Proyecto Fin de Máster
Máster Ingeniería Electrónica, Robótica y Automática

Modelado y simulación de robots terrestres para la inspección del alcantarillado

Autor: Simón Ernesto Martínez Rozas
Tutor: Jesús Iván Maza Alcañiz

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Master
Máster Ingeniería Electrónica, Robótica y Automática

Modelado y simulación de robots terrestres para la inspección del alcantarillado

Autor:
Simón Ernesto Martínez Rozas

Tutor:
Jesús Iván Maza Alcañiz
Profesor titular

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Master: Modelado y simulación de robots terrestres para la inspección del alcantarillado

Autor: Simón Ernesto Martínez Rozas

Tutor: Jesús Iván Maza Alcañiz

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mi novia Maria y su samilia

A mis amigos

A mis maestros

A Universidad de Antofagasta

A Universidad de Sevilla

A la vida …

AGRADECIMIENTOS

Quisiera agradecer a todas las personas que me han acompañado en este proceso, que me han ayudado durante todo este tiempo de forma incondicional y que han hecho de este logro algo tan suyo como mío.

A María, por acompañarme y brindarme su amor lleno de energía día a día. Has hecho de este proceso un verdadero regalo.

A mi familia, en especial a Luis, Rosa, Onel Nataly por creer en mí, apoyarme y siempre viajar a mi lado.

A mis amigos, que a la distancia me han llenado de su buena energía motivándome a seguir adelante.

A la Universidad de Antofagasta por creer en mi y brindarme la oportunidad de seguir estudiando.

A los profesores que me dieron la oportunidad de trabajar con ellos en este proyecto.

A la vida, por cada día brindar nuevas y lindas oportunidades.

RESUMEN

La constante incorporación de la tecnología robótica en diferentes actividades del día a día y los beneficios que en ella otorga, ha permitido que el campo de la robótica tenga un gran auge y rápida evolución en los últimos años. Por esta razón son cada vez más las oportunidades para el uso de la robótica en diversas aplicaciones, especialmente aquellas que involucran un riego para los trabajadores u operarios.

Bajo esta idea el proyecto SIAR (Sewer Inpection Autonomous Robot) ha desarrollado un robot terrestre para realizar tareas de inspección de alcantarillado, ambiente el cual presenta una serie de riesgos para los humanos debido a la generación de gases tóxicos, ambiente poco higiénico y especio reducido con difícil acceso en caso de emergencia. El inconveniente para el grupo a cargo del proyecto, es que el robot debe ser movilizado desde Sevilla a la ciudad de Barcelona, y una vez en esta movilizar el robot a través de camión y llevarlo hasta dentro del alcantarillado a través de una grúa.

Por esta razón, y como complemento al proyecto, nace la idea de realizar un simulador, que considera el modelo del robot SIAR con las mismas características mecánica y dinámicas de la plataforma real, y las condiciones hostiles del alcantarillado, como angostamiento secciones, canal, zonas curvas, etc. De esta manera se obtiene una representación del entorno real con el modelo que es manipulado a través de un control de lazo abierto teleoperado.

Para desarrollar el simulador, se considera la utilización de software de código abierto, que tengan un soporte y comunidad activa, que además incorpore motores físicos que otorguen realidad a la simulación y que permitan utilizar la misma programación de la plataforma real sobre la simulación. Así es como surge el simulador Gazebo como opción para realizar la simulación, el cual puede integrarse con ROS, para así generar una herramienta poderosa de simulación con potentes motores físicos y en la cual se pueden implementar los mismos códigos que la plataforma real. Además, ambos softwares son de código abierto, y, son soportados por la ORDF y nutridos constantemente por una comunidad muy activa. Cabe destacar que estos softwares son compatibles con otros softwares de código abierto como por ejemplo Meshlab, el cual permite editar figura 3D, lo cual genera un set de softwares que se complementan y permiten tener una simulación con buenos resultados.

Gazebo, al igual que ROS, cuanta con una gran librería y además API que permite generar un gran control sobre los elementos de la simulación. De esta manera, a partir del uso de estos elementos se desarrollarón plugins que manipulan de forma teleoperada los elementos de la simulación, como el movimiento de las ruedas o los brazos del modelo del robot SIAR, y que además comunican Gazebo con ROS para entregar mensajes con información de los elementos de la simulación. Además, se utilizan plugin que generan simulación de sensores como cámaras para visualizar el alcantarillado, y así contar con percepción del entorno.

ABSTRACT

The constant incorporation of robotic technology in different day-to-day activities and the benefits it provides, has allowed the field of robotics to have a great boom and rapid evolution in recent years. For this reason, there are more and more opportunities for the use of robotics in various applications, especially those that involve a risk for workers or operators.

Under this idea, the SIAR (Sewer Inpection Autonomous Robot) project has developed a terrestrial robot to perform sewer inspection tasks, an environment which presents a series of risks for humans due to the generation of toxic gases, an unhygienic environment and a reduced space with difficult access in case of emergency. The disadvantage for the group in charge of the project, is that the robot must be mobilized from Seville to the city of Barcelona, and once in this mobilize the robot through a truck and take it into the sewer through a crane.

For this reason, and as a complement to the project, the idea of creating a simulator was born, which considers the model of the SIAR robot with the same mechanical and dynamic characteristics of the real platform, and the hostile conditions of the sewer, such as narrowing sections, channel, curved areas, etc. In this way, a representation of the real environment is obtained with the model that is manipulated through a teleoperated open loop control.

To develop the simulator, it is considered the use of open source software, which have a support and active community, which also incorporates physical engines that give reality to the simulation and allow to use the same programming of the real platform on the simulation. This is how the Gazebo simulator emerges as an option to perform the simulation, which can be integrated with ROS, in order to generate a powerful simulation tool with powerful physical engines and in which the same codes can be implemented as the real platform. In addition, both softwares are open source, and, are supported by the ORDF and constantly nourished by a very active community. It should be noted that these softwares are compatible with other open source softwares such as Meshlab, which allows you to edit 3D figures, which generates a set of softwares that complement each other and allow you to have a simulation with good results.

Gazebo, like ROS, has a large library and API that allows to generate a great control over the elements of the simulation. In this way, from the use of these elements, plugins were developed that manipulate in a teleoperated way the elements of the simulation, such as the movement of the wheels or the arms of the model of the SIAR robot, and that also communicate Gazebo with ROS to deliver messages with information of the elements of the simulation. In addition, plugins are used that generate simulation of sensors such as cameras to visualize the sewerage, and thus have perception of the environment.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. Antecedentes

En nuestros ordenadores, teléfonos móviles e incluso en labores diarias, se encuentra a menudo un sistema inteligente que procesa información de manera rápida y eficaz, entregando así, grandes beneficios en tiempo y esfuerzo a las personas. Este es uno de los principios básicos de la robótica, es decir, usar tecnología para obtener beneficios, razón por la cual se observa hoy en día que esta es una de las áreas del conocimiento con mayor auge, lo que conlleva consigo la aparición dispositivos y plataformas robóticas destinadas a realizar distintas tareas, especialmente relacionadas a aquellas actividades con alto potencial de riesgo, en las cuales un operador puede exponer su integridad con consecuencia irreversibles.

La inspección de alcantarillado es una tarea de riesgo a la integridad de las personas, especialmente por el tipo de gases que se generan en este ambiente, condiciones higiénicas, sin mencionar la posibilidad de accidentes estructurales que puede ocurrir en este entorno cerrado con muy pocas vías de accesos expeditas.

Bajo esta idea nace el proyecto SIAR (Sewer Inspection Autonomous Robot) el cual consiste en desarrollar un robot terrestre completamente autónomo capaz de navegar e inspeccionar el sistema de alcantarillado de forma autónoma con una intervención humana mínima, y con la posibilidad de controlar manualmente el vehículo. El Consorcio a cargo de la elaboración de este proyecto está compuesto por una PYME portuguesa llamada IDMind (IDM) y dos Universidades, Universidad de Sevilla (USE) y Universidad Pablo de Olavide (UPO).

De esta manera el proyecto SIAR entrega los beneficios que se esperan de un proyecto robótico, pero, aun así, a pesar de su gran potencialidad, como todo proyecto tecnológico, esta plataforma robótica también puede recibir beneficios que hagan del proceso de desarrollo y prueba, etapas con resultados mas eficiente, menos inversión de tiempo, recursos y también con mayor seguridad a los operarios.

Uno de los principales recursos que favorece los proyectos robóticos y que en particular tiene un gran impacto, es el uso de simuladores. Los simuladores permiten virtualizar el robot y el ambiente donde se desarrollan las tareas para probar comportamiento del robot, funcionamiento estructural y de algoritmos robóticos, filosofías de control, potencialidad de sensores, entre otros. Para esto un simulador debe cumplir con ciertas características visuales, de procesamiento de datos y de interpretación de la física que permitan generar un entorno con condiciones muy similares a las que en realidad se realizan las tareas, y también un modelo con cualidades y características muy semejantes a la de la plataforma real.

De esta manera, el presente trabajo describe el proceso realizado para la modelación y simulación de la plataforma robótica SIAR encargada de realizar trabajos de inspección en alcantarillado, para lo cual se considera información utilizada dentro del proyecto, especialmente la relacionada con la estructura de la plataforma, filosofía de control a lazo abierto y los datos obtenidos de los procesos de mapping llevados a cabo por el equipo del proyecto. Cabe destacar que también se recogen por par de los operarios de la plataforma, la experiencia para conocer como es el comportamiento de esta, de tal manera de incorporarla dentro del comportamiento que debe tener la simulación.

Para llevar a cabo este proyecto se evalúa una gran cantidad de simuladores existentes y los cuales poseen diversas características, esto con el fin de satisfacer las necesidades del simulador con el proyecto, las cuales consideran principalmente una buena interpretación de las leyes físicas, buen soporte, pueda implementarse códigos robóticos de la plataforma real y en que sea de código libre, concepto que es tendencia el último tiempo en al robótica, ya que unifica los desarrollos científicos y que permite, avanzar de manera ágil en la realización de proyecto.

1.2. Objetivos

A continuación, se dan a conocer el objetivo general y específicos propuestos que orientaron la realización de este trabajo:

1.2.1. Objetivo General

El objetivo general de este trabajo consiste en modelar y simular la plataforma robótica SIAR encargada de realizar trabajos de inspección de alcantarillado.

1.2.2. Objetivo Especifico

Los objetivos específicos definidos fueron los siguientes:

1. Definir software de simulación que genere entornos y modelos virtuales con comportamientos similar a la realidad y que además permita ejecutar algoritmos de la plataforma real.
2. Modelar en el simulador la plataforma SIAR, con características mecánicas y dinámicas similares a la reales
3. Simular entorno con características similares en las que la plataforma SIAR realiza sus tareas, para probar la dinámica del modelo simulado
4. Implementar un control sobre el modelo el cual permita teleoperar los movimientos del modelo simulado.
5. Implementar cámaras sobre el modelo simulado y capturar imágenes de color y profundidad.
6. Generar una simulación del modelo y entorno abierta a incorporar control autónomo de desplazamiento.

1.3. Motivación

La plataforma SIAR realiza sus tareas de inspección en los alcantarillados de la ciudad de Barcelona, esto implica que cada vez que se quiere realizar una prueba del comportamiento o se realizan mejoras en el código, la estructura, etc, deben viajar desde Sevilla los operarios y la plataforma, además de solicitar un servicio de camión y grúa para desplazar el robot y bajarlo por el alcantarillado respectivamente.

Esta es una de las razones por la cual contar con una herramienta que permita:

- simular la plataforma con un alto grado de precisión y semejanza a la realidad,
- probar los algoritmos de navegación y control con alto grado de confiabilidad,
- probar comportamiento en la plataforma en este entorno

se vuelve un resultado muy favorable con beneficios en tiempo, recursos y seguridad de los operarios.

De esta manera, este trabajo busca generar una herramienta de simulación complementaria al proyecto SIAR. Por lo cual se desarrolla un modelo con características mecánicas y dinámicas semejantes a la plataforma real, un escenario con características similares al entorno donde la plataforma se desenvuelve, un control sobre el modelo que permite realizar tareas de manera tele operadas y además se hace el uso se softwares que permitan mantener los mismos algoritmos robóticos en la simulación y en la plataforma real. De esta manera la simulación realizada tendrá el alcance esperado de un buen simulador.

1.4. Metodología

El trabajo comienza con la revisión bibliográfica, etapa en donde se establecerán conceptos asociados a un simulador, como se originaron y cuáles son los requerimientos que un tipo de software como este debe poseer. Se realiza una pequeña revisión sobre los tipos de simuladores y cuales existen en la actualidad. Posteriormente y en base a los resultados de una encuesta que califica los simuladores, se obtiene una visión preferencias, áreas y tipos de robot en las que se utilizan simuladores. Ademase revisara la tendencia de los softwares relacionados a la robótica. Para llevar a cabo lo anterior se recurrió al análisis de papers en la web con referencia al uso de simuladores, así como también la revisión de sitios web que comentaran sobre uso o preferencias en el uso de simuladores, además de la tendencia de software en robótica.

Como segunda etapa, y una vez obtenido los resultados de punto anterior, se estudió los softwares más interesantes para el desarrollo de este trabajo, lo cual tiene como tendencia el uso de software de código abierto y con comunidad activa como lo son Gazebo y ROS. De esta manera se analiza el modo de instalación de estos softwares, uso, arquitectura y como comunicarlos entre ellos.

Posteriormente se revisa los documentos proporcionados por los participantes del proyecto SIAR, en los cuales se conoce completamente sobre la el comportamiento mecánico de la plataforma, partes importantes a considerar, características de dimensiones, pesos, etc, para el diseño, así como también como filosofías de control y sensores utilizados.

Como cuarto paso se procede a la realización del simulador. Para esto se realiza en Gazebo y basado en el formato SDF modelos diferentes para comenzar con una familiarización con los motores físicos hasta desarrollar finalmente el modelo semejante al real. Posterior a eso de realizan los primeros controles del modelo a través de plugin los que permite interactuar con movimiento del modelo. Se obtiene incorpora posteriormente el mapa 3D del alcantarillado para incorporar a la simulación y se prosigue a la incorporación de un nuevo plugin para tele-operar el modelo. Finalmente se incorporan cámaras como sensores de imagen del entorno y los tf, los cuales permiten generar frame de las partes del modelo de Gazebo en el software de simulación de ROS, Rviz.

Para finalizar, se realizan pruebas para comprobar el funcionamiento del simulador. Las primeras pruebas se realizan solamente con el modelo, y así probar los solamente movimientos de la estructura y de desplazamiento, del cual se obtiene valores que se comparan con los de la plataforma real. Las segundas pruebas se realizan en el entorno que considera el mapa 3D del alcantarillado, así se prueba el comportamiento de la plataforma en diversas superficies, y como a través del control teleoperado este puede sortear las dificultades que este escenario presenta.

Con esto se obtiene el primer simulador realizado para el proyecto SIAR.

2. ESTADO DE LA SIMULACIÓN EN ROBÓTICA

2.1. Introducción

"La simulación robótica es esencial en la caja de herramientas de cada experto en robótica. Un simulador bien diseñado permite probar rápidamente algoritmos, robots de diseño, y realizar pruebas de regresión utilizando escenarios realistas".

Open Source Robotics Fundation

La simulación en el campo de la robótica es un paso de gran importancia para el diseño e implementación de prototipos, algoritmos y sistemas, aportando beneficios económicos, en tiempo y seguridad. Por esta razón, en los últimos años, las herramientas de simulación dinámica[1] [1] de robots han crecido considerablemente, diversificando su uso y características de tal manera que puedan representarse en ellos el comportamiento de robots terrestres, drones, humanoides, entre otros. Si bien hoy en día las opciones de simuladores de este tipo siguen aumentando al igual que su versatilidad, no existe un simulador de propósito general que domine a los demás en términos de rendimiento o aplicación robóticas, si no que este depende de aspectos como el tipo de aplicación, usuario, entre otros, como explica [2].

La gran variedad de modelos y sistemas robotizados que surgen de empresas, universidades o instituciones ven necesario la incorporación de un software de simulación dinámica que les permita simular la manera en que una aplicación robotizada funcionará dentro de un entorno y sus características. De esta manera se puede facilitar la tarea de toma de decisiones y elegir o no, la adecuación o creación de un sistema robotizado. Para tal efecto, mediante el software de simulación se puede visualizar de manera gráfica y simulada un comportamiento aproximado del robot en acción, como por ejemplo brazos robóticos dentro de una línea de producción que pueden ejecutar operaciones de ensamble, soldadura, pintura, moldeado, transferencia de materiales, clasificación de piezas, etc.

2.2. El origen de los simuladores

Las tecnologías de simulación en la robótica tienen su origen en la industria de videojuego [2]. En este tipo de aplicaciones, el costo computacional juega un papel muy importante con el fin de identificar los avatares, por lo cual no se vuelve necesario calcular el Punto de Momento Cero (PMC) [3] para desplazarse y caminar sin caerse. Además, en los videojuegos las fuerzas inviables no son un gran problema, ya que las leyes de la física pueden ser violadas con el fin de disminuir el costo computacional. Sin embargo, en robótica el análisis virtual de la simulación tiene un

[1] La simulación dinámica evalúa los contactos de las articulaciones, las fuerzas de reacción y momentos de un cuerpo (robot) y su entorno. Además, la simulación dinámica permite evaluar criterios de estabilidad como el Punto de Momento Cero (PMC), condición necesaria para la generación de trayectorias de desplazamientos estable. La simulación dinámica es la que más se aproxima al comportamiento real de un robot al tener en cuenta las fuerzas y momentos generados en las articulaciones del robot en su desplazamiento.

enfoque diferente. En este caso los simuladores dinámicos deben cumplir requisitos más estrictos, se debe simular un entorno en las condiciones más realistas posibles, en comparación con los video juegos, donde el tiempo, la precisión computacional y física pueden ser menos restrictivos.

De esta manera, incluyendo motores físicos que respeten las leyes de la física, los simuladores han evolucionado para ser en el análisis y síntesis de comportamientos de robots. La mayoría de las herramientas utilizadas actualmente se basan en ODE [4] o Bullet [5] como motores físicos subyacentes, que originalmente también fueron diseñados para videojuegos, y por lo tanto no cubren todas las necesidades de robótica. Aun así, a pesar de la maduración sustancial del campo del modelado dinámico de los motores físicos [6] y la simulación en las últimas décadas [7], el control de movimientos de robots complejos, sigue planteando desafíos adicionales a los simuladores.

2.3. Requerimientos en simuladores

Dentro de los principales requisitos de los simuladores para los investigadores en robótica aparecen la rapidez y estabilidad de motores de física y software de código abierto [2]. Para esto, una cuestión clave es el cálculo (estabilidad numérica), lo cual presenta fuertes limitaciones en simulaciones con control en tiempo real [8], [9]. Para ser útil como un motor predictivo en bucles de control en tiempo real [10], el simulador debe ser muy rápido en el cálculo de la dinámica y necesita garantizar la convergencia a soluciones físicamente factibles dentro de un marco de tiempo limitado [11].

Un ejemplo de la importancia de esto, se puede apreciar en la simulación de cuerpos rígidos y blandos en contacto con entornos rígidos y dóciles [12] , [13] esto tiene un impacto crítico en las interacciones simuladas entre el robot y el entorno (por ejemplo, la locomoción en varios terrenos, como el hormigón o el césped). El soporte de diferentes tipos de contactos, por ejemplo, con materiales deformables, superficies flexibles y compatibles [14], es fundamental para optimizar los controladores de los robots en diferentes entornos y objetos. El cálculo inexacto de las fuerzas de contacto entre los cuerpos puede dar como resultado contactos poco realistas o fuerzas de contacto físicamente inviables y luego un comportamiento poco realista.

Otro requisito es la capacidad de modelar y simular nuevos tipos de sistemas de actuación, como la impedancia variable o los actuadores blandos [15] esta característica es fundamental para diseñar controladores para nuevos hardware y al mismo tiempo construir interfaces transparentes para escribir el mismo código para el robot simulado y el real.

2.4. Tecnología de simulación

Las herramientas de simulación dinámica en robótica pueden clasificarse descriptivamente, como muestra la Figura 1, entre: “Motores Físicos” (por ejemplo, ODE, Bullet) y software más complejo que aquí llamamos, y según [2] "Simuladores de sistema" (por ejemplo, Gazebo, V-Rep, ARGoS) que se basan en un motor de física y también incluyen simulación de sensores e interfaces robóticas.

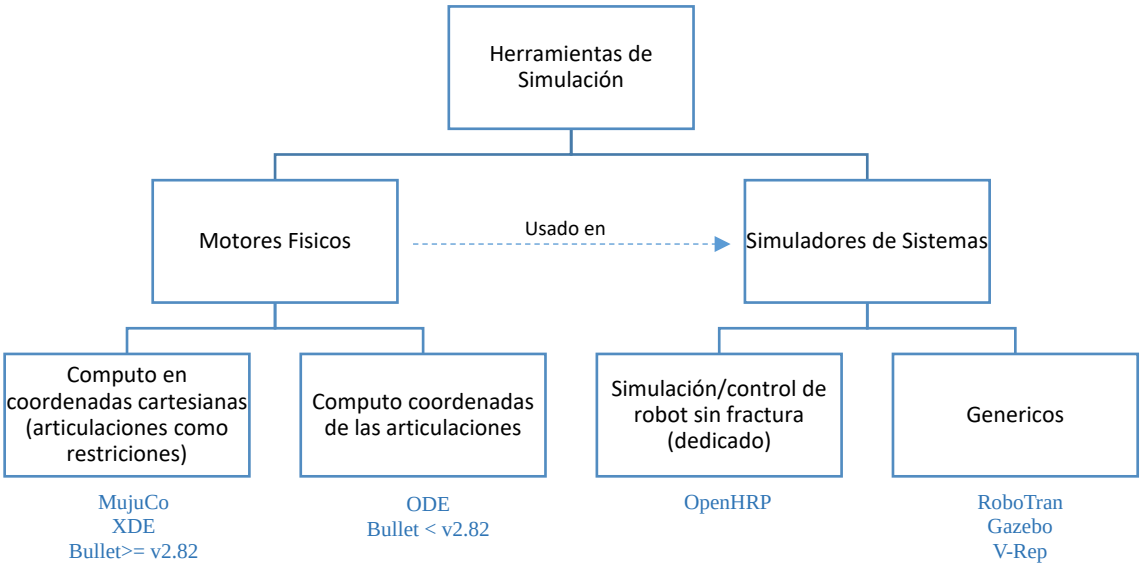


Figura 2. 1 Clasificación de herramientas de simulación

Los “Motores Físicos” pueden ser clasificados en dos grupos, según la forma en que representan las estructuras de cuerpos rígidos: por un lado, (a la izquierda) tenemos software como XDE, MujoCo, que representa la dinámica de los cuerpos rígidos como parámetros, donde las articulaciones son parte de la estructura robótica. Esta es de uso común en la comunidad robótica. Por el otro lado (a la derecha) tenemos herramientas de software como ODE, la cual es heredada de la comunidad de gráficos por computadora, representa las articulaciones como restricciones entre los cuerpos. El primer grupo se beneficia del cálculo directo de las cantidades que se repiten en el control del robot, como Jacobianos, matrices de masa, etc.

La diferencia crítica entre ambos grupos está en la forma en que se calculan las fuerzas de contacto. El segundo grupo considera las fuerzas de contacto como restricciones bilaterales / unilaterales, que se agregan a la lista de restricciones utilizadas para describir las uniones; luego, se usa el mismo solucionador para encontrar las fuerzas para el sistema global, incluidos los contactos y las articulaciones. El primer grupo resuelve las restricciones de los contactos, lo que simplifica notablemente el problema. En general, encontrar las fuerzas de contacto correctas puede ser una carga. El modelado y la simulación de contactos es un área de investigación propia, que se explora activamente mejorando los modelos de contactos [9] y los solucionadores [10].

Los “Simuladores de Sistema” hacen uso de los motores de física para simular la dinámica de los cuerpos en el entorno, pero también proporcionan características adicionales, como simulación de sensores, editores de modelos, desarrollo de controladores o interacción con el operador. En algunos casos, también proporcionan interfaces específicas que facilitan la simulación y el control sin fisuras entre robot real y su carácter virtual, es decir, emulan las interfaces del controlador del robot real de tal manera que el código que se ejecuta en los simuladores se puede cargar al robot sin costo alguno. Esto los hace en la práctica las herramientas "oficiales" para algunas plataformas: por ejemplo, OpenHRP para la serie de robots HRP y el Gazebo para el robot Atlas.

2.5. Variedad de herramientas de simulación disponibles

Como menciona [2], existen fácilmente más de 40 opciones de simuladores, provenientes de empresas, universidades e instituciones, pero también de los llamados “Home Developers” los cuales no comparten ni anuncian sus desarrollos, por lo que la cantidad de proyectos de software activos es probablemente mayor. Dentro de los simuladores más destacados se encuentran:

- **ODE** (Open Dynamics Engine) [4] es un motor físico de código abierto de alto rendimiento para simular la dinámica rígida del cuerpo. Cuenta con todas las características, estable, maduro e independiente de la plataforma con una API C / C ++ fácil de usar. Tiene avanzados tipos de articulaciones y detección de colisión integrada con fricción. ODE es útil para simular vehículos, objetos en entornos de realidad virtual y criaturas virtuales. Actualmente se utiliza en muchos juegos de computadora, herramientas de autoría 3D y herramientas de simulación.

- **Bullet** [5] es un motor de física de código abierto que simula detección de colisiones, y la dinámica cuerpos suaves y rígidos. Principalmente se utilizada para gráficos y animaciones de computadora. Se ha utilizado en videojuegos, así como para efectos visuales en películas (Megamente, Shrek 4, entre otras). La última versión (v.2.82) también admite el Algoritmo del cuerpo articulado de Featherstone[2] [16] y un problema de complementariedad lineal mixto[3]. Esto lo hace más adecuado para aplicaciones de robótica, ya que la dinámica se resuelve en coordenadas conjuntas y la resolución de los contactos es más estable.

- **DART** (Dynamic Animation and Robotics Toolkit) [17] es un motor físico, multiplataforma y de código abierto creada por Graphics Lab y Humanoid Robotics Lab en Georgia Institute of Technology con contribuciones continuas del Laboratorio de Robótica Personal de la Universidad de Washington y Open Source Robotics Foundation. La biblioteca proporciona estructuras de datos y algoritmos para aplicaciones cinemáticas y dinámicas en robótica y animación por computadora. DART se distingue por su precisión y estabilidad debido a su uso de coordenadas generalizadas para representar sistemas articulados de cuerpo rígido y el algoritmo del cuerpo articulado de Featherstone para calcular la dinámica del movimiento. Brinda acceso completo a cantidades cinemáticas internas y dinámicas, como la matriz de masas, Coriolis y fuerzas centrífugas, matrices de transformación y sus derivadas. También proporciona un cálculo eficiente de las matrices jacobianas para puntos corporales arbitrarios y marcos coordinados. Tiene aplicaciones en robótica y animación por computadora porque cuenta con un simulador dinámico multicuerpo y varias herramientas cinemáticas para el control y la planificación del movimiento.

- **Gazebo** [18] Es un simulador que permite de manera precisa y eficiente trabajar con poblaciones de robots en entornos interiores y exteriores complejos. Es de carácter libre, y se ha convertido en un proyecto vibrante con una comunidad muy activa, la cual ha permitido disponer a los usuarios de una gran cantidad de modelos y complementos de simulación. Al alcance de la mano tiene un motor robusto de física, gráficos de alta calidad e interfaces programáticas y gráficas convenientes. Admite múltiples motores de física (ODE, Bullet, DART, Simbody) y utiliza OGRE[4] [19] para proporciona una representación realista de entornos que incluyen iluminación, sombras y texturas de alta calidad. Gracias a su estructura modular y basada en complementos, se puede ampliar

[2] El algoritmo de Featherstone es una técnica utilizada para calcular los efectos de las fuerzas aplicadas a una estructura de articulación y enlaces, es decir, una "cadena cinemática".

[3] Método numérico que resuelve un problema con múltiples soluciones, optimizando resultado a través de la minimización de funciones de costo.

[4] OGRE (Object-Oriented Graphics Rendering Engine) es un motor 3D flexible y orientado a las escenas, escrito en C ++, diseñado para facilitar y hacer más intuitivo que los desarrolladores produzcan aplicaciones utilizando gráficos 3D acelerados por hardware.

fácilmente con nuevas funciones. Fue diseñado idealmente para trabajar con múltiples robots en entornos al aire libre. Es soportado por la Open-Source Robotics Fundation (OSRF) y cuenta con un gran soporte para conectar con ROS. Es la herramienta de software oficial para DARPA Robotics Challenge[5].

- **ARGoS** [20] es un simulador de robot multi-física. Puede simular enjambres de robots heterogéneos en gran escala de manera eficiente. Se puede personalizar fácilmente agregando nuevos complementos. Tiene un enfoque profundamente modular, esto implica que todos los aspectos relevantes de una simulación pueden ser anulados, para una flexibilidad máxima. Los módulos se implementan como complementos cargados en tiempo de ejecución. Con respecto a los simuladores existentes, ARGoS permite agregar funcionalidad en forma de nuevos sensores, actuadores, componentes de robot, visualizaciones e incluso nuevos motores de física y nuevos "medios" de comunicación. Un medio, es un complemento que implementa algoritmos adecuados para simular medios de comunicación del robot (por ejemplo, alcance y soporte, WiFi, stigmergy a través de RFID, etc.). ARGoS se ejecuta para realizar simulaciones detalladas. Los controladores de robot usan una API específica del robot llamada interfaz de control. La misma API se implementa en el robot, lo que permite una transición fluida del código de la simulación a la realidad. Respecto a los motores físicos, el usuario puede elegir para usar en una simulación. Además, el espacio físico se puede dividir en varias regiones, cada una controlada por un motor de física específico. En otras palabras, ARGoS permite simulaciones multimotor. Este simulador se ejecuta en Linux, Mac OSX y recientemente en Windows. Se publica bajo los términos de la licencia de MIT.

- **V-Rep** [21] es un simulador de robots con entorno de desarrollo integrado, el cual se basa en una arquitectura de control distribuido, es decir, cada objeto y/o modelo puede controlarse individualmente a través de un script incrustado, un complemento, un nodo ROS o BlueZero, un cliente API remoto o un sistema personalizado solución. Esto hace que V-REP sea muy versátil e ideal para aplicaciones multi-robot. Los controladores se pueden escribir en C / C ++, Python, Java, Lua, Matlab u Octave. Se utiliza para el desarrollo de algoritmos rápidos, simulaciones de automatización de fábricas, creación rápida de prototipos y verificación, educación relacionada con la robótica, monitoreo remoto, doble verificación de seguridad, etc. Es un software producido por Coppelia Robotics. Al igual que Gazebo, es compatible con múltiples motores de física (ODE, Bullet, Vortex).

- **WebotsTM** [22]es un software de simulación de robótica móvil que le proporciona un entorno de prototipado rápido para modelar, programar y simular robots móviles. Bibliotecas de robots proporcionadas permiten transferir sus programas de control a varios robots móviles reales disponibles comercialmente. Permite definir y modificar una configuración completa de robótica móvil, incluso varios robots diferentes que comparten el mismo entorno. Para cada objeto, puede definir varias propiedades, como la forma, el color, la textura, la masa, la fricción, etc. Puede equipar a cada robot con una gran cantidad de sensores y actuadores disponibles. Puede programar estos robots utilizando su entorno de desarrollo favorito, simularlos y, opcionalmente, transferir los programas resultantes a sus robots reales. WebotsTM ha sido desarrollado en colaboración con el

[5] DARPA Robotics Challenge (DRC) fue un concurso de premios financiado por la Agencia de Proyectos de Investigación Avanzada de Defensa de los Estados Unidos. Realizado de 2012 a 2015, tuvo como objetivo desarrollar robots terrestres semiautónomos que pudieran realizar "tareas complejas en entornos peligrosos, degradados y de ingeniería humana". También buscó hacer más accesible el desarrollo de software y sistemas robóticos más allá del final del programa.

Instituto Federal Suizo de Tecnología en Lausana, probado exhaustivamente, bien documentado y mantenido continuamente durante más de 7 años. Ahora es el principal producto comercial disponible de Cyberbotics Ltd.

- **OpenRave** [23]proporciona un entorno para probar, desarrollar y desplegar algoritmos de planificación de movimiento en aplicaciones de robótica en el mundo real. El foco principal está en la simulación y análisis de información cinemática y geométrica relacionada con la planificación del movimiento. Su naturaleza independiente permite que se integre fácilmente en los sistemas de robótica existentes. Proporciona muchas herramientas de línea de comandos para trabajar con robots y planificadores, y el núcleo en tiempo de ejecución es lo suficientemente pequeño como para ser utilizado dentro de controladores y marcos más grandes. Una aplicación objetivo importante es la automatización de la robótica industrial.

- **Robotran** [24] es un software que genera modelos simbólicos de sistemas de múltiples cuerpos (MBS), que se pueden analizar y simular en Matlab y Simulink. Es desarrollado por el grupo de investigación Multicuerpo del Centro de Investigación en Mecatrónica (CEREM) que forma parte del instituto del Instituto de Ingeniería Mecánica, de Materiales e Ingeniería Civil (iMMC) de la Universidad Católica de Lovaina (UCL).

- **Vortex Studio** [25] es una plataforma de simulación en tiempo real unificada que le permite respaldar la investigación de sistemas robóticos y mecatrónicos, y enseñar conceptos de simulación y prototipos virtuales basados en modelos a estudiantes que utilizan herramientas de nivel profesional. Le permite transformar modelos CAD estáticos en mecanismos virtuales interactivos, lo que le permite probar sistemas inteligentes y diseño de equipos robóticos en entornos 3D envolventes.

- **OpenSim** [26]es un conjunto de herramientas para el modelado musculoesquelético y la simulación dinámica del movimiento, desarrollado en la Universidad de Stanford y respaldado por los NIH de EE. UU. Y por DARPA. Está disponible de forma gratuita, de código abierto y extensible a través de complementos de usuario. El motor de física de este proyecto es **SimBody** [27] una API C ++ de código abierto que implementa los algoritmos de Featherstone para la mecánica corporal rígida, con soporte de diferentes modelos de contacto.
- **Adams** [28] Es definido propiamente como el software de dinámica de multicuerpos (MBD) más famoso y ampliamente utilizado en el mundo. Mejora la eficiencia de la ingeniería y reduce los costos de desarrollo de productos al permitir la validación temprana del diseño a nivel del sistema. Los ingenieros pueden evaluar y gestionar las complejas interacciones entre disciplinas, incluidos el movimiento, las estructuras, la actuación y los controles para optimizar mejor los diseños de productos en cuanto a rendimiento, seguridad y comodidad. Junto con amplias capacidades de análisis, Adams está optimizado para problemas a gran escala, aprovechando los entornos informáticos de alto rendimiento. Utilizando la tecnología de solución de dinámica multicuerpo, ejecuta dinámicas no lineales en una fracción del tiempo requerido por las soluciones FEA[6]. Las cargas y fuerzas calculadas por las simulaciones de Adams mejoran la

[6] El análisis por elementos finitos (FEA, siglas en inglés de *Finite Element Analysis*) es una técnica de simulación por computador usada en ingeniería. Usa una técnica numérica llamada método de los elementos finitos (FEM). Comúnmente se usa FEA en determinar los esfuerzos y desplazamientos en sistemas mecánicos.

precisión de FEA al proporcionar una mejor evaluación de cómo varían a lo largo de una gama completa de entornos operativos y de movimiento.

- **MORSE** [29] es un simulador genérico para la robótica académica. Se centra en la simulación 3D realista de entornos pequeños a grandes, en interiores o al aire libre, con una a décima parte de robots autónomos. Puede controlarse completamente desde la línea de comandos. Las escenas de simulación se generan a partir de scripts de Python simples. Viene con un conjunto de sensores estándar (cámaras, escáner láser, GPS, odometría, etc), actuadores (controladores de velocidad, controladores de puntos de ruta de alto nivel, controladores conjuntos genéricos) y bases robóticas (quadrotors, ATRV, Pioneer3DX, genéricos 4 vehículo de ruedas, PR2, entre otros). Los nuevos se pueden agregar fácilmente. La representación MORSE se basa en el Blender Game Engine. El Game Engine basado en OpenGL admite shaders, proporciona opciones avanzadas de iluminación, admite texturas múltiples y utiliza la biblioteca Bullet de última generación para la simulación física.

- **MuJoCo** [30] es un motor dinámico desarrollado principalmente por E. Todorov y ahora propiedad de Roboti LLC. Es uno de los motores de física más recientes, concebido para simular robótica y sistemas biomecánicos. Es compatible con cálculos paralelos, proporciona dinámica inversa con contactos y restricciones de igualdad, implementa varias dinámicas de contacto. Es adecuado para la optimización del control y se puede usar en tiempo real dentro de un lazo de control.

- **XDE** [31] es un motor de choque de realidad virtual que simula las operaciones de montaje, desmontaje y ensamblaje para el diseño de unidades mecánicas / trenes de potencia. A partir de los datos CAD (Computer Assisted Design) que se importan automáticamente, se ocupará de piezas mecánicas rígidas, flexibles (cables) y deformables, pero también de golpes, retrocesos, deslizamientos, fenómenos de deformaciones elásticas, etc. Estos códigos se pueden integrar en un entorno virtual e interactivo para simular el gesto del operador. Es desarrollado por CEA LIST. Utilizado para aplicaciones de realidad virtual en contextos industriales.

- **OpenHRP3** [32] OpenHRP3 (Open Architecture Human-centered Robotics Platform versión 3) es una plataforma de software integrada para simulaciones de robots y desarrollos de software. Esta versión sé distribuye como un software de código abierto. Permite a los usuarios inspeccionar un modelo de robot original y un programa de control mediante simulación de dinámica. Proporciona varios componentes de software y bibliotecas de cálculo que se pueden utilizar para desarrollos de software relacionados con la robótica. El motor de cálculo de dinámica está casi desarrollado por "Nakamura Lab, Depto. De Mechano Informatics, Universidad de Tokio" y la interfaz gráfica está a cargo de "General Robotix, Inc". Las otras partes se desarrollan como un trabajo cooperativo de "Humanoid Resarch Group" y "Task-Intelligence Research Group" en "Intelligent Systems Research Institute", "Instituto Nacional de Ciencia y Tecnología Indrustrial Avanzada (AIST)".

2.6. Preferencia de simuladores en los usuarios

Basándose en los resultados de la encuesta realizada en [2], y considerando esta como una de las pocas instancias para detectar la preferencia de los usuarios, se obtienen una idea de los simuladores robóticos más usados en Universidades, Centro de Investigación, Instituciones y Empresas. Esto resultado además están relacionado con una tendencia en el uso principalmente

de sistema GNU/Linux, código C++, herramienta de simulación preferiblemente de código abierto y uso de middleware robótico.

En dicha encuesta, primeramente, los participantes evaluaron según su experiencia, las características más importantes para una buena simulación, en la cual se destaca con mayor importancia la estabilidad de simulación. Los resultados se observan en la Tabla 1.1.

Tabla 2. 1 Características más importantes de un simulador

#	Características	Calificación media
1	Estabilidad de Simulación	5(4.50 $\pm$0.58)
2	Rapidez	4(4.05$\pm$0.75)
3	Precisión de simulación	4(4.02$\pm$0.71)
4	Precisión de resolución de contacto	4(3.91$\pm$0.92)
5	Misma interfaz entre el sistema real y el simulado	4(3.67$\pm$1.26)
6	Carga computacional (CPU)	3(3.53$\pm$0.85)
7	Carga computacional (memoria)	3(3.22$\pm$0.90)
8	Representación visual	3(3.02$\pm$1.02)

Además, los participantes indicaron los criterios más importantes para elegir un simulador, lo cual se resumen en la Tabla 1.2. Los resultados arrojaron como lo más importantes una simulación realista (cercana a la realidad) y licencia de código abierto del software, es decir, buenos motores de física y herramientas compartidas por parte de la comunidad. En un segundo nivel, se indicó simulaciones rápidas e interfaces sin fisuras para escribir el mismo código para un robot simulado y real. Con esto se infiere que los simuladores se utilizan para prototipar y optimizar los controladores, es decir, la velocidad es importante, así como el hecho de no tener que reescribir el código cuando se cambia de un sistema a otro.

Tabla 2. 2 Criterios más importantes para la elección de un simulador

#	Mas importantes criterios	Usuarios
1	Simulación muy cercana a la realidad	32%
2	Software libre	24%
3	Mismo código para el robot real y simulado	19%
4	Rápido y ligero	11%
5	Personalización	6%
6	No solapamiento de cuerpos	3%

Por otra parte, para tener una idea de los simuladores existentes, se pidió a los participantes que indicaran su familiaridad con algunas herramientas de simulación conocida y/o utilizada, siendo el simulador Gazebo el que se lleva el primer lugar en esta parte. Un resumen del porcentaje de respuestas para las herramientas más relevantes se muestra en la Tabla 1.3.

Tabla 2. 3 Conocimiento y uso pasado/presente de simuladores

Herramienta	Actualmente usado y es la principal herramienta	Actualmente usado, pero no es la principal herramienta	Actualmente usado solo para probarlo	Usado una vez solo para probar	Usado y luego abandonado	Conoce, pero nunca uso	Nunca oyó
Gazebo	13%	7%	3%	18%	10%	34%	15%
ODE	11%	15%	5%	18%	22%	22%	10%
Bullet	5%	13%	7%	12%	10%	29%	24%
V-Rep	5%	3%	3%	18%	3%	29%	39%
Webots	4%	7%	1%	16%	13%	32%	27%
OpenRave	5%	3%	2%	7%	5%	29%	49%
Robotran	4%	0%	1%	4%	2%	13%	76%
XDE	5%	3%	0%	3%	1%	14%	74%
Blender	5%	17%	7%	22%	6%	28%	15%
MujoCo	2%	0%	0%	4%	2%	21%	71%
Nvidia	1%	1%	4%	12%	7%	43%	32%
OpenSIM	3%	4%	3%	8%	1%	41%	40%
Vortex	3%	2%	0%	5%	5%	17%	68%
RoboRobo	3%	1%	0%	0%	1%	4%	91%

También se solicitó que indicaran la herramienta de simulación actual que están utilizando, lo cual nuevamente tiene en primer lugar al simulador Gazebo. Los resultados se muestran en la Figura 1.2.

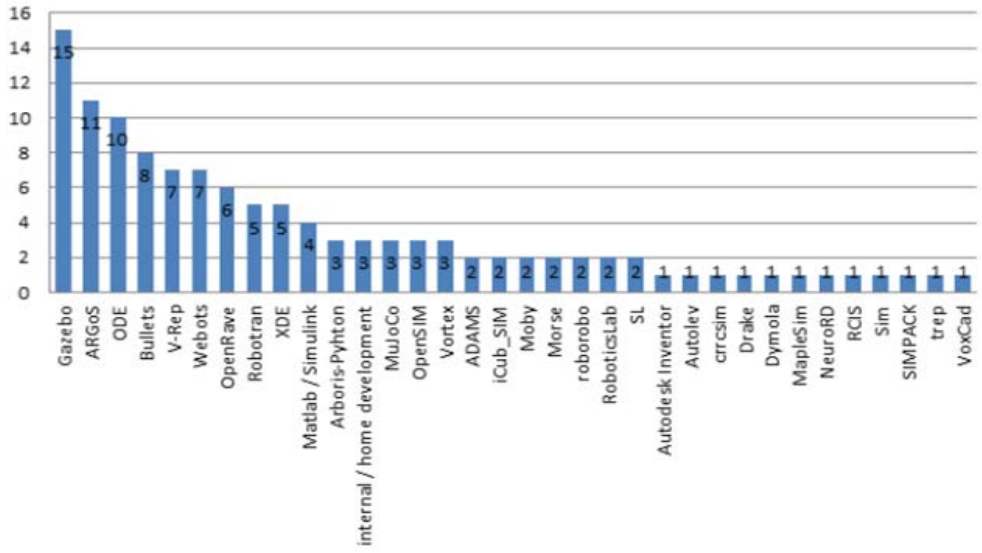


Figura 2. 2 Las herramientas de simulación actualmente en uso entre los participantes de la encuesta. El eje vertical informa el número de usuarios que indica que la herramienta es la principal en uso.

Para conocer el grado de satisfacción de los participantes, se solicitó evaluar los simuladores usados, así evitar elecciones inapropiadas de software y brindar sugerencias útiles a la comunidad. Los resultados de la tabla 1.4, ubican al simulador Gazebo como el mejor evaluado.

Tabla 2. 4 Calificaciones para el nivel de satisfacción del usuario de las herramientas más difundidas

Herramienta	Documentación	Soporte	Instalación	Tutoriales	Proyectos activos y comunicas	API	Global
Gazebo	3.47±0.99	4.00±1.07	3.93±1.03	3.53±1.12	3.80±0.45	3.67±0.82	3.88±0.91
ARGos	3.40±0.70	3.90±0.99	4.70±0.48	4.20±0.63	4.60±0.74	4.30±0.67	4.17±0.70
ODE	3.80±0.63	3.40±1.07	4.10±1.28	3.20±1.13	3.90±1.25	3.40±1.26	3.59±1.15
Bullet	3.37±1.06	3.62±0.91	4.75±0.46	4.00±0.76	3.75±0.74	3.87±0.83	3.96±0.78
V-Rep	4.28±0.76	4.43±0.79	4.71±0.76	4.14±0.90	4.28±0.53	4.41±1.07	4.25±0.80
Webots	3.86±1.07	3.57±1.13	4.43±0.79	3.43±1.51	4.42±0.69	4.57±0.53	4.20±0.96
OpenRave	3.50±0.55	4.67±0.52	4.17±0.75	3.50±1.22	4.33±0.52	4.33±0.52	4.12±0.70
Robotran	3.60±0.55	3.80±0.45	3.80±0.45	3.20±0.84	4.20±0.84	3.80±0.45	3.66±0.63
Vortex	3.33±1.15	3.67±1.53	5.00±0.00	2.67±0.58	3.67±1.15	3.33±0.58	3.48±0.80
OpenSIM	4.33±0.58	4.67±0.58	3.67±0.58	3.00±1.00	4.00±0.58	3.67±0.58	4.00±0.55
MujoCo	2.33±1.15	1.97±0.58	4.33±1.15	3.33±1.15	4.67±0.00	5.00±0.00	3.62±0.66
XDE	1.40±0.55	2.80±1.09	3.60±1.09	2.80±1.09	3.40±0.84	3.00±1.00	2.83±1.07

Considerando que cada área de investigación en robótica tiene necesidades específicas, los participantes mencionaron las principales aplicaciones y robots simulados en las herramientas más difundidas. Los resultados se aprecian en la Tabla 1.5

Tabla 2. 5 Información sobre la aplicación de los softwares más distribuidos, y los robots simulados

Herramienta	Principales Aplicaciones		Principales Robots Simulados	
Gazebo	33% robots móviles	Robots de servicios, robots humanoides	40% robot atlas, 33% plataforma personalizada	Vehículos con ruedas, quadrotor, turtlebot, PR2
ARGos	46% robots móviles, 36% robótica de enjambre	Robots voladores	64% plataformas khepera/epuck	marXbot/footbot, quadrotor
ODE	50% robots humanoides	Robots multi piernas, simulación numérica de sistemas físicos	40% robots multi piernas	Iclub
Bullet	25% robots humanoides, simulación numérica de sistemas físicos	Manipuladores industriales, análisis de movimiento humano, robots móviles, robots multi piernas	25% robots multi piernas	
V-Rep	29% robots móviles	Manipuladores industriales, robots humanoides, diseño mecánico, arquitectura cognitiva, rotica de servicio		29% Nao, quadrotor, veiculos con ruedad, Bioloid, khepera/ epuck/ thymio
Webots	43% robótica móvil, robótica multi piernas	Robótica humanoide		29% UKA, LWR, Lego Mindstorm, vehiculo con ruedas
OpenRave	50% robots humanoides, robótica de servicio		50% PR2	
Robotran	60% robótica humanoides	Análisis de movimiento humano, robots voladores	60% Coman	
Vortex	66% robótica móvil	Robótica humanoide	66% vehículos con ruedas	Brazo Barret
OpenSIM		33% robótica de asistencia, simulación numérica de sistemas físicos, robótica de humanoides		33% khepera/ e-puck/thymio
MujoCo		33% diseño mecánico, robótica humanoide, simulación numérica de sistemas físicos	66% Atlas, Nao, Shadow hand, Barret Arm, HRP2	
XDE	40% robótica humanoide	20% manipuladores industriales, simulación numérica de sistemas físicos, análisis de movimiento humano	40% robots industriales, KUKA LWR	20% vehículos con ruedas

Para complementar la información anterior, también se extrajo una idea de las herramientas más utilizadas para una selección de áreas de investigación. Los resultados se muestran en la Tabla 1.6.

Tabla 2. 6 Herramientas más difundidas por áreas de investigación

Area de Investigación	Usuarios	Softwares mas usados	Otros softwares usados
Robotica humanoide	32%	ODE, Gazebo, Robotran, OpenRave, Arboris-Python, XDE, iCub_Sim	Drake, MapleSim, MuJoCo, OpenSIM, RoboticsLab, SL, Vortex, V-Rep, Webots, own code
Robotica Movil	25%	Gazebo, ARGoS, Webots, VRep, Vortex	ADAMS, Autodesk Inventor, Bullet, ODE, Morse, roborobo, Sim, own code
Robotica multi piernas	13%	Webots, ODE	Gazebo, ADAMS, Autolev, Bullet, Moby, RoboticsLab, SIMPACK, VoxCad
Robotica de servicio	12%	Gazebo, OpenRave	OpenSIM, V-Rep, Morse, RCIS, SL
Simulación numerica de sistemas fisicos	8%	Bullet	MuJoCo, ODE, OpenSIM, Simulink, trep, XDE
Robots voladores	6%	ARGoS	Robotran, crrcsim, Gazebo, Simulink/Matlab
Robotica de enjambre	5%	ARGoS	roborobo
Manipuladores Industriales	5%		Bullets, Dymola, Matlab, V-Rep, XDE
Diseño Mecanico	4%		Moby, MuJoCo, V-Rep, own code
Analisis de Movimiento Humano	3%		Robotran, Bullet, XDE
Robots serpientes	3%	ODE	Matlab

2.7. Tendencias de software en Robótica

Dada las dificultades que presenta el desarrollo de un robot, ya que estos es un sistema complejo que requieren una experiencia en muchos campos, la comunidad robótica se ha volcado desde comienzos de milenio a fomentar nuevas herramientas software que potencien los resultados en investigación y aplicaciones. A consecuencia de esto, los desarrollos que se han llevado a cabo tienen principalmente un carácter colaborativo, abierto al público en general y con una gama de posibilidades muy potentes.

Esta idea ofrece una solución a la recurrente y conocida problemática de “reinventar la rueda”. Esto se refiere, por ejemplo, si en un caso se cuenta con un plazo determinado para desarrollar un robot programando, seguramente la mayor cantidad de tiempo será invertido en crear un sistema básico de comunicación y/o programación, lo que, en consecuencia, al final del proyecto, arribará en resultados similares a los que otros desarrolladores están realizando.

En respuesta a lo anterior, el principal software desarrollado bajo este concepto es ROS (Robotic Operating System), el cual proporciona al usuario un conjunto de herramientas y atajos para iniciar una aplicación desde un nivel superior, y lo incorpora en un mundo de colaboración con desarrolladores. Es decir, ROS permite contar con estructuras ya diseñadas y programadas que luego se pueden modificar y compartir, evitando, de esta manera, comenzar desde cero con cada diseño y superando la inversión de tiempo inherente a la construcción de algoritmos, piezas comunes como brazos y ruedas, herramientas de planificación de movimiento, reconocimiento de objetos, navegación 2D, entre otras cosas. De esta manera, se evita hacer lo que han hecho otros desarrolladores antes en sus proyectos.

La administración de ROS es llevada por la Open Source Robotics Foundation (OSRF), (Fundación para la programación robótica de Código Abierto), ubicada en la ciudad de San Francisco, Estados Unidos. La misión principal de esta institución es: *“apoyar el desarrollo, la distribución y la adopción de software de código abierto para su uso en la investigación de la robótica, la educación y el desarrollo de productos”*. Bajo esta idea la OSRF busca universalizar el emprendimiento y la investigación de la programación en el ámbito robótico mundial. [33]

Los dos principales proyectos supervisados por la OSRF son ROS y el precitado simulador

Gazebo. Aunque ambos tuvieron un origen diferente y fueron herramientas construidas de forma independiente por desarrolladores diferentes, con el tiempo resultaron ser complementarias y administradas, finalmente, por la OSRF.

2.7.1. Origen de ROS como software libre

A partir del año 2000, el Stanford Artificial Intelligence Laboratory (SAIL) (Laboratorio de Inteligencia Artificial de Stanford) de la Universidad de Stanford donde "Stanford Ai robot" (Stair) y robots Program (Pr), realizó grandes esfuerzos por desarrollar sistemas robóticos accesibles y sin restricciones, es decir, de código abierto, con el fin de ofrecer modelos al alcance de la comunidad, y así, fomentar la generación de conocimiento de forma colaborativa en el campo de la robótica. Tales trabajos tuvieron frutos el año 2007, año en el que ROS hace aparición.

El año 2008 el laboratorio de investigación robótica Willow Garage ubicado en Menlo Park, California, en los Estados Unidos, toma este interesante proyecto decide respaldarlo. Willow Garage es una verdadera incubadora de empresas en este tema y, de forma visionaria, puso su atención en el camino ya adelantado por los programas académicos de la Universidad de Stanford, y aún más, en sus desarrollos.

En el año 2013, la administración de ROS cambia y es transferida a la Open Source robotics Fundation (OSRF), la cual se mantiene vigente en la actualidad.

En los últimos años, la comunidad de usuarios de ROS ha crecido alrededor de todo el mundo, marcando presencia en los diversos continentes. Además, históricamente la mayoría de los usuarios de ROS estaban en laboratorios de investigación, pero ahora se observa cada vez más la incorporación de este al sector comercial, particularmente en la robótica industrial y de servicios.

La comunidad ROS es muy activa. Según mediciones realizadas por la organización, la comunidad ROS tiene más de 1.500 participantes en la lista de usuarios "ros-usuarios", más de 3.300 usuarios en la wiki de documentación colaborativa y unos 5.700 usuarios en el sitio web ROS Answers Q & A. La wiki tiene más de 22,000 páginas wiki y más de 30 ediciones de páginas wiki por día. El sitio web de preguntas y respuestas tiene 13,000 preguntas formuladas hasta la fecha, con una tasa de respuesta del 70% por ciento. La licencia está liberada bajo los términos de la licencia BSD (Berkeley Software Distribution), definido como un software de código abierto, gratuito para el uso comercial y de investigación. [34]

2.7.2. Origen de Gazebo como software libre

El desarrollo de Gazebo comenzó en el otoño de 2002 en la Universidad del Sur de California. Los creadores originales fueron el Dr. Andrew Howard y su alumno Nate Koenig. El concepto de un simulador de alta fidelidad surgió de la necesidad de simular robots en entornos al aire libre bajo diversas condiciones. Como simulador complementario de Stage, se eligió el nombre Gazebo como la estructura más cercana a un escenario al aire libre. El nombre se ha estancado a pesar del hecho de que la mayoría de los usuarios de Gazebo simulan ambientes interiores.

Con los años, Nate continuó desarrollando Gazebo mientras completaba su doctorado. En 2009, John Hsu, un ingeniero de investigación sénior en Willow, integró ROS y PR2 en Gazebo, que desde entonces se ha convertido en una de las principales herramientas utilizadas en la comunidad de ROS. Unos años más tarde, en la primavera de 2011, Willow Garage comenzó a brindar apoyo financiero para el desarrollo de Gazebo.

En 2012, Open Source Robotics Foundation (OSRF) se separó de Willow Garage y se convirtió en el administrador del proyecto Gazebo. Luego de un importante esfuerzo de desarrollo por parte de un equipo de individuos talentosos, OSRF usó Gazebo para ejecutar el Virtual Robotics Challenge, un componente del DARPA Robotics Challenge, en julio de 2013.

OSRF continúa desarrollando Gazebo con el apoyo de una comunidad diversa y activa, los cuales invitan a estar atentos al desarrollo de emocionantes simuladores relacionados con robots. [18]

2.8. Colaboración ROS-Gazebo

EL desarrollo de robótica ha llevado aparejada históricamente una serie de dificultades. Esto se ha visto largamente reflejado en desarrolladores alrededor del mundo, los cuales estuvieron emprendiendo esfuerzos aislados por dar solvencia y estabilidad a esta interesante rama del conocimiento, dando pasos carentes de conexión entre sí y apelando a metodologías diversas. Esta diversidad de metodologías ha llevado a que los desarrolladores inviertan grandes cantidades de tiempo y de esfuerzo en la formulación de plataformas y sistemas de base que permitan la operabilidad de los diversos aspectos de un robot (como, por ejemplo, algoritmos robóticos).

Como respuesta, los esfuerzo que realiza actualmente la OSRF a través de software de código abierto ha logrado dar solución a esta problemática, estableciendo una base para la comunidad robótica. Una muestra de esto, es el uso de estos softwares en el DARPA Robotic Challenge (DRC), competencia robótica que marcó un precedente en la forma de aplicar conocimiento en el desarrollo de diversos tipos de Robot, y la cual reunió a equipos de todo el mundo, con gran experiencia y jerarquía en esta disciplina.

La OSRF estuvo relacionada con el DRC desde el comienzo en junio de 2012 hasta su finalización el año 2015, donde la fundación pudo corroborar in situ, que el uso de ROS y Gazebo durante la competencia fue generalizado. Esto se refleja en que, de los 23 equipos de finales de la DRC, 18 equipos usaban ROS y 14 equipos usaban Gazebo, motivado principalmente a que este como simulador y gracias y sus características permitió a los desarrolladores probar conceptos en entornos virtuales robustos sin arriesgar sus hardware valiosos. Esto marco un hito de gran alegría e importancia para la fundación, especialmente al ver el impacto que tenía el uso de software de código abierto para robots en tal competición. [35]

Esto significo a la vez, realizar un exhaustivo trabajo de tres años en paralelo con la competencia, con el fin de actualizar y mejorar el desempeño de Gazebo para cumplir con las necesidades de simulación surgentes en los equipos durante la competencia, incluido el hosting del Virtual Robotics Challenge (VRC) en junio de 2013.

A través de conversaciones con equipos que utilizaron Gazebo en su desarrollo de software y pruebas, incluidos equipos que utilizan robots distintos al Atlas que modelamos para el VRC. De ellos, tanto el primer equipo Team KAIST como el tercer lugar Tartan Rescue discutieron el uso de Gazebo, en particular para desarrollar soluciones para la recuperación de caídas y la salida de un vehículo. Esto dado como realimentación para que Gazebo brinde a los involucrados en la robótica las herramientas que necesitan para desarrollar de forma segura software de robots para manejar situaciones inseguras, sin riesgo para las personas o el hardware.

En la actualidad Gazebo dispone de la herramienta “gazebo_ros_pkgs” el cual es un conjunto de paquetes ROS que proporcionan las interfaces necesarias para simular un robot en el simulador de cuerpo rígido 3D Gazebo para robots. Se integra con ROS utilizando mensajes ROS, servicios y reconfiguración dinámica. De esta manera se logra controlar y manipular el robot simulado en Gazebo a través de la filosofía en nodos de ROS. [36]

3. DESCRIPCIÓN DE SOFTWARE UTILIZADOS

3.1. Introducción

En el presente capítulo se describe los softwares utilizados para el desarrollo del proyecto. Para esto, se considera la información recogida en el capítulo anterior, donde destacaron los softwares distribuidos por la OSRF, es decir Gazebo y ROS como medios para llevar a cabo proyectos en el ámbito de la robótica. El primero para diseñar la plataforma robótica en un entorno y probar el funcionamiento del código robótico y el segundo permitiendo desarrollar el código a implementar en el robot. Otras razones es que son códigos abiertos, la comunidad y el soporte esta activo, conforman una interfaz sin fisuras para escribir el mismo código en un robot simulado y real, y, específicamente en el caso de Gazebo, utiliza motores físicos y herramientas de graficas 3D para simular robots en un entorno con condiciones reales Gazebo.

3.2. ¿Qué es Gazebo?

Gazebo es un simulador multi-robot 3D con dinámica. Ofrece la capacidad de simular de manera precisa y eficiente poblaciones de robots, objetos y sensores en entornos interiores y exteriores complejos. Gazebo genera realimentación realista por parte de sensores, interacciones físicamente verosímiles entre los objetos y una simulación precisa de la física de cuerpos rígidos. Cuenta con un motor robusto de física, gráficos de alta calidad e interfaces programáticas y gráficas convenientes. Es gratuito y cuenta con el respaldo de una comunidad vibrante. Puede ser descargado directamente desde página oficial para Ubuntu, Debian, Fedora, Mac y Windows.

3.2.1. Características de Gazebo

Algunas de las principales características de Gazebo son [18]:

- **Simulación Dinámica**: Usa múltiples motores de física de alto rendimiento.
- **Avanzadas Graficas 3D**: Proporciona realista representación de entorno.
- **Sensores y ruido**: Permite generar datos de sensor, opcionalmente con ruido.
- **Plugins**: Desarrollo de plugin personalizados para control de robots, sensores y entorno. Brindan acceso directo a la API de Gazebo.
- **Modelos de Robots**: Proporciona muchos modelos de robots y es posible crear el propio usando formato SDF[7] [37].
- **Transporte TCP/IP**: puede ejecutar la simulación en servidores remotos e interactuar con Gazebo a través de mensajes basados en socket utilizando Google Protobufs.
- **CloudSim**: es posible ejecutar Gazebo en Amazon AWS y GzWeb para interactuar con la simulación a través de un navegador.
- **Herramientas en línea de comandos**: Extensas herramientas de línea de comando que facilitan la introspección y el control de la simulación.

[7] SDF, simulation descripción format (formato de descripción de simulación), es un formato XML que describe objetos y entornos para simuladores, visualización y control de robots. Originalmente desarrollado como parte del simulador de robot Gazebo. Con los años, SDF se ha convertido en un formato estable, robusto y extensible capaz de describir todos los aspectos de los robots, los objetos estáticos y dinámicos, la iluminación, el terreno e incluso la física. En la actualidad es soportado por la OSRF.

3.2.1. Componentes de Gazebo

A continuación, se describen los elementos involucrados al ejecutar una simulación en Gazebo [38].

3.2.1.1 Archivo mundo – "World"

Contiene todos los elementos de una simulación, robots, luces, sensores y objetos estáticos. El formato del archivo es SDF [39], y, por lo general, tiene una extensión ".world".

El servidor Gazebo (gzserver) lee este archivo para generar y poblar un mundo.

3.2.1.2 Archivos modelo – "Model"

Representa una entidad física. Utiliza SDF, pero solo debe contener un único modelo. El propósito es facilitar la reutilización del modelo y simplificar los archivos del mundo.

Se proporcionan varios modelos en la base de datos modelo en línea que pueden ser insertados en Gazebo. Mas información en [38].

3.2.1.3 Variables de entorno

Gazebo utiliza una serie de variables de entorno[8] para localizar archivos y configurar las comunicaciones entre el servidor y clientes. Los valores predeterminados que funcionan para la mayoría de los casos se compilan en. Esto significa que no necesita establecer ninguna variable. Mas información en [38].

3.2.1.4 Servidor Gazebo - gzserver

Analiza un archivo de descripción "world" dando en la línea de comando. Luego simula el mundo utilizando un motor de física y sensor. No incluye interfaz gráfica. El servidor puede ejecutarse dado el siguiente comando: gzserver *nombre_archivo_worl*. Mas información en [38].

3.2.1.5 Cliente Gazebo - gzclient

Proporciona una interfaz gráfica (GUI) para visualizar e interactuar con los elementos de una simulación. Para esto se conecta a un gzserver en ejecución. También una herramienta que permite modificar la simulación en ejecución. Se ejecuta en la línea de comando como "gzclient".

3.2.1.6 Plugins

Los plugins proporcionan un mecanismo simple y conveniente para interactuar con Gazebo. Los plugins pueden cargarse en la línea de comandos o especificarse en un archivo SDF.

[8] Las variables de entorno son un valor dinámico cargado en la memoria, que puede ser utilizado por varios procesos que funcionan simultáneamente. En la mayoría de los sistemas operativos, la ubicación de algunas bibliotecas o de los archivos ejecutables del sistema más importantes puede variar según la instalación.

3.2.2. Arquitectura de Gazebo

Usa una arquitectura distribuida, con bibliotecas separadas para la comunicación, simulación física, renderización, interfaz de usuario y generación de sensores.

3.2.2.1 Gazebo Master

Es un servidor que proporciona búsqueda y administración de topics. Un master puede manejar múltiples simulaciones de física, generadores de sensores y GUI.

3.2.2.2 Biblioteca de comunicación

- Dependencias: Protobuf[9] y boost :: ASIO[10]
- API externa: Admite comunicación con nodos Gazebo sobre topics mencionados.
- API interna: Ninguna

Es utilizada por casi todas las bibliotecas. Actúa como el mecanismo de comunicación y transporte para Gazebo. Es compatible con el modo de comunicación publicar / suscribir, usando nodos y topic. Por ejemplo, un mundo simulado publica actualizaciones de pose corporal, y los sensores generados junto con la GUI utilizaran estos mensajes para producir respuestas.

3.2.2.3 Biblioteca de Física

- Dependencias: Motor de dinámica (con detección de colisión interna)
- API externa: proporciona una interfaz simple y genérica para la simulación física
- API interna: define interfaz para bibliotecas físicas de motores dinámicos de terceros.

Proporciona una interfaz simple y genérica para componentes de simulación fundamentales, incluidos cuerpos rígidos, formas de colisión y uniones para representar las restricciones de articulación. Esta interfaz se ha integrado con cuatro motores de física de código abierto:

- ODE [4]
- Bullet [5]
- Simbody [27]
- DART [17]

Un modelo descrito en el Formato de descripción de simulación (SDF) que utiliza XML puede cargarse con cada uno de estos motores de física. Esto proporciona acceso a diferentes implementaciones de algoritmo y características de simulación.

3.2.2.4 Biblioteca de renderizado

- Dependencias: OGRE
- API externa: Permite la carga, inicialización y creación de escenas
- API interna: Almacena los metadatos de la visualización, llama a la API OGRE para renderizar.

---

[9] Es un método de serialización de datos estructurados, útil en el desarrollo de programas para comunicarse entre sí a través de un cable o para almacenar datos. Google desarrolló este protocolo para uso interno y ha proporcionado un generador de código para múltiples idiomas bajo una licencia de código abierto.

[10] Es una biblioteca C ++ multiplataforma, de código abierto y de acceso libre para la programación en red. Proporciona a los desarrolladores un modelo de Entrada/Salida asíncrono consistente utilizando un enfoque moderno de C ++

La biblioteca de renderizado usa OGRE para proporcionar una interfaz simple para renderizar escenas 3D tanto para la GUI como para las bibliotecas de sensores. Incluye iluminación, texturas y simulación del cielo. Es posible escribir complementos para el motor de renderizado.

3.2.2.5 Generación de sensores

- Dependencias: Rendering Library, Physics Library
- API externa: proporciona funciones para inicializar y ejecutar un conjunto de sensores
- API interna: TBD

La biblioteca de generación de sensores implementa todos los diversos tipos de sensores, escucha las actualizaciones del estado mundial desde un simulador de física y produce la salida especificada por los sensores instanciados.

3.2.2.6 GUI

- Dependencias: Rendering Library, Qt[11]
- API externa: ninguna
- API interna: ninguna

La biblioteca de GUI usa Qt para crear widgets gráficos e interactuar con la simulación. Es posible controlar el tiempo de la simulación mediante GUI. También se puede modificar la escena agregando, modificando o eliminando modelos. Además, hay algunas herramientas para visualizar y registrar datos de sensores simulados.

3.2.2.7 Plugins

Las bibliotecas físicas, de sensores y de reproducción admiten plugin, que permiten controlar o manipular un modelo o el entorno(mundo). Estos proporcionan a los usuarios acceso a las bibliotecas respectivas sin utilizar el sistema de comunicación.

3.2.3. Modelo SDF en Gazebo

En Gazebo, los modelos SDF definen una entidad física con propiedades dinámicas, cinemáticas y visuales. Pueden ir desde formas simples hasta robots complejos, los cuales pueden ser cargados o creados. Además, puede tener uno o más plugin, que afectan el comportamiento del modelo.

Gazebo cuenta con una base de datos para almacenar y mantener modelos disponibles para su uso dentro de la simulación. La base de datos modelo es un recurso respaldado por la comunidad. Los modelos pueden ser cargados, ya sea mediante programación o mediante la GUI.

Hace referencia a la etiqueta <modelo> en un archivo SDF [40] Es esencialmente una colección de links, joints, objetos de collision, visual y plugins.

[11] Qt es un marco de desarrollo de aplicaciones multiplataforma para escritorio, integrado y móvil. Las plataformas compatibles incluyen Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS y otros. Qt no es un lenguaje de programación en sí mismo. Es un marco escrito en C ++. Un preprocesador, el MOC (Meta-Object Compiler), se utiliza para extender el lenguaje C ++ con funciones como señales y ranuras. Antes del paso de compilación, el MOC analiza los archivos fuente escritos en C ++ extendido por Qt y genera fuentes C ++ compatibles con ellos. Por lo tanto, el marco en sí y las aplicaciones / bibliotecas que lo utilizan pueden ser compilados por cualquier compilador estándar de C ++.

3.2.3.1 Componentes de un Modelo

3.2.3.1.1 Links

Contiene las propiedades físicas del modelo. Cada link puede contener muchos elementos "collision" y "visual". Mas información en [41].

- **Collision**: Elemento que encapsula una geometría que se usa para verificar colisiones. Un link puede contener muchos elementos de colisión.
- **Visual**: Elemento que visualiza partes de un link. Pueden ser 0 o más en un link.
- **Inertial**: Este elemento describe las propiedades dinámicas del link, como masa y matriz de inercia rotacional.
- **Sensor**: Recopila datos del mundo para uso en plugins. Pueden existir 0 o más en un link.
- **Light**: Describe una fuente de luz conectada a un link. Pueden existir 0 o más en un link.

3.2.3.1.2 Joints

Un "joint" conecta dos links. Se establece una relación entre padres e hijos junto con otros parámetros, como el eje de rotación y los límites comunes. Mas información en [41].

3.2.3.1.3 Plugins

Un plugin es un fragmento de código que se compila como una biblioteca compartida y se inserta en la simulación. El complemento tiene acceso directo a toda la funcionalidad de Gazebo a través de las clases estándar de C ++. Los complementos son útiles porque permite a los desarrolladores controlar casi cualquier aspecto de Gazebo, además son rutinas autónomas que se comparten fácilmente, y se puede insertar y eliminar de un sistema en ejecución. Existen otros 5 tipos de plugin aparte del plugin para Modelo, estos son: para Mundo, Sensor, Sistema, Visual y GUI. Más información en [42].

3.2.3.2 Estructura base de datos de un modelo

La base de datos de un modelo debe cumplir con un directorio y estructura de archivos especifica. La raíz de una base de datos modelo contiene un directorio por cada modelo, y, un archivo "*database.config*" con información sobre la base de datos del modelo. Cada directorio de modelo también tiene un archivo "*model.config*" que contiene metadatos sobre el modelo. Un directorio modelo también contiene el modelo SDF, materiales, meshes y plugins. Más información en [43].

La estructura es la siguiente:

- Base de datos
    - database.config: metadatos sobre la base de datos. Esto se completa automáticamente desde CMakeLists.txt
    - modelo_ejemplo: un directorio para modelo_ejemplo
        - model.config: metadatos sobre modelo_ejemplo.
        - model.sdf: descripción del modelo SDF.
        - meshes: directorio para los archivos COLLADA y STL.
        - materiales: directorio que contiene subdirectorios de texturas y scripts
            - texturas: un directorio para archivos de imagen (jpg, png, etc.).
            - scripts: un directorio para scripts de material OGRE
        - plugins: un directorio para fuente de plugins y archivos header.

3.2.4. Conexión con ROS

Para lograr la integración de ROS con Gazebo, ROS ofrece un package llamado gazebo_ros_pkgs. Este proporciona las interfaces necesarias para simular un robot en Gazebo usando mensajes ROS, servicios y reconfiguración dinámica. Más información en [44]. Algunas características son:

- Construye con CATKIN[12] [45].
- Trata URDF[13] [46] [47]y SDF de la manera más equitativa posible.
- Mejora el soporte para controladores que usan ros_control.
- Eficiencia del controlador en tiempo real del DARPA Robotics Challenge.
- Controlar modelos en Gazebo que definan un plugin en su código SDF a través de ROS.
- Lectura de sensores del modelo de Gazebo a través de ROS.
- Comunicación del entorno de Gazebo con ROS, con la posibilidad de manipulación.
- Lanzar modelos o mundos de Gazebo desde ROS.
- Controlar el modelo (sin usar plugin) mediante ros_control, si el modelo es URDF.

Para ejecutar una versión específica de ROS y usar los package gazebo, se debe considerar lo siguiente (en caso contrario es necesario instalar otros package para realizar conexión):

- ROS Lunar para la versión 7.x de Gazebo.
- ROS Kinetic para la versión 7.x de Gazebo.
- ROS Indigo para la versión 2.x de Gazebo.

3.2.5. Desarrollando propios "packages" Gazebo_ROS

Para generar robots que puedan usar ROS con Gazebo se debe seguir estándares de jerarquía de archivos. Según estándares, lo relacionado con el modelo debe estar en un package llamado "/*nombre_del_robot*_description" y los archivos de lanzamiento y del mundo deben estar en un package llamado /*nombre_del_robot* _gazebo. Con estos la jerarquía debe ser como muestra la Tabla 2.1:

Tabla 3. 1 Jerarquia de Carpetas para generar packages ROS

```
…/nombre_del_work_space/src
        /nombre_del_robot_description
                package.xml
                CMakeList.txt
                /urdf
                        nombre_del_robot.urdf
                /meshes
                        mesh1.dae, mesh2.dae, …
                /materials
        /nombre_del_robot_gazebo
                /launch
                        nombre_del_robot.launch
                /worlds
                        nombre_del_robot.world
              /models
                      world_object1.dae, …
                /materials
                /plugins
```

[12] Catkin es el sistema de construcción oficial de ROS y el sucesor rosbuild. Combina macros CMake y scripts de Python para proporcionar funcionalidad sobre el flujo de trabajo de CMake. El flujo de trabajo de catkin es similar al de CMake, pero agrega soporte a la infraestructura de 'encontrar paquete' y construir proyectos dependientes al mismo tiempo.

[13] El Universal Robotic Description Format (URDF) es un formato de archivo XML utilizado en ROS para describir todos los elementos de un robot. Para usar un archivo URDF en Gazebo, se deben agregar algunas etiquetas específicas de simulación adicionales para que funcionen correctamente con Gazebo..

3.3. ¿Qué es ROS? (Robotic Operating System)

El Robot Operating System (ROS) es un sistema meta-operativo de fuente abierta para robots. Proporciona los servicios que esperaría de un sistema operativo, incluida la abstracción de hardware, el control de dispositivos de bajo nivel, la implementación de funciones de uso común, el paso de “messages” entre procesos y la administración de “packages”. También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en múltiples computadoras. Es una colección de herramientas, bibliotecas y convenciones para escribir software de robots que tienen como objetivo simplificar la tarea de crear un comportamiento de robot complejo y robusto en una amplia variedad de plataformas robóticas.

El “runtime” (tiempo de ejecución) de ROS es una red de procesos punto a punto (potencialmente distribuida entre máquinas) que se acoplan de forma flexible utilizando la infraestructura de comunicación ROS. ROS implementa varios estilos diferentes de comunicación, incluida la comunicación sincrónica estilo RPC[14] sobre los “services”, la transmisión asincrónica de datos sobre “topics” y el almacenamiento de datos en un “parameter server” [48].

3.3.1. Conceptos de ROS

ROS tiene tres niveles de conceptos y dos tipos de nombres que se analizan a continuación. Mas información en [49].

3.3.1.1 Nivel Sistema de archivos ROS – ROS Filesystem level

Se refiere principalmente a los recursos de ROS que encuentren almacenados, tales como:

- **Packages**: (paquetes) son la unidad principal para organizar ROS. Puede contener “nodes”, bibliotecas dependientes de ROS, conjunto de datos, archivos de configuración o cualquier otra cosa que se organice de manera útil. Son el elemento de construcción y de lanzamiento más atómico en ROS. Lo que significa que lo más granular que puedes crear y lanzar es un paquete.
- **Metapackages**: Son paquetes especializados que solo sirven para representar un grupo de otros “packages” relacionados.
- **Package Manifests**: (package.xml) proporcionan “metadata” (metadatos) sobre un paquete, como nombre, versión, descripción, información de licencia, dependencias.
- **Repositories**: Es una colección de “packages” que comparten un sistema VCS común[15]. Los “packages” que comparten un VCS tienen la misma versión y se pueden lanzar juntos usando el “bloom”[16] de la herramienta de automatización de lanzamiento de “CATKIN”.
- **Message (msg) type**: (tipo de mensaje) Definen las estructuras de datos para los mensajes enviados en ROS. Se almacenan en “my_package/msg /MyMessageType.msg.
- **Service (srv) type**: (tipo de servicio) Definen la estructura de datos de solicitud y respuesta para servicios ROS. Almacenados en “my_package/srv/MyServiceType.srv”.

---

[14] RPC, Remote Procedure Call, (llamada a procedimiento remoto) consiste en un protocolo que permite a un software o programa ejecutar código en otra máquina remota sin preocuparse por la comunicación, por lo regular es bastante utilizado en el paradigma cliente y servidor.

[15] VCS, versión control system (sistema de control de versiones) es la administración y actualización de cambios en el programa Los cambios generalmente se identifican por un número o código, denominado "número de revisión” asociada con una marca de tiempo y la persona que realiza el cambio. Las revisiones se pueden comparar, restaurar y combinar con algunos tipos de archivos.

[16] Bloom es una herramienta de automatización de lanzamiento, diseñada para facilitar la generación de lanzamiento específicos de la plataforma desde proyectos fuente. Bloom está diseñado para funcionar mejor con proyectos catkin, pero también puede acomodar otros tipos de proyectos.

3.3.1.2 Nivel de gráficos de computación ROS – ROS Computation Graph Level

Es la red de los procesos de datos ROS. Los conceptos básicos que proporcionan datos de diferentes maneras son:

- **Nodes**: (Nodos) Procesos que realizan cálculos. Un sistema de control de robot generalmente comprende muchos "nodes" para las diferentes tareas que debe realizar. Se escribe con biblioteca cliente ROS, como **roscpp** o **rospy**.
- **Master**: Proporciona registro de nombre y búsqueda para el resto del gráfico de computación. Sin el "master", los "nodes" no podrían encontrarse, intercambiar "messages" o invocar "services".
- **Parameter Server**: (Servidor de parámetros) Permite que los datos se almacenen por clave en una ubicación central. Es parte del Master.
- **Messages**: (Mensajes) Los nodos se comunican entre sí por messages. Un mensaje es simplemente una estructura de datos, que comprende campos tipados. Los tipos primitivos estándar (entero, punto flotante, booleano, etc.), y las matrices de tipos primitivos son compatibles.
- **Topics**: (Temas) es un nombre que se usa para identificar el contenido del "message". Un "node" puede enviar un mensaje publicando un "topic" determinado, o bien, si está interesado en ciertos datos se suscribirá al "topic" apropiado. Los "messages" se envían a través de un sistema de transporte con semántica publicar/suscribir (publish/subscribe).
- **Services**: (Servicio) el modelo de publicar/suscribir es un paradigma de comunicación muy flexible, pero su transporte unidireccional de muchos a muchos no es apropiado para las interacciones de solicitud / respuesta (request/reply), que a menudo se requieren en un sistema distribuido. Para esto se utilizan los services, que define estructuras de mensaje para "request" y "reply". Un "node" proveedor ofrece un "service" con un nombre y un cliente utiliza el "service" enviando el mensaje de "request" y esperando "reply".
- **Bags**: Es un formato para guardar y reproducir datos de "message" ROS. Es un mecanismo importante para almacenar datos, como de sensores, que pueden ser difíciles de recopilar pero que son necesarios para desarrollar y probar algoritmos.

El ROS "master" actúa como un servicio de nombres en el Gráfico de Computación ROS. Almacena información de registro de "topics" y "services" para "nodes". Los "nodes" se comunican con "master" para reportar su información de registro. Como estos "nodes" se comunican con el "master", pueden recibir información sobre otros "nodes" registrados y realizar las conexiones, según corresponda. El "master" también realizará devoluciones de llamada a estos "nodes" cuando cambie esta información de registro, lo que permite a los "nodes" crear conexiones dinámicamente a medida que se ejecutan nuevos "nodes".

Los "nodes" se conectan a otros nodos directamente; el "master" solo proporciona información de búsqueda. Los "nodes" que suscriben a un "topic" solicitan conexión de los "nodes" que publican ese "topic", y establecerán conexión mediante un protocolo de conexión acordado. El protocolo más utilizado en ROS es TCPROS, Figura 2.1, que usa conectores estándar TCP / IP.

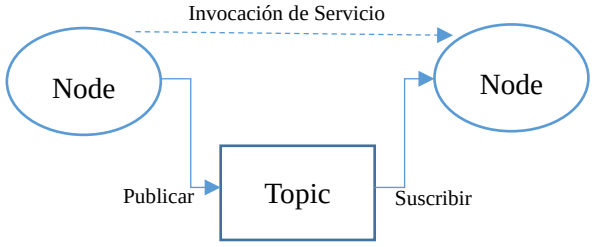


Figura 3. 1 Relación protocolo TCPROS entre nodos a través de topics.

3.3.1.3 Nivel de la comunidad ROS – ROS Community Level

Son recursos que permiten intercambiar software y conocimiento. Estos recursos incluyen:

- **Distributions**: (distribuciones) Son colecciones de "stacks"[17] versionadas que se puede instalar. Juegan un papel similar a las distribuciones de Linux: facilitan la instalación de software y también mantienen versiones consistentes en un conjunto de estos.
- **Repositories**: (Repositorios) ROS depende de una red federada de repositorios de códigos, donde diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software de robots.
- **ROS Wiki**: Foro para documentar información sobre ROS. Cualquier persona puede registrarse y contribuir documentación, correcciones o actualizaciones, tutoriales y más.
- **Bug ticket System**: Sistema de seguimiento de problemas.
- **Mailing List**: (Lista de Correos) Principal canal de comunicación sobre actualizaciones de ROS, así como foro para hacer preguntas sobre el software ROS.
- **ROS Answers**: un sitio de preguntas y respuestas relacionadas con ROS.
- **Blog**: El blog ros.org proporciona actualizaciones periódicas, que incluyen fotos y videos.

3.3.1.4 Nombres - Name

3.3.1.4.1 Nombres de recursos gráficos – Graph Resource Name

Proporcionan una estructura jerárquica de nombres que se utiliza para todos los recursos en un gráfico de computación ROS, como "nodes", "parameters", "topics" y "services". Estos nombres son muy poderosos en ROS y son fundamentales para la forma en que se componen los sistemas más grandes y complicados en ROS, por lo que es fundamental comprender cómo funcionan estos nombres y cómo puede manipularlos.

Son un mecanismo importante en ROS para proporcionar encapsulación. Cada recurso se define dentro de un "namespaces", que puede compartir con muchos otros recursos. En general, los recursos pueden crear recursos dentro de su "namespaces" y pueden acceder a recursos dentro o sobre su propio "namespaces". Se pueden establecer conexiones entre recursos en "namespaces" distintos, pero esto generalmente se hace mediante el código de integración que se encuentra sobre ambos "namespaces". Esta encapsulación aísla diferentes partes del sistema en caso de accidentalmente agarrar el recurso incorrecto con nombre o nombres de secuestro global.

Los nombres se resuelven relativamente, por lo que los recursos no necesitan saber en qué espacio de nombres se encuentran. Esto simplifica la programación ya que los nodos que funcionan juntos se pueden escribir como si estuvieran todos en el espacio de nombres de nivel superior. Cuando estos Nodos se integran en un sistema más grande, se pueden insertar en un espacio de nombres que define su colección de códigos.

3.3.1.4.2 Nombres de recursos del paquete – Package Resource Names

Se utilizan en ROS con conceptos de nivel de sistema de archivos para simplificar el proceso de referencia a archivos y tipos de datos en el disco. Los nombres de recursos de paquetes son muy simples: son solo el nombre del "package" en el que se encuentra el recurso más el nombre del recurso. Por ejemplo, el nombre "std_msgs / String" hace referencia al tipo de mensaje "String" en el "package" "std_msgs".

[17] Stacks (pilas) simplifican el proceso de intercambio de códigos. Los packages estan organizados en pilas. Las pilas son el mecanismo principal en ROS para distribuir software, tienen una versión asociada y puede declarar dependencias en otras pilas. Las dependencias declaran un número de versión, que proporciona una mayor estabilidad en el desarrollo.

Algunos de los archivos relacionados con ROS a los que se puede hacer referencia utilizando nombres de recursos de paquete incluyen:

- **Tipos de mensajes (msg)**
- **Tipos de servicio (srv)**
- **Tipos de nodo**

Los nombres de recursos de paquetes son muy similares a las rutas de archivos, excepto que son mucho más cortos. Esto se debe a la capacidad de ROS para localizar "package" en el disco y hacer suposiciones adicionales sobre sus contenidos.

3.3.2. Conceptos de nivel superior

La plataforma central de ROS intenta ser tan independiente de la arquitectura como sea posible. Proporciona varios modos diferentes de comunicación de datos (topics, services, parameter service), pero no establece cómo se usan ni cómo se nombran. Este enfoque permite que ROS se integre fácilmente con una variedad de arquitecturas, pero los conceptos de nivel superior son necesarios para construir sistemas más grandes sobre ROS. Más información en [50]

3.3.2.1 Marcos coordinados/ transfmormaciones - Coordinate Frames/transforms

El **tf package** proporciona un marco distribuido basado en ROS para calcular las posiciones de múltiples coordenadas a lo largo del tiempo. Esta es una herramienta muy útil y necesaria para desarrollar una simulación eficiente en la que se deseen probar algoritmos robóticos.

3.3.2.2 Acciones / Tareas – Actions/Tasks

El "package" **actionlib** define interfaz común basada en "topics" para tareas prioritarias en ROS.

3.3.2.3 Ontología del mensaje – Message Ontology

Base para sistemas robóticos. Define varias clases de mensajes, que incluyen:

- **actionlib_msgs**: mensajes para representar acciones.
- **diagnostic_msgs**: mensajes para enviar datos de diagnóstico.
- **geometry_msgs**: mensajes para representar primitivas geométricas comunes.
- **nav_msgs**: mensajes para navegación.
- **sensor_msgs**: mensajes para representar datos del sensor.

3.3.2.4 Plugins

pluginlib proporciona una biblioteca para cargar dinámicamente bibliotecas en código C ++.

3.3.2.5 Filtros - Filters

El paquete de filtros proporciona una biblioteca C ++ para procesar datos usando una secuencia de filtros.

3.3.2.6 Modelo de Robot

El **urdf package** define un formato XML para representar un modelo de robot y proporciona un analizador de C ++.

3.3.3. Librerias de Clientes – Client Libraries

Una biblioteca cliente ROS es una colección de código que facilita el trabajo del programador ROS. Toma muchos de los conceptos ROS y los hace accesibles a través de código. En general, estas bibliotecas le permiten escribir nodos ROS, publicar y suscribirse a temas, escribir y llamar a servicios, y usar el Servidor de parámetros. Dicha biblioteca puede implementarse en cualquier lenguaje de programación, aunque el enfoque actual está en brindar soporte robusto en C ++ y Python. Más información en [51].

3.3.3.1 Bibliotecas principales del cliente

- **roscpp**: roscpp es una biblioteca cliente C ++ para ROS. Es la biblioteca cliente ROS más utilizada y está diseñada para ser la biblioteca de alto rendimiento para ROS.
- **rospy**: rospy es la biblioteca de cliente de Python pura para ROS y está diseñada para proporcionar las ventajas de un lenguaje de scripts orientado a objetos a ROS. El diseño de rospy favorece la velocidad de implementación (es decir, el tiempo del desarrollador) sobre el rendimiento del tiempo de ejecución, de modo que los algoritmos pueden ser rápidamente prototipados y probados dentro de ROS. También es ideal para código de ruta no crítica, como código de configuración e inicialización. Muchas de las herramientas de ROS están escritas en rospy para aprovechar las capacidades de tipo de introspección. Las herramientas ROS Master, roslaunch y otras herramientas ros se desarrollan en rospy, por lo que Python es una dependencia central de ROS.
- **roslisp**: es una biblioteca cliente para LISP y actualmente se está utilizando para el desarrollo de bibliotecas de planificación. Admite la creación de nodos independientes y el uso interactivo en un sistema ROS en ejecución.

3.3.3.2 Bibliotecas de clientes experimentales

- **roscs** es una biblioteca cliente para Mono / .NET. Puede ser utilizado por cualquier lenguaje Mono / .NET, incluyendo C#, Iron Python, Iron Ruby, etc. El sistema de compilación ROS creará archivos .DLL y .so para cada paquete escrito en roscs.
- **roseus**: es una biblioteca cliente para el lenguaje EusLisp.
- **rosgo**: implementación pura en Go
- **roshask** es la biblioteca cliente de Haskell.
- **rosjava**: es una implementación de ROS en Java puro con soporte Android.
- **rosnodejs**: es una biblioteca cliente ROS Javascript nativa para Node.js que proporciona un medio directo de comunicación con un ROS Master en red.
- **RobotOS.jl** es un paquete nativo de Julia que se envuelve en rosa.
- **roslua**: Biblioteca cliente para Lua. lenguaje de scripting liviano pero potente e integrable. La biblioteca del cliente se encuentra en etapa de desarrollo experimental y activa.
- **PhaROS** es una biblioteca cliente bajo licencia MIT para el lenguaje Pharo free Smalltalk.
- **rosR**: es una extensión de lenguaje ROS para el lenguaje de programación estadística R.
- **rosruby**: soporte para ruby
- **Unreal-Ros-Plugin**: un complemento irreal usa rosbridge para recibir / enviar topics ros

3.4. Distribuciones de ROS

Una distribución ROS es un conjunto versionado de paquetes ROS. Estos son similares a las distribuciones de Linux (por ejemplo, Ubuntu). El propósito de las distribuciones de ROS es permitir que los desarrolladores trabajen en una base de código relativamente estable hasta que estén listos para él lanzamiento. Por lo tanto, una vez que se lanza una distribución,

La lista de distribución es larga y puede apreciarse en [52], como también para diferentes sistemas operativos [53]. A continuación, se presentan Tabla 2.2 con las distribuciones lanzadas en los últimos años:

Tabla 3. 2 Distribuciones de ROS lanzadas los últimos años

Distribución	Fecha de realización	Fecha termino de vida
ROS Melodic Morenia	Mayo, 2018	Mayo, 2023
Ros Lunar Loggerhead	Mayo, 23, 2017	Mayo, 2019
ROS Kinetic Kame (Recomendada)	Mayo, 23, 2016	Abril, 2021
ROS Jade Turtle	Mayo, 23, 2015	Mayo, 2017
ROS Indigo Igloo	Julio, 22, 2014	Abril, 2019

4. DESCRIPCIÓN DE LA PLATAFORMA SIAR

4.1. Introducción

El proyecto SIAR desarrolla un robot de terrestre completamente autónomo capaz de navegar e inspeccionar el sistema de alcantarillado con una mínima intervención humana, y con la posibilidad de controlar manualmente los movimientos del vehículo.

EL robot SIAR es una plataforma de seis ruedas con la capacidad de cambiar el ancho entre sus ruedas para abordar las diferentes secciones y desafíos de la red de alcantarillado.

El Consorcio encargado en desarrollar este proyecto está compuesto por una PYME IDMind (IDM) de origen portugués [54], Universidad de Sevilla (USE) y Universidad Pablo de Olavide (UPO).

La versión de la plataforma considerada para desarrollar la simulación corresponde a la V3, y esta fue finalizada a fines de agosto del año 2017.

La información utilizada para desarrollar este capítulo se obtiene del “Deliverable SIAR_D28.6” [55], documento oficial del proyecto.

4.2. Características principales de la plataforma

Las principales características de la plataforma a tener en consideración para el desarrollo del simulador son:

- 6 ruedas con 6 motores de locomoción independientes
- 1 motor lineal para controlar el ancho
- Peso: 58 Kg
- Dimensiones con ancho máximo (Alto x Ancho Max. x Largo): 44 x 70 x 84 cm
- Dimensiones con ancho mínimo (Alto x Ancho Min. x Largo): 44 x 50 x 98 cm
- Velocidad máxima 0.75 m/s
- Cámaras RGBD
- Cámaras de profundidad

4.3. Diseño mecánico de la plataforma

A continuación, se realiza una descripción del diseño mecánico de la plataforma SIAR. Para esto se presenta la Figura 3.1 como referencia, la cual permite dimensionar y tener una idea de las características de la estructura.

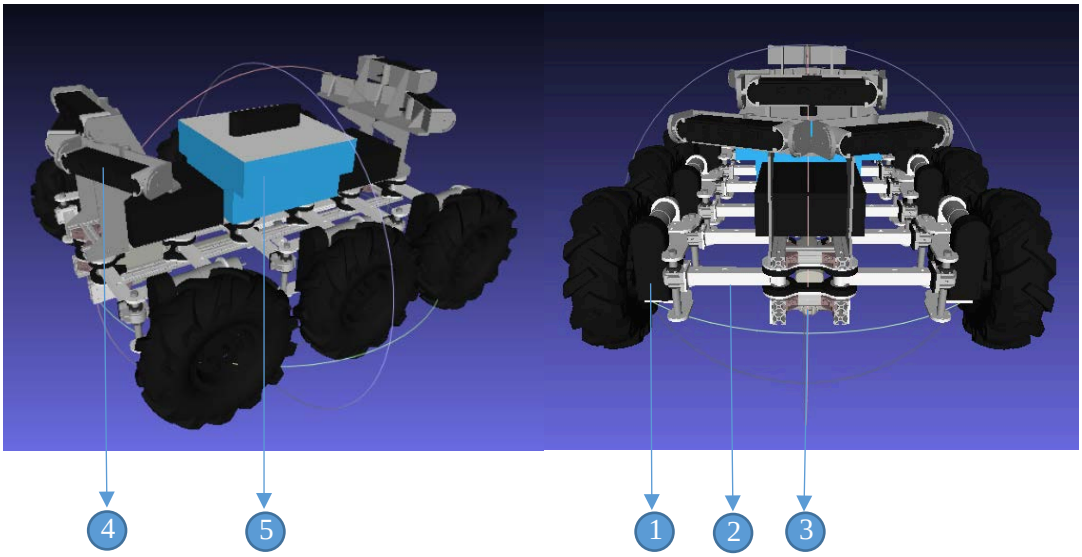


Figura 4. 1 Diseño 3D de la plataforma SIAR

1. El sistema de tracción de las ruedas es de tipo diferencial y está compuesto por 6 motores independientes con caja de cambios de 90º y ruedas a todo terreno. Cada uno de estos se conecta a un par brazo de suspensión independiente que une al marco central del robot.
2. El robot SIAR posee un mecanismo de ajuste de ancho. Este aumenta la adaptabilidad del robot a la configuración del alcantarillado. Es accionado por un motor lineal que genera el movimiento de los brazos.
3. El marco estructural central conecta el sistema de tracción de la rueda motorizada, el mecanismo de ajuste del ancho y para transportar el equipo de carga y la electrónica.
4. En el marco central se encuentran siete cámaras, las que se montan en soportes enlazados al marco, estos mantienen las cámaras a la altura e inclinaciones deseado en el proyecto.
5. Cuenta con una caja de fibra de vidrio para proteger los componentes electrónicos de las aguas sucias y los desechos

4.3.1. Mecanismo de ajuste de ancho

El mecanismo de ajuste de ancho, como se mencionó, mejora la adaptabilidad del robot a diferentes cambios estructurales en el entorno de trabajo. Este permite variar el ancho del robot mediante un cambio en la posición de las ruedas, lo cual se acciona mediante el uso de un motor lineal. El motor lineal está vinculado a la parte inferior del marco y al mismo tiempo a una guía lineal Figura 3.2 – i. El motor hace que un pistón se mueva a lo largo de la guía, Figura 3.2 - ii. Barras metálicas, Figura 3. 2 – iii, transmiten el movimiento del motor a los brazos conectados al marco. Una pareja de brazos está conectada a cada una de las seis ruedas del robot. El movimiento, rotación, de los brazos genera la variación del ancho del robot. La conexión de los brazos al marco y a las ruedas genera un paralelogramo. El paralelogramo asegura que las ruedas siempre se mantienen paralelas al robot. Finalmente, dos de las barras, metálicas transmiten el movimiento del pistón a los brazos centrales del robot (uno a cada lado). Luego, a cada lado del robot, los brazos se conectan entre sí mediante varillas metálicas, lo que impone un movimiento sincrónico de las seis ruedas del robot. De esta forma, las configuraciones del ancho robot puede variar de 500 mm (mínimo) a 700 mm (máximo). La variación de ancho implica una variación de longitud de la plataforma. Para un ancho mínimo de 500 mm, el robot tiene una longitud máxima de 980 mm. Para un ancho máximo de 700 mm, el robot tiene una longitud mínima de 845 mm. La altura del robot es independiente del ancho del robot.

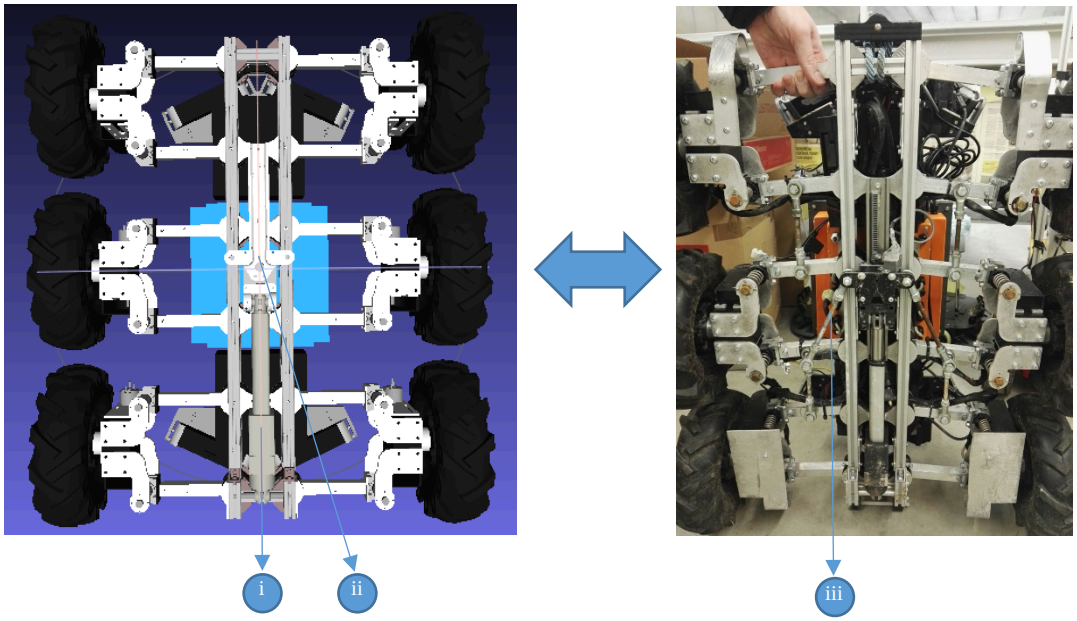


Figura 4. 2 Descripción del mecanismo de ajusto de la plataforma SIAR

4.4. Sensores a bordo del SIAR

Dentro de la plataforma SIAR, encontramos diferentes sensores que están dedicados a satisfacer la realización de distintas tareas. Ellos los podemos dividir en dos grupos:

- Sensores de navegación:
  - Codificadores motores: para controlar la velocidad de los de tracción
  - Codificador plataforma: para controlar la velocidad del actuador de ancho.
  - Potenciómetro plataforma: para controlar la posición del actuador de ancho.
  - Sensor de inercia: para estimar orientación del robot.
  - Cámaras de profundidad: para estimar distancias de obstáculos y/o paredes al robot. También para refinar las estimaciones de odometría.
- Sensores de percepción:
  - Cámaras RGBD: para analizar el entorno del alcantarillado y detectar cualquier situación anormal.

4.4.1. Ubicación de los sensores

A continuación, se describe la posición de cada uno de los sensores mencionados anteriormente, con el fin de ser considerados para la simulación.

- Sensores de navegación:
  - Codificadores motores: ubicado cada uno instalado en cada motor
  - Codificadores y potenciómetro plataforma: instalados en motor lineal.
  - Sensor de inercia: ubicado en el centro de rotación.
  - 4 cámaras de profundidad: ubicadas dos cámaras una al lado de la otra en la parte frontal inclinada 45º hacia abajo y dos cámaras una al lado de la otra en una inclinación de 45º hacia abajo.
  - 2 cámaras de profundidad: ubicadas en la parte delantera inclinada 5-10º hacia arriba y una en la parte posterior inclinada 5-10º hacia arriba.
- Sensores de percepción

- 4 cámaras de profundidad: ubicadas dos cámaras una al lado de la otra en la parte delantera inclinada 45º hacia abajo y dos cámaras una al lado de la otra en una inclinación de 45º hacia abajo.
- 2 cámaras de profundidad: ubicadas en la parte delantera inclinada 5-10º hacia arriba y una en la parte posterior inclinada 5-10º hacia arriba.
- Cámaras de profundidad: Ubicada en la parte superior del robot apuntando hacia arriba.

4.5. Actuadores a bordo del SIAR

El robot está equipado con actuadores para la locomoción, y para accionar el mecanismo de ajuste del ancho.

- Actuadores de Locomoción
  - 6 motores en las ruedas
  - Motor lineal para cambiar el ancho del robot.
  - 2 proyectores LED (uno en la parte delantera y trasera del robot).

4.6. Filosofía de control manual del SIAR

Dentro de todas las características que tiene la plataforma SIAR, se destaca en este apartado la del control manual, ya que es esta tarea la considerada a simular inicialmente para homologar y corroborar el comportamiento del robot real en Gazebo.

La filosofía de control manual utilizada, a rangos generales, por la plataforma SIAR es de lazo abierto y se explica mediante el esquema de la Figura 3.3.

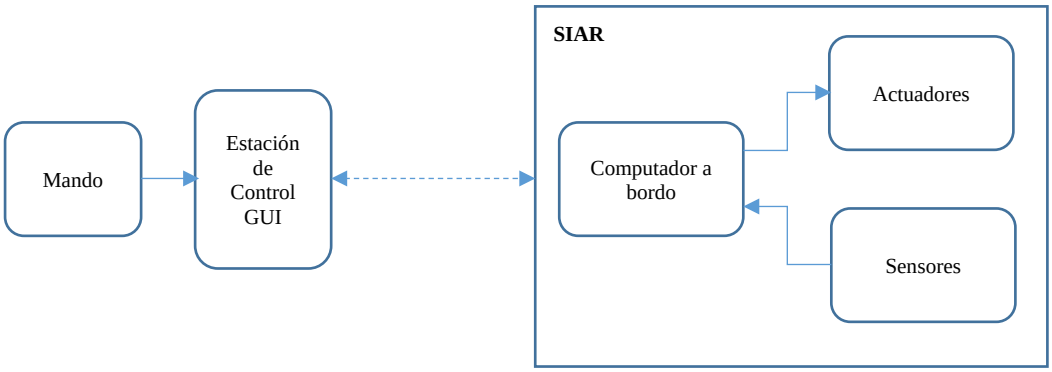


Figura 4. 3 Filosofía de control de lazo abierto de plataforma SIAR

De esta forma, se puede aprecia que la idea general de la filosofía de control consiste en generar, con el mando, información que es enviada a la estación de control, la cuales envían posteriormente al computador abordo del SIAR, una orden que se ejecuta en los actuadores. Este es un control a lazo abierto a pesar de que el computador reciba información desde los sensores.

4.7. Consideraciones para la simulación

Como se ha mencionado anteriormente, la plataforma SIAR fue desarrollada para inspeccionar alcantarillados, razón por la cual se debe considerar en la simulación la incorporación de un túnel con características reales a las que tiene el entorno en el que desenvuelve la plataforma.

Para esto se considera los datos obtenidos de la reconstrucción del mapa de alcantarillado

desarrollado en etapas anteriores del proyecto, donde se obtuvo como resultado el modelado digital del entorno con gran satisfacción. De esta manera, el simulador debe desarrollarse teniendo en consideración que deberían obtenerse una representación robot-entorno similar a la presentada en la Figura 3.4

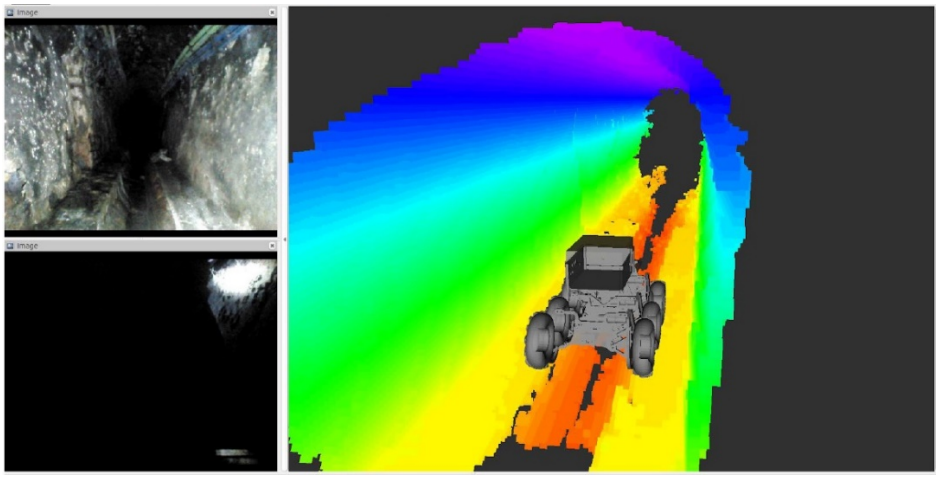


Figura 4. 4 Representación robot-entorno de la plataforma SIAR con alcantarillado a considerar para realizar el simulador

5. DESARROLLO DEL SIMULADOR

5.1. Introducción

En el presente capítulo se detalla la metodología y el procedimiento utilizado para desarrollar el simulador que representa a la plataforma SIAR. Para desarrollar esto se toman en consideración las características de la plataforma mencionadas en el capítulo anterior, descritas en la Figura 5.1, y las cuales fueron imitadas en simulación utilizando herramientas de Gazebo y ROS.

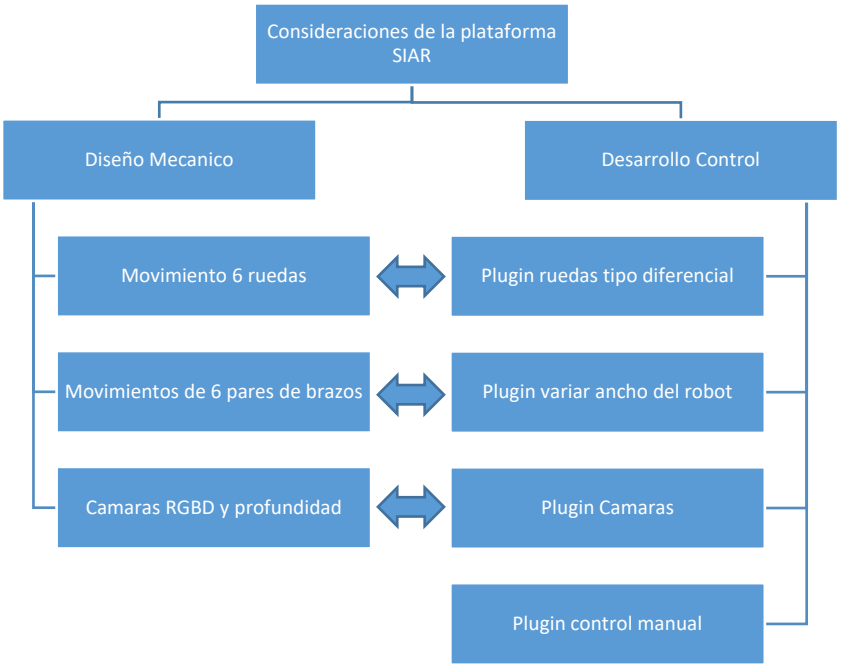


Figura 5. 1 Caracteristicas principales de la plataforma SIAR ha ser simuladas.

5.2. Primeros Pasos

5.2.1. Primer Modelo

El primer paso llevado a cabo para realizar la simulación de la plataforma SIAR consistió en modelar un robot simple con características generales, es decir, las seis ruedas y seis brazos. De esta manera, probar y conocer la reacción de un modelo con estas características con los motores físicos de Gazebo, especialmente lo relacionado con la inercia, roce y fricción, ya que la manipulación de estas variables tiene gran importancia para obtener una simulación cercana a la realidad. En consecuencia, se consideraron las siguientes características del modelo:

- 1 estructura central
- 6 brazos conectados a la estructura central
- 6 ruedas conectadas a los brazos
- Ancho real de las ruedas: 8 cm
- Ancho máximo real del robot: 70 cm
- Masa real de la plataforma: 58 kg
- Masa real rueda: 1 kg

- Masa real brazo: 1kg

Cabe destacar que, según lo mencionado en el Capítulo III, los conceptos generales más relevantes que se deben considerar en Gazebo para desarrollar un modelo son:

- Links
  - Collision
  - Visual
  - Inertial
- Joints
- Plugins

De esta manera, para describir cada una de las partes del modelo en Gazebo, se escribió un archivo formato SDF (también pudieron ser utilizadas las herramientas que ofrece la GUI de Gazebo), dado que este formato es conveniente, practico y preciso para intervenir un código en comparación a la GUI. Se siguió la estructura que ofrece [56] como muestra la Figura 5.2. El código completo de este primer modelo se puede obtenerse en [57].

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="box">
    <pose>0 0 0.5 0 0 0</pose>
    <static>false</static>

    <link name="link">
      ...
    </link>

    <joint type="revolute" name="my_joint">
      ...
    </joint>

    <plugin filename="libMyPlugin.so" name="my_plugin"/>

  </model>
</sdf>
```

Figura 5. 2 Estructura general formato SDF para la descripción de modelos.

La idea general para la construcción del modelo se basa en el siguiente diagrama de la Figura 5.3:

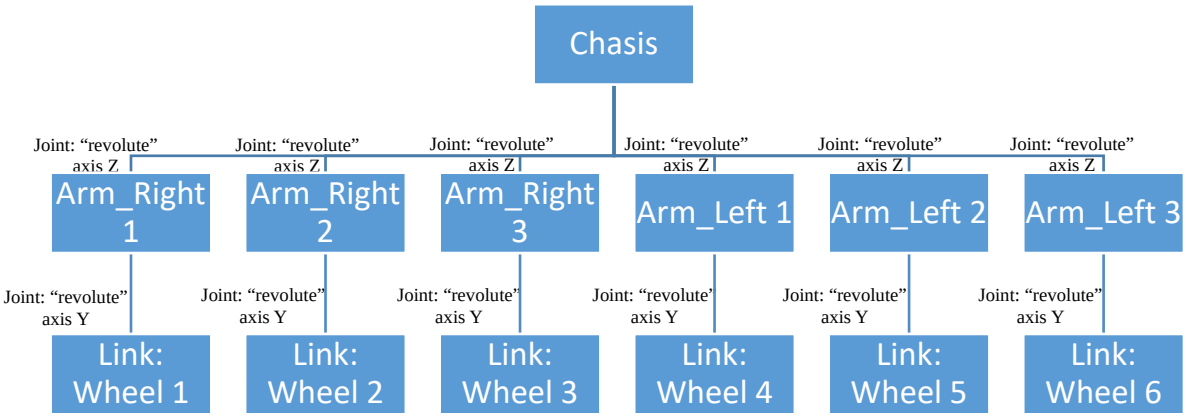


Figura 5. 3 Arquitectura general de la estructura del primer modelo desarrollado en Gazebo

El diseño de lo simulado se observa en la Figura 5.4

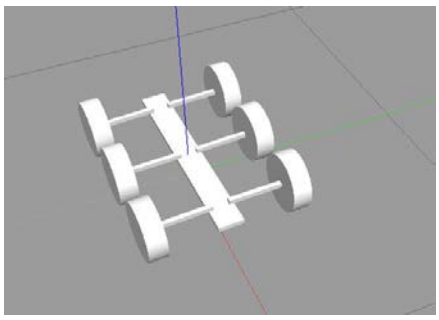


Figura 5. 4 Vista del primer modelo generado en Gazebo

Para comprobar el desempeño del modelo, se realizaron pruebas utilizando la opción del panel derecho de la GUI “Control de Joint” que ofrece Gazebo, con el cual es posible manipular fuerza posición y velocidad. Las pruebas realizadas fueron:

1. Visualizar la inercia del modelo y sus partes, la cual debe aproximarse a la forma de los links o más pequeña para obtener una correcta interacción del modelo y el motor físico.
2. Mover brazos del robot para observar comportamiento del modelo sin anormalidades o dentro de la realizar.
3. Desplazar el robot dentro del plano para observar comportamiento general de la estructura, como fricción, movimientos de desplazamiento y frenado real.

Cabe destacar que para realizar la estimación de las inercias se consideró lo planteado en las referencias de [58], las cuales entregan la información suficiente para calcular el valor de las matrices de inercia (I_{xx}, I_{yy}, $I_{zz,}$ para ejes X, Y y Z respectivamente) en cuerpos cilíndricos y paralelepípedos, y las restricciones que se deben tener en consideración las cuales principalmente responden a lo siguiente:

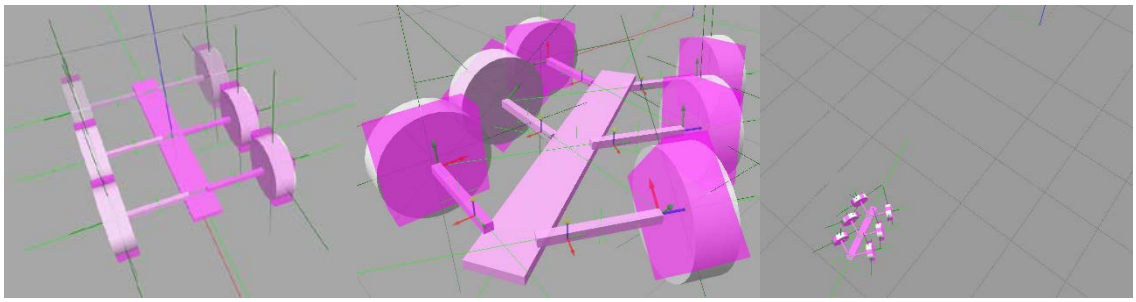
$$I_{yy} + I_{zz} - I_{xx} \geq 0 \tag{5.1}$$

$$I_{zz} + I_{xx} - I_{yy} \geq 0 \tag{5.2}$$

$$I_{xx} + I_{yy} - I_{zz} \geq 0 \tag{5.3}$$

Los resultados obtenidos fueron:

1. La inercia del modelo está caracterizada al tamaño de cada una de las partes que componen el modelo. El modelo se visualizó estable y sin ninguna anormalidad al someterse a las fuerzas que aplican los motores físicos, Figura 5.5.a
2. Los brazos se mueven sin ocasionar anormalidad en el modelo al aplicar una pequeña fuerza de rotación en el joint por medio de la GUI. Cabe destacar que el movimiento de las ruedas no es paralelo a la estructura como lo hace el robot real, 5.5.b
3. El modelo se desplaza con inconveniente por el escenario, ya que, al desplazarse también lo hacen los brazos, frenando el desplazamiento, razón por la cual tampoco es posible realizar movimientos diferenciales. Además, previo a que los brazos se muevan, si se desea frenar el modelo desliza, Figura 5.5.c.
4. En general el modelo simulado debe ser mejorados para acercarse a un comportamiento similar al del robot real.



(a) (b) (c)

Figura 5. 5 Visualización de pruebas comportamiento de inercia (a), movimiento de brazos (b) y y desplazamiento (c), realizadas al primer modelo

5.2.2. Segundo Modelo

Teniendo en cuenta que el modelo anterior se desempeña con inconvenientes, se prosiguió a mejorar este con el fin de obtener un comportamiento y diseño similar al real. Para esto se consideraron las siguientes características:

- 1 estructura central, con dimensiones y masa aprox. a la real: dos barras de 7 kg y 11 cm de largo y dos barras de 12 kg y 79,6 cm de largo, todas las barras de 2 cm de alto y ancho, en consecuencia, se tiene una estructura que forma un perímetro de 11 cm de ancho y 79,6 cm de largo con una masa de 38 kg.
- 6 pares de brazos conectados a la estructura central, con dimensiones y masa aproximados a la real: 17x2x2 cm cada uno y 1kg el par.
- 6 acoples con ejes conectados a cada par de brazos (para simular motor de las ruedas del sistema de tracción diferencial): 1,2 kg cada uno.
- 6 ruedas conectadas a los acoples, con dimensiones y masa aproximados a la real: radio 12,5 cm, ancho 8 cm y 1 kg.
- 4 varillas metálicas (2 a cada lado), con dimensiones y masa aproximada a la real: 15,6x1x1 cm y 0,2 gr.

En este modelo, a diferencia del anterior, se evaluaron diferentes valores de roce en los “links” de las ruedas con el fin de no deslizar al frenar, y de fricción en los “joints” con el propósito de mejorar el comportamiento de los brazos del robot ya estos no se muevan al desplazar el modelo. Por esto se tiene en consideración lo que describe [59] para el roce y en [60]para la dinámica de la fricción. Esto menciona que:

- Para formato SDF según las características del motor físico ODE, los valores de coeficiente de roce de un “link” pueden variar entre [0 – 1], siendo 0 el mínimo coeficiente de roce y 1 el mayor. Por otra parte, el valor del coeficiente de deslizamiento también varía entre [0 -1], donde 0 es el menor y 1 el mayor grado de deslizamiento respectivamente.
- También ODE ofrece la opción de modificar los valores de fricción entre [0 – 1], siendo 0 el mínimo valor de fricción y 1 el máximo.

Cabe destacar que la descripción de “roce” debe realizarse dentro de “collision” en el “link” (recordando que este representa las propiedades físicas del modelo) y “friction” dentro de “joint” (recordando que este describe la conexión entre dos cuerpos), y siguiendo el formato que describe [59] como se muestra en la Figura 5.6. El código completo se puede estudiarse en [57].

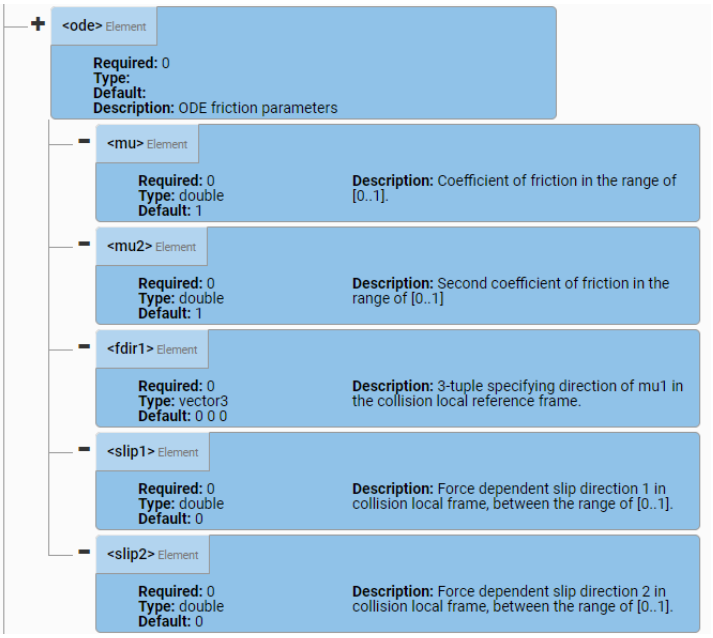


Figura 5. 6 Formato de descripción SDF de propiedades de roce y fricción para el modelo de Gazebo.

La idea general para la construcción del modelo se basa en el siguiente diagrama de la Figura 5.7:

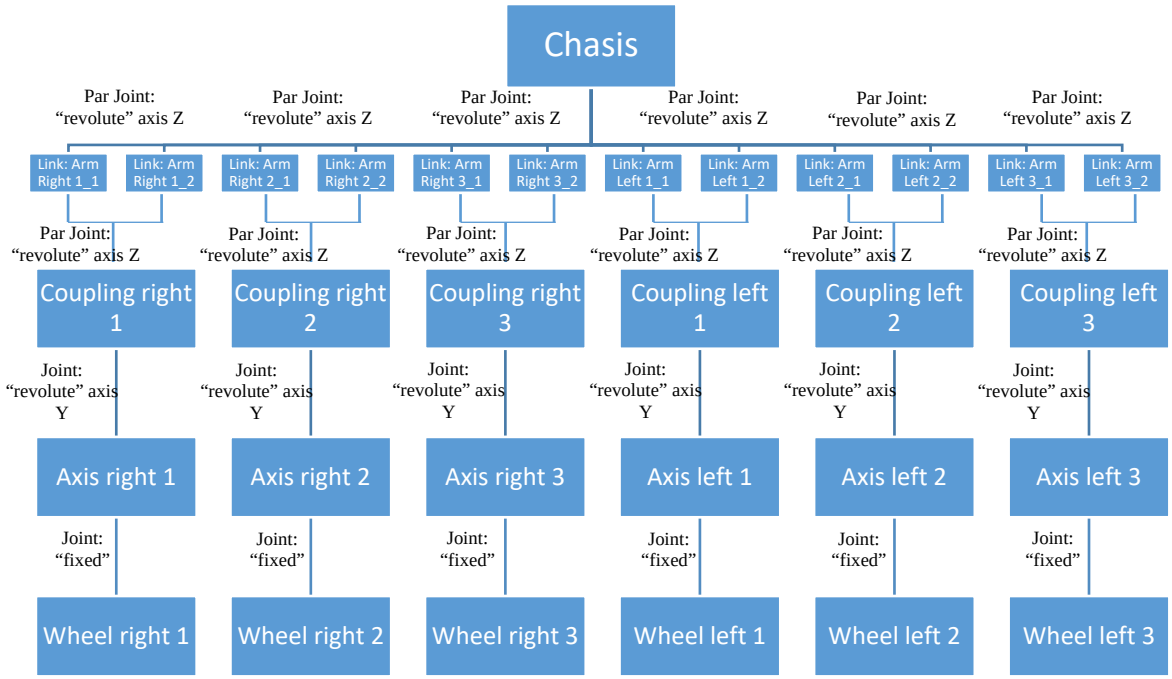


Figura 5. 7 Arquitectura general de la estructura, distribución de joints y links, del segundo modelo desarrollado a simular en Gazebo

En consecuencia, el diseño del modelo simulado se aprecia en la Figura 5.8.

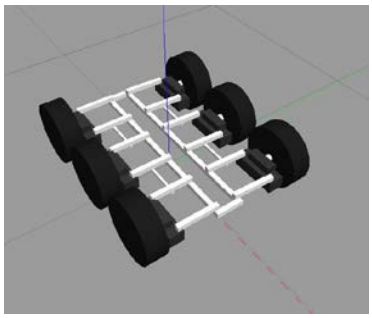


Figura 5. 8 Vista del primer modelo generado en Gazebo

Se realizaron pruebas para evaluar el comportamiento del modelo utilizando las mismas herramientas de Gazebo del punto anterior. Los resultados fueron los siguientes:

1. En cuanto a las inercias, el modelo se desempeña de manera casi normal, con la salvedad de que, al estar en reposo, por acción de los motores físicos los brazos del modelo se desplazan levemente en una dirección como si no existiera coeficiente de fricción, y, además, el modelo desliza sin que se aplique fuerza, Figura 5.9.ca
2. Modificando el valor del coeficiente de “roce” (mu) y “slip” de las ruedas a 0.98 y 0 respectivamente, el modelo tiene un mejor comportamiento de frenado (más inmediato). Al desplazarse, el modelo varía la posición de los brazos y en consecuencia el ancho. Se utilizando un coeficiente de “friction” de 0.5 y si bien el movimiento es más forzado y menos suave que el observado en el modelo anterior, el comportamiento no es el deseado. En consecuencia, se asume que la fuerza de roce de las ruedas con el entorno es mayor que la de fricción en los “joints” definidos en cada brazo, por eso aún se mueven, Figura 5.9.b
3. Al aplicar una fuerza sobre los brazos, el modelo modifica la posición sin inconvenientes, este se muestra estable y con una dinámica adecuada, Figura 5.9.c

(a) (b) (c)

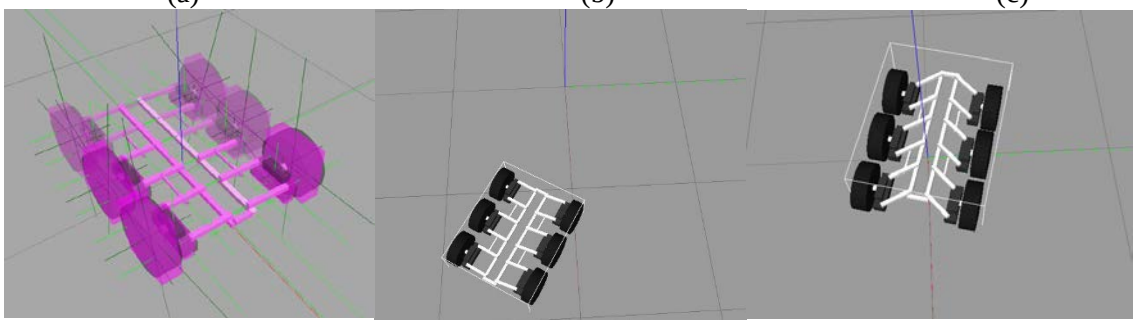
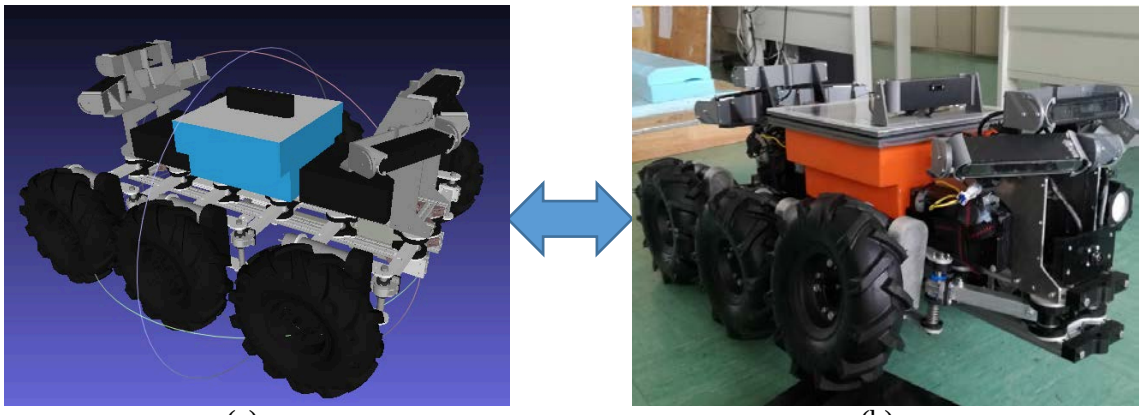


Figura 5. 9 Visualización de pruebas de comportamiento de inercia (a), desplazamiento (b) y movimiento de brazos (c) realizadas al segundo modelo

5.3. Simulación del modelo real

Una vez conocido el comportamiento general del modelo en Gazebo, se prosiguió a implementar el modelo con una aproximación exacta a la plataforma real, para eso se consideró la estructura que se presenta en Figura 5.10 tomada del “Deliverable SIAR_D28.6” la cual representa a la plataforma SIAR en su estado final.



(a) (b)

Figura 5. 10 Imagen del archivo 3D (a) y real (b) de la plataforma SIAR a considerar para la realización del tercer modelo con similitud al real.

En consecuencia, a lo realizado en la sección 5.2.2 se le agrego las siguientes características:

- Chasis Inferior con las mismas dimensiones del modelo anterior, pero con masa total 5 kg.
- Chasis superior, como base de la caja de la electrónica: con las mismas características de dimensión y masa que chasis inferior.
- Caja de electrónica: con dimensiones y peso aproximados al real: cuerpo inferior 26x19x7 cm y cuerpo superior de 26x26x7 cm y masa total 14 kg.
- Soportes para cámaras frontal y posterior: con dimensiones y peso aproximados al real: 1.65 kg
- Color de las distintas partes del modelo según plataforma real.

El archivo, llamado “siar.sdf”, con la descripción completa del modelo se encuentra en [57].

La idea general para la construcción del modelo se basa en el siguiente diagrama de la Figura 5.11.

El resultado obtenido de la simulación se aprecia en la Figura 5.12.

Se realizaron pruebas del comportamiento del modelo utilizando las mismas herramientas de la GUI de Gazebo de los apartados anteriores. Los resultados fueron los siguientes:

1. En cuanto a las inercias, el modelo se desempeña de manera casi estable, el único alcance es que, al estar en reposo, por acción de los motores físicos los brazos del modelo se desplazan en una dirección como si no existiera coeficiente de fricción, Figura 5. 13.a. A pesar de que se varió el coeficiente de roce de las ruedas, y de fricción de los “joints” el comportamiento se mantuvo. Se supuso que esto es a causa de la interacción del modelo con los motores físicos, los cuales generan sobre los momentos inerciales del modelo una rotación en “z”.
2. Al aplicar una fuerza sobre los brazos, el modelo modifica la posición sin inconvenientes, este se muestra estable y con una dinámica adecuada, Figura 5.13.b
3. Modificando valores de roce de las ruedas casi al máximo (a 0.98 de 1 según ofrece el formato SDF en Gazebo), el modelo tiene un mejor comportamiento de frenado (más inmediato). Al desplazar el modelo, los brazos aún se mueven, variando el ancho del robot, lo cual es un comportamiento no deseado, pero este movimiento es más forzado y menos suave que en modelo del punto anterior. En consecuencia, se asume que la fuerza de roce el mayor que la de fricción en los “joints” definidos en cada brazo, por eso aún se mueven. Figura 5.13.c.

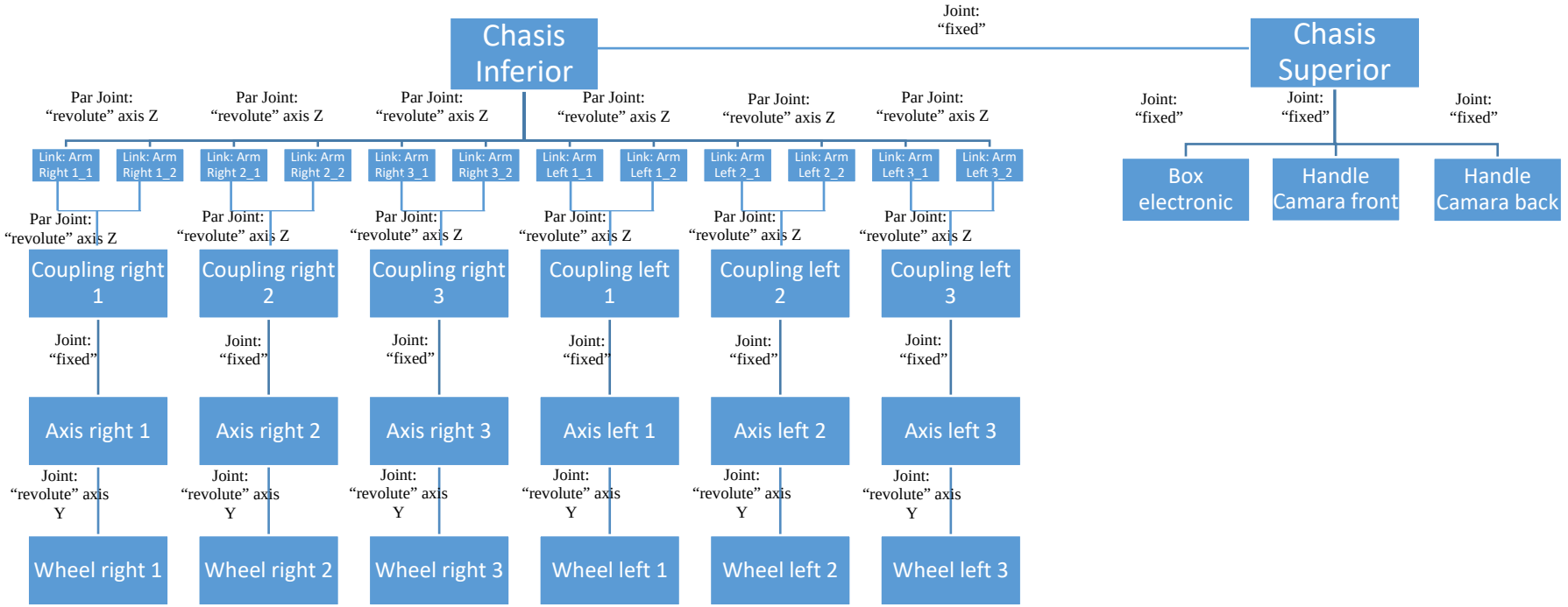


Figura 5. 11 Arquitectura general de la estructura, distribución de joints y links, del ultimo modelo desarrollado a simular en Gazebo

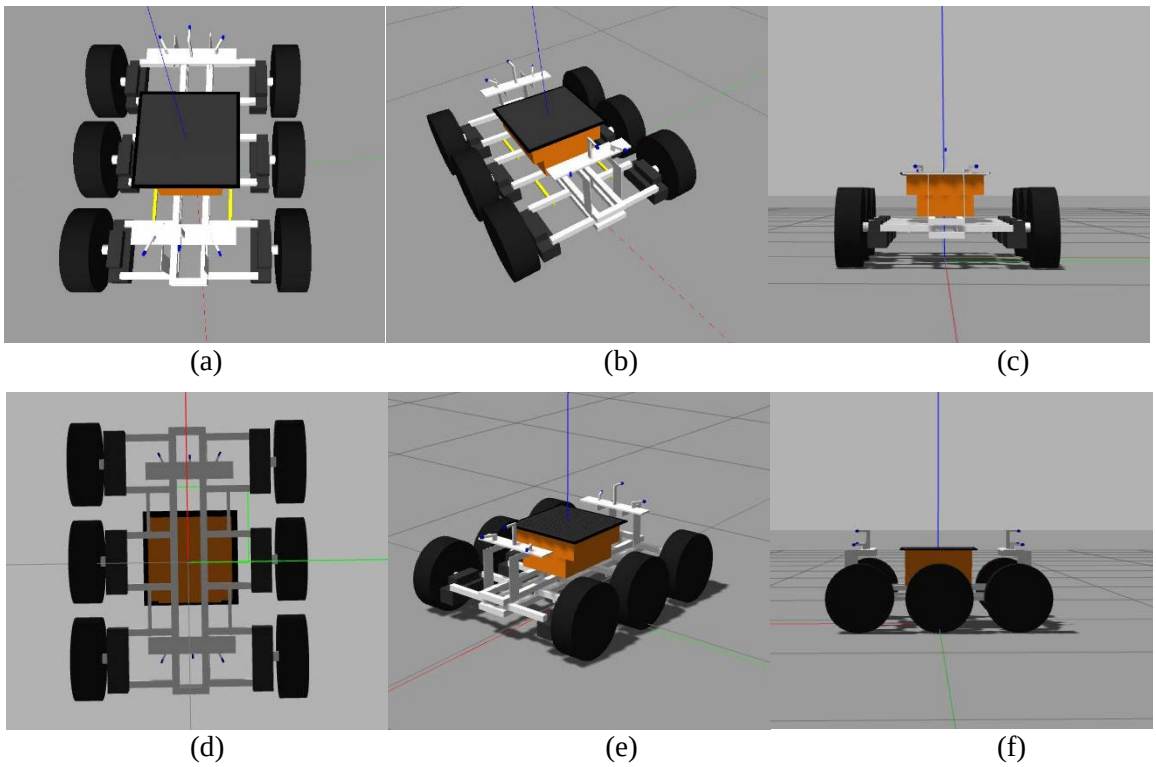


Figura 5. 12 Modelo final desarrollado en formato SDF para implementar en Gazebo que representa de manera estructuralmente semejante a la plataforma SIAR.

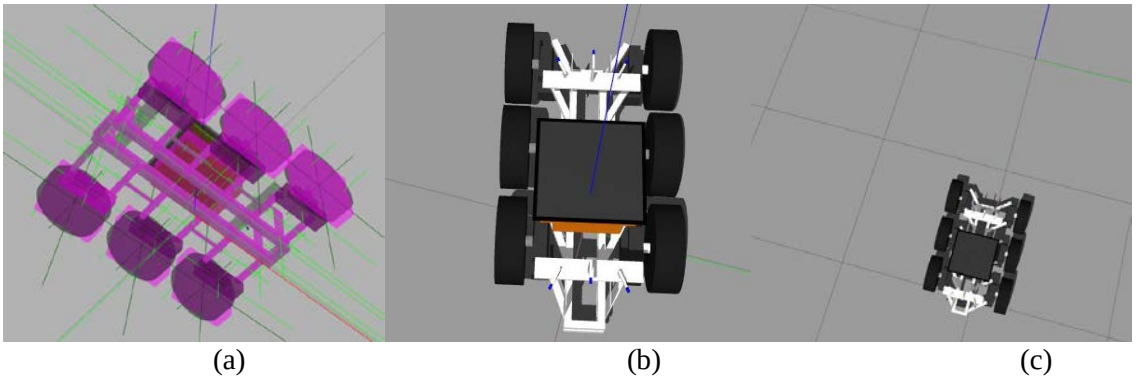


Figura 5. 13 Visualización de pruebas de comportamiento de inercia (a), desplazamiento (b) y movimiento de brazos (c) realizadas al segundo modelo

5.4. Usando ROS para comenzar Gazebo

Según lo revisado, existen muchas maneras de iniciar Gazebo, abrir modelos de mundos (worlds) y generar modelos de robots en el entorno simulado. Teniendo en cuenta lo analizado en los capítulos II y III, se desea utilizar ROS para implementar los algoritmos robóticos porque permite enviar mensajes y servicios (message y services) sobre el Gazebo para manipular la simulación, también, permite implementar el código de manipulación del robot simulado sin fisuras para ser traspasados a la plataforma real, además, es posible adquirir datos del entorno que interpretan los sensores simulados, entre otras características. El modo con el cual se utilizó ROS como interfaz para interactuar con Gazebo fue a través del uso de herramientas “rosrun” y “roslaunch”.

En esta sección cabe destacar dos hechos importantes que difieren de los instructivos ofrecidos por la website de Gazebo en la etapa de integración ROS- Gazebo. Estos se describen a continuación:

- El primero está relacionado con la instalación del "package" "gazebo_ros_pkgs" que se menciona en capitulo III, sección 3.2.5. En esta sección aparece que este paquete permite la integración entre estos dos softwares a través de intercambio de mensajes y otras características. Pero según lo realizado, este "package" es necesario y utilizado en su plenitud con URDF como formato de descripción del modelo simulado. Esto se visualizó una vez finalizado el modelo en Gazebo, momento en el que se deseaba utilizar las herramientas "ros_control" que ofrece este "package" para manipular el movimiento de los brazos y de las ruedas a través de un *achivo.yaml* (el cual describe controles sobre los "joints") pero esto no tuvo efecto, siendo incompatibles para el formato SDF utilizado. La solución de este problema se describe en la sección 5.5.
- El segundo hecho corresponde a la no utilización de la jerarquía que solicita ROS según lo mencionado en la sección 3.2.6. Esto se debió principalmente a que el formato SDF ofrece una mayor flexibilidad y versatilidad del uso descripciones del modelo, es decir, este formato permite incorporar en un solo archivo la descripción del modelo completo (links, joints, material, sensores, plugins, etc.) sin la necesidad de crear nuevos archivos que sean llamado desde el archivo principal del modelo según las características XML del formato URDF.

5.4.1. "roslaunch" para abrir modelo desde paquetes

Como consecuencia de lo mencionado anteriormente, al no ser necesario la incorporación del "package" "gazebo_ros_pkgs", para integrar ROS con Gazebo, la herramienta utilizada para ejecutar los archivos del modelo Gazebo en ROS fue "roslaunch". Este es el método estándar para iniciar nodos ROS y lanzar robots en ROS. Para lanzar un archivo lunch, es necesario crear un archivo lunch con diferentes argumentos en formato XML de ROS que llame al robot simulado en Gazebo, simplemente ejecutando:

roslaunch nombre_paquete_ros nombre_archivo_launch.launch

Además, con el fin de que el modelo de Gazebo en comunicación con ROS se ejecute correctamente, se siguieron los siguientes pasos:

- Crear un espacio de trabajo ROS (workspace) [61], lo cual permite construir paquetes ROS. Para esto se pueden utilizar los comandos:
  - mkdir -p ~/*nombre_del_paquete*_ws/src
  - cd ~/*nombre_paquete*_ws/
  - catkin_make
- Crear "packages" ROS [62]con las respectivas dependencias de librerías necesarias para describir el modelo. Para esto se pueden utilizar los comandos:
  - cd ~/*nombre_del_paquete*_ws/src/
  - catkin_create_pkg *nombre_paquete dependencia_1 dependencia_2* …
  - cd ~/*nombre_del_paquete*_ws/
  - catkin_make

  Esto permitirá contar con los archivos "*CMakelists.txt*" y "*package.xml*" que describen las características del "package" y además con las carpetas "build" (que configura y construye el "package") y "devel" (donde se almacenan las librerías y ejecutables del "package").
- Agregar el "workspace" generado en el entorno de ROS es necesario obtener el archivo de configuración generado [62]. Para esto se pueden utilizar los comandos:
  - . ~/*nombre_del_paquete*_ws/devel/setup.bash

5.4.1.1 Argumentos archivos “roslaunch”

Los argumentos que cambian el comportamiento de inicio en Gazebo son:

- paused: Para iniciar Gazebo en estado pausado.
- use_sim_time: Dice a nodos ROS que pidan tiempo para obtener” el tiempo de simulación publicado por Gazebo”, publicado sobre ROS topic/clock
- GUI: Inicia la ventana de interfaz de usuario de Gazebo
- recording: Habilitar grabación de registro de estado de Gazebo
- debug: Inicia gzserver en modo de depuración usando gdb.

El resultado del archivo launch desarrollado para lanzar el modelo en Gazebo se muestra a continuación en la Tabla 5.1.

Tabla 5. 1 Estructura del archivo “siar_model.launch” desarrollado para lanzar el modelo a Gazebo

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
        <arg name="world" default="empty"/>
        <arg name="paused" default="false"/>
        <arg name="use_sim_time" default="true"/>
        <arg name="gui" default="true"/>
        <arg name="headless" default="false"/>
        <arg name="debug" default="false"/>
    <env name="GAZEBO_MODEL_PATH" value="$(find siar_gazebo)/models:$(optenv GAZEBO_MODEL_PATH)"/>
        <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
        <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <!-- arg name="world_name" value="$(arg world)"/>  -->
                <arg name="paused" value="$(arg paused)"/>
                <arg name="use_sim_time" value="$(arg use_sim_time)"/>
                <arg name="gui" value="$(arg gui)"/>
                <arg name="headless" value="$(arg headless)"/>
                <arg name="debug" value="$(arg debug)"/>
        </include>
        <!-- Spawn the robot into Gazebo -->
                <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-file $(find
                siar_gazebo)/models/siar.sdf -sdf -model siar" output="screen"/>
</launch>
```

Por otra parte, la jerarquía de la carpeta desarrollada hasta esta etapa, quedo como muestra la Tabla 5.2.

Tabla 5. 2 Jerarquia de carpeta con workspace /siar_ws y package /siar_gazebo desarrollada para almacenar documentos del proyecto de simulación

```
…/siar_ws/
        build
        devel
        src
            /siar_gazebo
                package.xml
                CMakeList.txt
                /launch
                    siar_model.launch
                /models
                    model.config
                    siar.sdf
```

5.5. Desarrollo de plugin para mover ruedas y brazos.

Una vez obtenido el modelo de la sección 5.3 y comunicado con ROS, con el fin de dar funcionabilidad, controlar y manipular el modelo, y además vincular mensajes y servicios de ROS con sensores o actuadores simulados en Gazebo, se desarrolló un “plugin” basado en el lenguaje estándar “C++” llamado “plugin_siar_wheels_piston.cpp” el cual manipula las ruedas y los brazos del modelo siguiendo la idea de la plataforma real.

La idea principal recae en, a partir de un único plugin, realizar dos acciones, una sobre las ruedas y otra sobre los brazos de forma independiente, es decir que no se afecten una acción con la otra. Utilizando las herramientas que ofrecen ROS y Gazebo se desarrolló un plugin que genera un nodo ROS llamado “/Gazebo” que vincula el entorno de Gazebo con ROS a través de subscripción y publicación de mensajes y servicios sobre variables seleccionadas (en este caso sobre los “joints” de los ejes de las ruedas y de las articulaciones de los brazos). La explicado se aprecia en la Figura.5.14. El código completo puede ser revisado en [57].

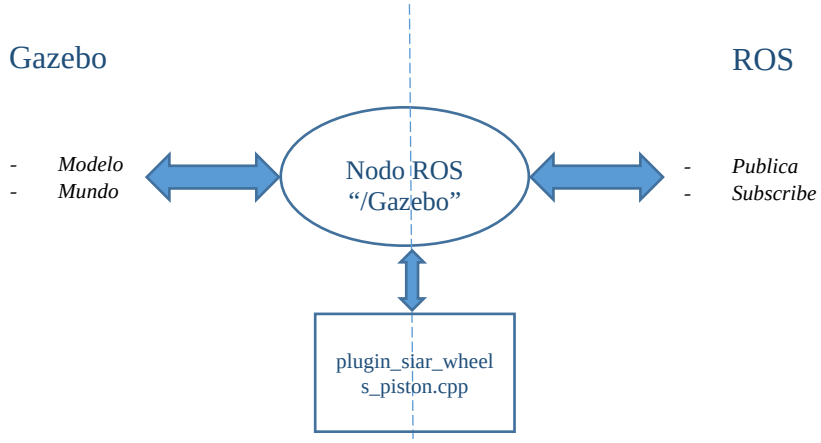


Figura 5. 14 Relación establecida entre Gazebo y ROS por medio del nodo /Gazebo generado por el plugin _siar_wheels_piston.cpp

El plugin desarrollado se incorporó en el archivo SDF desarrollado (siar.sdf) dentro de la descripción del modelo, lugar donde se definen parámetros necesarios para que el plugin asocie la funcionalidad a las partes del modelo, como se muestra en la Tabla 5.3. De esta manera al realizar el “launch” del modelo, también se ejecuta el plugin como elemento perteneciente a este.

Tabla 5. 3 Descripción en formato SDF del “plugin_siar_wheels_piston.cpp” dentro del modelo del robot simulado.

```
<plugin name="plugin_siar_wheel_piston" filename="libplugin_siar_wheels_piston.so">
            <alwaysOn>true</alwaysOn>
            <updateRate>25.0</updateRate>
            <leftJoints>move_axis_wheel_left_1 move_axis_wheel_left_2  move_axis_wheel_left_3</left Joints>
            <rightJoints>move_axis_wheel_right_1 move_axis_wheel_right_2 move_axis_wheel_right_3</rightJoints>
            <initWheelSeparation>0.602</initWheelSeparation>
            <wheelDiameter>0.25</wheelDiameter>
            <torque>15</torque>
            <interface:position name="position_iface_2"/>
            <robotNamespace>/siar_model</robotNamespace>
            <robotBaseFrame>piston_1</robotBaseFrame>
            <commandTopic>cmd_vel</commandTopic>
            <publishOdometryTf>0</publishOdometryTf>
            <publishOdometryMsg>1</publishOdometryMsg>
            <odometryFrame>chassis_3</odometryFrame>
            <odometryTopic>siar_model/odomTopic</odometryTopic>
</plugin>
```

Para realizar una integración de todos los elementos desarrollados, como se mencionó anteriormente, es necesario contar con una estructura de packages, de tal forma que ROS pueda relacionarlos. Por esta razón, se crea el package /siar_plugin, el cual almacena el "plugin_siar_wheels_piston.cpp" tal como muesta la Tabla 5.4

Tabla 5. 4 Jerarquia de carpeta con workspace /siar_ws y packages /siar_gazebo y /siar_plugins desarrollada para almacenar documentos del proyecto de simulación

```
…/siar_ws/
        build
        devel
        src
            /siar_gazebo
                package.xml
                CMakeList.txt
                /launch
                    siar_model.launch
                /models
                    model.config
                    siar.sdf
            /siar_plugins
                package.xml
                CMakeList.txt
                /include
                    /siar_plugins
                        plugin_siar_wheel_piston.h
                /src
                    plugin_siar_wheels_piston.cpp
```

5.5.1. Parte I: Consideraciones del Plugin para manipular ruedas

Para conseguir un comportamiento diferencial, se consideró una manipulación de cada lado independiente de las ruedas del modelo. Además, se consideró una velocidad limite según lo mencionado en "Deliverable SIAR_D28.6".

Para conseguir lo mencionado anteriormente, se tomó como referencia algunos plugins existentes, el primero desde la librería de Gazebo llamado "libgazebo_ros_diff_drive.so" para un robot móvil diferencial de dos ruedas [63], y el segundo desde los repositorios de GitHub "tu-darmstadt-ros-pkg/hector_gazebo" [64] para robots móviles multi-rueda. El segundo de estos, toma en consideración la estructura del primero, por lo cual se utilizó como código base para escribir el plugin deseado.

Para manipular los "joints" de las ruedas de forma deseada, se utilizó algunas APIs de Gazebo y librerías C++ de ROS. Las cabeceras utilizadas por parte de Gazebo y ROS fueron:

- <gazebo/physics/physics.hh>
- <gazebo/math/gzmath.hh>
- <sdf/sdf.hh>.
- <ros/ros.h>
- <geometry_msgs/Twist.h>
- <geometry_msgs/Vector3.h>

Lo realizado se menciona a continuación:

1. A través de una API de Gazebo que considera la clase para SDF, se utilizó "GetElement" que genera un puntero vinculado con los "joints" de giro del eje de las ruedas del modelo SDF de Gazebo (definidos en <leftJoints> y <rightJoints> de la tabla 5.3).
2. Mediante librerías de ROS, se genera un "Subscriptor" que lee la información enviada por ROS para establecer la velocidad lineal y angular del modelo.
3. Usando nuevamente APIs de Gazebo que consideran referencias de clases para modelo, link y joint. De esta manera, se pasa información del estado de los "joints" a una variable

a través de la referencia de clase “GetJoint”. Luego se obtiene el valor del ancho del modelo considerando la posición de las ruedas en el mundo utilizando la referencia de clase de modelo “GetLink” y la referencia de clase de link “GetWorldCoGPose”. Posteriormente, con la información del ancho del modelo y la información a la que se subscribe el nodo se procede a establecer la velocidad de giro del modelo con la referencia de clase de joint “SetVelocity”.

Para establecer la velocidad de cada rueda en el simulador se consideró las ecuaciones de modelo cinemático de un robot móvil diferencial, en las cuales, la velocidad lineal (v) es el promedio de las velocidades de las ruedas de cada lado (ecuación 5.4) y la velocidad angular (w) depende de la separación de las ruedas (d) como muestra la ecuación (ecuación 5.5). El valor de la velocidad lineal y angular son valores a los que se subscribe el nodo ROS. En consecuencia, despejando se obtienen los valores de la velocidad en ruedas izquierdas “v_l” (ecuación 5.6) y derechas “v_r” (ecuación 5.7), que se envían a los joints de los ejes de las ruedas.

$$v = \frac{v_r + v_l}{2} \quad (5.4)$$

$$w = \frac{v_r - v_l}{d} \quad (5.5)$$

$$v_l = v - w * \frac{d}{2} \quad (5.6)$$

$$v_r = v + w * \frac{d}{2} \quad (5.7)$$

Lo explicado anteriormente se puede apreciar en la Figura 5.15.

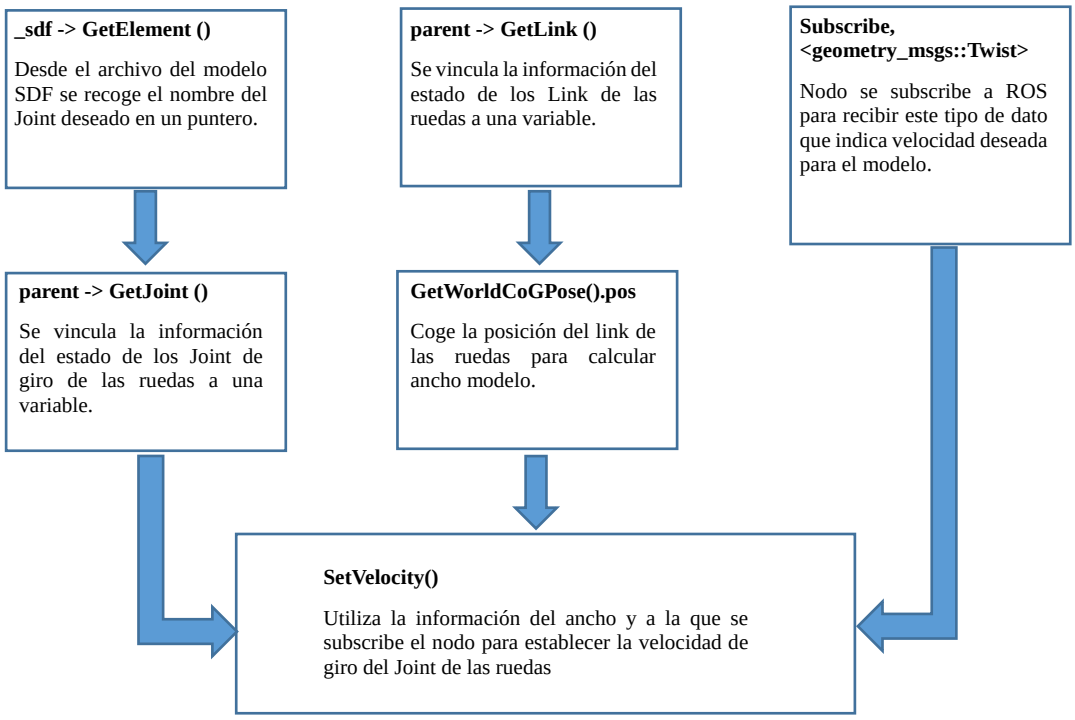


Figura 5. 15 Filosofia de funcionamiento del plugin para manipular las ruedas del robot simulado en Gazebo

5.5.2. Parte II: Consideraciones del Plugin para manipular brazos

Para realizar la configuración del movimiento de los brazos se consideró un utilizar un control PID que manipulara la posición de estos a través del movimiento de sus joints y así simular la acción del motor lineal que tiene la plataforma. Para esto se tomó en consideración el mecanismo de ajuste de ancho que se menciona en el "Deliverable SIAR_D28.6".

Para manipular los "joints" de forma deseada, se utilizó algunas APIs de Gazebo y librerías C++ de ROS. Las cabeceras utilizadas por parte de Gazebo y ROS fueron:

- <gazebo/physics/physics.hh>
- <gazebo/common/common.hh>
- <gazebo/math/gzmath.hh>
- <ros/ros.h>
- <geometry_msgs/Twist.h>
- <geometry_msgs/Vector3.h>
- <std_msgs/Bool.h>
- <std_msgs/Float32.h>

La lógica de los realizado se menciona a continuación:

1. A través de las APIs de Gazebo, referencia de clase "GetJoint", se asocia a un puntero cada una de "joints" manipulados (únicamente los "joints" de los brazos centrales a ambos lados).
2. Se utiliza la referencia de clase de link "GetWorldCoGPose" para obtener la posición de las ruedas y la caja central de la electrónica, así a través de cálculo vectorial conocer si los brazos se mueven hacia delante o hacia atrás, el valor es almacenado en la variable virtual "*dis_box_centralaxis*"
3. Mediante librerías de ROS, se genera un "Subscriptor" que lee la información enviada por ROS. Esta información es almacenada en la variable virtual "*move_Piston_cmd_"* y se utiliza para ordenar si los brazos se mueven hacia delante o atrás.
4. Utilizando la referencia de clase PID a través de "common::PID" se establecen los valores proporcionales, integrales y derivables para crear e inicializar el controlador. Estos van a estar sometidos a constantes cambios en la etapa de sintonización.
5. A través de los valores sintonizados de PID y la referencia de clase base de "physics" de Gazebo "GetScopeName" que vincula el nombre de una entidad con alcance al mundo del modelo, se establece los valores PID de la posición para un joint usando "SetPositionPID" de la referencia de clase JointController.
6. A través de la referencia de clase base de "physics" de Gazebo "GetScopeName" que devuelve el nombre de una entidad con alcance al mundo del modelo y el valor de la variable "*elec_pos_cmd_"*, que se nutre de "*move_Piston_cmd_"*, se calcula la posición objetivo de un controlador PID usando SetPositionTarget" de la referencia de clase JointController.

Finalmente el filosofía de control recae en situar los joint de los brazos centrales en cierta posición, el cual inicialmente es 0, lo cual significa que el modelo se encuentra en su ancho máximo, y variar la posición de estos hacia delante o hacia atrás, según sea necesario, variando el valor de la variable "*elec_pos_cmd_"* entre 1.2 y -1.2 lo que significa un movimiento de la electrónica hacia adelante o atrás con la reducción del ancho del robot.

Lo comentado anteriormente se explica en la Figura 5.16.

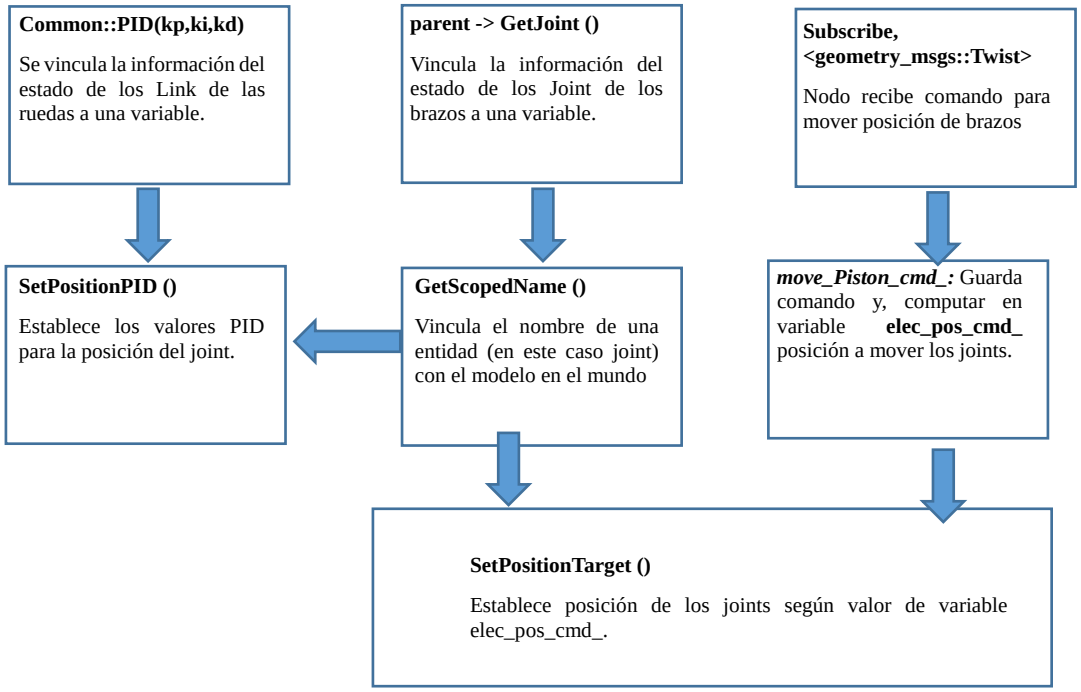


Figura 5. 16 Filosofia de funcionamiento del plugin para manipular brazos del robot simulado en Gazebo

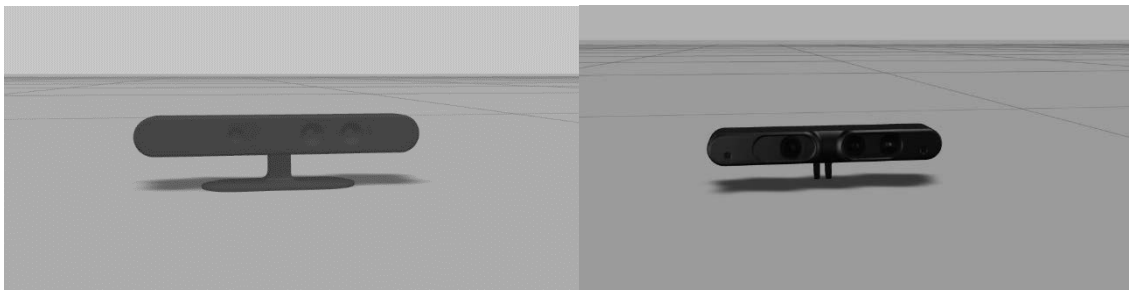
Cabe destacar que la consecuencia de haber implementado este plugin es muy favorable, especialmente la parte relacionada con el control de los, ya que de esta manera se logró fijar la apertura de los brazos a una posición fija sin variación en el caso que el modelo se desplace. Esto debido únicamente a que el movimiento de los se encuentra sujeto al valor que reciba la variable virtual “elec_pos_cmd_” por lo cual, si no se carga valor, no existe movimiento de estos.

5.6. Incorporación de cámaras

Según lo expuesto en el “Deliverable SIAR_D28.6” la plataforma SIAR cuenta con 7 cámaras, las cuales son capaces de identificar profundidad y color. El modelo de cámara utilizado en la plataforma SIAR corresponden a “ASUS XTion”, el cual es posible encontrar dentro la librería de Gazebo, pero con algunas diferencias estructurales comparada con la utilizada.

Figura 5. 17 Representación del posicionamiento de las camaras ASUS Xtion en la plataforma SIAR

La Figura 5.17 muestra el modelo de la cámara en la plataforma real, y, la Figura 5.18.a y 5.18.b muestran modelos que pueden ser cargados en Gazebo. El primero modelo corresponde a “Asus Xtion Pro Depth Cámara” disponible directamente en Gazebo, y el segundo corresponde a “Asus Xtion” descargable desde los repositorios de GitHub [65]. El parecido de estas es mayor de la primera cámara respecto a la utilizada en la plataforma real, la diferencia radica en que el formato “mesh” incluye la base de la cámara, a diferencia del segundo modelo.

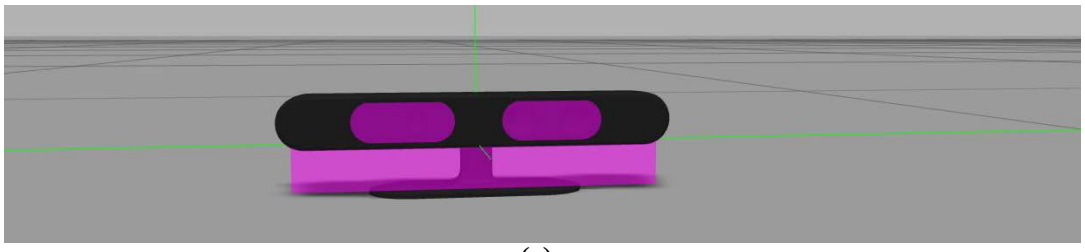


(a) (b)

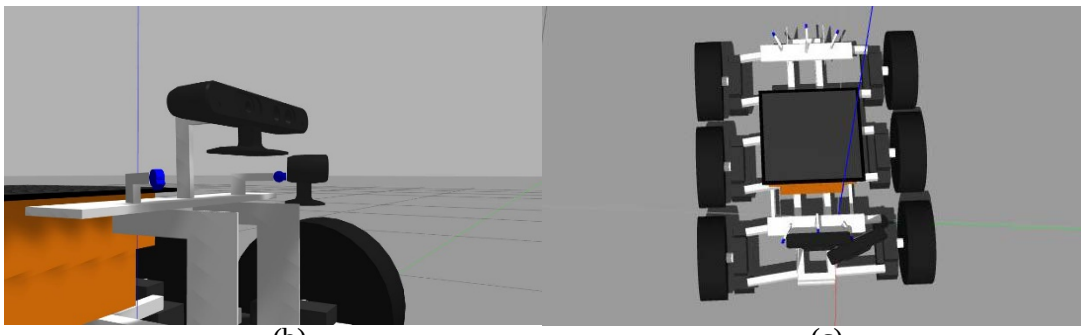
Figura 5. 18 Modelo de camaras ASUS Xtion disponible en Gazebo(a) y en repositorio GitHub(b)

Se realizaron pruebas con ambos tipos de cámaras para conocer comportamiento y desempeño dentro del simulador. Los resultados arrogaron con grandes diferencias entre ambas. La primera cámara está definida con una masa e inercia pequeña, coherente a las características de la geometría del cuerpo, Figura 5. 19.a. Sin embargo, al agregar dos o más cámaras de este tipo, el modelo simulado comienza a vibrar y a tener un comportamiento incoherente dentro de Gazebo, Figura 5.19.b y 5.19.c. Es posible que la causa de esto se deba a que la base de la cámara se solapa con el soporte de cámara del modelo del SIAR y genera así una gran vibración.

Por otra parte, se probó la segunda cámara y si bien esta pose una inercia enorme Figura 5.20.a, debido a que la masa y valores de matriz de inercia no están definidos, el modelo se comporta sin inconveniente y estable, Figura 5.20.b, incluso hasta el punto de agregar siete cámaras, Figura 5.21. Cabe destacar que también esta cámara no tiene base y no se solapa con el modelo simulado. Es por esta razón que es finalmente la cámara del modelo “Asus Xtion” la elegida para implementarse en el modelo. El código completo se puede revisar en [57].

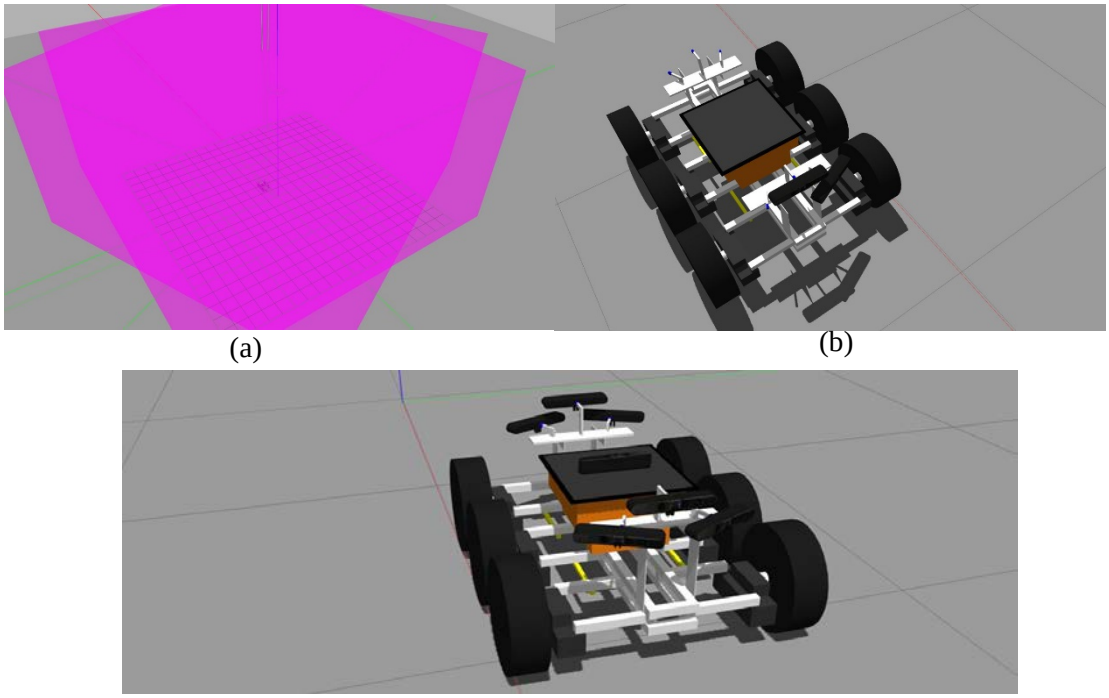


(a)



(b) (c)

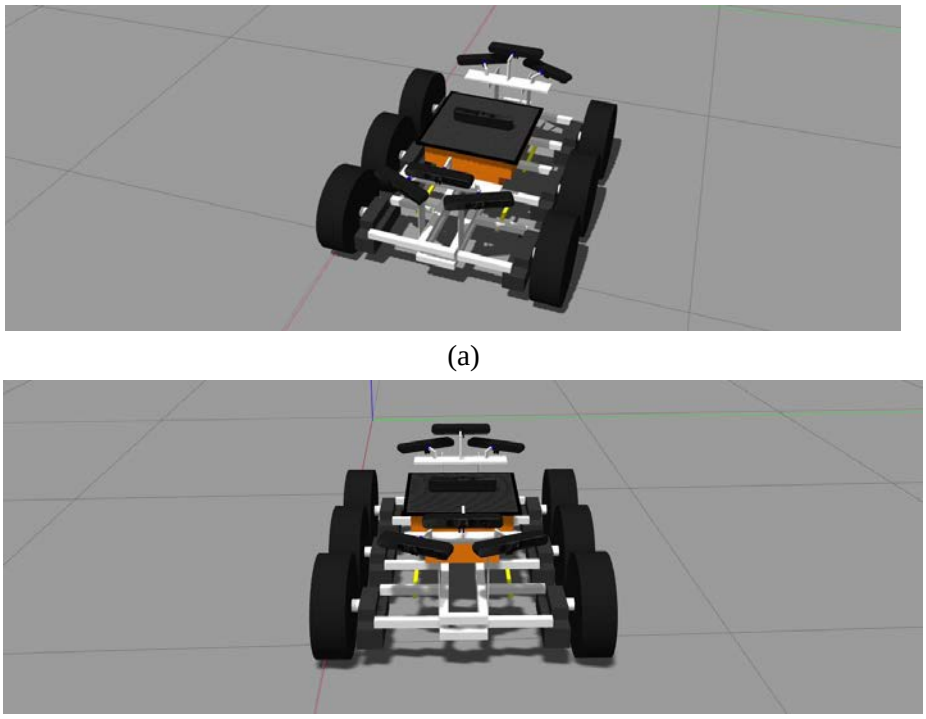
Figura 5. 19 Prueba de inercia de cámara ASUS Xtion ofrecida por simulador Gazebo



(a) (b)

(c)

Figura 5. 20 Prueba de inercia de cámara ASUS Xtion ofrecida descargada de repositorio GitHub



(a)

(b)

Figura 5. 21 Prueba de siete camaras Asus Xtion sobre modelo simulado con resultado estables.

El plugin de cada una de las cámaras es descrito dentro del archivo SDF del modelo desarrollado y utiliza una librería interna de Gazebo, por lo cual no es necesaria definirlo completamente, sino que simplemente llamarlo de la forma "*libgazebo_ros_openni_kinect.so"* y definir sus parámetros del plugin dentro del archivo "siar.sdf", Tabla 5.4. De esta manera una vez que se lanza el archivo del modelo además de lanzar el plugin para controlar las ruedas y los brazos, también se lanzan también se lanza el plugin de las cámaras conectadas al nodo ROS que se comunica con Gazebo, tal como muestra la Figura 5.22

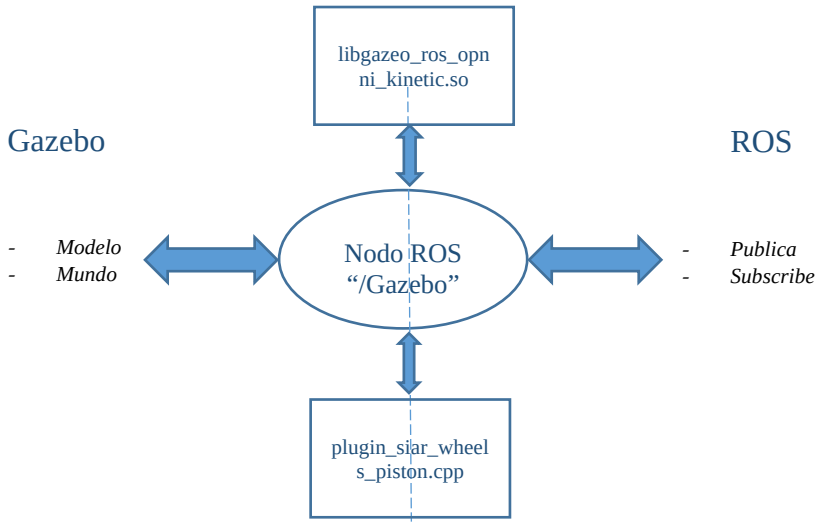


Figura 5. 22 Relación de los plugin de las camaras y plugin_siar_wheels_piston.cpp con nodo ROS "/Gazebo"

Tabla 5. 5 Descripción en formato SDF del plugin de camaras dentro del archivo siar.sdf

```
<plugin name="asusXtion_topMiddle_camera_controller" filename="libgazebo_ros_openni_kinect.so">
            <cameraName>asusXtion_topMiddle</cameraName>
            <robotNamespace>/siar_model</robotNamespace>
            <alwaysOn>true</alwaysOn>
            <updateRate>1.0</updateRate>
            <imageTopicName>rgb/image_raw</imageTopicName>
            <cameraInfoTopicName>/depth/camera_info</cameraInfoTopicName>
            <depthImageTopicName>depth/image_raw</depthImageTopicName>
            <pointCloudTopicName>depth/points</pointCloudTopicName>
             <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
            <depthImageCameraInfoTopicName>depth/camera_info</depthImageCameraInfoTopicName>
            <frameName>camera_link</frameName>
             <baseline>0.2</baseline>
            <distortion_k1>0.00000001</distortion_k1>
            <distortion_k2>0.00000001</distortion_k2>
            <distortion_k3>0.00000001</distortion_k3>
            <distortion_t1>0.00000001</distortion_t1>
            <distortion_t2>0.00000001</distortion_t2>0
            <pointCloudCutoff>0.5</pointCloudCutoff>
            <pointCloudCutoffMax>5.0</pointCloudCutoffMax>
</plugin>
```

5.6.1. Visualización de datos de las cámaras

Para analizar la información de profundidad y color que las cámaras procesan se utiliza el software de ROS, Rviz. Este permite analizar la información de cada una de las cámaras a través de la publicación al nodo de Ros que estas generan a partir de su propio plugin.

De esta manera, se sigue el siguiente procedimiento para visualizar las cámaras:

- Lanzar el launch del modelo (que incluye el plugin de las cámaras)
- Ejecutar en consola Rviz
- Agregar elementos de imagen.
- En el menú desplegable de “Image Topic” elegir el topic de imagen o profundidad, deseado.

El resultado de lo anterior se observa en la Figura 5.23.

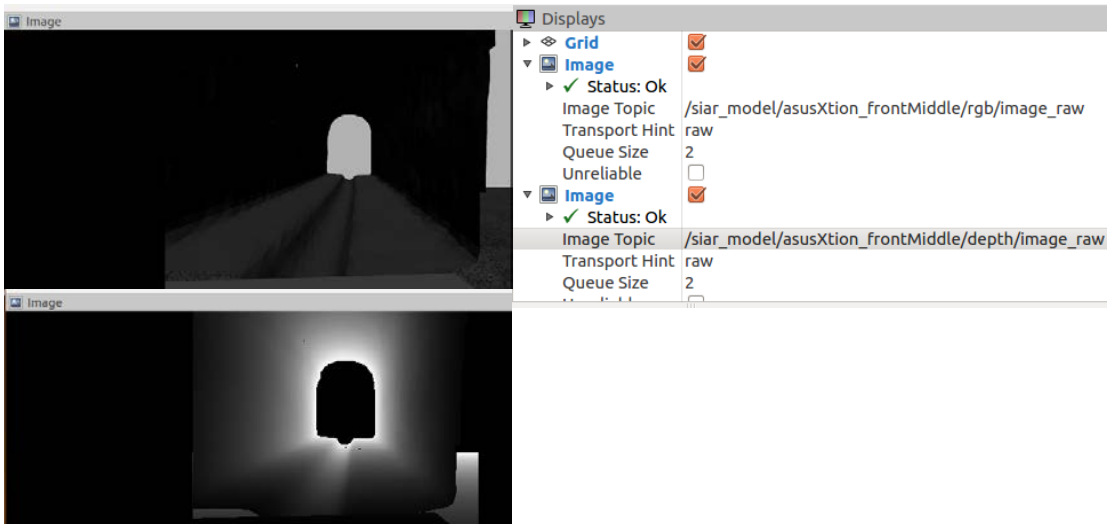


Figura 5. 23 Visualización de datos rgb y depth obtenidos de camara en software Rviz

5.7. Incorporación de Mapa

Para otorgar una mayor versatilidad al simulador, se incorporó una reconstrucción 3D del mapa en el que el SIAR se desenvuelve realmente para desarrollar sus tareas. Este mapa (mapping) fue desarrollado por el grupo de investigación a cargo del proyecto SIAR según “Deliverable SIAR_D28.6” utilizando el método “visual odometry” en primera instancia y luego método de optimización de grafos [66]. El grupo reconstruyo diversos sectores de la red de alcantarillado con una gran efectividad, del cuales, por motivo de reducir el costo computacional, solo se utilizó un pequeño sector de la red de alcantarillado, que considera una esquina (sección curva en donde el robot debe girar) la cual permite probar sin problemas el comportamiento que se espera del robot simulado.

Una vez identificado la sección de la red con la que se quería trabajar, se procedió a editar el mapa 3D (.ply), para esto se utilizó el software “MeshLab” el cual es recomendado dentro de los tutoriales de Gazebo para convertir objetos o mapas 3D [67] a archivo con formato tipo collada(mesh) que pueden ser cargados en Gazebo. El código completo se puede revisar en [57].

Se siguieron los siguientes pasos para editar el mapa:

1. En MeshLab, con la opción “Select Vertex” se selecciona el área y luego con la opción de la barra de herramientas “Delete the current set of selected vértices” se suprimió el área que no se deseaba.
2. Se aplicó la opción “Filters>Point Sets>Compute Normal for Point Sets” la cual permite detectar los puntos asociados al área restante del mapa. Esto puede ser visualizado en “Render>Show Normal”
3. Se aplicó la opción “Filter>Remeshing, Simplification and Reconstruction>Screened Poisson surface Reconstruction” para realizar la construcción del Mesh, la cantidad de “Depth” fue 14, lo cual entrego resultados de detalles muy favorables.

4. Se aplico la transferencia de colores de los puntos al Mesh (Transfer Color) en la opción “Filter>Sampling>Vertex Attribute Transfer”.
5. Se aplico la opción “Filters>Texture>Parametrization:Trivial Per-Triangle” para generar las caras que crean la textura del Mesh. El valor “Texture Dimension” fue de 3000.
6. Se aplico la opción “Filters>Texture> Transfer: Vertex Color to Texture” para generar el nombre del archivo de textura y la resolución del Mesh.
7. Finalmente exporto el archivo Mesh con al opción “File>Export Mesh As..” en el formato collada “.dae” que es compatible con Gazebo.

El resultado de lo realizado se observa en la Figura 5.24.

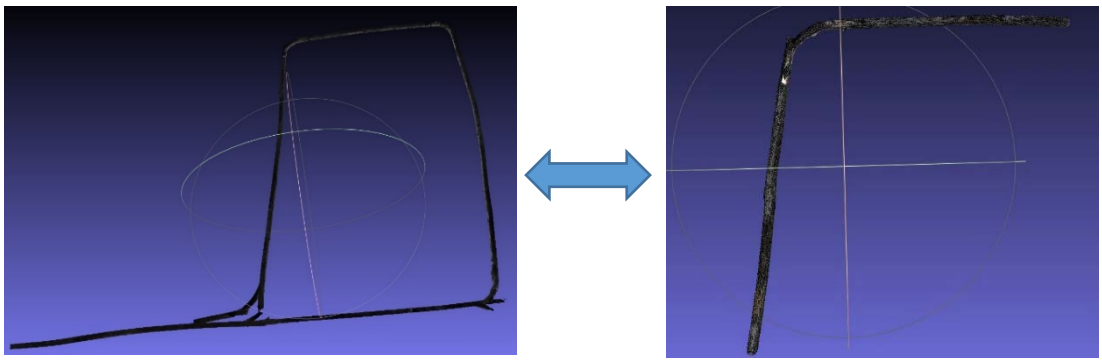


Figura 5. 24 Visualización de modificación de mapa de alcantarillado realizado a través del software Meshlab

Luego se generó un archivo world en formato SDF llamado “siar_map.world” el cual contiene el mesh del mapa modificado. Este es incorpordoa dentro de un archivo llamado siar_world.launch que al ser lanzado permite visualizar el mapa en Gazebo y también en el archivo “siar_gazebo.launch” el cual al ser lanzado permite visualizar el mapa del alcantarillado en Gazebo en conjunto con el modelo. El resultado de lo realizado se visualiza en la Figura 5.25.

Figura 5. 25 Entorno generado a partir de la modificación del mapa del alcantarillado e incorporándolo al archivo siar_model.launch” de simulación del modelo en Gazebo

Como se ha mencionado anteriormente, para que la simulación pueda ser efectiva, es necesario cumplir con una jerarquía de packages con el fin de que ROS pueda organizar los elementos de la simulación. En este caso, con la incorporación del mapa, la jerarquía queda como se muestra en la Tabla 5.6.

Tabla 5. 6 Jerarquia de carpeta con workspace /siar_ws y packages /siar_gazebo y /siar_plugins desarrollada para almacenar documentos del proyecto de simulación y lanzar simulación con mapa y modelo.

```
…/siar_ws/
        build
        devel
        src
            /siar_gazebo
                package.xml
                CMakeList.txt
                /launch
                    siar_model.launch
                    siar_gazebo.launch
                /models
                    model.config
                    siar.sdf
                /worlds
                    siar_map.world
            /siar_plugins
                package.xml
                CMakeList.txt
                /include
                    /siar_plugins
                        plugin_siar_wheel_piston.h
                /src
                    plugin_siar_wheels_piston.cpp
```

5.8. Incorporación de Joystick

Con el fin de contar con un control manual vía mando que permita manipular el simulador, se confecciono un segundo plugin llamado “siar_joy_node.cpp” y se incorporó a la package “siar_plugins”, revisar [57]. Este genera un nodo ROS llamado “siar_model” que lee los “messages” enviados desde un joystick y codifica “messages” legibles para Gazebo y que finalmente se convierte en la acción de movimiento de brazos y/o rotación de ruedas. Para esto, se incluyó un tercer nodo ROS llamado “/joy” el cual se obtiene a través de la instalación del paquete “ros-kinetic-joy” de ROS [68]. Este genera la interfaz virtual entre el ordenador y ROS, convirtiéndose en la principal fuente de generación de “messages” para el simulador. El diagrama de lo mencionado anteriormente se aprecia en la Figura 5. 26.

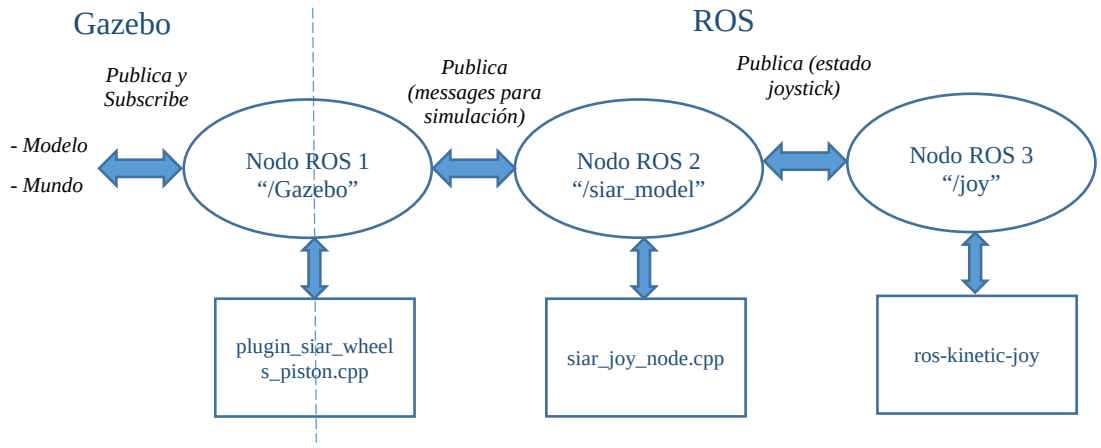


Figura 5. 26 Relación de nodos de ROS para la ejecución de la simulación controlado manualmente por joystick

- **Nodo 3:**
  - Configura y vincula el Joystick conectado al ordenador con ROS
  - Publica estado del Joystick (palanca y botones) al nodo 2.
- **Nodo 2:**
  - Subscribe a messages con la información publicada por nodo 3.
  - Codifica messages recibidos de nodo 3 ha messages entendibles por Gazebo, entre ellos:
    - Velocidad lineal: vel.angular.x = a_scale_*joy->axes[1];
    - Velocidad angular: vel.angular.z = a_scale_*joy->axes[0];
    - Movimiento de brazos: elec_pos_cmd.data = joy->axes[4];
    - Desbloqueo Joystick: deadman_pressed_ = joy->buttons[4];
  - Publica messages a nodo 1.
- **Nodo 1:**
  - Subscribe un messages con la información publicada por nodo 2.
  - Codifica messages y los convierte en acciones sobre el modelo simulado en Gazebo.
  - Publica estado de "links" del modelo e información de adquirida por cámaras.

El joystick utilizado corresponde a un "nacon controller GC-100XF" en el cual fueron configurado los botones y ejes como se muestra en la Figura 5.27.

Figura 5. 27. Distribuciones de ejes y botones utilizados en el joystick nacon controller GC-100XF para controlar manualmente la simulación del modelo en Gazebo.

Cabe mencionar que cada joystick debe ser revisado y configurado una vez conectado al conectado al ordenador. Esto se realizó median los pasos mencionados en [69].

5.9. Incorporación de tf

En las secciones anteriores se puede conocer como finalmente la plataforma fue simulada con un alto grado de éxito, obteniendo un comportamiento muy similar al que tiene la plataforma en la realidad. Sin embargo, para que el simulador tenga el impacto deseado y cumpla los objetivos que debe satisfacer un simulador, es necesario incorporar un nuevo elemento el cual permita realizar un control autónomo del simulador.

Este elemento es llamado tf[18], el cual es un “package” de ROS el cual permite realizar seguimiento de coordenadas de elementos, en el tiempo. De esta manera, es posible incorporar tf para que el robot simulado pueda identificar cada una de sus partes dentro de un frame como el mundo, y, por ejemplo, en caso de identificar con las cámaras un obstáculo como una pared, se estimara la distancia para ejecutar una acción correctiva. El “package” tf como corresponde a una herramienta de ROS, fue desarrollado inicialmente para el formato URDF en el software Rviz[19] [70] y [71], por lo cual, no es directamente aplicable a un formato SDF, por esta razón, se buscó “packages” alternativos que permitieran incorporar esta opción al modelo SDF simulado. De esta manera en el repositorio de GitHub, es posible encontrar “Gazebo2Rviz” [70] y [71], “package” que permite importa todas las entidades simuladas en Gazebo a Rviz y ser representadas en tf. En adición, es necesario agregar otro “package” llamado “pysdf” [72] y [73],el cual permite convertir a los ojos de Rviz el modelo en formato SDF a formato URDF.

De esta manera, una vez instalados estos dos paquetes, es necesario seguir el procedimiento de [74] para visualizar los tf del modelo SDF en Rviz:

- launch modelo en Gazebo lanzar Rviz
- add tf y Marker en Rviz
- lanzar nodos Gazebo2Rviz: roslaunch gazebo2rviz gazebo2rviz.launch

Para comprobar el funcionamiento se seleccionaron los tf de las ruedas, Figura 5,29, teniendo como frame fijo (Fixed Frame) la caja de la electrónica del modelo simulado, se tiene los visualizado en la Figura 5.28.

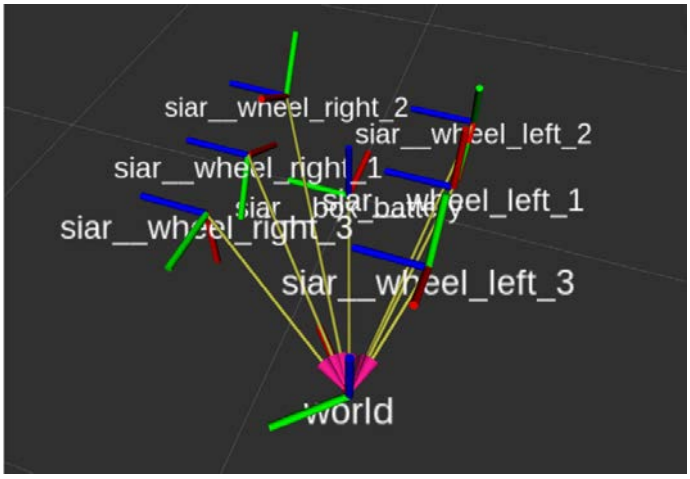


Figura 5. 28 Disposición del tf en modelo SIAR generado en Gazebo y comunicado con Rviz a traves de paquete gazebo2rviz

[18] tf es un paquete que le permite al usuario realizar un seguimiento de múltiples “frames” de coordenadas a lo largo del tiempo. tf mantiene la relación entre las tramas de coordenadas en una estructura de árbol almacenada en el tiempo, y permite al usuario transformar puntos, vectores, etc. entre dos fotogramas de coordenadas en cualquier punto del tiempo deseado. Un sistema robótico generalmente tiene muchos marcos de coordenadas 3D que cambian con el tiempo, como un frame mundial, frame base, frame de agarre, frame de cabeza, etc. tf realiza un seguimiento de todos estos frame o largo del tiempo y permite saber, por ejemplo, dónde está el marco de la cabeza en relación con el frame del mundo, la postura del objeto en una pinza en relación con la base, la pose actual del frame base en el frame del mapa.

tf puede operar en un sistema distribuido. Esto significa que toda la información sobre los marcos de coordenadas de un robot está disponible para todos los componentes de ROS en cualquier computadora del sistema. No hay un servidor central de información de transformación.

[19] rviz (visualización ROS) es un visualizador 3D para mostrar datos de sensores e información de estado de ROS. Usando rviz, puede visualizar la configuración actual de un modelo virtual del robot. También puede visualizar representaciones en vivo de los valores de los sensores que aparecen en ROS Topics, incluidos los datos de la cámara, las mediciones de distancia por infrarrojos, los datos del sonar y más.

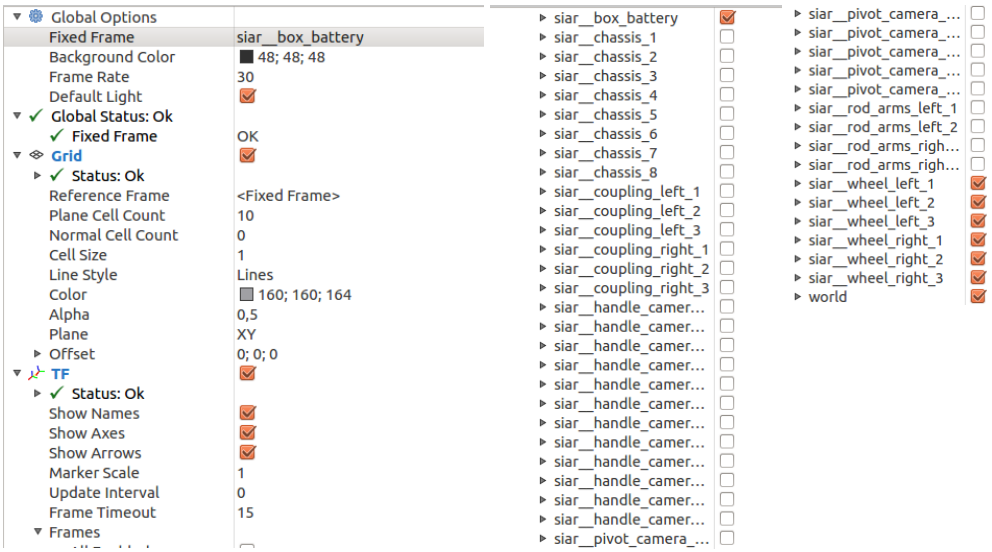


Figura 5. 29 Selección de links para la visualización de tf en Rviz

Además, ejecutando el comando “evince frames.pdf” es posible obtener un documento PDF con la relación en forma de árbol de todas las partes del modelo. Una pequeña parte de este árbol se cómo se aprecia en la Figura 5.28:

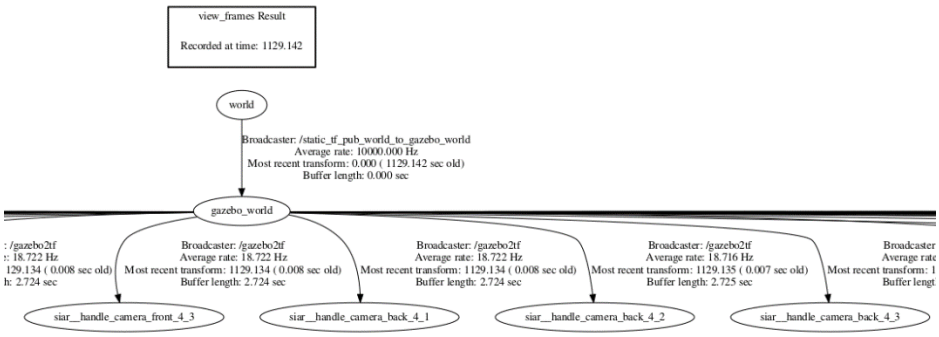


Figura 5. 30 Sección del arbol generado para visualizar la relación de los tf en el modelo

5.10. Incorporación del simulador a repositorio del Proyecto SIAR en Github

Como se mencionó en los primeros capítulos, una de las razones por las cuales se eligió Gazebo como simulador, es que este es de código abierto y permite compartir los desarrollos con la comunidad de OSRF, es por eso, que se decidió incorporar el desarrollo del simulador a los repositorios de GitHub [57]. Para esto se permitió el acceso como miembro del repositorio del proyecto. Esto se llevó a cabo por el Ingeniero a cargo administrar el repositorio, David Alejo Teissière. La incorporación del simulador en el repositorio se observa en la Figura 5.29

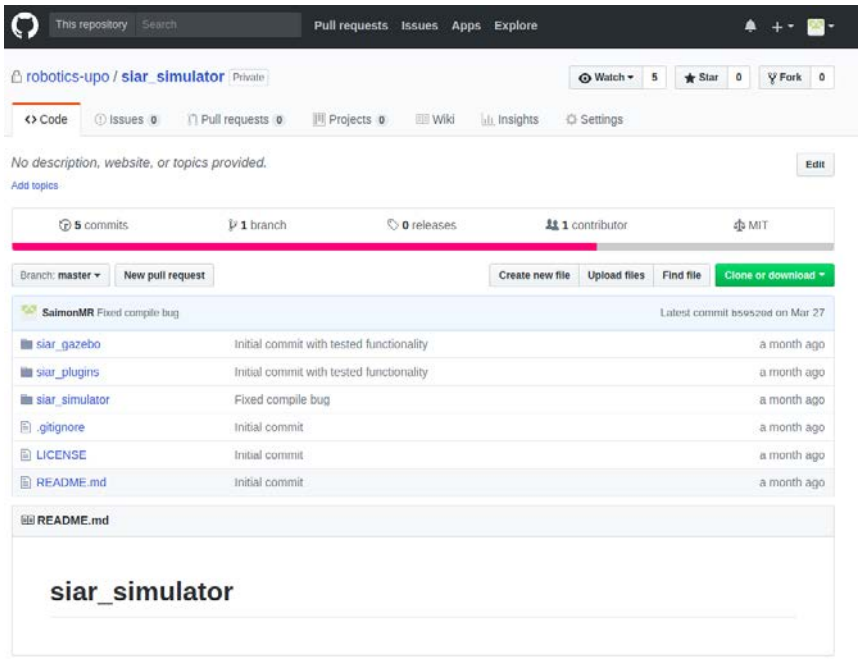


Figura 5. 31 Imagen del repositorio en GitHub de robotics-upo donde se visualiza los packages de la simulación.

6. RESULTADOS OBTENIDOS – SIMULACIONES

6.1. Introducción

En el presente capítulo se muestran los resultados obtenidos de la simulación de la plataforma SIAR en Gazebo. Los resultados se dan a conocer a través de imágenes en diferentes situaciones de simulación, recordando que el modelo es controlado de forma manual a través de un mando.

6.2. Primeras Simulaciones: Comportamiento del robot SIAR.

Antes que todo, como muestra la Figura 6.1, se debe ingresar a la carpeta donde se encuentran los paquetes desarrollado según lo mencionado en la sección 5.4 y ajustar las variables del entorno, lo cual nos da acceso a lanzar diferentes archivos desarrollado para este trabajo. En particular se muestra en el presente "siar_model.launch" y "siar_Joy_model.launch" los cuales son los archivos oficiales para probar el simulador sin entorno y con entorno, ya que el resto son archivos de pruebas. Una vez seleccionado el archivo "launch" se ejecuta y Gazebo comienza a ejecutarse, Figura 6.2.

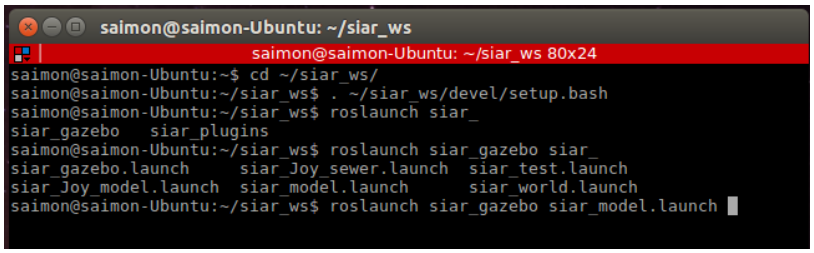


Figura 6. 1 Descrición en consola de los packages y elementos del workspace siar_ws

Figura 6. 2 Presentación de la inferfaz Gazebo al ser lanzado

Así, tenemos acceso al primer archivo "launch" realizado para mostrar el funcionamiento del simulador. "siar_model.launch" solo muestra la simulación de la plataforma SIAR en un entorno vacío, Figura 6.3, pero donde puede visualizar sin inconvenientes las características de desplazamiento y de ajuste de posición de los brazos, lo cual como se mencionó en la secciono en la sección 5.8 es controlado manualmente por un joystick a través de la interconexión de tres nodos ROS, Figura 6.4. El nodo "/joy", se obtiene como consecuencia de haber instalado el "package" ros_kinetic_joy y establece la comunicación entre el joystick y ROS. El nodo "/siar_model" se obtiene como resultado de implementar el plugin "siar_joy_node", es decir, aquel que realiza la codificación de los mensajes enviados por el joystick a mensajes entendibles

por Gazebo. El tercer nodo “Gazebo” es el resultado de implementar el plugin “plugin_siar_wheels_piston.cpp”, el cual se encarga que enviar mensajes para ejecutar el desplazamiento del robot o el movimiento de los brazos en el entorno Gazebo.

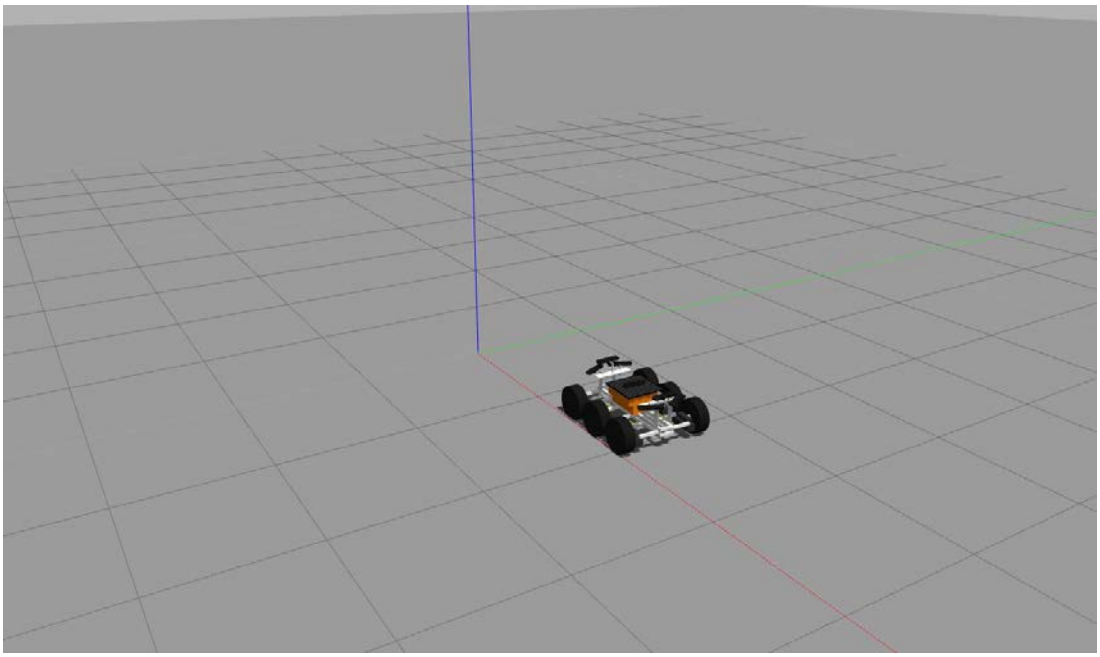


Figura 6. 3 Visualización en Gazebo del robot SIAR simulado al lanzar siar_model.launch

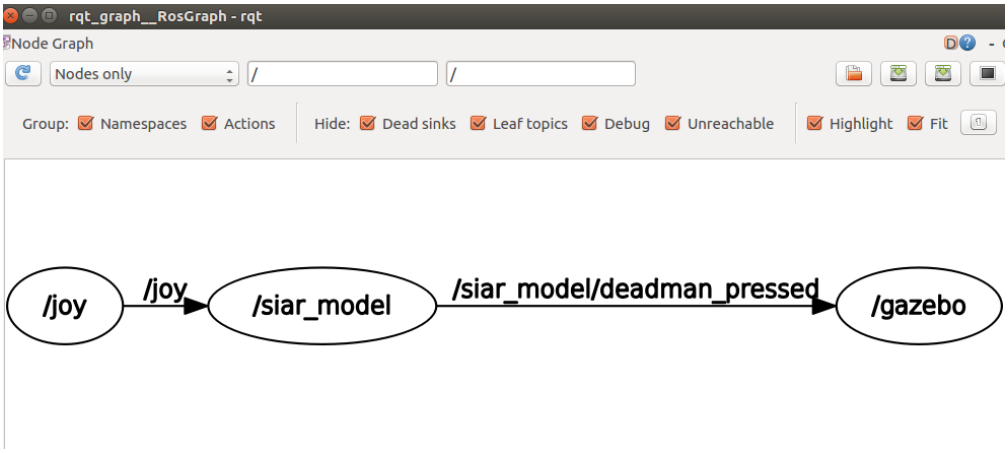


Figura 6. 4 Relación de nodos al lanzar siar_model.launch

A continuación, se muestra el comportamiento del desplazamiento del robot controlado a través de joystick, el cual posee una gran estabilidad para movimientos lineales, Figura 6.5 y 6.6, al hacer giros, Figura 6.7 y 6.8, además al momento de frenar, Figura 6.8, de manera muy similar a la plataforma real.

Recordar, según lo mencionado en sección 5.8 que para que activar el control manual es necesario presionar el botón “LB” del joystick, luego realizar mover el análogo 1 en dirección del eje 1, es decir arriba o abajo, para generar un movimiento hacia adelante o atrás respectivamente.

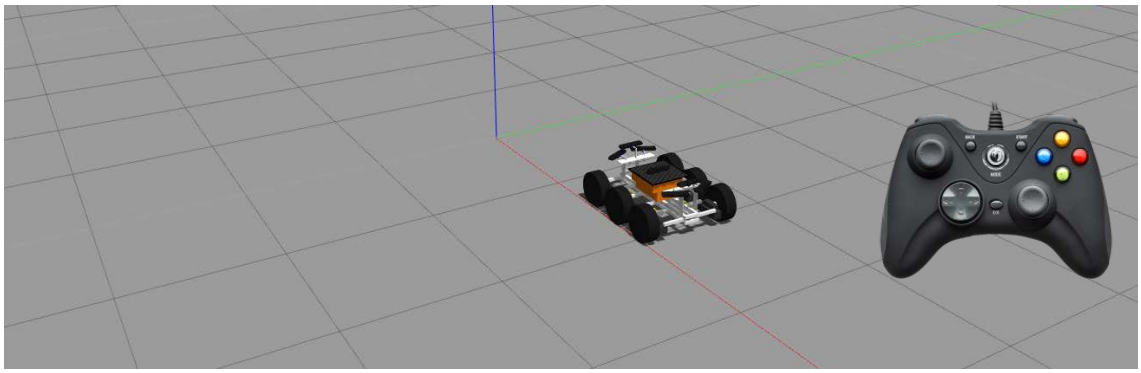


Figura 6. 5 Modelo robot SIAR simulado en Gazebo sin presionar botones generando un estado de reposo

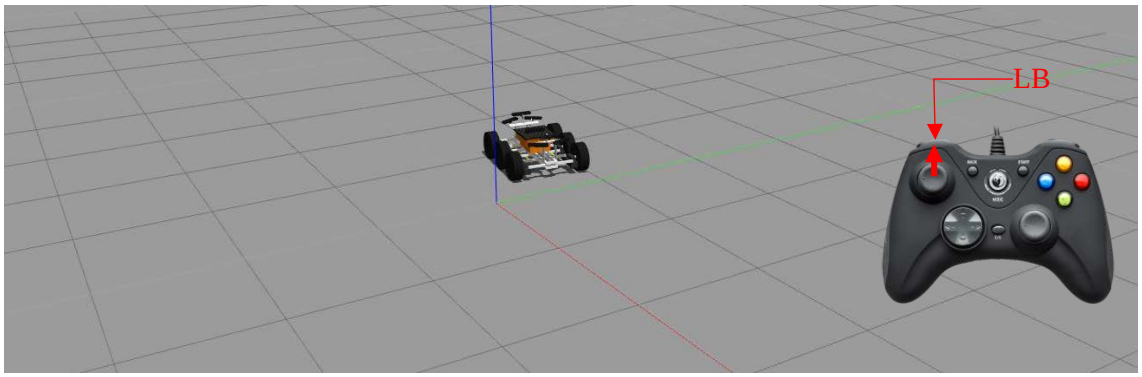


Figura 6. 6 Modelo robot SIAR simulado en Gazebo al presionar boton 4 (LB) y eje 1 (hacia arriba) generando movimiento lineal

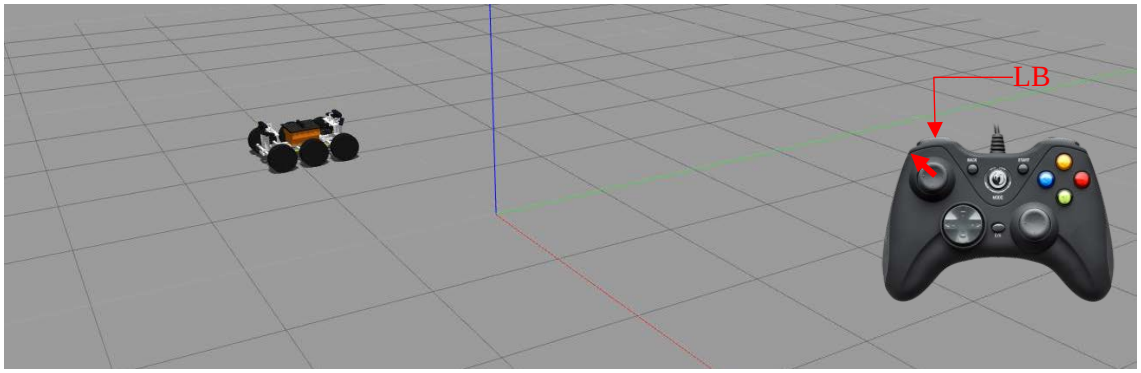


Figura 6. 7 Modelo robot SIAR simulado en Gazebo al presionar boton 4 (LB), eje 1 y eje 0 generando movimiento rotacional.

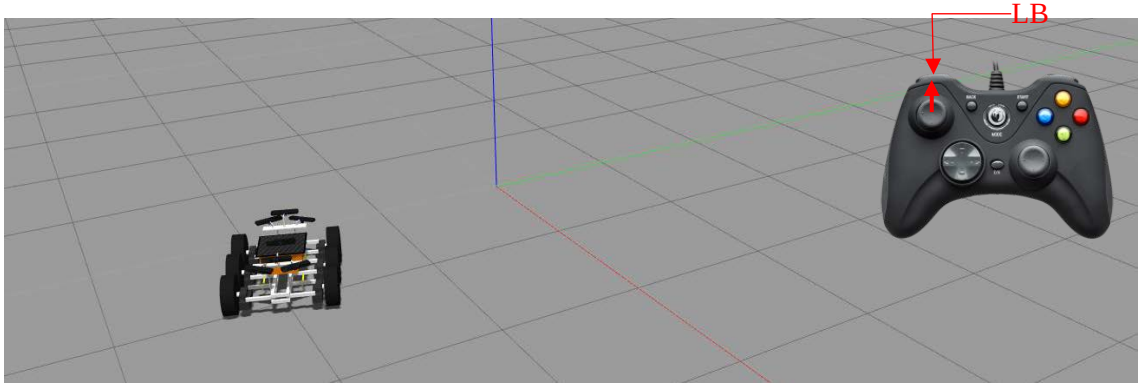


Figura 6. 8 Modelo robot SIAR simulado en Gazebo luego de girar al presionar boton 4 (LB) y eje 1 (hacia arriba) generando movimiento lineal

También se evaluó en el modelo, el estado de los centros de masas, joints e inercias, para conocer la ubicación geométrica de cada una de estas partes que permiten obtener una mejor estabilidad del modelo simulado. La Figura 6.10 muestra la ubicación de los centros de masas en el modelo, los cuales, y por la geometría del robot, se encuentras distribuidos simétricamente respecto al centro del robot, lo cual genera una buena estabilidad, ya que no hay que compensar masas para que el robot se mantenga en equilibrio. Por otra parte, se revisar el estado de los joints, Figura 6.9, los cuales también se encuentras simétricamente distribuidos respecto al centro del robot, esto también favorece a la estabilidad ya que los puntos de giro son simétricos por lo tanto es más simple obtener un balance de las masas de los cuerpos. También de evaluó el estado final de las inercias, Figuras 6.11, 6.12 y 6.13, el cual como se mencionó en las secciones anteriores 5.3 y 5.6 tiene una representación coherente por parte de los links del robot y por parte de las cámaras es enorme, aun así, la estabilidad se mantiene sin generar efectos adversos a la simulación, más bien, estabiliza mucho más la simulación.

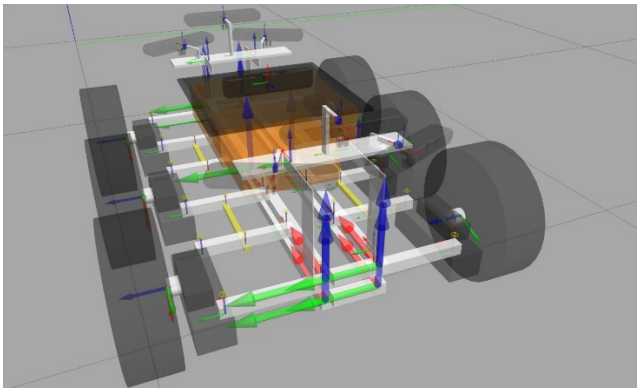


Figura 6. 9 Distribucion de los joints en el modelo del robot SIAR simulado Gazebo

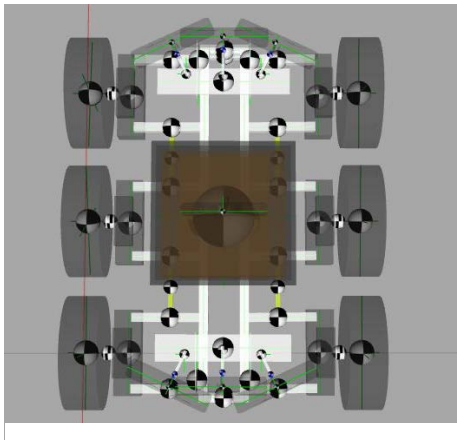


Figura 6. 11 Distribucion de centros de masas en el modelo del robot SIAR

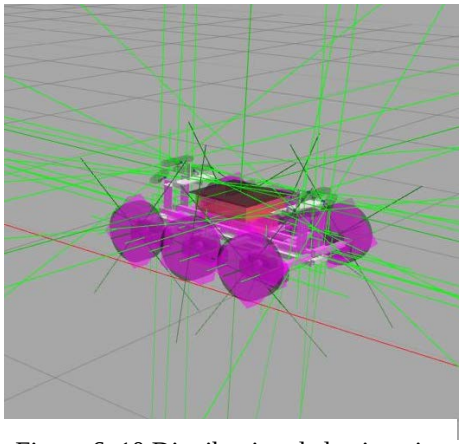


Figura 6. 10 Distribucion de los inercias en el modelo del robot SIAR

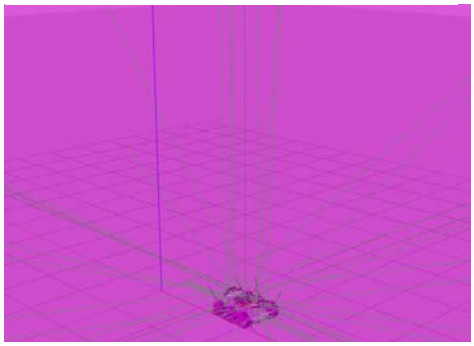


Figura 6. 13 Inercia total del modelo simulado vista desde cerca en Gazebo

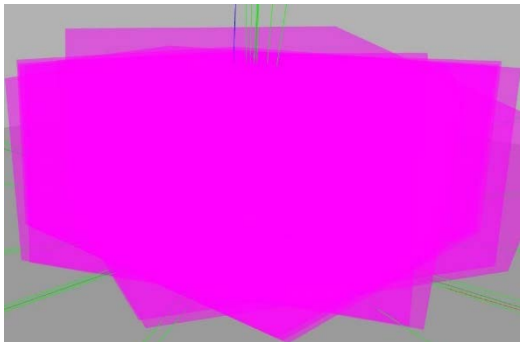


Figura 6. 12 Inercia del modelo vista desde lejos en Gazebo

Para conocer cuáles son los mensajes (messages) que se envían a través de los distintos nodos de ROS y que ofrecen información sobre la simulación, se ejecuta el comando “rostopic list” el cual muestra una lista de los topics tal como se visualiza en la Figura 6.14. Se puede apreciar cómo según la sección 3.3.1.2, se publica información del modelo, lo cual permite conocer su estado o ejecutar acción sobre él.

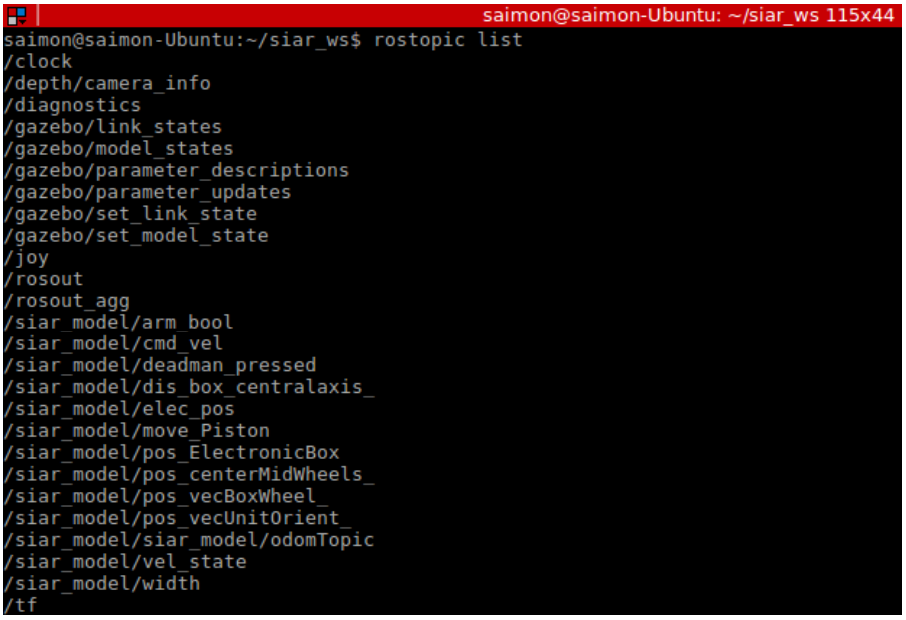


Figura 6. 14 Visualización de topics del modelo en consola al lanzar siar_model.launch

En particular se analiza los topics “/siar_model/width” y “/siar_model/dis_box_centralaxis” los cuales permiten conocer respectivamente, el ancho del modelo en cada instante y la distancia que existe entre el centro del modelo y el centro de la caja de la electrónica (con su sentido hacia adelante o atrás, es decir, valor positivo y negativo respectivamente). Comparando los valores medidos en la plataforma real del SIAR, Tabla 6.1, con los medidos en el simulador, Figuras 6.15, 6.16, 6.17, 6.18 y 6.19, se aprecia que la geometría final obtenida en el simulador representa muy bien al robot real, ya que las lecturas realizadas en los topics muestran valores de dimensiones muy similares.

Tabla 6. 1 Mediciónes realies de la plataforma SIAR

Distancia centro a caja electrónica (m)	Ancho (m)	Medida del sensor
0.14	0.51	0
0.12	0.58	10
0.09	0.64	20
0.06	0.68	30
0.04	0.7	40
0.02	0.71	50
0	0.71	60
-0.01	0.71	70
-0.03	0.7	80
-0.05	0.69	90
-0.07	0.68	100
-0.08	0.66	110
-0.1	0.64	120
-0.11	0.61	130
-0.12	0.58	140
-0.14	0.51	150

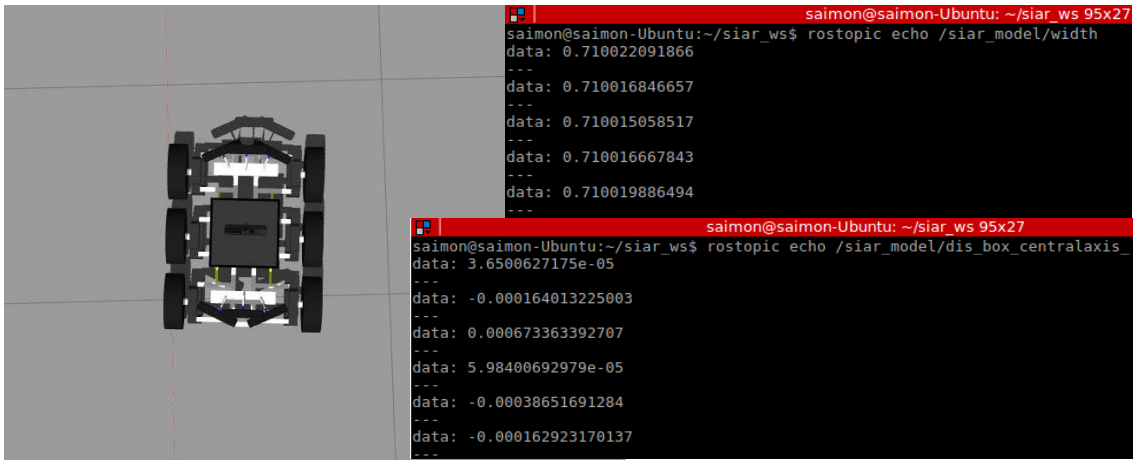


Figura 6. 15 Estado del robot SIAR simulado en Gazebo en ancho maximo

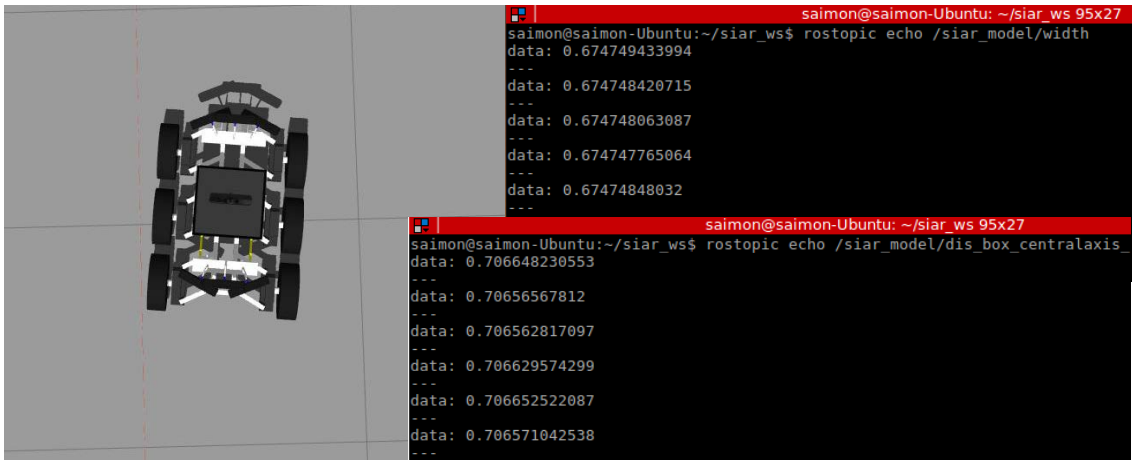


Figura 6. 16 Estado del robot SIAR simulado en Gazebo con electronica desplazada hacia adelante

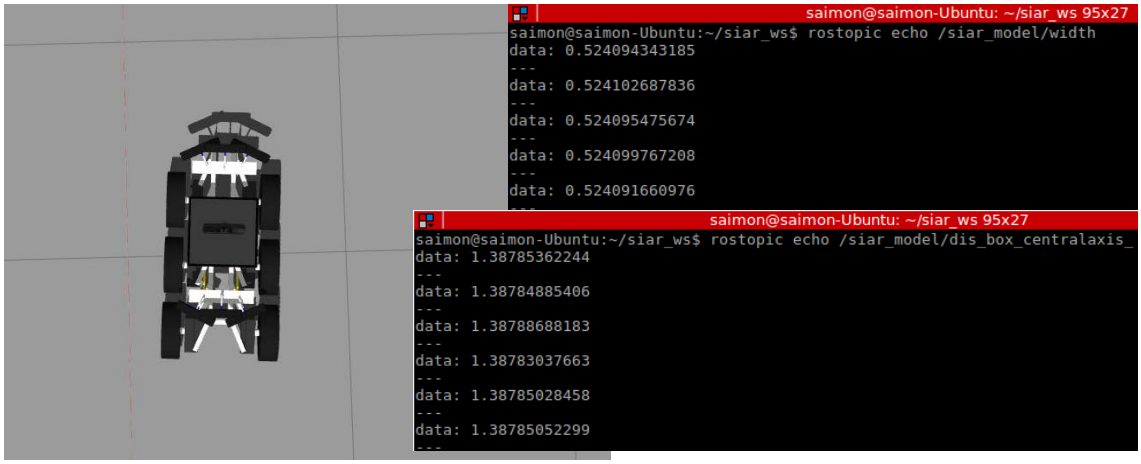


Figura 6. 17 Estado del robot SIAR simulado en Gazebo con electronica desplazada hacia adelante al ancho minimo.

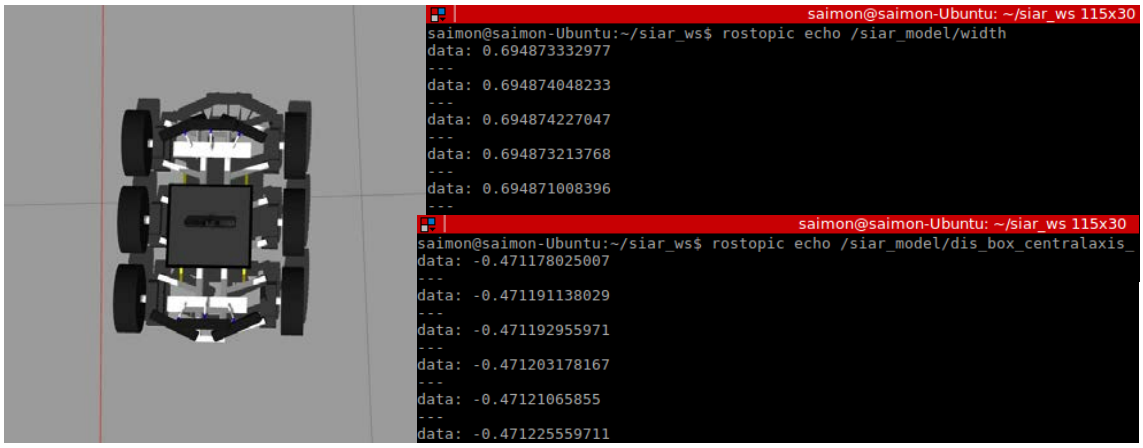


Figura 6. 18 Estado del robot SIAR simulado en Gazebo con electronica desplazada hacia atrás

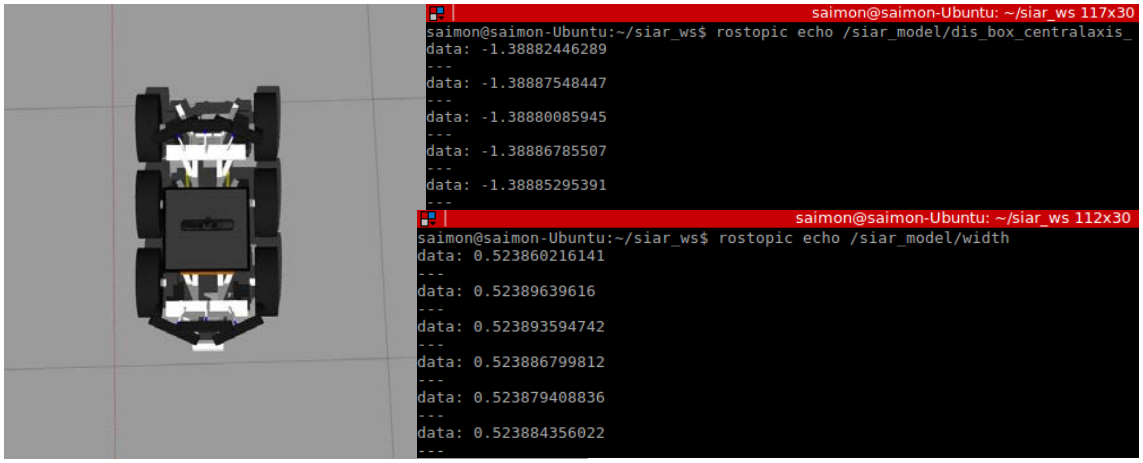


Figura 6. 19 Estado del robot SIAR simulado en Gazebo con electronica desplazada hacia atras al ancho minimo.

6.3. Segundas Simulaciones: Interacción de simulación SIAR con entorno

Como se ha mencionado, la plataforma SIAR fue desarrollada para realizar trabajos de inspección en alcantarillado, como muestra la Figura 6.20 y 6.21, razón por la cual en esta sección se muestra los resultados de la simulación realizados en un entorno similar al real.

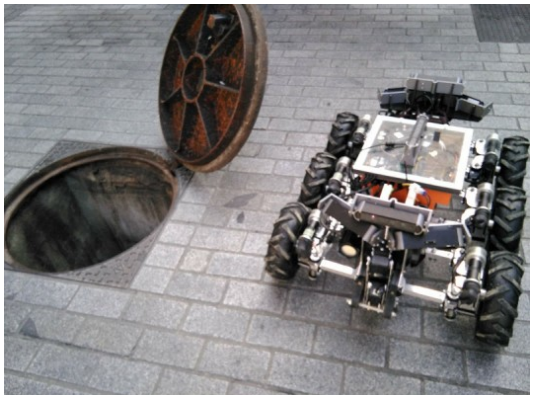


Figura 6. 21 Plataforma SIAR previa inspección a alcantarilla

Figura 6. 20 Plataforma SIAR en tareas de inspección dentro de alcantarilla

Para lanzar el archivo launch debe ejecutarse en consola “siar_Joy_model.launch” tal como muestra la Figura 6.22, lo cual da acceso a la simulación de la plataforma SIAR en un entorno con un alcantarillado, Figura 6.23. y 6.24, recreado a través de las técnicas de mapping mencionados en la sección 5.7. Este entorno ofrece al modelo simulado en un entorno similar al real, con el fin de comprobar el desempeño del robot en este tipo de entornos, comprobar el comportamiento de cámaras simuladas y cargar tf para futuro desarrollos de algoritmos de control autónomo de desplazamiento y movimientos de brazos al navegar por el alcantarillado.

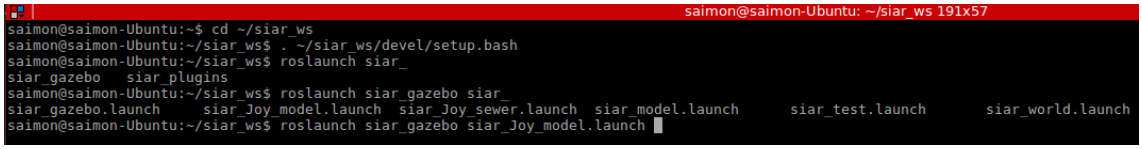


Figura 6. 22 Visualización de topics del modelo en consola al lanzar siar_Joy_model.launch

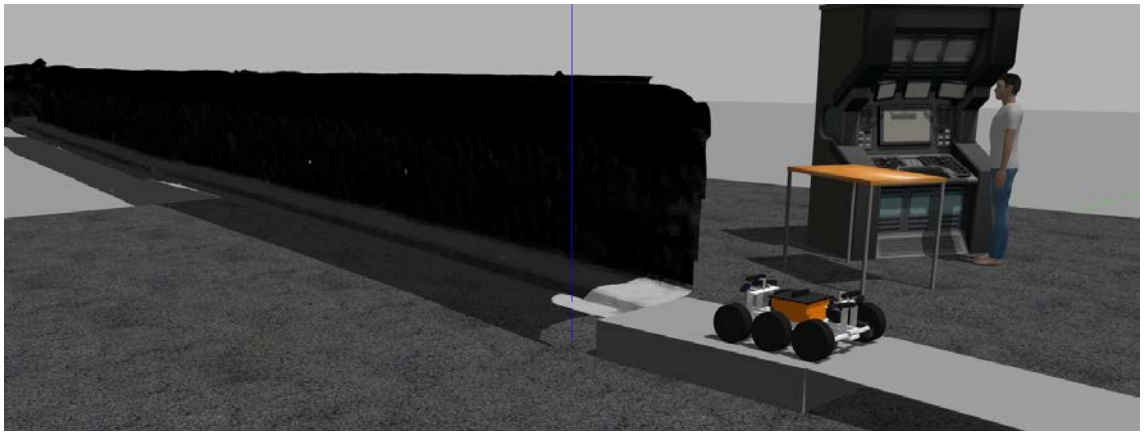


Figura 6. 23 Visualización cercana del entorno de simulacion desarrollado conla integración de mapa de alcantarillado y modelo robot SIAR.

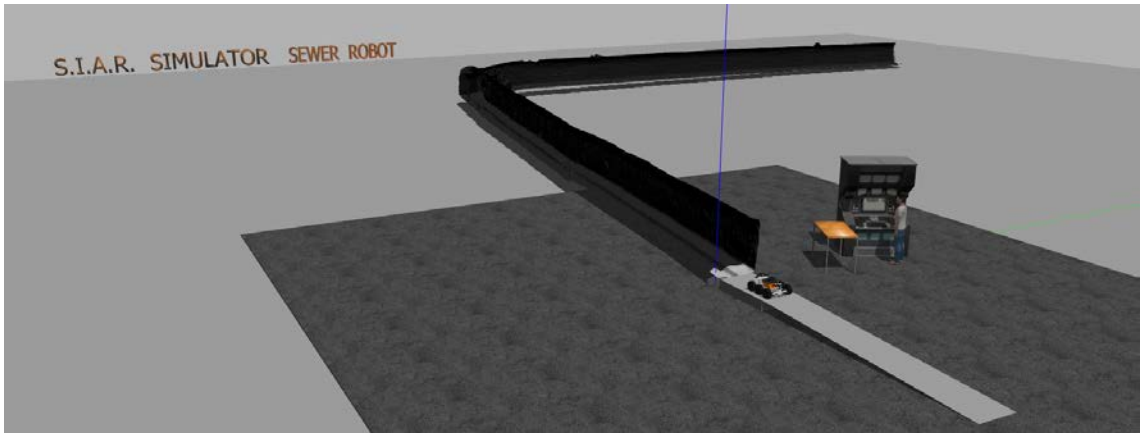


Figura 6. 24 Visualización completa del entorno de simulacion desarrollado conla integración de mapa de alcantarillado y modelo robot SIAR.

Las primeras pruebas realizadas en el entorno tienen como objetivo comprobar la estabilidad del modelo al someterse diferencia de niveles, como la rampa de acceso al alcantarillado, y observar si los motores físicos generan comportamiento indebido como puede ser un giro o una caída abrupta. En las Figuras 6.25 y 6.26 se observa como el robot se mantiene sobre la rampa a pesar de que algunas partes de él se encuentran en el aire. Esto demuestra como la distribución de masa está bien realizada lo cual permite mantener el robot en la rampa hasta el momento en que la mitad del robot sale de ella como en la Figura 6.27. Mientras que en la Figura 6.28 se observa como el modelo se mantiene colgado de las ruedas traseras sin perder el equilibrio o mostrar un comportamiento anormal, lo cual indica que los centros de masas y las inercias se encuentran bien definidas por lo cual es modelo no tiene un comportamiento errático.

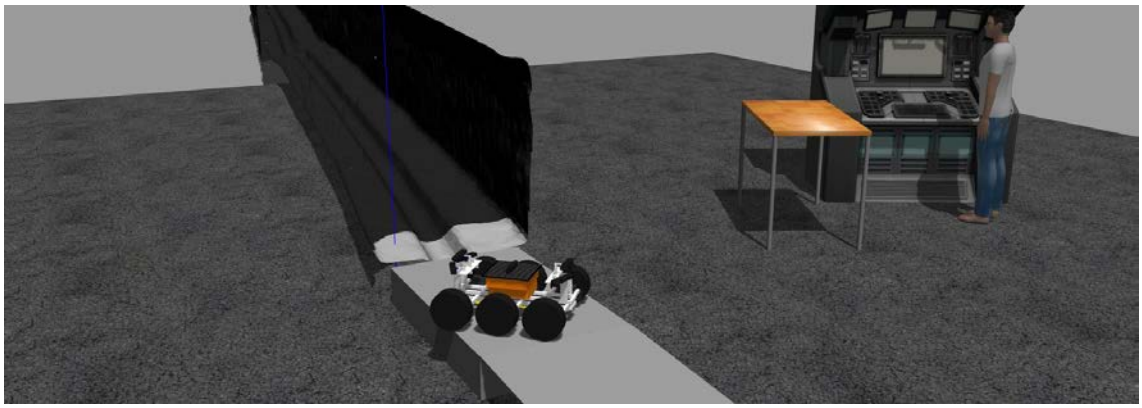


Figura 6. 25 Desplazamiento del robot SIAR simulado sobre plataforma de ingreso al tunel

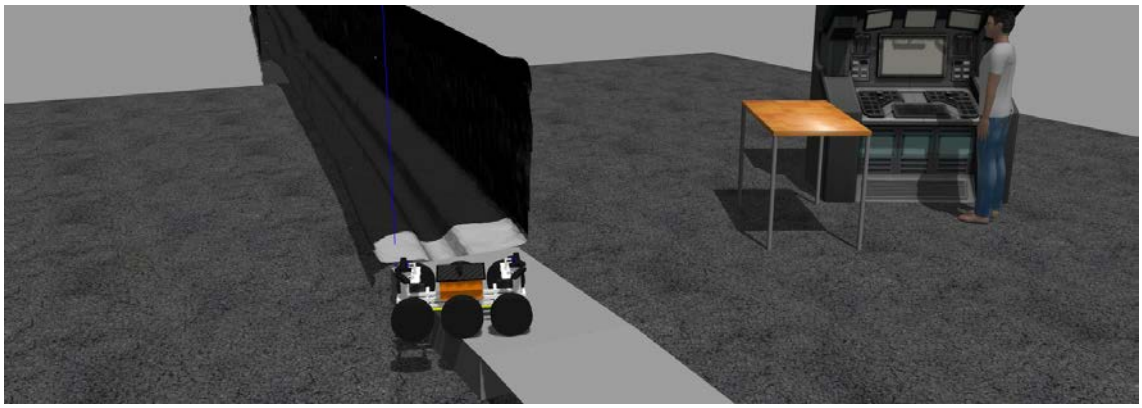


Figura 6. 26 Desplazamiento del robot SIAR simulado con elemento suspendidos en el aire y otros sobre plataforma de ingreso al tunel

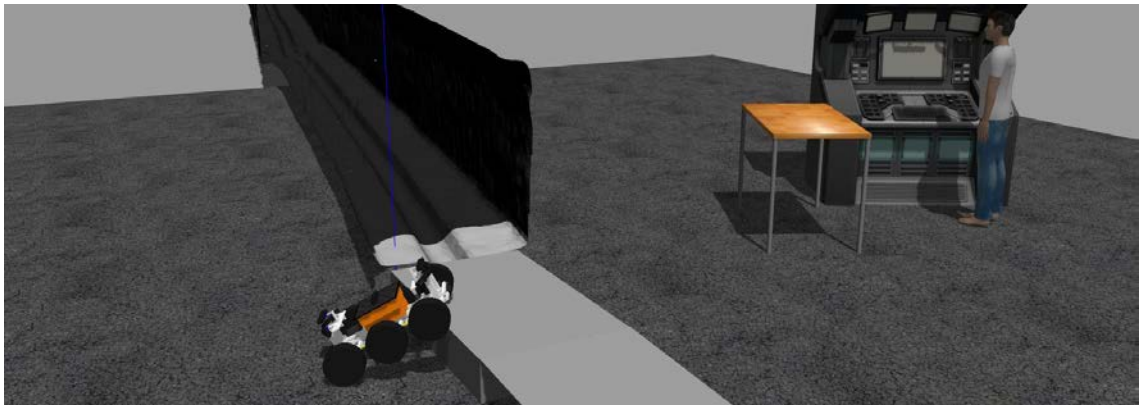


Figura 6. 27 Visualizacion del robot SIAR simulado con respuesta estable al someterse a movimientos bruscos de caida.

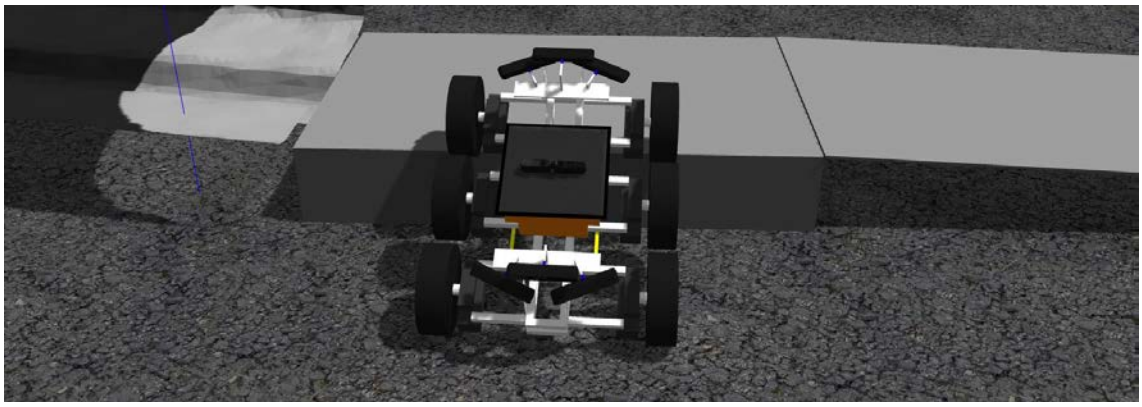


Figura 6. 28 Visualización del robot SIAR simulado con respuesta estable ante movimientos bruscos

Posteriormente, se procedió a dejar caer el robot para posicionarlo completamente en el asfalto, Figura 6.29 con el fin de probar su estabilidad al momento de impactar con el suelo. La respuesta fue favorable y el robot se mantiene estable y no presenta un comportamiento anómalo. Luego se comprobó el comportamiento de desplazamiento del robot a través del asfalto el cual es estable, no existe deslizamiento de las ruedas y el roce se presenta de forma normal, Figura 6.30 y 6.31. Se conduce el modelo hasta el punto donde puede interactuar con otros elementos del entorno, como la mesa, la consola y el hombre, Figura 6.32 y 6.33, con los cuales el comportamiento es normal existe interacción de fuerza entre los diversos cuerpos generando desplazamiento de los elementos con menos masa como la mesa y levemente con el hombre y no en la consola.

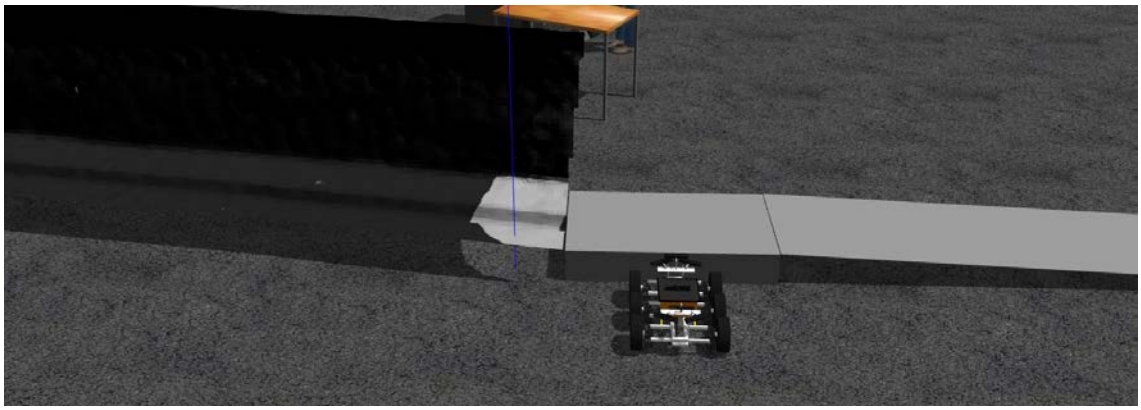


Figura 6. 29 Visualización robot SIAR simulado posicionado en asfalto con respuesta estable despues de caida desde rampa.

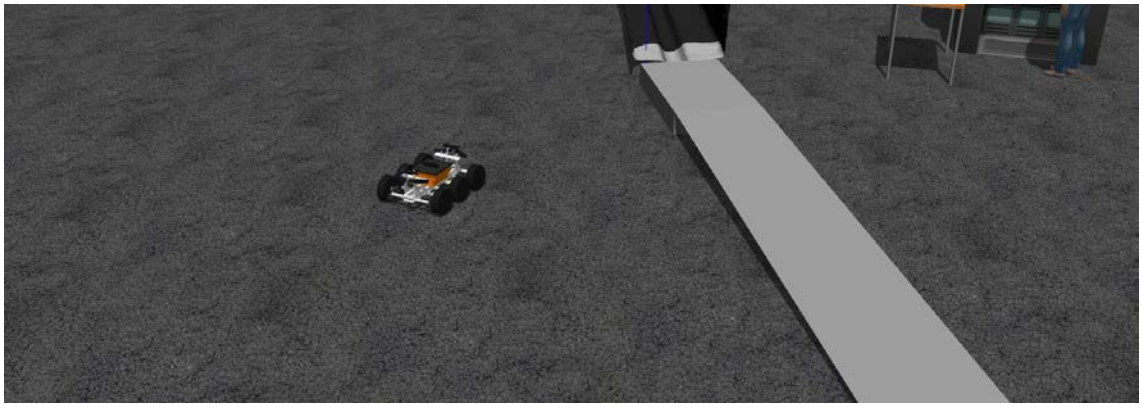


Figura 6. 30 Visualización robot SIAR simulado desplazandose sobre asfalto con respuesta estable a aceleración y tracción.

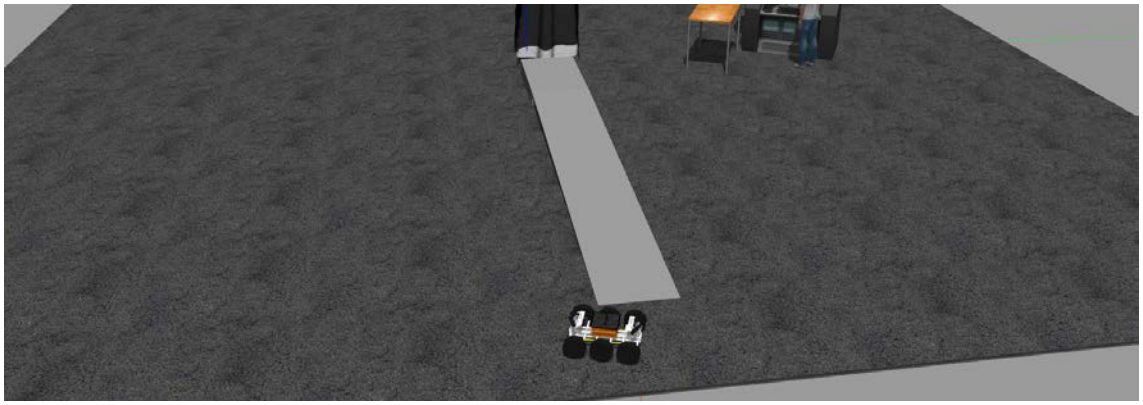


Figura 6. 33 Visualización robot SIAR simulado desplazandose sobre asfalto con respuesta estable de aceleracion y traccion luego de movimientos rotacionales

Figura 6. 31 Visualización de robot SIAR simulado con respuesta estable al interactuar con elementos del entorno (mesa y consola)

Figura 6. 32 isualización de robot SIAR simulado con respuesta estable al interactuar con elementos del entorno (operador)

Con el fin de conocer el fin de mostrar la capacidad de tracción y torque que posee el modelo, se muestra como este afronta y desplaza por la rampa, Figura 6.34, 6.35 y 6.36 que lo posiciona en la entrada del túnel. Esta acción se realiza sin inconvenientes y mostrando un comportamiento realista sin anomalías de deslizamiento o giros en torno a algún eje.

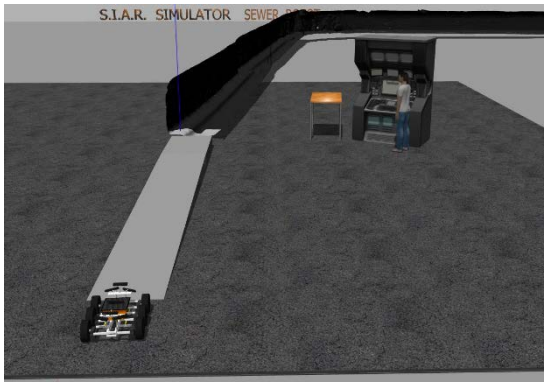


Figura 6. 35 Visualización de robot SIAR simulado al posicionarse en inicio de rampa

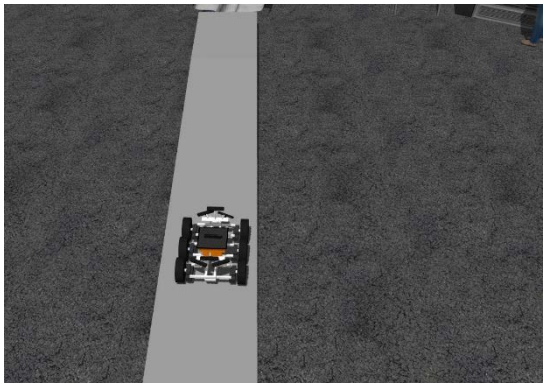


Figura 6. 36 Visualización de robot SIAR simulado al recorrer rampa

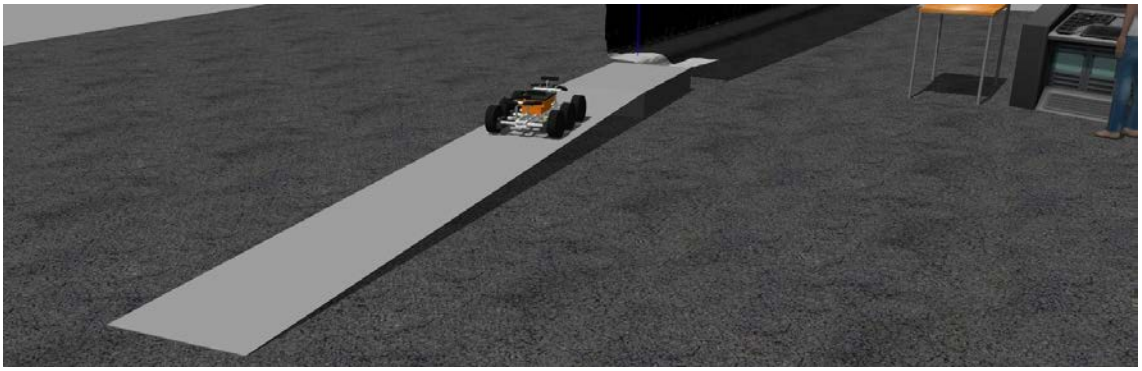


Figura 6. 34 Visualización de robot SIAR simulado al recorrer completamenta rampa

Una vez ubicado frente alalcantarillado, Figura 6.37, se dirige el robot SIAR al interior de este. El primer desafío aparece en la entrada, ya que la reproducción del alcantarillado no tiene superficie plana y paralela al suelo, Figura 6.38, sino que es inclinada teniendo una mayor altura en la cara izquierda, por lo cual el desplazamiento del SIAR no es paralelo respecto al plano si no que con un grado de inclinación. Este detalle es importante para probar el comportamiento del SIAR, ya que esto ofrece una no perfecta zona de desplazamiento a la cual el modelo responde sin problema al avanzar, aunque, con momentáneos deslizamientos, Figura 6.39 y 6.40, que tienen las mismas características a los deslizamientos existentes cuando este está en reposo, lo cual hace suponer que los motores físicos ejercen una acción en el eje z.

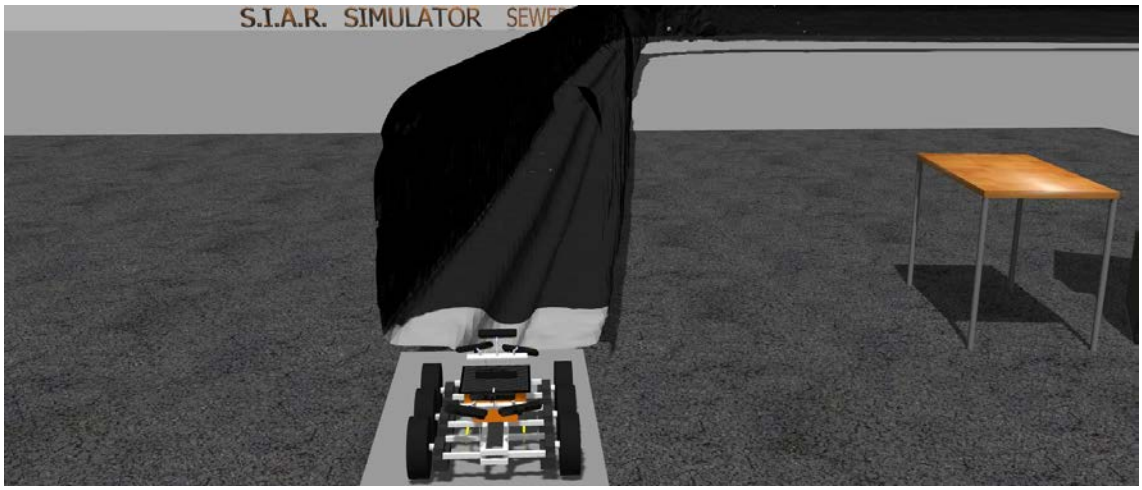


Figura 6. 37 Visualizacion del robot SIAR simulado luego de recorrer la rampa y posicionarse frente al alcantarillado

Figura 6. 38 Visualización del robot SIAR simulado al alcantarillado con respuesta estable a la irregularidad de nivel de la entrada

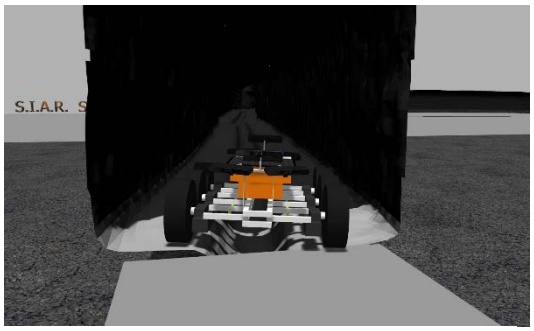


Figura 6. 40 Ingreso de robot SIAR al alcantarillado

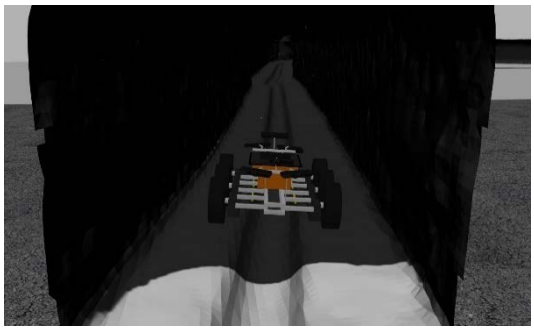


Figura 6. 39 Desplazamiento del robot SIAR al interior de alcantarillado

Figura 6. 42 Desplazamiento del robot SIAR al interior del alcantarillado con ligero deslizamiento

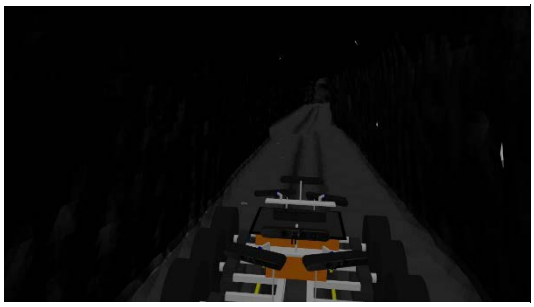


Figura 6. 41 Acercamiento al robot SIAR simulado al interior del alcantarillado

En la Figura 6.41 y 6.42 se observa como el modelo avanza sin mayor inconveniente por el interior del alcantarillado a pesar del deslizamiento que aparece espontáneamente. Cabe destacar que hasta este punto la mayor complejidad ocurre al inicio del trayecto, es decir en el desnivel, momento en el cual hay que ser precavido para no caer en canal, ya que una vez dentro no es posible sacar con el control manual al robot, si no hay que intervenir la simulación. Por otra parte, cabe mencionar que el comportamiento de los brazos es muy establece gracias al control que PID de posicionamiento que posee.

Para conocer más sobre el funcionamiento del simulador, se procede a evaluar el comportamiento de las cámaras Asus Xtion. Para esto se ejecuta en consola “rostopics list” en donde se podrá encontrar una serie una serie de topics correspondiente a las siete cámaras, Figura 6.43 (las tres delanteras, tres traseras, y en el medio). Cabe destacar entre todas las alternativas la relacionadas con “depth” y “rgb”, ya que estas permiten obtener una clara información sobre el entorno y así procesar la información para posibles acciones.

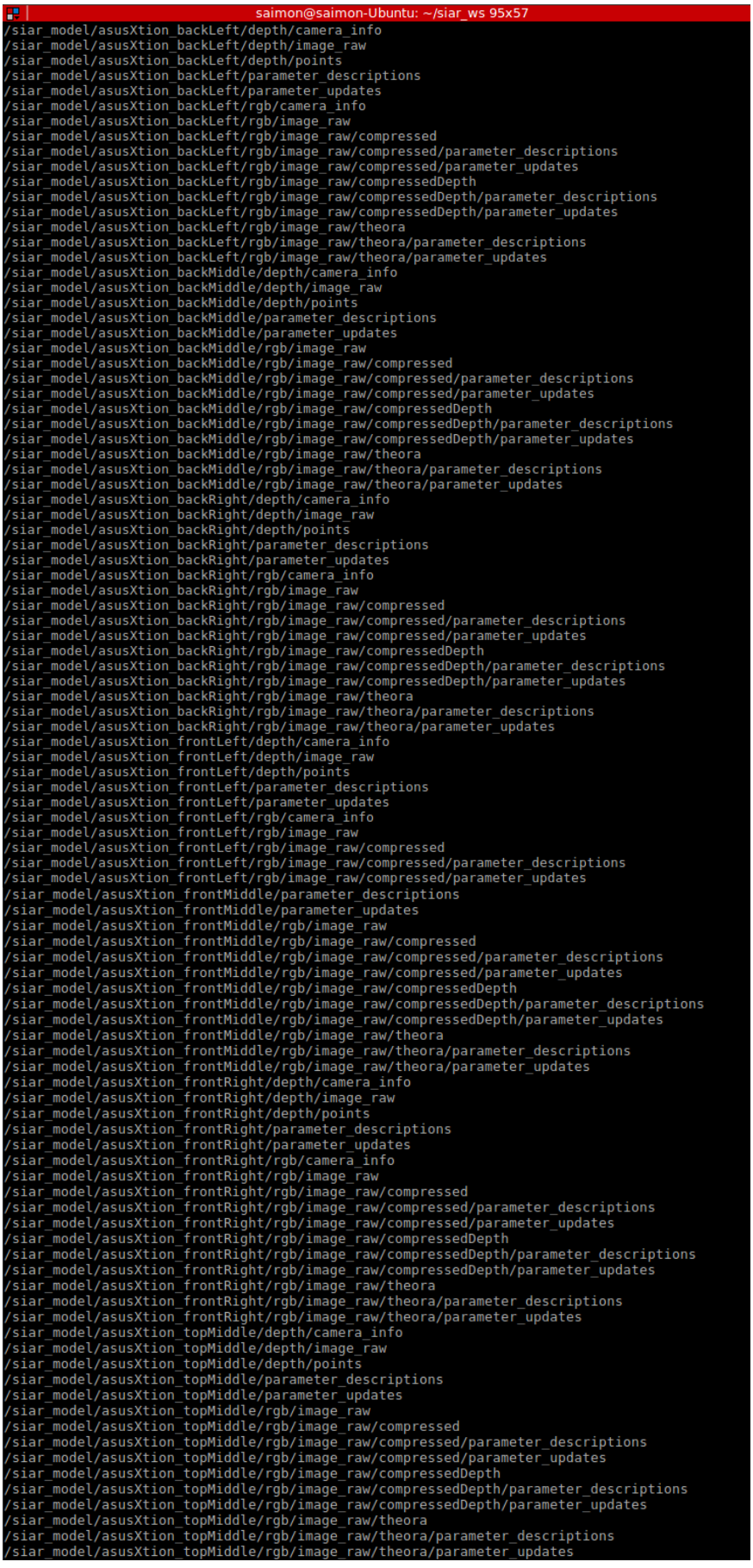


Figura 6. 43 Topics publicados por siar_Joy_model.launch relacionados a las camaras

Para comprobar como las cámaras captan la información se ejecuta el simulador de ROS llamado Rviz en consola, luego se incorporan (add) imágenes y se asocian al topics de la cámara deseada, Figura 6.44. En la Figura 6.45 se aprecia las imágenes “rgb” de las cámaras frontales en orden medio, izquierda y derecha, las cuales ofrecen clara información del interior del alcantarillado. En la Figura 6.45, se aprecia las imágenes “depth" de las medias, izquierda y derecha, las cuales entregan clara información de profundidad del alcantarillado.

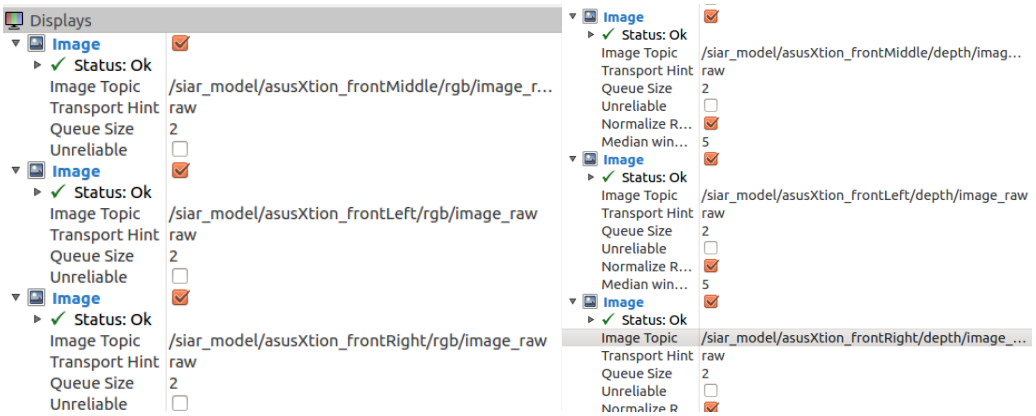


Figura 6. 44 Selección de Topics Camaras frontales en Rviz

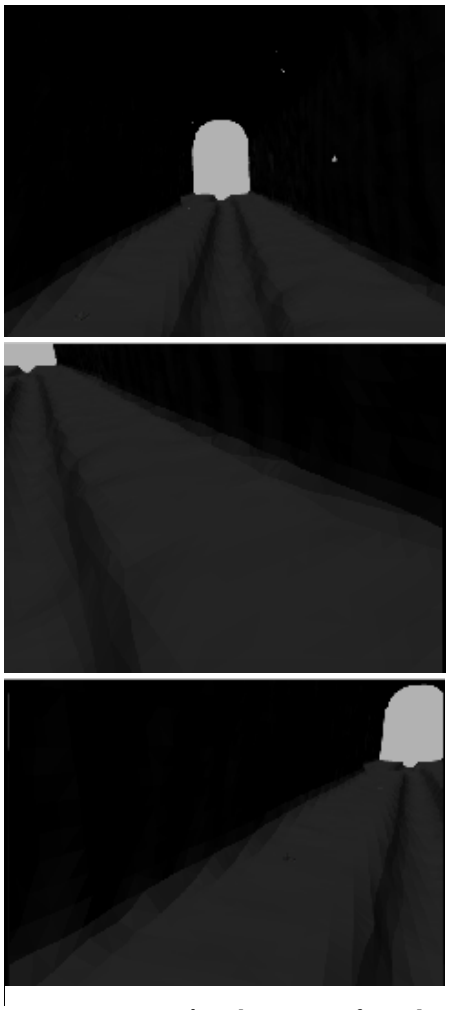


Figura 6. 45 Visión rgb camaras frontales

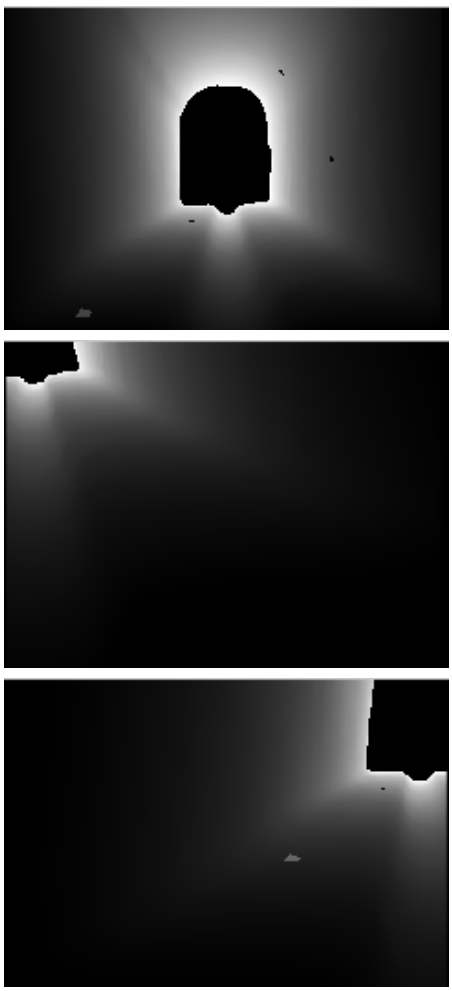


Figura 6. 46 Visión depth camaras frontales

La Figura 6.47 muestra la posición en la que se encontraba el robot en el momento en el que fueron capturadas las imágenes “rgb” y “depth" con las cámaras frontales.

Figura 6. 47 Posición de robot SIAR simulado al momento de realizar la captura de imágenes con cámaras frontales

Del mismo modo, en la Figura 6.48 se muestra la posición en la que se encontraba el robot en el momento en el que fueron capturadas las imágenes “rgb” y “depth" con las cámaras traseras.

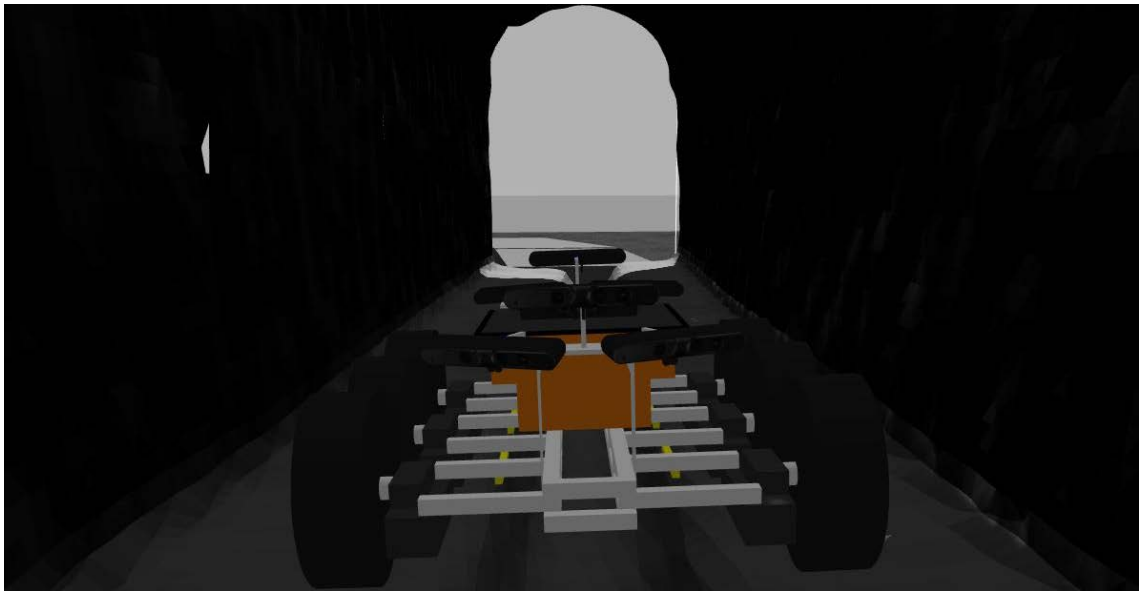


Figura 6. 48 Posición del robot SIAR simulado al momento de realizar la captura de imágenes con cámaras frontales

Luego se procedió a realizar la incorporación (add) de las imágenes asociados a los distintos topics de las cámaras traseras, Figura 6.49. En la Figura 6.50 se aprecia las imágenes “rgb” de las cámaras traseras en orden medio, izquierda y derecha, las cuales ofrecen clara información del interior del alcantarillado. En la Figura 6.51, se aprecia las imágenes “depth" de las caramas medias, izquierda y derecha, las cuales entregan clara información de profundidad del alcantarillado.

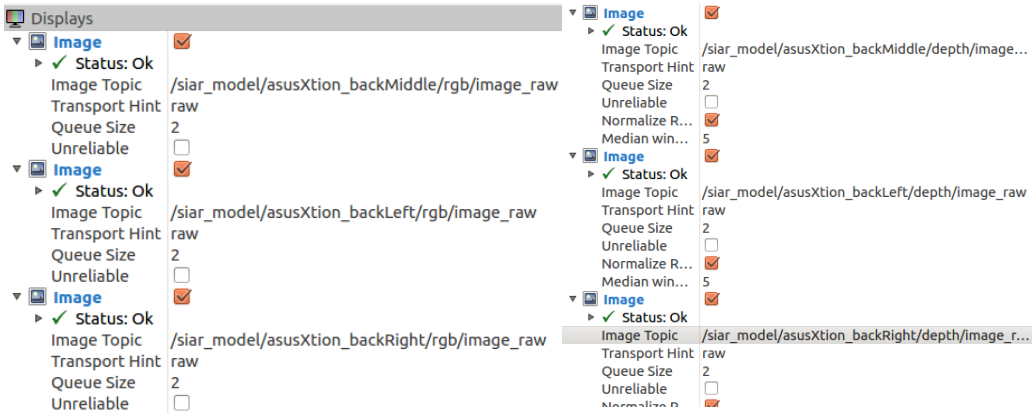


Figura 6. 49 Selección de Topics Camaras traseras en Rviz

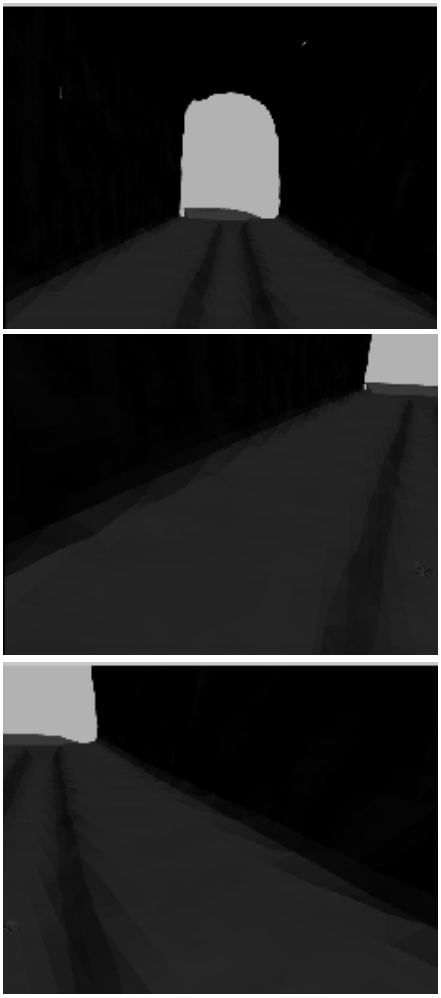


Figura 6. 51 Visión rgb camaras traseras

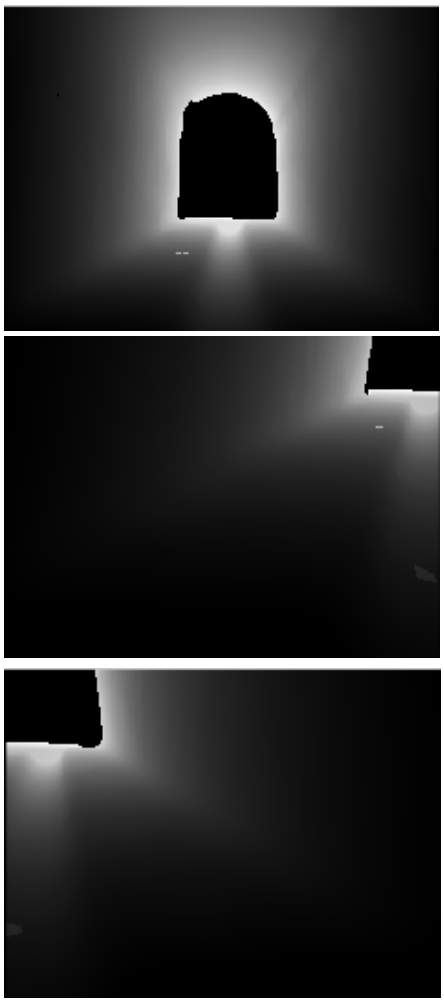


Figura 6. 50 Visión depth camaras traseras

Continuando con las pruebas que muestran el desempeño del robot en el entorno, se procedió a conducir el robot aún más al interior del alcantarillado, Figura 6.52, y generar una caída de este al interior del canal del alcantarillado, Figura 6.53, con el objetivo de conocer cómo se comporta mecánicamente y si se presente alguna anomalía o efecto extraño de fuerzas que le quiten estabilidad al modelo. Como consecuencia, se puede apreciar en la figura 6.54 como el robot se mantuvo estable y sin comportamiento anómalo al caer al canal, aun así, se condujo un par de

metros dentro del alcantarillado para intentar sacar al robot del canal, pero debido a que este es de gran altura y las dimensiones no permiten realizar maniobras de giro para acomodar el modelo, se utilizaron herramientas de la GUI de Gazebo para posicionar nuevamente fuera del canal como muestra la Figura 6.55.

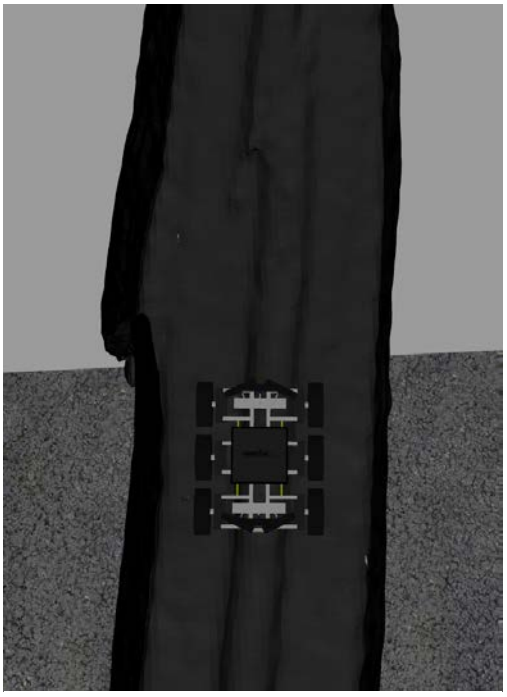


Figura 6. 53 Desplazamiento robot SIAR simulado anterior a caída en el canal

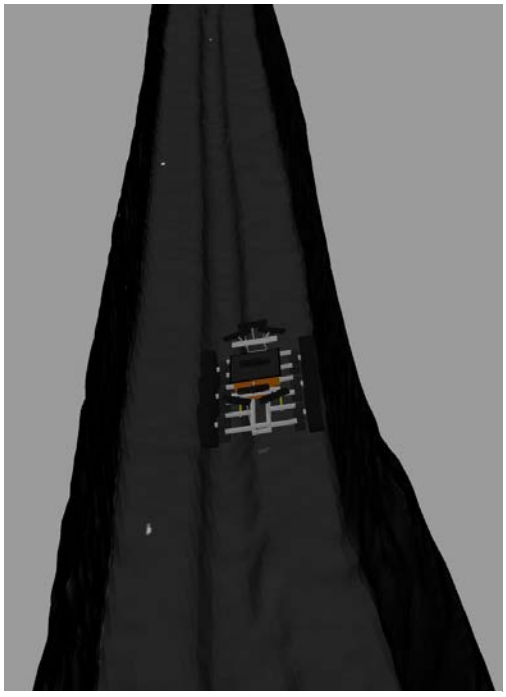


Figura 6. 52 Robot SIAR simulado en el interior del canal del alcantarillado

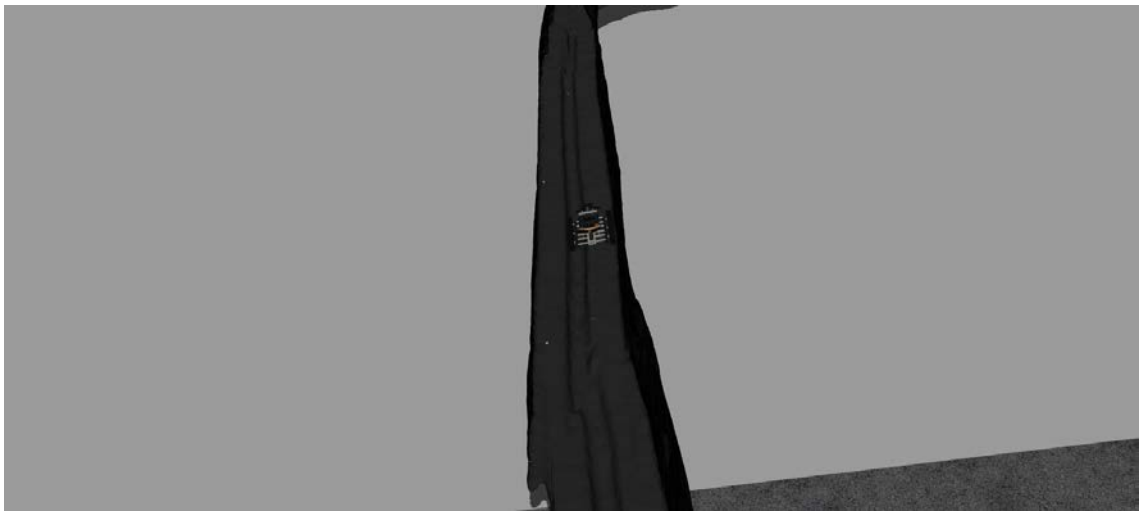


Figura 6. 54 Desplazamiento del robot SIAR por el canal del alcantarillado con respuesta estable ante el desnivel de superficies

Figura 6. 55 Robot SIAR posicionado nuevamente en alcantarillado fuera del canal gracias a la utilización de las herramientas del GUI de Gazebo

Una vez reubicado el robot en el alcantarillado se continuó avanzando por zonas más hostiles presentes en el escenario, Figura 6.56, las cuales, para afrontarlas requieren que el robot desplace el centro de masa del robot, hacia adelante o hacia atrás según sea el caso, a partir del recogimiento de los brazos y así generar mayor tracción en las ruedas delanteras o traseras. En la Figura 6.57 se observa como el robot comienza a mover el centro de masa hacia atrás para conseguir un mayor roce en las ruedas traseras, y así continuar avanzando como muestran la secuencia de imágenes de la Figura 6.58.

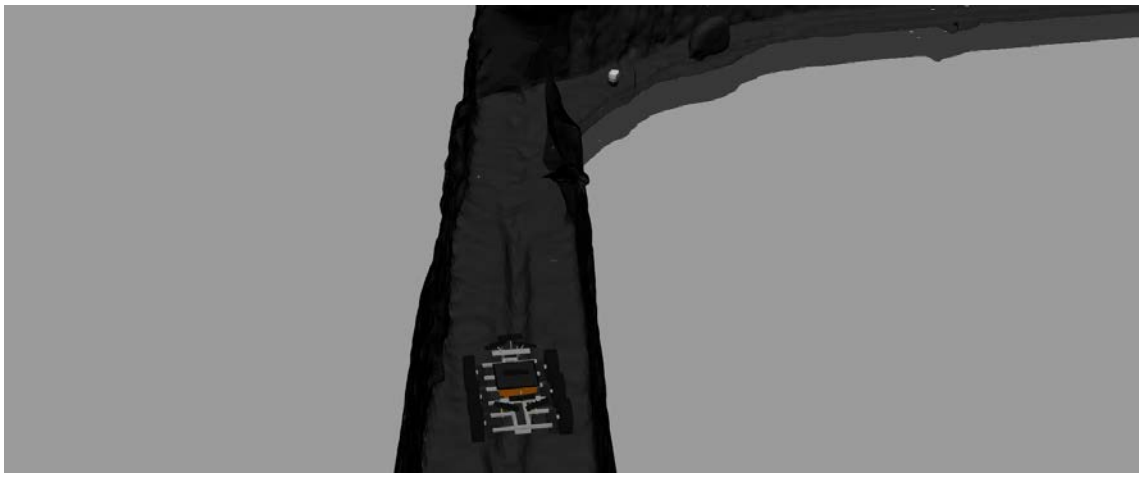


Figura 6. 56 Desplazamiento de robot SIAR simulado por zona hostil del escenario

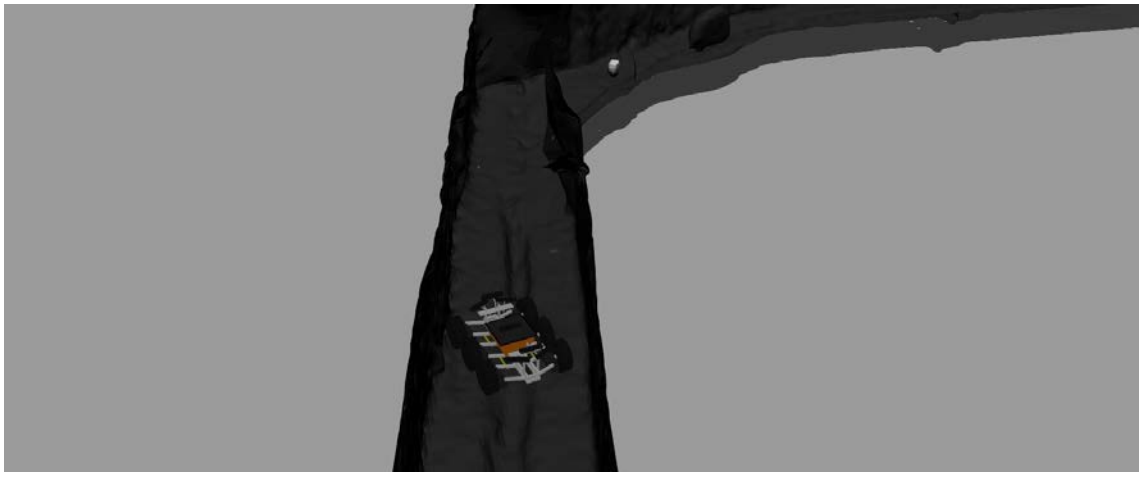


Figura 6. 57 Movimiento de brazos del robot SIAR simulado para modificar centro de masa

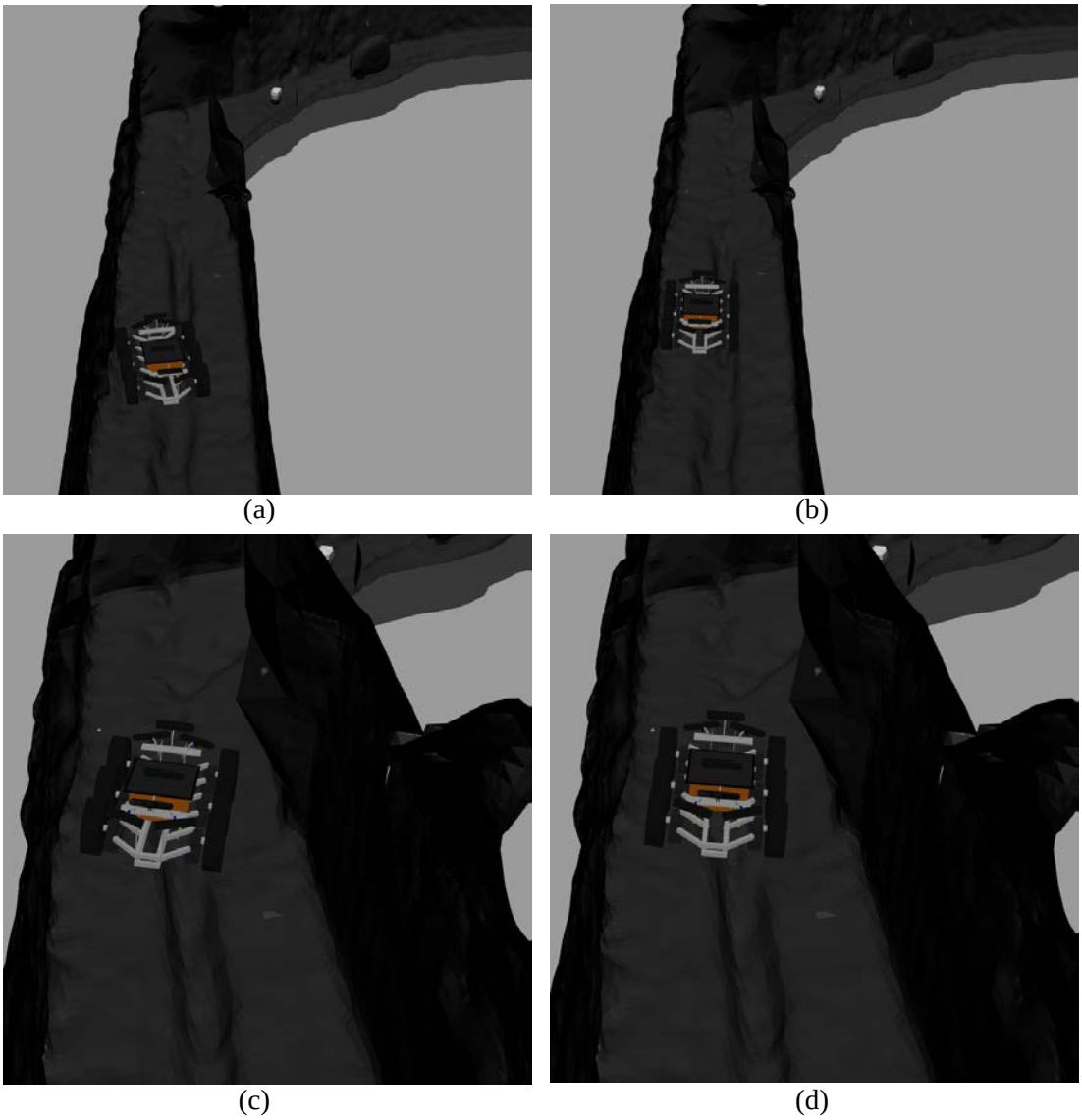


Figura 6. 58 Secuencia de imágenes que muestra la maniobra realizada por robot SIAR simulado para desplazar centro de masa (a), asi generar mejor tracción en ruedas traseras (b) que permita el desplazamiento en zona hostil (c) antes de enfrentar curva (d)

Luego de sortear esta etapa de terreno irregular gracias al desplazamiento del centro de masa, el robot se enfrenta a una zona con curva y posterior angostamiento como muestra la Figura 6.59. Para afrontar esta eventualidad, el robot nuevamente cambia su centro de masa, esta vez hacia adelante, retrayendo completamente los brazos hasta alcanzar el ancho mínimo, y así, contar con mayor roce en las ruedas delanteras. La Figura 6.59 muestra una secuencia de imágenes con lo mencionado anteriormente, comenzando con el cambio del centro de masa desde atrás hacia delante y continuando con este procedimiento hasta tomar la curva, instante en el que alcanza el ancho mínimo y pasa por la angostura del alcantarillado sin inconvenientes para continuar avanzando por este.

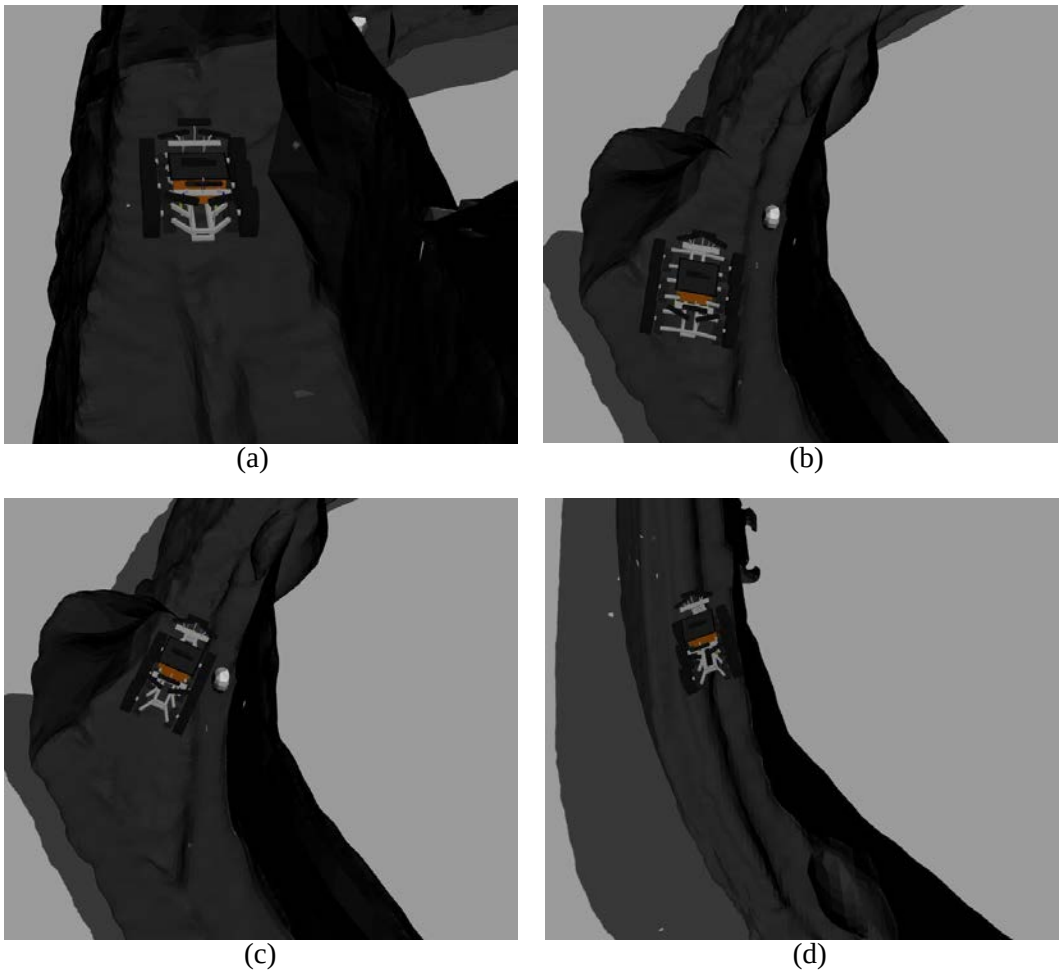
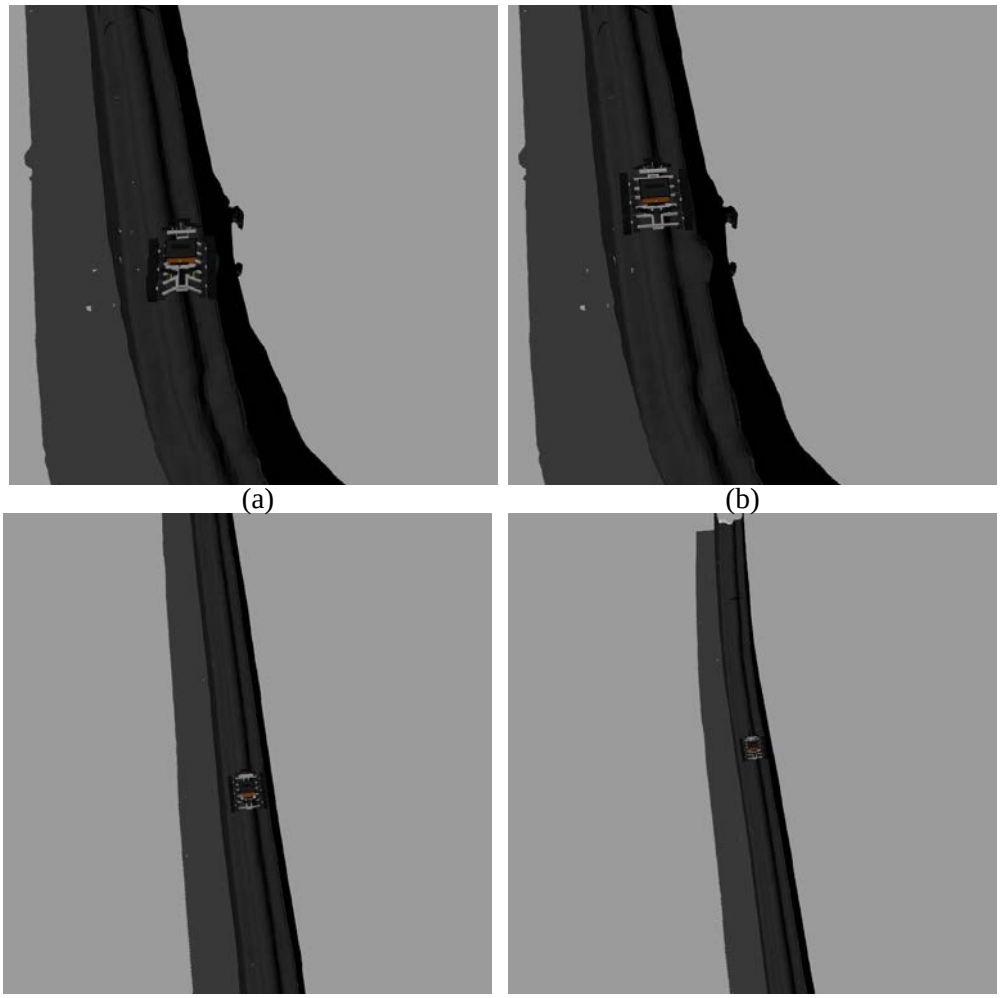


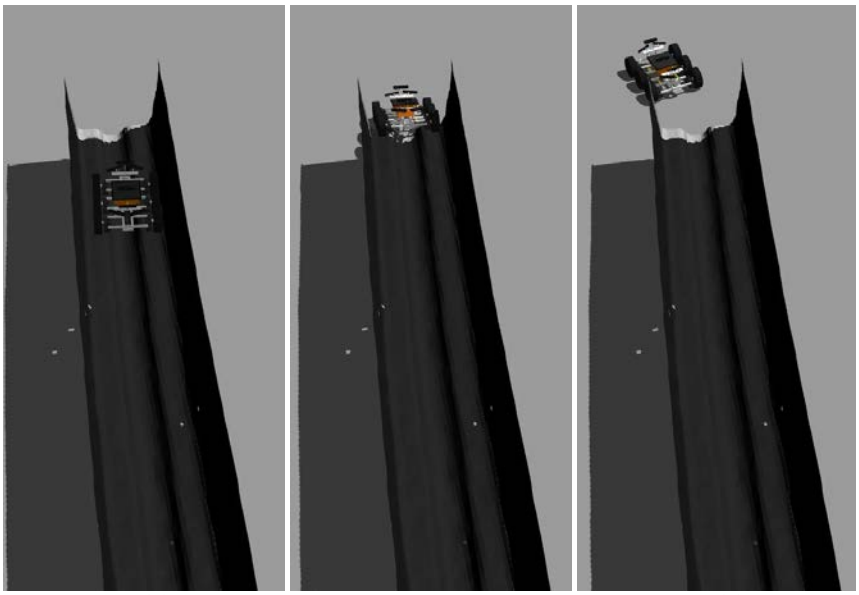
Figura 6. 59 Secuencia de imágenes que muestra la maniobra realizada por robot SIAR simulado desplazando el centro de masa hacia adelante (a) lo cual permite obtener mejor estabilidad y tracción en la superficie hostil antes de ingresar a la curva (b) , lugar donde alcanza el ancho minimo (c) que permite sortear el angostamiento y asi continuar avanzando por el alcantarillado (d)

Posteriormente, como se muestra en la secuencia de imágenes de la Figura 6.60 como el robot comienza a atravesar la última zona del alcantarillado, la cual es una zona recta con un canal ancho en el medio, razón por la cual el robot comienza a mover nuevamente su centro de masa y replegarse completamente hasta alcanzar su ancho máximo y así evitar que caiga al canal. Posterior a eso el control manual solo necesita dirigir hacia adelante al robot.



(a) (b)

Figura 6. 60 Secuencia de imágenes que muestra la maniobra realizada por robot SIAR simulado desplazando el centro de masa hacia el centro (a) lo cual permite obtener mejor estabilidad y no caer al canal (b) para luego avanzar por la zona recta del alcantarillado (c) hasta acercarse a la salida (d)



(a) (b) (c)

Figura 6. 61 Secuencia de imágenes que muestra el acercamiento a la salida del alcantarillado (a) y como el robot SIAR no pierde estabilidad al saltar el desnivel (b) para seguir deslizandose en la supercifie (c)

La última parte del trayecto finaliza con un desnivel, el cual como puede observarse en la secuencia de imágenes de la Figura 6.61, el modelo lo supera sin inconvenientes, mostrando estabilidad al momento de impactar con el suelo y continuando las condiciones de desplazarse después de eso. Además, los brazos no sufren modificación de posición lo que demuestra como el control PID de estos actúa de manera eficiente.

Por último, para culminar con las pruebas realizadas al simulador se evalúa el estado de los tf y si efectivamente estos generan en el programa Rviz. Para esto, se siguen los siguientes pasos como menciona la sección 5.9:

- Lanzar el archivo en consola: “siar_Joy_model.launch”
- Lanzar Rviz: “rosrun rviz rviz”
- Lanzar archivo transformación tf: “roslaunch gazebo2rviz gazebo2rviz”
- Agregar en Rviz (add) los tf

En consecuencia, se tiene lo que se observa en la Figura 6.62, la cual muestra todos los “tf relacionados al entorno simulado. Posteriormente se comenzó conducir el robot por el interior de túnel con el objetivo de visualizar si los tf en Rviz también lo hace. Como muestra la secuencia de imágenes de la Figura 6.63, efectivamente los tf del robot se desplazan con la misma dinámica que en Gazebo por lo cual se puede indicar que Rviz y Gazebo se encuentran correctamente comunicados.

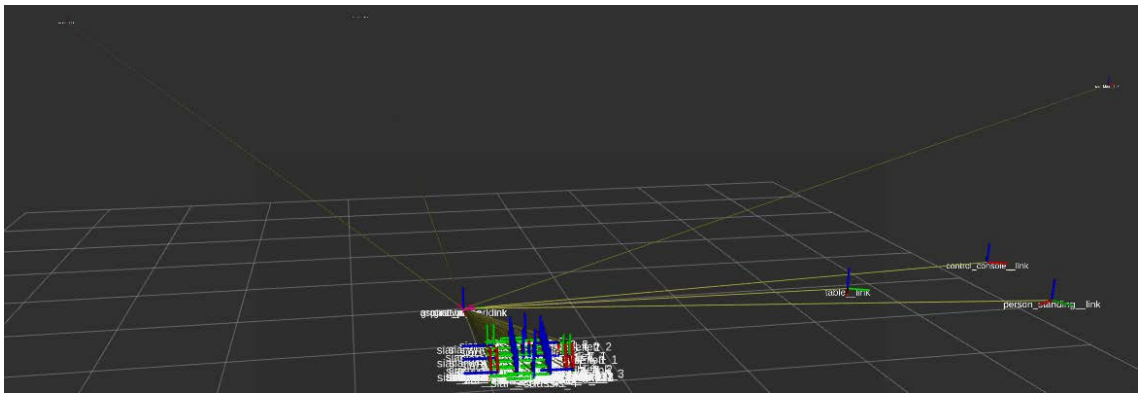


Figura 6. 62 Visualización de tf que representan modelo y entorno simulado

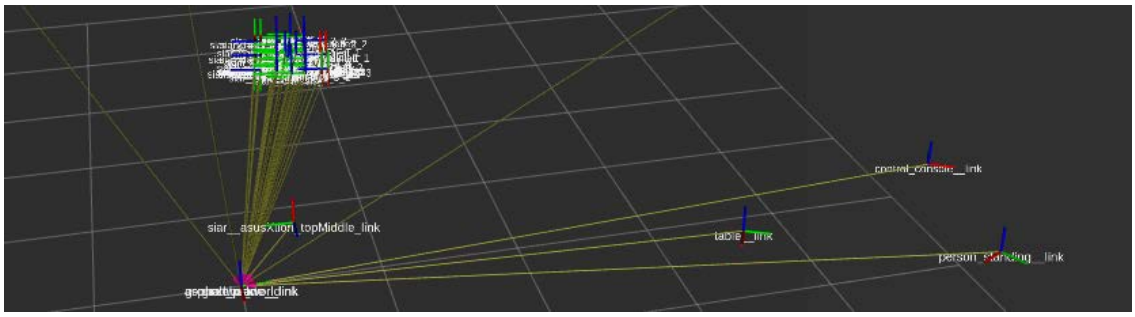


Figura 6. 63 Visualización del desplazamiento de tf del modelo respecto al en el entorno

Por último, la Figura 6.64 muestra a un acercamiento a los tf del robot con el objetivo de visualizar si efectivamente estos tienen movimiento y representan las partes del modelo desarrollado. El resultado es el deseado, los tf giran al mismo tiempo que brazos y/o ruedas y además, elementos del modelo desarrollado.

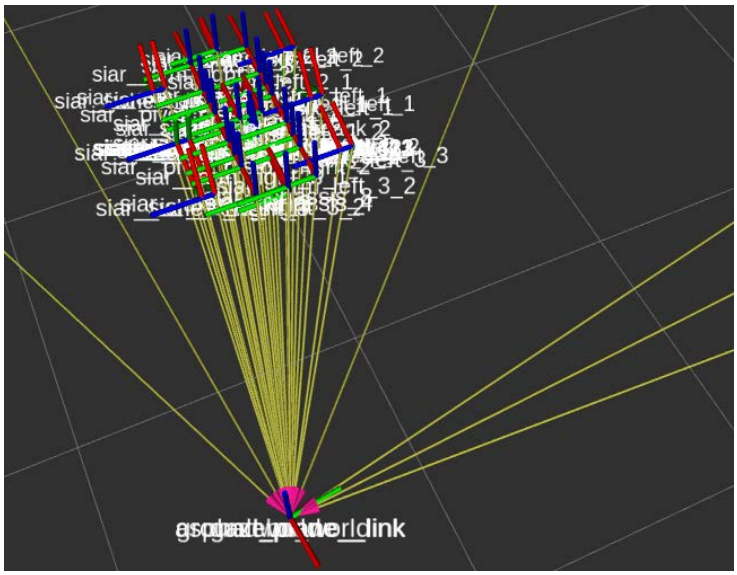


Figura 6. 64 Acercamiento a los tf del modelo SIAR simulado

7. CONCLUSIONES Y FUTURAS MEJORAS

7.1. Conclusiones

En mundo de simulación robótico, es un tema que aún no se satisface a cabalidad, dejando un gran espacio para la insurgencia de nuevos softwares que puedan satisfacer la demanda del usuario en general o bien desde un punto de vista más específico. Otro aspecto importante dentro de los simuladores son las herramientas que los complementan, como por ejemplo los motores físicos o gráficos 3D que permitan crear entorno realista, con propiedades físicas que actúan sobre el robot y que afecten su funcionamiento y dinámica.

Dentro de las tendencias simuladores usados, se identifica una preferencia por aquellos de código abierto y que poseen una comunidad o compañía que respalda constantemente su desarrollo, ya que de esta manera se logra tener retroalimentaciones de otros usuarios y compartir desarrollos que a futuro beneficiaran a otros.

El simulador que cumple con características mencionadas anteriormente corresponde a Gazebo, este es permite el uso de diversos motores físicos, es de código abierto, código, permite modelar multi-robot y de todo tipo (terrestres, humanoides, drones, etc), posee una gran cantidad de información y tutoriales, y cuenta con una comunidad muy activa. Este es soportado por la OSRF, quien se preocupa de mantener mejorando este de forma continua. Además, es posible conectar con ROS, software que, por sus características, especialmente de reutilizar y compartir códigos, cuenta hoy en día con un apoyo generalizado por parte de la comunidad robótica.

Por estas razones de puede indicar que, para el trabajo a realizar, Gazebo es una herramienta que convence y que ofrece una seguridad para concretar el desarrollo del proyecto.

Si bien para desarrollar la simulación y descripción del robot SIAR, se cuenta con la opción de utilizar el formato SDF de Gazebo, o el URDF de ROS, será el primero de estos utilizado. Esto se debe a lo mencionado en [46]. Aquí se comenta que, si bien los URDF son un formato útil y estandarizado en ROS, este carece de muchas características y no se han actualizado para hacer frente a las necesidades cambiantes de la robótica. URDF solo puede especificar las propiedades cinemáticas y dinámicas de un solo robot de forma aislada. URDF no puede especificar la postura del robot en sí mismo dentro de un mundo. Tampoco es un formato de descripción universal, ya que no puede especificar bucles de unión (enlaces paralelos). Además, y carece de fricción y otras propiedades. Tampoco puede especificar cosas que no son robots, como luces, mapas de altura, etc. Por otro lado, la implementación, la sintaxis de URDF rompe el formato adecuado con el uso intensivo de atributos XML, lo que a su vez hace que URDF sea más inflexible. Tampoco posee ningún mecanismo de compatibilidad con versiones anteriores.

Como bien se menciona, el SDF se creó para utilizarlo en Gazebo, y, resolver y satisfacer las deficiencias de URDF. Ya que el SDF describe completamente para todo, desde el nivel mundial hasta el nivel de robot. Es escalable y facilita agregar y modificar elementos. El formato SDF se describe a sí mismo utilizando XML, que facilita una herramienta de actualización simple para migrar versiones antiguas a nuevas versiones. También es autodescriptivo. Además, SDF cuenta con soporte activo por parte de la OSRF.

Por parte de ROS, se tendrán en consideración las herramienta y potencialidad que ofrecen los plugin para comunicar nodos, de esta manera se describirán las tareas robóticas que se desean realizar en los plugin. Dentro de las bibliotecas se considerará la “roscpp” ya que esta es definida como las más utilizada y de alto rendimiento para programación robótica en ROS. Se utilizará la versión de ROS Kinetic Kame, ya que presenta soporte a largo plazo (Lastest LTS) y es compatible con Ubuntu 16.04. En consecuencia, se utilizará también Gazebo 7.0

También cabe mencionar que se desarrollara un “package” para el modelo a realizar siguiendo la jerarquía de archivos.

Las ideas más importantes que se deben tener en consideración para llevar a cabo la simulación de la plataforma SIAR con un comportamiento similar al real, son:

- El modelo simulado debe ser de configuración diferencial, de seis ruedas y con motor independiente en cada eje, con velocidad máxima.
- Cada rueda debe estar conectada al marco central a través de un par de brazos.
- Los brazos que se conectan al marco central deben ser móviles y generar un ancho de máximo de 700 mm y uno mínimo de 500 mm.
- Las ruedas deben permanecer paralelas al marco central en cualquier momento.
- Captar imágenes y profundidad mediante cámaras.
- Una masa de 58 kg.
- Obtener la misma información que procesan los sensores, para futuro desarrollo de algoritmos robóticos en él simulador.
- El mando entrega información a una estación de control, la cual posteriormente se convierte en una ordenen que se envía a la plataforma.
- Simular la plataforma en un entorno similar al real.

Una vez desarrollado el simulador se obtuvieron una serie de conclusiones relacionadas al procedimiento realizado en comparación con los procedimientos sugerido en la documentación previa. Estas diferencias corresponden principalmente a:

- **gazebo_ros_package:** Sin duda una de las diferencias más importantes es la NO incorporación de este paquete en el simulador desarrollado. Esto porque desde el website de Gazebo, que promueve realizar modelo en formato SDF, se sugiere, para realizar la comunicación Gazebo-ROS instalar este paquete, el cual opera a plenitud y eficazmente con un modelo en formato SDF. De esta manera, si bien este paquete puede generar un control eficiente de cada uno de los elementos de la simulación de un modelo URDF, la utilización de un modelo SDF en conjunto con la potencialidad de los plugin de Gazebo, Ros y roslaunch cumple con el objetivo final de controlar el modelo simulado.
- **SDF:** El utilizar el archivo SDF en lugar URDF, recomendado para el paquete gazebo_ros_package, significa una gran ventaja en cuanto a la simplicidad de códigos y de generación de nuevos archivos. Esto se debe principalmente a que el formato SDF es muy versátil y tiene la capacidad de incorporar en un solo archivo una gran cantidad de elemento, con diferentes funciones, con el solo hecho de indentarlos correctamente. En comparación al archivo URDF que sigue el formato XML, se necesita de una gran cantidad de archivos para describir distintos aspectos del modelo.
- **Jerarquía**: Debido a la utilización del formato SDF para describir el modelo no es necesario contar con una estructura jerárquica de las carpetas que describe al modelo como se menciona en los capítulos anteriores. Estos por lo mencionado en el punto anterior, el formato SDF incluye en un solo archivo diferentes elementos del modelo simulado.

Por otra parte, en cuanto a la simulación en sí, algo importante y a tener en consideración siempre fueron las inercias y los coeficientes de roce y fricción. Esto se debe principalmente a que estos tres elementos dentro del motor físico son los que permiten obtener una representación más realista de la simulación. En cuando a las inercias porque estas indican capacidad que tiene un cuerpo para girar respecto a un eje por acción de una fuerza. Mientras que los coeficientes de roce y fricción, permitieron que la interacción entre los elementos de la simulación fuese realista y en el caso de las ruedas no deslicen cuando el modelo frena y que los joints de giro de ruedas y brazo no deslizaran sin la existencia de fuerzas que frenaras ese movimiento. Aunque se tuvo muchas consideraciones con estos valores y se realizaron distintas pruebas para obtener el resultado

deseado, la acción que no pudo ser contralada fue que, al estar el modelo simulado en reposo, gira lentamente sobre el eje z (incluso sin incorporar cámaras).

También, otro aspecto interesante y que generó bastante intriga fue la incorporación de las cámaras, ya que según lo revisado, con un menor valor de inercia rotacional el modelo debiera ser más estable, ya que existe una menor facilidad de torsión de los elementos sobre los ejes a raíz de la acción de los motores físicos, pero al incorporar las cámaras utilizadas del modelo Asus Xtion (sin masa y sin inercia rotacional), la inercia de estas es enorme, incluso mucho mayor que la del robot, y aun así, el modelo se comportó de manera estable, sin movimientos o acciones incoherente o fuera de la realidad.

Otro hecho importante es el número de software libres que pueden complementar Gazebo para hacer de este un software de simulación más potente, entre estos Meshlab. Este software cuenta con una gran cantidad de herramienta que permiten generar archivos “mesh” en formato “collada”(.dae) que pueden ser incluido en la simulación, ya sea como parte del robot o del entorno, al cual además se le pueden dar propiedades físicas de masa e inercia para una interacción más realista con los motores físicos del simulador.

Gazebo es una herramienta estable y fácil de entender para realizar simulación, la cual, si además se combina con ROS, genera una herramienta muy poderosa para manipular, controlar y probar comportamientos de robots en un entorno y para recoger una gran cantidad de información relacionada a la simulación. A esta información se puede tener acceso sin mayores inconvenientes a la a través de distintos topics que la integración Gazebo-Ros genera. La integración entre Gazebo y ROS puede realizarse de manera sencilla a través de la descripción de los elementos de la simulación (modelo, mundo, plugin, etc) como un package ROS, es decir crear un package que incorpore estos elementos, ya que los packages son es la manera en que ROS se organiza. De esta manera al lanzar (roslaunch) un paquete ROS que incluya los elementos de Gazebo, es posible dar inicio a una simulación que puede comunicarse con nodos ROS.

Gazebo incluye una gran cantidad de APIs y librerías, que pueden ser utilizados por archivos (plugin) en formato C++ o Python para el control de la simulación. Estas además permiten al usuario tener acceso a cada elemento de la simulación, desde Joint, Link, model, world, hasta información del estado de cada una de estas partes para posteriormente utilizarlas y realizar una acción determinadas. Esto permite que a través de un plugin de Gazebo se pueda realizar una gran cantidad de acciones para controlar la simulación y pueda ser comunicada con una red de nodos ROS.

Gazebo cuenta con una librería interna de plugin para realizar acciones en una simulación u obtener información de esta, y, además, en caso que sea necesario es posible crear o incorporar uno nuevo. Algunos de ejemplos de estos fueron utilizados en la esta simulación, es decir, se creó un plugin para manipular ruedas y brazos del modelo en Gazebo, cuyas acciones se publican y subscriben a un nodo ROS, este nodo se conecta con otro nodo que es generado por el plugin creado para codificar la información de joystick, y este nodo se conecta con un tercer nodo que genera la comunicación entre el joystick y ROS.

La incorporación de Rviz al proceso de simulación hace de este resultado una herramienta mucho más potente para el ámbito de la simulación. Esto se debe principalmente que en Rviz es posible identificar los tf de la simulación, elementos que sirven como “frame” de referencia de las coordenadas en el tiempo de un objeto. Con esto es posible tener un control espacial y temporal de cada una de las partes de la simulación, con los cual, ya que de esta manera se puede generar relaciones espaciales del movimiento del modelo con el entorno y generar un control completamente autónomo.

Por último, mencionar que gran parte de lo desarrollado tiene como fuente inicial trabajo realizado por otras personas, lo cual corrobora la importancia de contar con una comunidad en el ámbito de la robótica que se nutre mutuamente. Por esta razón, y como parte del proyecto SIAR, el simulador también fue incorporados a los repositorios de GitHub.

Cada uno de los archivos, paquetes y programas implementados fueron necesario para tener un correcto desempeño del simulador, donde se destaca:

- Gazebo
- ROS
- Meshlab
- Rviz
- Plugin “Camaras Asus Xtion”
- Plugin “plugin_siar_wheels_piston.cpp”
- Plugin “siar_Joy_node.cpp”
- Package “ros-kinetic-joy”
- Package “gazebo2rviz”
- Package “pysdf”

Estos se complementan correctamente y permiten generar un entorno de simulación con el modelo del robot SIAR y el mapa del alcantarillado en el cual se pueden probar de manera muy cercana a la realidad el comportamiento dinámico del robot, es decir, el desplazamiento y movimiento de brazos, estabilidad antes hostilidades en el escenario. Cabe destacar que el único inconveniente que se mantuvo hasta el final del desarrollo de este trabajo fue la presencia aleatoria de un fenómeno de deslizamiento, el cual no pudo ser controlado en su totalidad, y se presenta con mayor frecuencia en el interior del alcantarillado. Se cree que esto se genera por las irregularidades del mapa que somete al modelo a diferentes inclinaciones, por lo cual, a raíz de los motores físicos, se produce un momento de inercia sobre el eje z del robot.

Además, cabe destacar que los objetivos del proyecto fueron satisfechos, es decir:

- Se genero un simulador para la plataforma SIAR con dinámica semejante.
- Se genero un entorno con condiciones similar al real donde se desempeña el robot SIAR
- Se incorporaron las cámaras para procesamiento de datos del entorno
- Se genero un control manual teleoperado del modelo simulado
- Se genero la comunicación Gazebo-ROS para generar intercambio de mensajes entre ambos softwares.
- Se obtienen los tf del modelo y del entorno, lo cual deja abierto el simulador para generar mejoras y autonomía este.

7.2. Futuras mejoras

Por otra parte, cabe considerar que este trabajo puede tener a futuro una serie de mejorar, las cuales permitirán contar con un simulador de prueba de mayor calidad. Algunas de estas mejoras son:

- Comunicar con el soporte de Gazebo para dar a conocer el problema de deslizamiento que no pudo ser resuelto. De esta manera se podrá contar con un modelo más estable y con mayor semejanza a la plataforma real.
- Incorporar un mapa del alcantarillado de mayor dimensión, lo cual permitirá probar de mejor manera el desempeño del modelo a través de mayores obstáculos y dificultades.
- Obtener la nube de puntos del mapa en Rviz a través de las cámaras, lo cual permitirá interactuar de mejor manera con el entorno.
- La idea que toma mayor peso como mejora a futuro es utilizar el simulador como banco de prueba de algoritmo de navegación autónomos. Para esto sin duda son necesario cumplir con algunos objetivos previos como la generación de nube de puntos que se mencionaba anteriormente. De esta manera se podrá comprobar controles que den autonomía de navegación a través del mapa y que puedan ser probados en la plataforma real.

- Por último, y para ser que el simulador sea una copia fiel de la plataforma SIAR real, será necesario cambiar el chasis del modelo, ya que la última versión de la plataforma es diferente a la mostrada en este trabajo. Esto no significa mayores inconvenientes, ya que la mecánica de los brazos y ruedas se mantiene y solo sería necesario implementar un “mesh” con una cubierta superior.

8. BIBLIOGRAFÍA

[1] D. Bravo y C. F. Rengifo, «Simuladores en Robótica: Qué validar, La Cinemática o la Dinámica? Caso de estudio: Simulación de Postura de un Robot Bípedo,» *XVII COMRob2015/ID-1548,* Nov 2015.

[2] S. Ivaldi, J. Peters, V. Padois y F. Nori, «Tools for simulating humanoid robot dynamics: A survey based on user feedback," in Humanoid Robots (Humanoids),» *2014 14th IEEE-RAS International Conference on,* pp. 842-849, Nov 2014.

[3] M. Vukobratovic, «Zero-moment point. thirty five yearsof its life,» *International Journal of Humanoid Robotics,* vol. 01, nº 01, p. 157–173, 2004.

[4] «Open Dynamics Engine TM,» [En línea]. Available: http://www.ode.org/. [Último acceso: Mar 2018].

[5] «Bullet Real-Time Physics Simulation,» [En línea]. Available: https://pybullet.org. [Último acceso: Mar 2018].

[6] A. Jain, Robot and Multibody dynamics: analysis and algorithms., 2011.

[7] E. Todorov, «Analytically-invertible dynamics with contacts and constraints: theory and implementation in mujoco,» de *ICRA*, Hong Kong, 2014.

[8] E. Drumwright y . D. Shell, «An evaluation of methods for modeling,» de *ICRA*, 2011.

[9] E. Drumwright y D. Shell. , «Extensive analysis of linear complementarity problem (lcp) solver performance on randomly generated rigid body contact problems,» de *IROS*, Vilamoura, Portugal, 2012.

[10] E. Todorov, . T. Erez y Y. Tassa., «Mujoco: A physics engine for model-based control.,» de *IROS*, 2012.

[11] E. Todorov, «A convex, smooth and invertible contact model for trajectory optimization,» de *ICRA*, 2011.

[12] B. Brogliato, A. ten Dam, . L. Paoli, F. Génot y M. Abadie, «Numerical simulation of finite dimensional multibody nonsmooth v,» 2002.

[13] Y.-B. Jia, «Three-dimensional impact: energy-based modeling of tangential compliance,» *The International Journal of Robotic Research,* vol. 32, nº 1, pp. 56-83, 2013.

[14] C. Duriez, F. Dubois, A. Kheddar y C. Andriot, «Realistic haptic rendering of interacting deformable objects in virtual environments,» *IEEE Transaction on visualization and computer graphics,* vol. 12, nº 1, pp. 36-47, 2006.

[15] C. Duriez, «Control of Elastic Soft Robots based on Real-Time Finite Element Method,» *ICRA,* vol. 12, nº 1, pp. 56-83, 2013.

[16] B. Siciliano y O. Khatib, Handbook of Robotics, Springer, 2008.

[17] «DART,» Dynamic Animation and Robotics Toolkit, [En línea]. Available: http://dartsim.github.io/index.html. [Último acceso: Mar 2018].

[18] «Gazebo,» Open Source Robotics Fundation, 2012. [En línea]. Available: http://gazebosim.org/. [Último acceso: Mar 2018].

[19] «Ogre3D,» OGRE, [En línea]. Available: https://www.ogre3d.org/. [Último acceso: Mar 2018].

[20] «ARGoS,» Swarmanoid , 2011. [En línea]. Available: http://www.argos-sim.info/index.php. [Último acceso: Mar 2018].

[21] «V- REP (Virtual Robot Experimentation Platform),» Copellia Robotics, 2010. [En línea]. Available: http://www.coppeliarobotics.com/. [Último acceso: Mar 2018].

[22] O. Michel, «Webots: Professional mobile robot simulation,» *International Journal of Advanced Robotic Systems,* vol. 1, nº 1, pp. 39-42, 2004.

[23] «Open Rave,» TeamCity, 2006. [En línea]. Available: http://openrave.org/. [Último acceso: Mar 2018].

[24] «Robotran,» CEREM, [En línea]. Available: http://www.robotran.be/. [Último acceso: Mar 2018].

[25] «Voxtex Studio,» cmlabs, [En línea]. Available: https://www.cm-labs.com/simulation-solutions/mechatronic-engineering/. [Último acceso: Mar 2018].

[26] «OpenSim,» National Center for Simulation in Rehabilitation Research, 2010. [En línea]. Available: http://opensim.stanford.edu/. [Último acceso: Mar 2018].

[27] «SimTK,» National Institutes of Health, [En línea]. Available: https://simtk.org/projects/simbody/.

[28] «Adams,» MSC Software, [En línea]. Available: http://www.mscsoftware.com/product/adams. [Último acceso: Mar 2018].

[29] «MORSE,» LAAS CRNS, 2011. [En línea]. Available: https://www.openrobots.org/morse/doc/latest/what_is_morse.html. [Último acceso: Mar 2018].

[30] «MuJoCo,» Roboti LLC, [En línea]. Available: http://www.mujoco.org/. [Último acceso: Mar 2018].

[31] «XDE,» list cea tech, [En línea]. Available: http://www-list.cea.fr/en/innovating-for-industry/our-assets-for-industry/products. [Último acceso: Mar 2018].

[32] «OpenHRP3,» Open Architecture Human-centered Robotics Platform, 2010. [En línea]. Available: http://fkanehiro.github.io/openhrp3-doc/en/index.html. [Último acceso: Mar 2018].

[33] «OSRF,» Open Source Robotic Fundation, [En línea]. Available: https://www.osrfoundation.org/. [Último acceso: Mar 2018].

[34] «ROS,» Open Source Robotics Fundation, 2012. [En línea]. Available: http://www.ros.org/is-ros-for-me/. [Último acceso: Mar 2018].

[35] «ORDF,» Open Souerce Robotics Fundation, 2015. [En línea]. Available: https://www.osrfoundation.org/ros-gazebo-at-the-drc-finals/. [Último acceso: Mar 2018].

[36] «ROS,» Open Source Robotics Fundation, 2012. [En línea]. Available: http://wiki.ros.org/gazebo_ros_pkgs. [Último acceso: Mar 2018].

[37] «sdformat,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://sdformat.org/. [Último acceso: Mar 2018].

[38] «Gazebo Components,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=components&cat=get_started. [Último acceso: Mar 2018].

[39] «sdformat World,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://sdformat.org/spec?ver=1.6&elem=world. [Último acceso: Mar 2018].

[40] «sdformat Specification,» Open Souerce Robotics Fundation, 2014. [En línea]. Available: http://sdformat.org/spec. [Último acceso: Mar 2018].

[41] «Gazebo Make a Model,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=build_model&cat=build_robot. [Último acceso: Mar 2018].

[42] «Gazebo Plugin,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=plugins_hello_world&cat=write_plugin. [Último acceso: Mar 2018].

[43] «Gazebo Model Structure,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=model_structure&cat=build_robot. [Último acceso: Mar 2018].

[44] «Gazebo ROS,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros. [Último acceso: Mar 2018].

[45] «ROS Catkin,» Open Source Robotics Fundation, 2017. [En línea]. Available: http://wiki.ros.org/catkin/conceptual_overview. [Último acceso: Mar 2018].

[46] «Gazebo URDF,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials/?tut=ros_urdf. [Último acceso: Mar 2018].

[47] «ROS URDF,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://wiki.ros.org/urdf. [Último acceso: Mar 2018].

[48] «ROS Introduction,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://wiki.ros.org/ROS/Introduction. [Último acceso: Mar 2018].

[49] «ROS Concepts,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://wiki.ros.org/ROS/Concepts. [Último acceso: Mar 2018].

[50] «ROS Higher Level,» Open Source Robotics Fundations, 2014. [En línea]. Available: http://wiki.ros.org/ROS/Higher-Level%20Concepts. [Último acceso: Mar 2018].

[51] «ROS Client Library,» Open Source Robotics Fundation, 2014. [En línea]. Available: http://wiki.ros.org/Client%20Libraries. [Último acceso: Mar 2018].

[52] «ROS Distributions,» Open Source Robotics Fundation, 2017. [En línea]. Available: http://wiki.ros.org/Distributions. [Último acceso: Mar 2018].

[53] «ROS REP-3,» Open Source Robotics Fundations, 2017. [En línea]. Available: http://www.ros.org/reps/rep-0003.html. [Último acceso: Abr 2018].

[54] «IdMind,» IdMind Living Robotics, 2015. [En línea]. Available: http://www.idmind.pt/. [Último acceso: May 2018].

[55] S. Consortium, «Deliverable SIAR_D28.6,» SIAR Consortium, 2017.

[56] «sdfortmat Model,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://sdformat.org/spec?ver=1.6&elem=model. [Último acceso: Abri 2018].

[57] «GitHub Robotics UPO,» Service Robotics Lab, [En línea]. Available: https://github.com/robotics-upo/siar_simulator. [Último acceso: Abr 2018].

[58] «Gazebo Tutorial Inertia,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=inertia&cat=build_robot. [Último acceso: Abr 2018].

[59] «sdformat Collision,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://sdformat.org/spec?ver=1.6&elem=collision. [Último acceso: Abr 2018].

[60] «sdformat Joint,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://sdformat.org/spec?ver=1.6&elem=joint . [Último acceso: Abr 2018].

[61] «ROS Workspace,» Open Source Robotic Fundation, 2018. [En línea]. Available: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment. [Último acceso: Abr 2014].

[62] «ROS Package,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://wiki.ros.org/ROS/Tutorials/CreatingPackage. [Último acceso: Abr 2018].

[63] «Gazebo Plugin with ROS,» Open Source Robotic Fundation, 2014. [En línea]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins&cat=connect_ros. [Último acceso: Abr 2018].

[64] «GitHub Plugin MultiWheels,» Technische Universität Darmstadt, 2017. [En línea]. Available: https://github.com/tu-darmstadt-ros-pkg/hector_gazebo/tree/indigo-devel/hector_gazebo_plugins. [Último acceso: Abr 2018].

[65] «GitHub Asus Xtion Package,» Jimmy da Silva, 1 Sep 2015. [En línea]. Available: https://github.com/kuka-isir/gazebo_kinect_sim. [Último acceso: Abr 2018].

[66] F. J. Perez-Grau, F. R. Fabresse, F. Caballero y A. Ollero, «Long-term aerial robot localization based on visual odometry and radio-based ranging,» *International Conference onUnmanned Aircraft Systems (ICUAS),*, p. 608–614, June 2016.

[67] «GitHub Tutorial Mesh Gazebo,» Marija Popovic, 2015. [En línea]. Available: https://github.com/ethz-asl/rotors_simulator/wiki/Working-With-Meshes-in-Gazebo. [Último acceso: Abr 2018].

[68] «ROS Joy Package,» Austin Hendrix, 17 07 2016. [En línea]. Available: http://wiki.ros.org/joy. [Último acceso: Abr 2018].

[69] «ROS Tutorial Joy,» Melonee Wise, 09 Dic 2009. [En línea]. Available: http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick. [Último acceso: Abr 2018].

[70] «ROS gazebo2rviz Package,» Andreas Bihlmaier, 18 Feb 2016. [En línea]. Available: http://wiki.ros.org/gazebo2rviz . [Último acceso: Abr 2018].

[71] «GitHub gazebo2rviz package,» Andreas Bihlmaier, 10 Abr 2017. [En línea]. Available: https://github.com/andreasBihlmaier/gazebo2rviz. [Último acceso: Abr 2018].

[72] «ROS pysdf Package,» Andreas Bihlmaier, 18 Jun 2014. [En línea]. Available: http://wiki.ros.org/pysdf. [Último acceso: Abr 2018].

[73] «GitHub pysdf Package,» Andreas Bihlmaier, 12 Nov 2016. [En línea]. Available: https://github.com/andreasBihlmaier/pysdf. [Último acceso: 2018].

[74] «GitHub Tutorial gazebo2rviz,» Andreas Bihlmaier, [En línea]. Available: http://andreasbihlmaier.github.io/2014/02/19/gazebo2rviz.html . [Último acceso: Abr 2018].