UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN SIMULADOR EN GAZEBO PARA EL ROBOT MÓVIL PANTER

DEVELOPMENT OF A GAZEBO SIMULATOR FOR THE PANTER MOBILE ROBOT

GRADO EN INGENIERÍA ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE MÁLAGA

Autor: Jorge Avila Orero

Tutor: Javier Serón Barba

Área de conocimiento: Ingeniería de Sistemas y Automática

Cotutor: David Padial Allue

Área de conocimiento: Ingeniería de Sistemas y Automática

Málaga, julio de 2025

Resumen

Los simuladores son herramientas clave en robótica, ya que permiten validar el comportamiento y avanzar en el desarrollo sin necesidad de pruebas físicas. En un entorno virtual podemos reproducir múltiples condiciones (terrenos irregulares, obstáculos, cambios de inclinación, etc.) y registrar con precisión el rendimiento de actuadores y sensores.

En este proyecto se simula el vehículo Panter, partiendo de su diseño en SolidWorks, al que se le aplica un aligeramiento de componentes no esenciales para optimizar la ejecución. El modelo se describe en URDF, lo que garantiza plena compatibilidad con ROS 2, y se ejecuta en Gazebo Ignition Fortress.

Para el control, se implementa en ROS 2 un nodo de teleoperación por teclado que aplica la cinemática de Ackermann en las ruedas delanteras y envía ya sea consignas de par a cada rueda o referencias de velocidad al vehículo. Además, se incorporan sensores de fuerza y par en cada articulación, lo que permite capturar datos en tiempo real para afinar y validar el comportamiento de Panter.

Palabras clave: Gazebo, ROS, simulador, Panter, Ackermann.

Abstract

Simulators are key tools in robotics because they allow us to validate behavior and advance development without the need for physical testing. In a virtual environment, we can recreate multiple conditions (uneven terrain, obstacles, changes in slope, etc.) and accurately record actuator and sensor performance.

In this project, the Panter vehicle is simulated starting from its SolidWorks design, from which non-essential components are stripped out to optimize performance. The model is described in URDF, ensuring full compatibility with ROS 2, and is executed in Gazebo Ignition Fortress.

For control, a keyboard teleoperation node is implemented in ROS 2 that applies Ackermann kinematics to the front wheels and sends either torque commands to each wheel or speed setpoints to the vehicle. Additionally, force-torque sensors are integrated into each joint, allowing real-time data capture to fine-tune and validate Panter's behavior.

Keywords: Gazebo, ROS, simulator, Panter, Ackermann.

Índice

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Antecedentes

Este Trabajo de Fin de Grado se enmarca en el proyecto Panter de la Universidad de Málaga, que persigue el desarrollo de un vehículo eléctrico de rescate a partir del modelo de Panter 4×4 de la marca Movelco. Los componentes del vehículo se modifican con el objetivo de mejorar el rendimiento y adaptarlo a misiones de emergencia en entornos no estructurados.

El vehículo es el mostrado en la Figura 1, que pertenece al Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga.

Figura 1. Vehículo Panter de Molveco

Para el desarrollo de este proyecto se toma como referencia el trabajo anterior realizado por el departamento: 'Modelado 3D de un vehículo eléctrico para misiones de rescate en

entornos no estructurados' [1].

El vehículo actualmente sufre múltiples modificaciones; la Figura 2 representa su estado actual.

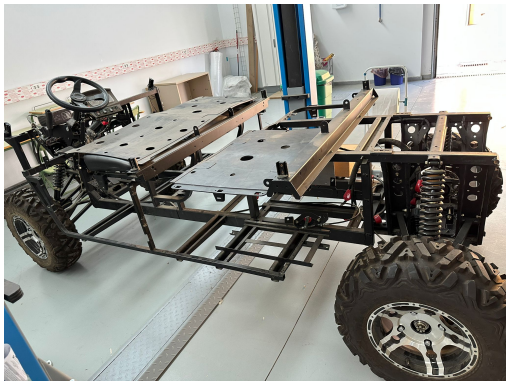

Figura 2. Reconstrucción Panter

1.2. Objetivos y tareas

El objetivo principal de este proyecto es crear un simulador del robot móvil Panter para utilizarlo como herramienta de desarrollo del vehículo. Este simulador se basa en la plataforma de simulación Gazebo Fortress y se implementa la interoperabilidad con el middleware ROS 2 Humble, lo que permite gestionar tanto el control como la obtención de datos del robot Panter.

Para conducir Panter se definen dos modos de operación de control por teclado:

- Plugin direccionamiento de Ackermann de Gazebo, controlado mediante velocidad.
- Controlador de esfuerzo y dirección, utilizando el framework ros2_control aplicado a cada rueda.

Como objetivo secundario, es obtener datos de par que se producen en las ruedas, de modo que sea posible analizar cómo responde 'Panter' en diferentes escenarios.

A continuación se describen las principales tareas realizadas:

1. Investigar, seleccionar y configurar ROS 2 y Gazebo, asegurando su correcta compatibilidad.

2. Aprendizaje de SolidWorks para la modificación de modelado 3D.

3. Investigación y desarrollo de descripción robot en URDF.

4. Investigación de estructura y jerarquía necesaria para el desarrollo del proyecto.

5. Diseñar el entorno de simulación en Gazebo, incluyendo el world en SDF y la configuración inicial.

6. Programar la comunicación entre ROS 2 y Gazebo, implementando los nodos necesarios para el control y la lectura de datos.

7. Crear los ficheros de lanzamiento de los programas.

8. Verificar y experimentar el correcto funcionamiento del simulador en distintos escenarios de prueba.

9. Elaborar la documentación detallada del proyecto.

1.3. Motivación

La motivación de este proyecto nace de mi deseo de involucrarme en una experiencia de robótica real al finalizar el Grado en Ingeniería Electrónica, Robótica y Mecatrónica, con Mención en Robótica y Automatización. A través de este trabajo, tengo la oportunidad de explorar un área totalmente nueva para mí: la creación de un simulador desde cero. Esto complementa los conocimientos que ya adquirí durante la carrera sobre montaje y programación de robots, y me permite profundizar en las principales fases de desarrollo de un proyecto robótico, definidas en la Figura 3.

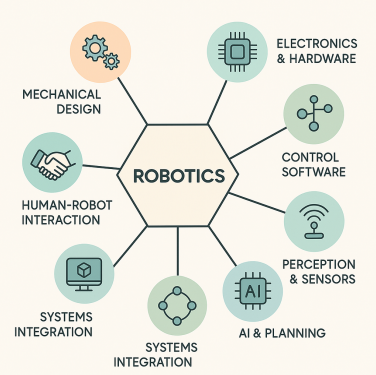

Figura 3. Campos en robótica

En este proyecto se emplean las herramientas más extendidas en la actualidad para la simulación de robots y el desarrollo de middleware robótico, lo que añade un valor profesional significativo.

1.4. Contenido de la memoria

Capítulo 1. Introducción. En este capítulo se abordan los antecedentes, las tareas y objetivos, la motivación, el contenido de la memoria y la terminología empleada.

Capítulo 2. Estado del Arte. Se lleva a cabo un estudio preliminar de los temas vinculados al proyecto.

Capítulo 3. Desarrollo del Simulador. Expone el contenido del proyecto, cómo se lleva a cabo, su estructura y jerarquía.

Capítulo 4. Resultados.

Capítulo 5. Conclusiones.

1.5. Terminología

A continuación se explican brevemente algunos de los términos empleados durante el proyecto:

- **Formato XML** (eXtensible Markup Language): es un lenguaje que emplea etiquetas para representar datos de forma legible por humanos y máquinas. En robótica se utiliza

para estructurar la descripción de componentes (enlaces, articulaciones, inercia, colisiones, visuales) de manera modular y extensible.

- **Python**: es el lenguaje preferido en ROS 2 para automatizar la generación y manipulación de archivos de descripción URDF, lanzar nodos y gestionar configuraciones.
- **URDF**: es el formato de archivo XML que se utiliza para describir la geometría y la estructura de un robot, nativo para ROS.
- **SDF**: es un estándar XML diseñado para describir mundos y modelos en simulación de robots, nativo en Gazebo.
- Archivos con extensión **.stl** (STereoLithography): fichero de malla 3D, ampliamente usado para representar la geometría superficial de piezas CAD.
- Archivos con extensión **.yaml**: son ficheros de texto estructurado, legibles por humanos, muy usados en robótica y en ROS2 para representar datos de configuración y parámetros.
- Archivos con extensión **.rviz**: son los archivos de configuración de la herramienta RViz. Permite cargar una interfaz ya configurada.
- Archivo con extensión **.world**: archivos que representan el entorno modificable (mundo) en Gazebo utilizando lenguaje XML. Es equivalente a SDF.
- Ficheros con extensión **.hpp**: son los archivos de cabecera C++, donde se declaran las interfaces del código. Aquí se definen las clases, métodos públicos y privados, atributos y constructores, sin incluir la implementación de cada función.
- Ficheros con extensión **.cpp**: Archivos C++ que contienen la implementación del código. Define los métodos y funciones, configura publicadores, suscriptores, timers y callbacks. Aquí se encuentra toda la lógica de ejecución del nodo.
- Archivo con extensión **.launch.py**: scripts en Python que orquestan el arranque de aplicaciones, nodos, procesos, parámetros y argumentos de entrada, y se pueden incluir otros archivos de lanzamiento.

- **Terminal**: también conocida como consola de texto. Está incorporada en el sistema operativo, permite comunicarnos a través del teclado. Será utilizada como intezfaz de la línea de comandos y del texto para el usuario. Para abrir una terminal de texto en Ubuntu: ctrl + alt + t.

Capítulo 2

Estado del Arte

2.1. Robótica móvil

La robótica móvil se encarga del diseño, la construcción y el control de máquinas autónomas o teledirigidas que se desplazan en entornos dinámicos. Estos robots tienen sistemas de percepción y navegación como LIDAR, cámaras y sensores inerciales que les permiten localizarse, trazar rutas óptimas y adaptarse en tiempo real a obstáculos.

Los robots móviles adoptan diversas arquitecturas y plataformas según la aplicación y el entorno en que operen. Los tipos más comunes son:

- **Ruedas diferenciales**: compuesto por dos ruedas motrices y ruedas locas de apoyo; muy usados en investigación y educación. (Por ejemplo: TurtleBot, Roomba).

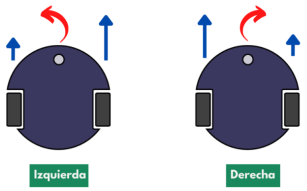

Figura 4. Direccionamiento Diferencial

- **Omnidireccionales** (ruedas Mecanum o esferas): permiten desplazamientos laterales y giros en el sitio; es ideal en espacios reducidos. (Por ejemplo: MiR100, Clearpath Ridgeback).

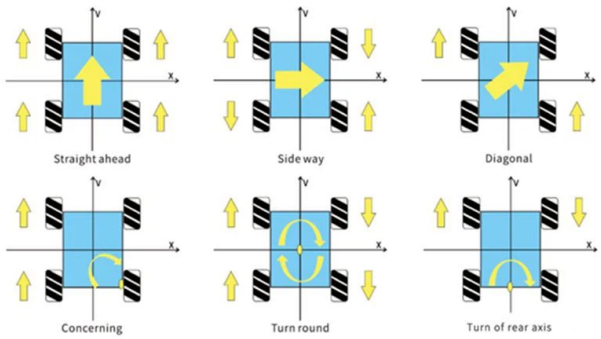

Figura 5. Direccionamiento con ruedas Mecanum

- **Ackermann**: imitan la geometría de dirección de un coche, con ruedas delanteras giratorias y traseras fijas; ideales para pruebas de conducción y mapeo en entornos reales. (Por ejemplo: vehículo común).

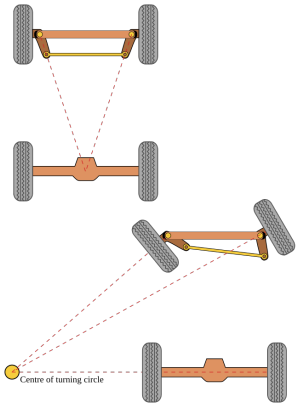

Figura 6. Direccionamiento de Ackermann

- **Robots con orugas**: tienen cadenas a ambos lados del vehiculo, se desplazan de forma diferencia, tienen alta tracción y estabilidad en terrenos irregulares o blandos. (Por ejemplo: Husky UGV, Pioneer 3-AT).
- **Robots con patas** (cuádrupedos, bípedos, hexápodos): imitan la locomoción de un animal. Superan obstáculos complejos y desniveles.
- **Drones multirrotor (UAVs)**: vehículos aéreos para inspección y cartografía.

- **Vehículos submarinos (AUV/ROV)**: sumergibles autónomos o guiados para exploración y salvamento.

2.2. Simulador de vehículos

Los simuladores de vehículos robóticos son entornos virtuales que imitan el comportamiento físico, sensorial y de control de robots móviles, desde pequeñas plataformas diferenciales hasta vehículos autónomos complejos. Su principal ventaja es permitir el desarrollo y la validación de algoritmos de navegación, percepción y actuación sin recurrir a costosos prototipos reales, reduciendo riesgos y acelerando ciclos de diseño.

Posibles simuladores

Las plataformas más destacadas para la simulación de vehículos robóticos incluyen:

- **Webots**: incluye un editor gráfico 3D, control en Python, C/C++ y MATLAB, y ejemplos preconfigurados para plataformas diferenciales y Ackermann. Su licencia Apache 2.0 facilita su uso académico y en proyectos de investigación.

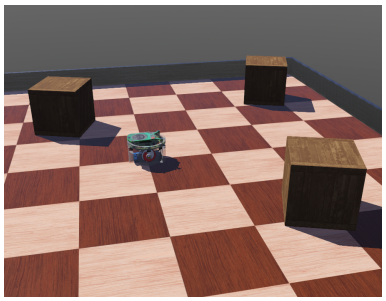

Figura 7. Simulador Webots

- **CoppeliaSim (V-REP)**: permite scripting distribuido en Lua, Python o C/C++, soporta motores de física avanzados (MuJoCo, Bullet, ODE, Vortex, Newton) y ofrece herramientas para ajustar trayectorias y parámetros dinámicos.

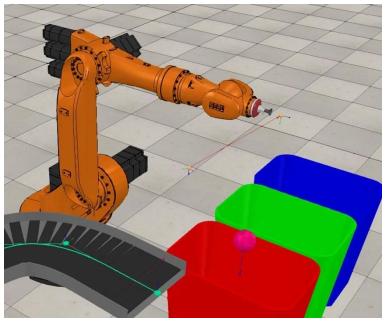

Figura 8. Simulador CoppeliaSim

- **CARLA**: construido sobre el motor de videojuegos Unreal Engine, proporciona entornos urbanos con semáforos, peatones y tráfico controlado por IA.

Figura 9. Simulador CARLA

- **Unity**: potente motor 3D con soporte para C#, con amplia comunidad y recursos. Mediante paquetes como Unity Robotics Hub se integra con ROS 2, permite personalizar físicas y gráficos. Facilita la creación de interfaces para visualizar y controlar robots. Ideal para desarrollos que requieran alta fidelidad visual y entornos interactivos.

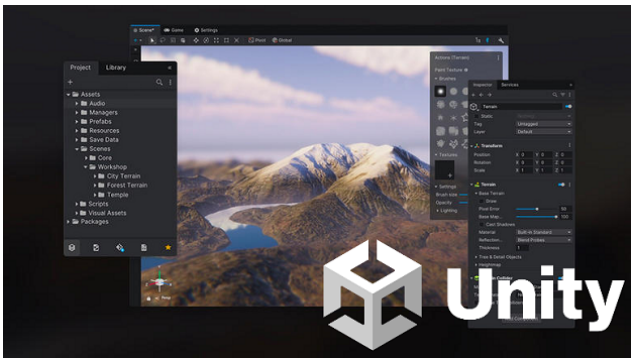

Figura 10. Simulador Unity

- **Gazebo / Ignition Gazebo**:

  Desarrollado por Open Robotics, es el simulador robótico más utilizado y extendido. Al pertenecer a la misma organización responsable de ROS, la integración nativa con este framework lo convierte en la mejor opción para el desarrollo y prueba de sistemas robóticos.

  Gazebo ha evolucionado a lo largo de tres grandes familias de versiones.

  La primera, Gazebo Classic, nació en 2002 y abarca las versiones numeradas del 1 al 11; incorporó de serie el motor ODE, añadió más tarde Bullet y DART, y estableció la integración nativa con ROS 1 y ROS 2.

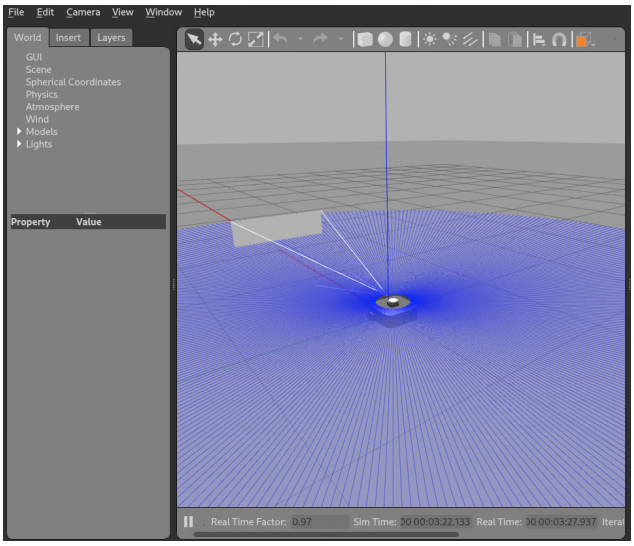

Figura 11. Gazebo Classic

En 2019 surgió Ignition Gazebo, que desmontó el monolito de Classic en componentes independientes de física, renderizado y sensores para mejorar la escalabilidad y el mantenimiento; sus principales lanzamientos son Acropolis, Blueprint, Citadel, Dome, Edifice y Fortress.

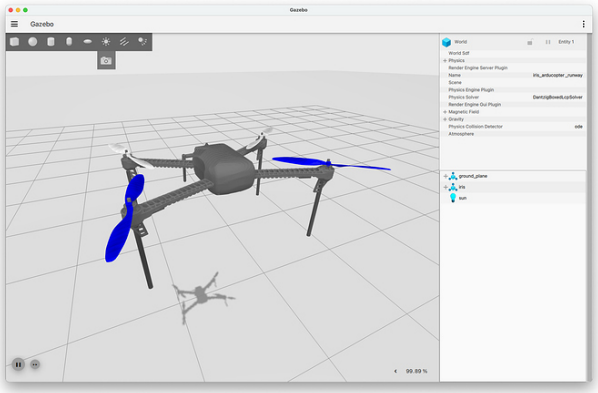

Figura 12. Ignition Gazebo

Finalmente, en 2023, Ignition Gazebo se rebautizó como Gazebo Sim, conservando la arquitectura modular de Ignition, adoptando un versionado semántico y garantizando la compatibilidad binaria durante todo su ciclo de vida; entre sus versiones más relevantes figuran Garden y Harmonic.

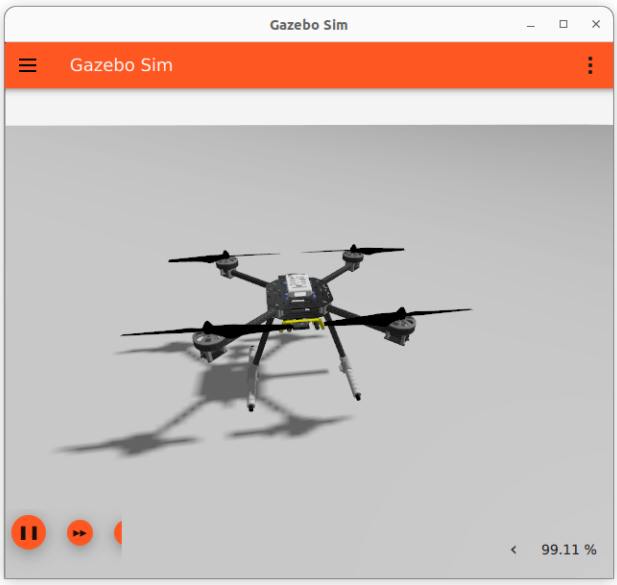

Figura 13. Gazebo Sim

Esta trayectoria muestra cómo Gazebo ha pasado de un simulador monolítico a una colección de bibliotecas independientes, a una plataforma unificada, sencilla de instalar y mantener.

2.3. Middleware robótica

Un middleware de robótica es un software encargado de simplificar la complejidad de la comunicación, coordinación y gestión de los distintos componentes de un sistema robótico.

En lugar de que cada nodo, sensor o actuador implemente su propio mecanismo de transmisión de datos y sincronización, el middleware ofrece servicios comunes (paso de mensajes, sincronización de relojes, descubrimiento de nodos, gestión de parámetros) que facilitan el desarrollo modular, escalable y distribuido de aplicaciones.

Se podría comparar un middleware de robótica con un "sistema operativo" específico para robots: proporciona infraestructuras de comunicación (publish/subscribe, request/response, transferencia de ficheros o parámetros), mecanismos de serialización y deserialización de datos, y herramientas de monitorización y diagnóstico.

Un middleware de robótica no solo acelera el desarrollo de software para robots, sino que aporta robustez, escalabilidad y modularidad a arquitecturas que, de otro modo, serían difíciles de mantener y evolucionar.

En la actualidad, contar con un middleware sólido ya no es una opción, sino un requisito casi imprescindible para cualquier proyecto robótico.

Posibles Middlewares

- **ROS 2** se posiciona hoy como la referencia indiscutible en robótica general: combina un ecosistema muy maduro con las ventajas del estándar DDS (QoS, descubrimiento, escalabilidad). Es la opción recomendada para la mayoría de proyectos académicos e industriales que requieran flexibilidad, compatibilidad y soporte comunitario.

- **YARP** sigue siendo una opción sólida en el ámbito de los humanoides gracias a su flexibilidad a la hora de conectar puertos en tiempo de ejecución y su eficiencia en entornos de procesamiento visual.

- **LCM** destaca en aplicaciones embebidas donde la latencia debe minimizarse al máximo, como robots de competición o sistemas de control ultra-precisos.

- **Orocos RTT** es bueno para desarrollos que demandan bucles de control en tiempo real,

especialmente en robots manipuladores o prótesis.

- **ROS-Industrial** resulta la alternativa más consolidada para introducir ROS en entornos de automatización industrial, aprovechando la colección de drivers de PLC y protocolos de campo.
- Los middlewares emergentes como **Micro-ROS o Locus** son cada vez más relevantes en IoT y microcontroladores, abriendo la puerta a integrar pequeños dispositivos con arquitecturas ROS 2.

2.4. Descripción del robot

La descripción del robot es el conjunto de archivos y configuraciones que definen su estructura mecánica, sus articulaciones, geometría, propiedades físicas y sensores. Sirve como vínculo entre el modelo virtual, el simulador y el middleware, permitiendo reproducir con precisión tanto la forma como el comportamiento dinámico del robot.

Para la simulación de robots, conviene analizar dos formatos de descripción que suelen emplearse: URDF y SDF. Cada uno presenta fortalezas y limitaciones que impactan directamente en la interoperabilidad y en la fidelidad de la simulación.

- **SDF (Simulation Description Format)**
  - **Características principales**
    - Es el formato nativo de Gazebo e Ignition Fortress, basado en XML, pensado para describir no solo la cinemática de un robot, sino también todos los elementos que componen un "mundo" de simulación: terrenos, luces, obstáculos y propiedades físicas.
    - Permite definir enlaces (*<link>*) y articulaciones (*<joint>*) con geometrías tanto para visualización como para colisión, especificando parámetros avanzados de física (coeficientes de fricción estática y dinámica, amortiguamiento, etc.).
    - Incluye de forma nativa la posibilidad de agregar sensores (cámaras, LiDAR, IMU), así como plugins personalizados de control o de percepción, y bloques *<world>* completos para configurar el entorno.

- **Ventajas**
  - Ofrece un modelado físico detallado y realista: se pueden ajustar con precisión los parámetros de inercia, fricción y motores de física.
  - Facilita la inclusión y configuración de múltiples sensores sin necesidad de recurrir a archivos externos o plugins adicionales.
  - Agrupa en un mismo archivo la descripción del robot y del entorno, simplificando el mantenimiento de escenarios complejos.
- **Limitaciones**
  - No es interpretado de forma nativa por ROS 2: para integrar un modelo SDF en un nodo de ROS 2 es necesario usar herramientas como *ros_gz_bridge* o paquetes específicos (*gz_ros2_control*), lo que añade complejidad en la configuración inicial.
  - No existe un equivalente a XACRO en SDF, de modo que repetir estructuras (por ejemplo, múltiples ruedas idénticas) puede resultar más pesado.
  - La curva de aprendizaje es más pronunciada: aprender todas las etiquetas y jerarquías de SDF requiere más tiempo que en URDF.

• **URDF (Unified Robot Description Format)**

- **Características principales**
  - Diseñado para integrarse de forma nativa con todo el ecosistema ROS (ROS 1 y ROS 2), prioriza la definición de la cinemática y la geometría visual/colisión de enlaces y articulaciones.
  - Su sintaxis basada en XML, junto a la herramienta XACRO, facilita parametrizar dimensiones y reutilizar fragmentos (por ejemplo, definir una plantilla para cada rueda o sensor).
  - En URDF se describe la masa, el centro de gravedad y el tensor de inercia de cada enlace.
- **Ventajas**

- Integración directa con nodos y paquetes de ROS 2.
- Más simple y legible para describir la estructura cinemática pura de un robot: fácil de depurar y editar cuando solo se trabajan aspectos geométricos o de articulaciones.
- Gracias al paquete *gazebo_ros*, Gazebo convierte automáticamente un URDF en SDF al iniciar la simulación, de modo que se puede seguir trabajando sobre URDF y aprovechar las ventajas de SDF sin esfuerzo extra.

– **Limitaciones**

- URDF no incluye nativamente bloques para especificar sensores complejos, ni detalles avanzados de física. Para ello, es necesario insertar en el URDF etiquetas *<gazebo>* con fragmentos de SDF embebidos (por ejemplo, para usar un plugin de Gazebo).
- Cuando se requiere modelar un entorno completo (terreno, iluminación, obstáculos estáticos), el URDF por sí mismo resulta insuficiente: se debe combinar con un archivo SDF de mundo o incluir fragmentos *<gazebo>*.

Capítulo 3

Desarrollo del simulador

3.1. Entorno de trabajo y herramientas

En esta sección se describe el entorno de trabajo utilizado en el proyecto, así como las herramientas y conceptos básicos que se emplean.

Se utilizan los siguientes entornos para el desarrollo del proyecto:

- Sistema operativo Ubuntu 22.04.5 LTS.
- Gazebo Ignition Fortress para la simulación.
- El middleware robótico empleado es ROS2, versión Humble.
- Para el modelado 3D del robot se emplea SolidWorks y para el escenario de Gazebo, Blender.
- Para la descripción del robot en Gazebo, URDF (Unified Robot Description Format).
- Para la descripción del entorno en Gazebo, SDF (Simulation Description Format).
- Visual Studio Code como entorno de programación.
- Git y GitHub, para el control de versiones del código y documentación del proyecto.

A continuación, se explican los programas anteriores y por qué han sido elegidos para desarrollar el proyecto.

ROS 2

Desarrollado por Open Robotics, actúa como plataforma unificada para el software y el hardware del robot, funcionando a modo de sistema operativo. ROS 2 surge como evolución de ROS 1, adoptando DDS (Data Distribution Service) como capa de transporte y un modelo de comunicación distribuida basado en topics (publicador-suscriptor) y servicios (cliente-servidor), garantizando una entrega de mensajes fiable. Cada funcionalidad (navegación, mapeo, localización, simulación, etc.) se implementa como un paquete reutilizable, agilizando tanto el desarrollo como el despliegue de soluciones en plataformas robóticas variadas.

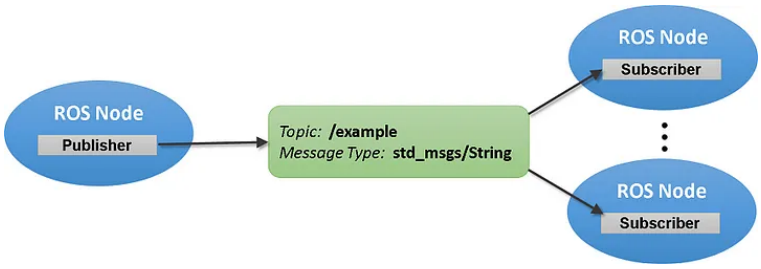

Figura 14. Esquema de funcionamiento ROS

Para este proyecto se utiliza la distribución LTS ROS 2 Humble Hawksbill, debido a que esta es la versión utilizada en el departamento y en los proyectos realizados sobre Panter. A su vez, es una versión con soporte garantizado hasta mayo de 2027. Humble Hawksbill está optimizada para ejecutarse de forma nativa en Ubuntu 22.04 Jammy Jellyfish, el sistema operativo utilizado, lo que asegura máxima compatibilidad, estabilidad y acceso a las últimas mejoras del ecosistema ROS 2.

Figura 15. ROS 2 Humble

Gazebo

Para garantizar un entorno de desarrollo robusto y coherente, se necesita escoger un middleware y un simulador que ofrezcan integración nativa, instalación sencilla y soporte a largo plazo. Gazebo ha sido el simulador seleccionado para este proyecto, porque junto con ROS ha sido desarrollado por Open Robotics, consiguiendo una conexión fluida entre ambos. Esto facilita el intercambio de datos entre nodos de software y componentes de simulación, acelera la configuración de escenarios virtuales y reduce el riesgo de incompatibilidades durante el desarrollo.

Además, al basarse en distribuciones con soporte a largo plazo, se asegura estabilidad en el ciclo de vida del proyecto y simplifica la gestión de actualizaciones y dependencias.

Para la elección de la versión de Gazebo, hay que tener en cuenta la compatibilidad con ROS 2 Humble. Como muestra la Figura 16, la versión de Gazebo con mayor compatibilidad es Gazebo Fortress y, por lo tanto, ha sido la versión de Gazebo seleccionada.

	GZ Fortress (LTS)	GZ Harmonic (LTS)	Gz Ionic
ROS 2 Rolling	❌	⚡	✅
ROS 2 Kilted	❌	⚡	✅
ROS 2 Jazzy (LTS)	❌	✅	❌
ROS 2 Humble (LTS)	✅	⚡	❌
ROS 1 Noetic (LTS)	⚡	❌	❌

- ✅ - Recommended combination
- ❌ - Incompatible / not possible.
- ⚡ - Possible, *but use with caution*. These combinations of ROS and Gazebo can be made to work together, but some effort is required.

Figura 16. Tabla compatibilidad ROS-Gazebo [2]

Tanto Humble como Fortress son distribuciones LTS (Long Term Support). Tiene soporte garantizado hasta mayo de 2027 para ROS 2 Humble y hasta septiembre de 2026 para Gazebo Fortress. La documentación oficial recomienda emparejar siempre las versiones LTS de ROS y Gazebo para asegurar estabilidad a largo plazo en desarrollo y despliegue.

Gazebo Fortress [3], lanzado en julio de 2023, inicia la línea Ignition Gazebo con una arquitectura completamente modular. Cada elemento de simulación: sensores, motores de física, renderizado, funciona como un componente independiente que puede desplegarse y actualizarse sin interferir en el resto, lo que acelera el desarrollo y permite escalar simulaciones complejas.

Para la física, ofrece múltiples motores (Bullet, DART, Simbody, TPE) que pueden seleccionarse en tiempo de ejecución según las necesidades de precisión o velocidad.

Figura 17. Gazebo Ignition Fortress

SolidWorks

Es el software de diseño asistido por computadora (CAD) 3D utilizado para la creación del modelo de Panter; su elección se debe a que es el programa de diseño con el que se venía trabajando en los proyectos alrededor de Panter.

Ha sido utilizado para modificar los elementos del robot y, finalmente, obtener las mallas (meshes) que permiten la visualización de Panter en Gazebo.

Visual Studio Code

Visual Studio es el editor de código fuente utilizado, ya que se trata de un entorno de desarrollo que es ligero y compatible con múltiples sistemas operativos, y ampliable mediante extensiones. Soporta una gran variedad de lenguajes.

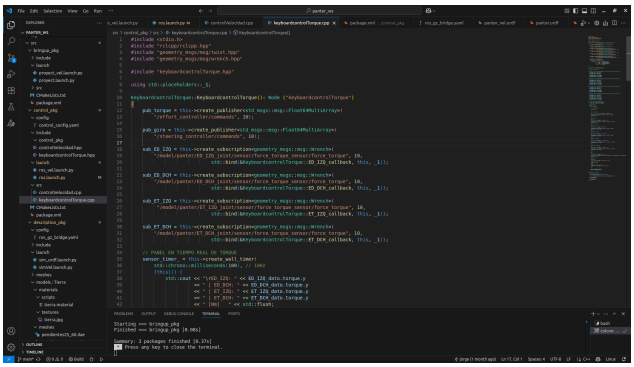

Figura 18. Proyecto en Visual Studio

Git y GitHub

He seleccionado Git como sistema de control de versiones distribuido que registra el historial de un proyecto, ya que permite trabajar de forma paralela mediante ramas y fusionar cambios sin comprometer la estabilidad del código.

Figura 19. Logo Git

GitHub es la plataforma web que almacena los repositorios Git. Esta colaboración facilita un flujo de trabajo estructurado y colaborativo en proyectos.

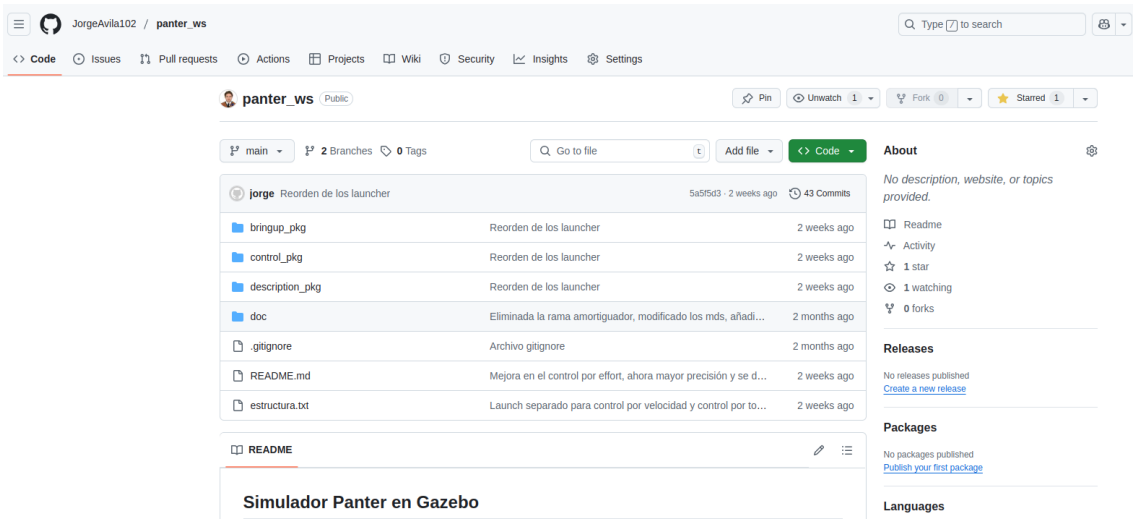

Figura 20. Repositorio proyecto simulador en GitHub [4]

RViz

RViz2 (ROS Visualization 2) es la herramienta de visualización en tiempo real de ROS 2. Permite representar de manera interactiva y configurable los datos que fluyen por los distintos topics de un robot: como estados de articulaciones y modelos URDF, mapas de navegación, trayectorias planificadas y sensores LIDAR o cámaras.

3.2. Modelo SolidWorks

El proyecto parte de un modelado en SolidWorks del vehículo Panter, el cual se modifica para que su estructura se adecúe a cómo será el robot Panter real cuando su desarrollo finalice, y que a su vez se adapta para que la simulación sea fluida.

Aunque el modelo proporcionado incluye prácticamente todas las piezas del robot, algunos componentes resultan innecesarios en la simulación porque no aportan valor físico (por ejemplo: baterías y cableado). Al eliminarlos, aligeramos la carga computacional y mejoramos notablemente el rendimiento.

El modelo de Panter utilizado para la simulación está compuesto por los siguientes elementos:

- **P-CH**: chasis del robot.
- **P-SD01**: amortiguadores delanteros (derecho e izquierdo).
- **P-ST01**: amortiguadores traseros (derecho e izquierdo).
- **P-SD02**: trapecios superiores de suspensión delantera (derecho e izquierdo).
- **P-ST02**: trapecios superiores de suspensión trasera (derecho e izquierdo).
- **P-SD03**: trapecios inferiores de suspensión delantera (derecho e izquierdo).
- **P-ST03**: trapecios inferiores de suspensión trasera (derecho e izquierdo).
- **MGD**: manguetas delanteras (derecha e izquierda).
- **MGT**: manguetas traseras (derecha e izquierda).
- **P-ED**: ruedas delanteras (derecha e izquierda).
- **P-ET**: ruedas traseras (derecha e izquierda).

Una vez que el número de elementos de robot se reduce, es necesario recortar la longitud del robot 40 cm, con el objetivo de que tenga las mismas proporciones que el robot real.

Para ello, el método que se realiza es extruir 40 cm todas las barras horizontales de la base de panter, obteniendo como resultado la Figura 21, y posteriormente, desplazar todos los elementos a su posición correcta, utilizando relaciones de posición.

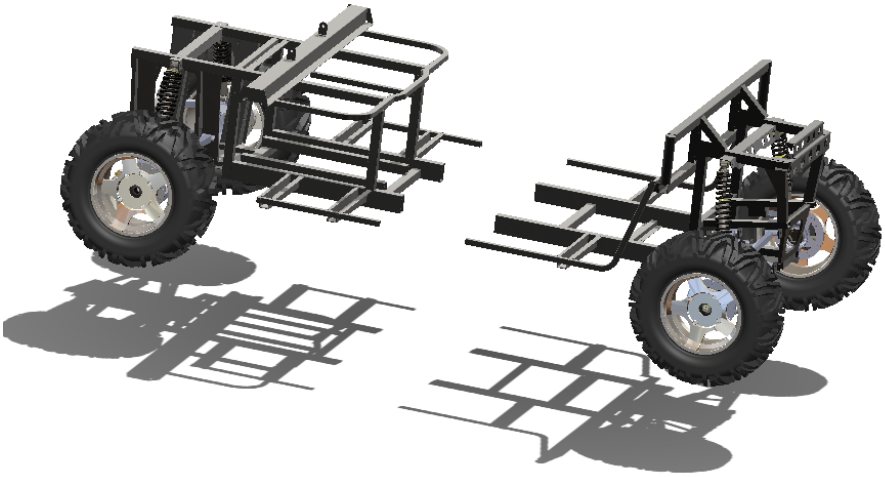

Figura 21. Panter extruido

En la Figura 22 se observa el estado final de panter con el que se realizará su descripción para el simulador.

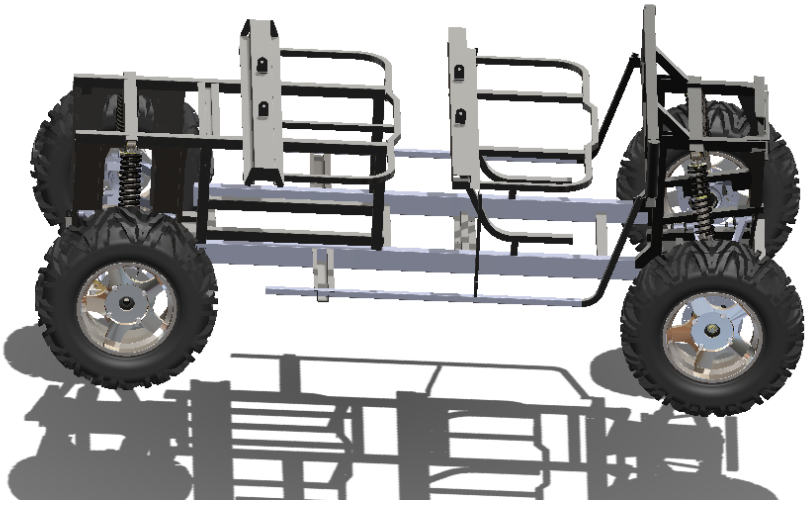

Figura 22. Panter reconstruido

Exportar de SolidWorks a URDF

URDF (Unified Robot Description Format), definido en el Apartado 3.4 es el estándar de descripción utilizado en el proyecto.

Tras modelar el robot en SolidWorks, se instala el plugin SolidWorks to URDF Exporter [5], que aprovecha el árbol jerárquico del ensamblaje para:

- Definir semi-automáticamente los links y joints, con la interfaz de la Figura 23.
- Generar los archivos .stl de cada componente.
- Exportar la descripción al formato URDF.

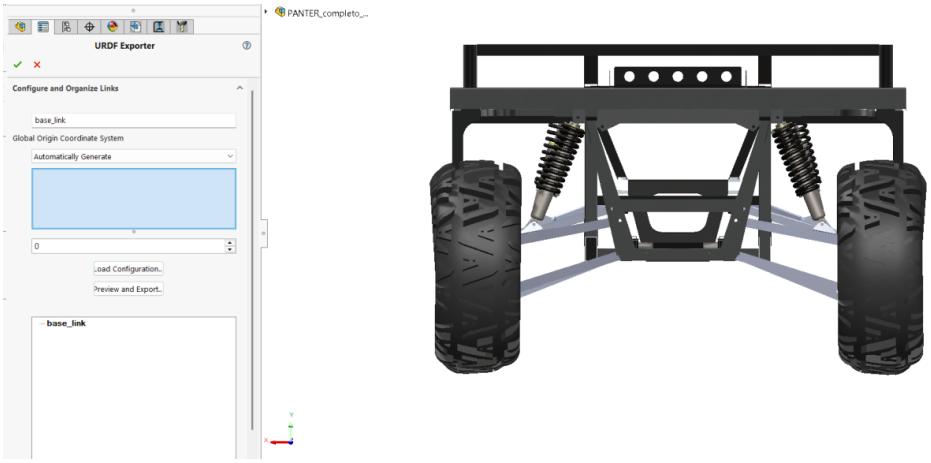

Figura 23. Interfaz herramienta SW2URDF

Sin embargo, al hacer pruebas con la exportación semi-automática, surgen varios inconvenientes, por lo que se llevan a cabo estas correcciones:

- **Ejes de rotación mal definidos**: el plugin no alinea correctamente los ejes de los joints. Para solventarlo, se crean los sistemas de coordenadas locales en cada pieza de SolidWorks, orientando los ejes según el movimiento deseado.
- **Sistema de coordenadas global**: al intentar fijarlo manualmente con la dirección deseada, el 'plugin' muestra errores. Finalmente se deja que lo genere de forma automática.

- **Parámetros físicos y visuales**: los valores de masa e inercia incorrectos, el tipo de 'joint' de los diferentes elementos se mezclan, y los colores no se exportan. Se recalcula y ajusta externamente al incorporarlos al URDF.

A pesar de estas dificultades, el plugin es de gran utilidad porque:

1. Proporciona una base URDF de partida, reduciendo el trabajo manual inicial.
2. Ofrece los modelos de cada pieza en formato .stl.
3. Facilita la configuración de posiciones relativas, sistemas de coordenadas y joints en el ensamblaje.

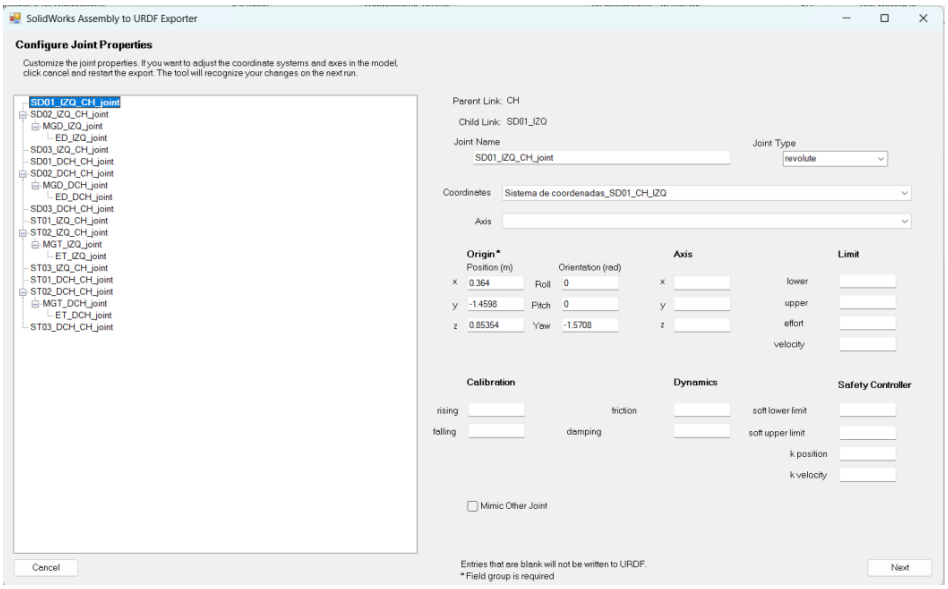

Figura 24. Configuración árbol de entidades, sw2urdf exporter [6]

En la parte izquierda de la Figura 24, se observa el árbol jerárquico de entidades, y en el lado derecho, características para establecer una de las articulaciones.

3.3. Jerarquía del proyecto

El proyecto se apoya en la plantilla oficial de ROS/Gazebo [7] presentada en la conferencia 'ROSCon' [8].

El espacio de trabajo de este proyecto, denominado *panter_ws*, se organiza en tres paquetes, como muestra la Figura 25, con funciones claras y diferenciadas:

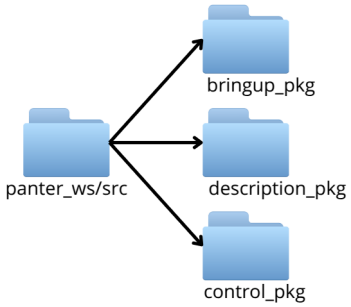

Figura 25. Paquetes proyecto

Paquete de control

El paquete *control_pkg* agrupa toda la lógica de conducción de Panter en ROS 2. La Figura 26 muestra su estructura, la cual contiene:

- **config/**: contiene el archivo .yaml de configuración de los controladores del frame_work *ros2_control*, explicado en el Apartado 3.8.2.
- **include/**: archivos de cabecera que definen las interfaces de los controladores.
- **launch/**: scripts en Python que automatizan la puesta en marcha de los nodos y la configuración inicial.
- **src/**: contiene la lógica de control por velocidad y por par.

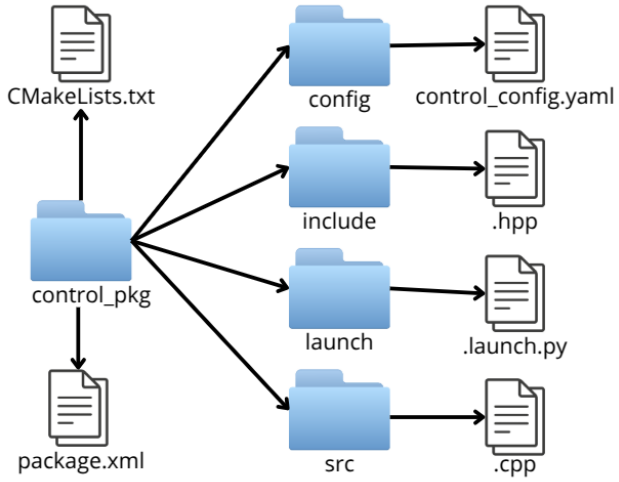

Figura 26. Estructura paquete control_pkg

Paquete de descripción

El paquete *description_pkg* está estructurado en subdirectorios para separar cada tipo de recurso. La Figura 27 muestra su estructura, que contiene:

- **config/**: contiene el archivo *ros_gz_bridge.yaml*, que realiza el mapeo de topics entre Gazebo Ignition y ROS 2.
- **launch/**: incluye el script *sim_urdf.launch.py*, responsable de iniciar Gazebo con el modelo URDF y cargar parámetros adicionales de simulación.
- **meshes/**: contiene los modelos CAD exportados en formato .stl para cada pieza de Panter (chasis, ruedas, suspensiones, etc.).
- **rviz/**: el fichero *'panter.rviz'* establece la configuración de RViz 2 necesaria para visualizar el robot y su estado en tiempo real.
- **urdf/**: contiene los archivos .urdf que describen la geometría, la masa y la cinemática del vehículo.
- **worlds/**: escenarios de Gazebo diseñados para probar Panter en diferentes entornos y condiciones de terreno.

- **models/**: almacena los archivos COLLADA (.dae) que definen la geometría y las texturas del escenario de simulación personalizado, permitiendo recrear fielmente el entorno de pruebas.

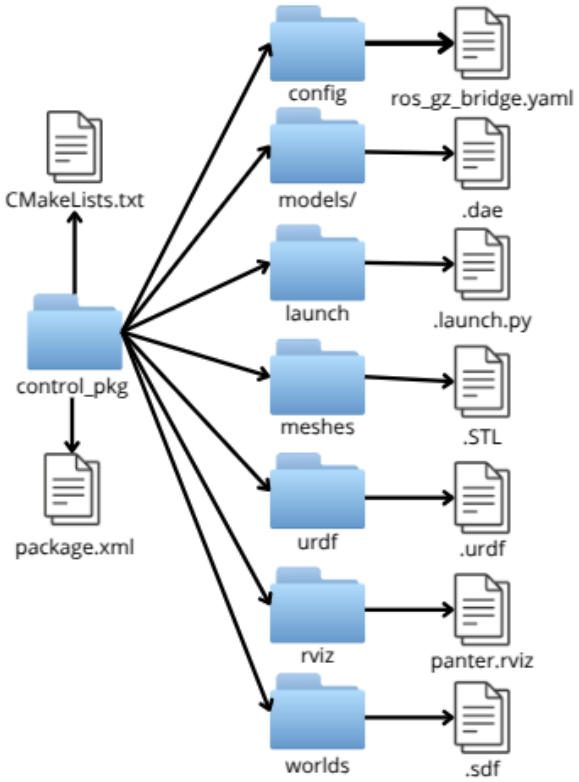

Figura 27. Estructura paquete description_pkg

Paquete de puesta en marcha

El paquete *bringup_pkg*, representado en la Figura 28, incluye dos lanzadores principales:

- **proyectVel.launch.py**: para el control por velocidad.
- **proyectTorque.launch.py**: para el control por par.

Ambos scripts de lanzamiento inician de forma secuencial todos los launchers y nodos de *description_pkg* y *control_pkg*, proporcionando un punto de entrada único y coherente para la simulación completa.

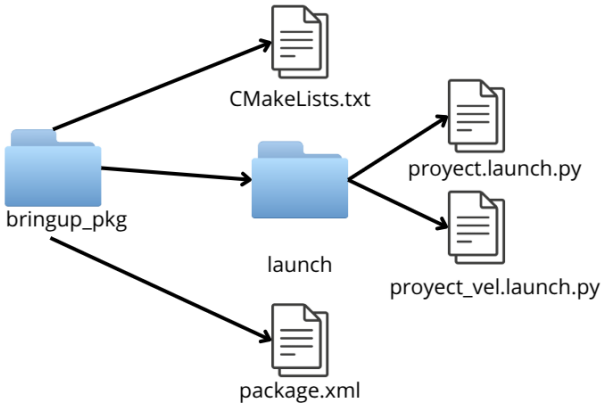

Figura 28. Estructura paquete bringup_pkg

Cada paquete incluye sus propios archivos CMakeLists.txt y package.xml, cuya configuración y contenidos se detallan en los Apartados 3.6.4 y 3.6.5.

3.4. Descripción Modelo Robot Panter

Para la simulación del vehículo móvil Panter se definen dos variantes de la descripción URDF, adaptadas a distintos modos de control: *panter.urdf* y *panter_vel.urdf*.

Estructura

Ambos archivos, *panter.urdf* y *panter_vel.urdf*, comparten los mismos componentes del robot, pero difieren en los plugins y bloques adicionales que especifican su modo de desplazamiento.

- **Links**: cada parte rígida del robot (chasis, amortiguadores, trapecios, manguetas, ruedas) se describe con un bloque <link> que incluye otros bloques:
  - **Inercial**: con la masa, centro de masa y el tensor de inercia para esa pieza.
  - **Visual**: referencia al archivo mesh (.stl) que se utilizará para el renderizado, así como el material o color aplicados.
  - **Collision**: describe la geometría que Gazebo emplea para calcular las colisiones de ese link con el entorno y con otros objetos.

La Figura 29 muestra la estructura de los elementos de un link.

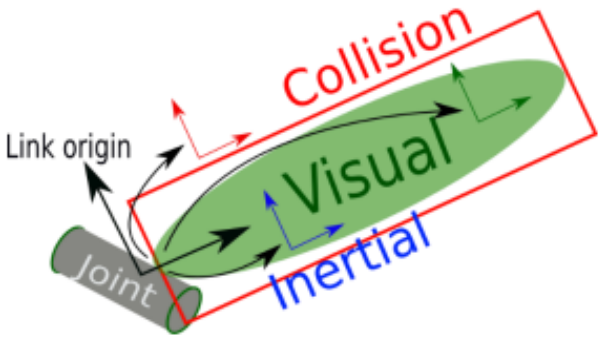

Figura 29. Links SDF

- **Joints**: cada articulación se define con un bloque <joint>, indicando:
  - Tipo de articulación que se establece entre dos links.
  - **Parent link**: el enlace rígido "padre", que actúa como soporte fijo para el movimiento.
  - **Child link**: el enlace rígido "hijo", que se desplaza (rota o traslada) en relación con el parent.
  - Límites de effort, velocidad y posición, que definen hasta dónde y con qué fuerza/-velocidad puede moverse esa articulación.
  - El eje sobre el que se produce el giro o la traslación.
  - La posición del joint en el espacio, es decir, las coordenadas relativas a alguno de sus links.

La Figura 30 representa dónde se encuentra cada elemento de un joint.

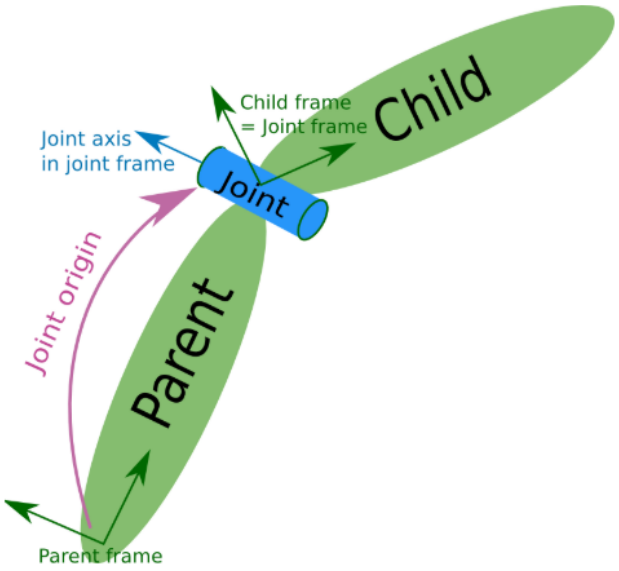

Figura 30. Joints SDF

- Se añade un sensor de esfuerzo y par en la articulación motriz de cada rueda, explicado en el Apartado 3.10, empleando un plugin nativo de Gazebo. Este plugin permite capturar durante la simulación los valores de fuerza y par aplicados a las articulaciones.
- A través de etiquetas <gazebo> en el URDF, se definen los parámetros de fricción de las ruedas de Panter, incluyendo los coeficientes de fricción estática y dinámica. De este modo, el modelo reproduce comportamientos más realistas al interactuar con el terreno.

La Figura 31 ilustra el modelo URDF de Panter en el simulador Gazebo, con la visualización de los ejes de sus articulaciones activada.

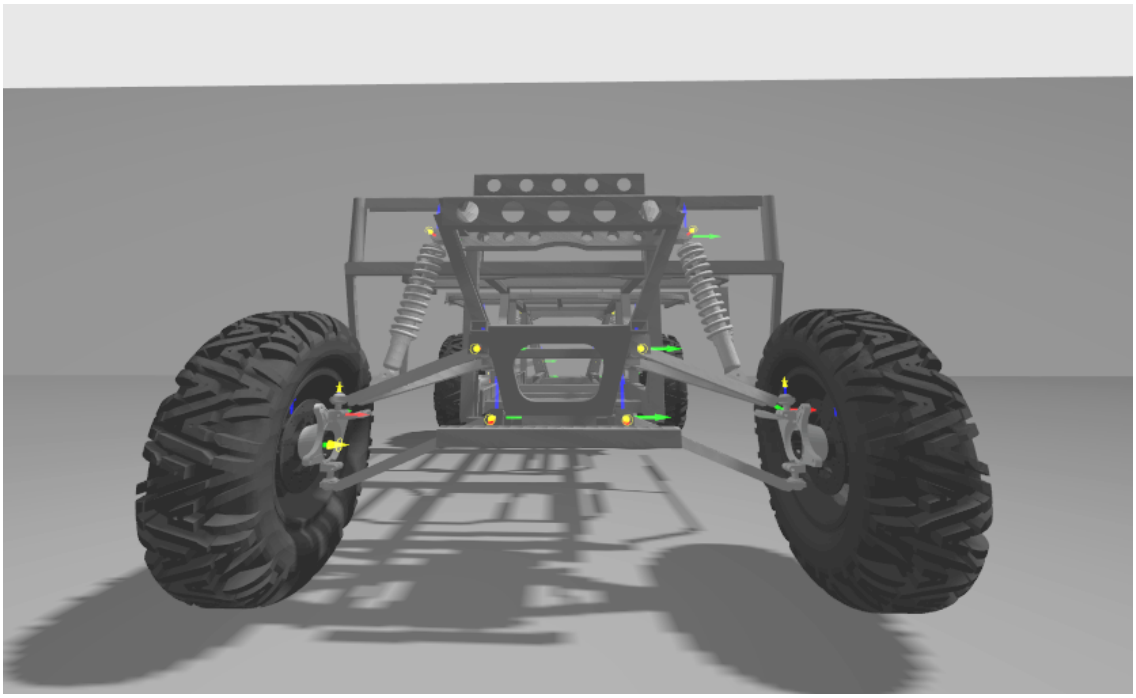

Figura 31. Visualización de los joints activos en Panter

Velocidades

La velocidad máxima del vehículo es 55 km/h [9], por ello, este valor es declarado en el parámetro *velocity* de las articulaciones que realizan el movimiento de las ruedas, con un valor de 15.28 m/s.

En cuanto a la velocidad máxima de giro de la dirección, experimentalmente se ha medido que al dar 3 giros de volante (desde el máximo a la izquierda hasta el máximo a la derecha), toma 1.6 s, por lo que la velocidad máxima de la dirección es de 11.78 rad/s, y se describe en la Ecuación 1.

$$\omega = 6\,\pi/1.6 \approx 11.78 rad/s \tag{1}$$

Elementos para el control

Luego, cada descripción se desarrolla por la necesidad de incluir una forma distinta de control sobre Panter:

- **panter.urdf**:

En el archivo *panter.urdf*, el movimiento del robot se controla enviando comandos de par a las ruedas a través de ROS 2. Para ello, se utiliza el framework *ros2_control*, un plugin de ROS 2 que permite regular los actuadores mediante par, posición o velocidad; este framework se detalla en el Apartado 3.6.3.

Con *ros2_control* se configuran dos controladores principales:

- **JointEffortController** para aplicar par a las ruedas motrices, garantizando un control fino del par en cada articulación.
- **JointPositionController** determinar el ángulo de las ruedas delanteras, implementando la cinemática de Ackermann y asegurando giros precisos y realistas.

Los comandos son enviados por los topics */effort_controller/commands* y */steering_controller/commands* con tipo de dato *std_msgs::msg::Float64MultiArray*.

- **panter_vel.urdf**

  En la descripción *panter_vel.urdf*, el movimiento del robot se controla enviando consignas de velocidad lineal y angular a través de ROS 2 por el topic *cmd_vel* . Para implementar este control se utiliza el plugin de direccionamiento Ackermann de Gazebo, que se encarga de traducir esas consignas en el movimiento del vehículo. Este plugin se describe en detalle en el apartado 3.10.

3.5. RViz

Para la configuración de RViz2, se crea un archivo de configuración personalizado (panter.rviz) que, utilizando el topic *robot_description*, carga automáticamente en cada lanzamiento el modelo de Panter y ajusta todos los parámetros de visualización previamente establecidos.

Mediante el nodo *joint_state_publisher*, es posible manipular en tiempo real la posición de cada articulación mediante controles deslizantes (ver Figura 32). Esta funcionalidad interactiva es clave al desarrollar el URDF de Panter, ya que facilita la comprobación visual de que todos los links y joints están correctamente definidos y conectados.

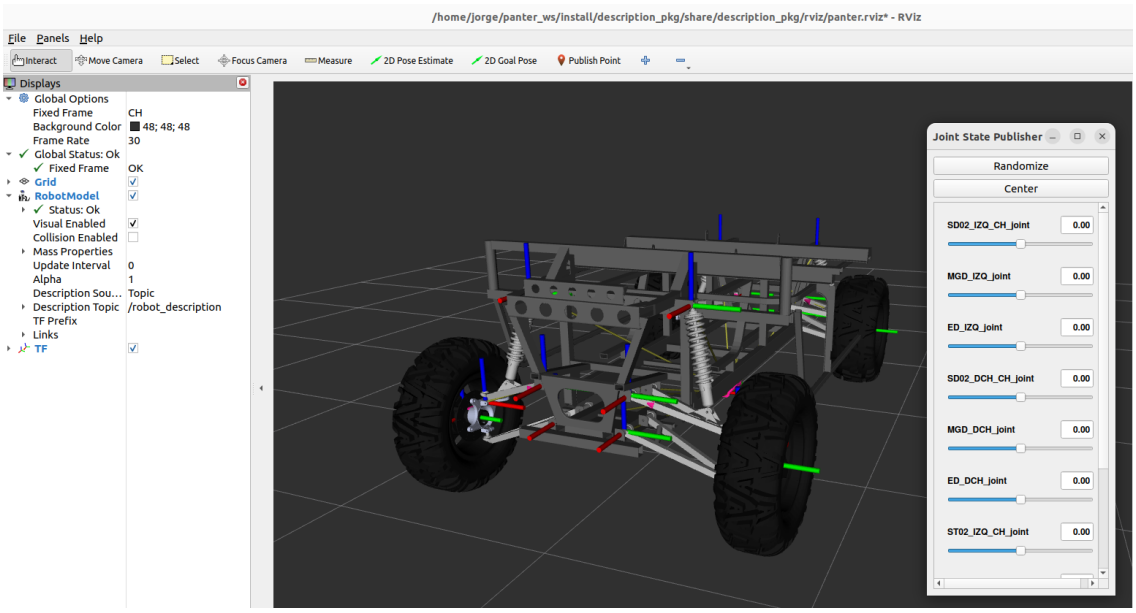

Figura 32. Panter en RViz

3.6. Programación en ROS

Para el control de Panter en ROS se desarrollan dos nodos teleoperados por teclado, que difieren en la forma de generar el movimiento.

3.6.1. Estructura general

En C++, y especialmente en un proyecto ROS 2, es buena práctica separar la interfaz (cabecera) de la implementación (cuerpo):

Cabecera

Se encargan de definir la interfaz de nuestras clases y funciones, sin incluir su lógica interna. En ellos se encuentra:

- Definición de clases: atributos (públicos y privados) y sus métodos.
- Declaración de constantes, tipos, clases/funciones, etc.
- Inclusiones de otros headers necesarios para referenciar tipos (por ejemplo, incluir <rclcpp/rclcpp.hpp> para nodos de ROS 2).

Cualquier otro módulo que necesite crear o manipular un objeto declarado en esa cabecera, sólo necesita incluir el .hpp para conocer su API pública.

Cuerpo

Este fichero implementa íntegramente el nodo de teleoperación de Panter. Su funcionamiento se compone de bloques bien diferenciados:

Constructor

Al instanciar el nodo, se crean los publicadores para actuar sobre el movimiento y las suscripciones tipo *geometry_msgs::msg::Wrench* a los sensores de fuerza/par de cada rueda.

- Sensor rueda delantera izquierda (ED_IZQ).
- Sensor rueda delantera derecha (ED_DCH).
- Sensor rueda trasera izquierda (ET_IZQ).
- Sensor rueda trasera derecha (ET_DCH).

De este modo, cada lectura entrante se almacena en variables internas. Finalmente, se programa un temporizador para informar en consola de los valores actuales de fuerza y par en todas las ruedas, medidos por el sensor.

Callbacks de sensores

Cada vez que llega un mensaje del sensor, el callback correspondiente copia sus componentes de fuerza y par en la estructura de datos asociada. Esto garantiza que, al actuar sobre Panter, el nodo pueda consultar las últimas mediciones y usarlas para ejercer un control real.

Bucle de teleoperación por teclado

El nodo arranca un hilo que está dedicado exclusivamente a leer pulsaciones desde la terminal en modo “raw”, lo que evita tener que pulsar “Enter” tras cada tecla. Dentro de este bucle, cada carácter presionado se interpreta según un conjunto predefinido de controles: avanzar, retroceder, girar en ambas direcciones y detenerse, recogidos en la Tabla 1.

Para cada acción, el bucle prepara los datos correspondientes de par o velocidad según el tipo de control, y los envía inmediatamente a los publicadores correspondientes. Pulsar

la barra espaciadora restaura la terminal a su modo original sin interrumpir el resto de la aplicación.

Tecla	Acción
`w`	Avanzar recto
`d`	Avanzar derecha
`a`	Avanzar izquierda
`s`	Retroceder recto
`c`	Retroceder derecha
`z`	Retroceder izquierda
`x`	Detener el robot
`+`	Aumentar par
`-`	Disminuir par
`1`	Aumentar ángulo giro
`0`	Disminuir ángulo giro

Tabla 1. Comandos del teclado para controlar el robot

Función principal 'main'

La función principal arranca ROS 2 y crea la instancia del nodo. A continuación, lanza en un hilo aparte el lector de teclado, para que pueda ejecutarse simultáneamente con la gestión de mensajes y el temporizador que gestiona las notificaciones del sensor. El hilo principal se bloquea escuchando a las funciones de callbacks hasta que solicita el fin de la teleoperación. Tras ello, espera a que el hilo de teclado finalice, apaga ROS 2 y el programa devuelve que todo es correcto.

3.6.2. Control de velocidad

Se define una clase *ControlVelocidad* con un nodo de ROS llamado *controlVelocidad*.

En el constructor, es donde se crean los publicadores, los suscriptores y cualquier otra inicialización que se defina: en particular, se crea un publicador de tipo *geometry_msgs::msg::Twist* en el topic */cmd_vel*, que permite publicar las consignas de velocidad.

En la función *vel_drive_panter()*: según el carácter leído, se construye un mensaje con las

consignas de velocidad necesarias.

El controlador establece una velocidad inicial de 0,1 m/s; durante la teleoperación por teclado, el usuario puede aumentarla de forma progresiva.

3.6.3. Control por esfuerzo

Paquete ros2_control

Ros2_control es un framework modular de ROS 2 que unifica el acceso al hardware y la ejecución de controladores, actuando como puente entre los algoritmos de control y los drivers físicos. En este proyecto se utiliza *ros2_control* para gestionar los actuadores, explicados en el Apartado 3.8.2, lo que posibilita:

- Gestionar de forma unificada tanto los controladores de par (esfuerzo) como los de velocidad para sus ruedas y manguetas, sin cambiar código ni la estructura de nodos.
- Definir en el URDF las transmisiones (*transmission*) y actuadores (*hardware_interface*) de cada joint (por ejemplo ED_IZQ_joint, MGD_IZQ_joint) para que el *controller_manager* reconozca automáticamente cada 'hardware interface'.
- Carga los plugins de control (Position, Velocity, Effort) en tiempo de ejecución.
- Publicar y suscribirse por ROS 2 a los topics.

Definición del nodo

Se define una clase *KeyboardcontrolTorque* con un nodo de ROS llamado *keyboardcontrol-Torque*.

En el constructor se crean los publicadores de */effort_controller/commands* y */steering_controller/commands* para publicar los comandos de par a las cuatro ruedas y la posición de la dirección.

En la función *keyboard_loop()*: se crean los mensajes de par y posición de la dirección según la tecla pulsada, y se publican inmediatamente en sus respectivos topics */effort_controller/commands* y */steering_controller/commands*.

El controlador aplica inicialmente un par de 50 Nm en cada rueda; a través de la teleoperación por teclado, el usuario puede incrementarlo de forma gradual hasta alcanzar el valor nominal máximo de 480 Nm (el motor ejerce 32 Nm nominales y la reductora es de 15:1).

3.6.4. Configuración de CMake

Para garantizar una configuración de compilación sólida y escalable, cada paquete de *panter_ws* sigue la siguiente estructura en su *CMakeLists.txt*:

1. **Declaración inicial**: se define la versión mínima de CMake y se nombra el paquete, indicando que se programará en C++.
2. **Localización de dependencias**: con *find_package()*, se cargan las librerías esenciales de ROS 2, de modo que los directorios y las bibliotecas estén disponibles durante la compilación.
3. **Definición de nodos**: cada nodo de control se compila como un ejecutable independiente y se vincula con sus dependencias.
4. **Instalación y exportación**: determinan qué elementos se instalarán en el espacio de destino y generan la información para que otros paquetes los localicen e importen.

3.6.5. Descripción del paquete

El archivo *package.xml* define la configuración y las dependencias de cada paquete en ROS:

- **Metadatos de los paquetes**: incluye nombre, versión, descripción, correo del responsable y licencia.
- **Herramienta de compilación**: especifica *ament_cmake* como herramienta de compilación.
- **Dependencias de ejecución**: enumera los paquetes necesarios en tiempo de ejecución (por ejemplo, *rclcpp* y *geometry_msgs*).

3.7. Lanzadores

Los ficheros de lanzamiento en ROS 2 son scripts en Python que, mediante la librería *launch*, arrancan de forma coordinada varios nodos, procesos y configuraciones. En este proyecto se definen tres lanzadores principales:

3.7.1. Lanzador simulador

El archivo *description_pkg/sim_urdf.launch.py* inicia la simulación de Gazebo con el modelo URDF de Panter, publica sus transformadas y habilita el puente con ROS 2. En el caso del control por velocidad, el lanzador es *simVel.launch.py*.

Estructura

1. Declaración de archivos, como el *urdf* del modelado, y el archivo *sdf* que pertenece al entorno de simulación (world): *urdf_file* y *world_file*.

2. Nodo *robot_state_publisher*: genera el parámetro */robot_description*, donde ROS almacena la descripción completa del robot en formato URDF.

3. Nodo *launch_gazebo*: ejecuta el simulador Gazebo Fortress, incluyendo el *world* desarrollado.

4. Nodo *spawn_panter*: se encarga de que el modelo de Panter aparezca en el simulador en unas coordenadas establecidas.

3.7.2. Lanzador control ROS

El archivo *control_pkg/ros.launch.py* inicia el nodo de control teleoperado por par y carga sus parámetros de funcionamiento. En el caso del control por velocidad, el lanzador es *ros_vel.launch.py*.

Estructura

1. Declaración de los archivos de configuración de los controladores y del archivo urdf: *control_pkg*, *controller_config* y *urdf_file*, respectivamente.

2. Nodo *control_node*, que se encarga de iniciar los controladores de *ros2_control*.

3. Nodo *joint_state_publisher*: lee la descripción de las articulaciones en el URDF y publica de forma periódica mensajes *sensor_msgs/JointState* con la posición de cada articulación, aportando los datos necesarios para que *robot_state_publisher* genere los frames TF correspondientes y para que RViz2 represente en tiempo real el movimiento articulado del robot según el estado de sus articulaciones [10].

4. Nodo *effort_spawner*: arranca el controlador de par de las articulaciones para gestionar el giro de las ruedas.

5. Nodo *steering_spawner*: pone en marcha el controlador de posición responsable de orientar las ruedas delanteras según la cinemática de Ackermann.

6. Nodo *ros_control*: se encarga de ejecutar el control de Panter a través de teclado, desarrollado en ROS2.

3.7.3. Lanzador puesta en marcha

El archivo *bringup_pkg/proyect.launch.py* ofrece un punto único de entrada que incluya los launchers anteriores, simplificando el lanzamiento completo. En el caso del control por velocidad, el lanzador completo es *proyect_vel.launch.py*.

Estructura

1. Nodo *gazebo_launch*: ejecuta el lanzador *sim_urdf.launch.py*.

2. Nodo *control_launch*: ejecuta el lanzador *ros.launch.py*.

3. Nodo *bridge_node*: inicia el fichero con el puente de comunicación entre ROS2 y Gazebo, *ros_gz_bridge.yaml*.

4. Nodo de lanzamiento de *RViz2*, programa de ROS para la visualización del robot.

3.8. Ficheros de configuración

3.8.1. Puente de comunicación ROS-Gazebo

En un entorno de simulación con ROS 2 e Ignition Gazebo, el archivo *ros_gz_bridge.yaml* actúa como un puente de configuración, que le indica al nodo qué canales de comunicación debe enlazar y cómo debe traducir los mensajes entre ROS y Gazebo. La Figura 33 muestra el flujo de comunicación.

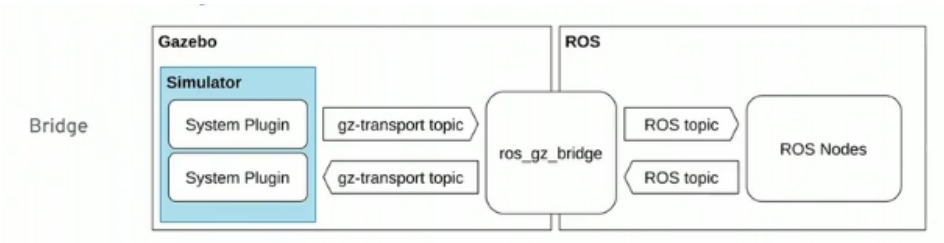

Figura 33. Flujo ros_gz_bridge

Para cada una de estas líneas de configuración se indica:

- El nombre del topic en ROS y su nombre equivalente en Gazebo.
- El tipo de mensaje que usa ROS, y el tipo paralelo en Ignition (por ejemplo, geometry_msgs/msg/Twist ↔ ignition.msgs.Twist [11]).
- La dirección del flujo de datos, representada en la Figura 34:
  - ROS→Gazebo para comandos y actuadores.
  - Gazebo→ROS para sensores y estados de simulación.
  - Bidireccional, cuando queremos mantener ambos ROS y Gazebo sincronizados en los dos sentidos.

```
- ros_topic_name: "/cmd_vel"
  gz_topic_name: "/cmd_vel"
  ros_type_name: "geometry_msgs/msg/Twist"
  gz_type_name: "ignition.msgs.Twist"
  direction: ROS_TO_GZ
```

Figura 34. Estructura de ros_gz_bridge.yaml [12]

3.8.2. Configuración controladores

El archivo *control_config.yaml* contiene la configuración de los controladores en ROS 2 control: define qué controladores cargar, a qué ritmo se actualizan y sobre qué articulaciones actúan.

En el archivo *control_config.yaml* se declaran los siguientes controladores:

- **effort_controller**:
  - Tipo: *effort_controllers/JointGroupEffortController*.
  - Función: regula el par aplicado a las articulaciones encargadas de impulsar las cuatro ruedas (ED_IZQ_joint, ED_DCH_joint, ET_IZQ_joint y ET_DCH_joint), garantizando que se ejerza el par deseado en el movimiento del vehículo.
- **steering_controller**:
  - Tipo: *position_controllers/JointGroupPositionController*.
  - Función: controla la posición angular de las articulaciones de las manguetas delanteras (MGD_IZQ_joint y MGD_DCH_joint), definiendo así la orientación de las ruedas y, por tanto, la dirección del vehículo.
- **joint_state_broadcaster**:
  - Tipo: *joint_state_broadcaster/JointStateBroadcaster*.
  - Función: publica periódicamente el estado actual (posición, velocidad y esfuerzo) de todas las articulaciones del robot.

3.9. Físicas

3.9.1. Motor de físicas Gazebo

En Gazebo, la simulación de la dinámica del robot (colisiones, fricción, gravedad y restricciones) se lleva a cabo por un motor de físicas. Aunque la interfaz de usuario y los formatos

de mundo (SDF) son comunes, internamente Gazebo puede emplear diferentes bibliotecas de simulación.

El motor de físicas utilizado en este proyecto es 'ODE', que ofrece buena relación entre precisión, estabilidad y velocidad de simulación.

3.9.2. Colisiones

La gestión de colisiones en Gazebo se basa en distinguir dos tipos de geometrías para cada enlace del robot: la visual (lo que se ve en pantalla) y la collision (la forma simplificada que usa el motor de físicas para detectar contactos). Esta separación permite optimizar el rendimiento, ya que las mallas de colisión pueden ser prismas, cilindros o cajas, que son mucho más ligeras computacionalmente que los modelos visuales detallados.

Cuando dos geometrías de colisión se solapan, el motor de físicas genera un contacto, la superficie de contacto, para evitar que los objetos penetren uno en otro y para modelar el deslizamiento o bloqueo.

Para definir las colisiones se utilizan figuras geométricas básicas: las ruedas se describen con cilindros, mientras que el chasis con un prisma rectangular. La visualización se representa en la Figura 35.

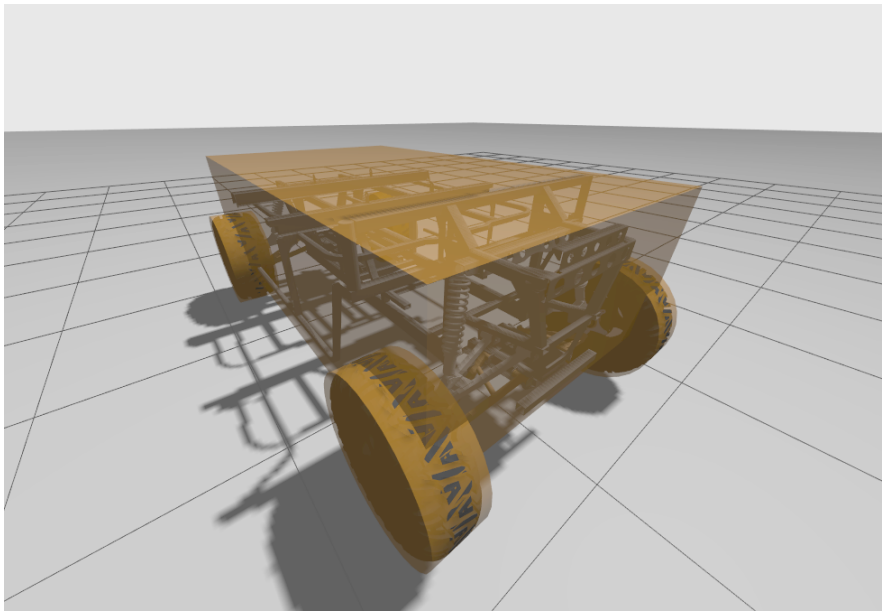

Figura 35. Representación colisiones de Panter en Gazebo

3.9.3. Masas

Conocer la masa de cada elemento del vehículo permite realizar cálculos precisos para las físicas del robot en el simulador.

Para obtener dichas masas, es necesario desmontar del vehículo real (véase la Figura 36) para realizar un pesaje de las piezas de forma aislada, como muestra la Figura 37.

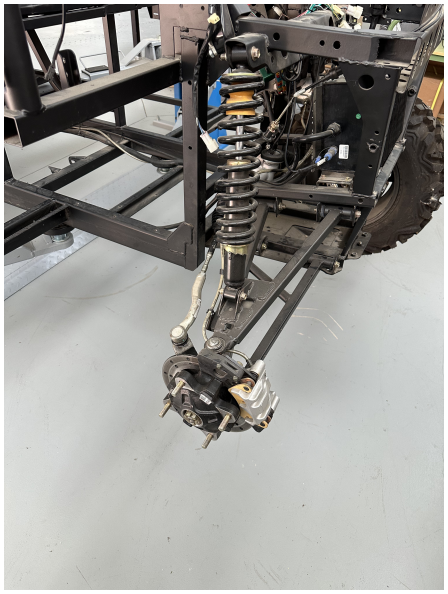

Figura 36. Desmontaje para pesaje

Figura 37. Pesaje rueda

En la Tabla 2 se recogen las masas medidas de todos los componentes principales, tanto del eje delantero como del trasero.

Elemento	**Masa (kg)**
Ruedas eje delantero	
Amortiguador Delantero (SD01)	3.75
Trapecio Superior Delantero (SD02)	2.75
Trapecio Inferior Delantero (SD03)	2.55
Mangueta Delantera	2.95
Palier Delantero	0.90
Disco Frenos Delantero	3.90
Ruedas Delanteras (ED)	15.95
Ruedas eje trasero	
Amortiguador Trasero (ST01)	5.35
Trapecio Superior Trasero (ST02)	2.65
Trapecio Inferior Trasero (ST03)	2.70
Mangueta Trasera	2.20
Palier Trasero	4.95
Disco Frenos Trasero	4.35
Ruedas Traseras (ET)	19.95

Tabla 2. Masas de todos los componentes de Panter.

Para determinar la masa del chasis, se pesa el vehículo completo, desde cada una de las ruedas, y a su vez se excluyen todos los componentes montados sobre el chasis (baterías, controladoras, reductoras planetarias, rodamientos, cadenas y motores), cuyos resultados se recogen en la Tabla 3.

	Izquierda (kg)	**Derecha (kg)**
Delantero	100.55	80.15
Trasero	80.95	81.70
TOTAL		343.35

Tabla 3. Distribución de masas por rueda (en kg).

Del valor total recogido en la Tabla 3 es preciso descontar la masa correspondiente a las ruedas y a los conjuntos de suspensión.

$$M_{\text{SoloChasis}} = M_{\text{ChasisEjes}} - M_{\text{Ruedas}} - M_{\text{Suspensión}} \tag{2}$$

$$M_{\text{SoloChasis}} = 343.35 - 102.2 - 39.5 = 201.65kg \tag{3}$$

A la masa calculada en la Ecuación 3 se debe sumar el peso de los componentes instalados sobre el chasis, descritos en la Tabla 4.

Elemento	**Masa unitaria (kg)**	**Unidades**	**Masa total (kg)**
Baterías	45	4	180
Controladoras	4.65	4	18.6
Reductoras planetarias	35	4	140
Motores	20.15	4	80.6
Margen (rodamientos, cadenas, ...)	20	4	80
TOTAL			**499.2**

Tabla 4. Masas de los elementos asociados al chasis de Panter.

$$M_{\text{ChasisCompleto}} = M_{\text{SoloChasis}} + M_{\text{ElementosAdicionales}} \tag{4}$$

$$M_{\text{ChasisCompleto}} = 201.65 + 499.2 = 700.85kg \tag{5}$$

Masas en la descripción URDF

Las masas utilizadas en los elementos de la descripción se encuentran en la Tabla 5.

Elemento	**Masa (kg)**
Chasis (CH)	700.85
Ruedas eje delantero	
Amortiguador Delantero (SD01)	3.75
Trapecio Superior Delantero (SD02)	2.75
Trapecio Inferior Delantero (SD03)	2.55
Mangueta Delantera (MGD)	2.95
Ruedas Delanteras Completas (ED)	20.75
Ruedas eje trasero	
Amortiguador Trasero (ST01)	5.35
Trapecio Superior Trasero (ST02)	2.65
Trapecio Inferior Trasero (ST03)	2.70
Mangueta Trasera (MGT)	2.20
Ruedas Traseras Completas (ET)	25.2

Tabla 5. Masas para la descripción URDF de Panter.

En la descripción URDF, las ruedas (delanteras y traseras) se modelan como conjuntos completos que incluyen la llanta, el disco de freno y el palier. En particular, a la rueda trasera se le asigna la masa del palier delantero, porque el trasero está unido a otro componente. Son denominadas en la Tabla 5 como 'rueda completa'.

3.9.4. Inercias

La matriz de inercia representa la distribución de la masa alrededor de los ejes principales de un cuerpo rígido y mide su resistencia a los cambios en la velocidad de giro. Calcularla para cada componente del robot es importante para reproducir un comportamiento dinámico preciso.

Si estos valores se estiman, el modelo dinámico no representa con exactitud la respuesta ante fuerzas y momentos, lo que puede resultar en movimientos erráticos e inestabilidad. Para aligerar el proceso de cálculo y evitar la complejidad de tratar geometrías complejas, cada

pieza del robot (chasis, trapecios de suspensión, amortiguador, ruedas, etc.) se representa mediante formas geométricas elementales (cilindros y prismas). Gracias a ello, podemos aplicar directamente fórmulas analíticas para obtener de forma rápida y rigurosa los tres momentos de inercia principales (I_x, I_y, I_z) de cada elemento.

Matriz inercia

El uso del prisma rectangular y del cilindro como aproximaciones geométricas básicas facilita el cálculo de la matriz de inercia [13] para los distintos componentes de Panter. Gracias a esta simplificación, cualquier variación en las dimensiones de las piezas puede incorporarse de forma rápida y directa, permitiendo recalcular la matriz de inercia con mínimo esfuerzo.

Prisma rectangular:

$$\begin{aligned} I_x &= \tfrac{1}{12}\, m\,(b^2 + h^2), \\ I_y &= \tfrac{1}{12}\, m\,(l^2 + h^2), \\ I_z &= \tfrac{1}{12}\, m\,(b^2 + l^2). \end{aligned} \tag{6}$$

Para el chasis y los trapecios de suspensión se adopta la aproximación mediante prismas rectangulares.

Cilindro:

$$\begin{aligned} I_x &= \tfrac{1}{12} m\Big(3r^2 + h^2\Big), \\ I_y &= \tfrac{1}{2} m r^2, \\ I_z &= \tfrac{1}{12} m\Big(3r^2 + h^2\Big). \end{aligned} \tag{7}$$

Para las ruedas y para los amortiguadores se adopta la aproximación mediante cilindros.

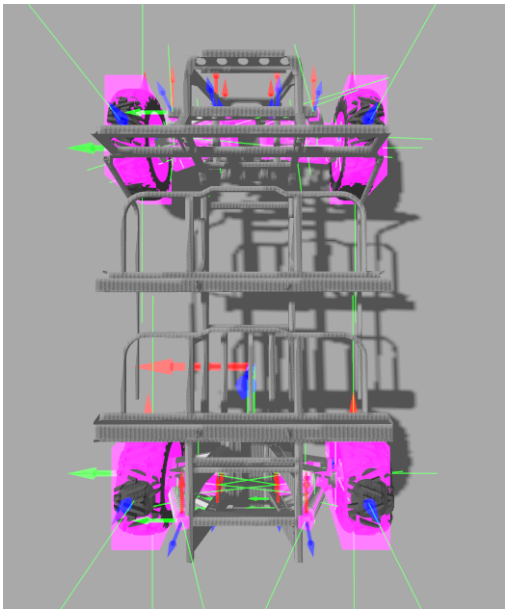

Figura 38. Representación inercias Panter

La Figura 38 muestra la representación de las inercias de Panter en Gazebo, exceptuando la inercia del chasis, que se ajusta a continuación.

Ajuste inercia Chasis

La Tabla 3 muestra que Panter no es totalmente simétrico respecto a su masa, entonces para determinar el centro de masas (CoM) del vehículo a partir de las lecturas de peso en cada rueda, se requieren dos elementos fundamentales:

1. **Distribución de cargas** (lecturas de peso en cada rueda):

$$\begin{aligned} \text{Delantero—Izquierda (DI)} &= 100.55\,\text{kg}, \\ \text{Delantero—Derecha (DD)} &= 80.15\,\text{kg}, \\ \text{Trasero—Izquierda (TI)} &= 80.95\,\text{kg}, \\ \text{Trasero—Derecha (TD)} &= 81.70\,\text{kg}, \\ \text{Total} &= 100.55 + 80.15 + 80.95 + 81.70 = 343.35\,\text{kg}. \end{aligned} \tag{8}$$

2. **Cotas geométricas del vehículo**:

   - $L = 2.20486m$: distancia entre el eje delantero y el eje trasero (“wheelbase”).

- $T = 1.36356m$: ancho entre ruedas izquierdas y derechas (“track width”).

Definición de la Convención de Ejes

El sistema de referencia es el mostrado en la Figura 39.

X>0; hacia la izquierda en planta (o a la derecha de frente).

Y>0; hacia atrás (hacia el eje trasero).

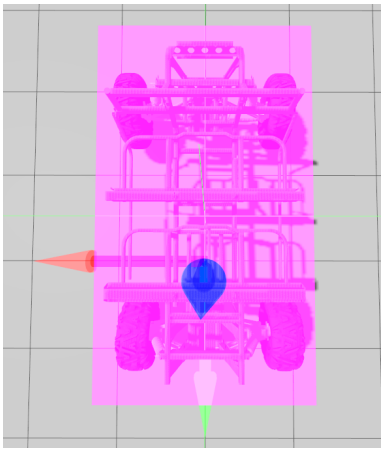

Figura 39. Sistema de referencia Panter

Cálculo del Centro de Masas

El centro de masas $\left(X_{\text{CoM}}, Y_{\text{CoM}}\right)$ se descompone en dos coordenadas:

X_{CoM} = desplazamiento lateral (izquierda/derecha).

Y_{CoM} = desplazamiento longitudinal (adelante/atrás).

Coordenada longitudinal Y_{CoM}

Para la coordenada longitudinal, se toma $Y = 0$ en el eje delantero y “hacia atrás” como $Y > 0$. Entonces, aplicando la regla de momentos:

$$Y_{\text{CoM}} = \frac{W_{\text{front}} \cdot 0 + W_{\text{rear}} \cdot L}{M_{\text{tot}}} = \frac{162.65 \times 2.20486}{343.35} \approx 1.0449 \text{ m}. \tag{9}$$

Es decir, el CoM se encuentra 1.0449 m detrás del eje delantero.

Coordenada lateral X_{CoM}

Definimos la línea central como $X = 0$. Las ruedas izquierdas están en $X = +\frac{T}{2}$, y las derechas en $X = -\frac{T}{2}$.

La fórmula del momento lateral es:

$$X_{\mathrm{CoM}} = \frac{W_{\mathrm{izq}}\left(+\frac{T}{2}\right) + W_{\mathrm{dch}}\left(-\frac{T}{2}\right)}{M_{\mathrm{tot}}} = \frac{181.50 \times \frac{1.36356}{2} - 161.85 \times \frac{1.36356}{2}}{343.35} \approx 0.0390\ \mathrm{m}\,. \quad (10)$$

Resultado final

Por tanto, en el sistema donde $X > 0$ apunta a la izquierda y $Y > 0$ apunta hacia atrás:

$$\left(X_{\mathrm{CoM}},\, Y_{\mathrm{CoM}}\right) \;=\; \left(+\,0.0390\ \mathrm{m},\; +\,1.0449\ \mathrm{m}\right).$$

- $X_{\mathrm{CoM}} = +0.0390$m: hacia la izquierda (en planta) de la línea central.
- $Y_{\mathrm{CoM}} = +1.0449$m: hacia atrás (desde el eje delantero).

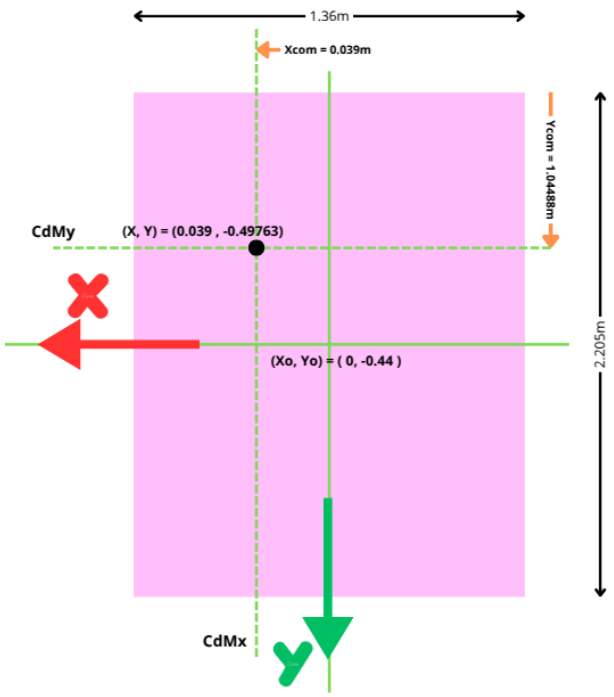

Figura 40. Esquema de ajuste inercia Chasis

En la Figura 40 se muestra el esquema de inercia con el que se ha ajustado el chasis, y finalmente, en la Figura 41 cómo es la inercia real de Panter en Gazebo.

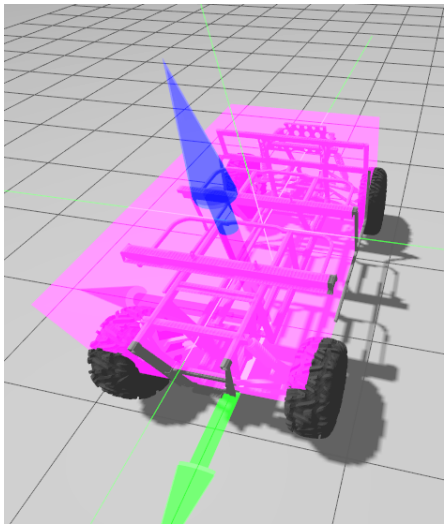

Figura 41. Inercia Chasis ajustada

3.9.5. Rozamiento

En el simulador, el rozamiento entre las ruedas y el terreno se modela a través de los parámetros de “surface” y “friction” que se declaran tanto en las geometrías de colisión de las ruedas como en el plano que representa el suelo. Dado que nuestro escenario es un terreno no estructurado (tierra, campo con cierta rugosidad), se introducen los coeficientes de fricción correspondientes, de modo que pueden transmitir el par de tracción sin deslizarse excesivamente.

En la descripción URDF, ’m’ y ’m2’ son los coeficientes de fricción estática y dinámica, respectivamente. Un valor de 1.5 significa que la máxima fuerza tangencial antes de que la rueda empiece a deslizarse es 1.5 veces la normal.

Superficie	**Valor máximo $\mu_{máx}$**	**Valor de deslizamiento μ**
Asfalto y hormigón secos	0.8–0.9	0.75
Asfalto mojado	0.5–0.7	0.45–0.6
Hormigón mojado	0.8	0.7
Grava	0.6	0.55
Tierra seca	0.68	0.65
Tierra húmeda	0.55	0.4–0.5
Nieve dura	0.2	0.15
Hielo	0.1	0.07

Tabla 6. Coeficientes de adherencia según superficie [14]

Los valores asignados son para un terreno de tierra seca son 'm' = 0.68 y 'm2' = 0.65, como indica la Tabla 6.

3.9.6. Par de dirección

El modelo de Panter no cuenta con un diseño 3D de la cremallera de dirección y cada rueda funciona físicamente de manera independiente, no está conectada por ningún mecanismo de dirección o eje común. Cuando una rueda pisa un terreno más elevado o más hundido que la otra, cada una adopta un ángulo distinto, lo que puede desestabilizar la trayectoria, como se ilustra en la Figura 42.

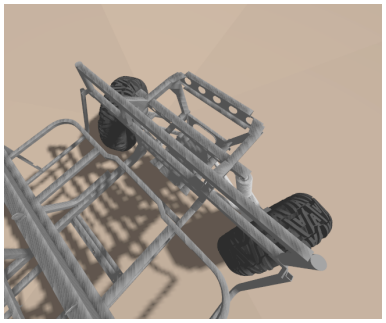

Figura 42. Dirección desviada

Para ajustar esto, hay que tener en cuenta el parámetro que limita el par máximo en las articulaciones [15] de las manguetas delanteras:

Esfuerzo (effort): limita el par máximo que puede aplicar el controlador de posición sobre la articulación, evitando que un par excesivo u obstáculos, desplacen la rueda más allá de su posición deseada. De esta forma, al recibir una perturbación muy grande del exterior, el controlador puede mantener su posición.

En la simulación, se determina experimentalmente que tiene un valor de 500 Nm. Con este ajuste, las ruedas delanteras responden de manera estable ante terrenos irregulares.

3.9.7. Suspensiones

En primer lugar, no es posible modelar la física en Gazebo del conjunto de suspensión tal como está definido actualmente. A continuación, se va a desarrollar la problemática e intentos realizados.

Para el modelado de las suspensiones debemos primero conocer cómo funciona el conjunto de suspensión de Panter, mostrado en la Figura 43.

Figura 43. Conjunto de suspensión

El sistema de suspensión está compuesto por dos trapecios de suspensión, que se encuentran unidos al chasis, y estos a la mangueta, que a su vez está unida a la rueda. Luego, el amortiguador se encuentra entre el chasis y el trapecio de suspensión superior. Este esquema está representado en la Figura 44.

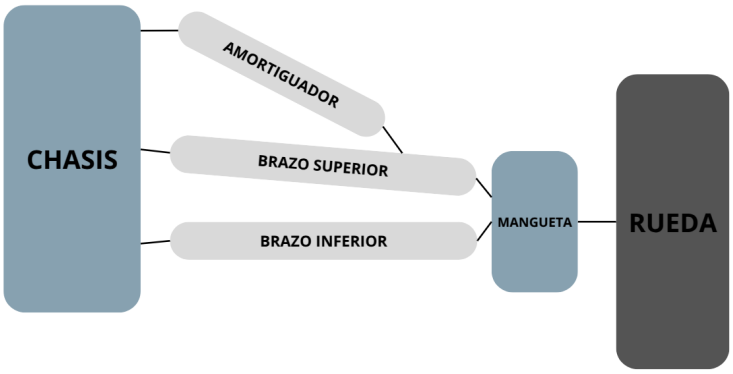

Figura 44. Esquema conjunto de suspensión

Para poder modelar la suspensión en el simulador gazebo, debemos introducir cómo se deben comportar las articulaciones que unen esos elementos en la descripción URDF. La unión de dos links se realiza con una articulación (joint).

Al declarar una articulación, debemos establecer qué link es el padre (parent), y qué link es el hijo (child) del movimiento, es decir, el que realizará el movimiento alrededor del padre.

Entonces, por la propia cinemática que tiene Gazebo, se establece una serie de normas:

- Un link puede ser parent de varios links child.
- Un link solo puede ser child de un solo parent.
- Un link puede ser a la vez parent y child.

A partir de estas normas, se trata de buscar la forma de resolver esta situación.

Planteamiento 1

El primer intento realizado para modelar el conjunto de suspensión es establecer joints que unan todos los elementos, como en la Figura 45, donde la 'P' representa a parent, y la 'C' representa a child. Las circunferencias rojas representan los child que están generando conflicto, debido a que tienen varios parents.

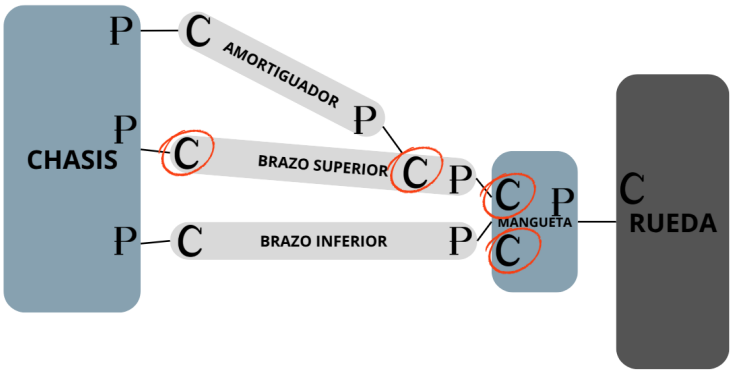

Figura 45. Planteamiento 1 suspensiones

El problema surge en que Gazebo no permite introducir bucles cerrados de cinemática (Closed Kinematic Loops), y en el diseño del conjunto de suspensión, aparecen dos bucles cerrados.

Planteamiento 2

El segundo intento realizado es establecer links falsos, con masa e inercia nula, situados entre los joints que entran en conflicto. Esta idea no resuelve el problema.

Planteamiento 3

Utilizar articulaciones fijas (fixed joints) en dos puntos de los bucles, para que rompa con el bucle de cinemática. Esta es la idea que propone el tutorial de gazebo, pero llevada a la práctica, la respuesta del sistema sigue siendo que hay links que tienen más de un parent.

El esquema está representado en la Figura 46.

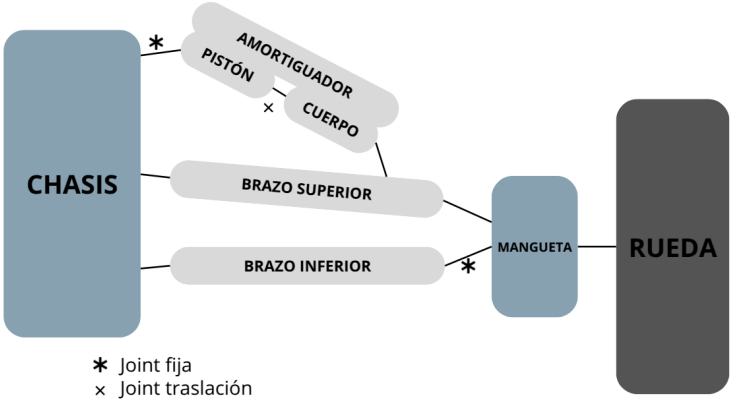

Figura 46. Planteamiento 3 suspensiones

Planteamiento 4

Otro planteamiento es el uso de mimic joints, cuya cualidad es hacer que una articulación realice el mismo movimiento que la otra articulación existente. Podríamos aplicarlo a los dos trapecios de suspensión, donde el inferior esté conectado al chasis y a la mangueta, y el superior no esté conectado a la mangueta e imite el movimiento del inferior. Pero surgen los siguientes problemas:

- La gravedad actúa sobre el vehículo al comenzar la simulación, por lo que el trapecio superior cae.
- Los trapecios de suspensión no son paralelos, por lo que su movimiento ante el trabajo de suspensión no será igual para ellos.
- Al solo unir la mangueta por un punto, esta no se mantendrá en su orientación correcta al encontrar cualquier perturbación, sino que perderá su verticalidad y por tanto también la de la rueda.

Conclusión conjunto de suspensión

No es posible describir el conjunto de suspensión de Panter, ya que contiene bucles cerrados de cinemática, que no son soportados por el simulador Gazebo.

Posible solución: simplificar el conjunto de suspensión, dejando únicamente un solo trapecio de suspensión y eliminando el amortiguador, y a su vez mantener las características de amortiguación que ofrecen estos elementos. No se ha implementado esta solución para conservar el conjunto de suspensión original.

3.9.8. Reductora

La reductora planetaria (o caja de engranajes planetarios) es un elemento mecánico que permite transformar la velocidad y el par de giro de un motor en valores más adecuados para impulsar las ruedas.

Panter utiliza el reductor planetario APEX AE205 con relación de transmisión 15:1 y juego angular menor a 12 arc-min ($J < 12$).

Es necesario incluir el valor de la reductora para que la simulación sea fiel a la realidad; para ello, en la descripción URDF del control por par, se introduce en el bloque *transmission*. Al multiplicar el par por 15, la velocidad de salida se divide entre 15.

3.10. Plugins

Plugin Direccionamiento de Ackermann

El plugin AckermannSteering de Ignition Gazebo [16] implementa un controlador de dirección tipo Ackermann, pensado para vehículos que cuentan con ruedas delanteras que giran (steering) y ruedas motrices traseras, como es el caso del robot Panter. Es necesario que el modelo tenga al menos un par de ruedas izquierdas y derechas, así como un par de articulaciones (joints) que controlen la dirección de las ruedas delanteras.

Está compuesto por elementos como:

- Definición de librería e identificación del plugin en el sistema.
- Articulaciones que controlan la dirección de las ruedas delanteras izquierda y derecha. El plugin usará estas referencias para aplicar el ángulo apropiado al girar.
- Articulaciones de las ruedas motrices. El plugin las usa para aplicar el par necesario que genere la velocidad deseada.
- Distancia (en metros) entre las ruedas izquierda y derecha de cada eje motriz *wheel_separation*. En este modelo equivale a 1.36 m. Este valor es esencial para calcular correctamente los ángulos Ackermann de cada rueda al girar.
- Radio (en metros) de las ruedas motrices *wheel_radius*. El valor del radio de las ruedas es de 0.343 m. El plugin convierte la velocidad lineal deseada (v en m/s) en la velocidad angular de las juntas motrices ($\omega = v/r$).
- Panter alcanza una velocidad máxima de 55 km/h (15.28 m/s) [9]. Por ello, el plugin define un rango de velocidades lineales de 0 a 15.28 m/s, asegurando que todas las consignas de velocidad se mantengan dentro de ese límite.

- Límites de aceleración que el sistema puede aplicar. Estos valores ayudan a suavizar la transición de velocidad y evitan aceleraciones bruscas que podrían desestabilizar la simulación. La aceleración máxima tiene un valor de 7.26 m/s^2.

  Con los datos de par nominal del motor ($T_m = 32$ Nm), reductora ($G = 15$) y radio de rueda ($r = 0.343$ m), la aceleración máxima teórica se obtiene:

$$T_{\text{rueda}} = T_m \times G = 32\,\text{Nm} \times 15 = 480\,\text{Nm} \tag{11}$$

$$F_{\text{rueda}} = \frac{T_{\text{rueda}}}{r} = \frac{480\,\text{Nm}}{0.343\,\text{m}} \approx 1400\,\text{N} \tag{12}$$

$$F_{\text{total}} = N_{\text{ruedas}} \times F_{\text{rueda}} = 4 \times 1400\,\text{N} = 5600\,\text{N} \tag{13}$$

$$a_{\max} = \frac{F_{\text{total}}}{m} = \frac{5600\,\text{N}}{771.7\,\text{kg}} = 7.26\,\text{m/s}^2 \tag{14}$$

- Topic de ROS al que el plugin se suscribe para recibir comandos de velocidad (*geometry_msgs/Twist*). A partir de cada mensaje en */cmd_vel*, el sistema calcula los movimientos de las articulaciones de dirección y las velocidades angulares de las ruedas motrices.

Plugin sensor force-torque

El plugin GazeboRosFTSensor [17] es el plugin que simula un sensor de fuerza y par (force/torque) en Gazebo.

Este plugin se acopla a una articulación (joint) de un modelo en Gazebo y calcula, en cada ciclo de simulación, la fuerza y el par que actúan en dicha articulación.

Emite estos valores a través de un mensaje *geometry_msgs/WrenchStamped* en un topic de ROS para cada rueda.

Asignación de topics force-torque

A continuación se nombran los topics de ROS donde publican sus datos de esfuerzo y par:

- Rueda delantera izquierda:

  /model/panter/ED_IZQ_joint/sensor/force_torque_sensor/force_torque

- Rueda delantera derecha:

  /model/panter/ED_DCH_joint/sensor/force_torque_sensor/force_torque

- Rueda trasera izquierda:

  /model/panter/ET_IZQ_joint/sensor/force_torque_sensor/force_torque

- Rueda trasera derecha:

  /model/panter/ET_DCH_joint/sensor/force_torque_sensor/force_torque

El sensor virtual publica el esfuerzo (fuerza y par) en el sistema de coordenadas del 'child' de la articulación y realiza la medición desde el enlace hijo hacia el padre 'parent' de la articulación.

3.11. Escenario Gazebo

Para conseguir una simulación realista, es necesario recrear un entorno que refleje las condiciones exteriores reales en las que operará Panter. Dado que este robot está diseñado para moverse sobre superficies irregulares, se modelan terrenos no estructurados utilizando Blender.

El proceso para diseñar cada escenario en Blender comienza con un plano simple, que se escala por 100, se subdivide en secciones de 12.5 metros cuadrados y se ajusta manualmente cada zona para generar una topografía específica.

Cada escenario completo mide 175 metros cuadrados, por lo que Panter tendrá suficiente terreno para realizar una simulación.

Escenario 1

En el primer escenario, el objetivo es conseguir un terreno no estructurado, simulando una zona sin carriles. La malla resultante se observa en la Figura 47.

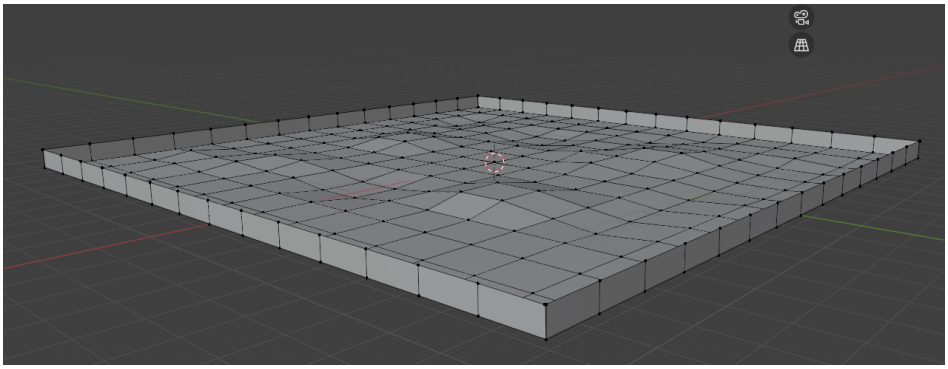

Figura 47. Malla Escenario 1 en Blender

Para incluir el escenario en el proyecto, se utiliza el paquete *description_pkg* y la carpeta *models/Tierra* con la siguiente organización:

- **model.sdf**: contiene la descripción del escenario en formato SDF, exportada desde Blender y adaptada al mundo de Gazebo.
- **model.config**: define el nombre del modelo *Tierra*, que es el identificador que usará Gazebo al instanciar el escenario desde el archivo world.
- **/meshes**: Guarda las mallas en formato Collada (.dae) y .stl de los terrenos irregulares que generamos en Blender (ver Figura 47).
- **/materials**: incluye las texturas y archivos de material necesarios para dar realismo al terreno dentro del simulador.

A continuación, en el archivo *tierra_world.sdf* se incorpora el modelo del terreno para que quede integrado en la simulación.

Escenario 2

El segundo escenario se diseña para establecer 8 pendientes diferentes con las características de la Tabla 7.

Cálculo de las pendientes: cada subdivisión del plano mide 12.5 metros cuadrados, y al saber la pendiente que se quiere conseguir, se calcula exactamente cuánto debe elevarse el terreno, con la Ecuación 15.

$$\text{pendiente } (\%) = \left(\frac{\text{altura}}{\text{base}}\right) \times 100 \tag{15}$$

% Pendiente	Grados	Elevación (m)	Longitud (m)
25	14.04	3.13	12.89
30	16.70	3.75	13.05
35	19.29	4.38	13.24
40	21.80	5.00	13.46
45	24.23	5.63	13.71
50	26.57	6.25	13.98
55	28.81	6.88	14.27
60	30.96	7.50	14.58

Tabla 7. Relación pendiente Escenario 2

En la Figura 48, se observa la malla diseñada.

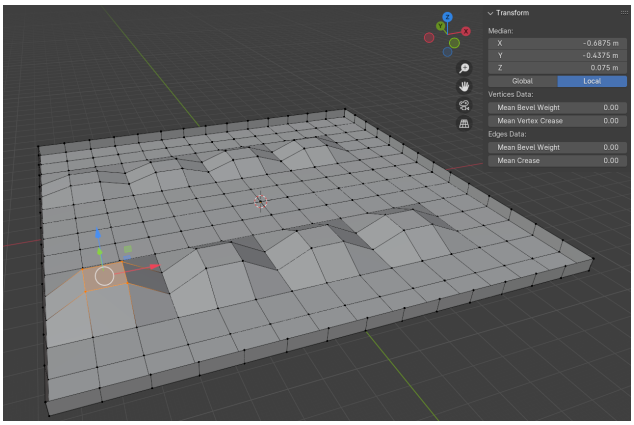

Figura 48. Malla Escenario 2

Escenario 3

En el caso de que Panter tenga la capacidad de superar pendientes del 60 %, este escenario contiene tres pendientes de 65 %, 70 % y 75 % (ver Tabla 8).

% Pendiente	Grados	Elevación (m)	Longitud (m)
65	33.02	8.1250	14.9086
70	34.99	8.7500	15.2582
75	36.87	9.3750	15.6250

Tabla 8. Relación pendiente Escenario 3

En la Figura 49 se observa la malla diseñada.

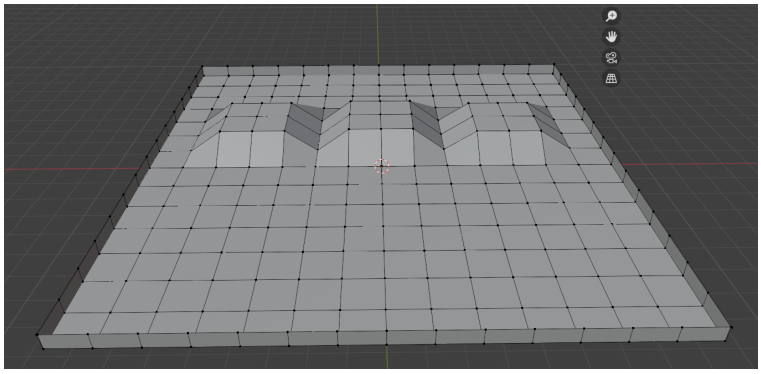

Figura 49. Malla Escenario 3

Capítulo 4

Resultados

Se evalúa el rendimiento de Panter en los escenarios de simulación personalizados, realizando el control por par en cada rueda y el control por velocidad.

Para cada control se asignan las siguientes consignas:

- Cada motor de Panter entrega nominalmente 32 Nm de par, que la reductora 15:1 amplifica a 480 Nm en cada rueda. El sistema de control planteado arranca con un par de 50 Nm por rueda y, durante la teleoperación por teclado, el usuario puede incrementarlo de forma progresiva hasta el máximo nominal de 480 Nm. El ángulo de giro de las ruedas delanteras también se ajusta en tiempo real desde el teclado, comenzando en 11.04 grados (0.1927 radianes), proporcionando un control directo sobre la orientación del vehículo.

- En el modo de control por velocidad, la velocidad inicial se fija en 1 m/s (3.6 km/h) y el usuario puede aumentarla progresivamente desde el teclado hasta la velocidad máxima de 15.28 m/s (55 km/h). La velocidad de giro también se modifica por teclado comenzando en 6 rad/s.

El sensor permite monitorizar en tiempo real el par ejercido en cada rueda, tanto por esfuerzos internos como externos.

4.1. Simulación escenario 1

En el escenario 1, Panter se desplaza sobre un terreno no estructurado con desniveles (véase la Figura 50).

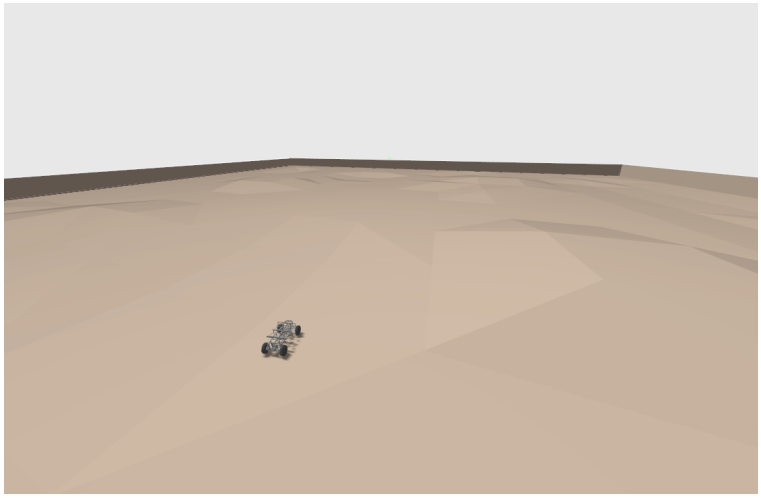

Figura 50. Panter en escenario 1

Este entorno es especialmente irregular, y pone a prueba sus capacidades de tracción y dirección. Además, sirve para verificar el ajuste de la dirección descrito en el Apartado 3.9.6, donde las pendientes asimétricas provocaban que las ruedas delanteras adoptasen ángulos distintos y desviaban la trayectoria.

Control Par

Panter sigue la trayectoria de manera exitosa, salvando cada desnivel sin desviaciones. Con cualquier par aplicado hasta los 480 Nm nominales, el vehículo mantiene la estabilidad y avanza sin problemas.

Control Velocidad

Al recibir comandos de velocidad lineal y angular, Panter sigue con precisión la trayectoria planificada y supera sin problemas los desniveles del escenario. Panter garantiza un comportamiento estable en todo el rango de velocidad hasta los 55 km/h.

4.2. Simulación escenario 2

El escenario 2 consta de ocho rampas cuyas pendientes aumentan en intervalos del 5 %, partiendo del 25 % hasta el 60 % (véase Figura 51). Este diseño evalúa la capacidad de Panter para ascender diferentes pendientes y determina los umbrales máximos de pendiente que puede superar con éxito.

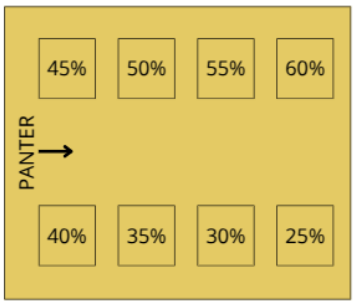

Figura 51. Esquema Escenario 2

Control Par

Para el análisis experimental, Panter arranca siempre desde reposo al pie de cada pendiente. Tras estas pruebas, en la Tabla 10 se determina el par necesario en cada rueda para superar las inclinaciones:

% Pendiente	Par por rueda	Resultado
25	200	Asciende
30	210	Asciende
35	230	Asciende
40	310	Asciende
45	380	Asciende
50	420	Asciende
55	470	Asciende
60	-	No Asciende

Tabla 9. Resultados Par Escenario 2

En la Figura 52 se observa a Panter subiendo una de las pendientes diseñadas.

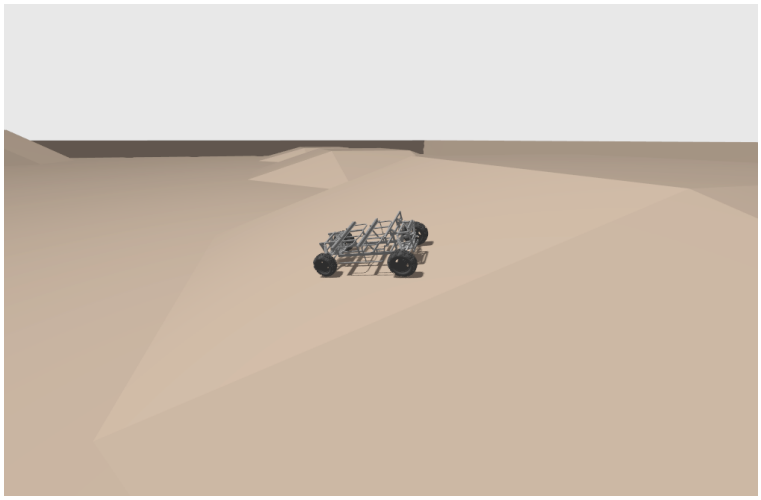

Figura 52. Panter en escenario 2

En consecuencia, Panter es capaz de ascender pendientes de hasta un 55 % sin necesidad de aprovechar ninguna inercia adicional.

Control Velocidad

Para el control de velocidad, he replicado la prueba empleada en el control por par. Se observa que el rendimiento no depende de la velocidad de avance de Panter, sino del par aplicado. El robot supera sin dificultad pendientes del 25 % al 55 %, exactamente igual que en la prueba anterior, independientemente de la velocidad, por lo que asciende las pendientes que podía ascender en el caso de control por par.

4.3. Simulación escenario 3

El escenario 3 se compone de tres pendientes de valores 65 %, 70 % y 75 %, como muestra la Figura 53.

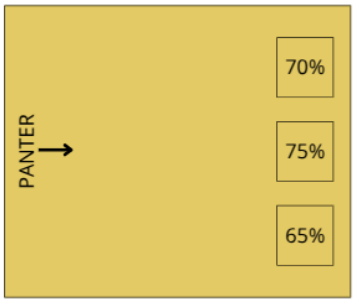

Figura 53. Esquema Escenario 3

Panter no puede superar desde reposo pendientes superiores al 55 %, por lo que no tiene sentido evaluar inclinaciones mayores: ningún par por debajo del nominal le permitiría superar dichas pendientes sin ayuda de inercia. Por ello, se verifica si, aprovechando una inercia inicial (alcanzando su velocidad máxima de 55 km/h) y el par nominal máximo, Panter es capaz de superar mayores pendientes.

La Tabla 10 muestra si Panter puede superar esas pendientes.

% Pendiente	**Resultado**
65	No Asciende
70	No Asciende
75	No Asciende

Tabla 10. Resultados Par Escenario 3

Con una inercia inicial, Panter no puede superar ninguna de las tres pendientes diseñadas del 65 %, 70 % y 75 %.

En la Figura 54 se observa a Panter en la pendiente del 75 % del escenario 3.

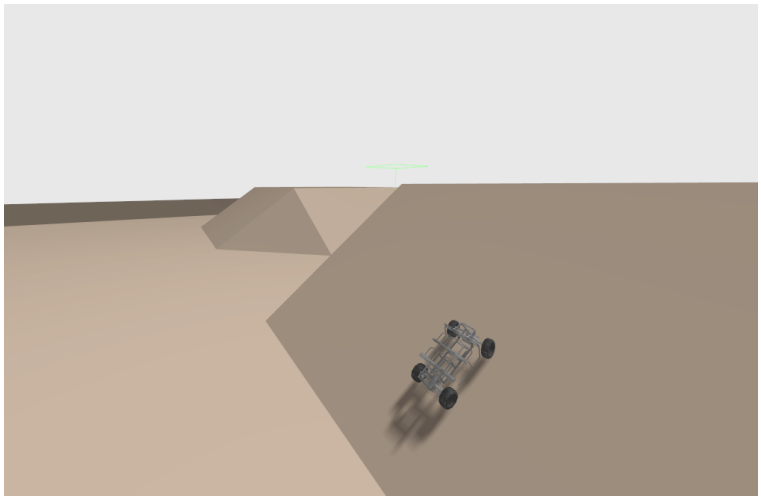

Figura 54. Panter en escenario 3

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Durante el proyecto se han conseguido una serie de hitos:

- Se ha configurado por completo el entorno de trabajo en Linux.
- Se ha implementado una jerarquía de proyecto que permite la comunicación y lanzamiento de varias aplicaciones y múltiples archivos que permiten que el simulador funcione correctamente.
- Se ha adquirido experiencia en herramientas fundamentales como:
  - SolidWorks para realizar modificaciones en diseños 3D.
  - Gazebo Fortress para la simulación de robots.
  - Integrar ROS 2 para implementar la lógica de control.
  - RViz2 para representar el vehículo y sus articulaciones durante el desarrollo de su modelo de simulación.
  - Blender para el diseño de escenarios de simulación en Gazebo.
  - Durante el proceso se ha programado por primera vez en Python y XML, y se ha profundizado en C++. También se ha empleado YAML para la parametrización y configuración de modelos.

- El vehículo ha sido acortado en el modelo 3D, por su zona central.
- Se ha desarrollado un simulador en Gazebo que funciona correctamente.
- Se han establecido dos formas de control del simulador mediante dos magnitudes: par en las ruedas y de posición en la dirección mediante controladores de ROS, y la otra mediante control por velocidad del vehículo, mediante un plugin de Gazebo.
- Se ha implementado un sensor que permite medir qué ocurre en términos de fuerza y par en las articulaciones de las ruedas.
- Se han diseñado tres escenarios en Blender, que se han exportado en formato COLLADA para introducirlos en los archivos SDF que definen el mundo de Gazebo.
- El simulador está preparado para utilizar cualquier escenario y que Panter sea ejecutado en él siguiendo una representación física fiel a la realidad, exceptuando las físicas de amortiguación. Por lo tanto, se cumple con éxito el objetivo principal de desarrollar un simulador en Gazebo.
- Se ha puesto en práctica el control de versiones (una competencia imprescindible en proyectos reales y trabajo en equipo), lo que garantiza la trazabilidad y la calidad del desarrollo.
- Para la redacción de esta memoria he utilizado LaTeX Overleaf.

Gracias a todo ello, se ha diseñado e implementado un simulador robótico funcional. Por lo que se tiene un simulador en el que se pueden obtener especificaciones que necesita Panter para realizar distintas tareas y obtener resultados más precisos de cálculos teóricos del vehículo. Los datos obtenidos de la simulación permiten desarrollar a Panter.

5.2. Líneas futuras de trabajo

Este proyecto abre muchas posibilidades de desarrollo y mejoras. Entre ellas destacan:

- **Simplificar el modelo de las suspensiones** para integrarlas en la descripción del robot y que puedan simularse con fidelidad.

- **Diseñar escenarios de prueba realistas**: recrear en Gazebo los entornos donde se evaluará el Panter físico permite anticipar su comportamiento y ajustar parámetros antes de la campaña en campo.
- **Añadir y ubicar nuevos sensores**: la simulación facilita estudiar qué tipo de sensor y qué posición en el chasis ofrecen la mejor cobertura y rendimiento antes de la instalación real.
- **Desarrollar algoritmos de navegación en ROS 2**: se pueden implementar y validar tanto planificación global como control local basados en la fusión de sensores y sistemas de localización, todo dentro del entorno virtual.
- **Migrar los controladores del robot físico al entorno virtual**: cuando se disponga del conjunto completo de controladoras (tracción delantera, tracción trasera, etc.), estas podrán integrarse en Gazebo para manejar el simulador con la misma arquitectura de control que el vehículo real.
- Aprovechar los resultados de la simulación para acelerar el desarrollo del Panter físico, ajustando diseño mecánico y lógica de control con datos previamente validados.

Panter no está finalizado, por lo que cualquier cambio que se quiera realizar en él puede ser probado en el simulador de Panter.

Bibliografía

[1] Calero, I. R., "Modelado 3d de un vehículo eléctrico para misiones de rescate en entornos no estructurados," 2023.

[2] Robotics, O., "Ros installation — gazebo documentation," 2025, https://gazebosim.org/docs/all/ros_installation/ (visitado el 2025-06-04).

[3] Robotics, O., "Tutorials – gazebo api," 2025, https://gazebosim.org/api/gazebo/6/tutorials.html (visitado el 2025-06-04).

[4] Ávila, J., "panter_ws – simulador de vehículo robot en ros2 y gazebo," 2025, https://github.com/JorgeAvila102/panter_ws (visitado el 2025-06-04).

[5] Contributors, R. W., "sw_urdf_exporter," 2025, https://wiki.ros.org/sw_urdf_exporter (visitado el 2025-06-04).

[6] arab-meet Contributors, "Export solidworks as urdf – sw2urdf tutorial," 2025,

[7] Robotics, O., "Ros-gz project template guide — fortress," 2025, https://gazebosim.org/docs/fortress/ros_gz_project_template_guide/ (visitado el 2025-06-04).

[8] y Dharini Dutia, M. C., "Ros 2 and gazebo integration best practices [vimeo]," 2025, https://vimeo.com/showcase/9954564/video/767127300 (visitado el 2025-06-04).

[9] Ceauss Caraga, D., "Adaptación del sistema de propulsión de vehículo eléctrico para tareas de rescate," 2023. Grado en Ingeniería Mecánica. Departamento de Ingeniería de Sistemas y Automática.

[10] Contributors, R. W., "joint_state_publisher," 2025, https://wiki.ros.org/joint_state_publisher (visitado el 2025-06-04).

[11] Contributors, G. S. G., "ros_gz_bridge – bridge communication between ros and gazebo," 2025, https://github.com/gazebosim/ros_gz/tree/ros2/ros_gz_bridge#bridge-communication-between-ros-and-gazebo (visitado el 2025-06-04).

[12] Robotics, O., "Ros 2 integration — gazebo fortress," 2025, https://gazebosim.org/docs/fortress/ros2_integration/ (visitado el 2025-06-04).

[13] contributors, W., "List of moments of inertia," 2025, https://en.wikipedia.org/wiki/List_of_moments_of_inertia (visitado el 2025-06-04).

[14] Aparicio, F., Vera, C., y Díaz, V., Teoría de los vehículos automóviles. Dextra Editorial, 2023. Ver capítulo 2.3.3: Coeficientes de adherencia según superficie.

[15] Foundation, O. S. R., "Especificación sdf – elemento joint," 2025, https://sdformat-org.translate.goog/spec?ver=1.12&elem=joint\&_x_tr_sl=en\&_x_tr_tl=es\&_x_tr_hl=es\&_x_tr_pto=sge\&_x_tr_sch=http (visitado el 2025-06-04).

[16] Robotics, O., "Ackermannsteering class reference," 2025, https://gazebosim.org/api/gazebo/6/classignition_1_1gazebo_1_1systems_1_1AckermannSteering.html#details (visitado el 2025-06-04).

[17] Foundation, O. S. R., "Tutorial: Force/torque sensor," 2025, https://classic.gazebosim.org/tutorials?tut=force_torque_sensor (visitado el 2025-06-04).

[18] Foundation, O. S. R., "Ros 2 humble - installation guide," 2025, https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html (visitado el 2025-06-04).

[19] Contributors, R. . C., "Getting started with ros2_control — humble documentation," 2025, https://control.ros.org/humble/doc/getting_started/getting_started.html (visitado el 2025-06-04).

[20] Robotics, O., "Gazebo fortress - install on ubuntu," 2025, https://gazebosim.org/docs/fortress/install_ubuntu/ (visitado el 2025-06-04).

[21] Chacon, S. y Straub, B., "Inicio – sobre el control de versiones: Instalación de git," 2025,

[22] Docs, G., "Clonar un repositorio," 2025, https://docs.github.com/es/repositories/creating-and-managing-repositories/cloning-a-repository (visitado el 2025-06-04).

Anexo A. Instalación del Simulador de Panter

El sistema operativo en el que se diseña el simulador de Panter es Ubuntu 22.04.5, sobre él, hay que hacer las siguientes instalaciones.

- Instalación de ROS 2 Humble de la web oficial [18].
- Instalación de controlador gz_ros2_control de la web oficial [19].
- Instalación de Gazebo Fortress de la web oficial [20].

Luego, para tener el proyecto en el dispositivo, es necesario crear un clon del repositorio de GitHub que contiene el proyecto [4]. Para ello, en primer lugar, hay que seguir las indicaciones de instalación de Git para Linux de la web oficial [21]. Luego, abrimos la Terminal de Linux con la siguiente combinación de teclas: ctrl + alt + t. En ella, escribimos los siguientes comandos, que permiten clonar el proyecto de forma local [22].

Clonamos el repositorio del proyecto

```
$ git clone https://github.com/JorgeAvila102/panter_ws.git
```

Configurar bash

```
$ echo source /opt/ros/humble/setup.bash >> ~/.bashrc
```

```
$ echo source $HOME/panter_ws/install/setup.bash >> ~/.bashrc
```

```
$ echo export IGN_GAZEBO_RESOURCE_PATH="$HOME/panter_ws/install/
description_pkg/share/description_pkg/worlds:$HOME/panter_ws/
install/description_pkg/share/description_pkg/
models:$IGN_GAZEBO_RESOURCE_PATH" >> ~/.bashrc

$ echo export GZ_SIM_RESOURCE_PATH="$HOME/panter_ws/install/
description_pkg/share/description_pkg/worlds:$HOME/panter_ws/
install/description_pkg/share/description_pkg/
models:$GZ_SIM_RESOURCE_PATH" >> ~/.bashrc

$ echo export IGN_GAZEBO_SYSTEM_PLUGIN_PATH="$IGN_GAZEBO_
SYSTEM_PLUGIN_PATH:/opt/ros/humble/lib" >> ~/.bashrc

$ echo export IGN_GAZEBO_PHYSICS_ENGINE_PATH="/usr/
lib/x86_64-linux-gnu/ignition/physics" >> ~/.bashrc

$ echo export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/opt/ros/humble/
lib:$HOME/panter_ws/install/gz_ros2_control/lib" >> ~/.bashrc

$ source ~/.bashrc
```

Organizamos la zona de trabajo (WorkSpace)

Abrimos una nueva terminal:

```
$ cd panter_ws
$ mkdir src
$ mv bringup_pkg control_pkg description_pkg src/
$ mv doc README.md src/
$ mkdir -p ~/panter_ws/src/bringup_pkg/include
$ mkdir -p ~/panter_ws/src/description_pkg/include
$ cd ~/panter_ws
```

```
$ colcon build
```

Otras instalaciones necesarias

- Visual Studio Code:

```
$ cd ~
$ sudo apt install ./code_1.72.0-1664925838_arm64.deb
```

- Xterm:

```
$ cd ~
$ sudo apt update
$ sudo apt-get install xterm
```

Establecer el entorno de trabajo de Visual Studio Code

```
$ cd panter_ws
$ mkdir .vscode
```

Y podemos añadir en la carpeta .vscode, los siguientes archivos de configuración que facilitarán su uso y compilación:

- **c_cpp_properties.json**:

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "/opt/ros/humble/include/**"
```

```
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "c17",
            "cppStandard": "gnu++17",
            "intelliSenseMode": "clang-x64"
        }
    ],
    "version": 4
}
```

- **tasks.json** permite compilar pulsando las teclas ctrl + Shift + b:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "command": "colcon",
            "args": [
                "build", "--symlink-install"
            ],

            "group": "build",
            "label": "colcon: build",

            "presentation": {
                "panel": "new",
                "focus": true,
                "reveal": "silent",
```

```
        "clear": true
    } } ] }
```

Modificación en URDF

Para el correcto funcionamiento, debemos realizar el siguiente cambio en el archivo panter.urdf: cambiar el usuario jorge, por el nuevo usuario (ver en la Figura 55).

```
1432  <gazebo>
1433    <plugin filename="gz_ros2_control-system" name="gz_ros2_control::GazeboSimROS2ControlPlugin">
1434      <robot_param>robot_description</robot_param>
1435      <robot_param_node>robot_state_publisher</robot_param_node>
1436      <parameters>/home/jorge/panter_ws/src/control_pkg/config/control_config.yaml</parameters>
1437    </plugin>
1438  </gazebo>
```

Figura 55. Modificar URDF

Ejecutar este comando en la terminal de Linux.

```
$ sed -i "s|/home/jorge/panter_ws|$HOME/panter_ws|g"
~/panter_ws/src/control_pkg/urdf/panter.urdf
```

Anexo B. Uso del Simulador

Una vez las herramientas se instalan y el entorno de trabajo se modifica, podemos utilizar el simulador.

Compilación del WorkSpace

- **Opción 1**: Sobre algún archivo del proyecto en Visual Studio Code, presionamos ctrl + Shift + b y posteriormente Enter.
- **Opción 2**: en la terminal:

```
$ cd ~/panter_ws
$ colcon build
```

Lanzamiento del simulador

- Control de velocidad:

```
$ cd ~/panter_ws/src
$ ros2 launch bringup_pkg proyect_vel.launch.py
```

- Control de par:

```
$ cd ~/panter_ws/src
$ ros2 launch bringup_pkg proyect.launch.py
```

Importante en el control de par: se debe pulsar el play Figura 56 en Gazebo antes de **5 segundos**, en el caso de que no sean pulsados, los controladores dan error (Figura 57) y es necesario volver a lanzarlo.

Figura 56. Play Gazebo

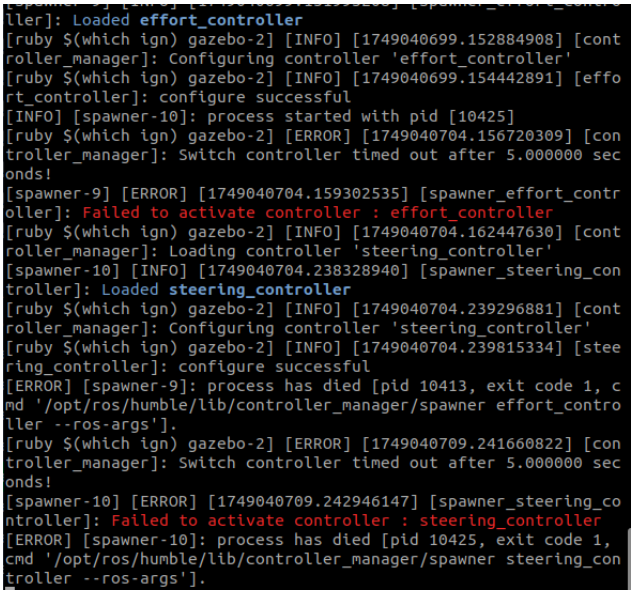

Figura 57. Controladores iniciados erróneamente

Cuando se inicia correctamente, podemos ver los mensajes de la Figura 58.

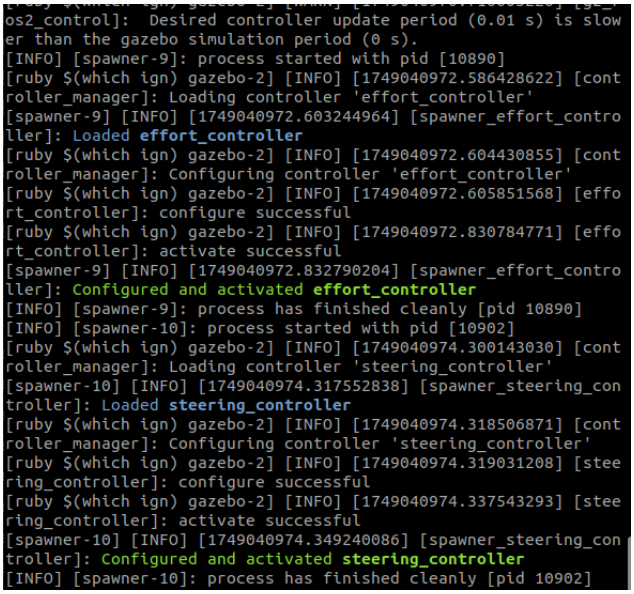

Figura 58. Controladores iniciados correctamente

Comandos de control

Los comandos de teclado para controlar el robot se definen en la Tabla 1.

Para ambos controladores, es necesario pulsar las teclas teniendo la terminal de XTerm abierta y pulsada.

Para finalizar la simulación, se recomienda pulsar 'ctrl + c' sobre la terminal de Linux.

Anexo C. Código del Proyecto

El proyecto está compuesto de una gran cantidad de archivos y con una estructura determinada. Todo código del proyecto estructurado se encuentra en el siguiente repositorio de GitHub [4].