta de 96 km se realizaba en entorno urbano, en el cuál existía tráfico real y obstáculos, por lo que se debían respetar las normas de circulación establecidas de forma estricta. Nuevamente, 6 equipos lograron superar el reto, extrayendo conclusiones interesantes [11]:

  - A pesar de un gran avance de los sensores comerciales, siguen siendo inviables para alcanzar una navegación completamente autónoma, tanto en aspecto de coste y empaquetado, como en el de robustez ante perturbaciones electromagnéticas del entorno.
  - La validación del sistema no es viable actualmente, debido a la complejidad y dinamismo del entorno físico, lo cual es un problema para asegurar la confianza en un producto a gran escala.
  - Implicar la realimentación humana para cerrar el lazo en escenarios no esperados y desarrollar un sistema semiautónomo reduce sumamente las dificultades, y por tanto es viable.
  - El uso del tiempo hasta colisión como principal medio para medir el riesgo tiene inconvenientes importantes. Por ejemplo, si un vehículo se encuentra temporalmente parado, el riesgo de colisión se calcularía bajo. Sin embargo, es muy posible que este inicie una maniobra por lo que el peligro real es mayor. Un enfoque a este problema sería usar la estadística.

- "**EU CYBERCARS Project**" [16]: Este proyecto europeo, con un techo económico de 5 millones, perseguía acelerar el desarrollo e implementación de sistemas de transporte urbanos basados en vehículos autónomos, para el transporte de bienes y personas, con el objetivo de reducir los efectos negativos del uso de coches privados en ciudades (polución, congestión…). Entre otras tareas, el proyecto involucró un experimento real en la ciudad de Laussane para evaluar el rendimiento y la aceptación de usuario, aunque en condiciones controladas y baja velocidad.

Por otro lado, es importante analizar y situar los avances dentro de una clasificación. Para ello, la Sociedad de Ingenieros de Automoción (SAE) definió 6 niveles [17]:

- Nivel 0: Sin automatización en la conducción. La experiencia de conducción está totalmente en las manos del conductor. Sin embargo, pueden intervenir sistemas de manera momentánea para asistir (ABS, ESP, control de crucero, alerta por punto ciego, emergencia de frenada automática…)
- Nivel 1: Asistencia al conductor. El conductor tiene la tarea de conducir el vehículo y monitorizar el sistema, aunque este puede acelerar, frenar y tomar giros. Se incluye asistencia de centrado en carril y control de crucero adaptativo.
- Nivel 2: Automatización parcial de la conducción. Sigue requiriendo la atención e intervención del conductor, pero incluye sistemas de asistencia avanzada (ADAS) que asisten de forma continua en la aceleración, frenado y giro. La gran mayoría de vehículos autónomos del mercado se encuentran en este nivel.
- Nivel 3: Automatización de conducción condicionada. Ofrece funciones de conducción autónoma totales. Sin embargo, requiere de un conductor, ya que en ciertos casos puede pedir la intervención del usuario.
- Nivel 4: Alta automatización de la conducción. A pesar de un mal funcionamiento, el sistema cuenta con el equipamiento necesario para subsanarlo por sí mismo sin la intervención de un conductor. Sin embargo, el conductor todavía tiene la capacidad de tomar el control manual del vehículo. Este nivel viene definido sobre todo por el contexto de las regulaciones, ya que en condiciones muy bien definidas y específicas, este tipo de vehículo está permitido por ciudad en ciertas zonas.
- Nivel 5: Automatización de la conducción completa. Independientemente de las condiciones o el estado de la carretera, el vehículo tiene capacidad para actuar sin intervención humana ante emergencias.

El estado actual en el que nos encontramos, puede situarse en niveles distintos en función del campo a analizar. Para el caso de vehículos a gran escala, el nivel 2 es el estándar actual, si bien es cierto que algunos vehículos de alta gama como los Tesla con el Autopilot pueden considerarse de nivel 3. No obstante, esto no quiere decir que los niveles superiores no hayan sido alcanzados. Para el nivel 4, compañías como NAVYA en U.S., Alphabet's Waymo en Arizona o Volvo&Baidu en China empiezan a generar negocio con taxis autónomos a baja velocidad y en zonas limitadas. El nivel 5 se encuentra en testeo dentro de la investigación, y no se ha conseguido del todo, pero la tendencia en los últimos años demuestra que la meta no está lejos de nuestro alcance.

Figura 2–2 Baidu Apollo Robotaxi [18]

3 ARQUITECTURA SOFTWARE

Dado que el proyecto es totalmente simulado, se ha creado un repositorio software siguiendo una estructura organizada y bien definida que permita un correcto desarrollo y uso como producto final a futuros usuarios. El objetivo de este capítulo es informar acerca de esta, además de asentar los precedentes de los posteriores puntos de este documento.

3.1 Framework ROS

ROS (Robot Operative System) es un framework para el desarrollo de software para sistemas robóticos. En otras palabras, consiste en una colección de herramientas, librerías y convenciones para los lenguajes de programación C++ y Python, que provee la funcionalidad de meta sistema operativo.

Empezó como un proyecto personal de Keenan Wyrobek y Erik Berger en la Universidad de Stanford [19], con la intención de evitar la reinvención y reimplementación de las infraestructuras necesarias en robótica, permitiendo así centrar los esfuerzos en nuevas investigaciones.

Continuó siendo impulsado durante un tiempo por el laboratorio de investigación robótica Willow Garage y reforzado por la comunidad robótica dado su naturaleza open-source (licencia BSD). Tras el abandono de Willow Garage, se creó la fundación 'Open Robotics', la cual hasta la fecha de hoy es encargada de su mantenimiento y actualización.

Se debe considerar ROS como un middleware, es decir, un software que comunica la interfaz de usuario final con la capa de sistema operativo. Bajo su arquitectura de grafos, provee servicios estándar [20] muy útiles tales como:

- Abstracción del hardware
- Control a bajo nivel de dispositivos
- Implementación de funcionalidad de uso común
- Paso de mensajes entre procesos
- Mantenimiento de paquetes software

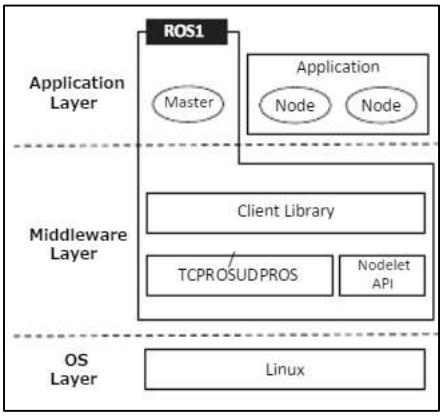


Figura 3–1 Arquitectura de ROS [41]

A pesar de no ser objetivo de este trabajo la enseñanza desde cero para el manejo de ROS (dado que ya se encuentra ampliamente documentado en su wiki oficial [21]), a continuación se menciona su funcionamiento básico para facilitar el seguimiento de las futuras explicaciones.

Existe un proceso principal por encima de todos los demás llamado 'Master', que se encarga gestionar una correcta comunicación entre los procesos lanzados por el usuario (nodos). Esta comunicación puede ser llevada a cabo de distintas maneras: servicios, acciones y tópicos. Siendo esta última la más común de todas, se basa en la característica de publicadores y subscriptores que pueden adquirir los nodos. Estos se encargan de enviar y recibir mensajes en un tópico (cola de mensajes) especificado, sin comprobar que la comunicación entre ambas partes sea satisfactoria, a diferencia de la clásica arquitectura cliente-servidor, que también puede ser utilizada como protocolo de comunicación para ciertas aplicaciones aunque no es lo común entre nodos.

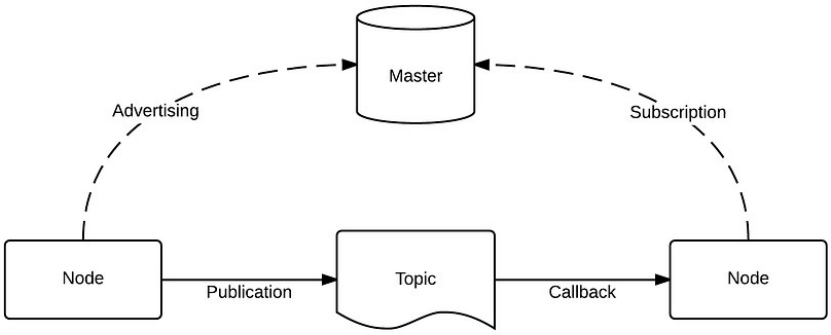


Figura 3–2 Esquema de funcionamiento de ROS [22]

Lo primero de todo, comentar que es requisito necesario ejecutar ROS bajo un sistema UNIX. Preferiblemente la distro de Ubuntu, dado que se considera el SO más estable sobre el que ejecutar ROS, al estar originado en este mismo.

Por otro lado, es importante saber que este proyecto ha sido desarrollado en ROS Noetic, última versión estable previa al cambio de paradigma que supone ROS2 [23]. Esto no quiere decir que no pueda funcionar con otras anteriores, ya que es posible que haya retrocompatibilidad, pero no puede asegurarse ya que no ha sido testeado.

Por último, comentar que ROS cuenta con un sistema de construcción de paquetes llamado 'catkin', que lleva a cabo la compilación y ejecución del software desarrollado. Además, aporta herramientas de línea de comandos que permiten una interfaz sencilla por terminal al usuario. Para su uso se require definir un directorio de trabajo, carpeta raíz dentro de la cual estará contenido todo nuestro software.

3.2 Simulador GAZEBO

Como puede encontrarse en su propia página oficial [23], el desarrollo de Gazebo comenzó en la Universidad del Sur de California en el año 2002, creado originalmente por Dr. Andrew Howard y su estudiante Nate Koenig. Posteriormente, John Hsu sentó los precedentes de su uso junto a ROS en uno de sus proyectos, lo cual alineó su trayectoria junto a este último, pasando por Willow Garage y siendo su actual encargado 'Open Robotics'.

Gazebo es un simulador 3D robótico open-source, que integra el motor de físicas Open Dynamics Engine (ODE) junto al motor de renderizado 3D Object-Oriented Graphics Rendering Engine (OGRE), permitiendo desarrollar simulaciones de un sistema con todo el realismo que se desee.

Gracias a su amplia acogida por la comunidad, Gazebo se encuentra integrado dentro del ecosistema de paquetes software de ROS, por lo que su uso para simulaciones de software robótico es ideal. Para este trabajo, se usa la versión 11, la cual viene instalado por defecto para ROS Noetic.

3.3 Estructura de paquetes

Tal y como se ha mencionado antes, es indispensable para un buen desempeño de la ejecución de ROS que se incluya todo el software del proyecto bajo la misma carpeta raíz, es decir, el directorio de trabajo. Para este caso, esta carpeta recibirá el nombre de 'GoCar'.

En ella podemos encontrarnos un archivo oculto .catkin_workspace el cual nos confirma el hecho de que esta carpeta se trata de un directorio de trabajo de ROS. También podremos ver las carpetas de build y devel, que son el resultado de la compilación llevada a cabo por la herramienta 'catkin'. La primera de ellas contiene los archivos generados para la correcta compilación del código fuente del repositorio, mientras la segunda almacena binarios, librerías y otros archivos relacionados con el espacio de desarrollo.

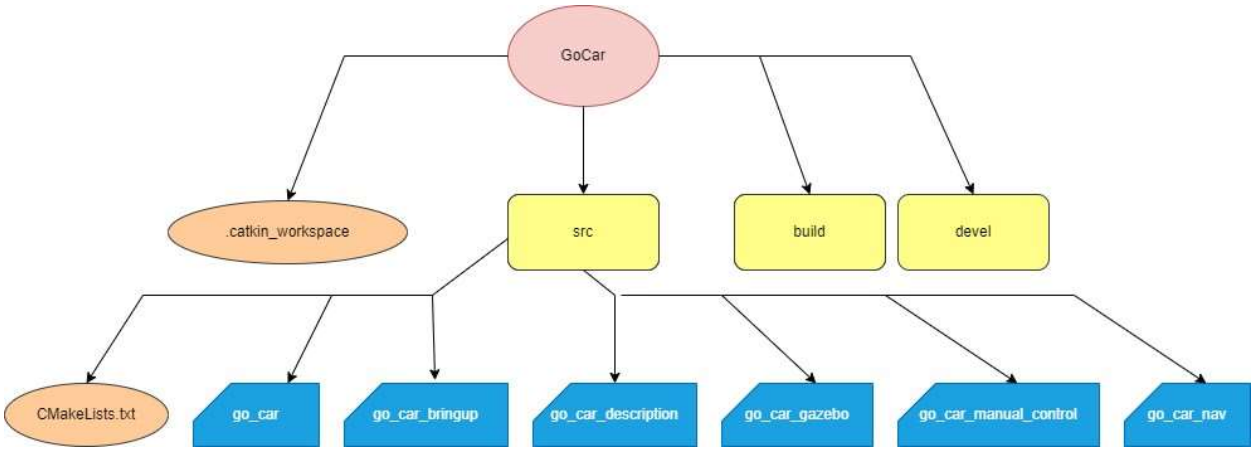


Figura 3–3 Árbol jerárquico del espacio de trabajo 'GoCar'

Como puede verse en la figura 3.3, dentro de la carpeta src (source), que almacena el código fuente de nuestro software, podemos encontrar cada uno de los paquetes en los que queda dividido, cumpliendo cada uno funcionalidades independientes:

- *go_car*: Se trata de un metapaquete, donde guardar información general extra que pueda ser de utilidad al usuario del proyecto, además de permitir listar todas las dependencias necesarias para ejecutar el entorno de simulación, de forma que gracias a la herramienta 'rosdep' el usuario pueda asegurarse de tener todas instaladas para la correcta compilación y ejecución del proyecto.

- *go_car_bringup*: Contiene los scripts necesarios para la configuración correcta de nuestro sistema para la ejecución de la aplicación (permiso de lectura de eventos del teclado, permiso de ejecución para binarios, carga de directories en catkin…), así como los lanzadores que se encargan de unificar en un único archivo todas las llamadas a cada uno de los binarios a ejecutar en nuestra aplicación.

- *go_car_description:* Este paquete aloja los archivos necesarios para definir el modelo del robot en simulación, tanto a nivel visual como de comportamiento físico, así como los sensores a incorporar en este.

- *go_car_gazebo:* Almacena todo lo relacionado con el entorno virtual a similar: mallas, materiales, scripts de renderizado, modelos de edificios, objetos, señales… que serán cargados en él. Dentro de él también se encuentra el código fuente del plugin de Gazebo diseñado para controlar las articulaciones del vehículo dado un comando de velocidad en km/h y giro de volante en radianes.

- *go_car_manual_control:* Contiene el código fuente de nodos y definición de mensajes utilizados para la lectura de las órdenes de control manual ejercidas por el usuario. Permite elegir entre control por mando o por teclado.

- *go_car_nav:* En este paquete residen todos los archivos de configuración y lanzadores necesarios para el mapeado, localización y generación de trayectorias, necesarios para la navegación autónoma del vehículo.

4 MODELADO DEL VEHÍCULO

Un vehículo no deja de ser un sistema robótico. En él podemos encontrar enlaces y articulaciones que permiten los grados de libertad necesarios para desplazarse de forma estable en un plano. En este proyecto se va a simular un vehículo de tracción trasera, simplificando los mecanismos de transmisión entre motor y ruedas motrices, así como la dirección electromecánica, hasta el punto de aplicar directamente los esfuerzos necesarios para conseguir la velocidad y posición angular deseada en cada momento para cada una de las ruedas.

4.1 Lenguaje URDF

El lenguaje URDF (unified robot description format) es un formato XML ampliamente utilizado por la comunidad robótica para la descripción de sus sistemas.

Permite definir los ejes locales para cada uno de los enlaces y articulaciones, así como las translaciones y rotaciones existentes entre ellos, que será más adelante publicado como transformada.

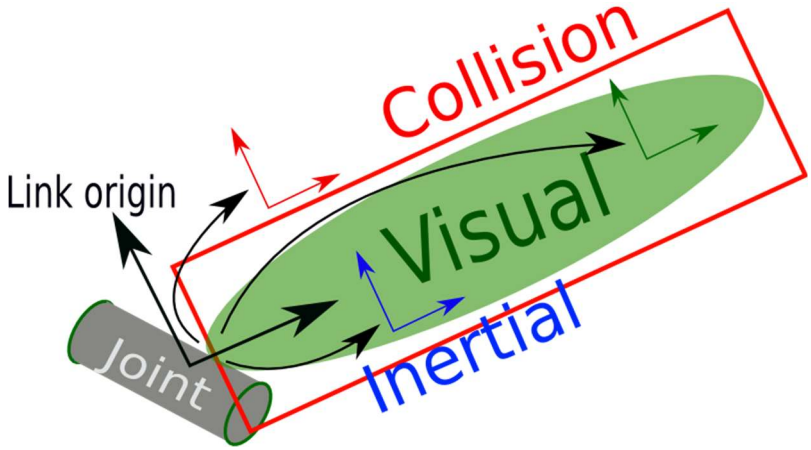


Figura 4–1 Esquema de elementos URDF principales para la definición de una articulación [21]

Podemos distinguir 3 partes principales:

- Visual: a través de mallas en formato dae se agrega el modelo 3d de ruedas y chasis, así como las texturas para dotarlo de mayor realismo.
- Colisión: Modelo simplificado del utilizado visualmente, formado con figuras primitivas (prismas rectangulares, cilindro, esfera) para indicar el volumen que ocupa el vehículo, a partir del cual generará fuerzas de repulsión para evitar atravesar superficies de otros modelos.
- Inercial: Permite definir la matriz de inercia de la articulación, la cual ha sido elegida por estimación con el cálculo de figuras primitivas similares.

Para el caso de los enlaces, se debe definir la articulación padre y la articulación hija, las cuales irán unidas por el tipo de enlace que sea definido, permitiendo movimiento en el eje indicado. Más en detalle:

- Tipo: Para este proyecto, la columna de dirección de las ruedas irá unida al chasis del vehículo por un enlace de revolución en torno al eje z, mientras que las ruedas motrices y delanteras también tendrán permitido movimiento de revolución, pero esta vez respecto al eje x.
- Límite: Para la columna de dirección, no nos interesa permitir un giro mayor a 0.8 radianes (aproximadamente 45 grados), mientras que las ruedas tendrán giro continuo sin límite. Además, también se especifica la velocidad máxima del giro de estos enlaces, simulando la potencia máxima que podrían dar los actuadores.

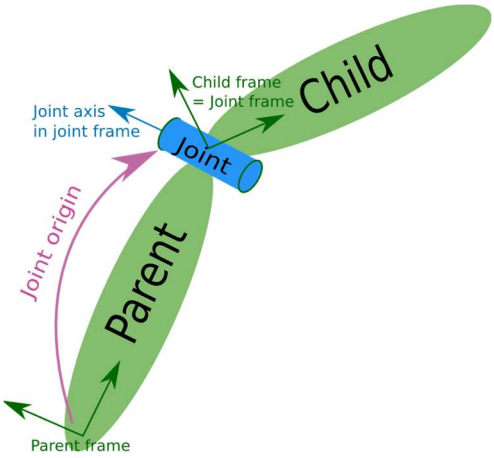


Figura 4–2 Esquema de elementos URDF principales para la definición de un enlace [21]

4.2 Descripción de enlaces y articulaciones del vehículo

Para la descripción del modelo, se ha utilizado el lenguaje Xacro frente al URDF. Realmente, este no es más que un macro lenguaje XML que permite los mismos elementos, pero da mayor versatilidad al permitir parametrizar, realizar cálculos y definir macros para elementos parecidos.

A la hora de la verdad, al cargarlo en el simulador, el proceso a seguir será un primer parseo y conversión del fichero xacro a urdf, y una segunda y última conversión a sdf, lenguaje que Gazebo ya sí es capaz de interpretar.

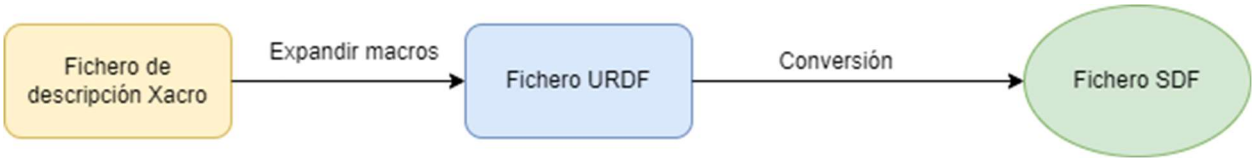


Figura 4–3 Línea de trabajo para cargar el model en simulador

El código para la implementación del modelo se ha dividido en distintas partes:

- go_car_model.xacro (ver en anexo 1.a): Archivo principal donde se definen los enlaces entre las distintas articulaciones especificando las transformadas entre estas, así como el tipo de movimiento según los grados de libertad y los límites en ángulo/distancia, esfuerzo máximo que puede aplicarse y velocidad máxima. Este archivo incluye importa a todos los demás.
- params.xacro (ver en anexo 1.b): Lista y da valor a todos los parámetros del modelo. Podemos encontrar:
  - Longitud, altura y anchura del chasis
  - Masa del chasis del vehículo y elementos para la matriz de inercia
  - Masa, elementos de matriz de inercia, radio y grosor de las ruedas
  - Longitud, peso y radio de grosor de las columnas de dirección del vehículo (por simplicidad en la implementación en URDF dadas limitaciones del lenguaje para definir enlaces mímicos se ha optado por desacoplar mecánicamente la dirección de cada rueda)
  - Vía (distancia entre ruedas del mismo eje) y batalla (distancia entre ejes) del vehículo
  - Altura de colocación de cámara y LiDAR

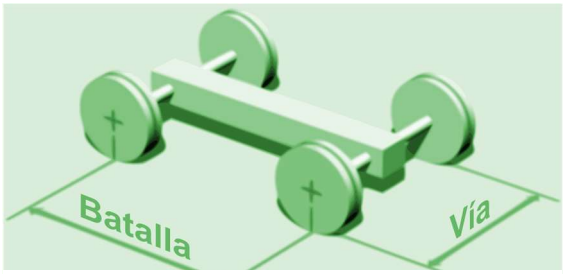


Figura 4–4 Definición geométrica en automoción [43]

También define 2 argumentos, como es 'details', que permite elegir el modelo de colisión a utilizar (performance: 'bounding box' que envuelve el vehículo para reducir la carga computacional, quality: malla completa con multitud de vértices para una correcta precisión) y 'car_model', que selecciona el modelo de vehículo visual a cargar, aunque actualmente sólo existe una opción: lincoln. La elección del primer argumento dependerá de la capacidad de cómputo del ordenador que ejecute la simulación.

- links.xacro (ver en anexo 1.c): Define las macros para los enlaces. En él, se define:
  - Masa y matriz de inercia de las articulaciones: En el caso del chasis, los valores han sido calculados gracias a la herramienta de AutoCAD a partir del modelo 3d obtenido gracias al trabajo previo de la Universidad de Arizona [24].

    Para el caso de las ruedas, por ejemplo, se usa una simplificación a un cilindro, para poder utilizar las expresiones ya conocidas [25] de su momento de inercia y tener una aproximación válida:

$$I = \begin{bmatrix} \frac{1}{12}m(3r^2 + h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2 + h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$$

  - Geometría de colisión: Dependiendo del valor que reciba el argumento 'details' comentado anteriormente en el lanzamiento de la simulación, utilizará figuras primitivas para ajustar un 'bounding box' que envuelva al coche, o recurrirá al archivo dae donde se encuentra la malla definida de mayor calidad.
  - Geometría visual: No coincidirá con la geometría de colisión, ya que incluso aun utilizando la malla en esta, la malla utilizada para la geometría visual tiene un mayor número de vértices (la del modelo de colisión ha sido extrapolada para reducir su complejidad y limitar su consumo de recursos). Esta malla es también autoría de la Universidad de Arizona [24].
- snippets.xacro (ver en anexo 1.d): Define funciones para calcular las ecuaciones de momentos de inercia de las figuras primitivas utilizadas.
- gazebo.xacro (ver en anexo 1.f): Define parámetros físicos para los enlaces y las articulaciones, tales como el rozamiento, constantes de 'kp' y 'kd' para definir la rigidez y amortiguación ante golpes de una articulación, así como la distancia mínima de penetración entre modelos. El ajuste de estos parámetros es importante ya que afecta en el correcto cálculo del rozamiento de las ruedas con el suelo.

  En este archivo también se hace llamadas a plugins de gazebo: "joint_state_publisher" para publicar el estado de los enlaces y articulaciones y poder realimentar al control con ello, como si fuera una especie de encoder simulado; "p3d_base_controller" para obtener los datos de la posición real del robot en todo momento en relación a los ejes globales del mundo simulado.

  Por último, se llama al plugin "ackermann_control" desarrollado por mí mismo, utilizando la librería compilada a partir del código fuente que será explicado más adelante, para asegurar el control del vehículo.
- Sensors.xacro (ver en anexo 1.e): Recurriendo a plugins oficiales de Gazebo, se definen los siguientes sensores:

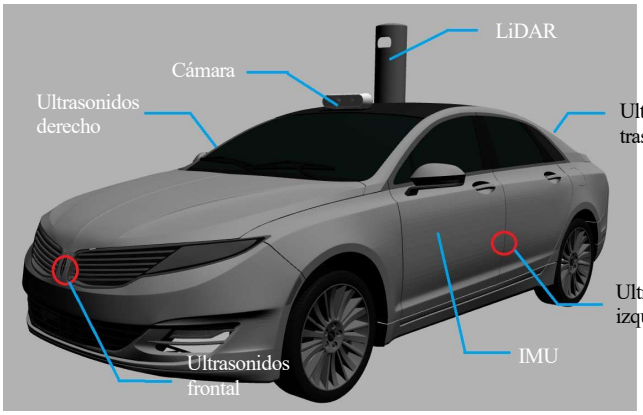


Figura 4–5 Disposición de sensores en el vehículo

- *Cámara RGB*: Se definen los parámetros característicos de la cámara reflejados en la siguiente tabla:

Tabla 4–1. Parámetros de la cámara simulada

Parámetro	Valor
Tasa de actualización de imagen	30 Hz
Campo de visión horizontal	1.4 rad
Dimensiones matriz de píxeles	800x800
Profundidad de color	32 bits (con formato R8G8B8)
Desviación típica del ruido gaussiano	0.007

- *LiDAR*: Se pretende emular lo mejor posible el comportamiento del sensor comercial HDL32e de Velodyne. Las características han sido extraídas de la hoja de datos del fabricante [26]:

Tabla 4–2. Parámetros del sensor LiDAR simulado

Parámetro	Valor
Tasa de actualización de medida	5 Hz
Resolución horizontal	300 muestras*
Ángulo mínimo/máximo horizontal	- π | π
Resolución vertical	40 muestras
Ángulo mínimo/máximo vertical	-30º | 10º
Rango	De 3 a 100 metros**
Desviación típica del ruido gaussiano	0.02

* Muy reducida respecto a la del fabricante dado que la emulación del sensor implica cálculos complejos de fondo que son costosos computacionalmente

**Para emular el propio filtro del fabricante que elimina las medidas inferiores a la distancia umbral fijada, se fija el mínimo en 3 metros

- *IMU*: Simulando la unidad inercial comercial 3DM-CX5, considerando los datos aportados por el fabricante [27]:

Tabla 4–3. Parámetros del sensor IMU simulado

Parámetro	Valor
Tasa de actualización de medida a la salida	100 Hz*
Desviación típica	200 ug**

* El sensor del fabricante permite ajustar la frecuencia de datos en salida entre 1 y 1000 Hz

** Calculado a partir de la densidad de ruido como $ND\sqrt{SR}$, con ND densidad de ruido y SR tasa de muestreo

- *Receptor GPS:* Basado en el receptor LS20033, con características [28]:

Tabla 4–4. Parámetros del receptor GPS simulado

Parámetro	Valor
Tasa de actualización de medida a la salida	1 Hz
Precisión	3m

- *De rango:* Se han utilizado valores del sensor ultrasonidos MB8450 [29]:

Tabla 4–5. Parámetros de los sensores de rango simulados

Parámetro	Valor
Tasa de actualización de medida a la salida	50 Hz
Campo de visión horizontal / vertical	16º|16º
Rango	10 cm
Resolución	20 cm

Visto en un esquema, el árbol de transformadas resultante de la descripción anterior, quedaría de la siguiente manera:

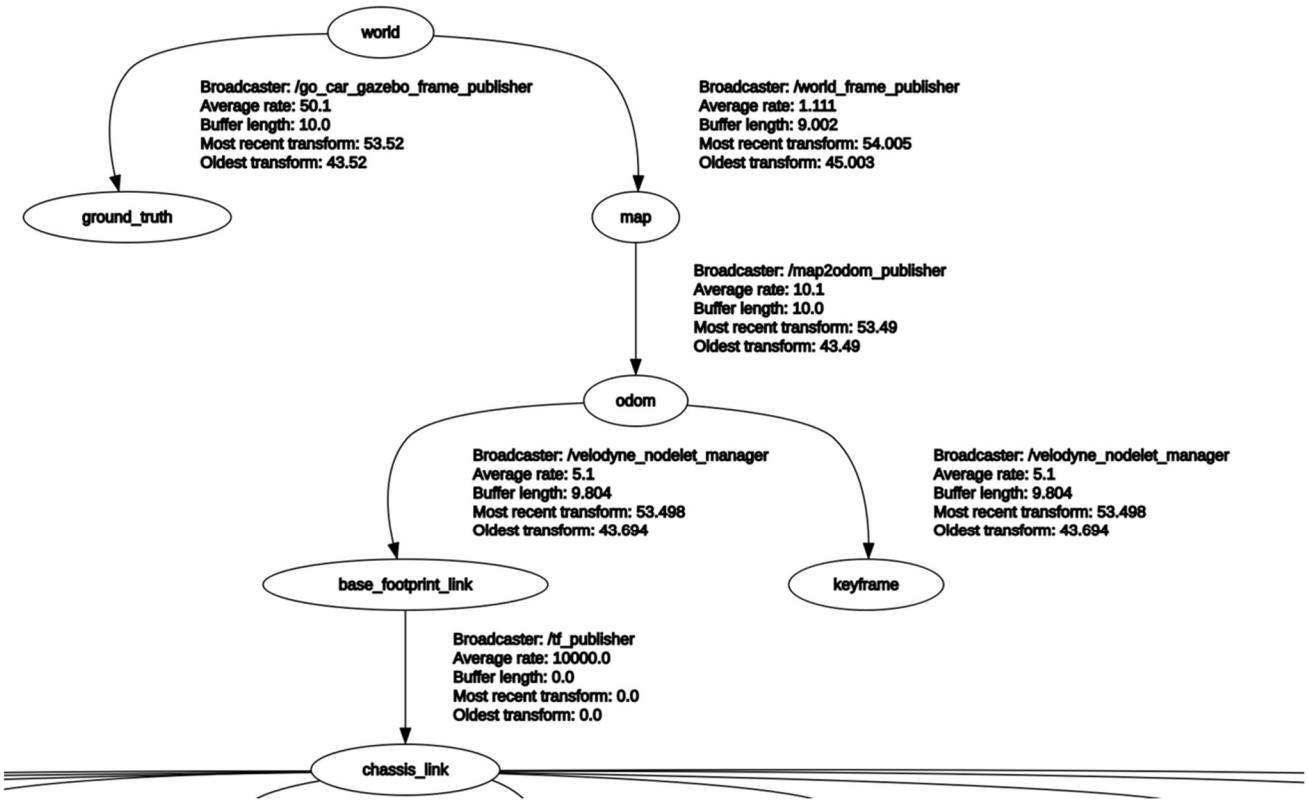


Figura 4–6 Niveles primarios del árbol de transformadas mostrado por RQT

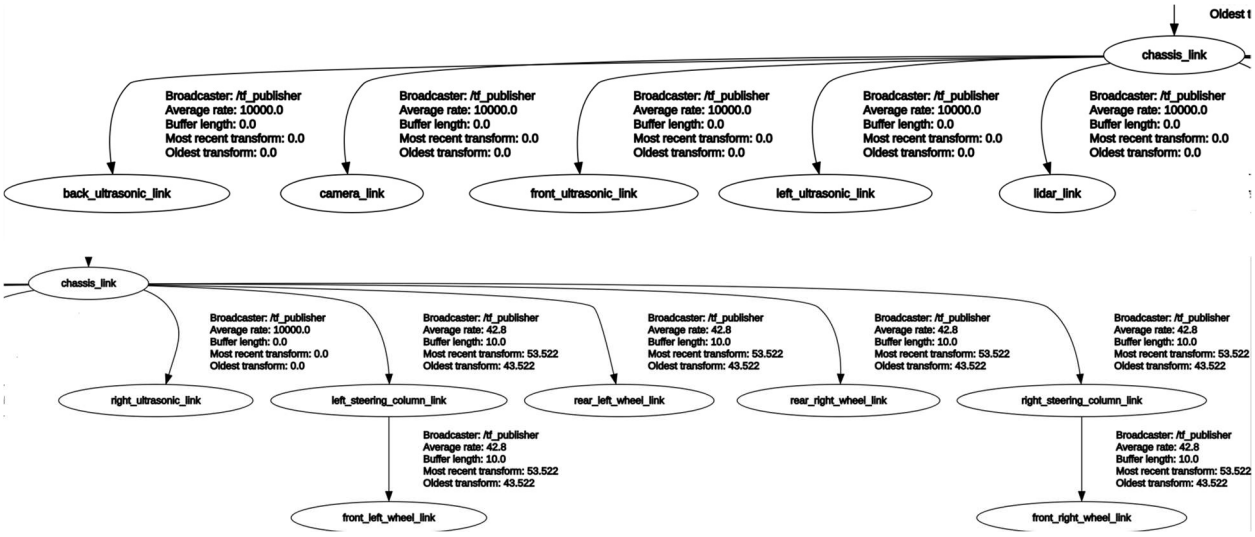
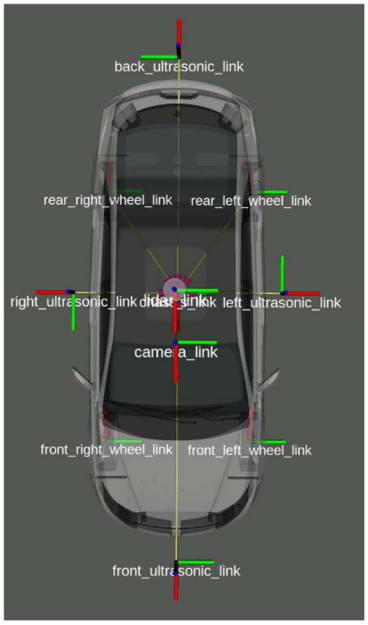


Figura 4–7 Articulaciones hijas de chassis_link en árbol de transformadas mostrado por RQT

Visualizando en Rviz se puede ver el resultado de transformadas entre sistemas de referencia de cada articulación del modelo, previo a su inclusión en el simulador:



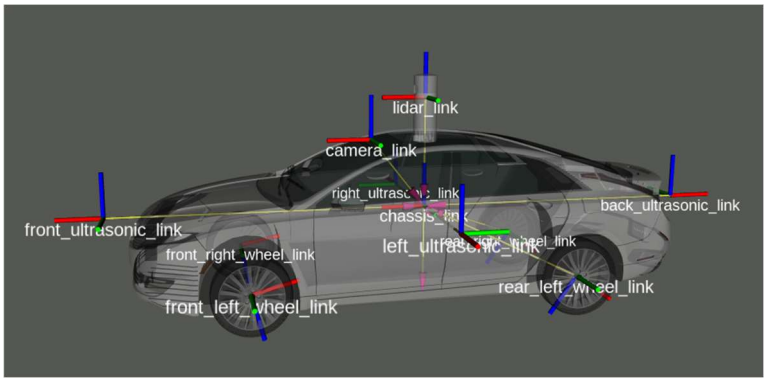


Figura 4–8 Resultado visual del modelo con ejes locales de articulaciones reflejados

4.3 Uso de API de Gazebo para inclusión de plugins

Se ha decidido recurrir a la API de Gazebo en C++ para comunicarse con la interfaz del simulador, permitiendo el manejo de las articulaciones del modelo del vehículo, y por tanto teniendo la posibilidad de implementar controladores PID que dado como referencia una velocidad lineal y posición angular, calculen la señal de actuación (fuerza a aplicar por los actuadores) para cerrar el bucle.

Para la implementación, se ha heredado de la clase padre 'gazebo::ModelPlugin' y se han sobrecargado los métodos:

-*Load:* Es una especie de constructor. Al cargar el plugin de un modelo se realiza una llamada a esta función, cuyo propósito es el de inicializar los objetos necesarios, leer los parámetros del SDF y de rosparam, así como setear las ganancias de los controles.

-*OnUpdate:* Es llamado en cada paso de simulación (configurado en este caso para producirse cada milisegundo). Por tanto, define la frecuencia de cálculo del controlador, así como se encarga de leer los estados de los enlaces para informar sobre las señales de actuación aplicadas en estas.

La arquitectura de control resultante tiene 4 PIDs independientes para cada una de los enlaces, aunque el comportamiento para ellos y por tanto sus constantes sean idénticas dada la simetría del Sistema respecto al eje longitudinal del vehículo. Esto es de suma importancia, ya que como se mencionaba anteriormente, no existe una única columna de dirección en el mecanismo del vehículo, sino 2 independientes.

Esto permite calcular vía software la geometría de giro de cada rueda por separado dado el giro que se comande. En un vehículo normal, se utiliza la geometría de Ackermann, que permite mejorar la estabilidad y el desgaste excesivo de los neumáticos. Por ejemplo, en un giro a izquierdas la rueda izquierda tiene un ángulo mayor a la derecha. Esto se hace para que el punto de corte de todos los ejes de las ruedas coincida en un único punto, que marcará la capacidad de giro del vehículo (radio de giro).

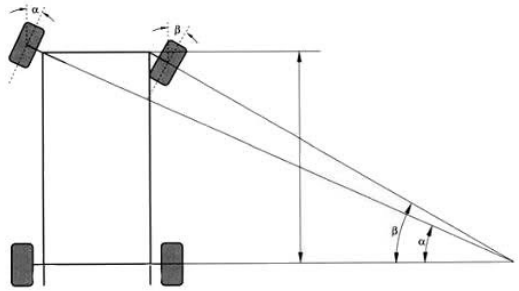


Figura 4–9 Principio de funcionamiento de la geometría Ackermann [30]

Sin embargo, esto se ha despreciado en el modelado del vehículo y los ejes de las ruedas directrices están en paralelo, ya que permitía la simplicidad del sistema y no afectaba de forma notable al comportamiento (los errores de cálculo del simulador en los puntos de contacto entre neumático y suelo son más influyentes). Esta geometría de Ackermann permite mejorar la estabilidad y el desgaste excesivo de los neumáticos

Los valores de ganancia ajustados experimentalmente pueden ser consultados en la siguiente tabla:

Tabla 4–6. Ganancias y saturaciones para controlador PID en velocidad

Parámetro	Valor
Proporcional	500 Nmh/km
Integral	0 Nmh/km
Derivativo	0 Nmh/km
Error integral máximo	100 Nmh/km
Error integral mínimo	-100 Nmh/km
Salida máxima	1500 Nm
Salida mínima	-1500 Nm

Tabla 4–7. Ganancias y saturaciones para controlador PID en posición

Parámetro	Valor
Proporcional	15000 Nms/rad
Integral	10000 Nms/rad
Derivativo	1000 Nms/rad
Error integral máximo	500 Nms/rad
Error integral mínimo	-500 Nms/rad
Salida máxima	1500 Nm
Salida mínima	-1500 Nm

Con el objetivo de ver los resultados obtenidos tras su implementación, se procede a la realización y recopilación de resultados en distintos experimentos.

Primero de todo, se realiza un experimento en línea recta dando escalones de velocidad a la referencia para ver el comportamiento obtenido en el vehículo de forma desacoplada con el giro.

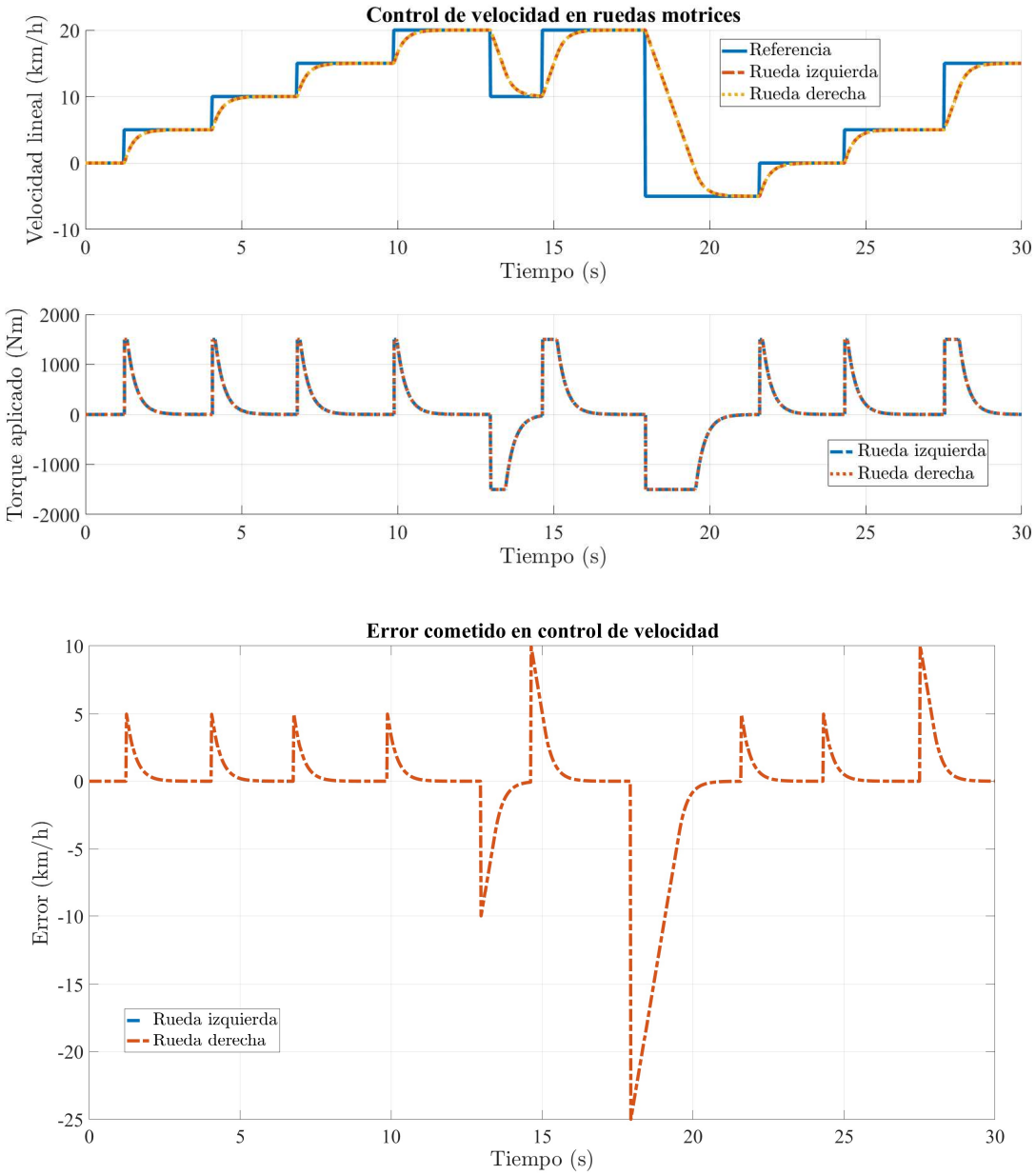


Figura 4–10 Seguimiento de referencia de velocidad en controladores de bajo nivel del vehículo

Como se puede ver, el tiempo de subida está en torno a los 1.5 segundos. Además, el error en régimen permanente es nulo y, por otro lado, el sistema tiene un comportamiento lo suficientemente lineal como para trabajar correctamente dentro de su zona de funcionamiento.

La figura 4.8 también permite comprobar como la saturación en la señal de actuación está teniendo efecto, y en escalones de mayor magnitud como el aplicado a los 17 segundos, provoca que el tiempo de subida sea algo mayor, pero dentro de lo permisible.

Por último, se puede destacar la simetría de las señales controladas en ambas ruedas como se hacía referencia anteriormente. A pesar de no haber un mecanismo que asegure que ambos enlaces tengan el mismo estado, está ocurriendo tal y como se esperaba por la naturaleza del propio sistema.

En otro de los experimentos llevado a cabo, se realiza un testeo en parado del control de giro de las ruedas directrices. De esta manera, no existe acoplamiento con el control de velocidad.

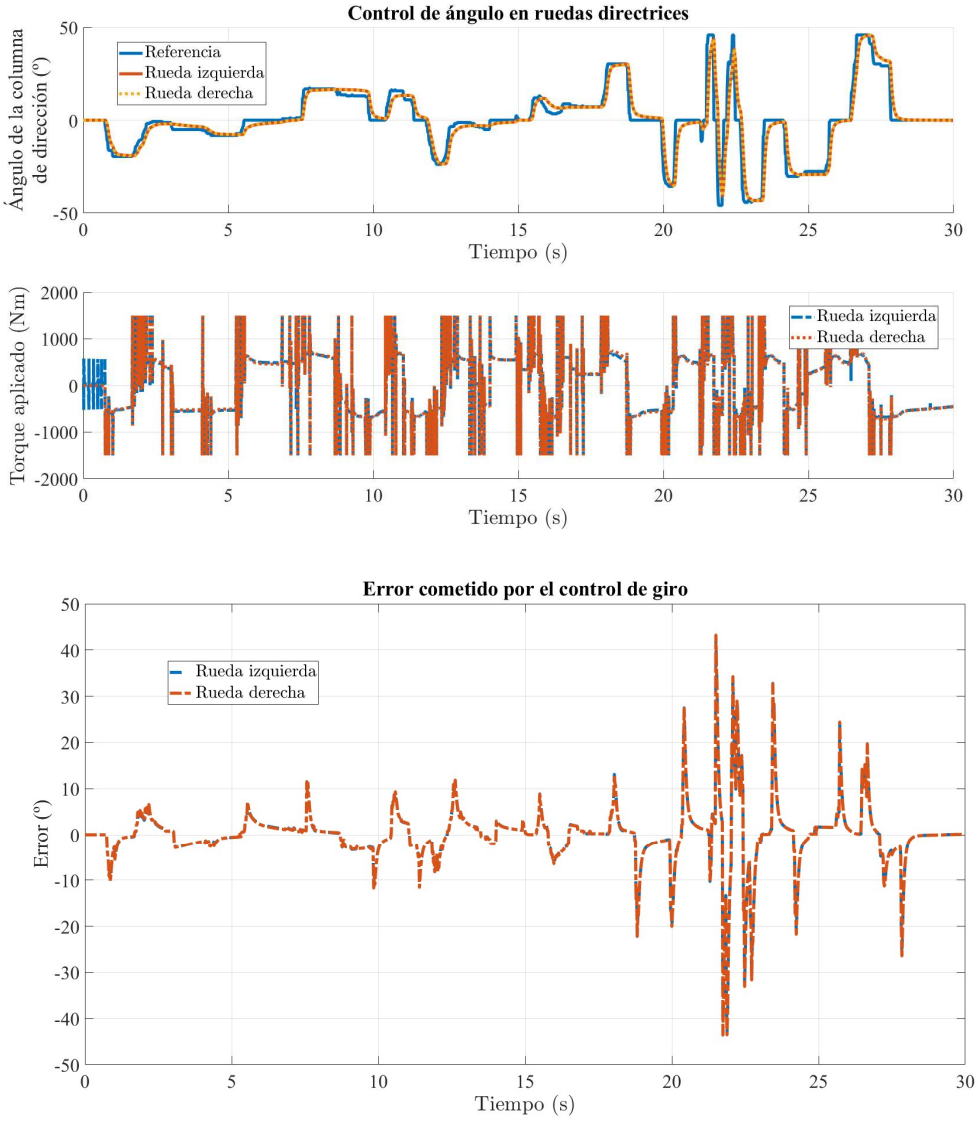


Figura 4–11 Seguimiento de referencia de giro en controladores de bajo nivel del vehículo

Como se puede ver a primera vista, la señal de actuación aplicada por el controlador no es la esperable. Sin embargo, podemos ver como se consigue seguir la referencia con error en régimen permanente nulo y con tiempos de subida cortos (dentro de las especificaciones deseadas para que el vehículo sea reactivo).

La explicación de este fenómeno se debe principalmente a 2 motivos:

-La no linealidad del rozamiento del suelo con el neumático.

-El motor de físicas que implementa Gazebo, ODE. El contacto de la rueda con el suelo no es parecido al que habría en la realidad. Normalmente, las fuerzas de fricción del neumático se estiman a partir de la huella de contacto entre esta y el suelo. Sin embargo, en Gazebo, a pesar de disminuir la rigidez del neumático y permitir penetración entre los elementos (la colisión es calculada entre un plano y un cilindro), solo se consiguen 4 puntos de contacto, sobre los que el motor aplica el torque del controlador. Además, este contacto no es continuo ya que existe un rebote mínimo entre ambos.

Por la limitación del simulador anterior, se ha decidido hacer el ajuste experimental que pudiera dar el comportamiento deseado, a pesar de que ello haya supuesto poner ganancias demasiado altas (la señal de salida está constantemente saturando) que provocan una sobrerespuesta del sistema.

Una vez más, como se puede ver, a pesar de no existir un enlace mecánico, la simetría del sistema y las ganancias idénticas para los controles de ambos lados hacen que el comportamiento sea el mismo.

Aunque las pruebas anteriores fueran satisfactorias, el fenómeno de acoplamiento entre velocidad y giro tiene efecto a la salida. Por tanto, se lleva a cabo un experimento en el que se conduce el coche trazando curvas y variando la velocidad, esta vez no aplicando sólo escalones en la referencia.

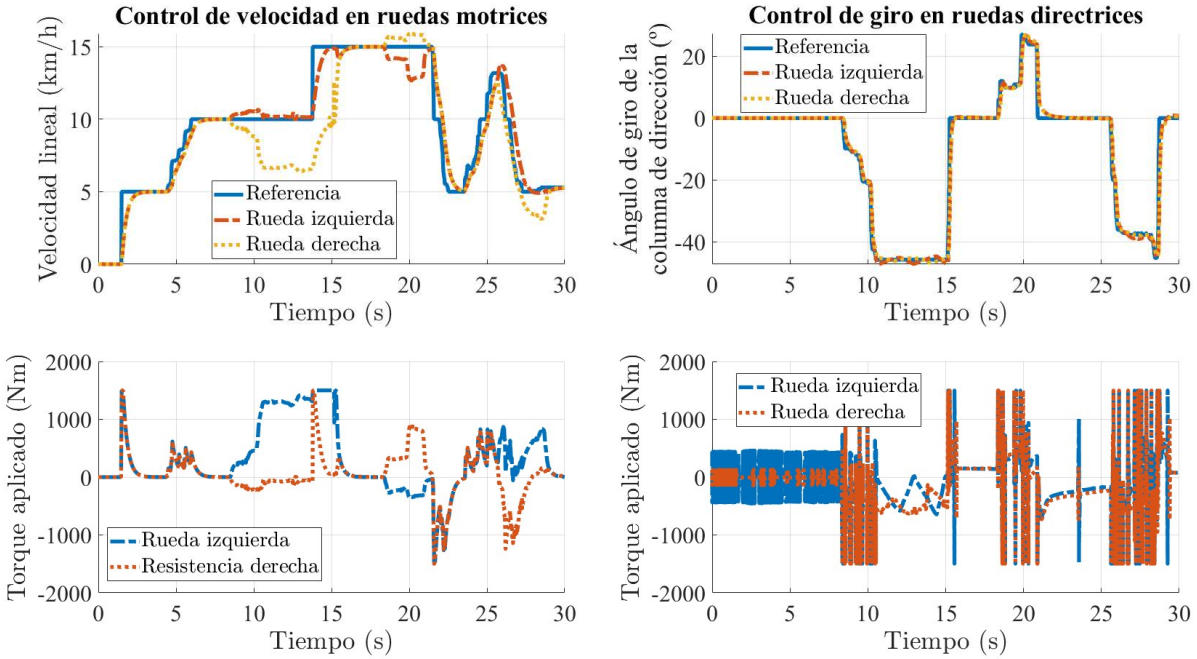


Figura 4–12 Seguimiento de ambas referencias en prueba de conducción

Las gráficas resultantes muestran como los controladores de giro no se ven afectados, mientras que los de velocidad tienen problemas para seguir la referencia de velocidad cuándo el giro realizado sobrepasa los 35º.

A pesar de no ser lo deseado, estos resultados eran los esperados, ya que el sistema no es del todo lineal en estos casos. Al hacer giros, la resistencia que debe vencer el neumático interior respecto al exterior es notoria, por lo que da pie a estos errores.

Aún con este problema, el comportamiento del control en general es correcto y dentro de especificaciones.

5 INTERFAZ HMI PARA CONTROL MANUAL

Para mejorar la experiencia de usuario con el simulador, se ha decidido dedicar esfuerzo a desarrollar un sistema de envío de comandos manuales. Para ello, se permite elegir entre 2 dispositivos hardware diferentes: mando de consola o teclado.

En la implementación software de las soluciones para conseguir esta interacción, se ha implementado una estructura wrapper que envuelve la funcionalidad de driver de los dispositivos en un código incluyendo la capa de ROS. Gráficamente el resultado es:

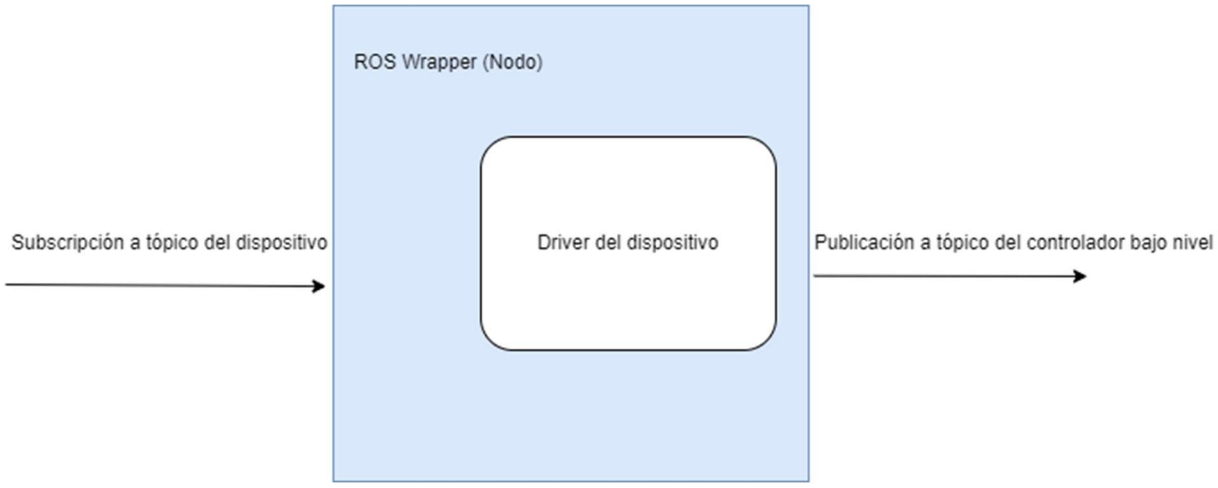


Figura 5–1 Esquema de implementación de funcionalidad HMI en ROS

Esto consigue hacer más robusta y ordenada la implementación del nodo, permitiendo depurar por separado cada una de las partes para encontrar posibles fallos que puedan surgir de manera desacoplada.

5.1 Envío de comandos vía 'Gamepad'

En los sistemas operativos Unix como Ubuntu, todos los dispositivos externos conectados al ordenador son registrados en el sistema de ficheros del SO creándose un fichero virtual en la ruta /dev/ con el nombre de jsX.

Gracias a un nodo ya desarrollado por la comunidad [31] conocido como 'Joy' que se encarga de leer los eventos que se envían en forma de buffer de bits al conectar un dispositivo con joystick, se procesa la lectura de cada una de las entradas recibidas (potenciómetros, pulsaciones de botones…) para publicar en un driver que traduce estos estímulos en comandos de referencia para el plugin de control de Gazebo, siguiendo el siguiente esquema:

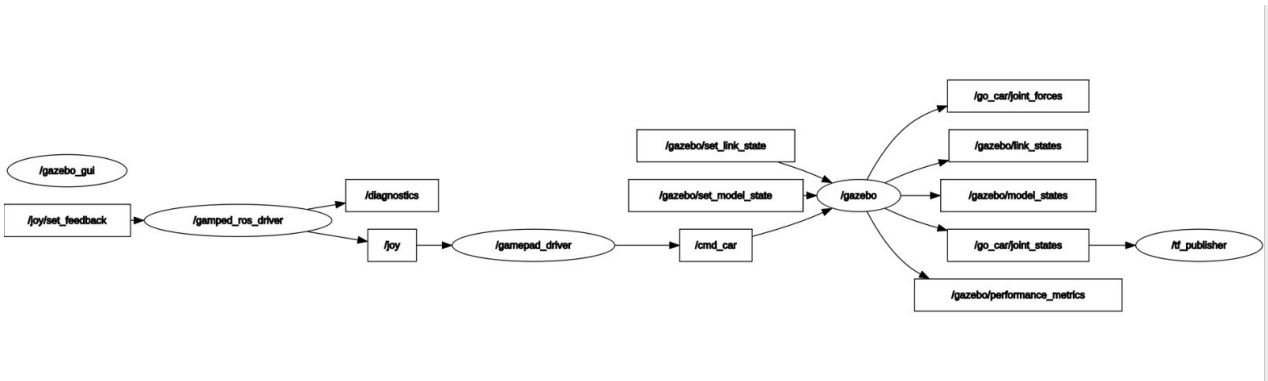


Figura 5–2 Conexión entre nodos para funcionalidad HMI de control manual con mando

Este nodo intermediario (ver anexo 2.a) suple varias funciones:

1) Mapeado lineal de la entrada analógica de los joysticks dentro de los límites de velocidad establecidos en sus parámetros, para dar una referencia al control correspondiente a la inclinación aportada por el usuario.

2) Define una clase enumerada de marchas, que funciona como selector de velocidades. Sobrescribiendo los operadores + y – para esta, se permite el manejo del vehículo con 4 marchas de avance, neutral y reversa.

Como pequeña guía para el usuario, se indica a continuación que entradas corresponden con las actuaciones del vehículo:

Figura 5–3 Asignación de entradas del mando para el driver

Se debe tener en cuenta que en marcha neutral el vehículo estará parado a pesar de intentar acelerarse. Por otro lado, cada marcha selecciona una velocidad de offset, sobre la cual se puede reducir/aumentar hasta 5 km/h en función de la inclinación del joystick. La velocidad máxima del vehículo en cuarta marcha es de 25 km/h. En reversa, no se podrá ir hacia delante, por lo que sólo puede retrocederse y a una velocidad máxima de 5 km/h.

5.2 Lectura de entradas por teclado

Lo primero, se especifica un diseño de funcionamiento. Se pretende registrar toda pulsación del teclado, y saber tanto si una tecla ha sido pulsada, sigue siendo pulsada o ha sido levantada. Por otro lado, es necesario tener capacidad para lectura simultánea de varias pulsaciones a la vez, ya que el usuario querrá girar y acelerar a la vez en más de una ocasión.

En un primer intento, se optó por resolver este problema leyendo la entrada del teclado desde la interfaz de terminal. No obstante, tan pronto como se probó, se pudo ver una limitación importante en dicha implementación.

1) No era posible detectar pulsación simultánea de varias teclas a la vez, ya que la terminal sólo envía la última tecla que ha sido pulsada.

2) No tenía una buena experiencia de usuario, ya que forzaba a utilizar la consola de comandos, no pudiendo tener la ventana gráfica del simulador activa en primer plano.

Como solución a dichas limitaciones, se planteó la lectura en fondo del archivo virtual de eventos del teclado, el cuál puede ser encontrado en la ruta **/dev/inputX** para sistemas Unix.

Para conseguir este objetivo, primero de todo se implementó el código fuente (ver anexo 2.b) de un nodo capaz de parsear el fichero **/proc/bus/input/devices**, en busca de información acerca de los dispositivos conectados al host. Posteriormente, se filtra esta lista de dispositivos en búsqueda de aquellos cuyos eventos soportados sean compatibles con un teclado, y se selecciona la primera opción de todas las posibles (de haber más de un teclado conectado, solo funcionará la entrada para uno de ellos).

Se abrirá el fichero de eventos del dispositivo para lectura en modo de buffer de bits no bloqueante (lo que

significa que de no recibir ningún buffer del archivo nuestra función no quedará a la espera), por lo que la lectura se realizará por polling en un bucle a 50 Hz. En caso contrario, cuando el buffer sea recibido se revisará su tamaño para dividirlo en estructuras de eventos, y procesar el código de tecla asociado [32], distinguiendo así que tecla ha sido utilizada, así como calcular el comando de salida a publicar para el controlador.

Como detalle a tener en cuenta, respecto al primer enfoque, también consigue realizar la lectura de las pulsaciones del dispositivo físico del teclado de fondo, por lo que permite trabajar en paralelo a otra aplicación que también necesite la lectura del teclado, así como tener la ventana gráfica del simulador activa en primer plano o cualquier otra y seguir controlando el vehículo.

Sin embargo, como inconveniente encontramos el lado de la seguridad de Linux. Para poder acceder a la lectura de los eventos del dispositivo, se debe pedir permiso para el grupo de nuestro usuario. Para ello se ha implementado un pequeño script en bash que permite al usuario elegir si quiere o no dar dichos permisos, informando que es necesario para esta funcionalidad.

Figura 5–4 Asignación de entradas del teclado para el driver

El funcionamiento a nivel de usuario es similar a la interfaz con mando. Con las teclas de dirección, se controla la dirección o aceleración/frenado del vehículo. Con los números se puede ajustar la marcha, y para utilizar la reversa se debe presionar la tecla '-'. En este caso no se puede dar entradas analógicas, por lo que al pulsar UP o DOWN se aumenta/disminuye en 5 km/h la velocidad del vehículo respecto al offset de la marcha.

6 CREACIÓN DE ENTORNO VIRTUAL

De cara a una simulación realista que permita un estudio exhaustivo de los algoritmos implementados para la conducción autónoma del vehículo, es necesario desarrollar un entorno virtual que contenga las diferentes condiciones habituales a las que se enfrentaría un coche en un viaje del día a día.

Por ello, en este capítulo se aborda la construcción de un mundo basado en una ciudad ficticia, en la cual se incluyen marcas viales, peatones, señales de tráfico, otros coches aparcados, edificios… Y en general una variedad de condiciones sobre las que testear la robustez de los algoritmos en experimentos.

6.1 Lenguaje SDF

Un archivo SDF (Simulation Description Format), contiene la información necesaria estructurada como elementos XML para describir entornos y objetos virtuales.

Para este proyecto, se utiliza la version 1.7, la cual permite definir [33], entre otros:

- World: Encapsula todos los elementos del mundo (modelos, escena, físicas y plugins, configuración de cámara…).
- Scene: Permite definir las condiciones ambientales (iluminación, nubes, niebla, viento, humedad…)
- Physics: Permite la elección y ajuste de los parámetros del motor de físicas a utilizar.
- Model: Define el objeto en sí, dividido en distintas partes: link, joint, collision, visual, material, geometry…

6.2 Mundo

Siguiendo la especificación de formato anterior, utilicé el programa de edición de imágenes GIMP, para, a partir de imágenes 2d de vías de tráfico, crear un modelo de geometría plana donde colocar el asfalto y conectar las vías entre sí, generando el suelo de las calles de la ciudad.

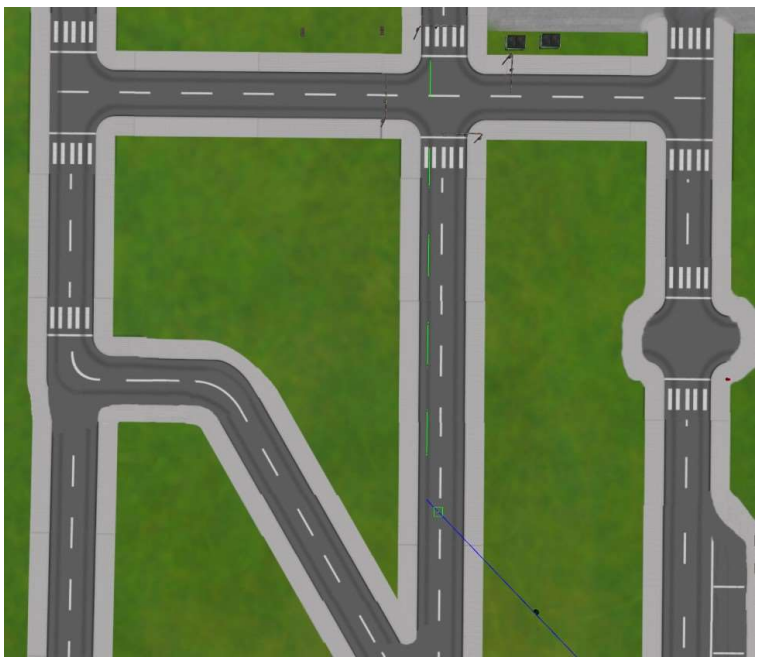


Figura 6–1 'Roadmap' del entorno simulado

Por otro lado, en el apartado de los modelos añadidos al mundo, muchos de ellos han sido obtenidos del repositorio en común de Gazebo [34]:

- Edificios: comisaría, tienda de regalos, salón de peluquería, oficina de abogados, casas, escuela, aparcamiento de varias plantas…

Figura 6–2 Edificios del entorno simulado

- Semáforos y señales de stop

Figura 6–3 Señales verticales de circulación en cruces del entorno simulado

- Vehículos estacionados: modelos de coche de distintos tamaños (berlinas, todoterrenos, utilitarios), quads, autobuses, ambulancia…

Figura 6–4 Aparcamiento con vehículos estacionados en el entorno simulado

- Comercios con aparcamiento a las afueras y estaciones de servicio y repostaje

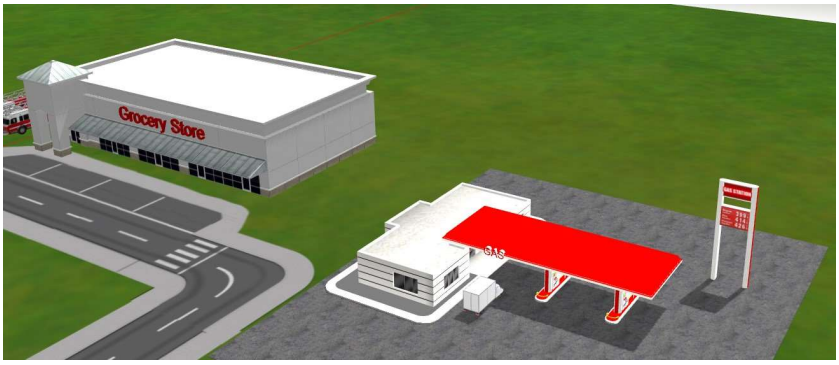


Figura 6–5 Gasolinera y supermercado con espacio para aparcamiento del entorno simulado

- Peatones: En un principio, se pretendió hacerlos móviles gracias a la opción de actores que soporta Gazebo [35]. No obstante, tras implementar una animación simple para recorrer los pasos de peatones, el rendimiento de la simulación se vio sumamente mermado. Se concluye que no se encuentra lo suficientemente bien optimizado, por lo que no es viable de implementar en este proyecto.

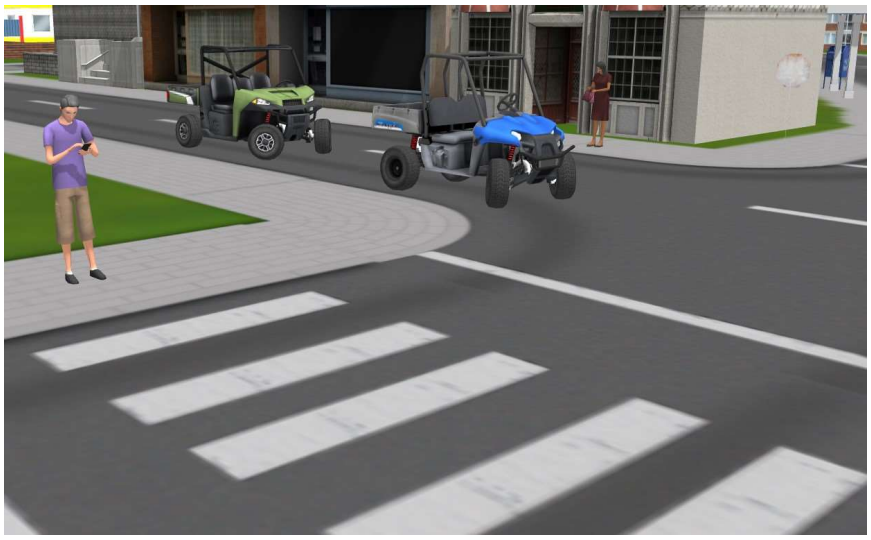


Figura 6–6 Peatones alrededor del entorno simulado

Además de estos modelos, he desarrollado otros propios por mí mismo recurriendo a texturizado de imágenes y scripts de materiales (ver anexo 3 completo): bloques de apartamentos modernos, antiguos y rascacielos.

Figura 6–7 Modelos de edificios desarrollados por mí mismo del entorno simulado

Todo este entorno 3d construido servirá posteriormente para validar el funcionamiento de algoritmos de SLAM (localización y mapeado simultáneo) así como del planificador de trayectorias que se usará en la parte de navegación.

Gracias a la variedad en las distintas partes de la ciudad, podrá comprobarse cuáles son sus debilidades y bajo qué situaciones el funcionamiento es mejor.

El resultado final, bajo vista cenital, queda de la siguiente manera:

Figura 6–8 Vista aérea de la ciudad

7 PILA DE NAVEGACIÓN AUTÓNOMA

Con el propósito de dotar al vehículo de navegación autónoma para llegar de un punto de origen al destino que se le especifica, es requisito indispensable:

- Poseer un mapa del entorno que se debe recorrer o en su defecto, ir generándolo por el camino.
- Localizar al robot dentro de este mapa, para así poder planificar una ruta en función de los pesos definidos.

7.1 Localización y mapeo simultáneo

La tarea de conducción es más compleja de lo que uno puede pensar a priori. En la mayoría de ocasiones no conocemos la zona que debemos recorrer, por lo que no se puede dar por hecho que se conoce el mapa del entorno. Además, pueden existir condiciones cambiantes: vehículos aparcados, peatones, obras, firme en mal estado… Es por ello que el uso de un mapa estático para la navegación no es viable.

Por ello, en este proyecto se recurre a técnicas SLAM de mapeado y localización simultánea. En concreto, se utiliza el enfoque de Graph SLAM para resolver este problema, el cual consiste en la construcción y optimización de un grafo donde cada nodo representa las poses estimadas a ajustar, mientras que las conexiones entre estos representan restricciones, principalmente dadas por los sensores.

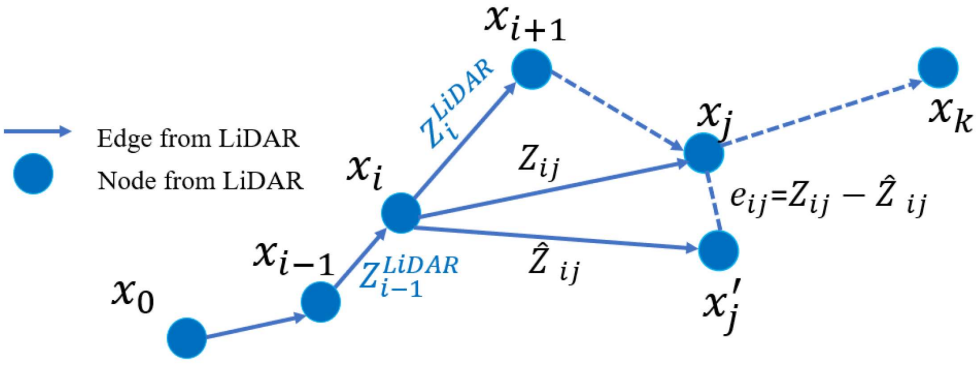


Figura 7–1 Proposición de arquitectura grafo para Graph-SLAM con LiDAR [36]

Siendo x_k el vector de parámetros que describe la pose del nodo k, z_k la media y y Ω_k la matriz de información relacionada a ese nodo, y e_k una función de error entre parámetros y restricciones, podríamos definir la función objetivo como [37]:

$$F(x) = \sum e_k(x_k, z_k)^T \Omega_k e_k(x_k, z_k)$$

El aborde matemático del problema consistiría en la linealización y posterior minimización de esta función usando los algoritmos de Gauss-Nuewton o Levenberg-Marquardt [37].

Teniendo en mente el problema base, es necesario la elección de sensores y su colocación en el vehículo. Esta no fue nada trivial. En un principio, colocar unidades de LiDAR 2D a distintas alturas podría servir para la tarea, dado que la navegación del vehículo es principalmente en un plano (a pesar de existir diferentes inclinaciones por el camino, los gradientes son pequeños por lo que la aproximación podría servir en gran parte de las situaciones). Esto permite reducir el número de datos a procesar, por tanto reducir la carga computacional.

Sin embargo, esta configuración tiene varios inconvenientes. Primero de todo, a pesar de la enorme reducción de precios de los sensores LiDAR en la última década dada su evolución tecnológica y su inclusión desde el mundo militar al civil, sus costes siguen siendo difíciles de llevar a producción a gran escala, por lo que incluir varias unidades no ayudaría.

Por otro lado, a pesar de contar con datos a distintas alturas, habría que recurrir a filtro de las medidas para eliminar puntos de planos no paralelos al plano del vehículo, ya que en caso de no hacerlo, se detectarían obstáculos falsos.

Por último, y más importante, el algoritmo de matching que se utilizará para ver la correlación entre medidas anteriormente tomadas y asignar una estimación de la pose, tendrá menos información con la que distinguir perfiles, por lo que será menos robusto y más propenso a fallar.

Dado los motivos anteriores, se decidió contar con un LiDAR 3D modelo HDL32e de Velodyne, con las características indicadas en capítulos anteriores.

A nivel de implementación, aprovechando la liberación del paquete hdl_graph_slam como open-source para ROS en el portal de Github [38], se procedió a su integración en la simulación, consistente en la creación de un lanzador para remapear tópicos y cargar todos los parámetros necesarios, así como los propios archivos yaml donde registrar el valor a asignar a cada uno de estos (ver anexo 4.a).

Comentando por encima los detalles más importantes para su ajuste correcto, se muestra la siguiente tabla de parámetros, el valor dado y su efecto/utilidad:

Tabla 7–1. Ganancias y saturaciones para controlador PID en velocidad

Parámetro	Valor	Comentario
use_distance_filter distance_near_threshold	True 5m	Permite no tener en cuenta las partículas medidas del propio chasis del vehículo
Outlier_removal_method radius_radius radius_min_neighbors	RADIUS 0.5m 2	Permite extraer mediciones erróneas filtrando en el radio indicado si no suficientes medidas cercanas a su entorno
registration_method	FAST_GICP	Uso del algoritmo GICP para la minimización de diferencia entre 2 nubes de puntos utilizando paralelización en varios núcleos
floor_pts_thresh	512	Umbral de puntos mínimos para aceptar un plano
Floor_normal_thresh	10º	Diferencia máxima aceptable del vector normal formado por el plano del triángulo de puntos frente al plano global
enable_gps	false	Añade la restricción de posición del GPS. No habilitada dado que experimentalmente demuestra un peor resultado debido al drift
Enable_imu_acceleration	false	Añade la restricción del ángulo estimado a partir del vector de fuerza gravedad medido por la IMU. Deshabilitado ya que supone movimiento cuasiestático, no despreciable para un vehículo.

Parámetro	Valor	Comentario
Enable_imu_orientation	false	Añade la restricción de yaw dado por la orientación magnética captada por la IMU. Deshabilitado ya que es muy sensible a perturbaciones externas, y por tanto no fiable.
distance_thresh	20m	Máxima distancia permitida para utilizar en el cierre de lazo
fitness_score_thresh	0.5	Correlación mínima entre nube de puntos para permitir el cierre de lazo
graph_update_interval	1.0s	Marca la frecuencia de actualización del grafo (en segundos). Es importante elegir un valor intermedio para no sobrecargar el cómputo, pero no perder demasiada información entre medias
map_cloud_update_interval	map_cloud_resolution	Resolución de la nube de puntos del mapa 3d generado

Para comprobar la robustez de este método de SLAM, se procede a realizar varias pruebas siguiendo el mismo recorrido por el entorno para todas ellas, pero variando la velocidad a la que se realiza.

Figura 7–2 Trazada en entorno virtual seguida para los experimentos de SLAM

A continuación, se muestran los resultados recopilados:

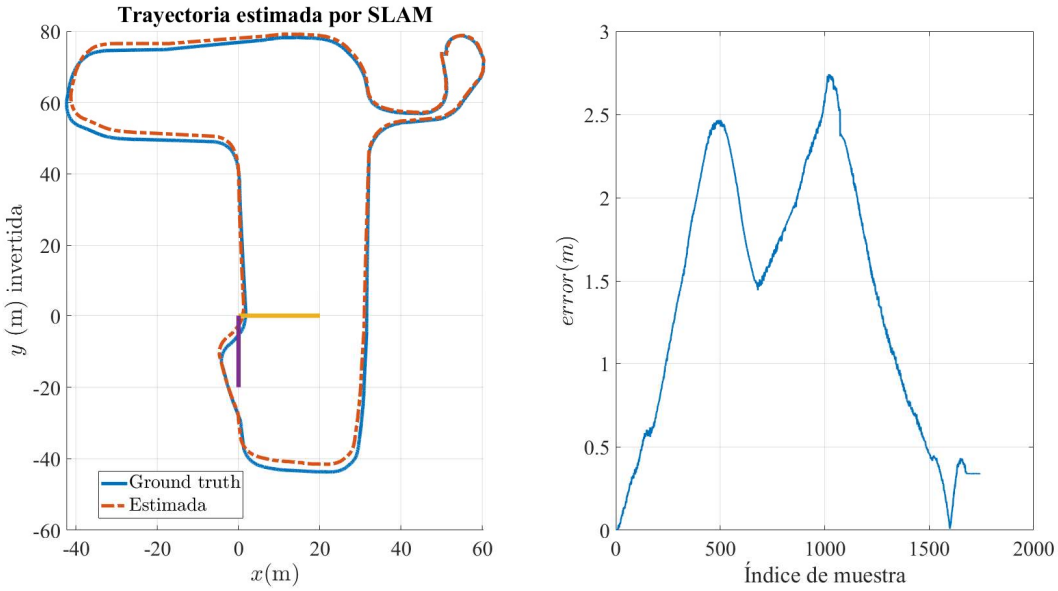


Figura 7–3 Comparación entre trazada real y estimada a 5 km/h de velocidad

En el primer experimento a baja velocidad, podemos ver como los resultados son bastantes buenos, ya que el error máximo en distancia cometido no supera los 3 metros. Por otro lado, hay un detalle interesante que puede extraerse a primeras a partir de ello. La localización es peor cuándo el vehículo atraviesa zonas más llanas y con menor obstáculos, cosa lógica ya que el número de puntos en la nube para comparar es menor y por tanto más difícil, la cantidad de información aportada por las medidas del sensor es menor.

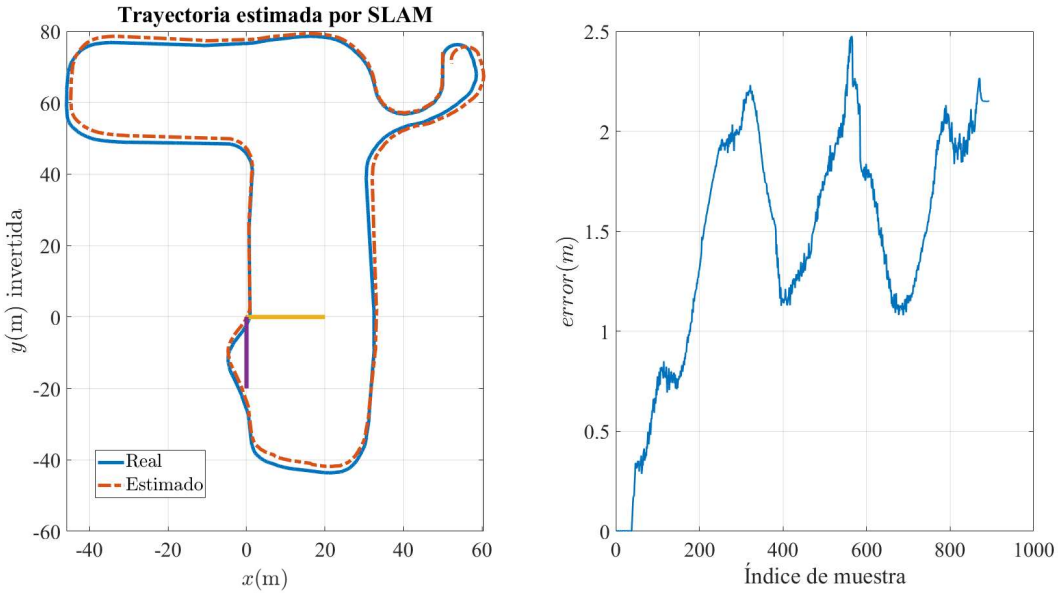


Figura 7–4 Comparación entre trazada real y estimada a 10 km/h de velocidad

A 10 km/h, podemos comprobar como efectivamente existe una tendencia de cometer mayor error en las zonas más vacías, las mismas que para el experimento anterior. No obstante, es posible extraer una temprana conclusión de estas nuevas gráficas: el error final no se ha visto reducido en la etapa final. Repitiendo el experimento, encuentro que esto no se debe a una casualidad, por lo que puedo concluir que a mayor velocidad el cierre de lazo es más complicado para el algoritmo de SLAM. Sin embargo, esto será confirmado o desmentido en el próximo experimento.

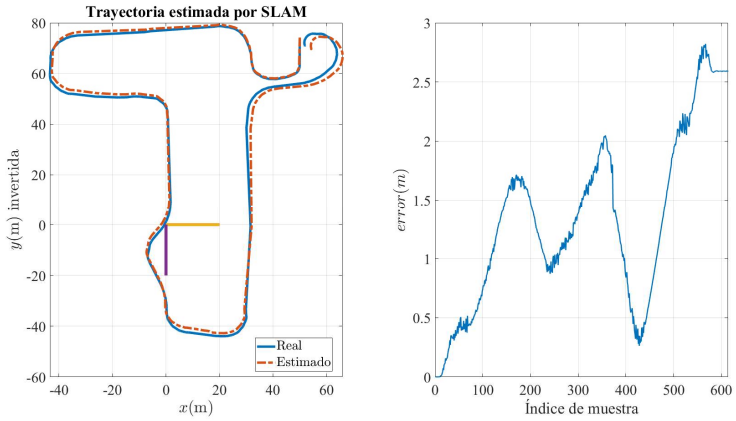


Figura 7–5 Comparación entre trazada real y estimada a 15 km/h de velocidad

Tal y como se adelantaba, podemos ver como el error aumenta en las mismas zonas que el experimento anterior. No obstante, la conclusión será tomada una vez se realicen todos los experimentos y se vean los resultados estadísticos.

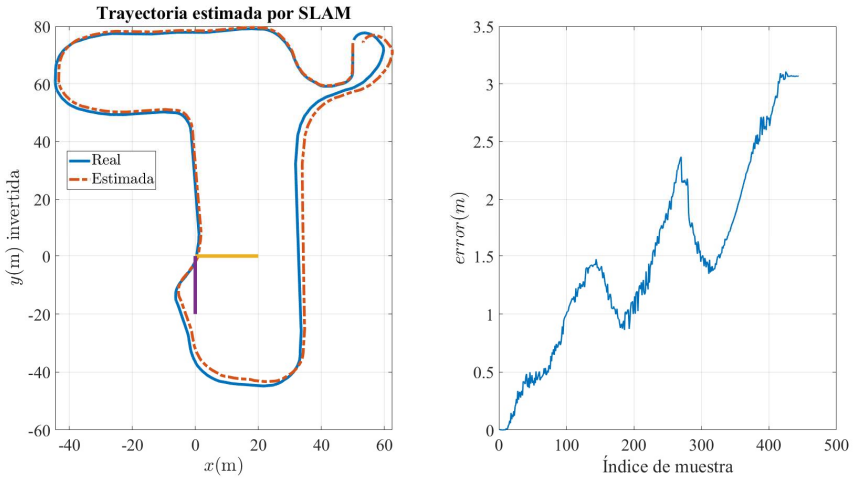


Figura 7–6 Comparación entre trazada real y estimada a 20 km/h de velocidad

Se puede observar resultados similares a 20 km/h, aunque a priori parece que el error final sigue aumentando al subir la velocidad, lo que hace pensar que la capacidad para reubicarse en el mapa ya escaneado empieza a disminuir.

Se realizan pruebas intermedias subiendo de 5 en 5 km/h la velocidad, hasta llegar a los 40 km/h, la máxima velocidad que permite el circuito trazar, obteniendo:

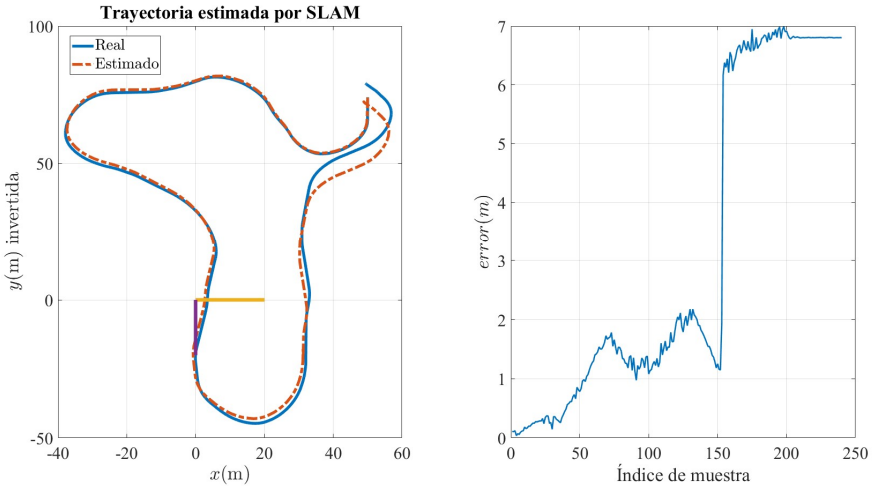


Figura 7–7 Comparación entre trazada real y estimada a 40 km/h de velocidad

De este último experimento, se puede confirmar viendo lo sucedido a partir de la muestra 150, que la conclusión anterior era la correcta. La técnica SLAM utilizada sigue funcionando de forma aceptable para alta velocidad (por entorno de ciudad). Sin embargo, en caso de perder la referencia ya no es capaz de relocalizarse dentro del mapa para disminuir de nuevo el error cometido.

Tabla 7–2. Resultados estadísticos de los experimentos

Velocidad media	Error medio	Desviación típica	Error máximo
5 km/h	1.38m	0.79m	2.74m
10 km/h	1.46m	0.60m	2.47m
15 km/h	1.32m	0.73m	2.8m
20 km/h	1.48m	0.81m	3.11m
25 km/h	1.23m	0.62m	2.63m
30 km/h	1.70m	0.84m	3.04m
35 km/h	4.76m	2.02m	6.57m
40 km/h	3.15m	2.75m	6.99m

A vista de la tabla 7-2, se puede afirmar que el error medio para velocidades de 25 km/h es admisible, con errores medios en torno al metro y medio y desviación típica menor al metro.

Sin embargo, podemos ver como a partir de 30 km/h, el error medio aumenta considerablemente por encima de 3m (lo cuál deja de ser admisible para la conducción autónoma). Esto puede deberse a la incapacidad para cerrar el bucle, ya que si observamos los errores máximos, se sobrepasan los 3 metros, con lo cual se supera el umbral máximo permitido y las nubes de puntos, a pesar de parecerse, quedan descartadas por la distancia entre estas. Una primera solución podría aumentar este umbral permitido, aunque esto afectaría al funcionamiento a baja velocidad (mayor carga computacional y por tanto obligaría a disminuir la frecuencia de actualización), por lo que se decide dejar en la configuración elegida, ya que funciona bien en la zona de trabajo deseada.

En cuanto a la nube de puntos generada, o ciertamente el mapa 3d resultante, de forma cualitativa no se aprecian diferencias a pesar de la velocidad. Sin embargo, para todos los casos el resultado es satisfactorio, como se puede ver en la siguiente imagen, obtenida a partir del archivo pgm generado:

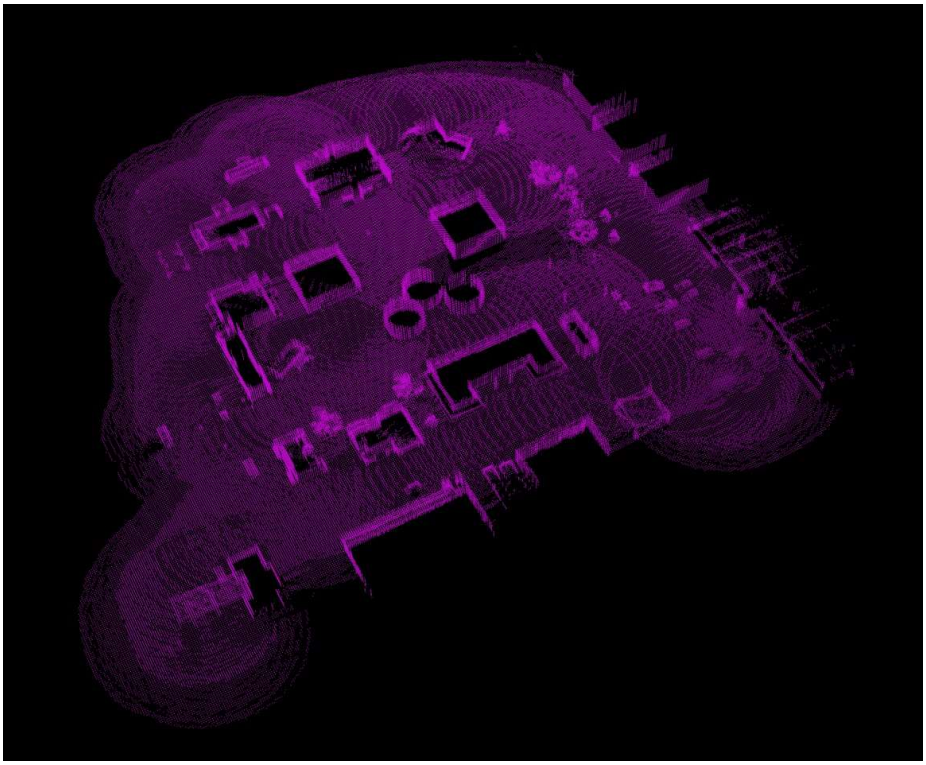


Figura 7–8 Mapa en 3 dimensiones resultante del escaneo

7.2 Planificación de trayectorias con Move Base

En ROS, existe un estándar para la navegación de bases móviles de robots, un framework en el que se toma la información de odometría, medición de los sensores y la pose objetivo a alcanzar, dando como resultado los comandos de velocidad necesarios a aplicar para llegar hasta el objetivo de forma segura.

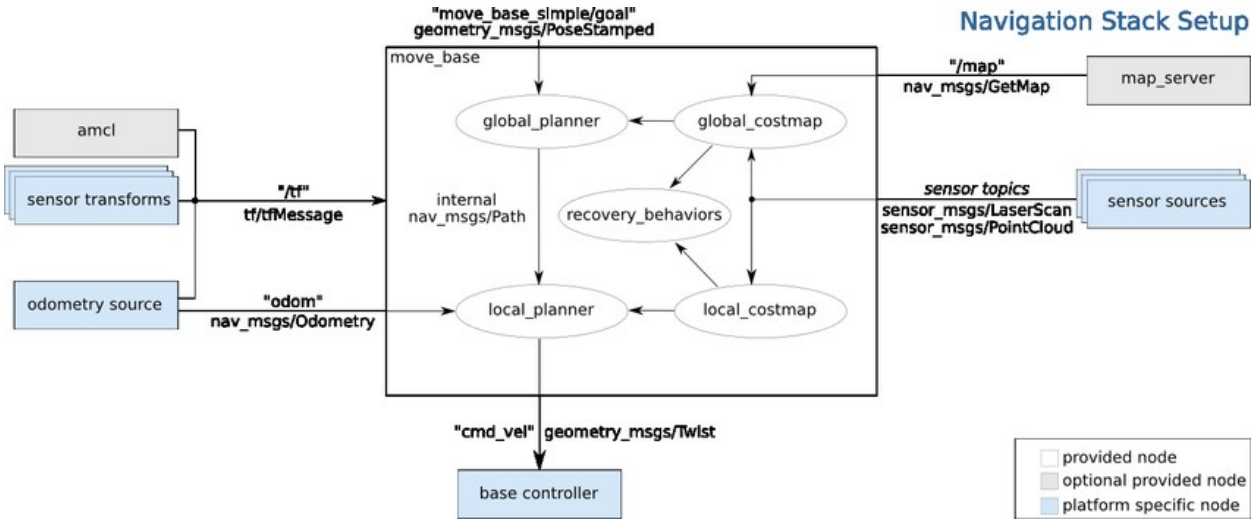


Figura 7–9 Pila de navegación estándar de ROS [39]

Como puede verse, la pila de navegación se encuentra dividida en diferentes módulos, donde cada uno cumple funciones independientes, pero se integran para trabajar en común en la navegación autónoma. A continuación, se explicará el objetivo que cumple cada uno específicamente en este proyecto:

- **global_costmap**: Contiene la información acerca de todo el entorno, que será usada para evitar obstáculos por el planificador. Al ser de gran tamaño, su resolución es pequeña (1m) para evitar que el coste computacional explote exponencialmente. No obstante, en teoría el tamaño del mapa podría ser infinito, lo que no sería manejable. Por tanto, se limita por defecto a 250x250 metros, lo cuál da capacidad para definir trayectorias largas (cubre todo el entorno de simulación).

  Se encuentra dividido en 2 capas:

  - Capa vóxel: Convierte la nube de puntos generada por graph slam como resultado del mapeo 3d en una cuadrícula tridimensional de cubos a los que asigna valor de ocupación en caso de tener partículas en su interior. A partir de ello, definiendo la altura mínima y máxima a la que se considera un obstáculo, se convierte a una matriz de celdas de ocupación 2d.
  - Capa de inflación: Por un lado, atendiendo al 'footprint' o huella del vehículo en el suelo que se le especifica, crea una zona segura que el frame principal ('base_link') del vehículo no debería atravesar nunca. Por otro lado, añade un efecto de gradiente alrededor de los obstáculos, de forma que el coste del planificador para generar caminos que pasen cerca de estos sea mayor, pero aún sea posible si no queda otra alternativa.

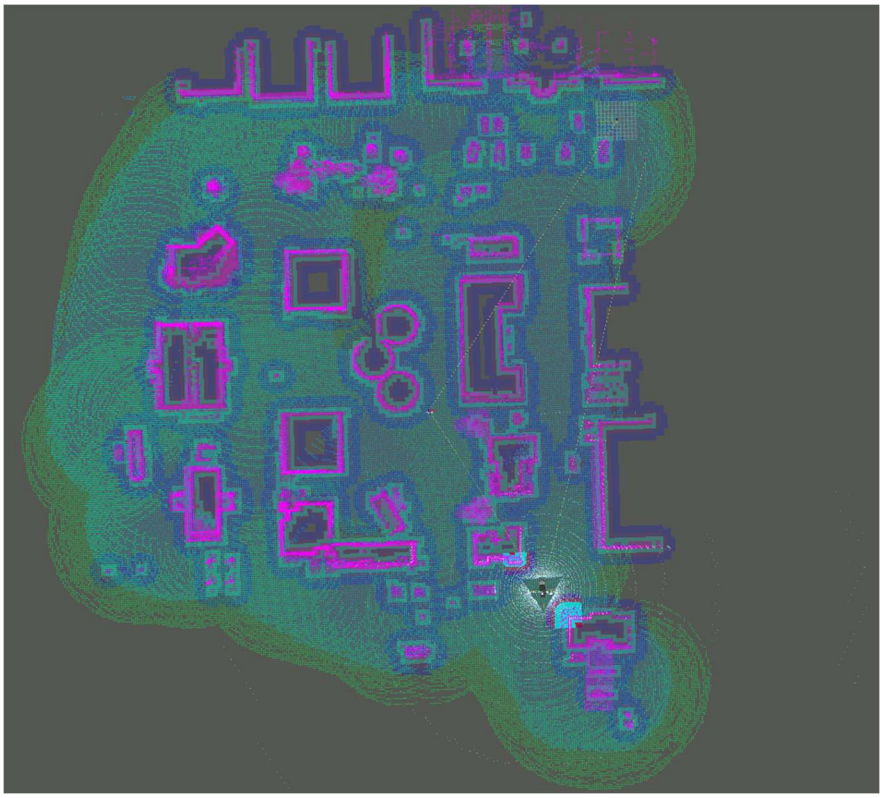


Figura 7–10 Mapa de coste global

- **local_costmap:** Funciona de manera similar al mapa de coste global, pero en este caso la capa vóxel genera la matriz de celdas ocupadas a partir de los datos del sensor LiDAR. Por otro lado, también se utilizan los sensores ultrasonidos laterales, frontal y trasero, para detectar posibles obstáculos que se encuentren en la zona muerta del escáner (aquellas que no son cubiertas ya que son tapadas por el chasis).

  Además, este mapa, a diferencia del anterior, no es estático, sino que se encuentra centrado en todo momento en el marco local del chasis del vehículo. Por otro lado, su tamaño es mucho más reducido (20x20m), por lo que permite una resolución mayor (0.2m) con la que el planfiicador local será más preciso evitando los obstáculos.

  Por último, este mapa permite la detección de obstáculos dinámicos, dado que permite ejecutarse y actualizarse a una frecuencia mayor (2 Hz frente a 0.5 Hz del global).

- **global_planner**: Planificador que dado el mapa de coste global y una pose meta, traza la ruta de puntos por donde debe pasar el robot para llegar hasta allí sin colisionar con los obstáculos conocidos. Permite elegir entre 2 algoritmos de búsqueda del camino más corto: Dijkstra/A*. Ambos son óptimos y completos, lo que quiere decir que encuentra la solución más corta posible y siempre aseguran una de existir. Sin embargo, se opta por el uso de A* en este proyecto ya que, al utilizar una función heurística para guiar la búsqueda, suele ser más rápido y por tanto permite aumentar la frecuencia del planificador, seteada a 0.2 Hz. Esta puede parecer demasiado lenta, sin embargo, no lo es realmente ya que se cuenta con el planificador local a un horizonte de 20 metros, lo cuál no se recorre en ese tiempo a las velocidades que se comandan.

- **local_planner**: Este planificador es mucho más complejo que el anterior, ya que no sólo debe plantear una ruta, sino dar las órdenes de comando en velocidad para que el vehículo alcance los ‘waypoint’ (puntos intermedios del camino) con la orientación deseada. Esto obliga a tener en cuenta la restricción no holónoma del vehículo, que obliga a respetar el radio de giro máximo del vehículo, así como anticipar el posible cambio de sentido que haya que realizar para cumplir con la orientación del punto objetivo.

A pesar de que el mapa de coste que deba manejar no contenga mucha información (es pequeño y con una resolución limitada), la complejidad del cálculo a realizar es alta, por lo que no permite una frecuencia de actualización muy elevada. Esto es un problema para la conducción, ya que se necesita una tasa alta para ser reactivos a cualquier imprevisto que surja.

Gracias a los parámetros de paralelismo para ejecutar múltiples hilos, esto mejora. No obstante, dado que la simulación maneja una gran cantidad de datos y consume ya de por sí muchos recursos, la máxima tasa conseguida ha sido de 5Hz. Esta cifra no es un problema en principio, aunque limita la velocidad máxima que se puede conseguir con la navegación autónoma (este problema puede desaparecer en función de la capacidad de cómputo del equipo que ejecute o si no se necesita simular nada y puede destinarse todos los recursos a la navegación).

Tras varios tests ajustando los parámetros teniendo en mente los principios anteriormente mencionados para conseguir el mejor funcionamiento, se obtienen los siguientes resultados:

Figura 7–11 Maniobra sencilla de aparcamiento autónoma

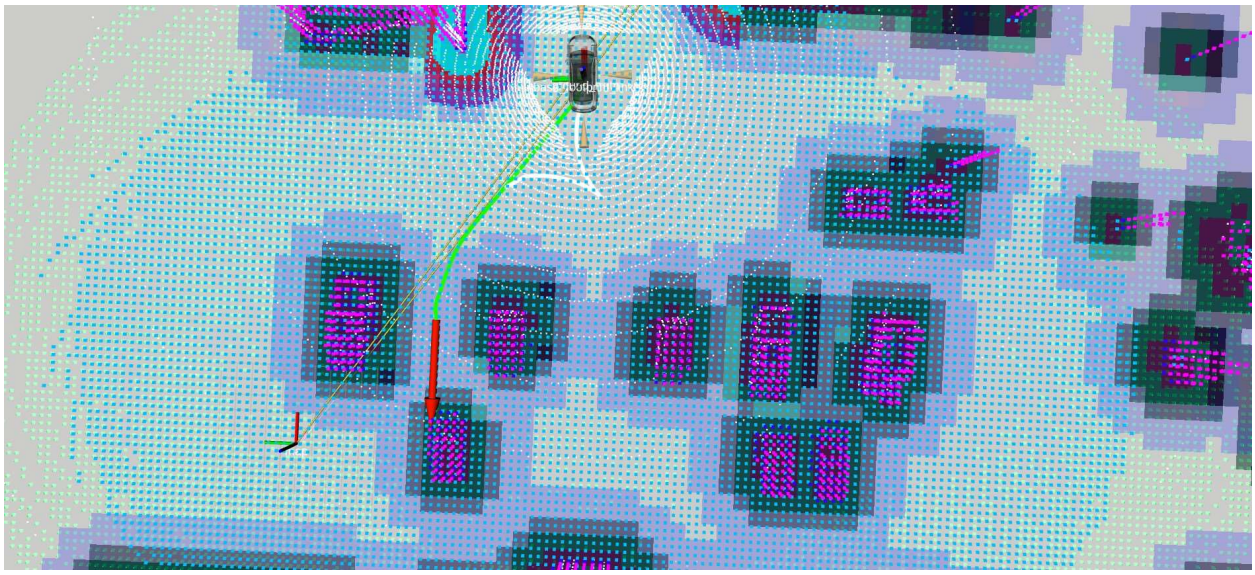


Figura 7–12 Maniobra de aparcamiento con cambio de sentido autónoma

Como se puede ver, el planificador local calculará la desviación necesaria respecto a la trayectoria global, para respetar el radio de giro mínimo del vehículo, además de la orientación final indicada. Además, en la medida de lo posible, el planificador local intentará avanzar hacia adelante dada la alta penalización que tiene puesta la marcha atrás.

Sin embargo, esto no es perfecto. Si el objetivo se encuentra muy cerca de la pose actual del vehículo, no será capaz de calcular la maniobra correcta para hacerlo, como puede verse en el siguiente ejemplo:

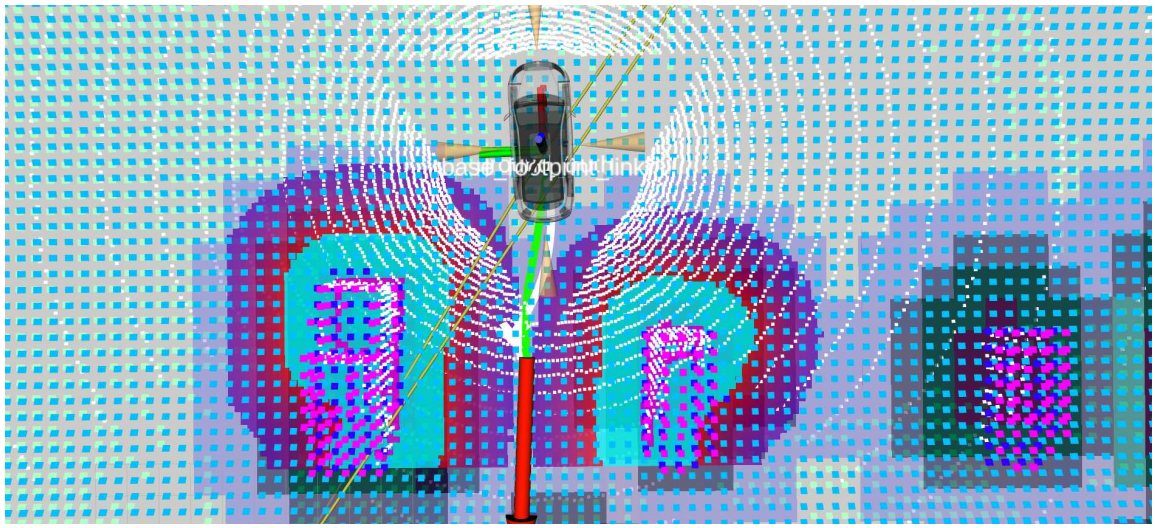


Figura 7–13 Maniobra compleja de aparcamiento fallida por la navegación autónoma

A pesar de ello, el planificador alcanzaría el timeout o número de intentos máximos y devolvería un mensaje de error diciendo que no es posible llevar a cabo la maniobra. En ese caso, el usuario deberá retomar el control para solventarlo.

8 GUÍA DE USUARIO PARA EL USO DE LA APLICACIÓN

La utilidad principal de este proyecto como aplicación final al usuario, es la de dotar de las herramientas de simulación necesarias para testear y recopilar información acerca del funcionamiento de un algoritmo en su etapa inicial, de forma cómodo y permitiendo esquivar fenómenos complejos del mundo real.

Por tanto, el código de este trabajo fin de grado queda liberado como open-source bajo la licencia GPL v3, lo que significa que cualquier persona puede usar el software, modificarlo y avanzar en ello, con la condición de no restringir el acceso a su futura versión.

Con el objetivo de acercar la robótica al mayor número de personas y facilitar el uso del proyecto, se ha intentado automatizar todo el proceso de dependencias, permisos y lanzamiento de la aplicación, para evitar el uso complejo de la terminal en la medida de lo posible.

Por ello, los pasos a seguir para el lanzamiento de la aplicación quedan limitados a los siguientes:

1) Clonar el repositorio de Github y sus submódulos (paquetes externos) :

   ```
   git clone –recurse-submodules https://github.com/RoboticsLeon/GoCar.git
   ```

2) Acceder a la carpeta, ejecutar el script de setup (ver anexo 5.a):

   ```
   cd GoCar && ./setup_env.sh
   ```

   Este script ha sido programado en bash e incluye sentencias condicionales, para dotar de mayor flexibilidad y customización al usuario:

   -Elegir intérprete preferido donde añadir las líneas de configuración.

   -Aceptar o no la lectura de eventos del teclado (es obligatoria para el correcto funcionamiento por teclado, pero no necesaria para usarse con mando).

3) Compilar código fuente:

   ```
   catkin_make -j8
   ```

4) Lanzamiento de la aplicación llamando al lanzador de ROS (ver anexo 5.b):

   ```
   roslaunch go_car_bringup start_simulation.launch control_type:=gamepad world_name:=city details:=performance rviz_visualization:=true;
   ```

   Este archivo sirve como lanzador centralizado de todos los nodos, tópicos y servicios necesarios para el uso de la aplicación, además de permitir la customización mediante argumentos:

   -**world_name**: Nombre del mundo virtual a cargar en simulación, actualmente sólo existe uno. Opciones: ‘city’ (por defecto).

   -**details**: Permite elegir la calidad de la geometría utilizada para el cálculo de colisión entre el vehículo y el resto de su entorno, la primera opción utiliza una malla simplificada del modelo 3d del vehículo mientras la segunda utiliza figuras primitivas para envolver la carrocería. Opciones: ‘quality’(por defecto) | ‘performance’.

   -**car_model**: Selecciona el modelo del vehículo, sólo altera el aspecto visual de la malla 3d cargada en el elemento visual. Opciones: ‘lincoln’ (por defecto).

   -**control_type**: Marca el tipo de control deseado, para su control manual o autónomo indicando el objetivo a alcanzar. Opciones: ‘keyboard’ (por defecto) | ‘gamepad’ | ‘autonomous’.

   -**rviz_visualization**: Dependiendo de su valor, se cargarán los parámetros para abrir y ajustar la interfaz de rviz para visualizar los datos de la simulación. Opciones: ‘true’ | ‘false’ (por defecto).

-**real_time_telemetry**: De activarse, abre una interfaz en plotjugger que permite al usuario visualizar gráficas en tiempo real sobre la telemetría del vehículo (torques aplicados, velocidades y ángulos de giro). Opciones: 'true' | 'false' (por defecto).

En su interior se realizan llamadas a otros lanzadores, cada uno especializado en una tarea concreta (tal y como se vio en el árbol jerárquico del repositorio de la figura 3.3, cada paquete resuelve un problema distinto y cada uno cuenta con un lanzador propio).

Esto es interesante, dado que, si alguien quiere desarrollar parte de este software, es modular, lo que quiere decir que no hay dependencias cruzadas entre los paquetes, más allá del remapeado necesario de tópicos en caso querer utilizarlo en otro proyecto.

Por último, comentar que al principio aparecían problemas por el orden de llamada de los programas. Estos son llamados de forma secuencial uno detrás de otro, pero sin esperar a que estos ya estén cargados. Esto provocaba que el simulador terminase cargando después que la ventana de visualización de Rviz o que el nodo de graph slam, provocando fallos y errores. Para solucionarlo, se ha propuesto llamar como paso intermedio a un script (ver anexo 5.c para más detalle) que ejecuta previamente un delay de los segundos especificados como parámetro. De esta forma se consigue hacer un lanzamiento ordenado y libre de errores.

5) Crear un launcher para escritorio para facilitar la ejecución con doble click (con la desventaja de no poder ajustar los parámetros):

```
[Desktop Entry]

Version=1.0

Name=GoCar Simulation

Comment=App launching simulation of GoCar

Exec=zsh -c 'source ~/.zshrc; ; roslaunch go_car_bringup start_simulation.launch
control_type:=gamepad world_name:=city details:=performance rviz_visualization:=true;'

Icon=/home/leon/Imágenes/GoCar_logo.png

Terminal=true

Type=Application
```

9 CONCLUSIONES

El resultado final del proyecto es satisfactorio. Se ha logrado cumplir con el objetivo principal de construir desde cero el modelado de un vehículo, adaptándolo por tanto a las distintas necesidades que han ido surgiendo en el transcurso. Además, el control desarrollado para este, a pesar de no ser perfecto, ha conseguido cumplir su función correctamente a pesar de los problemas experimentados con la no linealidad del rozamiento contra el suelo, con técnicas de control sencillas como son los controladores PIDs.

Por otro lado, el proyecto también ha abordado la emulación de sensores, involucrando el estudio de sensores reales comerciales para equiparar sus características y el modelado de su ruido como variable gaussiana. Inclusive la creación de un entorno virtual donde probar estos y la conducción del vehículo con variedad de situaciones.

Por otro lado, se ha trabajado en una interfaz humano-máquina cómoda y sencilla, que permita usar la aplicación a un usuario medio sin conocimiento de ROS o inclusive de programación, dotando de flexibilidad al usuario en cuanto a opciones a elegir para el control y la carga del simulador.

Además, se ha afrontado victoriosamente el problema de la carga computacional, muy presente al emular todos los sensores necesarios y el coste computacional de los algoritmos para navegar en un mundo de gran tamaño. Principalmente, la solución adoptada ha pasado por la paralelización de los trabajos para repartir la carga en los distintos núcleos del procesador, y la optimización gracias a la programación compilada en C++, frente a la alternativa en Python, que al interpretarse en tiempo de ejecución está más limitada.

Por último, se ha integrado conjuntamente técnicas de SLAM junto a la pila de navegación estándar de ROS, permitiendo realizar múltiples tareas a la vez: mapeado 3d del entorno, creación de mapas de coste y planificación y seguimiento de trayectorias basadas en mapas de coste. En un principio, su funcionamiento no era el esperado, ya que no se conseguían cumplir unas especificaciones realistas para la conducción. Sin embargo, tras la comprensión del efecto de su multitud de parámetros y su posterior ajuste, se consiguen buenos resultados.

No obstante, existe mucho trabajo por delante para proseguir avanzando este proyecto. La navegación autónoma aún no es fiable, ya que falla en ocasiones al forzar ciertas condiciones como el movimiento dinámico de obstáculos, zonas con pocas referencias para su escaneo donde la localización empieza a fallar.

Por otro lado, existen restricciones de la vida real que no se están teniendo en cuenta, tales como las normas de circulación, por lo que se requeriría trabajar en sistemas de percepción para la detección de carriles, incluyendo la posición de las líneas en el mapa de coste local, detección y clasificación de señales verticales como semáforos y stops…

Además, es muy importante tener en cuenta que se trata de una simulación, que a pesar de intentar no ser ideal del todo, y replicar fenómenos como el ruido gaussiano de los sensores y el deslizamiento con el suelo, las pruebas en la realidad podrían sufrir mayor dificultad: firme en mal estado, ruido no gaussiano, mayor coste computacional al aumentar la cantidad de datos generados por los sensores, limitaciones de cómputo para sistemas embebidos (aunque esto podría resolverse con la descentralización en la nube), etc.

En resumen, este proyecto termina con una versión inicial de una aplicación de utilidad a posibles usuarios, tales como estudiantes del mundo académico para levantar el interés en el mundo académico, investigadores para tener un entorno donde hacer pruebas preliminares sencillas donde estimar el resultado de sus implementaciones, o desarrolladores que puedan seguir construyendo encima para lograr más y mejores funcionalidades.

ANEXO

En este apartado de anexo, se adjunta el código de los archivos más importantes del proyecto, mencionados en el documento.

1. Modelo en Xacro

a) go_car_model.xacro:

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

<robot name="go_car"
  xmlns:xacro="go_car">

  <xacro:include filename="$(find go_car_description)/urdf/include/params.xacro"/>
  <xacro:include filename="$(find go_car_description)/urdf/include/snippets.xacro"/>
  <xacro:include filename="$(find go_car_description)/urdf/include/links.xacro"/>
  <xacro:include filename="$(find go_car_description)/urdf/include/sensors.xacro"/>

  <!-- First tf level -->
  <joint name="base_to_chassis" type="fixed">
    <parent link="base_footprint_link"/>
    <child link="chassis_link"/>
    <origin xyz="0.0 0.0 0.84" rpy="0.0 0.0 0.0"/>
  </joint>
  <joint name="chassis_to_left_steering_column" type="revolute">
    <parent link="chassis_link"/>
    <child link="left_steering_column_link"/>
    <axis xyz="0.0 0.0 1"/>
    <origin xyz="${distanceX_CM_to_front_wheel_axis} ${distanceY_CM_to_wheel_center}
${distanceZ_CM_to_wheel_center}" rpy="0.0 0.0 0.0"/>
    <limit lower="${-6/9*pi/2}" upper="${6/9*pi/2}" effort="100000" velocity="100000"/>
  </joint>
  <joint name="chassis_to_right_steering_column" type="revolute">
    <parent link="chassis_link"/>
    <child link="right_steering_column_link"/>
    <axis xyz="0.0 0.0 1"/>
    <origin xyz="${distanceX_CM_to_front_wheel_axis} ${-distanceY_CM_to_wheel_center}
${distanceZ_CM_to_wheel_center}" rpy="0.0 0.0 0.0"/>
    <limit lower="${-6/9*pi/2}" upper="${6/9*pi/2}" effort="100000" velocity="100000"/>
  </joint>

  <!-- Second tf level -->
  <xacro:chassis_link details="$(arg details)"/>
  <joint name="chassis_to_rear_left_wheel" type="continuous">
    <parent link="chassis_link"/>
    <child link="rear_left_wheel_link"/>
    <origin xyz="${distanceX_CM_to_rear_wheel_axis} ${distanceY_CM_to_wheel_center}
${distanceZ_CM_to_wheel_center}" rpy="0.0 0.0 0.0"/>
```

```
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="100000" velocity="100000"/>
  </joint>
  <joint name="chassis_to_rear_right_wheel" type="continuous">
    <parent link="chassis_link"/>
    <child link="rear_right_wheel_link"/>
    <origin xyz="${distanceX_CM_to_rear_wheel_axis} ${-distanceY_CM_to_wheel_center}
${distanceZ_CM_to_wheel_center}" rpy="0.0 0.0 0.0"/>
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="100000" velocity="100000"/>
  </joint>

  <xacro:steering_column_link RL="left"/>
  <xacro:steering_column_link RL="right"/>
  <joint name="steering_column_to_front_left_wheel" type="continuous">
    <parent link="left_steering_column_link"/>
    <child link="front_left_wheel_link"/>
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="100000" velocity="100000"/>
  </joint>
  <joint name="steering_column_to_front_right_wheel" type="continuous">
    <parent link="right_steering_column_link"/>
    <child link="front_right_wheel_link"/>
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="100000" velocity="100000"/>
  </joint>

  <!-- Third tf level -->
  <xacro:wheel_link FR="rear" RL="left"/>
  <xacro:wheel_link FR="rear" RL="right"/>
  <xacro:wheel_link FR="front" RL="left"/>
  <xacro:wheel_link FR="front" RL="right"/>
  <joint name="chassis_to_camera" type="fixed">
    <parent link="chassis_link"/>
    <child link="camera_link"/>
    <origin xyz="0.5 0.0 ${camera_height}" rpy="0.0 0.0 0.0"/>
  </joint>
  <joint name="chassis_to_lidar" type="fixed">
    <parent link="chassis_link"/>
    <child link="lidar_link"/>
    <origin xyz="0.0 0.0 ${lidar_height}" rpy="0.0 0.0 0.0"/>
  </joint>
  <joint name="chassis_to_back_ultrasonic" type="fixed">
    <parent link="chassis_link"/>
    <child link="back_ultrasonic_link"/>
    <origin xyz="${-chassis_length/2} 0.0 0.0" rpy="0.0 0.0 ${pi}"/>
  </joint>
  <joint name="chassis_to_right_ultrasonic" type="fixed">
    <parent link="chassis_link"/>
    <child link="right_ultrasonic_link"/>
    <origin xyz="0.0 ${-chassis_width/2} 0.0" rpy="0.0 0.0 ${-pi/2}"/>
```

```
  </joint>
  <joint name="chassis_to_left_ultrasonic" type="fixed">
    <parent link="chassis_link"/>
    <child link="left_ultrasonic_link"/>
    <origin xyz="0.0 ${chassis_width/2} 0.0" rpy="0.0 0.0 ${pi/2}"/>
  </joint>
  <joint name="chassis_to_front_ultrasonic" type="fixed">
    <parent link="chassis_link"/>
    <child link="front_ultrasonic_link"/>
    <origin xyz="${chassis_length/2+0.35} 0.0 0.0" rpy="0.0 0.0 0.0"/>
  </joint>

  <!-- Gazebo -->
  <xacro:include filename="$(find go_car_description)/urdf/include/gazebo.xacro"/>

</robot>
```

b) *params.xacro:*

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

<robot xmlns:xacro="macros_xacro">

    <xacro:arg name="details" default="quality"/>
    <xacro:arg name="car_model" default="lincoln"/>


    <xacro:property name="chassis_height" value="1.3"/>
    <xacro:property name="chassis_length" value="5"/>
    <xacro:property name="chassis_width" value="2.2"/>
    <xacro:property name="chassis_mass" value="2050"/>
    <xacro:property name="chassis_mass_ixx" value="647"/>
    <xacro:property name="chassis_mass_iyy" value="3291"/>
    <xacro:property name="chassis_mass_izz" value="3360"/>
    <xacro:property name="chassis_mass_ixz" value="-31"/>

    <xacro:property name="tyre_radius" value="0.356"/>
    <xacro:property name="tyre_mass" value="40"/>
    <xacro:property name="tyre_width" value="0.229"/>
    <xacro:property name="tyre_mass_ixx" value="2.0"/>
    <xacro:property name="tyre_mass_iyy" value="2.0"/>
    <xacro:property name="tyre_mass_izz" value="2.0"/>

    <xacro:property name="steering_column_weight" value="2.0"/>
    <xacro:property name="steering_column_radius" value="0.05"/>
    <xacro:property name="steering_column_length" value="0.5"/>
    <xacro:property name="distanceX_CM_to_rear_wheel_axis" value="-1.15"/>
    <xacro:property name="distanceX_CM_to_front_wheel_axis" value="1.65"/>
    <xacro:property name="distanceY_CM_to_wheel_center" value="0.8"/>
    <xacro:property name="distanceZ_CM_to_wheel_center" value="-0.5"/>
```

```
    <xacro:property name="wheelbase" value="${distanceX_CM_to_front_wheel_axis-
distanceX_CM_to_rear_wheel_axis}"/>
    <xacro:property name="axle_track" value="${2*distanceY_CM_to_wheel_axis}"/>

    <!-- Sensors location setup params -->
    <xacro:property name="lidar_height" value="1.0"/>
    <xacro:property name="camera_height" value="0.625"/>

</robot>
```

c) *links.xacro:*

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

<robot xmlns:xacro="properties_xacro">

    <link name="base_footprint_link"/>

    <xacro:macro name="chassis_link" params="details">
        <link name="chassis_link">
            <inertial>
                <mass value="${chassis_mass}"/>
                <inertia ixx="${chassis_mass_ixx}" ixy="0.0" ixz="${chassis_mass_ixz}"
iyy="${chassis_mass_iyy}" iyz="0.0" izz="${chassis_mass_izz}"/>
            </inertial>
            <collision>
                <xacro:if value="${details == 'quality'}">
                    <geometry>
                        <mesh
filename="package://go_car_description/meshes/CarModel_reduced_for_collision.dae"/>
                    </geometry>
                </xacro:if>
                <xacro:if value="${details == 'performance'}">
                    <origin xyz="0.25 0.0 0.2" rpy="0.0 0.0 0.0"/>
                    <geometry>
                        <box size="${chassis_length} ${chassis_width} ${chassis_height}"/>
                    </geometry>
                </xacro:if>
            </collision>
            <visual>
                <geometry>
                    <mesh filename="package://go_car_description/meshes/car.dae"/>
                </geometry>
            </visual>
        </link>
    </xacro:macro>

    <xacro:macro name="steering_column_link" params="RL">        <!-- RL:'right','left' --
>
```

```
        <link name="${RL}_steering_column_link">
            <inertial>
                <mass value="${steering_column_weight}"/>
                <xacro:inertial_tensor_cylinder M="${steering_column_weight}"
R="${steering_column_radius}" L="${steering_column_length}"/>
            </inertial>
        </link>
    </xacro:macro>

    <xacro:macro name="wheel_link" params="FR RL">        <!-- FR:'front','rear' |
RL:'right','left' -->
        <link name="${FR}_${RL}_wheel_link">
            <inertial>
                <origin xyz="0.0 0.0 0.0" rpy="${-pi/2} 0.0 0.0"/>
                <mass value="${tyre_mass}"/>
                <xacro:inertial_tensor_cylinder M="${tyre_mass}" R="${tyre_radius}"
L="${tyre_width}"/>
            </inertial>
            <collision>
                <origin xyz="0.0 0.0 0.0" rpy="${-pi/2} 0.0 0.0"/>
                <geometry>
                    <cylinder radius="${tyre_radius}" length="${tyre_width}"/>
                </geometry>
            </collision>
            <visual>
                <xacro:if value="${RL == 'left'}">
                    <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 ${pi/2}"/>
                </xacro:if>
                <xacro:if value="${RL == 'right'}">
                    <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 ${-pi/2}"/>
                </xacro:if>
                <geometry>
                    <mesh filename="package://go_car_description/meshes/wheel.dae"/>
                </geometry>
            </visual>
        </link>
    </xacro:macro>

    <link name="camera_link">
        <inertial>
            <mass value="0.25"/>
            <inertia ixx="1e-3" ixy="0" ixz="0" iyy="1e-3" iyz="0" izz="1e-3"/>
        </inertial>
        <visual>
            <origin xyz="0.0 -0.2 0.0" rpy="0.0 0.0 ${pi/2}"/>
            <geometry>
                <mesh
filename="package://go_car_description/meshes/intel_realsense_camera.dae" scale="4.0 4.0
4.0"/>
            </geometry>
        </visual>
```

```
    </link>

    <link name="lidar_link">
        <inertial>
            <mass value="0.25"/>
            <inertia ixx="1e-3" ixy="0" ixz="0" iyy="1e-3" iyz="0" izz="1e-3"/>
        </inertial>
        <visual>
            <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 ${pi/2}"/>
            <geometry>
                <mesh filename="package://go_car_description/meshes/hdl32e.dae" scale="2.5
2.5 2.5"/>
            </geometry>
        </visual>
        <visual>
            <origin xyz="0.0 0.0 -0.4" rpy="0.0 0.0 0.0"/>
            <geometry>
                <mesh filename="package://go_car_description/meshes/hdl32e_base.dae"
scale="2.5 2.5 7.5"/>
            </geometry>
        </visual>
    </link>

    <link name="back_ultrasonic_link"/>
    <link name="right_ultrasonic_link"/>
    <link name="left_ultrasonic_link"/>
    <link name="front_ultrasonic_link"/>

</robot>
```

d) *snippets.xacro:*

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

<robot xmlns:xacro="macros_xacro">

  <xacro:macro name="inertial_tensor_cylinder" params="M R L">
    <inertia ixx="${1/12*M*(3*R**2+L**2)}" ixy="0.0" ixz="0.0"
iyy="${1/12*M*(3*R**2+L**2)}" iyz="0.0" izz="${1/2*M*R**2}"/>
  </xacro:macro>

</robot>
```

e) *sensors.xacro:*

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

<robot xmlns:xacro="gazebo_sensors">

    <gazebo reference="camera_link">
        <sensor type="camera" name="camera">
            <update_rate>30.0</update_rate>
            <camera name="head">
                <horizontal_fov>1.3962634</horizontal_fov>
                <image>
                    <width>800</width>
                    <height>800</height>
                    <format>R8G8B8</format>
                </image>
                <clip>
                    <near>0.02</near>
                    <far>300</far>
                </clip>
                <noise>
                    <type>gaussian</type>
                    <!-- Noise is sampled independently per pixel on each frame.
                That pixel's noise value is added to each of its color
                channels, which at that point lie in the range [0,1]. -->
                    <mean>0.0</mean>
                    <stddev>0.007</stddev>
                </noise>
            </camera>
            <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
                <alwaysOn>true</alwaysOn>
                <updateRate>0.0</updateRate>
                <cameraName>go_car/front_height_camera</cameraName>
                <imageTopicName>image_raw</imageTopicName>
                <cameraInfoTopicName>camera_info</cameraInfoTopicName>
                <frameName>camera_link</frameName>
                <hackBaseline>0.07</hackBaseline>
                <distortionK1>0.0</distortionK1>
                <distortionK2>0.0</distortionK2>
                <distortionK3>0.0</distortionK3>
                <distortionT1>0.0</distortionT1>
                <distortionT2>0.0</distortionT2>
            </plugin>
        </sensor>
    </gazebo>

    <gazebo reference="lidar_link">
        <sensor type="gpu_ray" name="HDL32E">
            <pose>0 0 0 0 0 0</pose>
            <visualize>false</visualize>
```

```
            <update_rate>5</update_rate>
            <ray>
                <scan>
                    <horizontal>
                        <samples>300</samples>                          <!-- Real sensor
will have up to 1200 -->
                        <resolution>1</resolution>
                        <min_angle>${-pi}</min_angle>
                        <max_angle>${pi}</max_angle>
                    </horizontal>
                    <vertical>
                        <samples>40</samples>
                        <resolution>1</resolution>
                        <min_angle>${-30*pi/180}</min_angle>
                        <max_angle>${10*pi/180}</max_angle>
                    </vertical>
                </scan>
                <range>
                    <min>3.0</min>
                    <max>100.0</max>
                    <resolution>0.005</resolution>
                </range>
                <noise>
                    <type>gaussian</type>
                    <mean>0.0</mean>
                    <stddev>0.02</stddev>
                </noise>
            </ray>
            <plugin name="gazebo_ros_laser_controller"
filename="libgazebo_ros_velodyne_gpu_laser.so">
                <topicName>velodyne_points</topicName>
                <frameName>lidar_link</frameName>
                <organize_cloud>true</organize_cloud>
                <min_range>3.0</min_range>
                <max_range>100.0</max_range>
                <gaussianNoise>0.02</gaussianNoise>
            </plugin>
        </sensor>
    </gazebo>

    <gazebo>
        <plugin name="navsat_sensor" filename="libhector_gazebo_ros_gps.so">
            <updateRate>1.0</updateRate>
            <bodyName>chassis_link</bodyName>
            <topicName>/gps/coordinates</topicName>
            <velocityTopicName>/gps/velocity</velocityTopicName>
            <referenceLatitude>37.38283</referenceLatitude>
            <referenceLongitude>-5.97317</referenceLongitude>
            <drift>3.0 3.0 3.0</drift>
            <gaussianNoise>0.05 0.05 0.05</gaussianNoise>
            <velocityDrift>0 0 0</velocityDrift>
```

```
        <velocityGaussianNoise>0.05 0.05 0.05</velocityGaussianNoise>
    </plugin>
</gazebo>


<!-- <gazebo reference="2d_lidar_link">
    <sensor type="lidar" name="liDAR_2d">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>20</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>720</samples>
                    <resolution>1</resolution>
                    <min_angle>-1.570796</min_angle>
                    <max_angle>1.570796</max_angle>
                </horizontal>
            </scan>
            <range>
                <min>1.0</min>
                <max>250.0</max>
                <resolution>0.01</resolution>
            </range>
            <noise>
                <type>gaussian</type>
                <mean>0.0</mean>
                <stddev>0.01</stddev>
            </noise>
        </ray>
        <plugin name="intel_L515_lidar" filename="libgazebo_ros_laser.so">
            <topicName>/laser/scan</topicName>
            <frameName>lidar_link</frameName>
        </plugin>
    </sensor>
</gazebo> -->

<gazebo reference="chassis_link">
    <gravity>true</gravity>
    <sensor name="imu_sensor" type="imu">
        <always_on>true</always_on>
        <update_rate>100</update_rate>
        <visualize>false</visualize>
        <topic>__default_topic__</topic>
        <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
            <topicName>imu</topicName>
            <bodyName>chassis_link</bodyName>
            <updateRateHZ>10.0</updateRateHZ>
            <gaussianNoise>0.0002</gaussianNoise>
            <xyzOffset>0 0 0</xyzOffset>
            <rpyOffset>0 0 0</rpyOffset>
```

```
            <frameName>chassis_link</frameName>
        </plugin>
        <pose>0 0 0 0 0 0</pose>
    </sensor>
</gazebo>

<gazebo reference="front_ultrasonic_link">
    <sensor type="ray" name="TeraRanger">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>50</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </horizontal>
                <vertical>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </vertical>
            </scan>
            <range>
                <min>0.01</min>
                <max>2</max>
                <resolution>0.02</resolution>
            </range>
        </ray>
        <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
            <gaussianNoise>0.005</gaussianNoise>
            <alwaysOn>true</alwaysOn>
            <updateRate>50</updateRate>
            <topicName>range_sensor/front</topicName>
            <frameName>front_ultrasonic_link</frameName>
            <radiation>INFRARED</radiation>
            <fov>0.2967</fov>
        </plugin>
    </sensor>
</gazebo>

<gazebo reference="back_ultrasonic_link">
    <sensor type="ray" name="TeraRanger">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>50</update_rate>
        <ray>
            <scan>
```

```
                <horizontal>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </horizontal>
                <vertical>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </vertical>
            </scan>
            <range>
                <min>0.01</min>
                <max>2</max>
                <resolution>0.02</resolution>
            </range>
        </ray>
        <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
            <gaussianNoise>0.005</gaussianNoise>
            <alwaysOn>true</alwaysOn>
            <updateRate>50</updateRate>
            <topicName>range_sensor/back</topicName>
            <frameName>back_ultrasonic_link</frameName>
            <radiation>INFRARED</radiation>
            <fov>0.2967</fov>
        </plugin>
    </sensor>
</gazebo>

<gazebo reference="right_ultrasonic_link">
    <sensor type="ray" name="TeraRanger">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>50</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </horizontal>
                <vertical>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </vertical>
            </scan>
```

```
            <range>
                <min>0.01</min>
                <max>2</max>
                <resolution>0.02</resolution>
            </range>
        </ray>
        <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
            <gaussianNoise>0.005</gaussianNoise>
            <alwaysOn>true</alwaysOn>
            <updateRate>50</updateRate>
            <topicName>range_sensor/right</topicName>
            <frameName>right_ultrasonic_link</frameName>
            <radiation>INFRARED</radiation>
            <fov>0.2967</fov>
        </plugin>
    </sensor>
</gazebo>

<gazebo reference="left_ultrasonic_link">
    <sensor type="ray" name="TeraRanger">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>50</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </horizontal>
                <vertical>
                    <samples>10</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.14835</min_angle>
                    <max_angle>0.14835</max_angle>
                </vertical>
            </scan>
            <range>
                <min>0.01</min>
                <max>2</max>
                <resolution>0.02</resolution>
            </range>
        </ray>
        <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
            <gaussianNoise>0.005</gaussianNoise>
            <alwaysOn>true</alwaysOn>
            <updateRate>50</updateRate>
            <topicName>range_sensor/left</topicName>
            <frameName>left_ultrasonic_link</frameName>
            <radiation>INFRARED</radiation>
```

```
                <fov>0.2967</fov>
            </plugin>
        </sensor>
    </gazebo>

</robot>
```

f) *gazebo.xacro:*

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

<robot xmlns:xacro="gazebo_properties">

  <!-- Friction parameters -->
  <gazebo reference="rear_right_wheel_link">
    <mu1>1</mu1>
    <mu2>1</mu2>
    <kp>10000000</kp>
    <kd>1</kd>
    <minDepth>0.01</minDepth>
    <maxVel>10.0</maxVel>
  </gazebo>

  <gazebo reference="rear_left_wheel_link">
    <mu1>1</mu1>
    <mu2>1</mu2>
    <kp>10000000</kp>
    <kd>1</kd>
    <minDepth>0.01</minDepth>
    <maxVel>10.0</maxVel>
  </gazebo>

  <gazebo reference="front_right_wheel_link">
    <mu1>1</mu1>
    <mu2>1</mu2>
    <kp>10000000</kp>
    <kd>1</kd>
    <minDepth>0.01</minDepth>
    <maxVel>100.0</maxVel>
  </gazebo>

  <gazebo reference="front_left_wheel_link">
    <mu1>1</mu1>
    <mu2>1</mu2>
    <kp>10000000</kp>
    <kd>1</kd>
    <minDepth>0.01</minDepth>
    <maxVel>100.0</maxVel>
  </gazebo>

  <gazebo reference="front_left_wheel_link">
```

```
    <mu1>1</mu1>
    <mu2>1</mu2>
    <kp>10000000</kp>
    <kd>1</kd>
    <minDepth>0.01</minDepth>
    <maxVel>100.0</maxVel>
  </gazebo>

  <!-- Suspension parameters -->
  <!-- <gazebo reference="right_steering_column_link">
    <mu1>0.6</mu1>
    <mu2>0.5</mu2>
    <kp>1000000</kp>
    <kd>100</kd>
    <minDepth>0.01</minDepth>
    <maxVel>10.0</maxVel>
  </gazebo> -->

  <!-- <gazebo reference="left_steering_column_link">
    <mu1>0.6</mu1>
    <mu2>0.5</mu2>
    <kp>1000000</kp>
    <kd>100</kd>
    <minDepth>0.01</minDepth>
    <maxVel>10.0</maxVel>
  </gazebo> -->

  <!-- Plugins -->
  <gazebo>
    <plugin name="ackermann_control_plugin" filename="libackermann_control_plugin.so"/>
    <plugin filename="libgazebo_ros_joint_state_publisher.so"
name="joint_state_publisher">
      <jointName>chassis_to_rear_left_wheel, chassis_to_rear_right_wheel,
chassis_to_left_steering_column, steering_column_to_front_left_wheel,
chassis_to_right_steering_column, steering_column_to_front_right_wheel</jointName>
      <updateRate>50.0</updateRate>
      <robotNamespace>/$(arg robot_name)</robotNamespace>
      <alwaysOn>true</alwaysOn>
    </plugin>
    <plugin name="p3d_base_controller" filename="libgazebo_ros_p3d.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>50.0</updateRate>
      <bodyName>base_footprint_link</bodyName>
      <topicName>ground_truth_pose</topicName>
      <gaussianNoise>0.0</gaussianNoise>
      <frameName>world</frameName>
      <xyzOffsets>0 0 0</xyzOffsets>
      <rpyOffsets>0 0 0</rpyOffsets>
    </plugin>
  </gazebo>
</robot>
```

2. Drivers para interfaz HMI

a) *Gamepad_driver:*

- *gamepad_driver.hpp:*

```
#pragma once

#include <iostream>
#include <stdexcept>
#include <utility>
#include <vector>

namespace controller_adapter {

constexpr double MAX_STEERING_ANGLE = 0.8;
constexpr double MAX_SPEED = 20.0;

enum class Gears {
  Reverse = -1,
  Neutral = 0,
  First = 1,
  Second = 2,
  Third = 3,
  Fourth = 4,
};

// Addition operator overload
void operator++(Gears &gear, int);

// Subtraction operator overload
void operator--(Gears &gear, int);

std::pair<double, double>
controller_input_processing(const std::vector<float> &axes,
                            const std::vector<int> &buttons);

} // namespace controller_adapter
```

- *gamepad_driver.cpp:*

```
#include "gamepad_driver.hpp"

namespace controller_adapter {

void operator++(Gears &gear, int) {
  int newGear;
  if (gear != Gears::Fourth) {
    newGear = static_cast<int>(gear) + 1;
  } else {
    throw std::runtime_error("[controller_adapter]: Invalid gear value");
```

```
  }
  gear = static_cast<Gears>(newGear);
}

void operator--(Gears &gear, int) {
  int newGear;
  if (gear != Gears::Reverse) {
    newGear = static_cast<int>(gear) - 1;
  } else {
    throw std::runtime_error("[controller_adapter]: Invalid gear value");
  }
  gear = static_cast<Gears>(newGear);
}

std::pair<double, double>
controller_input_processing(const std::vector<float> &axes,
                            const std::vector<int> &buttons) {
  const double steering_percentage = axes[0];
  const double throttle_percentage = axes[4];
  static Gears current_gear{Gears::Neutral};
  const double desired_steering_angle =
      steering_percentage * MAX_STEERING_ANGLE;
  double desired_speed = 0.0;

  if (static_cast<double>(current_gear) >= 0) {
    if (current_gear == Gears::Neutral)
      desired_speed = 0.0;
    else {
      desired_speed =
          (throttle_percentage + static_cast<double>(current_gear)) *
          (MAX_SPEED / 4);
    }
  } else if (throttle_percentage < 0) {
    desired_speed = throttle_percentage * (MAX_SPEED / 4);
  } else {
    desired_speed = 0;
  }

  if ((buttons[5] == 1) && (current_gear != Gears::Fourth)) {
    current_gear++;
    std::cout << "Marcha actual:" << static_cast<double>(current_gear)
              << std::endl;
  } else if ((buttons[4] == 1) && (current_gear != Gears::Reverse)) {
    current_gear--;
    std::cout << "Marcha actual:" << static_cast<double>(current_gear)
              << std::endl;
  }

  return std::make_pair(desired_steering_angle, desired_speed);
} // namespace controller_adapter
} // namespace controller_adapter
```

- gamepad_driver_ros_wrapper.hpp:

```
#pragma once

#include "go_car_manual_control/car_control_command.h"
#include <ros/ros.h>
#include <sensor_msgs/Joy.h>

class ControllerAdapterRosWrapper {
public:
  ControllerAdapterRosWrapper(ros::NodeHandle &nh) {
    _joy_sub = nh.subscribe("joy", 10,
                            &ControllerAdapterRosWrapper::callbackJoySub, this);
    _cmd_vel_pub = nh.advertise<go_car_manual_control::car_control_command>(
        "/cmd_car", 10);
  }
  void callbackJoySub(const sensor_msgs::JoyConstPtr &input_msg);
  void publishCmdVel();

private:
  ros::Subscriber _joy_sub;
  ros::Publisher _cmd_vel_pub;
};
```

- gamepad_driver_ros_wrapper.cpp:

```
#include "gamepad_driver.hpp"
#include "gamepad_driver_ros_wrapper.hpp"

go_car_manual_control::car_control_command output_msg;

void ControllerAdapterRosWrapper::callbackJoySub(
    const sensor_msgs::JoyConstPtr &input_msg) {
  const std::vector<float> axes(input_msg->axes);
  const std::vector<int> buttons(input_msg->buttons);

  std::pair<double, double> calculated_command =
      controller_adapter::controller_input_processing(axes, buttons);
  output_msg.desired_steering_angle = calculated_command.first;
  output_msg.desired_speed = calculated_command.second;
}

void ControllerAdapterRosWrapper::publishCmdVel() {
  _cmd_vel_pub.publish(output_msg);
}

int main(int argc, char **argv) {
  ros::init(argc, argv, "controller_adapter");
  ros::NodeHandle nh;
  ros::AsyncSpinner spinner(4);
```

```
  spinner.start();

  ControllerAdapterRosWrapper controllerAdapterRosWrapper(nh);
  output_msg.desired_steering_angle = 0.0;
  output_msg.desired_speed = 0.0;
  ros::Rate rate(50);
  while (ros::ok()) {
    controllerAdapterRosWrapper.publishCmdVel();
    rate.sleep();
  }

  ros::waitForShutdown();
}
```

b) *Keyboard_driver:*

- *keyboard_driver.hpp:*

```
#pragma once

#include <cstring>
#include <fcntl.h>
#include <fstream>
#include <iostream>
#include <linux/input.h>
#include <regex>
#include <string>
#include <unistd.h>

namespace keyboard_driver {

constexpr double MAX_STEERING_ANGLE = 0.8;
constexpr double MAX_SPEED = 20;
constexpr int BUFFER_SIZE{64};
constexpr int INPUT_DEVICE_ID_SELECTOR{1};

enum class key_status { pressed, non_pressed };

enum class Gears {
  Reverse = -1,
  Neutral = 0,
  First = 1,
  Second = 2,
  Third = 3,
  Fourth = 4,
};

struct keys {
  key_status upArrowKey{key_status::non_pressed};
  key_status leftArrowKey{key_status::non_pressed};
```

```
  key_status rightArrowKey{key_status::non_pressed};
  key_status downArrowKey{key_status::non_pressed};
  key_status minusKey{key_status::non_pressed};
  key_status numberZeroKey{key_status::non_pressed};
  key_status numberOneKey{key_status::non_pressed};
  key_status numberTwoKey{key_status::non_pressed};
  key_status numberThreeKey{key_status::non_pressed};
  key_status numberFourKey{key_status::non_pressed};
};

std::vector<std::string> getKeyboardEventFile();
keys getKeyStatus(int virtualKeyboardEventFileDescriptor);
std::pair<double, double> calculateCarCmd(keys keysStatus);

} // end namespace keyboard_driver
```

- *keyboard_driver.cpp:*

```
#include "keyboard_driver.hpp"

namespace keyboard_driver {

void operator++(Gears &gear, int) {
  int newGear;
  if (gear != Gears::Fourth) {
    newGear = static_cast<int>(gear) + 1;
  } else {
    throw std::runtime_error("[controller_adapter]: Invalid gear value");
  }
  gear = static_cast<Gears>(newGear);
}

void operator--(Gears &gear, int) {
  int newGear;
  if (gear != Gears::Reverse) {
    newGear = static_cast<int>(gear) - 1;
  } else {
    throw std::runtime_error("[controller_adapter]: Invalid gear value");
  }
  gear = static_cast<Gears>(newGear);
}

std::vector<std::string> getKeyboardEventFile() {
  std::string virtualInputDevicesFile_path = "/proc/bus/input/devices";
  std::string readLine;
  std::string possibleEventFile;
  std::vector<std::string> keyboardEventFile;

  std::ifstream virtualInputDevicesFile(virtualInputDevicesFile_path);
  if (virtualInputDevicesFile.is_open()) {
    while (virtualInputDevicesFile) // iterates for every line in file
```

```
    {
      std::getline(virtualInputDevicesFile, readLine);
      if (readLine.find("Handlers") != std::string::npos) {
        std::regex pattern("event\\d+"); // matches "event"+<some number>
        std::smatch matches;
        if (std::regex_search(readLine, matches, pattern)) {
          possibleEventFile = matches.str(0);
        }
      }
      if (readLine.find("EV") !=
          std::string::npos) { // search for supported events characteristic
        // field of device
        size_t equalSignPos = readLine.find('=');
        size_t newLinePos = readLine.find('\n');
        std::string supportedEventsBitmask = // extract supported events bitmask
            readLine.substr(equalSignPos + 1, newLinePos - equalSignPos - 1);
        if ((std::stoi(supportedEventsBitmask, nullptr, 16) & 0x120013) ==
            0x120013) { // check for supporting minimum features for being
          // considered a keyboard
          std::cout << "Keyboard event file found: " << possibleEventFile
                    << std::endl;
          keyboardEventFile.push_back(possibleEventFile);
        }
      }
    }
  } else {
    std::cout << "Error opening input devices virtual file" << std::endl;
  }
  return keyboardEventFile;
}

keys getKeyStatus(int virtualKeyboardEventFileDescriptor) {
  struct input_event inputBuffer[BUFFER_SIZE];
  static keys KeysStatus;

  if (virtualKeyboardEventFileDescriptor != -1) { // check file could be opened
    ssize_t readedBytes = read(virtualKeyboardEventFileDescriptor, &inputBuffer,
                               sizeof(struct input_event) * BUFFER_SIZE);
    int numberReadedEvents = readedBytes / sizeof(struct input_event);
    for (int numberSavedEvents = 0; numberSavedEvents < numberReadedEvents;
         numberSavedEvents++) {
      struct input_event *event = &inputBuffer[numberSavedEvents];
      if (event->type == EV_KEY) {
        switch (event->code) {
        case KEY_UP:
          if (event->value == 1) {
            std::cout << "Tecla UP pulsada" << std::endl;
            KeysStatus.upArrowKey = key_status::pressed;
          } else if (event->value == 0) {
            std::cout << "Tecla UP soltada" << std::endl;
            KeysStatus.upArrowKey = key_status::non_pressed;
```

```
        }
        break;
      case KEY_LEFT:
        if (event->value == 1) {
          std::cout << "Tecla LEFT pulsada" << std::endl;
          KeysStatus.leftArrowKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla LEFT soltada" << std::endl;
          KeysStatus.leftArrowKey = key_status::non_pressed;
        }
        break;

      case KEY_RIGHT:
        if (event->value == 1) {
          std::cout << "Tecla RIGHT pulsada" << std::endl;
          KeysStatus.rightArrowKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla RIGHT soltada" << std::endl;
          KeysStatus.rightArrowKey = key_status::non_pressed;
        }
        break;

      case KEY_DOWN:
        if (event->value == 1) {
          std::cout << "Tecla DOWN pulsada" << std::endl;
          KeysStatus.downArrowKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla DOWN soltada" << std::endl;
          KeysStatus.downArrowKey = key_status::non_pressed;
        }
        break;

      case KEY_SLASH: // for spanish keyboard configuration this key is -
        if (event->value == 1) {
          std::cout << "Tecla - pulsada" << std::endl;
          KeysStatus.minusKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla - soltada" << std::endl;
          KeysStatus.minusKey = key_status::non_pressed;
        }
        break;

      case KEY_0:
        if (event->value == 1) {
          std::cout << "Tecla 0 pulsada" << std::endl;
          KeysStatus.numberZeroKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla 0 soltada" << std::endl;
          KeysStatus.numberZeroKey = key_status::non_pressed;
        }
        break;
```

```
      case KEY_1:
        if (event->value == 1) {
          std::cout << "Tecla 1 pulsada" << std::endl;
          KeysStatus.numberOneKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla 1 soltada" << std::endl;
          KeysStatus.numberOneKey = key_status::non_pressed;
        }
        break;

      case KEY_2:
        if (event->value == 1) {
          std::cout << "Tecla 2 pulsada" << std::endl;
          KeysStatus.numberTwoKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla 2 soltada" << std::endl;
          KeysStatus.numberTwoKey = key_status::non_pressed;
        }
        break;

      case KEY_3:
        if (event->value == 1) {
          std::cout << "Tecla 3 pulsada" << std::endl;
          KeysStatus.numberThreeKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla 3 soltada" << std::endl;
          KeysStatus.numberThreeKey = key_status::non_pressed;
        }
        break;

      case KEY_4:
        if (event->value == 1) {
          std::cout << "Tecla 4 pulsada" << std::endl;
          KeysStatus.numberFourKey = key_status::pressed;
        } else if (event->value == 0) {
          std::cout << "Tecla 4 soltada" << std::endl;
          KeysStatus.numberFourKey = key_status::non_pressed;
        }
        break;

      default:
        std::cout << "Tecla no valida pulsada" << std::endl;
        break;
      }
    }
  }
} else {
  std::cout << "Error opening keyboard event virtual file" << std::endl;
}
```

```
  return KeysStatus;
} // end getKeyStatus

std::pair<double, double> calculateCarCmd(keys keysStatus) {
  double cmdSpeed{0.0};
  double cmdSteering{0.0};
  static Gears current_gear{Gears::Neutral};

  if ((keysStatus.upArrowKey == key_status::pressed) &&
      (keysStatus.downArrowKey == key_status::non_pressed)) {
    if ((current_gear != Gears::Neutral) && (current_gear != Gears::Reverse)) {
      cmdSpeed = (1 + static_cast<double>(current_gear)) * (MAX_SPEED / 4);
    } else {
      cmdSpeed = 0.0;
    }
  } else if ((keysStatus.upArrowKey == key_status::non_pressed) &&
             (keysStatus.downArrowKey == key_status::pressed)) {
    if (current_gear == Gears::Reverse) {
      cmdSpeed = -(MAX_SPEED / 4);
    } else if (current_gear == Gears::Neutral) {
      cmdSpeed = 0.0;
    } else {
      cmdSpeed = (-1 + static_cast<double>(current_gear)) * (MAX_SPEED / 4);
    }
  } else {
    if (current_gear == Gears::Reverse) {
      cmdSpeed = 0.0;
    } else {
      cmdSpeed = static_cast<double>(current_gear) * (MAX_SPEED / 4);
    }
  }

  if ((keysStatus.leftArrowKey == key_status::pressed) &&
      (keysStatus.rightArrowKey == key_status::non_pressed)) {
    cmdSteering = MAX_STEERING_ANGLE;
  } else if ((keysStatus.leftArrowKey == key_status::non_pressed) &&
             (keysStatus.rightArrowKey == key_status::pressed)) {
    cmdSteering = -MAX_STEERING_ANGLE;
  } else {
    cmdSteering = 0.0;
  }

  if (keysStatus.minusKey == key_status::pressed) {
    current_gear = Gears::Reverse;
  } else if (keysStatus.numberZeroKey == key_status::pressed) {
    current_gear = Gears::Neutral;
  } else if (keysStatus.numberOneKey == key_status::pressed) {
    current_gear = Gears::First;
  } else if (keysStatus.numberTwoKey == key_status::pressed) {
    current_gear = Gears::Second;
  } else if (keysStatus.numberThreeKey == key_status::pressed) {
```

```
    current_gear = Gears::Third;
  } else if (keysStatus.numberFourKey == key_status::pressed) {
    current_gear = Gears::Fourth;
  }

  return std::pair<double, double>(cmdSpeed, cmdSteering);
} // namespace keyboard_driver

} // end namespace keyboard_driver
```

- *keyboard_driver_ros.hpp:*

```
#pragma once

#include "go_car_manual_control/car_control_command.h"
#include <ros/ros.h>
#include <std_msgs/Int32.h>

class KeyboardDriverRosWrapper {
public:
  KeyboardDriverRosWrapper(ros::NodeHandle &nh) {
    _cmd_vel_pub = nh.advertise<go_car_manual_control::car_control_command>(
        "/cmd_car", 10);
  }
  void publishCmdVel();

private:
  ros::Publisher _cmd_vel_pub;
};
```

- *keyboard_driver_ros.cpp:*

```
#include "keyboard_driver_ros_wrapper.hpp"
#include "keyboard_driver.hpp"

go_car_manual_control::car_control_command output_msg;

void KeyboardDriverRosWrapper::publishCmdVel() {
  _cmd_vel_pub.publish(output_msg);
}

int main(int argc, char **argv) {
  ros::init(argc, argv, "keyboard_driver");
  ros::NodeHandle nh;

  KeyboardDriverRosWrapper keyboardDriverRosWrapper(nh);
  output_msg.desired_steering_angle = 0.0;
  output_msg.desired_speed = 0.0;

  std::vector<std::string> keyboardEventFileOptions =
```

```
      keyboard_driver::getKeyboardEventFile();

  std::string keyboardEventFile = keyboardEventFileOptions.at(1);
  std::string globalPath_to_eventFile = "/dev/input/";
  std::string virtualKeyboardEventFile_path =
      globalPath_to_eventFile + keyboardEventFile;
  int virtualKeyboardEventFile_descriptor =
      open(virtualKeyboardEventFile_path.c_str(), O_RDONLY | O_NONBLOCK);

  ros::Rate rate(50);
  while (ros::ok()) {
    keyboard_driver::keys keysStatus =
        keyboard_driver::getKeyStatus(virtualKeyboardEventFile_descriptor);
    std::pair<double, double> cmdOutput =
        keyboard_driver::calculateCarCmd(keysStatus);
    output_msg.desired_speed = cmdOutput.first;
    output_msg.desired_steering_angle = cmdOutput.second;

    keyboardDriverRosWrapper.publishCmdVel();
    rate.sleep();
  }
  close(virtualKeyboardEventFile_descriptor);

  return 0;
}
```

3. Modelos propios del mundo de Gazebo

a) *roadmap_floor*

- model.sdf

```
<?xml version="1.0" ?>
<sdf version="1.7">
  <model name="roadmap_floor">
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
          </plane>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>25</mu>
              <mu2>25</mu2>
            </ode>
          </friction>
        </surface>
```

```
      </collision>
      <visual name="visual">
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>300 300</size>
          </plane>
        </geometry>
        <material>
          <script>
            <uri>model://roadmap_floor/materials/scripts</uri>
            <uri>model://roadmap_floor/materials/textures</uri>
            <name>roadmap_floor/Image</name>
          </script>
          <lightning>false</lightning>
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

- roadmap_floor.material

```
material roadmap_floor/Image
{
  receive_shadows off
  technique
  {
    pass
    {
      texture_unit
      {
        texture roadmap_floor.png
      }
    }
  }
}
```

b) skycrapper

- model.sdf

```
<?xml version="1.0" ?>
<sdf version="1.7">
  <model name="skycrapper">
    <static>true</static>
    <link name="link">
      <collision name="collision">
```

```
        <geometry>
          <cylinder>
            <radius>5</radius>
            <length>40</length>
          </cylinder>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <cylinder>
            <radius>5</radius>
            <length>40</length>
          </cylinder>
        </geometry>
        <material>
          <script>
            <uri>model://skycrapper/materials/scripts</uri>
            <uri>model://skycrapper/materials/textures</uri>
            <name>skycrapper/Image</name>
          </script>
          <lightning>false</lightning>
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

- skycrapper.material

```
material skycrapper/Image
{
  receive_shadows off
  technique
  {
    pass
    {
      texture_unit
      {
        texture windows.jpg
      }
    }
  }
}
```

c) *new_apartment_block*

- model.sdf

```
<?xml version="1.0" ?>
<sdf version="1.7">
  <model name="new_apartment_block">
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <box>
            <size>15 15 20</size>
          </box>
        </geometry>
      </collision>
      <visual name="block">
        <geometry>
          <box>
            <size>15 15 20</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>model://new_apartment_block/materials/scripts</uri>
            <uri>model://new_apartment_block/materials/textures</uri>
            <name>new_apartment_facade/Image</name>
          </script>
          <lightning>false</lightning>
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
        </material>
      </visual>
      <!-- <visual name="ceil">
        <pose>0 0 20.01 0 0 0</pose>
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>15 15</size>
          </plane>
        </geometry>
        <material>
          <script>
            <uri>model://new_apartment_block/materials/scripts</uri>
            <uri>model://new_apartment_block/materials/textures</uri>
            <name>concrete/Image</name>
          </script>
          <lightning>false</lightning>
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
```

```
        </material>
      </visual> -->
    </link>
  </model>
</sdf>
```

- new_apartment_facade.material

```
material new_apartment_facade/Image
{
  receive_shadows off
  technique
  {
    pass
    {
      texture_unit
      {
        texture new_apartment_facade.jpg
      }
    }
  }
}
```

d) old_apartment_block

- model.sdf

```
<?xml version="1.0" ?>
<sdf version="1.7">
  <model name="old_apartment_block">
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <box>
            <size>15 15 20</size>
          </box>
        </geometry>
      </collision>
      <visual name="block">
        <geometry>
          <box>
            <size>15 15 20</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>model://old_apartment_block/materials/scripts</uri>
            <uri>model://old_apartment_block/materials/textures</uri>
            <name>old_apartment_facade/Image</name>
          </script>
          <lightning>false</lightning>
```

```
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
        </material>
      </visual>
      <!-- <visual name="ceil">
        <pose>0 0 20.01 0 0 0</pose>
        <geometry>
          <plane>
            <normal>0 0 1</normal>
            <size>15 15</size>
          </plane>
        </geometry>
        <material>
          <script>
            <uri>model://old_apartment_block/materials/scripts</uri>
            <uri>model://old_apartment_block/materials/textures</uri>
            <name>concrete/Image</name>
          </script>
          <lightning>false</lightning>
          <ambient>1 1 1 1</ambient>
          <diffuse>0 0 0 1</diffuse>
          <specular>0 0 0 1</specular>
        </material>
      </visual> -->
    </link>
  </model>
</sdf>
```

- old_apartment_facade.material

```
material old_apartment_facade/Image
{
  receive_shadows off
  technique
  {
    pass
    {
      texture_unit
      {
        texture old_apartment_facade.png
      }
    }
  }
}
```

4. Archivos de lanzamiento y configuración de navegación

a) *hdl_graph_slam.launch:*

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

    <node pkg="nodelet" type="nodelet" name="velodyne_nodelet_manager" args="manager"
output="log"/>

    <!-- prefiltering_nodelet -->
    <node pkg="nodelet" type="nodelet" name="prefiltering_nodelet" args="load
hdl_graph_slam/PrefilteringNodelet velodyne_nodelet_manager" output="log">

        <param name="base_link_frame" value="base_footprint_link" />
        <!-- distance filter -->
        <param name="use_distance_filter" value="true" />
        <param name="distance_near_thresh" value="5.0" />
        <param name="distance_far_thresh" value="100.0" />
        <!-- NONE, VOXELGRID, or APPROX_VOXELGRID -->
        <param name="downsample_method" value="VOXELGRID" />
        <param name="downsample_resolution" value="0.25" />
        <!-- NONE, RADIUS, or STATISTICAL -->
        <param name="outlier_removal_method" value="RADIUS" />
        <param name="radius_radius" value="0.5" />
        <param name="radius_min_neighbors" value="2" />
    </node>

    <!-- scan_matching_odometry_nodelet -->
    <node pkg="nodelet" type="nodelet" name="scan_matching_odometry_nodelet" args="load
hdl_graph_slam/ScanMatchingOdometryNodelet velodyne_nodelet_manager" output="log">
        <param name="points_topic" value="/velodyne_points" />
        <!-- published_odom_topic -->
        <param name="published_odom_topic" value="/odom" />
        <param name="odom_frame_id" value="odom" />
        <param name="keyframe_delta_trans" value="1.0" />
        <param name="keyframe_delta_angle" value="1.0" />
        <param name="keyframe_delta_time" value="10000.0" />
        <param name="transform_thresholding" value="false" />
        <param name="max_acceptable_trans" value="1.0" />
        <param name="max_acceptable_angle" value="1.0" />
        <!-- ICP, GICP, NDT, GICP_OMP, NDT_OMP, FAST_GICP(recommended), or FAST_VGICP -->
        <param name="registration_method" value="FAST_GICP" />
        <param name="reg_num_threads" value="0" />
        <param name="reg_transformation_epsilon" value="0.01"/>
        <param name="reg_maximum_iterations" value="64"/>
        <param name="reg_max_correspondence_distance" value="2.5"/>
        <param name="reg_max_optimizer_iterations" value="20"/>
        <param name="reg_use_reciprocal_correspondences" value="false"/>
        <param name="reg_correspondence_randomness" value="20"/>
        <param name="reg_resolution" value="1.0" />
        <param name="reg_nn_search_method" value="DIRECT7" />
```

```
    </node>

    <!-- floor_detection_nodelet -->
    <node pkg="nodelet" type="nodelet" name="floor_detection_nodelet" args="load
hdl_graph_slam/FloorDetectionNodelet velodyne_nodelet_manager" output="log">
        <param name="points_topic" value="/velodyne_points" />
        <param name="tilt_deg" value="0.0" />
        <param name="sensor_height" value="1.85" />
        <param name="height_clip_range" value="3.0" />
        <param name="floor_pts_thresh" value="512" />
        <param name="floor_normal_thresh" value="10.0"/>
        <param name="use_normal_filtering" value="true" />
        <param name="normal_filter_thresh" value="20.0" />
    </node>

    <!-- hdl_graph_slam_nodelet -->
    <node pkg="nodelet" type="nodelet" name="hdl_graph_slam_nodelet" args="load
hdl_graph_slam/HdlGraphSlamNodelet velodyne_nodelet_manager" output="log">
        <remap from="gpsimu_driver/imu_data" to="/imu"/>
        <remap from="gps/navsat" to="/gps/coordinates"/>
        <param name="points_topic" value="/velodyne_points" />
        <!-- published_odom_topic -->
        <param name="published_odom_topic" value="/odom" />
        <!-- frame settings -->
        <param name="map_frame_id" value="map" />
        <param name="odom_frame_id" value="odom" />
        <!-- optimization params -->
        <!-- typical solvers: gn_var, gn_fix6_3, gn_var_cholmod, lm_var, lm_fix6_3,
lm_var_cholmod, ... -->
        <param name="g2o_solver_type" value="lm_var_cholmod" />
        <param name="g2o_solver_num_iterations" value="512" />
        <!-- constraint switches -->
        <param name="enable_gps" value="false" />
        <param name="enable_imu_acceleration" value="false" />
        <param name="enable_imu_orientation" value="false" />
        <!-- keyframe registration params -->
        <param name="max_keyframes_per_update" value="10" />
        <param name="keyframe_delta_trans" value="2.0" />
        <param name="keyframe_delta_angle" value="2.0" />
        <!-- fix first node for optimization stability -->
        <param name="fix_first_node" value="true"/>
        <param name="fix_first_node_stddev" value="10 10 10 1 1 1"/>
        <param name="fix_first_node_adaptive" value="true"/>
        <!-- loop closure params -->
        <param name="distance_thresh" value="20.0" />
        <param name="accum_distance_thresh" value="35.0" />
        <param name="min_edge_interval" value="5.0" />
        <param name="fitness_score_thresh" value="0.5" />

        <!-- scan matching params -->
        <param name="registration_method" value="FAST_GICP" />
```

```
        <param name="reg_num_threads" value="4" />
        <param name="reg_transformation_epsilon" value="0.01"/>
        <param name="reg_maximum_iterations" value="64"/>
        <param name="reg_max_correspondence_distance" value="2.5"/>
        <param name="reg_max_optimizer_iterations" value="20"/>
        <param name="reg_use_reciprocal_correspondences" value="false"/>
        <param name="reg_correspondence_randomness" value="20"/>
        <param name="reg_resolution" value="1.0" />
        <param name="reg_nn_search_method" value="DIRECT7" />
        <!-- edge params -->
        <!-- GPS -->
        <param name="gps_edge_robust_kernel" value="NONE" />
        <param name="gps_edge_robust_kernel_size" value="10.0" />
        <param name="gps_edge_stddev_xy" value="20.0" />
        <param name="gps_edge_stddev_z" value="20.0" />
        <!-- IMU orientation -->
        <param name="imu_orientation_edge_robust_kernel" value="NONE" />
        <param name="imu_orientation_edge_stddev" value="5.0" />
        <!-- IMU acceleration (gravity vector) -->
        <param name="imu_acceleration_edge_robust_kernel" value="NONE" />
        <param name="imu_acceleration_edge_stddev" value="5.0" />
        <!-- ground plane -->
        <param name="floor_edge_robust_kernel" value="NONE" />
        <param name="floor_edge_stddev" value="10.0" />
        <!-- scan matching -->
        <!-- robust kernels: NONE, Cauchy, DCS, Fair, GemanMcClure, Huber, PseudoHuber,
Saturated, Tukey, Welsch -->
        <param name="odometry_edge_robust_kernel" value="NONE" />
        <param name="odometry_edge_robust_kernel_size" value="10.0" />
        <param name="loop_closure_edge_robust_kernel" value="Huber" />
        <param name="loop_closure_edge_robust_kernel_size" value="1.0" />
        <param name="use_const_inf_matrix" value="false" />
        <param name="const_stddev_x" value="0.5" />
        <param name="const_stddev_q" value="0.1" />
        <param name="var_gain_a" value="20.0" />
        <param name="min_stddev_x" value="0.1" />
        <param name="max_stddev_x" value="5.0" />
        <param name="min_stddev_q" value="0.05" />
        <param name="max_stddev_q" value="0.2" />
        <!-- update params -->
        <param name="graph_update_interval" value="1.0" />
        <param name="map_cloud_update_interval" value="1.0" />
        <param name="map_cloud_resolution" value="0.5" />
    </node>

    <node pkg="hdl_graph_slam" type="map2odom_publisher.py" name="map2odom_publisher"
output="log">
        <param name="map_frame_id" value="map" />
        <param name="odom_frame_id" value="odom" />
    </node>
</launch>
```

b) *move_base:*

- move_base.launch:

```
<?xml version='1.0' encoding='UTF-8'?>
<launch>

    <node name="cmd_vel_to_cmd_car" pkg="go_car_nav" type="cmd_vel_to_cmd_car"
output="screen"/>
    <!-- Defining parameters for move_base node -->
    <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
        <rosparam file="$(find go_car_nav)/config/move_base_params.yaml" command="load"/>
        <rosparam file="$(find go_car_nav)/config/global_costmap_params.yaml"
command="load" />
        <rosparam file="$(find go_car_nav)/config/local_costmap_params.yaml"
command="load" />
        <rosparam file="$(find go_car_nav)/config/base_global_planner_params.yaml"
command="load" />
        <rosparam file="$(find go_car_nav)/config/base_local_planner_params.yaml"
command="load" />
    </node>
</launch>
```

- Move_base_params.yaml:

```
base_global_planner: global_planner/GlobalPlanner
base_local_planner: teb_local_planner/TebLocalPlannerROS
recovery_behaviors: [{name: conservative_reset, type:
clear_costmap_recovery/ClearCostmapRecovery}, {name: rotate_recovery, type:
rotate_recovery/RotateRecovery}, {name: aggressive_reset, type:
clear_costmap_recovery/ClearCostmapRecovery}]
controller_frequency: 5.0
planner_patience: 20.0
controller_patience: 20.0
conservative_reset_dist: 10.0
recovery_behavior_enabled: true
clearing_rotation_allowed: true
shutdown_costmaps: false
oscillation_timeout: 0.0
oscillation_distance: 1.0
planner_frequency: 0.2
max_planning_retries: 10
```

- base_global_planner_params.yaml:

```
GlobalPlanner:
  use_dijkstra: true
  outline_map: false
  use_grid_path: false
  orientation_mode: 0 #None
  cost_factor: 1.0
  lethal_cost: 100
  neutral_cost: 20
```

- base_local_planner_params.yaml:

```yaml
TebLocalPlannerROS:

 map_topic: map
 odom_topic: odom

 # Trajectory
 teb_autosize: True
 dt_ref: 0.2
 dt_hysteresis: 0.02
 global_plan_overwrite_orientation: True
 allow_init_with_backwards_motion: True
 max_global_plan_lookahead_dist: 10.0
 global_plan_viapoint_sep: 2.0
 feasibility_check_no_poses: -1
 feasibility_check_lookahead_distance: 10.0
 exact_arc_length: False
 publish_feedback: False
 prevent_look_ahead_poses_near_goal: 5.0

 # ViaPoints
 global_plan_viapoint_sep: 2.0
 via_points_ordered: False

 # Robot
 max_vel_x: 5.6
 max_vel_x_backwards: 1.0
 max_vel_y: 0.0
 max_vel_theta: 0.2
 acc_lim_x: 6.0
 acc_lim_theta: 0.2
 is_footprint_dynamic: True
 use_proportional_saturation: False
 transform_tolerance: 10.0

 ## Robot/Carlike ##
 min_turning_radius: 6.0
 wheelbase: 2.8
 cmd_angle_instead_rotvel: True
 footprint_model: # types: "point", "circular", "two_circles", "line", "polygon"
   type: "polygon"
   vertices: [ [3.25, 1.5], [3.25, -1.5], [-2.5, -1.5], [-2.5, 1.5] ]

 # GoalTolerance

 xy_goal_tolerance: 0.5
 yaw_goal_tolerance: 0.2
 free_goal_vel: True
 complete_global_plan: False
```

```
# Obstacles

min_obstacle_dist: 0.5
inflation_dist: 0.5
dynamic_obstacle_inflation_dist: 0.75
include_dynamic_obstacles: True
include_costmap_obstacles: True
costmap_obstacles_behind_robot_dist: 5.0
obstacle_poses_affected: 15

## Obstacle/VelocityRatio
obstacle_proximity_ratio_max_vel: 0.5
obstacle_proximity_lower_bound: 0.5
obstacle_proximity_higher_bound: 2.5

# Optimization

no_inner_iterations: 5
no_outer_iterations: 4
optimization_activate: True
optimization_verbose: False
penalty_epsilon: 0.0
weight_max_vel_x: 100
weight_max_vel_theta: 0.1
weight_acc_lim_x: 10
weight_acc_lim_theta: 0.1
weight_kinematics_nh: 10000
weight_kinematics_forward_drive: 10000
weight_kinematics_turning_radius: 10
weight_optimaltime: 100 # must be > 0
weight_shortest_path: 10
weight_obstacle: 1000
weight_inflation: 10.0
weight_dynamic_obstacle: 10
weight_dynamic_obstacle_inflation: 0.2
weight_viapoint: 1
weight_adapt_factor: 2
obstacle_cost_exponent: 1
switching_blocking_period: 10.0

# Homotopy Class Planner

enable_homotopy_class_planning: True
enable_multithreading: True
max_number_classes: 4
selection_cost_hysteresis: 1.0
selection_prefer_initial_plan: 10.0
selection_obst_cost_scale: 10.0
selection_alternative_time_cost: False

roadmap_graph_no_samples: 10
```

```
 roadmap_graph_area_width: 5
 roadmap_graph_area_length_scale: 1.0
 h_signature_prescaler: 0.5
 h_signature_threshold: 0.1
 obstacle_heading_threshold: 0.45
 viapoints_all_candidates: True
 delete_detours_backwards: True
 max_ratio_detours_duration_best_duration: 5.0
 visualize_hc_graph: False
 visualize_with_time_as_z_axis_scale: False

 costmap_converter_plugin: "costmap_converter::CostmapToPolygonsDBSMCCH"
 costmap_converter_dspin_thread: True

# Recovery

 shrink_horizon_backup: True
 shrink_horizon_min_duration: 10
 oscillation_recovery: True
 oscillation_v_eps: 0.1
 oscillation_omega_eps: 0.1
 oscillation_recovery_min_duration: 10
 oscillation_filter_duration: 10
```

- global_costmap_params.yaml:

```
global_costmap:
   footprint: "[ [3.25, 1.5], [3.25, -1.5], [-2.5, -1.5], [-2.5, 1.5] ]"
   plugins:
      - {name: 2d_slam_map_layer, type: "costmap_2d::VoxelLayer"}
      - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
   2d_slam_map_layer:
      observation_sources: 3d_map
      3d_map:
         sensor_frame: "map"
         data_type: PointCloud2
         topic: /hdl_graph_slam/map_points
         marking: true
         clearing: true
         min_obstacle_height: 1.5
         max_obstacle_height: 5.0
         obstacle_range: 250.0
         raytrace_range: 250.0
   inflation_layer:
      inflation_radius: 5.0
      cost_scaling_factor: 2.0

   global_frame: map
   robot_base_frame: base_footprint_link
   update_frequency: 0.5
   publish_frequency: 0.5
   rolling_window: false
```

```
   origin_x: -75.0
   origin_y: -199.1
   width: 250.0
   height: 250.0
   resolution: 1.0
   transform_tolerance: 10.0
```

- local_costmap_params.yaml:

```
local_costmap:
   footprint: "[ [3.25, 1.5], [3.25, -1.5], [-2.5, -1.5], [-2.5, 1.5] ]"
   plugins:
      - {name: sonar, type: "range_sensor_layer::RangeSensorLayer"}
      - {name: laser_sensor_layer, type: "costmap_2d::VoxelLayer"}
      - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
   sonar:
      topics: ['/range_sensor/front', '/range_sensor/left', '/range_sensor/right',
'/range_sensor/back']
      clear_max_reading: true
   laser_sensor_layer:
      observation_sources: velodyne_sensor
      velodyne_sensor:
         sensor_frame: lidar_link
         data_type: PointCloud2
         topic: /velodyne_points
         marking: true
         clearing: true
         min_obstacle_height: 0.25
         max_obstacle_height: 3.0
         obstacle_range: 20.0
         raytrace_range: 20.0
   inflation_layer:
      inflation_radius: 3.5
      cost_scaling_factor: 0.5

   global_frame: map
   robot_base_frame: base_footprint_link
   update_frequency: 2.0
   publish_frequency: 2.0
   rolling_window: true
   width: 20.0
   height: 20.0
   resolution: 0.2
   transform_tolerance: 10.0
   footprint: [ [3.25, 1.5], [3.25, -1.5], [-2.5, -1.5], [-2.5, 1.5] ]
   robot_radius: 0.0
```

5. Lanzador de aplicación

a) *setup_env.sh:*

```
#!/usr/bin/env bash

# Ask for the preferred command interpreter in order to define config file to modify
valid_interpreter_prompts=("zsh" "bash")

read -p "Choose your preferred command interpreter (zsh/bash): " interpreter
while [[ ! " ${valid_interpreter_prompts[@]} " =~ " ${interpreter} " ]]; do
    echo "Invalid interpreter. Please type it again."
    read -p "Choose your preferred command interpreter (zsh/bash): " interpreter
done

file_path="/home/$USER/.${interpreter}rc"
gazebo_patch_command="export IGN_IP=127.0.0.1"

# Check if the line does not exist in file and add it if it is the case
if ! grep -Fxq "$gazebo_patch_command" "$file_path"; then
    echo "$gazebo_patch_command" >> $file_path
fi

# Install packages dependencies
rosdep install --from-paths src --ignore-src -r -y

# Ask for permission to read keyboard input for manual control
read -p "Do you want to give access to keyboard input in order to enable manual control
keyboard)(y/n) " keyboard_permission
valid_answers_keyboard_access_permission_prompts=("y" "n")
while [[ ! " ${valid_answers_keyboard_access_permission_prompts[@]} " =~ "
${keyboard_permission} " ]]; do
    echo "Invalid answer. Please type it again."
    read -p "Do you want to give access to keyboard input in order to enable manual
control keyboard)(y/n) " keyboard_permission
done
if [[ "$keyboard_permission" == "y" ]]; then
    sudo usermod -a -G input $USER
    echo "Keyboard listener permission granted"
fi

# In order to force a source of the config file
exec /bin/$interpreter
```

b) *start_simulation.launch*:

```
<?xml version='1.0' encoding='UTF-8'?>
<launch>
    <arg name="world_name" default="city"/>
    <arg name="details" default="quality"/>
    <!-- car_model options: lincoln -->
    <arg name="car_model" default="lincoln"/>
    <!-- control_type options: keyboard, gamepad, autonomous -->
    <arg name="control_type" default="keyboard"/>
    <arg name="rviz_visualization" default="false"/>
    <arg name="real_time_telemetry" default="false"/>

    <include file="$(find go_car_gazebo)/launch/load_gazebo_simulation.launch">
        <arg name="world" value="$(arg world_name)"/>
        <arg name="details" value="$(arg details)"/>
        <arg name="car_model" value="$(arg car_model)"/>
    </include>

    <node name="map_publisher_2d" pkg="map_server" type="map_server" args="$(find
go_car_gazebo)/maps/2d_scanned_map.yaml"/>

    <node name="hdl_graph_slam_launch_delayer" pkg="go_car_bringup"
type="delay_execution.sh" args="20 go_car_nav hdl_graph_slam.launch silent" output="log"/>

    <group unless="$(eval control_type == 'autonomous')">
        <include file="$(find go_car_manual_control)/launch/manual_control.launch">
            <arg name="type" value="$(arg control_type)"/>
        </include>
    </group>
    <group if="$(eval control_type == 'autonomous')">
        <node name="move_base_launch_delayer" pkg="go_car_bringup"
type="delay_execution.sh" args="25 go_car_nav move_base.launch" output="screen"/>
    </group>

    <group if="$(arg rviz_visualization)">
        <node name="rviz_launch_delayer" pkg="go_car_bringup" type="delay_execution.sh"
args="30 go_car_bringup open_rviz.launch" output="screen"/>
    </group>

    <group if="$(arg real_time_telemetry)">
        <node name="rt_telemetry_plotter" pkg="go_car_bringup" type="delay_execution.sh"
args="30 go_car_bringup open_plotjugger.launch" output="screen"/>
    </group>

</launch>
```

c) delay_execution.sh:

```
#!/bin/bash

# This bash script is used as a workaround in order to avoid unsync issues appearing in
roslaunch,
# as Gazebo takes some time to start, to ensure algorithm are not called previous to
# complete loading of simulation.

# Params:
# 1-> Seconds to wait
# 2-> Package name where to find launch file
# 3-> Launch file name
# 4-> Output mode

echo "Preloading..."
sleep $1
echo "Preload completed!"
if [[ "$4" == "silent" ]]; then
    roslaunch $2 $3 2>&1
else
    roslaunch $2 $3
fi
```

REFERENCIAS

[1] R. Braun, Autonomous Vehicles: From Science Fiction to Sustainable Future, vol. 2, 2019.

[2] Y. Song, «medium.com,» [En línea]. Available: https://medium.com/@overthevehicle/autonomous-vehicles-the-history-from-dream-to-reality-e136403a7252. [Último acceso: 5 Abril 2023].

[3] E. Dickmanns, «The development of machine vision for road vehicles in the last decade,» *IEEEE,* vol. 1, 2002.

[4] D. Gerónimo, «Red de Agentes para la Innovación,» [En línea]. Available: https://www.eoi.es/blogs/redinnovacionEOI/files/2015/09/vamors-merge-1024x351.jpg. [Último acceso: Abril 2023].

[5] C. G. Oliva, «Esta es la historia del coche autónomo ¿Es tan nueva?,» 2018. [En línea]. Available: https://www.autonocion.com/historia-coche-autonomo/. [Último acceso: Abril 2023].

[6] R. B. S. F. F. T. a. E. D. M. Maurer, «A compact vision system for road vehicle guidance,» *IEEE,* vol. 3.

[7] M. Chapman, «Modern Classic: Mercedes-Benz W140 S-Class -- AutoMuse,» [En línea]. Available: https://www.automuse.co.nz/news/modern-classic-mercedes-benz-w140-s-class. [Último acceso: Abril 2023].

[8] M. M. S. e. a. Ahmed SK, «Road traffic accidental injuries and deaths: A neglected globalhealth issue,» *Health Sci Rep.,* 2023.

[9] K. V. a. e. Singh Harnam, «Fatal Road Traffic Accidents: Causes and Factors Responsible,» *Journal of Indian Academy of Forensic Medicine,* vol. 38, 2016.

[10] «Road Safety Atlas,» [En línea]. Available: https://ec.europa.eu/transport/road_safety/specialist/statistics/map-viewer/. [Último acceso: Abril 2023].

[11] M. T. M. A. T. O. Saeed Asadi Bagloee, «Autonomous vehicles: challenges, opportunities, and future implications for transportation policies,» *Springer,* 2016.

[12] A. ., «Predicting traffic patterns, one Honda at a time,» *MSN Auto,* 2012.

[13] C. L. J.-D. Y. S. T. Javier Ibañez Guzmán, «Autonomous Driving: Context and State-of-the-Art,» de *Handbook of Intelligent Vehicles*, Springer, 2012, pp. 1271-1310.

[14] «Darpa Grand Challenge - Wikipedia,» 3 Junio 2023. [En línea]. Available: https://en.wikipedia.org/wiki/DARPA_Grand_Challenge. [Último acceso: 15 Junio 2023].

[15] «Archive Trib Live CCPA,» TRIB, [En línea]. Available: https://archive.triblive.com/ccpa/. [Último acceso: 15 Junio 2023].

[16] E. Commision, «CORDIS- CYBERnetic CARS for a new transportation system in the cities,» 13 Junio 2005. [En línea]. Available: https://cordis.europa.eu/project/id/IST-2000-28487. [Último acceso: 2023 Junio 2023].

[17] «J3016_202104 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles,» 30 Abril 2021. [En línea]. Available: https://www.sae.org/standards/content/j3016_202104/. [Último acceso: 15 Junio 2023].

[18] S. Dai, «IOT Automotive News Baidu Apollo Robotaxi,» IOT Automotive News, [En línea]. Available: https://iot-automotive.news/baidu-apollo-robotaxi/. [Último acceso: 16 Junio 2023].

[19] R. Tellez, «TheConstructSim,» [En línea]. Available: https://www.theconstructsim.com/history-ros/. [Último acceso: Abril 2023].

[20] AmandaDattalo, «ROS/Introduction - ROS Wiki,» [En línea]. Available: https://wiki.ros.org/ROS/Introduction. [Último acceso: 2023 Abril].

[21] J. E. Riva, «ROS Wiki,» [En línea]. Available: https://wiki.ros.org/es. [Último acceso: Abril 2023].

[22] N. Martignoni. [En línea]. Available: https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png. [Último acceso: Abril 2023].

[23] Y. Liao, «Introduction of Robot Operative Systems 2: ROS2- SAFER,» [En línea]. Available: https://etn-sas.eu/2020/03/23/introduction-of-robot-operating-systems-2-ros2/. [Último acceso: Abril 2023].

[24] «GazeboSim,» [En línea]. Available: https://classic.gazebosim.org/. [Último acceso: Abril 2023].

[25] J. S. M. B. Rahul Bhadani, «The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications,» *EPTCS,* pp. 32-47, 2018.

[26] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Anexo:Momentos_de_inercia. [Último acceso: 5 Abril 2023].

[27] «HDL32E Datasheet- Velodyne Lidar,» VelodyneLidar, [En línea]. Available: https://velodynelidar.com/wp-content/uploads/2019/12/97-0038-Rev-N-97-0038-DATASHEETWEBHDL32E_Web.pdf. [Último acceso: Abril 20 2023].

[28] «Mouser,» [En línea]. Available: https://www.mouser.com/datasheet/2/1083/3dm_cx5_10_datasheet_8400_0114_rev_f-2322180.pdf.

[29] «Datasheet,» Sparkfun, [En línea]. Available: https://www.sparkfun.com/datasheets/GPS/Modules/LS20030~3_datasheet_v1.2.pdf.

[30] «Datasheet CarSonar WR,» Maxbotix, [En línea]. Available: https://maxbotix.com/pages/carsonar-wr-datasheet.

[31] «Ingeniería y Mecánica,» 21 Octubre 2019. [En línea]. Available: https://www.ingenieriaymecanicaautomotriz.com/que-es-y-como-funciona-el-principio-de-ackerman/.

[32] J. Bohren, «Joy ROS Wiki,» [En línea]. Available: https://wiki.ros.org/joy.

[33] «Linux Kernel,» [En línea]. Available: https://www.kernel.org/doc/html/v4.15/input/event-codes.html. [Último acceso: 20 Mayo 2023].

[34] «SDFormat,» Open Source Robotics Foundation, 2020. [En línea]. Available: http://sdformat.org/spec?elem=sdf&ver=1.7. [Último acceso: 3 Mayo 2023].

[35] «Gazebo Models Database,» Open Source Robotics, [En línea]. Available: https://github.com/osrf/gazebo_models.

[36] «GazeboSim,» Open Robotics Foundation, [En línea]. Available: https://classic.gazebosim.org/tutorials?tut=actor&cat=build_robot. [Último acceso: 13 Mayo 2023].

[37] L.-T. H. G. Z. Weisong Wen, «Performance Analysis of NDT-based Graph SLAM for Autonomous Vehicle in Diverse Typical Driving Scenarios of Hong Kong,» *Sensors,* 2018.

[38] J. M. E. M. Kenji Koide, «A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement,» *International Journal of Advanced Robotic Systems,* 2019.

[39] K. Koide, «Github,» 2023. [En línea]. Available: https://github.com/koide3/hdl_graph_slam. [Último acceso: 2 Junio 2023].

[40] K. A. M. K. R. B. G. Granosik, «USING ROBOT OPERATING SYSTEM FOR AUTONOMOUS CONTROL OF ROBOTS IN EUROBOT, ERC AND ROBOTOUR COMPETITIONS,» *Acta Polytechnica CTU Proceedings,* vol. 6, 2016.

[41] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/V%C3%ADa_(automoci%C3%B3n).

[42] «Wikipedia,» 16 Mayo 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Batalla_(automoci%C3%B3n). [Último acceso: 24 Mayo 2023].