Tecnológico de Costa Rica

Área Académica de Ingeniería Mecatrónica


# Desarrollo de un modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Mecatrónica con el grado académico de Licenciatura

Steven Palma Morera

Cartago, 14 de marzo de 2018

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Steven Palma Morera

Cartago, 14 de marzo de 2018

Céd: 1-1628-0749

Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Tecnológico de Costa Rica.

Miembros del Tribunal

MSc. Marta Vílchez Monge
Profesora Lectora

MSc. Eduardo Interiano Salguero
Profesor Lector

Dr. Ing. Juan Luis Crespo Mariño
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Cartago, 8 de marzo del 2018

# Resumen

En el presente trabajo, se detalla el desarrollo de un modelo computacional de simulación para el Lego Mindstorms EV3 dirigido a los estudiantes de robótica de la Universidade Da Coruña. Se modela tridimensionalmente la plataforma robótica y se estudian sus componentes mecánicos y electrónicos para realizar su diseño programático mediante los *software* Gazebo y ROS. Se desarrollan diversas herramientas básicas del Lego Mindstorms EV3 para la disposición del usuario en el entorno de simulación y se elabora la documentación necesaria para su uso y manipulación. Se validan los resultados de la simulación contrastando el comportamiento de la plataforma robótica simulada contra la real, a nivel de sensores y actuadores. Finalmente, se obtiene un modelo computacional de simulación práctico y fiel a su semejante real, con lo que respecta a funcionalidades, sentido físico y comportamientos.

**Palabras clave:** Lego Mindstorms EV3, simulación, Gazebo, ROS

# Abstract

In the present work, the development of a computer simulation model for the Lego Mindstorms EV3 aimed at robotics students of the University of A Coruña is detailed. The robotic platform is modeled three-dimensionally and its mechanical and electronic components are studied to accomplish its programmatic design through the Gazebo and ROS software. Several basic tools of the Lego Mindstorms EV3 are developed for the user's disposition in the simulation environment and the necessary documentation for its use and manipulation is elaborated. The results of the simulation are validated by contrasting the behavior of the simulated robotic platform against the real one, at the level of sensors and actuators. Finally, it's obtain a practical computational model of simulation that is faithful to its real similarity, with respect to functionalities, physical sense and behavior.

**Keywords:** Lego Mindstorms EV3, simulation, Gazebo, ROS

*A Dios y a la mejor, magnífica y maravillosa madre. Esto es para ti mom.*

# Agradecimientos

A Dios, a mi madre, a mi familia, a mis amigos, a mis profesores y a todos aquellos quienes me introdujeron y ayudaron en la aventura de la ciencia y la tecnología.

Steven Palma Morera

Cartago, 14 de marzo de 2018

# Índice general

# Índice de figuras

# Índice de tablas

# Lista de símbolos y abreviaciones

## Abreviaciones

| | |
|---|---|
| ADC | Analog-to-digital converter |
| GII | Grupo Integrado de Ingeniería |
| GPS | Global Positioning System |
| ISO | International Organization for Standardization |
| LDD | Lego Digital Designer |
| ROS | Robot Operating System |
| SDF | Simulation Description Format |
| SI | Sistema Internacional de Unidades |
| UART | Universal asynchronous receiver-transmitter |
| UDC | Universidade Da Coruña |
| XML | Extensible Markup Language |

# Capítulo 1

# Introducción

## 1.1 Entorno del proyecto

La Universidade Da Coruña (UDC) es una institución de educación superior ubicada en España, creada por la Ley 11/1989 del 20 de Julio. Como universidad pública, su finalidad es contribuir al avance cultural, social y económico de la sociedad por medio de la generación, gestión y difusión de cultura y conocimiento científico, tecnológico y profesional, [5]. La Universidade Da Coruña está territorialmente estructurada en dos sedes, el campus de A Coruña y el campus de Ferrol. Este último campus, puesto a disposición de la UDC el 20 de Enero de 1992, es dónde el presente proyecto toma participación.

Entre la amplia oferta académica que ofrece la Universidade Da Coruña, se encuentra el grado en ingeniería informática. En este grado, los estudiantes cursan una asignatura de robótica, en la cual aprenden los fundamentos de esta rama de conocimiento. En dicho curso, se emplean diversas herramientas de laboratorio para ejemplificar los conceptos aprendidos en clase y para formar una primera aproximación a la aplicación de estos conocimientos a nivel práctico. El dispositivo Lego Mindstorms EV3 es una de estas herramientas que utilizan en la asignatura de robótica.

Lego Mindstorms EV3 es un sistema robótico puesto en venta desde el 2013 por la compañía danesa LEGO, y es la tercera generación del *kit* robótico en la línea Mindstorms, [11]. Su objetivo principal es brindar un acercamiento a la robótica, electrónica, mecánica y programación de forma didáctica, sencilla y versátil. Dicho *kit* proporciona todos los materiales necesarios para fabricar – de forma simple - robots Lego de pequeña escala para fines didácticos. Entre los componentes del *kit* se pueden encontrar servo motores, sensores, una unidad central de procesamiento, módulos de comunicación electrónica y todos los componentes de conexión física necesaria para el ensamblaje tales como, engranes, fajas de transmisión y ladrillos Lego, entre otros (ver 1.1). Además, proporciona también soporte *software* mediante un programa de computador y una aplicación digital para dispositivos móviles. La versatilidad de sus elementos de construcción permite crear y comandar más de cien configuraciones de robots Lego.


**Figura 1.1:** Sistema robótico Lego Mindstorms EV3. Fuente: [11].

Actualmente, los estudiantes de la asignatura de robótica del grado en ingeniería informática necesitan del sistema robótico en una forma física o presencial cuando realizan las prácticas del curso. Es decir, cada grupo de trabajo ocupa el modelo físico del Lego Mindstorms EV3 para desarrollar su trabajo, situación poco favorable para la eficiencia del desarrollo del mismo. A estos estudiantes les sería de gran utilidad avanzar en las prácticas del curso que involucren el sistema robótico, sin la necesidad de contar con el mismo de manera física. Esto no solo mejoraría la eficiencia en el tiempo de trabajo de los estudiantes sino también, la experiencia educativa del curso.

El Grupo Integrado de Ingeniería (GII), creado en 1999 y ubicado en el Campus de Ferrol de la Universidade Da Coruña, es un grupo interdisciplinar de investigación aplicada en ingeniería, orientado hacia la transferencia de conocimiento y la generación de nuevos productos en el entorno industrial, [6]. Dicho grupo, al estar vinculado a las Escuelas de Ingeniería de la UDC, ha decidido extender su ayuda a los estudiantes de la asignatura de robótica del grado en ingeniería informática, y han externado la necesidad de que un estudiante de Mecatrónica aporte a esta iniciativa ya que, el Lego Mindstorms EV3 es en esencia un sistema mecatrónico. El Grupo Integrado de Ingeniería desea que el estudiante de Mecatrónica, desarrolle un proyecto que permita a los estudiantes de la Universidade Da Coruña realizar los trabajos de la asignatura de manera eficiente, solucionando las complicaciones actuales entorno a la necesidad de utilizar el *kit* robótico de manera física.

Cabe rescatar que este proyecto no se engloba dentro de otro más grande, ni se han realizado esfuerzos con anterioridad para implementar una simulación del sistema Lego Mindstorms EV3, por parte del GII o de la UDC. Sin embargo, actualmente se cuenta con plataformas *software* de simulación 3D para sistemas robóticos, tales como Gazebo y ROS las cuales se explican a profundidad en las secciones siguientes. También cabe rescatar que, al ser este proyecto de trascendencia educativa, no competen particularidades y detalles sobre procesos productivos industriales, aspectos legales comerciables ni administrativos,

que influyan en la caracterización del mismo. Sin embargo, sí competen detalles de índole didáctica como manuales de uso e instalación, sistemas amigables con el usuario, licencias de libre uso y las documentaciones necesarias del desarrollo del proyecto para ponerlas a disposición del futuro usuario.

## 1.2 Definición del problema

### 1.2.1 Generalidades

Los estudiantes del grado en ingeniería informática de la Universidade Da Coruña necesitan trabajar con el sistema robótico Lego Mindstorms EV3 de manera física para poder desarrollar las prácticas de la asignatura de robótica. Esta situación trae consigo diversas complicaciones para los estudiantes, a continuación se estudian y detallan las mismas. Sin embargo, cabe rescatar que dicha situación puede no tratarse como un problema sino más bien como una oportunidad de mejora, en donde se busca un curso de mayor calidad didáctica y en donde los estudiantes reciban una experiencia académica enriquecedora.

1. Los estudiantes tienen un tiempo de trabajo limitado con el sistema robótico para desarrollar las prácticas de la asignatura debido a la alta demanda del *kit* por parte de los demás grupos. Al ocuparse el *kit* físico, solo un grupo a la vez puede trabajar en la práctica, situación ineficiente en términos de tiempo de trabajo pues no se consigue avanzar en la práctica mientras otro grupo esté utilizando el Lego Mindstorms EV3.

2. Al Lego Mindstorms EV3 ser parte del equipo del laboratorio, este pertenece a la Universidade Da Coruña y por ende, no es permitido su uso fuera de las instalaciones. Es decir, los estudiantes pueden trabajar con el *kit* robótico únicamente en el laboratorio y no pueden trasladarlo a su lugar de habitación para seguir trabajando en la práctica del curso. Nuevamente, esta situación se traduce a una limitación en el tiempo de trabajo disponible.

3. Los estudiantes al tener poco tiempo de exposición y uso con el sistema robótico y al ser esta práctica una de las primeras aproximaciones con este tipo de sistemas, son propensos a cometer errores de ejecución los cuales ocasionan el constante degrado o daños de mayor importancia en los diversos elementos del *kit*. Estos daños ocasionan paros indeseables en los tiempos de trabajo.

4. Al no tener ninguna aproximación previa de la práctica con el sistema robótico, se pierde tiempo en situaciones ajenas a los objetivos de la misma, reduciendo el tiempo de trabajo efectivo con el *kit*.

### 1.2.2 Síntesis del problema

Los estudiantes de la asignatura de robótica del grado en ingeniería informática de la Universidade Da Coruña tienen un tiempo de trabajo insuficiente e ineficiente con las prácticas de curso en las que utilizan el sistema robótico Lego Mindstorms EV3, al no tener forma de avanzar con la misma cuando no cuentan con dicho *kit* en físico.

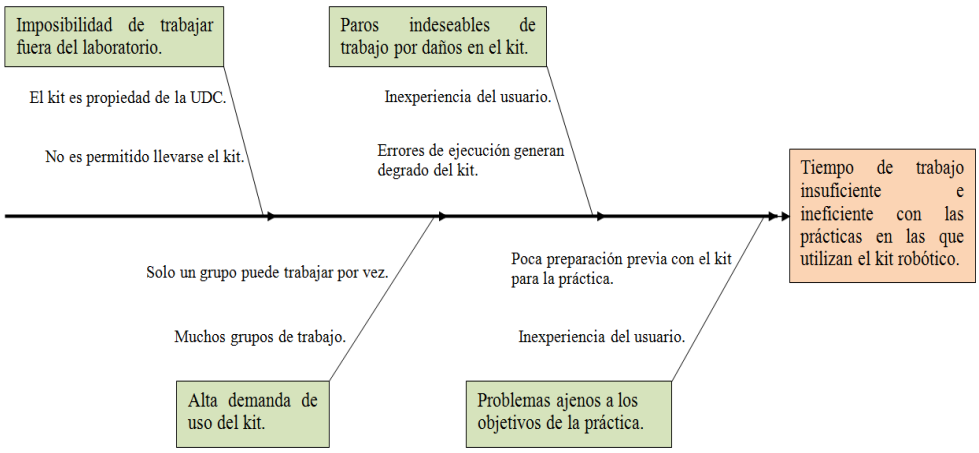



**Figura 1.2:** Diagrama Ishikawa del proyecto. Fuente: Elaboración propia.

## 1.3 Enfoque de solución

Planteado el problema anterior, se optó – junto al Grupo Integrado de Ingeniería – por proveer a los estudiantes de la asignatura de robótica del grado en ingeniería informática de la Universidades Da Coruña una forma de poder avanzar con las prácticas del curso que involucren el Lego Mindstorms EV3 sin la necesidad de contar con el *kit* robótico en físico. Esta solución es un modelo computacional que simule los comportamientos reales del sistema robótico físico, que los estudiantes necesitan para desarrollar las prácticas académicas. De esta forma, las complicaciones expuestas se solucionan de la siguiente manera:

1. Los estudiantes podrían trabajar en sus prácticas de curso sin dependencia de los demás grupos de trabajo. Cada grupo tendría su propio modelo de simulación lo que permitiría que múltiples grupos trabajen simultáneamente en la asignatura, aun cuando otro grupo se encuentra utilizando el *kit* físico. Esto representaría un aumento en el tiempo de trabajo disponible para desarrollar la práctica.

2. Al cada grupo de trabajo tener su propio modelo de simulación del sistema robótico en su computador personal, no es necesario utilizar el *kit* físico fuera de las instalaciones de la institución. Los estudiantes podrían avanzar con su práctica del curso

desde su lugar de habitación, realizando las simulaciones necesarias del comportamiento del sistema robótico para la misma. Esto representaría un aumento en el tiempo de trabajo disponible para desarrollar la práctica.

3. Con el modelo computacional de simulación, los estudiantes podrían realizar aproximaciones previas al sistema robótico que les ayude a formar experiencia sobre el uso y manejo del mismo. Con el modelo computacional el estudiante podría darse cuenta de qué ha hecho mal y qué debe corregir, para cuando realice la práctica con el sistema en físico cometa menos errores que comprometan negativamente el estado del *kit*. Así, al reducir los daños ocasionados por la inexperiencia del usuario, se conservaría el buen estado del sistema robótico y se evitarían paros indeseables en los tiempos de trabajo.

4. El modelo de simulación ofrece la posibilidad de experimentar con el *kit* y simular las prácticas de curso con anterioridad. Esto permitiría a los estudiantes realizar la práctica con conocimiento previo, y preparados para las eventuales situaciones que ameriten la práctica. Los estudiantes podrían entonces, sacar más provecho al tiempo de uso con el *kit* en físico y enfocarse en los objetivos de las prácticas académicas.

La solución planteada, junto al Grupo Integrado de Ingeniería, de desarrollar un modelo en simulación del sistema robótico Lego Mindstorms EV3 para el problema presentado se justifica con los siguientes argumentos:

1. Actualmente se encuentran disponibles diversas herramientas computacionales que permiten realizar este tipo de aproximaciones para plataformas robóticas. *software* para modelado 3D, simulación de comportamientos dinámicos, y bibliotecas *software* para la construcción de aplicaciones robóticas son herramientas respaldadas y depuradas. Mediante el desarrollo de esta solución con el uso de estas herramientas, se siguen y fortalecen las tendencias actuales de manejo, monitoreo y mantenimiento digital de equipo industrial en una empresa, en las que un ingeniero mecatrónico encuentra gran participación.

2. El problema presentado sucede también a nivel industrial. Es muy común encontrar a un operario que no puede ejecutar su trabajo de manera eficiente con su máquina industrial por limitaciones de tiempo, espacio y recursos energéticos, y también se encuentran mal funcionamientos del equipo industrial por la inexperiencia del operario. Buscando entonces una solución realista, en donde una empresa descarta la posibilidad de comprar maquinaria industrial por insuficiente capacidad económica, el desarrollo de un modelo de simulación computacional de la máquina industrial se apega más a una solución industrial. Es por esto que se ha descartado la opción de comprar más sistemas robóticos Lego Mindstorms EV3 para suplir la demanda de uso de los estudiantes, y se ha optado por utilizar las herramientas tecnológicas a disposición para solventar el problema planteado. Además, aunque se compraran

más *kit*s robóticos, esto no solventa las complicaciones relacionadas al degrado de los mismos por inexperiencia del estudiante ni ayuda a brindar un conocimiento experimental previo para la eficiente ejecución de la práctica académica.

3. El desarrollo de un modelo de simulación computacional del sistema robótico Lego Mindstorms EV3 representa un aporte intelectual a la comunidad científica y tecnológica pues, no se han realizado esfuerzos con anterioridad para implementar una simulación de este *kit* didáctico. Como se mencionó anteriormente, la situación por la que atraviesan los estudiantes de la UDC, se puede tratar como una oportunidad de mejora. El desarrollo del modelo computacional de simulación para el *kit* robótico no solo solventa las complicaciones actuales, si no que también mejora la calidad didáctica del curso de robótica y enriquece la experiencia y aprendizaje académico del estudiante. Con la solución planteada, se abre un abanico de posibilidades de estudio, investigación y experimentación para cualquier futuro usuario y sirven a fortalecer los objetivos del curso actual.

Es importante aclarar que, la solución planteada no elimina por completo la necesidad de uso del *kit* robótico en físico, pues es claro que las prácticas del curso tienen como objetivo la vinculación del estudiante con el sistema en físico y que la experimentación con el mismo es fundamental para el aprendizaje de los temas de la asignatura. Es más bien, una herramienta, una extensión o una alternativa disponible para el estudiante tenga la oportunidad de profundizar, aprender y practicar con mayor libertad, tiempo y eficiencia con el sistema robótico Lego Mindstorms EV3. También, cabe rescatar que la solución planteada no tiene como alcance desarrollar modelos de simulación de situaciones y comportamientos del *kit* robótico ajenos a los objetivos de la asignatura de robótica y a los usos que se le dan para el desarrollo de sus prácticas académicas.

## 1.4 Estructura del proyecto

La naturaleza multidisciplinaria del sistema robótico Lego Mindstorms EV3 conlleva a que para el correcto desenvolvimiento de esta iniciativa sean necesarios conocimientos sobre electrónica y dinámica de movimiento - para comprender su funcionamiento y configuración, y conocimientos sobre programación y robótica - para el diseño de su simulación computacional. Se evidencia entonces, la afinidad con la ingeniería mecatrónica, la cual integra dichas áreas para desarrollar la solución en su totalidad. A continuación se presenta un esquema general para el diseño recurrente de proyectos mecatrónicos, conformado por una etapa de investigación y una etapa de ejecución. Este flujo pretende dar una estructura preliminar a sobre cómo se soluciona el problema planteado. El lector podrá encontrar los resultados de la fase de investigación en el capítulo 2, mientras que todo lo relevante a la fase de ejecución se trata en el capítulo 3 y 4.

1. Investigación

- *kit* robótico Lego Mindstorms EV3: Se investigan a profundidad los pormenores relevantes del sistema robótico para comprender su manejo y funcionamiento, además de aprender a utilizarlo. El *kit* posee sensores, actuadores electromecánicos, micro controlador, elementos de sujeción mecánicos y demás componentes los cuales se deben estudiar y analizar para caracterizar su función en el sistema y en la simulación.
- Entorno didáctico: Se delimita el alcance de la solución con respecto al sistema robótico. La versatilidad del *kit* permite utilizar diversas configuraciones (morfología y sistema sensorial del robot) por lo que se debe establecer con cuál se va a desarrollar la solución. Para esto, se investiga la configuración de más utilidad para los estudiantes – siendo esta la que utilicen de manera más recurrente en el curso-, y el rol que juega en los objetivos y procedimientos de las prácticas en donde lo utilizan, con respecto a su comportamiento. De esta manera, se determinan los requerimientos de la simulación.
- Herramientas tecnológicas: La exposición a herramientas *software* no utilizadas previamente requiere de un primer paso de aprendizaje. Se desarrolla el conocimiento y habilidades suficientes para el apropiado manejo de las mismas. Se conocen de antemano los programas *software* que se utilizaran para el desarrollo del proyecto pues, se acordaron en conjunto con el GII. Si bien se podría argumentar que el uso de estas herramientas no representa la forma más sencilla para llevar a cabo el proyecto, se sabe que las herramientas escogidas ofrecen la mayor flexibilidad y versatilidad en el modelo computacional, de forma en que desarrolladores, estudiantes o investigadores puedan adaptarlo a sus necesidades en un futuro. A continuación se presenta una lista de las principales plataformas *software* utilizadas, más adelante se explican a detalle.

  1.1. Lego Digital Designer

  1.2. Blender

  1.3. Gazebo

  1.4. ROS

2. Ejecución

- Diseño: Con base en el conocimiento adquirido anteriormente, se diseña a plenitud técnica los diferentes elementos del modelo computacional. Aquí –mayormente- se desarrolla el modelado tridimensional del sistema robótico, el entorno de simulación, los códigos del control de sus comportamientos y demás archivos para la ejecución de la solución programática.
- Documentación: Debido al entorno didáctico en el que se encuentra el proyecto, se elabora un manual de usuario que permita al estudiante conocer los distintos elementos desarrollados relevantes al funcionamiento y manejo del modelo computacional. Además, se elabora el informe escrito y la presentación para hacer efectivo el proyecto.

- Validación: Se comprueba que los diferentes elementos desarrollados satisfacen los requerimientos para su aplicación especificada. Es un proceso de prueba recursiva, en dónde se identifican errores u oportunidades de mejora en las programaciones y documentaciones elaboradas, y se regresa a la mesa de diseño para realizar los cambios adecuados.

## 1.5 Objetivos del proyecto

### 1.5.1 Objetivo general

1. Desarrollar un modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3.

### 1.5.2 Objetivos específicos

1. Diseñar el modelo tridimensional de una configuración estándar (morfología y sistema sensorial) del sistema robótico Lego Mindstorms EV3 y su entorno de simulación, en Gazebo.

2. Diseñar comportamientos básicos en la simulación del sistema robótico Lego Mindstorms EV3 mediante la interconexión programática con ROS.

3. Elaborar una guía de usuario de los elementos desarrollados orientado a los estudiantes de la asignatura de robótica del grado en ingeniería informática de la Universidade Da Coruña.

# Capítulo 2

# Marco teórico

Para el desarrollo del presente proyecto, se trabaja con conocimientos y herramientas provenientes de diversas ramas de información que ameritan investigación y dominio. Aquí el lector encontrará una compilación concisa -resultado de la investigación previa- de las distintas herramientas y conocimientos directamente relacionados y necesarios para la comprensión de la solución implementada. Sin embargo, no se explicará a detalle conceptos que formen parte normal del currículum de Mecatrónica y se tratarán los temas con una profundidad adecuada, en función de su relevancia al proyecto. A continuación, las líneas de conocimiento fundamentales para el presente trabajo.

## 2.1 Lego Mindstorms EV3

### 2.1.1 Generalidades

Lego Mindstorms EV3 es un sistema robótico puesto en venta desde el 2013 por la compañía danesa LEGO, y es la tercera generación del *kit* robótico en la línea Mindstorms. Su objetivo principal es brindar un acercamiento a la robótica, electrónica, mecánica y programación de forma didáctica, sencilla y versátil. Dicho *kit* proporciona todos los materiales necesarios para fabricar – de forma simple - robots Lego de pequeña escala para fines didácticos. La versatilidad de sus elementos de construcción permite crear y comandar más de cien configuraciones de robots Lego, [11].

Se pusieron en venta dos versiones del *kit* robótico. EV3 Home (3131) es la versión más común y comercial -esta enfocada mayormente al público general-, y la versión EV3 Education Core Set (45544) que -como su nombre lo indica- esta enfocada hacia el campo educativo e investigativo. Esta última es la más utilizada en competencias como olimpiadas de robótica, [13]. Ambas versiones fueron bien recibidas por su público meta y desarrollaron toda una comunidad aficionada.

Interesa particularmente la versión EV3 Education Core Set (45544) para el desarrollo

de este proyecto. Dicha versión difiere con la otra en la complejidad técnica de los diversos componentes y manuales de instrucción. Entre los componentes del *kit* educativo se pueden encontrar servo motores, cuatro tipo de sensores, una unidad central de procesamiento programable, módulos de comunicación electrónica y todos los componentes de conexión física necesaria para el ensamblaje tales como ladrillos Lego, ruedas, engranes, fajas de transmisión, entre otros. Además, proporciona también soporte *software* mediante un programa de computador y una aplicación digital para dispositivos móviles. A continuación, se detallan los pormenores relevantes de los componentes del *kit* Lego Mindstorms EV3 Education Core Set (45544), [12].

### 2.1.2 Bloque programable EV3

El bloque programable del Lego Mindstorms EV3 es la unidad de procesamiento central del *kit* robótico. Dicho bloque es el encargado de coordinar y ejecutar cualquier tarea programática y electrónica del *kit* robótico para habilitar su amplia gamma de funcionalidades. Se compone de un microcontrolador, una interfaz física de usuario con *display*, LEDs y botones programables, puertos de entrada y de salida, un parlante para la reproducción de sonidos, comunicaciones inalámbricas Wi-Fi y *Bluetooth*, módulos de memoria expandible y el sistema de su alimentación eléctrica. Este bloque programable se puede considerar como el cerebro de todas las plataformas robóticas del *kit* EV3. En la Fig.2.1 se muestra un diagrama de alto nivel de la conexión entre los bloques electrónicos de la unidad de procesamiento central.

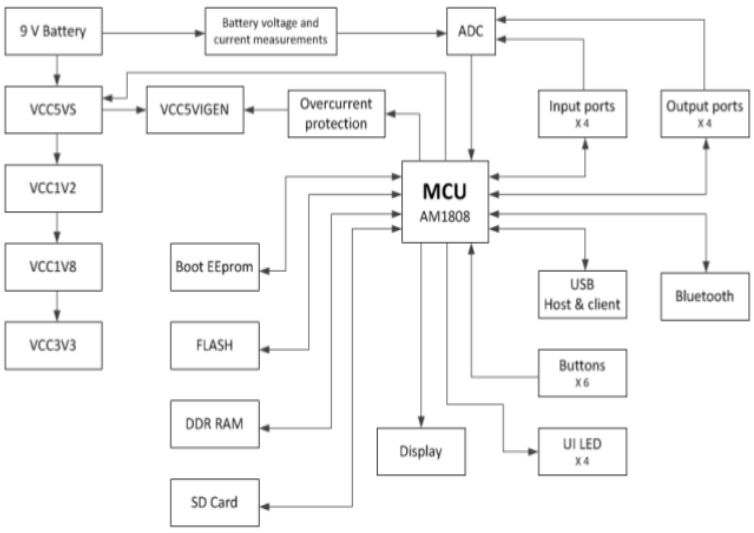



**Figura 2.1:** Diagrama de bloques a alto nivel del bloque programable EV3. Fuente: [12].

### 2.1.3 Sensores y actuadores

#### Sensor táctil

El sensor táctil es un simple interruptor mecánico que se activa si se presiona y se desactiva si se libera. Se puede considerar como un pulsador eléctrico no permanente. Es utilizado comúnmente para evitar que el robot colisione bruscamente contra un obstáculo o para indicar que ha llegado a un punto de interés.

Su implementación es enteramente analógica, la señal eléctrica del interruptor se conecta a un *analog-to-digital converter* (ADC) y el bloque programable EV3 determina si el interruptor ha sido presionado o liberado. El usuario -una vez que el sensor se presiona- dispone en el bloque programable EV3 de un valor lógico de 1 y cuando no está presionado, un 0.

#### Sensor de color

El sensor de color tiene tres diferentes modos de funcionamiento: modo de color, modo de luz ambiente y modo de luz reflejada. Para este trabajo, interesa únicamente el modo de luz ambiente. Cuando el sensor está trabajando en el modo de luz ambiente, es capaz de medir la intensidad de luz ambiente de sus alrededores.

La implementación de este sensor es digital. Su funcionamiento consiste en un foto-diodo PD-33BRC que funciona como transductor al convertir los rayos de luz visible incidentes en una señal eléctrica. Dicha señal eléctrica es controlada por un microcontrolador de 8-bit que realiza el procesamiento de la señal y la envía mediante comunicación UART al bloque programable EV3.

El usuario dispone en el bloque programable EV3 de un valor entre 0 y 100 que representa porcentualmente la medición del sensor. El microcontrolador utiliza una tabla de conversión algorítmica para transformar la señal eléctrica proveniente del circuito del foto-diodo a dicha escala, y es capaz de generar nuevas mediciones de luz ambiente a una frecuencia de 1000 Hz. Cabe rescatar que si bien se conoce que se utiliza una tabla de conversión algorítmica, se desconoce el algoritmo exacto que transforma la señal eléctrica a la escala entre 0 y 100.

#### Sensor ultrasónico

El sensor ultrasónico tiene tres diferentes modos de funcionamiento: modo de medición de distancia en centímetros y en pulgadas, y modo de escucha. Para este trabajo, interesa únicamente el modo de medición de distancia en centímetros. Cuando el sensor está trabajando en modo de medición de distancia en centímetros es capaz de medir la distancia al objeto más cercano en centímetros.

La implementación de este sensor es digital. Su funcionamiento lo logra mediante un

transductor emisor AW8T40 que envía una señal sonora ultrasónica de doce ráfagas a 40 kHz, que reflejan en el objeto y vuelven al sensor donde un transductor receptor AW8R40 las recibe. El sensor, al relacionar el tiempo de envío y recepción de la señal sonora, es capaz de aproximar la distancia a la que se encuentra el objeto. La medición del sensor es controlada por un microcontrolador de 8-bit que realiza el procesamiento de la señal e información y la envía mediante comunicación UART al bloque programable EV3.

El sensor es capaz de medir de 3 cm a 250 cm con una resolución de 0.1 cm y con una precisión de 1 cm, y es capaz de generar nuevas mediciones de distancia a una frecuencia mínima de 66 Hz -debido a que el tiempo que tarda la señal sonora en ir, reflejarse en un objeto a 250 cm y regresar, es de 15 ms. El usuario dispone en el bloque programable EV3 del valor de la distancia, con la única consideración de que cuando ninguna señal sonora regresa al sensor -es decir, que no hay ningún objeto dentro del rango de medición- se dispone de un valor de 255.

### Sensor giroscopio

El sensor giroscopio tiene tres diferentes modos de funcionamiento: modo de medición de ángulo relativo, modo de funcionalidad estándar y un modo que combina los primeros dos. Para este trabajo, interesa únicamente el modo de medición de ángulo relativo. Cuando el sensor está trabajando en modo de medición de ángulo relativo, es capaz de medir el ángulo de rotación con respecto al eje z relativo a la orientación inicial de cuando se reinició el sensor por última vez. La implementación de este sensor es digital. Su funcionamiento lo logra mediante un circuito integrado ISZ-655 que mide y almacena la velocidad angular del *kit* robótico y la transforma a un ángulo relativo. Dicha medición es controlada por un microcontrolador de 8-bit que realiza el procesamiento de la señal y la envía mediante comunicación UART al bloque programable EV3.

El sensor es capaz de medir hasta una velocidad angular de 440 grados por segundo con una precisión de 3 grados para giros menores a 90 grados, y es capaz de generar nuevas mediciones a una frecuencia de 1000 Hz. El usuario dispone en el bloque programable EV3 del valor del ángulo relativo.

### Motores

El sistema robótico contiene tres motores de dos tipos, dos motores “grandes” y un motor “mediano”. Ambos tipos son motores CD de 9 V que incluyen un sensor de rotación con resolución de 1 grado para un control de velocidad y sincronización preciso. Los motores grandes usualmente se utilizan para la fuerza motriz de avance del robot -pues son más potentes-. Mientras que el motor mediano al ser más compacto, reacciona a mayor velocidad y se utiliza para accionar cualquier otro tipo de actuador que contenga el robot. En la Tabla 2.1, se muestran las especificaciones esenciales de ambos motores.

| Condición | Motor grande | Motor mediano |
|---|---|---|
| 9 V, sin carga | 175 rpm<br>60 mA | 260 rpm<br>80 mA |
| Con carga | 45 Ncm<br>1.8 A | 15 Ncm<br>800 mA |

**Tabla 2.1:** Especificación técnica de los motores. Fuente: [12]

### 2.1.4 Elementos mecánicos

La estructura mecánica del *kit* robótico consiste mayormente en una serie de ladrillos Lego hechos de plástico ABS interconectados entre sí que logran formar más de cien configuraciones estructurales para colocar los cuatro sensores, los tres motores, el bloque programable, dos llantas y una rueda giratoria esférica - que se utiliza usualmente para darle estabilidad al robot sin comprometer su movilidad por fricción.

### 2.1.5 Soporte digital

El Lego Mindstorms EV3 también tiene soporte digital, el cuál consiste en un programa para computadora y en una aplicación para móvil. En ambas, se pueden diseñar en una programación de tipo bloque, secuencias lógicas de acción que formulen un comportamiento deseado. Bloques encargados de enviar las señales a los motores para que el robot gire una cantidad de grados deseados o para que el bloque programable lea los valores de los sensores, se encuentran en total disposición del usuario para que pueda diseñar comportamientos de forma secuencial. Adicionalmente, en la versión de computador es posible visualizar información de los sensores, guardar registros de datos, consultar manuales y acceder a la base de datos *online* de la comunidad del *kit* robótico.

## 2.2 Lego Digital Designer

Lego Digital Designer (LDD) es un programa *software* de libre uso disponible para Windows y macOS desarrollado por la corporación danesa LEGO en el 2004, [14]. Es un programa que permite el diseño asistido por computadora de cualquier *kit* Lego. Ofrece gran variedad de ladrillos y accesorios Lego de forma virtual para que el usuario pueda ensamblar su propio modelo tridimensional, facilitando grandemente el proceso de diseño 3D para sistemas Lego. La extensión de un archivo generado con este programa *software* es .ldr, formato creado específicamente para el Lego Digital Designer.

## 2.3 Blender

Blender es un programa *software* de libre uso disponible para macOS, Windows y Linux desarrollado por la corporación holandesa The Blender Foundation en 1995, [2]. Es una suite robusta de creación 3D la cual permite realizar la totalidad del modelado tridimensional, simulación y renderizado, entre otras funciones.

Interesa particularmente sus capacidades de edición para sólidos tridimensionales, en las que se incluyen tareas de manipulación de escalas, orientaciones, ubicaciones, creaciones de ensambles y manipulación de *meshes*. Esta última es especialmente relevante en el entorno Blender. Un *mesh* es una colección de vértices, bordes y caras que describen la forma de un objeto tridimensional en computador. La forma más común y más básica del modelado tridimensional es mediante el modelado de *meshes*, el cual consiste en crear todo el sólido mediante una serie de estos mismos de diferentes tamaños y formas. Blender, permite accesar y manipular la información de cada *mesh* en un objeto tridimensional. En la Fig.2.2, se aprecia un sólido tridimensional en forma de mono en el cual se puede editar cada detalle de sus *meshes*.

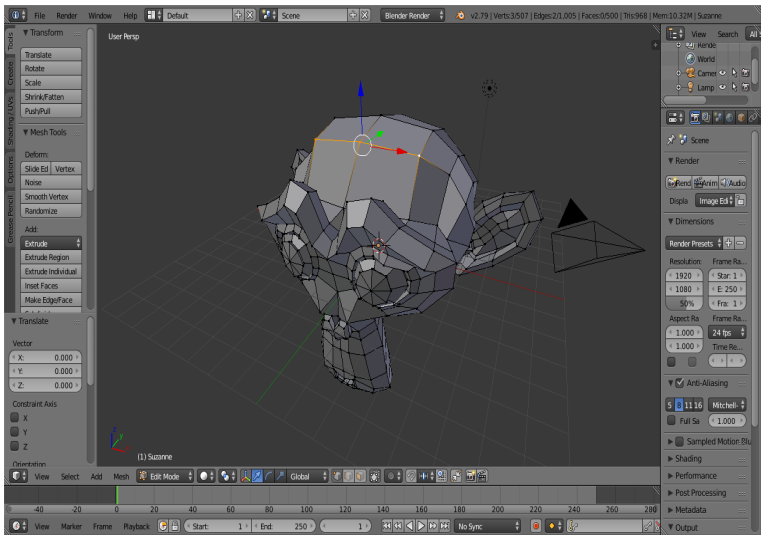


**Figura 2.2:** Interfaz gráfica de usuario de Blender. Fuente: Elaboración propia.

Por último, cabe rescatar que Blender ofrece la posibilidad de importar y exportar archivos de extensión Digital Asset Exchange (.dae). Esta extensión es la acordada entre diversos programas de diseño asistido en computadora para intercambiar entre ellos los archivos digitales bajo el mismo formato. Archivos de dicha extensión son basados en el formato XML Collada, el cúal fue adoptado por la normativa ISO como el formato para aplicaciones tridimensionales interactivas en la norma ISO/PA17506, [7].

## 2.4 Gazebo

### 2.4.1 Generalidades

Gazebo es un programa *software* de libre uso desarrollado mayormente por la corporación estadounidense Open Source Robotics Foundation. Es un simulador 3D de robots el cual ofrece motores de simulación para dinámicas de movimiento, visualización de gráficas 3D, construcción de modelos robóticos y entornos de simulación, manejo de sensores, variedad de *plugins* y extensa cantidad de comandos para facilitar la simulación y control robótico, [8]. En la Fig.2.3, se aprecia un modelo en Gazebo del robot Baxter y una gasolinera.

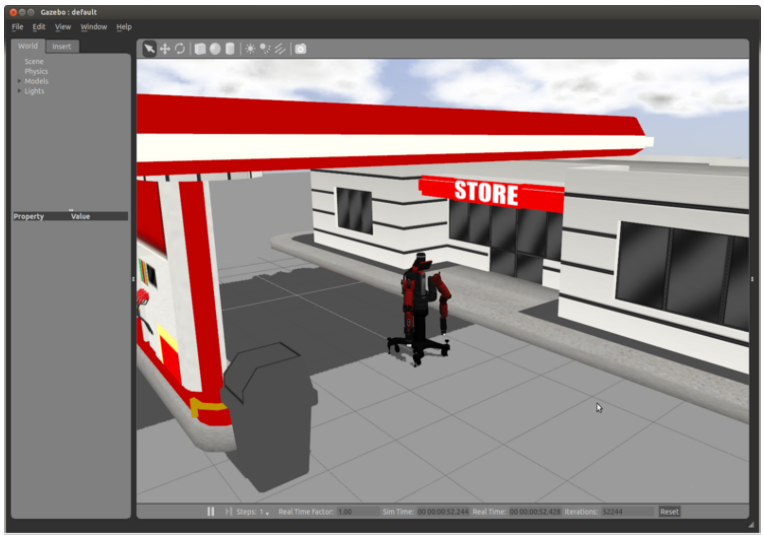



**Figura 2.3:** Interfaz gráfica de usuario de Gazebo. Fuente: [8].

En Gazebo, la descripción de objetos y entornos de simulaciones robóticas se lleva a cabo mediante el formato SDF (Simulation Description Format). Este formato permite describir todos los aspectos de una plataforma robótica, objetos estáticos y dinámicos, iluminación, terrenos, la física de movimiento, entre otros, que requiera una simulación, [10]. A continuación, una lista de los principales elementos de un entorno de simulación descrito con el formato SDF.

### 2.4.2 World

Un elemento tipo “world” o “mundo” en el formato SDF, es el conjunto de todos los siguientes elementos de simulación en listados. Específicamente, se pueden modificar propiedades de la simulación como el viento, gravedad, campo magnético terrestre y atmósfera. Pero además, se pueden incluir elementos de física, iluminación, modelos y *plugins*. En general, las simulaciones en Gazebo se llevan a cabo mediante la descripción de un mundo junto a todos sus demás elementos anidados.

### 2.4.3 Scene

Un elemento tipo “scene” o “escena” debe ser hijo de un elemento tipo mundo. El elemento escena se encarga de la apariencia visual del mundo en simulación. Esto incluye, la apariencia del cielo, del piso, de la neblina, del color e intensidad de la luz ambiente, activar o desactivar sombras, y algunas otras más propiedades de la interfaz gráfica de usuario de Gazebo.

### 2.4.4 Physics

Un elemento tipo “physics” o “física” debe ser hijo de un elemento tipo mundo. El elemento física se encarga de establecer el tipo y propiedades del motor de simulación para las dinámicas de movimiento que Gazebo utilizará para procesar todas las interacciones físicas del mundo. El motor de simulación física más utilizado en Gazebo es ODE.

### 2.4.5 Light

Un elemento tipo “light” o “luz” debe ser hijo de un elemento tipo mundo. El elemento luz se encarga de describir propiedades de cualquier cantidad de fuentes de luz presentes en el mundo. Una fuente de luz puede tener una dirección específica, una atenuación, difusión, entre otras características.

### 2.4.6 Model

Un elemento tipo “model” o “modelo” puede ser hijo de un elemento tipo mundo o puede ser un archivo SDF por aparte. Un elemento modelo describe por completo cualquier plataforma robótica o cualquier otro objeto presente en un mundo de simulación. Los modelos pueden variar en complejidad pero esencialmente, son un conjunto de elementos tipo enlaces, sensores, articulaciones, objetos de colisión, elementos visuales y *plugins*. Es importante aclarar que usualmente un modelo se describe fuera de un elemento tipo mundo para de esta manera poder simular el modelo en diversos mundos. Particularmente útil para cuando se debe simular una plataforma robótica en diferentes condiciones.

### 2.4.7 Link

Un elemento tipo “link” o “enlace” debe ser hijo de un elemento tipo modelo. Un elemento tipo enlace contiene las propiedades físicas de un cuerpo del modelo. Este cuerpo puede ser una llanta, un sensor, un brazo, o cualquier otro cuerpo de una plataforma robótica. Un elemento tipo enlace debe tener propiedades visuales, de colisión e inerciales.

### 2.4.8 Joint

Un elemento tipo "joint" o "articulación" debe ser hijo de un elemento tipo enlace. Un elemento tipo articulación se utiliza para describir la conexión cinemática y dinámica entre dos enlaces.

### 2.4.9 Sensor

Un elemento tipo "sensor" debe ser hijo de un elemento tipo enlace. Un elemento tipo sensor puede ser de diversos tipos y cada tipo tiene sus propias características. Por ejemplo, se puede especificar que cierto enlace sea un sensor tipo GPS y otro enlace sea un sensor infrarrojo. A partir de ahí, cada uno de estos enlaces tendrá diferentes características configurables que Gazebo utilizará para generar la data correspondiente del sensor.

### 2.4.10 Collision

Un elemento tipo "collision" o "colisión" debe ser hijo de un elemento tipo enlace. Un elemento tipo colisión es la geometría de cómo el motor de simulación física de movimiento de Gazebo procesará las interacciones de un enlace con el entorno. Puede ser tan simple como un cubo o como tan compleja como un brazo robótico fiel a su implementación real.

### 2.4.11 Visual

Un elemento tipo "visual" debe ser hijo de un elemento tipo enlace. Un elemento tipo visual es la geometría de cómo el motor gráfico de Gazebo muestra la apariencia del enlace en pantalla. Al igual que un elemento de colisión, puede ser tan simple como un cubo o como tan compleja como un brazo robótico fiel a su apariencia real. Note que, un enlace puede tener una geometría de colisión distinta a su geometría visual.

### 2.4.12 Plugin

Un elemento tipo "plugin" debe ser hijo de un mundo, modelo o sensor. Un elemento tipo *plugin* es un código de programación cargado dinámicamente en Gazebo. De forma práctica, mediante un *plugin* se puede modificar las configuraciones y procesos de Gazebo con respecto a un mundo, modelo o sensor a cualquier nivel de profundidad.

### 2.4.13 Ejemplo

Con el fin de ejemplificar los conceptos más básicos aquí explicados, se adjunta la Fig.2.4. El modelo del brazo robótico mostrado, se compone a nivel de formato SDF de cinco enlaces y cuatro articulaciones. El primer enlace es la base rectangular azul y el cilindro rojo, este enlace se conecta mediante una articulación de tipo rotacional con el segundo enlace, la barra amarilla horizontal. Esta articulación permite que el segundo enlace gire - alrededor de un eje y punto determinado - con respecto al primero. Luego la barra amarilla horizontal se conecta mediante una articulación de tipo rotacional con el tercer enlace, que se conforma por dos cilindros rojos y una barra amarilla. Este tercer enlace, se conecta mediante una articulación de tipo prismática con el cuarto enlace, la barra amarilla vertical. Esta articulación permite que el cuarto enlace se desplace - por un eje y punto determinado - con respecto al tercer enlace. Por último, la barra amarilla vertical se conecta mediante una articulación fija con el último enlace. Esta articulación ocasiona que el pequeño cilindro rojo, no tenga movimiento relativo a la barra amarilla vertical.

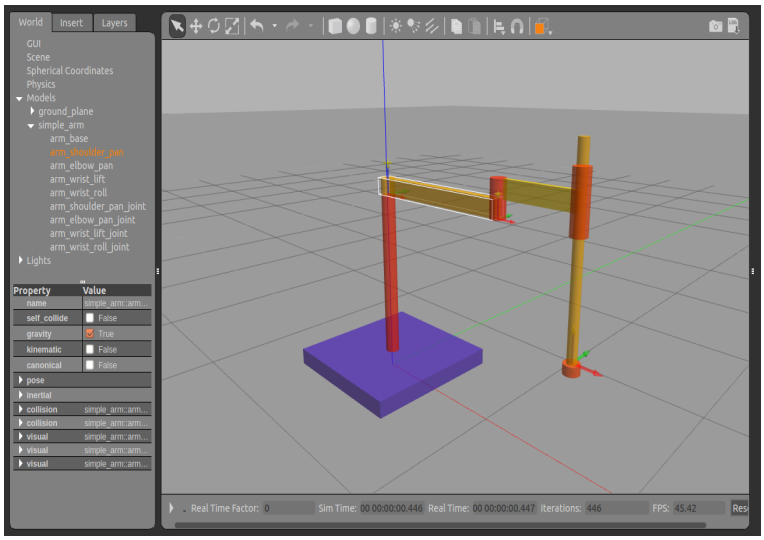



**Figura 2.4:** Modelo de un brazo robótico simple en Gazebo. Fuente: Elaboración Propia.

## 2.5 Robot Operating System

### 2.5.1 Generalidades

Robot Operating System (ROS) es un programa *software* de libre uso desarrollado por la comunidad de investigación robótica a lo largo del mundo. Es un marco de referencia flexible capaz de crear aplicaciones *software* para robots mediante la adaptación de estructuras de datos pre-diseñadas. Provee a las plataformas robóticas (incluyendo simulaciones y *hardware*) con herramientas y bibliotecas programáticas para controlar los comportamientos robóticos complejos de manera simple, [9]. A continuación, algunos

conceptos básicos del entorno de programación de ROS para el uso y desarrollo de sus diversas funcionalidades.

### 2.5.2 Packages

El *software* programático desarrollado en ROS se encuentra organizado en lo que se denominan "packages" o "paquetes". Cada paquete puede contener bibliotecas, ejecutables, *scripts* u otros elementos de data relevantes para constituir un módulo de aplicación robótica. Todo paquete en ROS tiene dos elementos. Un archivo package.xml que se le llama "manifest" o "manifiesto" el cual provee información meta sobre el paquete, ya sea el autor o responsable, el número de versión, licencias y dependencias entre paquetes. Y un segundo archivo CMakeList.txt en donde se especifican las direcciones de las bibliotecas y paquetes por compilar. La principal ventaja de los paquetes es que ponen a disposición aplicaciones robóticas pre-diseñadas para facilitar su reuso y adaptación.

### 2.5.3 Nodes

Un "node" o "nodo" es un proceso que realiza computación. Es decir, cálculos, funciones y comunicaciones. Es un ejecutable en ROS que puede comunicarse con otros nodos, subscribirse o publicar mensajes en "topics" o proveer y solicitar "services". Un nodo que envía un mensaje a un tema se denomina "publisher" o "emisor" y un nodo que escucha constantemente los mensajes publicados en un tema se le denomina "subscriber" o "subscriptor". Un nodo puede ser tanto emisor como subscriptor de varios temas.

### 2.5.4 Topics

Un "topic" o "tema" es un bus de comunicación por donde un nodo comunica información con otro nodo. En general, un nodo no es consciente de los nodos con los que se comunica -sino más bien de los temas en los que recibe o envía mensajes. Los temas son una comunicación de transmisión unidireccional y solo se pueden comunicar mensajes de un tipo.

### 2.5.5 Messages

Los nodos se comunican entre ellos publicando y escuchando "messages" o "mensajes" en temas. Un mensaje es simplemente una estructura de datos. Pueden ser de diversos tipos - mensajes tipo número entero, punto flotante, cadena de caracteres, booleanos, o mensajes anidados, entre otros.

### 2.5.6 Services

Un "service" o un "servicio" es un nodo, el cual puede mantener una comunicación bidireccional con otros nodos. Un servicio esta a la espera de que un nodo - llamado cliente- solicite sus procesos enviándole un mensaje, luego el servicio ejecuta sus rutinas y comunica un mensaje de respuesta al cliente. Para el cliente, el tipo de mensaje enviado puede ser distinto al mensaje recibido.

### 2.5.7 Ejemplo

Con el fin de ejemplificar los conceptos más básicos aquí explicados, se adjunta la Fig.2.5. En el ejemplo mostrado, existe un paquete en el entorno ROS, tres nodos, un tema y un tipo de mensaje. El primer nodo por notar es el llamado "ROS Master", este nodo siempre está presente en un entorno ROS, proporciona servicios de nomenclatura y registro para los demás nodos y realiza un seguimiento de emisores y subscriptores de temas y servicios. Su principal rol es permitir que los demás nodos se encuentren y comuniquen entre sí. El segundo nodo es llamado "Talker" y el tercer nodo "Listener". El tema en el entorno ROS mostrado se llama "Chat" y se envían mensajes "String".

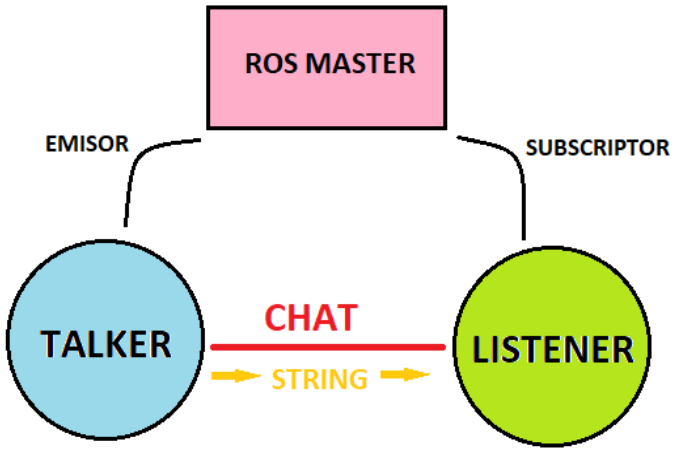



**Figura 2.5:** Ejemplo de un paquete de ROS a alto nivel. Fuente: Elaboración Propia.

Entonces, el funcionamiento del ejemplo mostrado consiste en lo siguiente. El nodo Talker se declara como un nodo emisor, que publica mensajes en el tema Chat. El nodo Listener se declara como un nodo subscriptor, que escucha constantemente mensajes en el tema Chat. Talker, en algún momento de su procesamiento publica un mensaje de tipo String en Chat y en ese momento Listener captura el mensaje y ejecuta sus procesamientos con ese argumento de entrada. Note que, Listener no puede publicar mensajes de vuelta pues esta declarado como un subscriptor y no como un servicio ni tampoco conoce cual nodo ha

enviado el mensaje, pues solo se relaciona con Chat. Todo este proceso de funcionamiento, es un paquete en el entorno ROS.

# Capítulo 3

# Modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3

## 3.1 Introducción

Una vez analizado el problema que se presenta, se ha dejado claro que el desarrollo de un modelo computacional que simule los comportamientos reales del sistema robótico Lego Mindstorms EV3 físico es una solución tangible. En este capítulo, se presentan todos los pormenores relacionados con la formación de dicha solución, se profundiza en la etapa de ejecución y se trabaja en los objetivos planteados. En esta sección, se muestra un acercamiento de la solución que brinda una primer idea de la metodología desarrollada. En las secciones siguientes, el lector podrá encontrar con mucho más detalle cada proceso llevado a cabo.

Para lograr la descripción del modelo computacional del *kit* robótico, se siguen las etapas mostradas en la Fig.3.1. La primer etapa consiste en obtener un modelo tridimensional digital del *kit* robótico, es decir un archivo de extensión .dae en el cual se encuentre el ensamble de la plataforma robótica diseñada. Luego, se procede a modificar dicho archivo para acondicionar la plataforma robótica modelada para su correcta inserción en Gazebo. Como segunda etapa se tiene la descripción del mundo de simulación en Gazebo que incluya la plataforma robótica como un elemento tipo modelo en el formato SDF. Como tercera etapa, se tiene la vinculación de Gazebo con ROS para manipular el flujo de la simulación y poner al usuario en control.

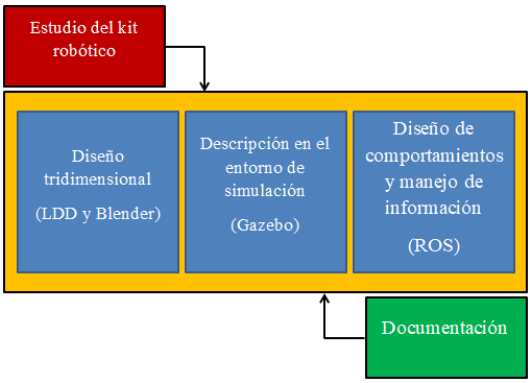



**Figura 3.1:** Esquema general del desarrollo del modelo computacional de simulación del *kit* robótico. Fuente: Elaboración propia.

## 3.2 Morfología y sistema sensorial del robot

Antes de empezar con el desarrollo del modelo computacional de simulación, es imperativo decidir una morfología y sistema sensorial del Lego Mindstorms EV3 con la que se va a trabajar. Junto a los profesores de la asignatura de robótica del grado en ingeniería informática de la Universidade Da Coruña y junto al Grupo Integrado de Ingeniería, se decide realizar el proyecto con la configuración "Lego Mindstorms Education EV3 Core Set (45544) Driving Base" con algunas pequeñas adiciones (ver Fig.3.2). Esta decisión se justifica por lo siguiente:

- Dicha configuración es la que utilizan de manera más recurrente en la asignatura de robótica y es en general la más común, por lo que el número de estudiantes potencialmente beneficiados es mayor.

- Dicha configuración trabaja de manera simultánea con los cuatro sensores disponibles. Esta condición aumenta el alcance educativo y experimental de la plataforma robótica.

- Dicha configuración es versátil y práctica. Pero más que todo, es la que mejor se desempeña para tareas de movimiento. Las demás configuraciones no tienen un sistema de movimiento eficiente y los sensores no juegan un rol significativo. Con esta configuración, se pueden realizar prácticas como de seguimiento de línea, navegación autónoma, recolección de datos, entre otras. Tiene mayor número de acciones útiles de interés educativo e investigativo.

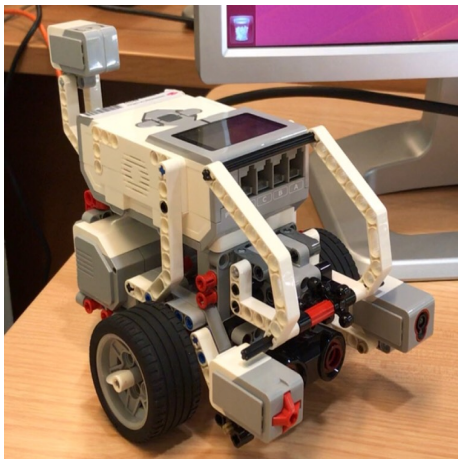


**Figura 3.2:** Configuración del Lego Mindstorms EV3 por modelar. Fuente: Elaboración propia.

## 3.3 Diseño tridimensional del robot

Una vez definida la morfología y sistema sensorial de la plataforma robótica, y en general su apariencia y constitución, se procede a diseñar su modelado tridimensional. Este modelado tridimensional no es más que una serie de construcción de *meshes* los cuales describen la forma y textura de cada pieza que constituye al sistema robótico. Sin embargo, cada pequeña pieza o bloque Lego tiene una alta complejidad de forma lo cual dificulta su modelado tridimensional en un *software* tradicional para esto. Además, existe escasa información oficial sobre las dimensiones de cada pieza.

Se utiliza entonces, el *software* Lego Digital Designer. Como se mencionó anteriormente, con LDD no es necesario modelar tridimensional-mente cada bloque Lego ya que, este *software* provee una extensa biblioteca de estos, los cuales tienen las dimensiones y texturas iguales a los fabricados en físico. Siguiendo la guía disponible en [15] sobre la metodología de construcción para "Driving Base" se ensambla pieza por pieza la morfología y sistema sensorial de la configuración escogida. Se realizan las adiciones o ajustes necesarios para obtener un modelo computacional tridimensional completo y fiel a la plataforma robótica -como la adición de los sensores y del motor mediano.

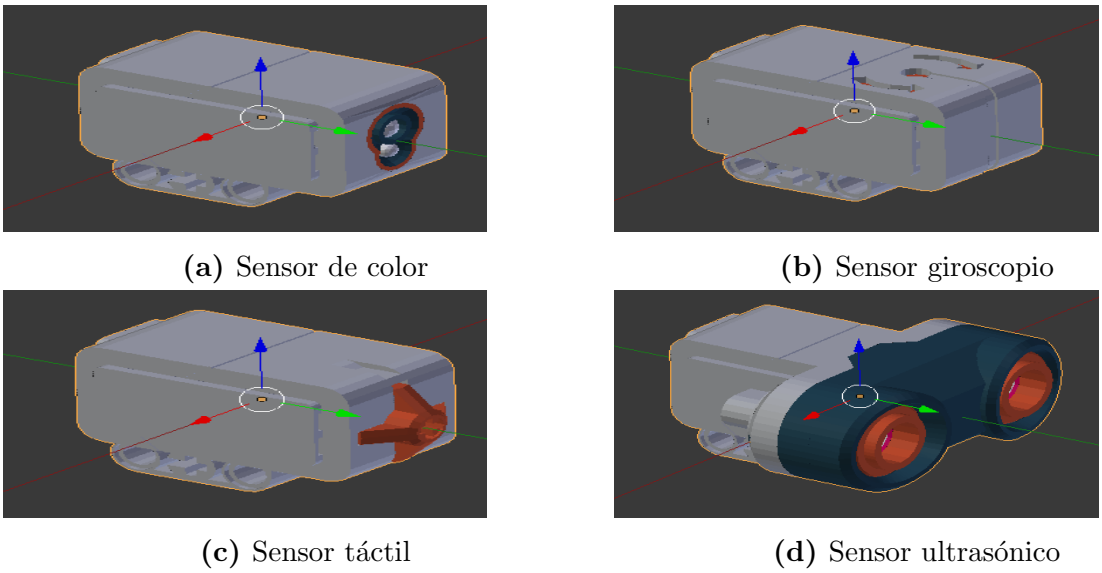
El formato SDF solo permite – para *meshes* de plataformas robóticas personalizadas - la extensión .dae y el archivo del modelo tridimensional resultante de Lego Digital Designer tiene una extensión .ldr. Cuando se obtiene la configuración deseada, se guarda el archivo digital a formato .ldr y se importa al *software* LeoCad, [16]. Seguidamente en LeoCad, se exporta el archivo a la extensión .dae. El uso de este *software* es únicamente para este proceso, por lo que no se consideró de interés explicar sus pormenores. Finalmente, se obtiene entonces una archivo digital del modelo tridimensional de la plataforma robótica deseada, en una extensión de formato compatible con Gazebo.

Si bien se obtiene un archivo digital de extensión .dae, es necesario aún procesar el modelo tridimensional - para lo que se procede a usar Blender. El principal objetivo de este *software* es preparar los segmentos por utilizar del modelo tridimensional creado en Lego Digital Designer para facilitar su ensamble en Gazebo. A continuación, las tareas desarrolladas en Blender.

1. Lo primero por considerar es que el modelo tridimensional actual consiste en un ensamble de muchas partes separadas, esto no es conveniente pues hay muchos ladrillos Lego que no tienen ningún rol dentro de los comportamientos de la plataforma robótica - además de brindar estabilidad mecánica. Esta situación ocasionaría una carga computacional innecesaria en la simulación en Gazebo, pues entre mayor cantidad de piezas, mayor número de cálculos. Para evitar esto, se identifican las partes esenciales de la plataforma robótica y se unen todos los ladrillos Lego que la componen para generar un sub-ensamble. Luego, se genera un archivo digital para cada una de estas partes por separado. Las partes que se consideran esenciales son el chasis, los cuatro sensores, las dos ruedas, la rueda giratoria esférica y el brazo actuador (ver Fig.3.3 y Fig.3.4).

2. Una vez que se cuenta con un archivo .dae para cada sub-ensamble de la plataforma robótica, se debe reubicar su eje de referencia hacia su respectivo centro de masa y luego, trasladar el centro de masa hacia el origen geométrico del entorno de Blender. Esto es importante por dos motivos, el primero es para facilitar el re-ensamble de los sub-ensambles en Gazebo y el segundo es para que cuando se inserten a Gazebo y se les atribuya propiedades inerciales, Gazebo ubique y calcule las interacciones físicas con el correcto punto de masa - esto se aclarará en la sección de Gazebo.

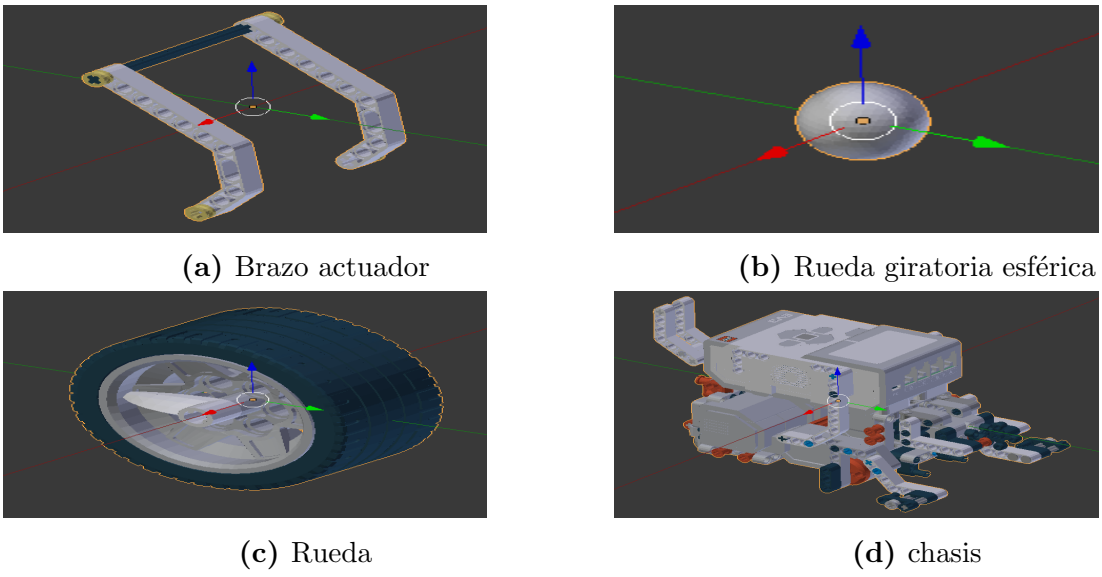
3. Por último, se modifica la escala de todos los sub-ensambles para que las piezas tengan la dimensión fiel a su real y se especifican las unidades en metros -pues Gazebo trabaja estrictamente con el Sistema Internacional de Unidades (SI). También, se orientan los ensambles de manera conveniente para reducir el trabajo en la descripción como modelo en el formato SDF.



**(a)** Sensor de color **(b)** Sensor giroscopio

**(c)** Sensor táctil **(d)** Sensor ultrasónico

**Figura 3.3:** Modelo tridimensional de los sensores del Lego Mindstorms EV3. Fuente: Elaboración propia.



**(a)** Brazo actuador **(b)** Rueda giratoria esférica

**(c)** Rueda **(d)** chasis

**Figura 3.4:** Modelo tridimensional de los elementos mecánicos del Lego Mindstorms EV3. Fuente: Elaboración propia.

Cabe rescatar, que se puede considerar reducir el conteo de triángulos o caras de los *meshes* para reducir la exigencia y recursos computacionales de la simulación, mediante las herramientas de modelado tridimensional que ofrece el *software* Blender. Sin embargo, esto no fue necesario -se analiza en el capítulo 4. Finalmente, se obtiene el modelo tridimensional de la plataforma robótica y se encuentra debidamente acondicionado para su sincronización con Gazebo.

## 3.4 Descripción del robot en el entorno de simulación

Los archivos obtenidos en la sección anterior, no son más que piezas simples las cuales se apegan a nivel de dimensión y apariencia a partes de la plataforma robótica real. No poseen propiedades físicas como masa, inercia o ejes de revolución que permitan establecer relaciones de movimiento, ni tampoco son capaces de medir un entorno programático simulado. Es decir, por ahora con lo único que se cuenta es con modelos tridimensionales, fiel a sus semejantes reales pero inanimados. Gazebo, permite superar estas barreras proporcionando - gracias a sus motores gráficos e interacciones de movimiento - un entorno acondicionado para la simulación, capaz de transformar unas piezas inanimadas a una plataforma robótica que funciona como conjunto.

El objetivo de esta sección es entonces, diseñar un mundo de simulación que incluya la plataforma robótica como un modelo que interactué con su entorno programático fiel a como lo dictan las leyes físicas a su semejante real, con la capacidad de movimiento de sus actuadores y con la obtención de información de los sensores. Se procede a ensamblar la plataforma robótica pieza por pieza, considerando las propiedades físicas de cada una y sus relaciones de movimientos relativas a las demás. Para conseguir esto, se aprovecha el formato SDF al describir todas las piezas como uno de los elementos disponibles en el mismo, explicados en el capítulo 2. Se busca que la plataforma robótica sea un elemento tipo modelo, conformado por todas sus piezas como enlaces, relacionadas por medio de articulaciones.

### 3.4.1 Ensamblaje de los componentes mecánicos de la plataforma robótica

Cabe aclarar que cuando se habla de piezas mecánicas se hace referencia a las piezas tridimensionales que dentro de la plataforma robótica solo tienen un funcionamiento mecánico, ya sea de soporte o movimiento. Como por ejemplo, las ruedas, el chasis o las que se muestran en la Fig.3.4. Estas piezas se describen en el formato SDF como un elemento tipo enlace, para los cuales es esencial especificar propiedades de masa, inercia, pose y tramas y sus elementos de geometría de colisión y visual - aunque cabe rescatar que, estas propiedades también es necesario especificarlas para los elementos tipo sensor. Una vez descritas de tal manera, Gazebo es capaz de simular su comportamiento en el entorno programático. Esto es claro, pues al saber dichas propiedades de un cuerpo y las fuerzas con las que interactúa, se puede calcular su cinemática y dinámica. A continuación, se muestra el flujo de trabajo seguido para cada pieza:

1. Debido al tratamiento de cada pieza en Blender, las piezas tienen su eje de referencia y sus tramas asignadas de manera conveniente para asignar su pose en el conjunto. Es decir, para ensamblar la plataforma robótica como tal, se debe colocar cada pieza tridimensional en su debida ubicación y orientación. Al Blender permitir acceder a la información de los *meshes* de cada pieza, se logra obtener y relacionar la pose de las

piezas en relación con las demás. Se conoce, entonces, con exactitud la pose a la que debe ir cada pieza para conformar la plataforma y dicha información se especifica en el debido apartado en el formato SDF. Como detalle importante, el eje de referencia para todas las piezas está ubicado en su centro de masa (calculado por Blender), esto por que Gazebo considera las propiedades inerciales desde el eje de referencia de la pieza y si el eje de referencia de la pieza no se encuentra en su centro de masa, entonces dichas propiedades inerciales no serán las adecuadas. Como ejemplo de este paso, se adjunta la Fig.3.5, en dicha figura se aprecia como el eje de referencia de cada pieza se ubica en el centro de masa de la misma y en la Fig.3.6 se aprecia como se llega a conocer la ubicación exacta para cada pieza, gracias a poder acceder a la información de los *meshes* (y cada punto que lo conforma). También, en la Tabla.3.1. se muestra la pose final de cada pieza en el conjunto robótico.

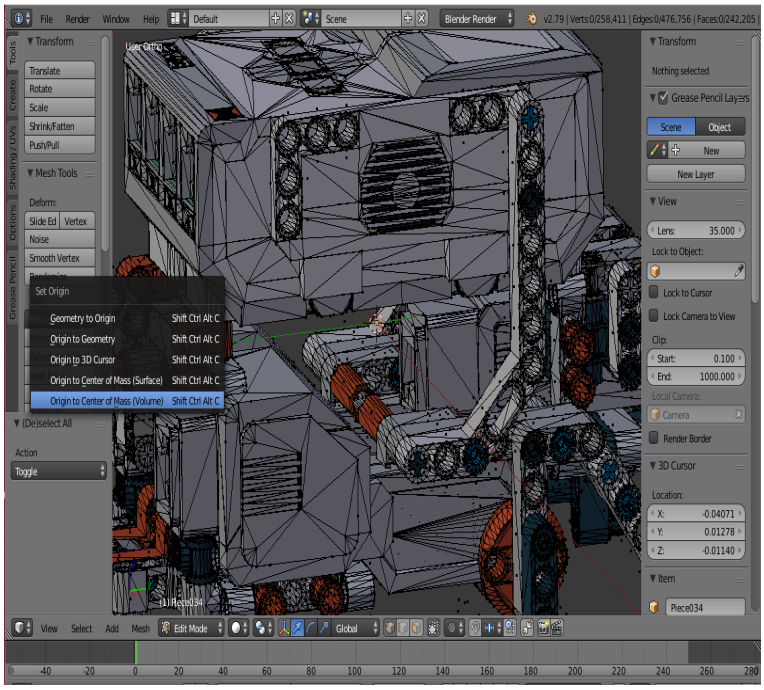



**Figura 3.5:** Ubicación del eje de referencia en el centro de masa para la pieza del chasis. Fuente: Elaboración propia.

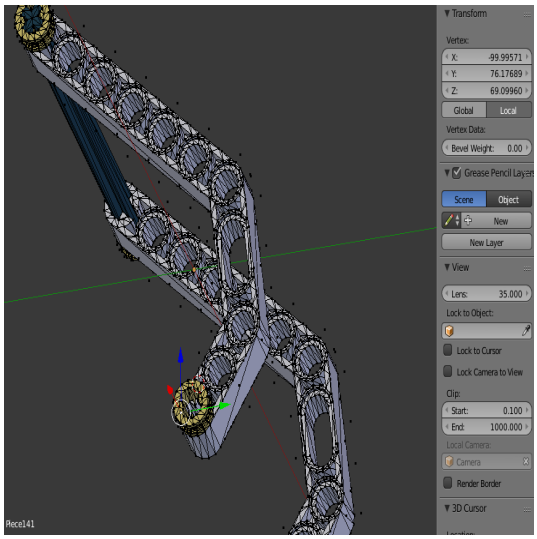



**Figura 3.6:** Proceso para encontrar el punto exacto de ubicación para el brazo actuador. Fuente: Elaboración propia.

| Pieza | Ubicación (x,y,z) [m] | Orientación (*roll*, *pitch*, *yaw*) [rad] |
|---|---|---|
| Brazo actuador | -0.00021, 0.08363, 0.03085 | 0, 0, 0 |
| chasis | 0, 0, 0 | 0, 0, 0 |
| Rueda Derecha | 0.06653, 0.02888, -0.04029 | 0, 0, 0 |
| Rueda Izquierda | -0.06653, 0.02888, -0.04029 | 0, 0, 3.14159 |
| Rueda giratoria esférica | -0.00021, -0.07499, -0.05629 | 0, 0, 0 |
| Sensor de color | -0.05627, 0.08821, -0.02597 | 0, 0, 0 |
| Sensor giroscopio | -0.00029, -0.09333, 0.07106 | 0, 0, 0 |
| Sensor táctil | 0.05574, 0.08779, -0.02612 | 0, 0, 0 |
| Sensor ultrasónico | -0.00026, 0.06814, -0.04048 | 0, 0, 0 |

**Tabla 3.1:** Pose de las piezas de la plataforma robótica. Fuente: Elaboración propia.

2. Para conocer la masa de cada una de las partes de la plataforma robótica, se utiliza una báscula MS 2045 Mettler Toledo para realizar las mediciones. Se realizan tres mediciones para cada conjunto de bloques Lego que conforman la pieza modelada tridimensional-mente y luego se promedia su valor y se especifica dicho dato en el formato SDF. En la Tabla.3.2 se aprecian las mediciones obtenidas.

   Una vez obtenida la masa, y al conocer las dimensiones de la pieza se procede a calcular el tensor de inercia para cada pieza. Para esto existen dos acercamientos, el primero es utilizar las herramientas del *software* MeshLab [4] y el segundo es aproximar la forma de cada pieza con sus dimensiones máximas a una figura geométrica simple (como un cubo, cilindro o esfera) y calcular el tensor de inercia para esta figura geométrica. La limitación de este último acercamiento es que piezas como el chasis, al ser aproximadas como un cubo pierden exactitud en el cálculo, pues las

| Pieza | Med 1 [g] | Med 2 [g] | Med 3 [g] | Promedio [g] |
|---|---|---|---|---|
| Brazo actuador | 8.3964 | 8.3967 | 8.3967 | 8.3966 |
| Chasis | 0.59759 | 0.59759 | 0.59760 | 0.59759 |
| Rueda | 23.6076 | 23.6074 | 23.6073 | 23.607 |
| Rueda giratoria esférica | 26.9103 | 26.9102 | 26.9102 | 26.9102 |
| Sensor de color | 13.6444 | 13.6444 | 13.6445 | 13.6444 |
| Sensor giroscopio | 12.5519 | 12.5517 | 12.5516 | 12.5517 |
| Sensor táctil | 15.0369 | 15.0368 | 15.0368 | 15.0368 |
| Sensor ultrasónico | 24.0403 | 24.0404 | 24.0402 | 24.0403 |

**Tabla 3.2:** Mediciones de masa de las piezas de la plataforma robótica. Fuente: Elaboración propia.

diferencias entre el chasis y un cubo son evidentes. Es por esto que se decide utilizar el *software* MeshLab para calcular con exactitud el tensor de inercia de las piezas y posteriormente confirmarlo con el cálculo de las figuras geométricas simples. Cabe considerar dos aspectos, el primero es que MeshLab permite calcular el tensor de inercia de un *mesh* mediante una aproximación triangular [8], en la cual asume una densidad unitaria y por lo tanto una masa igual al volumen. Esto quiere decir de que al tensor de inercia obtenido, es necesario realizar un ajuste posterior que incluya el verdadero valor de masa. La segunda es que MeshLab indica información geométrica del *mesh* con una precisión de 6 dígitos, esto es poco favorable pues las piezas al ser pequeñas se limita grandemente la precisión del tensor de inercia obtenido. Entonces, se procede a escalar en MeshLab todas las piezas por un factor de 10 y al tensor de inercia obtenido aplicar la transformación descrita por 3.1, para reajustar la escala y calcular con el correcto valor de masa.

$$I_{gz} = I_{ml}\frac{m}{Ve^2} \tag{3.1}$$

Siendo "Igz" el tensor de inercia correcto y el especificado en el formato SDF, "Iml" el tensor de inercia calculado por MeshLab, "m" el valor de masa de la Tabla .3.2 y "e" el factor de escala. Finalmente, se obtiene el tensor de inercia para cada pieza, mostrado en la Tabla 3.3 . Cabe rescatar que las unidades son

$$kgm^2 10^- 7$$

y se han decidido no mostrar en tabla por una cuestión de tamaño y con cinco cifras significativas.

3. Si bien cada pieza tiene ahora propiedades de pose, masa e inercia, todavía es necesario especificar su geometría de colisión y su geometría visual - explicadas en el capítulo 2. Idealmente, para un enlace de una plataforma robótica, se obtiene una simulación más exacta si la geometría de colisión es fiel a la forma real del enlace y es visualmente atractiva si la geometría visual también lo es. Pero, existen dos cuellos

| Pieza | Ixx | Iyy | Izz | Ixy | Ixz | Iyz |
|---|---|---|---|---|---|---|
| Brazo actuador | 62.663 | 92.013 | 134.45 | 0 | 0 | 16.731 |
| Chasis | 16919 | 8524.0 | 15164 | 37.720 | 45.478 | 185.45 |
| Rueda | 92.541 | 65.176 | 65.176 | 0 | 0 | 0 |
| Rueda giratoria esférica | 8.7189 | 8.7189 | 8.7189 | 0 | 0 | 0 |
| Sensor de color | 27.107 | 18.637 | 29.819 | 0 | 0 | 0 |
| Sensor giroscopio | 24.936 | 14.225 | 22.761 | 0 | 0 | 0 |
| Sensor táctil | 34.084 | 17.042 | 31.477 | 0 | 0 | 0 |
| Sensor ultrasónico | 59.961 | 76.308 | 104.856 | 0 | 0 | 0 |

**Tabla 3.3:** Tensor de inercia para cada pieza de la plataforma robótica. Fuente: Elaboración propia.

de botella por considerar. El primero es la capacidad de procesamiento requerida para simular una geometría de colisión exacta, entre más apegada sea la geometría de colisión al modelo tridimensional de la pieza, mayor poder de procesamiento será necesario para realizar los cálculos de sus comportamientos e interacciones con el entorno programático. El segundo es la capacidad de la tarjeta gráfica requerida para simular una geometría visual exacta, entre más apegada sea la geometría visual al modelo tridimensional de la pieza, mayor exigencia a la tarjeta gráfica, pues se deberá mostrar en pantalla una construcción de *meshes* compleja en constante movimiento. De forma análoga, entre más sencilla sea la geometría de colisión y la geometría visual - si bien se requerirá menor poder computacional -, como por ejemplo modelar el chasis como un cubo, se pierde exactitud en la simulación comparado con los comportamientos, interacciones y apariencia de la plataforma robótica real. Estos factores se consideran a la hora de decidir que geometrías se utilizan para cada pieza, sin embargo, se decide eventualmente utilizar los modelos tridimensionales diseñados para cada pieza para tanto la geometría de colisión como para la geometría visual. Es decir, cada pieza tiene una geometría de colisión y visual lo más apegada posible a la realidad. Si bien esto supone una exigencia mayor de procesamiento y de tarjeta gráfica, en el capítulo 4 se podrá apreciar que esto no supone ningún problema.

4. Las piezas mecánicas como las ruedas y la rueda giratoria esférica necesitan una especificación más, fricción. El formato SDF en Gazebo, permite especificar el coeficiente de fricción entre superficies. Para conseguir un comportamiento dinámico de movimiento realista es necesario especificar valores de fricción acordes y para esto, se estudia el material de las superficies que experimentan la fricción. La especificación máxima que se logra encontrar sobre el material de las ruedas del Lego Mindstorms EV3 y de la rueda esférica giratoria - según [11]- es que la superficie externa de las ruedas esta hecha de gaucho y la rueda giratoria esférica de metal. Si bien es poca información con la que se cuenta sobre dichos materiales, con dicha es suficiente para aproximar un coeficiente de fricción que si bien no es exacto al real,

cumple con el sentido práctico de la simulación. Luego, se acuerda junto al GII que el espacio de trabajo sobre el cual se trabaja con la plataforma robótica es mayormente cerámica común, por lo que se procede a buscar el coeficiente de fricción entre estos materiales. En la Tabla.3.4 se muestran los coeficientes de fricción especificados para dichos enlaces.

| Pieza | Coeficiente de fricción |
|---|---|
| Ruedas (gaucho - cerámica) | 0.8 |
| Rueda giratoria esférica (metal - cerámica) | 0.1 |

**Tabla 3.4:** Coeficientes de fricción para las superficies en contacto para el movimiento de la plataforma robótica. Fuente: [1]

Finalmente, se cuenta con una descripción en el formato SDF de las propiedades mecánicas esenciales para cada enlace de la plataforma robótica. Sin embargo, note que si bien se especificaron dichas propiedades de los enlaces todavía no se conoce la relación de movimiento entre los mismos ni tampoco se cuenta con algún medio capaz de medir alguna variable del entorno programático. Estas dos funcionalidades son las que se explican a continuación.

### 3.4.2 Ensamblaje de los componentes electrónicos de la plataforma robótica

Cabe aclarar que cuando se habla de componentes electrónicos se hace referencia a las piezas tridimensionales que dentro de la plataforma robótica - si bien también cumplen con un funcionamiento mecánico - cumplen con un rol electrónico. Como por ejemplo, el sensor ultrasónico, el sensor táctil o las que se muestran en la Fig.3.3. Estas piezas tienen un elemento en particular, adicional a las mecánicas en su descripción en el formato SDF, el elemento tipo sensor - explicado en el capítulo 2. Una vez el elemento sensor descrito en el enlace, el enlace es capaz de generar información correspondiente a la medición de alguna variable programática en la simulación. Es decir, se ensambla un sensor en la simulación.

Gazebo -al ser una plataforma de simulación robótica-, tiene disponible un conjunto de sensores descritos en el formato SDF. Al escoger Gazebo como el *software* de simulación, se presenta la limitación de utilizar únicamente los sensores presentes en su catálogo. En general, se cuentan con diversos sensores multi-propósito que se ponen a disposición del usuario con características y parámetros configurables para que mediante la correcta especificación se adecue al sensor que se desea simular. Por ejemplo, se cuenta con un elemento sensor tipo cámara pero, cada cámara es distinta pues tiene diferente campo de visión o resolución. Entonces, prácticamente se parte desde un sensor base, al cual se le configuran sus funcionalidad y parámetros electrónicos de tal manera en que se asemeje

lo más posible al sensor real y así lograr una simulación más precisa. Se plantea una situación en donde se debe decidir qué sensor del catálogo de Gazebo puede modelar de mejor manera los sensores del Lego Mindstorms EV3 y a partir de ahí especificar sus propias características electrónica según su hoja de datos, en la descripción como elemento sensor del formato SDF. A continuación, se detalla el estudio y las decisiones para cada sensor.

### Sensor de color

Lo primero por notar es que el catálogo de sensores de Gazebo no ofrece un sensor capaz de medir luz ambiente de forma directa. Esto obliga a escoger un sensor que si bien no es propiamente un sensor de luz ambiente, pueda cumplir su rol práctico en la simulación. Junto al GII - y luego de una exhaustiva investigación-, se decide realizar la simulación del sensor de color del Lego Mindstorms EV3 mediante un elemento sensor tipo cámara en el formato SDF. Esta decisión se lleva a cabo debido a que se pretende -a partir de la imagen tomada por la cámara en el entorno de simulación programático- obtener información sobre la iluminación del ambiente en el que se encuentra la plataforma robótica funcionando. Para esto, se entra en un proceso de mapear las características electrónicas del sensor de luz ambiente provenientes de la hoja de datos a características configurables del elemento sensor tipo cámara de Gazebo.

Si bien se cuenta con información detallada sobre los parámetros del sensor de color -gracias a la documentación del Lego Mindstorms EV3-, mayor parte de estos parámetros no son considerados en el ensamblaje del elemento cámara. Esto debido a que, la limitación de parámetros electrónicos esta definido por la descripción del elemento cámara en el formato SDF. Con esto se quiere decir que, -por ejemplo- aunque se sepa que el sensor encuentra su mayor sensibilidad a una longitud de onda de 900 nm, esta especificación no se encuentra disponible en la descripción de la cámara, por lo que se desprecia. No especificar estas características provoca que la medición y funcionamiento del sensor simulado pierda exactitud con su semejante real, pero es un compromiso que se acepta al ser esta una limitación de herramientas disponibles y no de diseño. Sin embargo, existen otras parámetros configurables del elemento sensor tipo cámara que se pueden mapear fácilmente desde los parámetros electrónicos obtenidos de la hoja de datos del sensor de color real, de manera confiable. Y también, para aquellos parámetros que son necesarios para simular un sensor cámara que no se encuentran en la información del sensor real, se realizan estudios y mediciones con el sensor para intentar aproximarlas. En el formato SDF, para un enlace con un elemento sensor tipo cámara, se debe especificar:

1. Frecuencia de generación de información: Es un valor numérico en hertz que indica la cantidad de imágenes por segundo que se capturan del campo de visión. Este parámetro se puede mapear directamente con la frecuencia de funcionamiento del sensor de color, cuyo valor es 1000 Hz [12]. Recalcando la diferencia de que el sensor real toma medidas de luz ambiente mientras que el sensor simulado captura

imágenes.

2. Campo de visión: Es un valor numérico en radianes que indica el ángulo de visión de la cámara. Este parámetro se puede mapear directamente con la intensidad relativa del sensor de color, cuyo valor promedia -de forma conservativa- 0.125 [12], (ver Fig.3.7).

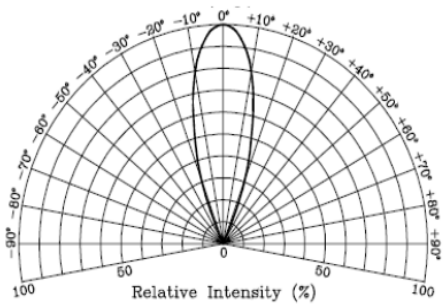



**Figura 3.7:** Intensidad relativa del sensor de color del Lego Mindstorms EV3. Fuente: [12].

3. Resolución: Se trata de dos valores numéricos los cuales indican la cantidad de píxeles lógicos presentes en la imagen capturada de manera vertical y horizontal. Este parámetro no se logra mapear con ninguna característica del sensor real. Junto al GII, se acepta simplemente establecer una resolución relativamente baja debido a que se puede considerar que una imagen distorsionada es lo más parecido que debería ver un sensor de luz ambiente. Por lo que se acuerda en describir una resolución de 240x240.

4. Clip: Se trata de dos valores que indican la distancia mínima y máxima a la que la cámara va a renderizar la imagen. Es decir, si el clip máximo es de 2m, un objeto a 3m no va a ser visto por la cámara y por ende no aparecerá en la imagen capturada. El valor de esta característica, si bien no se extrae directamente de la hoja de datos del sensor real, se puede realizar un estudio a dicho sensor para determinar a qué distancia deja de sensibilizar un rayo de luz. Para esto, de manera sistemática, se toman mediciones -cada 10 cm- del sensor de color en una habitación de baja iluminación cuando apunta hacia él una bombilla omnidirectional de 12 W y 1.144 lm. En la Tabla. 3.5 se aprecian las mediciones tomadas y en la Fig.3.8 se muestran de manera gráfica.

| Distancia [cm] | Intensidad medida [ptc] |
|---|---|
| 10 | 38 |
| 20 | 36 |
| 30 | 31 |
| 40 | 26 |
| 50 | 23 |
| 60 | 18 |
| 70 | 17 |
| 80 | 14 |
| 90 | 10 |
| 100 | 9 |
| 120 | 7 |
| 140 | 5 |
| 160 | 4 |
| 180 | 3 |
| 200 | 2 |

**Tabla 3.5:** Mediciones de la intensidad relativa del sensor de color en modo luz ambiente cada 10 cm para una bombilla omnidireccional de 12 W y 1.144 lm. Fuente: Elaboración propia

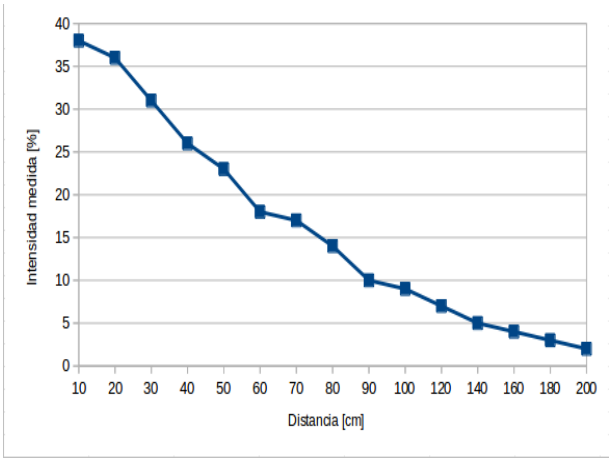



**Figura 3.8:** Curva descrita por las mediciones del sensor de color. Fuente: Elaboración propia.

A una distancia de 2m, el ruido predomina sobre la medición y la luz de la bombilla deja de tener mayor efecto sobre la medición del sensor. Por lo que - de manera conservativa- se decide que el clip máximo sea igual a 2m. Para el clip mínimo, [12] indica que a la distancia mínima de lectura de medición del sensor es de aproximadamente 0.0159 m (la mitad de la longitud de un bloque Lego clásico, [3]), por lo que se decide que el clip mínimo sea igual a 0.0159 m.

Cabe nuevamente rescatar, que este mapeo no es algo que se encuentre justificado en libros de electrónica, pues son sensores totalmente distintos. Si no que, mediante un proceso de prueba y error, se determinó la configuración adecuada para -de manera práctica y funcional- simular el sensor de luz ambiente. Es decir, se adaptaron las herramientas con las que se cuentan a las necesidades por cumplir, aceptando sus limitaciones y diferencias - pero con el objetivo de conseguir un comportamiento semejante.

### Sensor giroscopio

Gazebo cuenta en su catálogo de sensores descritos en el formato SDF con un elemento sensor tipo IMU. Este caso es todo lo contrario al anterior, el sensor disponible en Gazebo es justamente el sensor que se busca. Sin embargo, sucede que el sensor en simulación ofrece mayores parámetros configurables que el propio comportamiento del sensor real y de lo que su información dicta. Como por ejemplo, el elemento sensor tipo IMU dispone al usuario información sobre la orientación del sensor en los tres ejes al igual que la velocidad angular y aceleración lineal que experimenta en el momento de la medición. Esto claramente el sensor real no lo realiza, en el Lego Mindstorms EV3 el usuario solo dispone de un valor numérico que representa el ángulo de inclinación del sensor con respecto al eje Z, sin embargo esta es una situación que se atacará en la siguiente sección de ROS. En el formato SDF, para un enlace con un elemento sensor tipo IMU, se debe especificar:

1. Frecuencia de generación de información: Es un valor numérico en hertz que indica cada cuanto se consulta la información del sensor IMU. Este parámetro se puede mapear directamente con la frecuencia de funcionamiento del sensor giroscopio, cuyo valor es 1000 Hz [12].

2. Velocidad angular: Es un valor numérico en rad/s que indica la velocidad angular máxima a la cual el sensor puede realizar una medición dentro de los criterios de confiabilidad del mismo. Por ejemplo, si el sensor gira a mayor velocidad que este valor, las mediciones no serán fidedignas. Según [12], un sensor ISZ-655 puede viajar a máximo una velocidad angular de aproximadamente 7.6792 rad/s, por lo que se procede a especificar este valor en la descripción del elemento en el formato SDF.

3. Aceleración angular y demás características: Si bien Gazebo da la opción de especificar estas otras características, no es realmente necesario hacerlo. Por lo que al desconocer - y ante la incapacidad de realizar un estudio sistemático para aproximarlas- se decide no incluirlas en la descripción. Sin embargo, la exactitud del sensor simulado con el sensor real no se ve comprometida por esta decisión (esto se tratará a fondo en la siguiente sección).

### Sensor táctil

El sensor táctil es el más sencillo de describir. Gazebo ofrece un elemento sensor tipo contacto, el cual da información sobre todos los contactos, fuerzas, momentos y puntos donde actúan, que experimenta el enlace. Sucede lo mismo del sensor giroscopio, donde el usuario en el EV3 dispone solo de un valor numérico binario el cual le indica si el botón se presionó o no (o se podría ver como si existe una fuerza actuando sobre una pared del sensor) mientras que el sensor en simulación ofrece información más profunda sobre el estado del enlace. Sin embargo, - como ya se mencionó- esta situación se tratará en la siguiente sección. En el formato SDF, para un enlace con un elemento sensor tipo contacto, se debe especificar:

1. Geometría de colisión: El sensor de contacto responde sobre una la geometría de colisión y no sobre el enlace, pues se conoce el estado de las fuerzas, momentos y puntos de aplicación desde el motor de física de Gazebo, y la geometría considerada para esos motores computacionales es la geometría de colisión. Por lo que simplemente se especifica la geometría de colisión como el modelo tridimensional del sensor táctil.

2. Frecuencia de generación de información: Es un valor numérico en Hz que indica cada cuanto se consulta la información de las fuerzas, momentos y puntos de aplicación que actúan sobre la geometría de colisión al motor computacional de cálculos físicos de Gazebo. Este parámetro se puede aproximar, considerando que su funcionamiento es semejante a un interruptor mecánico básico analógico. En teoría entre mayor sea la frecuencia de generación de información mayor exactitud tendrá el sensor simulado con respecto al real. Sin embargo, aquí entra el compromiso de exactitud de cálculo contra capacidad de procesamiento, por lo que se decide establecer este valor en 1000 Hz, con lo que se considera que el sensor simulado obtiene un comportamiento semejante al real con una carga computacional aceptable.

### Sensor ultrasónico

Dentro del catálogo de sensores disponibles en Gazebo, existen dos tipos que se pueden considerar para modelar un sensor ultrasónico. El primero es un elemento sensor tipo rayo y el segundo es un elemento sensor tipo sonar. Si bien se puede pensar que cualquiera de estos dos tipos de sensores pueden llegar a modelar un comportamiento semejante el sensor real, es necesario considerar las limitaciones de Gazebo con respecto al fenómeno físico que sucede. Como bien se conoce, un sensor ultrasónico basa su funcionamiento en la emisión y recepción de ondas ultrasónicas. De forma simple, el emisor emite una onda de ultrasonido a una frecuencia específica, la onda se refleja en el objeto más cercano y el receptor recibe la onda de vuelta. Al relacionar el tiempo que tardó la onda en regresar, se aproxima la distancia del objeto.

Gazebo no es capaz de simular este fenómeno físico, no existe en el entorno programático de Gazebo variables correspondientes a ultrasonido o reflexión de ondas, simplemente sus capacidades de simulación no llegan a tal profundidad capaz de considerar estas variables físicas. Es por esto que ambos tipos de sensores realizan la aproximación de la distancia al objeto - es decir, la medición del sensor - mediante un funcionamiento distinto, utilizando las herramientas disponibles en el entorno Gazebo. Por ejemplo, el sensor tipo sonar funciona declarando una geometría de colisión en forma de cono - semejante al cono de medición ultrasónico del sensor real-, luego accesa a la información de contacto entre esa geometría y el objeto y determina cual es el punto de contacto más cercano al sensor. Dicho punto será la medición del sensor. Mientras que el sensor tipo rayo traza un conjunto de líneas que en conjunto, forman una figura semejante a un cono y luego cuando se entra en contacto con un objeto, se busca la longitud de línea mas corta y esa es la medición del sensor. Más adelante se mostrará de manera gráfica este proceso.

Cabe rescatar, que independientemente del tipo de sensor que se decida utilizar, el comportamiento característico de un sensor ultrasónico resultado de la interacción física de la emisión y recepción de ondas, no se modela. Es decir, -por ejemplo- bien se sabe que si se intenta medir la distancia a la que se encuentra un objeto orientado a 45 grados con respecto al sensor ultrasónico, es probable que la medición sea inexacta, pues las ondas de sonido al reflejarse en el objeto a 45 grados, no regresan al receptor del sensor. O si bien, si se intenta medir la distancia entre el sensor y un objeto cuyo material absorbe mayor parte de las ondas de ultrasonido, la medición también será inexacta, pues nuevamente las ondas de sonido no regresan al receptor. Gazebo al no tener la capacidad de simular estos fenómenos físicos, estos comportamientos comunes del sensor real no se logran simular. Esta limitación se conoce, y sus implicaciones se estudiaran a profundidad en el siguiente capítulo.

Finalmente, se decide describir el sensor ultrasónico del Lego Mindstorms EV3 mediante un elemento sensor tipo rayo. Este tipo de sensor ofrece mayor cantidad de parámetros configurables en su descripción SDF los cuales se pueden ajustar para conseguir un comportamiento lo más semejante posible al sensor real. Además, en general es el que se encuentra mayormente trabajado por la comunidad de Gazebo, y es el más utilizado para aplicaciones de ultrasonido semejantes, por lo que se encuentra más información sobre su uso y configuración. En el formato SDF, para un enlace con un elemento sensor tipo rayo, se debe especificar:

1. Frecuencia de generación de información: Es un valor numérico en Hz que indica cada cuanto se estudia la longitud mínima de las líneas trazadas, es decir, la distancia entre el sensor y un posible objeto. Este parámetro se puede mapear de forma directa a la frecuencia de funcionamiento del sensor ultrasónico real. Según [12], para un objeto a una distancia de 250 cm, el sensor genera una nueva medición cada 15 ms - esto debido a que es el tiempo que tarda la onda en ser emitida y recibida -. Por supuesto, al un objeto estar más cerca del sensor real, la nueva medición puede generarse a una frecuencia mayor. Sin embargo, Gazebo solo admite un valor

de frecuencia para su generación de mediciones, por lo que se decide - de forma conservativa - utilizar una frecuencia de 66 Hz (1/15ms), dado que es el peor de los casos.

2. Ángulo de medición: Es un valor numérico en radianes que indica el ángulo mínimo y máximo al cual se traza el conjunto de líneas del sensor. Este parámetro se puede mapear de forma directa con el ángulo de funcionamiento del cono del sensor real. Según [12], el ángulo de medición es de 90 grados. Sin embargo, esto se determina experimentalmente que es erróneo, por lo que se decide realizar el siguiente estudio. Para determinar el ángulo de medición efectivo del sensor ultrasónico, se estudia la amplitud máxima a la que el sensor puede detectar obstáculos. Es decir, se coloca un objeto a una distancia determinada y luego se procede a desplazarlo hacia la izquierda y derecha hasta que la medición del sensor pierda suficiente exactitud. Se realiza este proceso de manera sistemática para diferentes distancias, con el fin de trazar manualmente el cono de medición del sensor. Algunas consideraciones importantes en este estudio es que se consideró que la amplitud máxima - ya sea para la izquierda o derecha- se determina en el momento en que la medición del sensor tiene un porcentaje de error igual -o cercano- al 10 porciento; o bien si antes de llegar al 10 porciento de error, la detección del objeto se pierde del todo. También, el objeto utilizado para las mediciones es una tabla lisa de madera la cual utilizan los estudiantes en la asignatura de robótica en la UDC. Por lo que el estudio del ángulo de medición del cono del sensor ultrasónico del Lego Mindstorms EV3 será conservativo a un 90 porciento y será válido solo para este material. Estas consideraciones se acuerdan con el GII, pues eventualmente lo que se busca es un funcionamiento práctico en simulación y no la realización de todo un estudio a profundidad del sensor. En la Tabla.3.6 se encuentra las mediciones tomadas y en la Fig.3.9 se aprecia el cono de medición obtenido, con su debida amplitud. Note que se realizan mediciones cada 10cm, hasta llegar a 100cm donde se empiezan a realizar mediciones cada 40cm.

| Distancia [cm] | Med(centro) [cm] | Med(izq) [cm] | Amplitud(izq) [cm] | Med(der) [cm] | Amplitud(der) [cm] | Ángulo de medición [°] |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 10.2 | 10.8 | 0.5 | 10.7 | -0.3 | 4.58 |
| 20 | 20.5 | 22 | 1.5 | 22 | -0.5 | 5.72 |
| 30 | 30.3 | 32.8 | 3.3 | 32.8 | -2.5 | 11.04 |
| 40 | 40.8 | 43.8 | 5.1 | 43.8 | -4.5 | 13.68 |
| 50 | 50.9 | 55.1 | 5.7 | 54.9 | -5 | 12.21 |
| 60 | 60.7 | 66 | 8.7 | 66 | -6.7 | 14.62 |
| 70 | 71 | 76.6 | 11.7 | 75.7 | -11 | 18.42 |
| 80 | 80.6 | 86.5 | 14.7 | 86 | -12.7 | 19.43 |
| 90 | 90.8 | 98 | 12.7 | 97 | -10.7 | 14.81 |
| 100 | 100.8 | 107 | 14.7 | 108 | -13 | 15.77 |
| 120 | 120.1 | 126.5 | 15 | 126 | -13.5 | 13.54 |
| 160 | 161.1 | 165.5 | 12.5 | 165.5 | -11.5 | 8.58 |
| 200 | 201.4 | 206 | 13.5 | 205 | -13 | 7.58 |
| 240 | 242 | 246 | 10.5 | 245.5 | -10 | 4.89 |

**Tabla 3.6:** Mediciones de la amplitud del sensor ultrasónico del Lego Mindstorms para el cono de medición. Fuente: Elaboración propia.

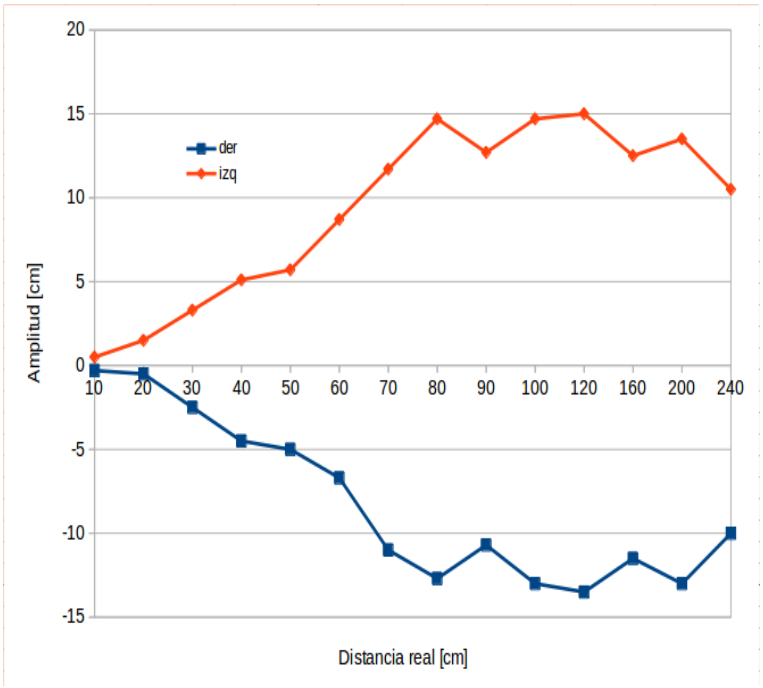



**Figura 3.9:** Representación gráfica del cono de medición. Fuente: Elaboración propia.

Como se puede apreciar, las medidas son irregulares. Esto plantea un problema, pues en la descripción SDF del elemento sensor tipo rayo, solo se puede especificar un valor para el ángulo. De tal manera en que se apreciaría en la Fig.3.9 como dos rectas con una pendiente constante y contraria. Se decide entonces, nuevamente considerar el sentido práctico de la simulación y establecer el ángulo de medición correspondiente a las mediciones más habituales del sensor y a sus condiciones de trabajo comunes. La plataforma robótica usualmente trabaja en espacios reducidos, demarcados por tablas de 80 cm de longitud. Junto al GII, se decide entonces que

las mediciones entre 40cm y 100cm son las de mayor recurrencia y por lo tanto se consigue un mayor sentido práctico si se especifica un ángulo de medición que mejor modele este intervalo. Entonces, en la Tabla.3.6 promediando el valor de ángulo de medición para las distancias 40, 50, 60, 70, 80, 90 y 100 se obtiene un resultado igual a 15.56 °. Se procede a especificar este valor en la descripción SDF.

3. Muestras y resolución: Es un valor numérico que indica de forma práctica la cantidad de líneas por trazar dentro del ángulo de medición especificado. Por la naturaleza de su funcionamiento, el valor de este parámetro se decidió de manera experimental, al observar qué valor consigue modelar de forma simple, el comportamiento que experimenta el sensor real cuando mide la distancia a un objeto a una orientación de por ejemplo 45 grados con respecto al mismo. La exactitud de la medición en la simulación es directamente proporcional a la cantidad de muestras y resoluciones especificadas. El efecto de esta decisión se estudiará a profundidad en el capítulo 4.

4. Rango de medición: Son dos valores numéricos que especifican los rangos mínimos y máximos de distancia de medición del elemento sensor. Este parámetro se puede mapear de forma directa con el mismo rango de medición del sensor real. Según [12], el sensor ultrasónico del Lego Mindstorms EV3 logra realizar mediciones de distancia a un objeto que se encuentre entre 3 cm a 250 cm. En la descripción SDF, se especifican los valores de rango mínimo igual a 0 cm y rango máximo igual a 255. Esto se debe a que más adelante, este rango será conveniente para modelar de mejor manera el comportamiento de la medición disponible para el usuario desde el bloque programable.

El efecto de la implementación de todas estas decisiones se estudiaran en el siguiente capítulo. Y si bien, la mayoría de las consideraciones particulares que se toman restan exactitud a la simulación, se estudiará también en el siguiente capítulo que en realidad no generan un impacto significativo en la fiabilidad de la simulación ni en su practicidad.

Cabe rescatar que, si bien se ha explicado el proceso de creación de los elementos electrónicos, nunca se habló sobre algunos parámetros eléctricos que en la realidad son fundamentales. Fenómenos físicos como la tensión de alimentación del sensor o la sensibilidad en la señal de salida, son variables que nunca se consideraron. Y esta situación no es debido a que simplemente se decidieron ignorar, es debido a que a nivel de simulación programática, en donde se cuentan con herramientas limitadas por parte del *software* utilizado, en realidad no son necesarias de estudiar. Para Gazebo, no existe tal cosa como tensión, pues solo debe habilitar el funcionamiento del sensor mediante la ejecución de un algoritmo. Al no existir el fenómeno físico de tensión en el entorno de simulación, tampoco existen sus semejantes - tales como sensibilidad o corriente, que a diferencia de la realidad, son parámetros realmente importantes. Otro factor importante, es el rango de temperatura de funcionamiento de cada sensor. Gazebo puede simular condiciones ambientales tales como temperatura y humedad. A partir de esto se podría programar un algoritmo que se encargue de distorsionar proporcionalmente las mediciones de los sensores en función a la

temperatura. Sin embargo, se ha decidido que esta funcionalidad no es práctica, pues la plataforma robótica nunca trabaja en dichas condiciones ambientales extremas y por lo tanto solo representaría una carga computacional sin ningún beneficio práctico a cambio. Un último aspecto por considerar, es el ruido característico de cada sensor. Esta variable es fundamental para que la simulación se apegue a la realidad y es una variable que no se puede ignorar. Se conoce la información de ruido para cada sensor y se encuentran las herramientas para implementarlo. Dicha adición se implementará en la próxima sección.

Finalmente, se obtiene una descripción completa en el formato SDF de los elementos electrónicos de la plataforma robótica. Para este punto, los diversos enlaces electrónicos del robot logran generar información sobre las condiciones del entorno, es decir, los sensores se encuentran funcionando y midiendo su variable correspondiente. Sin embargo, note que no precisamente la información que generan es fiel al formato o comportamiento que se cuenta en la plataforma robótica real. El sensor de contacto indica información sobre fuerzas y momentos que actúan en un punto del enlace, el sensor IMU indica información sobre la orientación de los tres ejes, el sensor cámara por el momento solo captura imágenes y el sensor ultrasónico, ¿acaso se ha considerado la precisión de la medición?. Todos estos aspectos se trabajaran más adelante, en la sección de ROS.

### 3.4.3 Ensamblaje de las relaciones de movimiento entre enlaces de la plataforma robótica

Las propiedades mecánicas y electrónicas se encuentran especificadas, lo único que hace falta es especificar en el formato SDF las relaciones de movimiento relativo entre cada pieza. Esto se logra, mediante elementos tipo articulación, explicado en el capítulo 2. De manera práctica, cada elemento tipo articulación indica una relación de movimiento relativo entre dos piezas. Esta relación de movimiento puede ser por ejemplo, de revolución - la cual permitiría que una pieza gire entorno a un eje con relativa a otra pieza- o prismática - la cual permitiría que una pieza se desplace sobre un eje relativo a otra pieza. Entonces se entra en un estudio de las interacciones entre cada pieza y se analizan como una verdadera plataforma robótica en donde se deben considerar tramas, poses, ejes de movimiento, entre otros. Lo primero por analizar, es que el chasis es la pieza central de la plataforma robótica por ensamblar, esto quiere decir que todos los movimientos de las demás piezas, estarán referenciadas a esta. Prácticamente el chasis es el eje de referencia para los elementos tipo articulación dentro de la descripción en el formato SDF. Lo segundo por analizar es que en este paso, se realiza una descripción implícita de los actuadores de la plataforma robótica. Esto debido a que, los actuadores en un robot son los que generan el movimiento de las piezas, y como en este paso se describe propiamente las condiciones de movimiento de cada pieza, se esta describiendo a su vez los actuadores presentes en la plataforma robótica. Esto ocasiona por ejemplo, que la descripción de las relaciones de movimiento sea más extensa en los componentes mecánicos que en los componentes electrónicos. Por supuesto, por que los componentes electrónicos, no juegan un rol significativo en el movimiento de la plataforma, mientras que los compo-

nentes mecánicos - como las ruedas o el brazo actuador- juegan un papel importante en el comportamiento de movimiento de la plataforma robótica. A continuación, se detalla el estudio y las decisiones para cada articulación.

### Rueda izquierda y derecha

Para estas piezas se debe utilizar una articulación tipo revolución. La articulación tipo revolución en el formato SDF, es una articulación de bisagra que gira en un solo eje con un rango de movimiento fijo o continuo. En este caso, interesa que ambas ruedas solo tengan un movimiento giratorio con respecto al chasis sobre un eje determinado y de movimiento continuo - es decir, de giro no limitado.

### Rueda giratoria esférica

Para esta pieza se debe utilizar una articulación tipo bola. La articulación tipo bola en el formato SDF, es un tipo de articulación sinovial en la que la superficie esférica de un hueso redondeado encaja en la depresión en forma de copa de otro hueso. El hueso distal es capaz de moverse alrededor del número indefinido de ejes, que tienen un centro común. En este caso, interesa que la rueda giratoria esférica gire libremente con respecto a su depresión en forma de copa presente en el chasis de la plataforma robótica, ya que de esta manera aporta estabilidad mecánica al robot sin comprometer su movimiento por fricción.

### Brazo actuador

Para esta pieza se debe utilizar también una articulación tipo revolución. Sin embargo hay un par de diferencias con respecto a las descritas para las ruedas. La primer diferencia es que se necesitan especificar dos articulaciones en la misma pieza, ya que el brazo actuador se une al chasis en dos puntos diferentes de apoyo. La segunda diferencia es que en lugar de ser un rango de movimiento continuo, es fijo. Esto se debe a que el movimiento del brazo tiene limitaciones físicas, no puede girar libremente los 360 grados debido a que - por la construcción (y objetivo) de la plataforma robótica - impactaría con otras piezas que le evitan continuar su movimiento. Es por esto que se estudia y se determina el rango de movimiento del brazo actuador en la plataforma robótica real. Mediante el uso del *encoder* del motor mediano, se observa que el brazo realiza un movimiento giratorio de aproximadamente 161 grados (2.84 rad) con respecto a sus puntos de apoyo en el chasis.

### Sensor de color, giroscopio, táctil y ultrasónico

Como se comentó anteriormente, estas piezas no juegan ningún rol considerable en el movimiento de la plataforma robótica. Prácticamente, no importa el movimiento que la plataforma describa, estas piezas nunca van a moverse de su lugar de ubicación. Es por

esto que, para todos estos componentes electrónicos se debe utilizar una articulación fija. Como lo indica su nombre, una articulación fija en la descripción del formato SDF, es una articulación que no permite grados de libertad. Esto quiere decir que estas cuatro piezas no tendrán movimiento relativo con respecto al chasis, pues se encuentran adheridas a él.

### 3.4.4 Ensamblaje del mundo de simulación

Al querer diseñar un modelo computacional de simulación para el Lego Mindstorms EV3 no es solo suficiente con ensamblar la plataforma robótica como tal. Si no que, también es necesario describir todo un mundo de simulación en donde la plataforma robótica ensamblada exista, y pueda realizar sus comportamientos de movimiento y de medición. De nada sirve un sensor ultrasónico, si no se pueden colocar de manera dinámica objetos en el espacio de trabajo, o de nada funciona hacer girar las ruedas de la plataforma robótica si esta no tiene en que plano apoyarse y desplazarse. Es por esto que se debe también describir un espacio de trabajo -elemento tipo mundo- para el elemento tipo modelo ensamblado en el formato SDF. La construcción de este mundo se puede realizar tan compleja como se quiera. Se podría modelar las condiciones de iluminación de la clase de robótica junto a sus muebles y superficies, las condiciones ambientales (incluidas la ventilación, gradientes de temperatura y presión), el color del cielo y la distribución de nubes, inclusive se podría modelar un elemento tipo actor que represente al profesor evaluando el comportamiento de la plataforma robótica. Sin embargo, por supuesto que esto no es práctico. Además de generar una apariencia más fiel al entorno real, para el verdadero objetivo de estudiar las funcionalidades y comportamientos que interesan de la plataforma robótica, no hace más que sumar carga computacional. Es por esto que se decide ensamblar un mundo de simulación básico y multipropósito. De tal manera que, tenga los elementos esenciales necesarios para garantizar el correcto funcionamiento de la plataforma robótica pero, con la posibilidad de adaptar el mundo a las diferentes necesidades que se puedan presentar en el estudio y análisis de la misma. Es decir, se diseña un mundo de simulación el cual si el estudiante lo desea puede modificar cada aspecto del mismo, agregar un objeto de la clase, una pista de carreras, simular el comportamiento de la plataforma robótica en el aire libre a medio día o en las intimidades de su habitación; las herramientas para esto se desarrollan de tal manera en que la única limitación para el mundo de simulación sea la creatividad del estudiante. A continuación, en la Tabla. 3.7 se muestran algunos parámetros básicos que se consideraron en el ensamble del mundo de simulación simple. Cabe rescatar que, existen muchos otros elementos del tipo mundo en la descripción del formato SDF que se implementaron pero que no se muestran en la tabla. Esto debido a que no tienen mayor importancia o porque no hubo ninguna decisión detrás de ellos. Es decir, se especificaron con su valor *default* - siempre y cuando no afectara la exactitud de la simulación de la plataforma robótica.

| Parámetro | Valor |
|---|---|
| Iluminación | Se simula un sol con una luz tipo direccional que ilumina todo el espacio de trabajo. |
| Superficie del suelo | Un plano de 100m x 100m de color gris claro con el movimiento desactivado. |
| Gravedad | -9.8 m/s2 |
| Escena | Un ambiente y fondo de color gris claro y con la generación de sombras activado |
| Cámara | Una imagen ortogonal que apunta hacia el origen del mundo de simulación con un campo de visión adecuado, donde se inserta y aprecia la plataforma robótica. |

**Tabla 3.7:** Algunos parámetros del mundo de simulación diseñado. Fuente: Elaboración propia

## 3.5 Diseño de comportamientos básicos del robot

En la sección anterior se diseñó todo el entorno de simulación relacionado con la plataforma robótica y su mundo. La plataforma robótica ensamblada cuenta con propiedades físicas básicas, elementos mecánicos y electrónicos descritos. Los sensores realizan mediciones sobre alguna variable del mundo simulado y las relaciones de movimiento especificadas permiten el movimiento de la plataforma robótica. Además, se cuenta con una apariencia física muy semejante al robot real.

Todo esto Gazebo lo realiza con fieles simulaciones que se apegan al sentido físico real - gracias a sus motores gráficos y de cálculos -. Dentro de Gazebo, se puede -por ejemplo- especificar manualmente que la articulación de una rueda gire a una velocidad angular específica y Gazebo realizará entonces todos los cálculos necesarios para mostrar qué sucedería con la plataforma - a nivel de movimiento y mediciones de sensores - si esa rueda gira de tal forma y mostrar su animación en tiempo real. O bien, se puede preguntar por el dato actual de un sensor en específico y se pondrá en disposición dicha información. Sin embargo, esto está lejos de ser un entorno programático en donde se puedan diseñar comportamientos complejos. ¿Como haría un estudiante si desea ver cuánto tardan las ruedas de la plataforma robótica en girar 10 revoluciones a máxima velocidad? Podría especificar manualmente en la interfaz gráfica de Gazebo, un valor de velocidad angular para las articulaciones correspondientes y luego estar atento para que en el momento en que las ruedas completaron sus 10 revoluciones, especificar manualmente una velocidad angular de 0 y observar el tiempo de simulación. Esto es totalmente impráctico y definitivamente no es el objetivo del modelo computacional de simulación del Lego Mindstorms EV3 que se desea desarrollar.

Se necesita un entorno de programación el cual permita al estudiante diseñar comportamientos que desea estudiar y luego poder ver sus resultados en la simulación de Gazebo de manera práctica, para de esta manera cumplir con los beneficios que un modelo computacional como este puede aportar. Es aquí entonces, donde entra en juego ROS. ROS funciona como el cerebro de la simulación, al enviar datos a las articulaciones de la plataforma robótica y en recibir las mediciones de los sensores, de forma programática y a gusto del estudiante o usuario. Es decir, al habilitar las herramientas de ROS en este modelo computacional, se podrían diseñar los comportamientos básicos del sistema robótico Lego Mindstorms EV3, y posibilita al usuario tener el control sobre el flujo de la simulación mediante el manejo de información de los sensores (lectura) y actuadores (envío de órdenes) del *kit* en Gazebo. Entonces se pueden anotar tres funcionalidades de ROS en el modelo computacional - cuya implementación es el objetivo de esta sección.

1. Diseñar un entorno de espacio de trabajo compatible con ROS. Aquí se trata de acondicionar un paquete ROS para las necesidades del modelo de simulación.

2. Manipular la medición de los sensores. Se trata de colgar la medición de los sensores de Gazebo en ROS y adecuarlos para que sean fiel al comportamiento de la plataforma robótica real, además de ponerlos a disposición de uso al usuario de forma simple.

3. Manipulación de órdenes a los actuadores. Se trata de que el usuario pueda desde ROS accionar los actuadores de Gazebo (descritos por las articulaciones) de una forma semejante a la plataforma robótica real.

Para los últimos dos puntos, cabe realizar una aclaración. Para la lectura de los sensores y el control de los actuadores, si bien primero se debe realizar un trabajo para que estas funcionalidades se habiliten, también se debe procurar que la manera en que funcionen sean fieles al comportamiento de la plataforma robótica real. El Lego Mindstorms EV3 tiene por sí solo un ambiente de programación, en el cual se cuenta con todas las funciones necesarias para diseñar cualquier comportamiento. Cuenta con funciones para la lectura de sus sensores y diversas funciones para el control de sus actuadores. Entonces la idea es, modelar las funciones esenciales que pone a disposición el *software* original de programación del Lego Mindstorms EV3 para tenerlas disponibles en la simulación y así lograr un estilo de programación semejante cuando se esta diseñando un comportamiento. Por ejemplo, en el *software* original de programación del Lego Mindstorms EV3, se encuentran funciones como "Move tank" la cual mueve ambas ruedas de manera simultánea a una velocidad específica. O una otra función "Get sensor" que devuelve el valor de un sensor específico. Si en el modelo computacional, a la hora de diseñar un comportamiento, este tipo de funcionalidades son muy complejas de implementar entonces el modelo computacional falla en su objetivo de ser simple de manejar y fiel a su semejante real. Por lo que se debe recalcar que no es solo una cuestión de poder accesar la información de los sensores o actuadores en ROS, si no que es parte esencial que este proceso se realice de

forma similar al estilo de programación del real. A continuación, el desarrollo realizado en ROS.

### 3.5.1 Creación del entono de trabajo en ROS

Esta primera parte consiste en ensamblar un paquete de ROS apto para el modelo computacional. Un espacio de trabajo que interconecta ROS y Gazebo para poder usar en sincronía las herramientas que ofrecen ambos *software*. Lo más tangible de este proceso es un directorio de carpetas y archivos, donde cada parte juega un rol en el funcionamiento del modelo computacional. Prácticamente se describe un paquete de ROS acondicionado para que el estudiante pueda diseñar sus comportamientos en simulación. Un paquete de ROS, que deba interconectarse con herramientas de Gazebo, debe tener la estructura y componentes explicados a continuación, en un directorio con la estructura mostrada en la Fig.3.10. De primera instancia, se aprecian los archivos package.xml y CMakeList.txt esenciales en la descripción de un paquete de ROS, y cuya funcionalidad fue explicada en el capítulo 2

```
../lmev/src
  /lmev_gazebo
    package.xml
    CMakeList.txt
    /include
      plugin_sensor.h
      ...
    /launch
      lmev.launch
      ...
    /models
      /lmev
        model.sdf
        model.config
        /meshes
          Rueda.dae
          Chassis.dae
      ...
    /scripts
      comportamiento1.py
      ...
    /src
      plugin_sensor.cpp
      ...
    /worlds
      lmev.world
      ...
```

**Figura 3.10:** Estructura del directorio del paquete de ROS. Fuente: Elaboración propia.

#### models

La carpeta llamada “models” es la carpeta donde se van a contener todos los elementos tipo modelo descritos en el formato SDF en Gazebo y todos los archivos adicionales para la descripción de la misma. Por ejemplo, en esta carpeta se encuentra el modelo lmev, que corresponde a la descripción de la plataforma robótica Lego Mindstorms EV3 realizada en

la sección anterior y todas las piezas tridimensionales construidas (archivos .dae). Note que en esta carpeta se podría agregar cualquier otro modelo que el usuario desee incluir en la simulación y sus respectivos *meshes*.

#### worlds

La carpeta llamada "worlds" es la carpeta donde se van a contener todos los elementos tipo mundo descritos en el formato SDF en Gazebo. Por ejemplo, en esta carpeta se encuentra el mundo lmev_world, que corresponde a la descripción del mundo de simulación realizada en la sección anterior. Note que en esta carpeta se podría agregar cualquier otro mundo que el usuario desee simular. Aquí precisamente, es donde entra la utilidad de haber descrito la plataforma robótica como un elemento modelo. Pues ahora, si se quiere simular su comportamiento pero con una pista de carreras distinta o en el aula de trabajo, basta solo con cambiar el archivo .world utilizado. De esta manera, se puede realizar simulaciones con distintas condiciones de manera eficiente.

#### launch

La carpeta llamada "launch" es la carpeta donde se van a contener instrucciones para la ejecución de distintos nodos de ROS. Por ejemplo en esta carpeta se encuentra el archivo lmev.launch, que se encarga de ejecutar un nodo cuya funcionalidad es inicializar Gazebo y cargar el mundo de simulación especificado, un otro nodo cuya funcionalidad es cargar en el mundo de simulación el modelo lmev en una posición especificada. Note que en esta carpeta se podría agregar cualquier otro archivo "launch" que ejecute diversos nodos en diverso orden, según como el usuario desee en su simulación.

#### scripts

La carpeta llamada "scripts" es la carpeta donde se van a contener los algoritmos programáticos que describen un comportamiento de movimiento y medición de la plataforma robótica ensamblada. Por ejemplo en esta carpeta el estudiante puede agregar archivos .py o .cpp (provenientes de Python o C++, respectivamente) que dentro de sus líneas de código hayan instrucciones de lectura de sensores, una lógica programática y luego instrucciones de control de actuadores de la plataforma robótica en función al algoritmo. Es decir, cada archivo de estos, funciona como un nodo individual en ROS. Más adelante, en esta carpeta se agregará un nodo que describe un comportamiento en la simulación de la plataforma robótica de esta seguir la luz en un entorno creado, o también un nodo que describe un comportamiento en la simulación de la plataforma robótica de esta avanzar 10 revoluciones hacia adelante, girar 90 grados a la derecha, tomar la medición del sensor ultrasónico y luego bajar el brazo actuador, todo en un código programático escrito en Python. Prácticamente los archivos de esta carpeta se pueden comparar con los archivos que se generan al diseñar un comportamiento del Lego Mindstorms EV3 en su *software* de

programación oficial. En el siguiente capítulo se retomarán los archivos de esta carpeta, su importancia, funcionalidad y sus demás detalles.

#### src e include

En las carpetas llamadas "src" e "include" se encuentran los *plugins* desarrollados para cumplir con los últimos dos objetivos de ROS en el modelo de computacional diseñado. En estas dos carpetas se encuentran por ejemplo, el *plugin* encargado de obtener la medición del sensor giroscopio del modelo de Gazebo, procesarlo según sea necesario y ponerlo a disposición del usuario para poder ser consultado desde un algoritmo en la carpeta *scripts*. O también, el *plugin* encargado de constantemente escuchar por una instrucción -en algún algoritmo en la carpeta *script*- de movimiento de algún actuador y en el momento en que llegue, modificar los valores del estado de las articulaciones en la simulación en Gazebo. Todas las consideraciones y procedimientos explicados en los siguientes dos objetivos, se encuentran codificados en estas carpetas. Estos archivos se cargan de manera dinámica a la descripción SDF de la plataforma robótica y por lo tanto se ejecutan cuando se inicializa la simulación en Gazebo.

### 3.5.2 Diseño de *plugins* para los componentes electrónicos

Como se mencionó anteriormente, el objetivo de estos *plugins* es primero colgar la información de los sensores de Gazebo en el entorno ROS. Segundo modificar dicha información de manera tal en que la medición se apegue lo más posible al comportamiento real. Y tercero, poner a disposición del usuario dicho resultado para que pueda ser accesado de forma programática desde un nodo de la carpeta *script* que pretenda la lectura de sensores de forma dinámica en la simulación. Para el primer paso, se aprovecha la condición de que ROS es un entorno con herramientas reciclables y adaptables. Prácticamente, ROS ya dispone de nodos que logran la lectura de los sensores de Gazebo mediante *plugins*. Sin embargo, esto no sucede con todos los sensores con los que se trabaja y con los que sucede, resulta que la información que se cuelga en ROS no es precisamente la que se busca. Es por esto que si bien se encuentran avances ya desarrollados para el primer paso - y solo para algunos sensores-, aún es necesario realizar el segundo y tercer paso para conseguir el modelo computacional que se desea. Pero, estos avances ya desarrollados se utilizan como una base para implementar las adiciones algorítmicas y programáticas necesarias para los demás pasos. Prácticamente, se aprovechan las herramientas ya desarrolladas pero se modifican según las necesidades que se desea suplir. A continuación se describe todas las modificaciones, diseños y decisiones realizadas para el tratamiento de los *plugins*. Siempre recordando que todos estos *plugins* funcionan como pequeños nodos individuales que ejecutan el procesamiento para cada medición del sensor generada por Gazebo.

### Sensor de color

En la sección pasada, se decidió que la mejor manera de simular el sensor de color en modo ambiente fuese mediante el análisis de una imagen capturada por un sensor tipo cámara. Una vez que la es imagen capturada por el sensor cámara en Gazebo, mediante un *plugin* reciclado, se obtiene en el entorno ROS un arreglo de matrices con información sobre los valores de intensidad de brillo (ente 0 y 255), correspondientes a sus componentes R,G,B. Se decide realizar una aproximación a la luz ambiente en la que se encuentra la plataforma robótica, promediando la intensidad de brillo de la imagen capturada. Para esto, se promedia el valor de la matriz R, G, y B de tal forma en que se obtengan un punto R,G,B promedio de la imagen. Es decir, todos los valores de intensidad de brillo de los píxeles de la matriz R se promedian y esto resulta en un único valor de intensidad de brillo para la componente R - y así con G y B. Luego se computa 3.2. Dicha relación sigue la norma ITU BT.709 que se utiliza en diversos monitores actuales para indicar el nivel de luminancia que el usuario experimenta, a partir de la intensidad de los píxeles. La ecuación considera que el ojo humano es más sensible en cuestión de intensidad de brillo para el color verde y menor para el color azul. Con esto se busca que el estudio de la imagen y resultado del sensor simulado se acerque más a una medición de luz ambiente.

$$Luminancia = 0.2126R + 0.7152G + 0.0722B \tag{3.2}$$

Cabe aclarar un detalle importante, el sensor de color en modo ambiente real interactúa con el fenómeno físico de la luz. Si bien su medición se da en una escala de 0-100, primero hubo una medición del flujo luminoso incidente sobre la superficie del sensor (iluminancia) cuya unidad de medición es lux. En la simulación, debido a que una imagen capturada posee solo información sobre intensidad de brillo en los píxeles, este fenómeno físico no se considera, por lo que no se puede esperar que haya una relación exacta entre las mediciones. Este tema se retomará en el siguiente capítulo con mayor detalle. Sin embargo, lo importante aquí es que con la ecuación 3.2 no se pretende transformar de un fenómeno físico a otro, si no que solo se busca mejorar la percepción del sensor simulado para el usuario y así conseguir un comportamiento cercano al sensor real.

Luego de esto, se sigue 3.3 para conseguir un único valor promedio de intensidad de brillo de la imagen y para escalar dicho valor de 0 a 100. Ahora se cuenta con un valor numérico que representa de alguna manera las condiciones de iluminación en las que se encuentra el sensor simulado. Sin embargo, cabe considerar un par de detalles adicionales.

$$Med_{sc} = 100\frac{Luminancia}{255Res_{camara}} \tag{3.3}$$

El primer detalle - que quedó pendiente de explicación en la sección pasada - es la adición de ruido en la medición. No es necesario explicar la importancia de consideración del ruido en sensores de una plataforma robótica para saber lo fundamental que es equipar los sensores simulados con un ruido fiel al de su semejante real. La adición del ruido

se lleva acabo en los *plugins* de ROS debido a que se desea que dicho parámetro sea altamente configurable, pues así tiene mayor alcance educativa. Se decide, junto al GII, que el ruido de los sensores simulados se modele mediante una distribución gausseana. Es decir, a los sensores se les agregará un ruido gausseano, con media y desviación estándar acorde a sus comportamientos.

Para el sensor de color en modo ambiente, no se logra encontrar información confiable sobre el ruido que experimenta la medición de iluminación. Esto debido a que al pasar por la tabla de conversión algorítmica, se vuelve difícil realizar un estudio objetivo. Por lo que se decide, establecer un valor de ruido de manera experimental con base en las variaciones visibles de la medición. Sin embargo, en el siguiente capítulo se analizará que en realidad, para este sensor, esta decisión no compromete negativamente la fiabilidad de su comportamiento con respecto al real.

El segundo detalle, es considerar el formato del valor numérico disponible en la medición del sensor real. Dicha medición tiene una resolución de 0 decimales (es un número entero), por lo que se debe también redondear el resultado obtenido a un número entero. Estos detalles son esenciales para realmente obtener un comportamiento de simulación semejante al real. De nada sirve que en la simulación el usuario tenga a disposición una medición con 100 decimales de precisión si cuando pasa a trabajar con la plataforma real, esta no la tiene.

Una vez que se consigue procesar la medición del sensor proveniente de Gazebo, ya se ha cumplido tanto la tarea en ROS de obtener la información proveniente de Gazebo y modificarla de tal manera en que sea fiel a su comportamiento real. Por lo que, lo único que hace falta es poner a disposición del usuario dicha medición resultante para que pueda ser accesada desde cualquier otro nodo. Para esto, se sigue el proceso habitual de comunicación entre nodos de ROS. El *plugin* desarrollado al funcionar como un nodo de procesamiento, se puede declarar como un nodo publicador. Entonces, se crea un tema de tipo número entero -bajo el concepto del entorno ROS, explicado en el capítulo 2- y el *plugin* del sensor de color publica un mensaje tipo número entero cuyo valor es resultado del procesamiento aquí explicado. Por lo que, finalmente el usuario puede preguntar desde cualquier otro nodo el mensaje que se trasmite en el tema declarado y la respuesta será la medición del sensor de color en modo ambiente, actuando dinámicamente en la simulación de Gazebo.

### Sensor giroscopio

En la sección pasada, se decidió que la mejor manera de simular el sensor giroscopio fuese mediante el uso de un elemento sensor tipo IMU en la descripción del formato SDF. Existe un *plugin* disponible para poner en disposición la información de dicho sensor de Gazebo en el entorno ROS. Sin embargo, como se mencionó anteriormente, dicho sensor genera más información de la que el sensor real. Esto significa que la información del sensor simulado que llega a ROS debe ser modificada y adaptada al comportamiento del sensor

real.

El *plugin* de comunicación entre Gazebo y ROS para este sensor pone en disposición un mensaje -bajo el concepto del entorno ROS, explicado en el capítulo 2- tipo IMU. Este mensaje tiene 7 partes: Un encabezado tipo *string*, un mensaje anidado tipo cuaternión, un *float* para la covarianza de la orientación, un vector de velocidad angular, un *float* para la covarianza de la velocidad angular, un vector de aceleración lineal y un *float* para la covarianza de la aceleración lineal. Note la complejidad del mensaje recibido y el exceso de información que posee en comparación con el sensor real. Sucede que, si se decide que este mensaje sea el resultado final de la medición del sensor simulado, el usuario estará obligado a realizar más procedimiento para llegar al dato que le interesa. En el Lego Mindstorms EV3 el usuario basta con una línea de código para obtener la medición del sensor giroscopio, un simple número listo para ser evaluado en el resto del algoritmo. Si este mensaje tipo IMU no se modifica, el usuario cuando este diseñando su comportamiento algorítmico en la simulación y pregunte por el valor del sensor giroscopio en un momento determinado, será abrumado por excesiva información y tendrá que entrar en un proceso de discriminación para encontrar el segmento del mensaje que realmente busca. Y no termina ahí, por que cuando encuentre el segmento de mensaje que le interesa, se llevará la sorpresa de que no está en el formato que esperaba. En lugar de recibir un simple número, recibirá una matriz de cuaterniones en radianes, lo cual lo obligará a seguir procesando el dato. Esto es totalmente impráctico. Y es por esto que se deben realizar los siguiente esfuerzos.

Se decide utilizar únicamente el segmento del mensaje IMU correspondiente a los cuaterniones. El procesamiento consiste - como primer paso- en transformar, mediante las herramientas programáticas que ofrece C++, los cuaterniones en ángulos de Euler en grados. Una vez que se tiene los ángulos de Euler, se discrima cúal es el ángulo que interesa (*yaw*, *pitch* o *roll*) que esté acorde al eje z del sensor - tal y como el sensor real. Ahora, existen dos detalles por considerar al estudiar con detalle el comportamiento del sensor real.

El primero consiste en que el sensor real considera la dirección de giro. Si gira igual a las manecillas del reloj, entonces la medición de ángulo del sensor es positiva, en caso contrario es negativa. Bien se sabe que a partir de los cuaterniones, se obtienen mediciones que van de 0 a 180 y de -180 a 0. Esto implica un problema, pues si por ejemplo el *kit* robótico real gira 190 grados en la dirección de las manecillas del reloj, el sensor resulta una medición de 190 mientras que la medición del sensor simulado daría -170. Esto implicaría una confusión para el estudiante y no modela fielmente el comportamiento del real. Es por esto que se diseña un algoritmo que evalúa constantemente los incrementos de la medición para determinar la dirección de giro con la que viaja la plataforma robótica y así realizar el ajuste necesario.

El segundo consiste en que el sensor real realiza una medición del ángulo de inclinación con respecto al eje z acumulativa. Es decir, si el *kit* robótico gira 386 grados en dirección contraria a las manecillas del reloj, el usuario dispondría de un valor de -386 grados. En

la simulación sucedería que - debido a que ahora se considera la dirección - el usuario dispondría de un valor igual a -6, pues debido a la naturaleza no acumulativa de los cuaterniones. Es por esto que se diseña un algoritmo que evalúa constantemente la dirección de giro y los incrementos de la medición para determinar si debe empezar a acumular el valor del ángulo de giro. Finalmente se obtiene, un simple número fiel al comportamiento de su semejante real. Resta ahora solo, realizar el redondeo para que la medición tenga una resolución de 0 decimales (número entero) y la adición de ruido gausseano. Para el sensor giroscopio, según [12], se conoce que tiene una precisión de más/menos 3 grados en su medición para giros menores a 90 grados. Por lo que se procede a especificar de forma práctica una media y desviación estándar acorde a dicha información.

Por último, después de conseguir obtener la información del sensor simulado de Gazebo en ROS y de haberla procesado de tal manera en que sea fiel al formato y comportamiento de su semejante real, resta solo poner a disposición del usuario dicha medición resultante para que pueda ser accesada desde cualquier otro nodo. Para esto, se sigue el proceso habitual de comunicación entre nodos de ROS. El *plugin* desarrollado al funcionar como un nodo de procesamiento, se puede declarar como un nodo publicador. Entonces, se crea un tema de tipo número entero -bajo el concepto del entorno ROS, explicado en el capítulo 2- y el *plugin* del sensor giroscopio publica un mensaje tipo número entero cuyo valor es resultado del procesamiento aquí explicado. De esta manera, el usuario puede preguntar desde cualquier otro nodo el mensaje que se trasmite en el tema declarado y la respuesta será la medición del sensor giroscopio, actuando dinámicamente en la simulación de Gazebo.

### Sensor táctil

En la sección pasada, se decidió que la mejor manera de simular el sensor táctil fuese mediante el uso de un elemento sensor tipo Contacto en la descripción del formato SDF. Existe un *plugin* disponible para poner en disposición la información de dicho sensor de Gazebo en el entorno ROS. Sin embargo, pasa la misma situación que el sensor giroscopio, la información del sensor simulado son muy distintos a los que se desea que el usuario accese. Esto significa que la información del sensor simulado que llega a ROS debe ser modificada y adaptada al comportamiento del sensor real.

El *plugin* de comunicación entre Gazebo y ROS para este sensor pone en disposición un mensaje -bajo el concepto del entorno ROS, explicado en el capítulo 2- tipo Contacto. Este mensaje tiene 8 partes: un encabezado tipo *string*, un *string* que indica las dos geometrías de colisión en contacto, una matriz de mensajes anidado tipo llave que posee información sobre los vectores fuerza y momento para cada punto, un mensaje anidado tipo llave que indica la fuerza y momento total en el sensor, un vector con información sobre los puntos en contacto, un vector con información sobre los ejes normales del contacto y una matriz de *float* que indica la profundidad del contacto. Note nuevamente la complejidad del mensaje recibido y su poco sentido práctico comparado con la simple medición del sensor real.

Para el procesamiento de este sensor, lo que se realiza es discriminar la superficie o plano frontal del elemento enlace del sensor táctil (la zona donde tiene el botón rojo) dentro de toda la información del mensaje. Luego, se pregunta si existe alguna fuerza actuando en dicha superficie y si la hay y es mayor a 0.5 N, entonces una variable booleana establece su valor en 1, de lo contrario 0. Tal y como sucede en el sensor real, la medición que dispone el usuario es un 0 cuando el interruptor no está presionado y un 1 cuando lo está. En la simulación se establece que se 'presiona el interruptor' cuando existe una fuerza normal al plano de la geometría de colisión frontal especificado mayor a 0.5 N.

Adicional a esto, se decide no implementar ruido gausseano a esta medición, pues el sensor real es bantante robusto a este (por su mecanismo mecánico) ni tampoco es necesario considerar resolución de decimales de la medición. Entonces, finalmente -al igual que todos los sensores- se declara el nodo del *plugin* como un nodo publicador en un tema tipo número entero. Donde luego del procesamiento del mensaje tipo IMU, se publicará la variable resultado del procesamiento aquí explicado. De esta manera, el usuario puede preguntar desde cualquier otro nodo el mensaje que se trasmite en el tema declarado y la respuesta será la medición del sensor táctil - un simple 0 o 1, actuando dinámicamente en la simulación de Gazebo y fiel a su semejante real.

### Sensor ultrasónico

En la sección pasada, se decidió que la mejor manera de simular el sensor ultrasónico fuese mediante el uso de un elemento sensor tipo Rayo en la descripción del formato SDF. Existe un *plugin* disponible para poner en disposición la información de dicho sensor de Gazebo en el entorno ROS. Y para el caso de este sensor, dicho *plugin* funciona de manera más conveniente. Pero, esto no excluye la necesidad de modificar y adaptar la información para se apegue cada vez más a su comportamiento real.

El *plugin* de comunicación entre Gazebo y ROS para este sensor pone en disposición un mensaje -bajo el concepto del entorno ROS, explicado en el capítulo 2- tipo Rango. Este mensaje tiene prácticamente 4 partes: un encabezado tipo *string*, un *float* que indica el ángulo de medición del cono al que se encuentra funcionando el sensor, dos floats que indican el rango mínimo y máximo de medición utilizado y finalmente, la medición de distancia del sensor al objeto más cercano en una variable de tipo *float* en metros.

Para este caso, el segmento de información que se debe utilizar es muy directo, se utiliza únicamente el *float* correspondiente a la distancia medida y se procede a realizar pequeñas modificaciones. Entre ellas se pueden anotar: La conversión de metros a centímetros, el redondeo necesario para que la medición tenga la misma resolución que el real - en este caso, según [12], el real tiene una resolución de 0.1 cm, y también la adición de ruido gausseano. [12] indica que el sensor ultrasónico del Lego Mindstorms EV3 tiene una precisión de 1 cm, por lo que se procede a establecer la media y la desviación estándar correspondiente. También, el sensor real tiene un comportamiento típico en el que cuando el objeto se encuentra fuera del rango de medición devuelve un valor igual a 255. Por lo

que se procede a realizar esta condicional, si la medición proveniente de Gazebo es menor a 3 o mayor a 250, entonces el nuevo valor de la medición será 255.

Luego de conseguir modelar los comportamientos del sensor real al sensor simulado, el resultado de la medición esta listo para ponerse a disposición del usuario. Al igual a como los *plugins* pasados, se realiza el mismo procedimiento. Se declara el nodo como publicador, se declara un tema - esta vez de tipo *float*- y se publica cada medición generada mediante un mensaje tipo *float*. De esta manera, el usuario puede preguntar desde cualquier otro nodo el mensaje que se trasmite en el tema declarado y la respuesta será la medición del sensor ultrasónico - un simple valor flotante, actuando dinámicamente en la simulación de Gazebo y fiel en comportamiento a su semejante real.

### Sensores *encoders*

Hasta ahora no se había hablado de los sensores *encoder* que posee cada motor del Lego Mindstorms EV3. Estos sensores son esenciales de modelar para obtener información sobre el movimiento de la plataforma robótica, comportamientos programáticos como avanzar una cantidad específica de revoluciones o mover el brazo actuador a un ángulo deseado se consiguen estudiando la medición de dicho sensores. No se habían considerado en la sección pasada debido a que en realidad estos sensores no poseen una descripción SDF en Gazebo como los otros, si no que todo su ensamble y funcionamiento se lleva a cabo mediante *plugins* de ROS.

De forma práctica, se simulan estos tres *encoders* (uno para cada motor) accediendo a la información sobre las articulaciones de Gazebo. Es decir, se crea un *plugin* encargado de leer y monitorear constantemente la pose de la trama de la articulación. Para las ruedas poder girar, sus articulaciones primero tienen que girar entonces si se accede a la información sobre la orientación de la articulación, se esta accediendo indirectamente a la orientación de la rueda. Y de esta forma se puede determinar la cantidad de grados que ha girado, simulando el funcionamiento del sensor *encoder* del *kit* robótico real.

A partir de aquí, se sigue el mismo procedimiento y manipulación de medición realizado para el sensor giroscopio, pues al acceder a la información de la articulación, esta describe su orientación mediante cuaterniones. Por lo que en resumen, se transforma de cuaterniones a ángulos de Euler en grados, luego se aplican los algoritmos para considerar dirección de giro y ángulo acumulativo, luego el redondeo para garantizar una misma resolución en la medida igual al real y por último la adición del ruido gausseano. Según [12], los sensores *encoder* de los motores tienen precisión de un grado, para garantizar movimiento en sincronía, por lo que se procede a especificar la debida media y desviación estándar del ruido gausseano.

Además de esto, y al igual que los demás sensores, se declara el nodo del *plugin* como un publicador, un tema de tipo número entero, y la medición de los sensores se publica mediante un mensaje tipo número entero. Así, el usuario tiene disponible la medición de los *encoders* de forma fácil, práctica y fiable.

### 3.5.3 Diseño de *plugins* para los componentes mecánicos

Actualmente el modelo computacional es en diversos ejes, funcional. El usuario posee la capacidad de simular todo un entorno físico de la plataforma robótica, crear mundos, leer mediciones de los sensores disponible en ROS, entre otras funcionalidades. Sin embargo, todavía no es capaz de movilizar la plataforma robótica a través del entorno de simulación de manera práctica, desde un *script* programático. El objetivo los *plugins* de los componentes mecánicos es proporcionar al usuario con herramientas para que pueda enviar comandos a los actuadores de la plataforma robótica en simulación desde un nodo de procesamiento. Para esto, se debe establecer la comunicación entre ROS y Gazebo -pero esta vez en dirección contraria- para enviar datos de ROS hacia Gazebo con las órdenes que se desea ejecutar.

A lo largo del capítulo el lector podrá haber notado que en realidad nunca se ha modelado un motor de la plataforma robótica como tal. Pero en realidad - como se mencionó anteriormente- si se ha hecho, de manera indirecta. Los actuadores de la plataforma robótica son los motores, estos motores son los que imprimen la potencia en las ruedas o en el brazo actuador para generar su movimiento. En el modelo de simulación ensamblado este movimiento esta descrito por las articulaciones de cada enlace. Por lo que si se logra modificar el estado de movimiento de las articulaciones, se logra imprimir movimiento sobre los actuadores y de esta forma, las articulaciones estarían cumpliendo el rol de los motores. Esto resulta conveniente, pues Gazebo tiene diversas herramientas para alterar el movimiento de las articulaciones. A continuación el análisis y la solución implementada para estos *plugins*.

#### Motores grandes y mediano

Los *plugins* para los dos motores grandes y para el motor mediano son en esencia el mismo, sin embargo trabajan en nodos distintos por una cuestión de mayor confianza de funcionamiento. Sin embargo, se expone aquí las consideraciones generales que se tomaron para cada uno considerando que su implementación es la misma.

Para cada uno de los tres *plugins*, se declara un nodo subscriptor. Se crea un tema de tipo *float* en el que el usuario publicará un mensaje tipo *float* con el valor de la velocidad angular (rad/s) que desea mover la articulación. Posteriormente dicho valor de velocidad angular se acondicionará para que siga el comportamiento del motor real y el resultado será enviado a Gazebo mediante uno de los métodos que pone a disposición este *software* para establecer la velocidad de articulaciones.

Gazebo tiene tres métodos para establecer la velocidad de las articulaciones de sus modelos. El primer método es establecer la velocidad en la articulación de manera instantánea. Este método se aprovecha de que Gazebo puede manipular cualquier parámetro en cualquier instante de la simulación. Dicho, provoca que la articulación - y por lo tanto el enlace relacionado - empiece a (en este caso) girar de manera instantánea a la veloci-

dad especificada, sin ninguna consideración de aceleración y sin importar las propiedades físicas de los enlaces relacionados con la articulación. El segundo método es el control de velocidad mediante un control tipo PID. Este método considera todos los parámetros que existen en un algoritmo de control, sobre impulso, tiempo de estabilización, error de estado estacionario y por supuesto, las constantes P, I y D.

El tercer y último método, convenientemente, es específico para cuando se desea establecer la velocidad de una articulación tal y como lo haría un motor. Esto quiere decir que considera un fenómeno de aceleración y las limitaciones físicas que tendría un motor real. Por ejemplo, se considera la potencia del motor que se desea simular y a partir de ahí la articulación se mueve según estas limitaciones. Para un motor es imposible establecer una velocidad de manera instantánea ni tampoco le es posible alcanzar velocidades o torques fuera de su capacidad. Cabe rescatar, que una vez ejecutado este método, Gazebo es el encargado de calcular todas las interacciones físicas siguientes en el movimiento, dada la inercia, fricciones, centros de masa y todos las demás propiedades que puedan interferir en el movimiento de la plataforma robótica, Gazebo las considera y resulta en un comportamiento lógico.

Si bien cada método tiene sus ventajas y desventajas, y cada uno vendría bien para una necesidad en especifico, es claro que el método que se debe utilizar en este caso es eventualmente el tercero, diseñado por Gazebo para mover las articulaciones tal y como lo haría un motor. Para el uso de este método es solo necesario especificar la potencia del motor por simular, en cuestión de velocidad máxima y torque máximo. Entonces, según [12], se especifican los siguientes valores de la Tabla.3.8.

| Pieza | Torque máximo [Nm] | Velocidad máxima [rad/s] |
|---|---|---|
| Motor grande - Ruedas | 0.45 | 18.33 |
| Motor mediano - Brazo actuador | 0.15 | 27.22 |

**Tabla 3.8:** Valor de torque y velocidad máxima especificada para el tercer método. Fuente: Elaboración propia

### Movimiento sincronizado

Una de las funcionalidades más importantes de la plataforma robótica es el movimiento de sus dos ruedas en sincronía para garantizar un movimiento adecuado para el traslado eficiente. En el entorno programático del *software* original del Lego Mindstorms, existe una función encargada de simultáneamente mover ambas llantas con una velocidad específica de la plataforma robótica con una dirección en particular. Esta función es muy utilizada, por lo que se decide que es fundamental modelar este comportamiento.

Para lograr esto, lo que se diseña es un nodo tanto subscriptor como publicador. La idea es crear un tema tipo Twist y subscribir el nodo al mismo para escuchar constantemente por los mensajes tipo Twist que envía el usuario. Un mensaje tipo Twist es un mensaje

compuesto por dos vectores, el primer vector corresponde a una especificación de velocidad lineal (m/s) en los tres ejes y el segundo vector es una especificación de la velocidad angular (rad/s) en los tres ejes. Mediante este mensaje el usuario describirá el movimiento que desea que la plataforma como tal realice. Note que, la plataforma robótica simulada en realidad solo les es posible moverse con una velocidad lineal sobre un sólo eje, y que solo puede girar a una velocidad angular especifica en torno a un eje también. Claro, pues si fuese de lo contrario, la plataforma sería por ejemplo capaz de moverse lateralmente, comportamiento que no tiene sentido por la orientación de las ruedas.

Este mensaje tipo Twist creado por el usuario, debe de alguna forma de transformase en una combinación de valores de velocidad angular para las articulaciones de la plataforma, pues es la única manera de mover los actuadores. Para determinar esta relación, se realiza el siguiente análisis dinámico para implementarlo de forma programática en el *plugin*. En la Fig.3.11 se muestra un diagrama libre de las dos ruedas de la plataforma robótica. W1 es la velocidad angular especificada por el usuario en el mensaje Twist, R1 es la distancia entre ruedas, W2 es la velocidad angular de la articulación, R2 es el radio de la rueda, V0 (no se encuentra en la figura) se trata como la velocidad lineal especificada por el usuario en el mensaje Twist, V1 es la velocidad lineal resultante por el movimiento circular ocasionado por W1 y V2 es la velocidad lineal del extremo de la rueda resultante del movimiento circular ocasionado por W1 y W2.

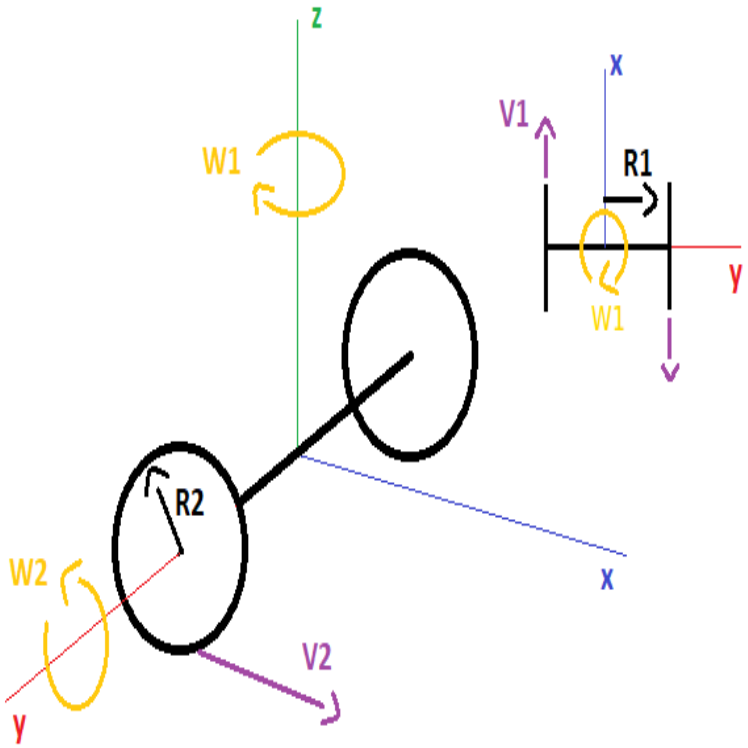



**Figura 3.11:** Diagrama de cuerpo libre de la plataforma robótica. Fuente: Elaboración propia.

Primero, se asume un cuerpo rígido que conecta ambas ruedas -que ese puede interpretar

como el chasis- dicho cuerpo, según [1], se encuentra en un movimiento tipo plano general, por lo que V2 se describe como:

$$V_{2d} = V_{traslacion} + V_{rotacion} \tag{3.4}$$

La velocidad de traslación, corresponde a la velocidad lineal especificada por el usuario en el mensaje Twist y la velocidad de rotación corresponde al movimiento circular ocasionado por W1. Entonces, debido a que:

$$V = WxR \tag{3.5}$$

$$V_{2d} = V_0 + W_1 R_1 \tag{3.6}$$

Sin embargo, se conoce la descripción de V2, de tal forma que sustituyendo:

$$V_{2d} = W_{2d} R_{2r} \tag{3.7}$$

$$W_{2d} R_{2d} = V_0 + W_1 R_1 \tag{3.8}$$

Finalmente:

$$W_{2d} = \frac{V_0 + W_1 R_1}{R_{2d}} \tag{3.9}$$

Note que, 3.6, es solo válida para una rueda - en este caso la derecha. Esto por que la rueda izquierda, debido al movimiento circular descrito por W1, experimenta dicha velocidad de rotación en sentido contrario, tal que:

$$V_{2i} = V_0 - W_1 R_1 \tag{3.10}$$

Entonces,

$$W_{2i} = \frac{V_0 - W_1 R_1}{R_{2i}} \tag{3.11}$$

De esta forma se mapea del mensaje tipo Twist especificado por el usuario a la única variable en capacidad de manipular, la velocidad angular de las articulaciones. Aquí cabe hacer otra consideración, hay que recordar que W2 tiene un límite dado por la potencia del motor. Dicha circunstancia se analizará en el siguiente capítulo. También cabe rescatar, que en la plataforma robótica real el usuario no especifica valores de velocidad lineal ni angular para el movimiento de los motores. Lo único que especifica es un valor entre 0 y 100 y el motor ajusta su velocidad acorde. Este comportamiento se pensó en modelar,

sin embargo - junto al GII- se decidió no poner a disposición dicho comportamiento. Ya que, de esta forma el estudiante esta obligado a analizar la relación entre los datos que ingresa en la simulación y los datos que ingresa en el real, aumenta el impacto educativo. Pero, el análisis que indica dicha relación de todas maneras sí se realiza aquí, por motivos de validación y documentación. Sin embargo, se realizará en el siguiente capítulo.

Una vez obtenida la relación entre el mensaje Twist y la velocidad angular de cada articulación, cada vez que el nodo subscriptor escuche un mensaje, este se procesará y resultará en valores tipo *float*. Luego, se declara el nodo como un publicador y se publican los valores floats en los temas creados para el control individual de los motores, explicados en la subsección anterior.

Además de esta funcionalidad, también se desarrolla un nodo de procesamiento encargado de mover los motores grandes y el motor mediano cierta cantidad de ángulos definidos.Ya que, la plataforma real también cuenta con dicha función. Sin embargo, esto se realiza más como para ejemplificar las herramientas de uso y su funcionalidad que de habilitar la función como *plugin* por su importancia.

## 3.6 Consideraciones generales de la programación

El lector se podrá imaginar que todas las consideraciones y especificaciones tratadas en realidad tienen un trasfondo programático. Mediante el uso de los *software* en cuestión y herramientas de programación se diseñan múltiples algoritmos que eventualmente son los encargados de habilitar todas las funcionalidades que se han explicado. Esto es claro, pues el desarrollo de un modelo computacional de simulación efectivamente supone un trabajo en esta área de conocimiento. Por cada decisión y procedimiento explicado, se puede esperar extensos diseños programáticos. Es claro que no se pretende en esta sección explicar cada detalle de los mismos pero se comentan algunas consideraciones generales para que este trabajo no pase por desapercibido.

- El trabajo programático es realizado en el lenguaje C++, XML y Python. El lenguaje C++ trata directamente debido a que las herramientas a disposición relativas a ROS y Gazebo se encuentran mayormente escritas en este lenguaje, por lo que su reuso y aplicación se simplifica si los demás desarrollos también se encuentran en dicho lenguaje. Las descripciones en Gazebo, con lo que respecta a los modelos, mundos y demás elementos sólo se pueden diseñar mediante el lenguaje XML, en el que a su vez se utilizan las herramientas de la descripción SDF. Y por último, el lenguaje Python se utiliza únicamente en los comportamientos adicionales diseñados para la plataforma robótica. Aquellos de interés de estudio y modificación por parte de los estudiantes.

- Todos los códigos de programación desarrollados se encuentran debidamente comentados y son amigables con el usuario. Esto debido a que eventualmente, es de

interés que al pasar el tiempo el modelo computacional vaya evolucionando acorde a las necesidades del momento. Por lo que siempre se encuentran a disposición las explicaciones de los algoritmos con el fin de agilizar este proceso.

- Los diseños algorítmicos se enfocan en un funcionamiento eficiente y versátil.

- A nivel de funcionalidad, no solo se encuentran los códigos necesarios para habilitar todos los componentes de la simulación si no que también se desarrollan elementos programáticos para facilitar la experiencia del usuario. A lo que respecta sobre los pasos necesarios para inicializar la simulación o para utilizar sus diversas funcionalidades o para que el modelo computacional sea portable entre diferentes ordenadores.

## 3.7 Elaboración de la documentación

Todo el trabajo desarrollado debe quedar debidamente documentado, para aprovechamiento del diseñador, estudiantes de la asignatura de robótica e investigadores y desarrolladores que interactúen con el proyecto. Es por esto, que se ha elaborado una documentación -disponible en los Apéndices- que consiste en una guía de usuario que cuenta con:

1. Un manual de instalación de las herramientas *software* necesarias para la utilización y aprovechamiento del modelo computacional desarrollado.

2. Un manual de uso básico de los diferentes elementos del modelo computacional desarrollado que provea el conocimiento necesario para el manejo del mismo en breve tiempo.

3. Ejemplos sencillos sobre los comportamientos del sistema robótico en donde se detallen los pormenores de la lectura de sensores y envío de órdenes a los actuadores de la configuración del *kit*.

4. Explicación de los códigos de programación más importantes para la comprensión de los ejemplos y uso de la simulación, los cuales se encuentren debidamente comentados.

Cabe rescatar que dicha guía de usuario debe ser orientada a los estudiantes de la asignatura de robótica del grado en ingeniería informática de la Universidade Da Coruña, pues estos son los que mayormente utilizaran el modelo computacional. Por lo que se decide realizar la guía lo más concisa y práctica posible, para que el estudiante pueda adaptarse a la simulación en breve tiempo.

## 3.8 Estudio económico

Si bien el presente proyecto no es de índole industrial o comercializable, se detalla un pequeño estudio económico sobre la viabilidad y costo de la solución llevada a cabo. En la Tabla.3.9 se presenta el presupuesto necesario para el desarrollo del proyecto. Se ha decidido que todos los precios aquí mostrados fuesen obtenidos de manera web, sin considerar el pago por envío ni impuestos.

La viabilidad de esta solución radica en que a la Universidade Da Coruña le cuesta disponer de un *kit* robótico a cada grupo de estudiantes de la asignatura de robótica del grado de ingeniería informática -considerando que en promedio hay treinta estudiantes y que las prácticas de laboratorio se realizan en parejas - una inversión total de $ 5849.25. Y esto sin considerar la reparación o sustitución de los elementos de la plataforma robótica por mal uso o desgaste. Y ahora bien, el Grupo Integrado de Ingeniería al desarrollar este proyecto no debe realizar una inversión igual al costo total mostrado en la Tabla.3.9, esto debido a que el GII ya cuenta con los recursos ahí listados. La única nueva inversión consiste únicamente en el estudiante de mecatrónica y sus servicios básicos. Entonces, se contrasta una situación en donde para solucionar el problema planteado se puede invertir $ 5849.25 en la compra de nuevos *kit* robóticos ó invertir $ 2900 (correspondiente al costo del estudiante y sus servicios básicos) en el presente proyecto. Con esto, se evidencia la factibilidad de la solución desarrollada.

| Nombre | Cantidad | Observaciones | Precio total ($) |
|---|---|---|---|
| *kit* Robótico Lego Mindstorms EV3 | 1 | ID: 5003400 | 389.95 |
| Laptop Acer Aspire E5-575G-75MD | 1 | 7th Gen Intel Core i7, GeForce 940MX, 15.6" Full HD, 8GB DDR4, 256GB SSD, Ubuntu 16.04 64-bit | 699.99 |
| *software* Lego Digital Designer | 1 | Open-source, descargable en la web. Versión 4.3 | 0.00 |
| *software* Blender | 1 | Open-source, descargable en la web. Versión 2.78c. | 0.00 |
| *software* Gazebo | 1 | Open-source, descargable en la web. Versión 7. | 0.00 |
| *software* ROS | 1 | Open-source, descargable en la web. Versión Kinetic Kame. | 0.00 |
| *software* LibreOffice | 1 | Descargable en la web. Versión 5.4.3. | 0.00 |
| Otros | - | Cables, accesorios, elementos de almacenamiento digital de datos y repuestos | 100.00 |
| Estudiante mecatrónico | 1 | Remuneración total para 5 meses | 2750.00 |
| Técnico asesor | 1 | Remuneración total para 500 horas | 1250.00 |
| Servicios básicos | 2 | Agua, luz e internet | 300.00 |
| | | **Costo total** | **5489.94** |

**Tabla 3.9:** Presupuesto necesario para la ejecución del proyecto. Fuente: Elaboración propia.

# Capítulo 4

# Resultados y análisis

## 4.1 Introducción

A lo largo del desarrollo del modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3, se han implementado diversas herramientas y tomado decisiones con base en cuatro grandes ejes de diseño. El primer eje es el de eventualmente, desarrollar herramientas de carácter esencial y de aprovechamiento del usuario. El segundo eje consiste en que el modelo computacional desarrollado se apegue al comportamiento real, con lo que respecta a su sentido físico. El tercer eje trata sobre proporcionar un entorno de simulación práctico, sencillo y fiel al comportamiento de la plataforma real, ante los ojos del usuario. Y el cuarto eje, consiste en encontrar un equilibrio entre la exactitud y funcionalidades, y la capacidad computacional requerida para esto.

Para ejemplificar, considérese el brazo actuador de la plataforma robótica. El primer eje, garantiza que -efectivamente- exista en el entorno de simulación un brazo actuador, capaz de moverse o de medir una variable. El segundo eje, garantiza que - este brazo actuador- realice sus funcionalidades fiel a su semejante real. Ya sea moviéndose como lo dictan las leyes de la física o realizando mediciones de la misma variable que su contraparte real. El tercer eje, garantiza que estas funcionalidades, el usuario pueda utilizarlas de una manera semejante a como las utiliza en la plataforma real. Como por ejemplo que, controle su movimiento mediante herramientas parecidas a como lo realiza en el real. Y por último, el cuarto eje garantiza que toda esta situación sea posible sin comprometer el rendimiento eficiente - en términos de capacidad de procesamiento computacional - de la simulación.

Estos ejes, guiaron en general el desarrollo del modelo computacional de simulación para el sistema robótico, impactando ya sea de forma negativa o positiva la fiabilidad de la simulación. En este capítulo el lector encontrará los resultados obtenidos de cada sección explicada en el capítulo anterior. Además, encontrará un análisis objetivo de las consecuencias finales de la implantación de los cuatro ejes en cada uno de estos desarrollos. Y por último, se comentará un poco sobre funcionalidad en general del modelo computacional resultante.

## 4.2 Modelo tridimensional de la plataforma robótica

Prácticamente, se puede decir que el objetivo de esta sección era conseguir una plataforma robótica computarizada con una apariencia igual al *kit* robótico real. En la Fig.4.1, se muestra el resultado de todos los procesos de modelado tridimensional desarrollados. El parecido con su semejante real, mostrado en la Fig.3.2, evidencia que el criterio para la escogencia de *software* y la utilización de sus debidas herramientas fueron las adecuadas. El proceso de ensamblaje de la plataforma en LDD y su tratamiento en Blender para la construcción de sub-ensambles, ubicación, reensamble y modificación de escalas, resultó en una plataforma robótica fiel a su apariencia real. En la Fig.4.2, se evidencian los detalles de los *meshes* utilizados en el entorno de simulación.

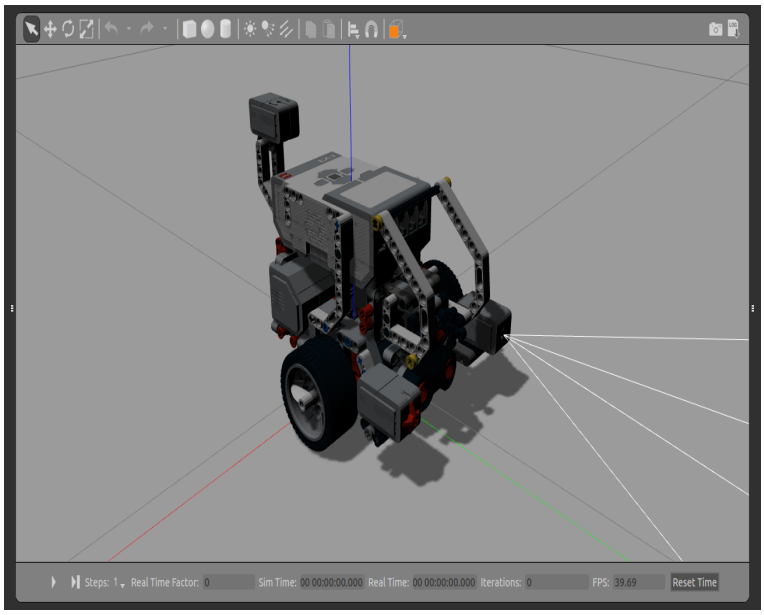



**Figura 4.1:** Modelo tridimensional en simulación de la plataforma robótica Lego Mindstorms EV3. Fuente: Elaboración propia.

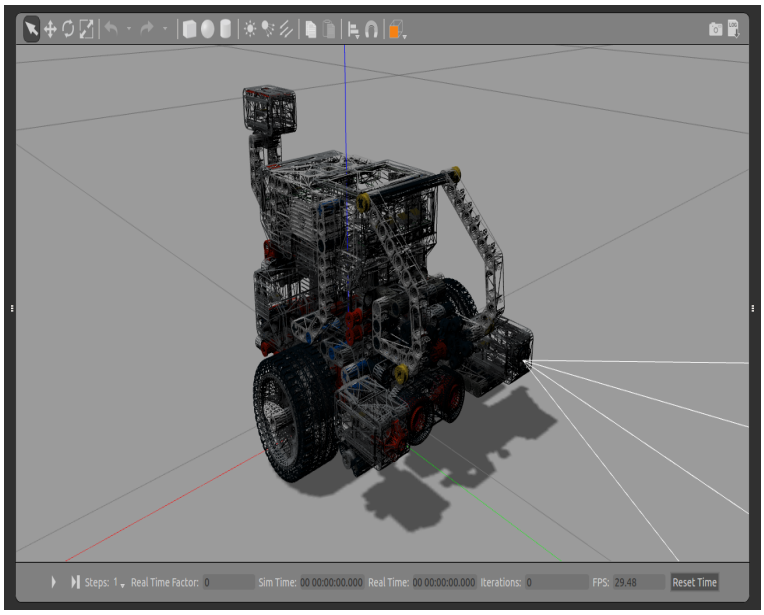



**Figura 4.2:** *Wireframe* de la plataforma robótica en simulación. Fuente: Elaboración propia.

## 4.3 Descripción de la plataforma robótica en el entorno de simulación

El objetivo de esta sección era crear un modelo en la descripción SDF de Gazebo de la plataforma robótica Lego Mindstorms EV3. De tal manera en que el modelo diseñado tuviese todas las herramientas necesarias para que los motores gráficos y físicos de Gazebo pudiesen calcular sus interacciones con el entorno. Al especificar propiedades físicas, relaciones de movimiento y elementos sensor, se consigue transformar de unas piezas tridimensionales inanimadas a una plataforma robótica que funciona como conjunto. El resultado de estas especificaciones, se evidencia en las siguientes figuras. Donde en primer plano, el lector podrá observar el resultado de la descripción SDF de los enlaces de la plataforma robótica, ya sea para elementos mecánicos o para elementos electrónicos.

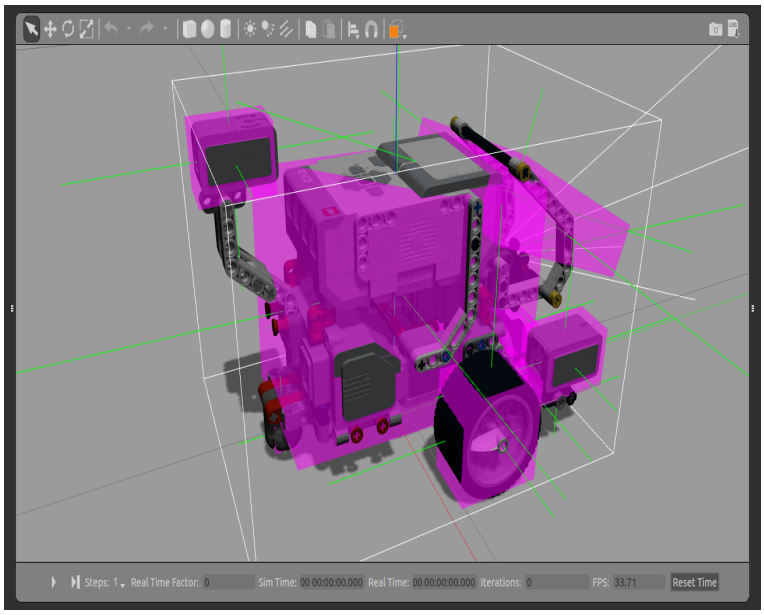



**Figura 4.3:** Inercia de cada una de las partes de la plataforma robótica en simulación. Fuente: Elaboración propia.

En la Fig.4.3, se muestra la inercia de cada enlace de la plataforma robótica resultante. Según [8], en general cada cubo morado debe coincidir aproximadamente con el tamaño del enlace al que esta asociado. Note que esto no significa que la inercia del enlace fue calculada y modelada como un cubo, es solo una representación gráfica que Gazebo dispone para indicar que el enlace no carece de propiedades dinámicas y por lo tanto, puede realizar una simulación fiable de su comportamiento cuando interactúe con fuerzas o momentos. En realidad, la forma de validar estos momentos de inercia es observando el movimiento de la plataforma a partir de diversas excitaciones y que esta no se comporte de manera extraña - en comparación con el real. Sin embargo, esto se validará de mejor manera - a vista del lector- en los vídeos de más adelante. Luego de correr la simulación exhaustiva cantidad de veces, se puede garantizar que las propiedades inerciales especificadas para cada enlace son adecuadas para modelar un comportamiento apegado a la realidad.

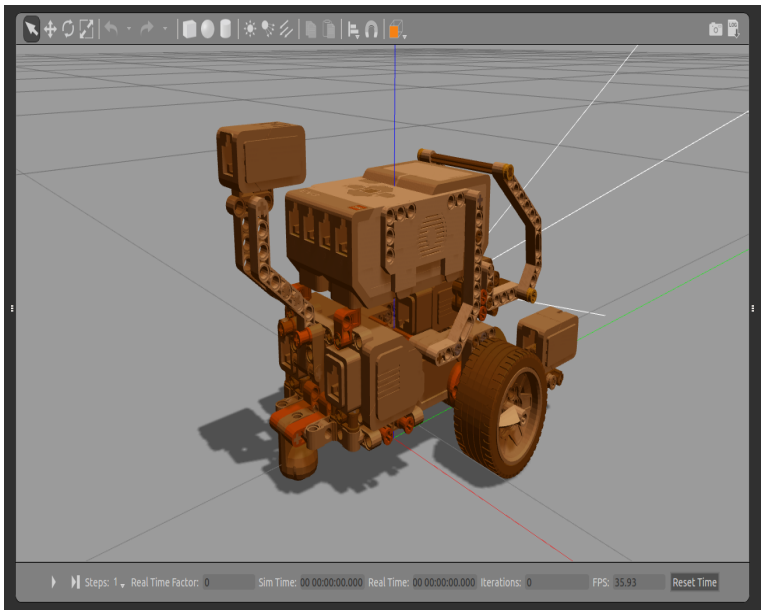


**Figura 4.4:** Geometrías de colisión del modelo computacional. Fuente: Elaboración propia.

En la Fig.4.4, se muestra la geometría de colisión implementada para cada enlace. Note que, es precisamente la pieza tridimensional creada. Esto significa que, todas las interacciones de choque y en general de movimiento, Gazebo las calculará lo más apegadas a la realidad. Esto supone una carga computacional mayor - sin embargo, esto se analizará más adelante. Por el momento, es seguro que al escoger estas geometrías de colisión se gana exactitud y precisión. Igual sucede con las geometrías visuales, solo que en este caso, se gana una apariencia física agradable, apegada a su semejante real.

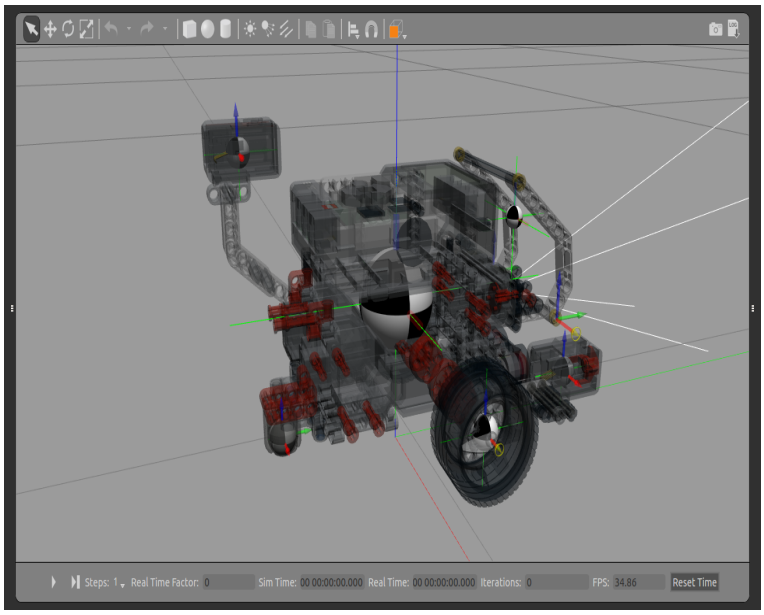


**Figura 4.5:** Centros de masa y articulaciones de la plataforma robótica en simulación. Fuente: Elaboración propia.

En la Fig.4.5, se muestra de manera gráfica y en la simulación de Gazebo, los centros de masa (representados por la esfera de color blanca y negra) de cada enlace, que responden al punto especificado en su creación en Blender y descripción en el formato SDF. Además, note los pequeños ejes de referencia presentes en cada enlace, estos ejes en realidad representan las articulaciones descritas y son las que permiten el movimiento de la plataforma. Si se presta atención, la articulación de la rueda y del brazo actuador tiene un anillo amarillo en su eje de revolución, el cual con cuerda con su semejante real si se estudiara como verdadera plataforma robótica con tramas y eslabones. Estas descripciones permiten finalmente el movimiento fiel de la plataforma, describiendo y habilitando un movimiento con sentido físico.

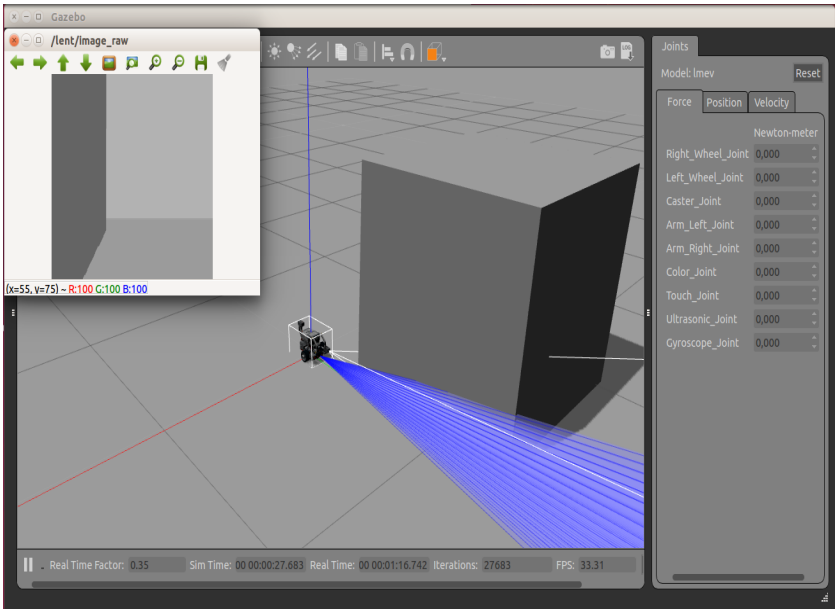



**Figura 4.6:** Funcionamiento de sensores y actuadores en el modelo computacional. Fuente: Elaboración propia.

En la Fig.4.6, se muestra lo que se podría considerar como el resultado final de la etapa de Gazebo. Lo primero por notar, es el funcionamiento de los sensores. Note por ejemplo la imagen capturada por el sensor tipo cámara en la esquina superior izquierda, o el cono de medición descrito por el elemento sensor tipo rayo. Si bien en la imagen no se aprecia el funcionamiento del sensor giroscopio y táctil, todos los sensores se encuentran funcionando y midiendo una variable del entorno programático, acorde a su representación real. Recordando por supuesto, que dicha información recolectada luego será procesada en los nodos de ROS. Lo segundo por notar, es la pestaña del lado derecho de la imagen. En dicha pestaña se encuentran las articulaciones descritas en el formato SDF para la plataforma robótica en simulación. En este panel, se puede modificar de manera manual la velocidad o fuerza aplicada en cada articulación, permitiendo entonces el movimiento de la plataforma robótica. A partir del valor especificado Gazebo procederá a, primero considerar el tipo de articulación y los grados de libertad que esta tiene y segundo, a aplicar dicha modificación y así calcular todas las demás implicaciones físicas que supone

en la plataforma, al igual que realizar la simulación visual de dicha situación. Lo tercero y último por notar, es el mundo de simulación en donde la plataforma robótica se encuentra. Un entorno libre, con descripciones físicas básicas en el formato SDF, propenso a ser modificado a gusto del usuario. Por ejemplo, se puede ver el cubo gris oscuro que se incluyo en el mundo para este interactuar con los demás modelos. Ahora la plataforma robótica tiene vida dentro del entorno de Gazebo y esta acondicionada para su simulación física en el mismo.

## 4.4 Comportamientos básicos del modelo computacional de simulación

Al ser esta la última sección en la que se agregan funcionalidades o se modifican detalles de la simulación, se encuentra la mayor parte de resultados y por ende de análisis. El uso de ROS en el modelo computacional de simulación tiene tres objetivos explicados en el capítulo anterior. El cumplimiento de estos objetivos garantiza un modelo computacional de simulación que cumple con los primeros tres de los cuatros ejes de diseño. Se procede a estudiar a profundidad los resultados obtenidos para cada objetivo ROS, comparando de forma objetiva los comportamientos simulados contra los comportamientos reales de la simulación.

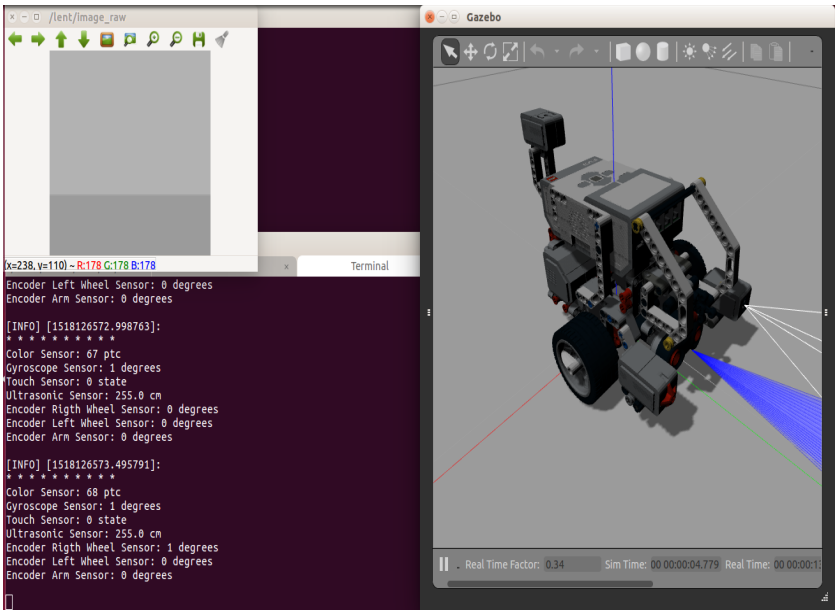



**Figura 4.7:** Funcionamiento de *plugins* en los sensores del modelo computacional. Fuente: Elaboración propia.

### 4.4.1 Componentes electrónicos

En la Fig.4.7, se muestran los resultados obtenidos tras la implementación de ROS en el modelo computacional. En dicha figura, se puede notar mayormente las mediciones resultantes de los sensores al haber sido procesadas por sus debidos *plugins*. De primera instancia se evidencia que el formato de las mediciones es exactamente igual a su contra parte real. Y no solo su formato sino también los comportamientos característicos de cada sensor. La terminal con las mediciones de los sensores, en realidad ejecuta un algoritmo escrito en Python el cual pregunta por los mensajes que se trasmiten por los temas creados por cada *plugin*, guarda su valor y los imprime en pantalla. Todo mediante una línea de código en dicho *script*. La medición de los sensores existe con sentido físico y se encuentra disponible al usuario de forma práctica y sencilla. Con esto se cumple - para los componentes electrónicos - el primer y tercer eje de diseño. Sin embargo, ¿cómo se comparan estas mediciones con sus semejantes reales?. A continuación, una análisis para cada sensor desarrollado - para propósitos del segundo eje de diseño.

#### Sensor de color

El sensor de color experimenta una situación en particular. En realidad las mediciones del sensor en simulación y las mediciones del sensor real son distintas. Esto se debe a que como se explicó anteriormente, el sensor simulado en realidad funciona bajo otro fenómeno físico distinto a la luz. Sin embargo, esto no significa que el sensor de color no cumpla con su sentido práctico en simulación.

El objetivo final de que los sensores se comporten tal y como lo hacen en la realidad es para que - después de que se diseñó un comportamiento y se validó en simulación- el usuario pueda trasladar ese algoritmo a la plataforma robótica real y utilizarlo ahí sin tener que realizar mayor cambio. Sin embargo, aún si esto no sucede, la simulación funciona para validar el flujo de programación del comportamiento diseñado. Por ejemplo, suponga que se desea diseñar un comportamiento en que la plataforma lleve bloques de un lado a otro y que la medición de los sensores entre la simulación y la realidad son totalmente distintos. El usuario, podría adaptarse al comportamiento de los sensores en simulación, diseñar su comportamiento y a lo largo de las pruebas en la simulación darse cuenta que le hace falta alguna condición para detener el movimiento de la plataforma si ya no hay bloques que trasladar. Entonces para este tipo de errores, la simulación -aún si los sensores no funcionan igual al real- serviría, un proceso de validación del flujo lógico del comportamiento. Luego, el usuario cuando pasa a trabajar con la plataforma real podría reajustar los valores de los sensores y actuadores y ya tendría su comportamiento funcionando en la realidad.

Si bien, esta es una funcionalidad tangible de la simulación, se evita lo más que se pueda. Debido a que es mucho más práctico y enriquecedor en cuestión de tiempo y esfuerzo que el usuario no deba hacer este reajuste. Sin embargo, cabe rescatar que un pequeño reajuste nunca va a poder evitarse del todo, pues es claro que aunque los sensores funcionen lo

más parecidos que se pueda, un entorno real es mucho más complejo que un entorno simulado. Por lo que siempre se tendrá que hacer algún cambio. Entonces, se entra en un proceso de, igual sabiendo que la simulación tiene funcionalidad sin necesidad de que los sensores se comporten exactamente igual, tratar de que lo hagan - aun sabiendo que siempre pequeños ajustes serán necesarios.

El sensor de color experimenta precisamente esta situación. Su medición en la simulación es distinta a su medición en la realidad. Pero, se obtiene un comportamiento en el sensor de color en modo luz ambiente que se comporta de igual manera al sensor real. El valor de la medición del sensor real, es directamente proporcional a la cantidad de luz. Pero en Gazebo, tal fenómeno físico no existe, pues la luz eventualmente en todo el entorno programático se modela con intensidad de píxeles. Pero, la medición del sensor de color en modo ambiente en la simulación es proporcional a la intensidad de píxeles en la imagen capturada. Entonces, se consigue un sensor que si bien no da exactamente el mismo valor en su medición que su semejante real, el entorno se modifica de tal forma en que su comportamiento se asemeje al real. En lugar de agregar luz, se agrega intensidad de píxeles en el ambiente y por lo tanto la medición del sensor cambia acorde.

Esta situación de hecho, es más conveniente para el usuario a que si el sensor de color en modo ambiente funcionara tal y como funciona en la realidad. Para esto, suponga la siguiente situación. Primero, - como ya se dijo- que el sensor de color funciona igual a su contra parte real. Segundo, que el usuario trabaja con la plataforma real en su habitación. Entonces - como el objetivo de que el sensor real y el sensor simulado tengan la misma medición es para que el usuario después de probar su algoritmo en la simulación, el mismo algoritmo sin ajustes funcione en su habitación- se obligaría al usuario a -en la simulación- describir todas las condiciones de iluminación que experimenta su habitación. Esto es complejo, pues dichas condiciones no son parametrizables en la simulación. El usuario, para no tener que cambiar nada en su algoritmo, entonces tendría que modelar toda su habitación en el entorno de Gazebo.

Esto supone un trabajo ajeno al diseño del algoritmo de comportamiento de la plataforma robótica que se desea simular muy pesado y poco productivo. Y además, aunque se realice, siempre existirán otras variables en el entorno real que obligaran al usuario a realizar ajustes. Es por esto que si bien el sensor de color en modo ambiente no resulta mediciones iguales a su contra parte real - las cuales no se ajuntan por que igual ni se podrían establecer condiciones iguales para la comparación - cumple con el sentido práctico de que su comportamiento varía con respecto a una variable que simula la variable real.

Entonces, el sensor de color en modo ambiente, cumple con efectivamente ser una herramienta a disposición capaz de medir una variable programática en el entorno de Gazebo con su funcionamiento como cámara, no cumple con que su sentido físico sea igual al de su contra parte real - pero sigue siendo funcional y práctico -, y cumple con seguir un comportamiento semejante al real en cuestión de ser una medición de 0 a 100 y en que le es sencillo al usuario acceder a su información.

### Sensor giroscopio y *encoders*

Se decidió realizar una sección para ambos sensores debido a que como se explicó el capítulo pasado, la implementación de su funcionamiento y el procesamiento de su información, es la misma. Con estos sensores, sucede todo lo contrario. La medición del sensor de simulación - además de seguir el mismo formato y comportamiento (dirección de giro y ángulo acumulativo - se asemeja a la medición del sensor real. Al igual que el sensor de color, el usuario puede acceder a la medición del sensor con una simple línea de código en un nodo de procesamiento. En la Tabla.4.1, se presenta la comparación de las mediciones para diferentes ángulos de orientación de la plataforma robótica para el sensor giroscopio. Y para los *encoders*, se puede utilizar el estudio a los motores realizado en la próxima sección, donde se programa que las ruedas giren 10 revoluciones o 3600 grados. Para cada dato mostrado, se realizaron cinco mediciones.

| Orientación [°] | Medición real [°] | Medición simulada [°] | Porcentaje de error [ptc] |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 30 | 29 | 30 | 3.45 |
| 45 | 46 | 45 | 2.17 |
| 60 | 62 | 59 | 4.84 |
| 90 | 89 | 89 | 0 |
| 180 | 182 | 180 | 1.10 |
| 270 | 271 | 270 | 0.37 |
| 360 | 363 | 361 | 0.55 |

**Tabla 4.1:** Mediciones del comportamiento del sensor giroscopio real y simulado. Fuente: Elaboración propia

Lo primero por notar es que los porcentajes de error son menores al 5 porciento. Sin embargo, cabe rescatar que este sensor giroscopio es el más ruidoso de todos, es por esto que para algunas mediciones el porcentaje de error pareciese ser elevado con respecto a las demás. Inclusive, aunque el sensor de simulación funcione idealmente, nunca se va a conseguir un porcentaje de error igual a cero. Esto se debe a que las mediciones siempre se ven afectadas en diferente medida por el ruido. Sin embargo, a pesar de esto, se obtienen porcentajes de error muy bajos, lo cual valida el segundo eje de diseño. Lo segundo por notar es que los comportamientos de los sensores se modelan tal sus semejantes reales. En la Fig.4.7 se aprecia la resolución de cada medición corresponde a la real y en le Fig.4.8 se aprecia el comportamiento de dirección de giro y acumulación de ángulo para los *encoders* - recordando que el sensor giroscopio funciona bajo el mismo algoritmo.

El sensor giroscopio y los sensores *encoders* entonces, cumplen con ser efectivamente un sensor que mide la orientación con respecto al eje z de la plataforma robótica en simulación o de las ruedas, también - según la Tabla.4.1- el sensor giroscopio cumple con que su medición sea físicamente fiable y semejante a su contra parte real, y por último, cumplen en que su comportamiento y uso a ojos del usuario sea semejante al real. El

sensor giroscopio y los sensores *encoders* cumplen con los primeros tres ejes de diseño.

### Sensor táctil

El sensor táctil, funciona a la perfección. Primero, se dispone de un sensor el cual indica información sobre el contacto que experimenta. Segundo su valor cambia entre 0 y 1 dependiendo de si actúa una fuerza en la superficie frontal del enlace - donde se encuentra el botón-, tal y como lo hace su semejante real. Y tercero, su uso y comportamiento es sencillo - pues se accesa a su valor con una línea de código - y fiable - pues modela el comportamiento de la medición real. En la Fig.4.8 se muestra el sensor táctil funcionando - pero si se mira con atención, también se logra apreciar otras funcionalidades del modelo computacional. El sensor giroscopio cumple con los primeros tres ejes de diseño.

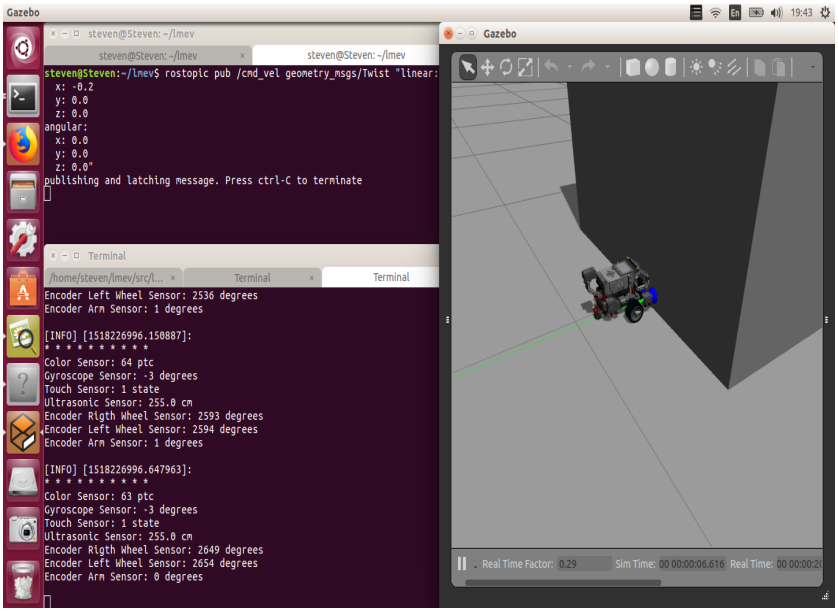



**Figura 4.8:** Funcionamiento del sensor táctil, funcionamiento de los sensores *encoders*, funcionamiento del sensor ultrasónico y envío de mensaje tipo Twist. Fuente: Elaboración propia.

### Sensor ultrasónico

El sensor ultrasónico tiene diversos enfoques de estudio. El primero y más directo es que efectivamente es un sensor capaz de medir la distancia al objeto más cercano con parámetros configurables. El segundo es que su comportamiento a nivel de formato precisión y rango de medición sea igual a su semejante. Esto se evidencia en las pasadas figuras, donde se puede notar que la medición disponible en el entorno ROS del sensor ultrasónico tiene una resolución de 1 decimal y donde cuando un objeto se encuentra fuera de su rango de medición, el valor disponible es igual a 255. El tercer estudio en con

respecto a la fiabilidad de sus mediciones, por lo que se adjunta la Tabla.4.2. Para cada dato mostrado, se realizaron cinco mediciones.

| Distancia [cm] | Medición real [cm] | Medición simulada [cm] | Porcentaje de error [ptc] |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 10 | 10.2 | 10.1 | 0.98 |
| 20 | 20.5 | 20.3 | 0.98 |
| 30 | 30.3 | 29.9 | 1.32 |
| 40 | 40.8 | 40.1 | 1.72 |
| 50 | 50.9 | 50.1 | 1.57 |
| 60 | 60.7 | 60.2 | 0.82 |
| 70 | 71 | 69.9 | 1.55 |
| 80 | 80.6 | 79.8 | 0.99 |
| 90 | 90.8 | 90.3 | 0.55 |
| 100 | 100.8 | 100.1 | 0.69 |
| 120 | 120.1 | 120.2 | 0.1 |
| 160 | 161.1 | 160.4 | 0.43 |
| 200 | 201.4 | 200.2 | 0.60 |
| 240 | 242 | 240.3 | 0.70 |

**Tabla 4.2:** Mediciones del comportamiento del sensor ultrasónico real y simulado para un objeto a diferentes distancias perpendiculares. Fuente: Elaboración propia.

Lo primero por aclarar es que el porcentaje de error esta dado con respecto a la medición real y simulada. Pues eventualmente lo que interesa es que la simulación se comporte como la plataforma real, aunque esta no sea la mejor. Por lo demás, se puede decir que los resultados de la medición son exactos, al no sobrepasar el 2 porciento de error. Esto indica que el sensor en simulación modela de forma exitosa el sensor real para mediciones de distancia a objetos perpendiculares. Sin embargo, cabe todavía estudiar el comportamiento del sensor para objetos no perpendiculares. Por lo que se adjunta la Tabla.4.3.

| Ángulo teórico [°] | Medición real [cm] | Medición simulada [cm] | Porcentaje de error [ptc] |
|---|---|---|---|
| 0 | 40.8 | 40.3 | 1.23 |
| 30 | 36.6 | 36.9 | 0.77 |
| 45 | 32.1 | 34.8 | 8.41 |
| 60 | 76.0 | 30.9 | 40.62 |
| 90 | 43.6 | 41.2 | 5.44 |

**Tabla 4.3:** Mediciones del comportamiento del sensor ultrasónico real y simulado para un objeto a 40 cm a diferentes orientaciones. Fuente: Elaboración propia

Los porcentajes de error elevados para objetos cada vez más inclinados con respecto al sensor, indica que el sensor en simulación no modela bien este comportamiento del real.

Esto ya se sabía pues, Gazebo no realiza la simulación del mismo fenómeno físico de ondas ultrasónicas. Sin embargo, se podría aun considerar que para ángulos menores a 30 grados, se obtiene una exactitud aceptable para el sensor en simulación.

Aún que este comportamiento no se modele a la perfección,el sensor ultrasónico cumple en su mayoría con sus objetivos prácticos. Y en realidad, que no se modele este comportamiento no es un problema grave, pues una vez que se conoce la limitación, se puede trabajar para aplacarla. Es decir, el estudiante esta frente una situación donde puede decidir ignorar la limitación y diseñar un algoritmo que funcione bien en la simulación. Solo para darse cuenta que cuando llegue a probar en el real, no va a funcionar igual. O puede, ya que sabe qué funciona bien y qué funciona mal, diseñar un algoritmo robusto para esta limitación desde un inicio en la simulación para de esta forma, dicho algoritmo funcione de mejor manera en la plataforma real.

El sensor ultrasónico cumple entonces, con ser una herramienta disponible para medir distancia a un objeto, cumple con su semejanza en la medición física con respecto al real para objetos perpendiculares al sensor y a menor medida para objetos orientados a menos de 30 grados, y además cumple con ser del mismo formato que la medición real con mismo comportamiento y facilidad de uso.

### 4.4.2 Componentes mecánicos

Para los componentes mecánicos, lo único relevante de estudiar es el comportamiento de los actuadores. Bien se sabe ya, que los actuadores de la plataforma robótica son dos motores grandes y un motor mediano. La implementación programática de estos tres motores es exactamente la misma, por lo que se realiza el estudio para los tres. Primero, efectivamente los tres actuadores existen en la simulación y son capaces de provocar movimiento en la plataforma robótica simulada. Para el segundo eje de diseño, se realiza el siguiente análisis.

Para validar el diseño de los motores, se decide probar su comportamiento en cuestión de velocidad, tiempo y distancia. Para la plataforma robótica, se crea una algoritmo en su *software* de programación original que ocasiona que ambas ruedas giren 10 revoluciones hacia adelante a una velocidad de 50 y luego se detengan. Se realizan las medidas del tiempo que tarda en completar dicho movimiento y la distancia que avanza. Se muestran en la Tabla.4.4.

Luego, se programó el mismo algoritmo con las herramientas desarrolladas en el modelo computacional, con fin de realizar la comparación entre tiempos y distancia. Sin embargo, se debe realizar primero un análisis. En el capítulo anterior, se decidió que la especificaciones de los motores no fuera exacta a la real. Ya que en lugar de especificar un valor entre 0 y 100 para su control, se especifica un valor de velocidad angular - si se utiliza el *plugin* de movimiento individual - o un mensaje tipo Twist - si se utiliza el *plugin* de movimiento sincronizado. Entonces es necesario encontrar la relación entre estos tipos de control.

| Número de rotaciones | Velocidad [ptc] | Tiempo (s) | Distancia [m] |
|---|---|---|---|
| 10 | 50 | 7.155 | 1.735 |
| 10 | 50 | 7.150 | 1.735 |
| 10 | 50 | 7.142 | 1.740 |
| 10 | 50 | 7.223 | 1.735 |
| 10 | 50 | 7.149 | 1.735 |
| | Promedio | 7.164 | 1.736 |
| | Desviación estándar | 0.033 | 0.0022 |

**Tabla 4.4:** Mediciones del comportamiento de los motores grandes en el avance de la plataforma robótica real. Fuente: Elaboración propia

Para el movimiento individual de motores, el análisis es muy sencillo. Se asume una relación lineal entre la escala utilizada por la plataforma real y la velocidad máxima a la que puede llegar cada motor. Entonces, siendo Vs la velocidad en rad/s especificada en la simulación y Vr la velocidad porcentual especificada en la plataforma robótica real, para los motores grandes:

$$\frac{V_s}{V_r} = \frac{18.33}{100} \tag{4.1}$$

Y para el motor mediano:

$$\frac{V_s}{V_r} = \frac{27.22}{100} \tag{4.2}$$

Para cuando se quiere utilizar el mensaje tipo Twist, el análisis es ligeramente más complicado. A partir de la Ec.4.3, se conoce que la velocidad angular de la rueda es una combinación de la velocidad lineal en m/s y la velocidad angular en rad/s especificadas en el mensaje Twist.

$$W_{2d} = \frac{V_0 + W_1 R_1}{R_{2d}} \tag{4.3}$$

Sin embargo, W2 tiene un límite dado por la potencia del motor. Entonces para cuando no hay velocidad angular.

$$W_{2d} = \frac{V_0}{R_{2d}} \tag{4.4}$$

Despejando para V0:

$$V_0 = W_{2d} R_{2d} \tag{4.5}$$

Sustituyendo W2 por su máximo (18.33 rad/s) y R2 por el radio de la rueda (0.028 m), entonces:

$$V_0 = 18.33 * 0.028 \tag{4.6}$$

$$V_0 = 0.513 m/s \tag{4.7}$$

Dada que esa es la velocidad lineal máxima que se puede especificar en el mensaje Twist y dado que implícitamente provoca que las ruedas giren a máxima velocidad, entonces nuevamente se asume una relación lineal entre la simulación y la plataforma robótica lineal. Por lo que entonces:

$$\frac{V_s}{V_r} = \frac{0.513}{100} \tag{4.8}$$

Ahora, considerando la 4.3, para cuando no hay velocidad lineal:

$$W_{2d} = \frac{W_1 R_1}{R_{2d}} \tag{4.9}$$

Despejando para W1:

$$W_1 = \frac{W_{2d} R_{2d}}{R_1} \tag{4.10}$$

Sustituyendo W2 por su máximo (18.33 rad/s), R2 por el radio de la rueda (0.028 m) y R1 por la distancia entre ruedas entre dos:

$$W_1 = \frac{18.33 * 0.028}{0.06653} \tag{4.11}$$

$$W_1 = 7.71 rad/s \tag{4.12}$$

Por lo que:

$$\frac{W_1}{W_r} = \frac{7.71}{100} \tag{4.13}$$

El usuario entonces tendrá que jugar con el equilibrio de V0 y W1 si desea mover la plataforma robótica simultáneamente hacia adelante o atrás y hacia los lados. Sin embargo, la mayoría de las veces en las prácticas de clase, se realizan los movimientos por separado. Es decir, primero avanza hacia adelante (utilizando solo velocidad lineal), se detiene y luego gira hacia un lado (utilizando solo velocidad angular).

Ahora, volviendo al estudio de los motores. Debido a que el *plugin* del movimiento sincronizado utiliza el *plugin* de movimiento individual de los motores -mediante los procesos de comunicación de ROS- si se prueba el primero, se prueba el segundo. Entonces en el estudio de los motores de la plataforma real se utilizó una velocidad en los motores de 50. Según 4.23, despejando:

$$V_s = V_r \frac{0.513}{100} \tag{4.14}$$

$$V_s = 50 \frac{0.513}{100} \tag{4.15}$$

$$V_s = 0.256 m/s \tag{4.16}$$

Una velocidad en la realidad de 50, corresponde en teoría y en la simulación a una velocidad de 0.257 m/s. Finalmente, se procede a especificar dicho valor en la simulación y realizar las mismas mediciones del estudio para su comparación. En la Tabla.4.5, se muestra las mediciones de tiempo y distancia obtenidas al mover la plataforma robótica a una velocidad lineal de 0.256 m/s por 10 revoluciones.

| Número de rotaciones | Velocidad [ptc] | Tiempo (s) | Distancia [m] |
|---|---|---|---|
| 10 | 0.256 | 7.038 | 1.726 |
| 10 | 0.256 | 6.995 | 1.732 |
| 10 | 0.256 | 6.976 | 1.731 |
| 10 | 0.256 | 7.056 | 1.729 |
| 10 | 0.256 | 6.997 | 1.732 |
| | Promedio | 7.012 | 1.730 |
| | Desviación estándar | 0.033 | 0.0025 |

**Tabla 4.5:** Mediciones del comportamiento de los motores grandes en el avance de la plataforma robótica en simulación. Fuente: Elaboración propia

Estos datos de tiempo y distancia para el estudio definido, en realidad se pueden determinar de manera teórica. Para la distancia, al conocer el radio de la rueda y la circunferencia de un círculo, se tiene que:

$$Dist_i = 2\pi r Rev \tag{4.17}$$

$$Dist_i = 2\pi * 0.028 * 10 \tag{4.18}$$

$$Dist_i = 1.760 m \tag{4.19}$$

Mientras que para calcular el tiempo que tarda en recorrer dicha distancia a una velocidad de 0.256 m/s o 50 (en la plataforma real):

$$Tiempo_i = \frac{Dist_i}{V_s} \tag{4.20}$$

$$Tiempo_i = \frac{1.76}{0.256} \tag{4.21}$$

$$Tiempo_i = 6.875s \tag{4.22}$$

Estos cálculos teóricos solo se realizan para dar perspectiva física. Pues en realidad, interesa comparar los datos de simulación contra los reales. Sin embargo, cabe mostrar los datos ideales para que le sea evidente al lector la validación del análisis dinámico realizado y la semejanza entre tanto el fenómeno físico como el simulado. Hay varios motivos por los cuales las mediciones reales son ligeramente distintas a los datos teóricos. El primero es que en realidad la rueda de la plataforma física no es un círculo perfecto. El segundo es que el degrado del *kit* robótico ocasiona que si bien se especifica una velocidad de 50, los motores no llegan a esta de manera precisa. El tercero es que el *kit* robótico real no avanza perfectamente hacia adelante, si no que se desvía ligeramente hacia un lado, afectando la medición de distancia. Cuarto, es que el cálculo teoríco aquí realizado no considera los pequeños intervalos de aceleración y desaceleración de los motores. Y el quinto, se puede considerar impresiones en la toma de mediciones.

En la Tabla.4.6, se muestra la comparación entre los valores promedio obtenidos para cada variable entre la plataforma robótica real y la plataforma robótica simulada. Note que se consigue un porcentaje de error menor al 3 porciento para ambas variables. Esto valida el funcionamiento programático de los motores del modelo computacional. Este porcentaje de error se puede justificar mediante, primero, la desviación hacia un lado que experimenta la plataforma robótica real al intentar moverse en línea recta y segundo, debido a las impresiones de cálculo que realiza Gazebo en los movimientos.

| Parámetro | Simulado | Real | Porcentaje de error [ptc] |
|---|---|---|---|
| Tiempo [s] | 7.012 | 7.164 | 2.12 |
| Distancia [m] | 1.730 | 1.736 | 0.35 |

**Tabla 4.6:** Comparación entre los valores promedio de tiempo y velocidad para la prueba realizada entre los motores de simulación y los motores reales. Fuente: Elaboración propia

Finalmente, luego de validar el segundo eje de diseño para los actuadores del modelo computacional. Para el tercer eje de diseño, cabe decir, que el control de los motores en la simulación responde fielmente al tipo de control utilizado en la plataforma robótica. Únicamente, en lugar de especificar un valor de 0 a 100, se envía por medio de un tema

ROS un valor en rad/s (si es movimiento de motores individual) o un mensaje tipo Twist (si el movimiento de motores es sincronizado) desde un nodo de procesamiento. Y esto realizado al propio para aumentar el impacto educativo en este tema. Por lo que se considera que los actuadores resultantes también cumplen con el tercer y último eje de diseño.

### 4.4.3 Creación del entono de trabajo en ROS

Todo el esfuerzo del modelo computacional en Gazebo y ROS, pierde importancia si no se pone a disposición las herramientas necesarias para utilizarlo libremente. Para esto, se desarrollaron las carpetas del entorno ROS para garantizar la interconexión dinámica y configurable con Gazebo. El usuario esta en capacidad de agregar modelos, modificar el mundo, correr nodos ROS a gusto y lo más importante, de diseñar comportamientos programáticos en la plataforma robótica de simulación semejantes a su contra parte real. Esto mediante un *script* que realiza la lectura de sensores por medio de ROS, ejecuta sus algoritmos de comportamiento y luego envía órdenes a los actuadores en Gazebo. Para motivos de validación, se adjunta el siguiente enlace.

https://splice.gopro.com/v?id=VZYz9Z

El mismo sirve para validar diversas funcionalidades. En el vídeo se puede apreciar el diseño de un comportamiento en el modelo computacional de simulación y luego el mismo comportamiento ejecutándose en la plataforma real. Dicho comportamiento consiste en que la plataforma inicie su movimiento solo si existe un nivel de luz adecuado (validando de forma práctica el sensor de color en modo ambiente). Una vez esto, la plataforma se moverá 10 revoluciones hacia adelante a media velocidad (validando tanto los *encoders* como los motores grandes -para cuando el mensaje Twist solo tiene componente lineal). Al final de dicho movimiento, girará hacia un lado hasta los 90 grados (validando el sensor giroscopio y los motores grandes -para cuando el mensaje Twist solo tiene componente angular). Por último, tomará la medición del sensor ultrasónico y si esta es menor a 50 cm, entonces bajará el brazo (validando el sensor ultrasónico y el motor mediano).

La plataforma robótica se programa mediante RobotC, un *software* el cual permite la programación del bloque del Lego Mindstorms EV3 mediante el lenguaje de programación C y algunas herramientas especialmente diseñadas por este *software*, como los comandos de lectura de sensores y envío de órdenes a los actuadores. Si bien se podría argumentar que en este vídeo lo único que se sabe con certeza es que se valida el flujo lógico de la programación del comportamiento, esto no es cierto. Pues también se valida que las mediciones de los sensores tengan el sentido físico acorde y aproximen de buena manera su medición real. Se comparó anteriormente, las mediciones entre cuando la plataforma de simulación y la real avanzan 10 revoluciones hacia adelante. También, se comparó la medición del sensor giroscopio para un ángulo de 90. Además, se comparó la medición del sensor ultrasónico para 50 cm. Y por último, en general el funcionamiento de los actuadores. En la Tabla. 4.7 se adjunta los valores umbrales correspondientes a los

sensores y actuadores que se utilizaron tanto en la simulación como en el real.

| Parámetro | Real | Simulación | Porcentaje de error [ptc] |
|---|---|---|---|
| Velocidad de motores grandes | 50 ptc | 0.2562 m/s | 0 |
| Sensor giroscopio [°] | 90 | 90 | 0 |
| Sensor ultrasónico [cm] | 50 | 50 | 0 |
| *encoders* [°] | 3600 | 3600 | 0 |
| Sensor de color [ptc] | 40 | 70 | 75 |

**Tabla 4.7:** Comparación de los valores utilizados en la simulación y en la realidad. Fuente: Elaboración propia

Note la semejanza de los valores utilizados, y dado que eventualmente la simulación y la realidad se comportan de igual manera, se puede decir con autoridad que los ejes de diseño se cumplen en totalidad. Para el sensor de color, note que si bien sus mediciones no son las mismas cumple con el sentido práctico del comportamiento, al poder modificar de diferente manera - pero igual de fácil - la variable de su entorno medible a la realidad. El proceso de lectura y envío de información, se lleva a cabo con una simple línea de código, semejante a la implementación en RobotC y en el *software* de programación oficial. Además, el formato de todas las mediciones en los temas ROS cumplen con sus debidos comportamientos característicos. No solo el modelo computacional funciona tal y como funciona la plataforma real, si no que también se puede analizar el entorno programático de ROS y sus funcionalidades. Sin embargo, para esto es mejor el siguiente vídeo:

https://splice.gopro.com/v?id=q55DDE

En el vídeo, se muestra la práctica más habitual que los estudiantes realizan en la asignatura de robótica. Un algoritmo de búsqueda de luz, en donde la plataforma debe encontrar el punto máximo de luz en su entorno. Este algoritmo funciona con cuatro tareas que se ejecutan constantemente. La primera, es una simple tarea de avance. La segunda, una tarea que permite a la plataforma seguir la pared más cercana. Una tercera, que ocasiona que la plataforma compare dos puntos de luz y se mueva al mayor. Y una cuarta que evita que la plataforma colisione con paredes.

Antes de entrar a analizar con detalle la interacción de la plataforma con el entorno simulado y su comparación con el real, primero note el funcionamiento del entorno ROS. El usuario ha sido capaz de inicializar el mundo de simulación de Gazebo desde ROS e insertar de manera dinámica tres modelos SDF. Además, ha modificado las propiedades de iluminación del mundo, para mejor rendimiento del algoritmo. Los modelos que se insertaron son la plataforma robótica, la pista de carreras - la cual fue diseñada a gusto del usuario y con toda la capacidad de configuración - y la bombilla. Finalmente, crea un nodo de procesamiento en la carpeta *script* responsable de ejecutar el comportamiento diseñado para que la plataforma ejecute el algoritmo. El paquete ROS -y sus componentes- esta debidamente acondicionado para suplir las necesidades de simulación que el usuario desee.

Con este vídeo y algoritmo de comportamiento, se pretende validar todos los demás detalles del modelo computacional desarrollado. En dicho, se evidencia el alcance de los tres ejes de diseño. Existen las herramientas necesarias para el funcionamiento de la plataforma, su interacción física responde a su contra parte real y su uso y comportamientos a ojos del usuario es semejante al *kit* robótico real. En la Tabla.4.8, se muestra la comparación entre los valores umbrales usados en la simulación y en la realidad.

| Parámetro | Real | Simulación | Porcentaje de error [ptc] |
|---|---|---|---|
| Distancia cerca de pared [cm] | 40 | 40 | 0 |
| Distancia lejos de pared [cm] | 60 | 60 | 0 |
| Distancia para evitar choque [cm] | 20 | 20 | 0 |
| Valor mínimo de luz [ptc] | 20 | 40 | 200 |
| Valor máximo de luz [ptc] | 40 | 80 | 200 |
| Velocidad angular de giro | 25 ptc | 2.0 rad/s | 0 |
| Velocidad lineal de avance | 30 ptc | 0.15 m/s | 0 |
| Velocidad lineal de retroceso | 40 ptc | 0.2 m/s | 0 |

**Tabla 4.8:** Comparación de los valores utilizados en la simulación y en la realidad para el algoritmo de seguimiento de luz. Fuente: Elaboración propia

Para entender los parámetros mostrados, es necesario explicar de forma sencilla la programación del algoritmo. Distancia cerca de pared, corresponde al valor umbral en el que si la plataforma se encuentra a una distancia menor a esta, entonces se alejará de la pared. Distancia lejos de pared, corresponde al valor umbral en el que si la plataforma se encuentra a una distancia mayor a esta, entonces se acercará a la pared. Distancia para evitar choque, corresponde al valor umbral en el que si la plataforma se encuentra a una distancia menor a esta, esta cerca de chocar por lo que ejecuta su tarea de moverse hacia atrás a una velocidad igual a Velocidad lineal de retroceso. Valor mínimo de luz, es el valor umbral que al sensor de color en modo ambiente detectar, empieza a ejecutar su tarea de seguimiento de luz. Valor máximo de luz, es el valor de la medición del sensor de color en modo ambiente el cual se considera que la plataforma ha encontrado la luz. La velocidad angular de giro, es la velocidad a la que los motores grandes se mueven para hacer girar la plataforma robótica. La velocidad lineal de avance es la velocidad a la que los motores grandes se mueven para hacer avanzar la plataforma en línea recta. Todo esto se evidencia mejor en el vídeo.

El lector podrá notar, el bajo porcentaje de error entre los valores usados en la simulación y en la realidad. Para las velocidades implementadas, se sigue la relación 4.23, en donde para la velocidad lineal de avance y retroceso se utiliza un Vr igual a 30 y 40 respectivamente.

$$\frac{V_{as}}{V_{ar}} = \frac{0.513}{100} \tag{4.23}$$

$$V_{as} = 30\frac{0.513}{100} \tag{4.24}$$

$$V_{as} = 0.15m/s \tag{4.25}$$

$$V_{rs} = 40\frac{0.513}{100} \tag{4.26}$$

$$V_{rs} = 0.20m/s \tag{4.27}$$

Y para la velocidad angular de giro, se sigue la relación 4.28, en donde para su valor real se utiliza un valor igual a 26.

$$\frac{W_1}{W_r} = \frac{7.71}{100} \tag{4.28}$$

$$W_1 = 26\frac{7.71}{100} \tag{4.29}$$

$$W_1 = 2.0rad/s \tag{4.30}$$

Ambas velocidades, simuladas y reales son iguales. Sin embargo, esto no quiere decir de que siempre se presentará una situación como esta. Considere por ejemplo el sensor ultrasónico, para este caso los valores fueron iguales pero, para otro comportamiento en el que se interactué con objetos a una orientación diferente, esto puede ocasionar cambios en los valores utilizados. Sin embargo, - como se mencionó anteriormente- una vez que se conoce la limitación se puede trabajar para esta. Por ejemplo, en este algoritmo se hubiese conseguido un algoritmo más robusto si las mediciones de distancia entre la pared y la plataforma siempre se hubiesen realizado perpendicularmente (ayudándose con el giroscopio). Esto en realidad termina sumando mayor peso educativo al modelo computacional.

También considere la situación por la que pasa el sensor de color en modo ambiente. Sus valores no son para nada iguales, pero -como se vio en el vídeo- esto no significa que no sea práctico. Definitivamente, si el sensor de color no cumpliera su objetivo, tratar de simular un algoritmo buscador de luz sería impensable. Pero como se evidenció, se logra conseguir funcionalidad del mismo. Es solo tratar con la variable de diferente forma en el entorno programático y en el entorno real. Para una vez validado el algoritmo, realizar los ajustes de valor necesarios.

Aún con que los valores umbrales establecidos hayan sidos los mismos, se podrá notar que en realidad la plataforma en simulación y la realidad no se comportan exactamente igual, al realizar todo el algoritmo. Esto se debe a diversas fuentes de error. La primera

es por que -como se mencionó anteriormente- el entorno real es mucho más complejo que el entorno simulado, y siempre habrán variables ajenas a la simulación que afectan el comportamiento real. Segundo es la afectación del ruido en las mediciones de los sensores, aunque el ruido es considerado en la simulación, esta afectación siempre sucederá de diferente manera lo que puede ocasionar ligeras diferencias en los comportamientos afectados. La tercera es la complejidad del algoritmo, el algoritmo al depender de las mediciones de los sensores y en el movimiento de los actuadores, no siempre tomará los mismos caminos, por lo que a largo plazo se puede esperar encontrar diferencias de comportamiento. La cuarta es la diferencia entre herramientas utilizadas, la simulación utiliza ROS y Python (para escribir el algoritmo de luz) mientras que la plataforma real utiliza las herramientas de RobotC y se encuentra programado en C. Si bien estas dos diferencias se pueden mapear de una a otra, nunca se puede garantizar su exactitud.

A pesar de todo esto, el comportamiento obtenido entre la simulación y la realidad es muy semejante por lo que se cuenta con una simulación robusta y exitosa. Se ha conseguido entonces un modelo computacional de simulación para el Lego Mindstorms EV3 fiable y válido para sus fines educativos.

## 4.5 Requisitos computacionales del modelo de simulación

El cuarto eje de diseño que se ha estado considerando a lo largo de todo el desarrollo del modelo computacional de simulación, es el rendimiento computacional del mismo a nivel de exigencia de procesador y de tarjeta gráfica de vídeo. Este eje es importante debido a que por ser un desarrollo *software*, se deben especificar los requerimientos mínimos para su funcionamiento. Además, al trabajar con Gazebo y ROS que son plataformas digitales capaces de requerir tanto poder computacional como se necesite, es fundamental establecer límites. Decisiones tomadas en las etapas de solución relacionadas con los *meshes*, generación de datos, geometrías de colisión y visuales utilizadas, juegan todas un rol en este eje de diseño.

El desarrollo de la totalidad del modelo computacional de simulación del Lego Mindstorms EV3, se llevó a cabo utilizando dos ordenadores. Una portátil y un ordenador de escritorio con las especificaciones de la Tabla.4.9. En la Fig.4.9, se muestra la exigencia computacional del procesador para simular el modelo computacional desarrollado.

| Especificación | Portátil | Ordenador de escritorio |
|---|---|---|
| Sistema operativo | Ubuntu 16.04 LTS | Ubuntu 16.04 LTS |
| CPU | Intel Core i3-3110M x4 de 2.40 GHz | Intel Core i5-2500 x4 de 2.80 GHz |
| GPU | Intel Ivybridge Mobile (integrada) | Nvidea GTX 1050 Ti (dedicada) |
| RAM | 6 GB | 8 GB |

**Tabla 4.9:** Especificaciones de ambos computadores utilizados para el desarrollo, prueba y uso del modelo computacional de simulación. Fuente: Elaboración propia

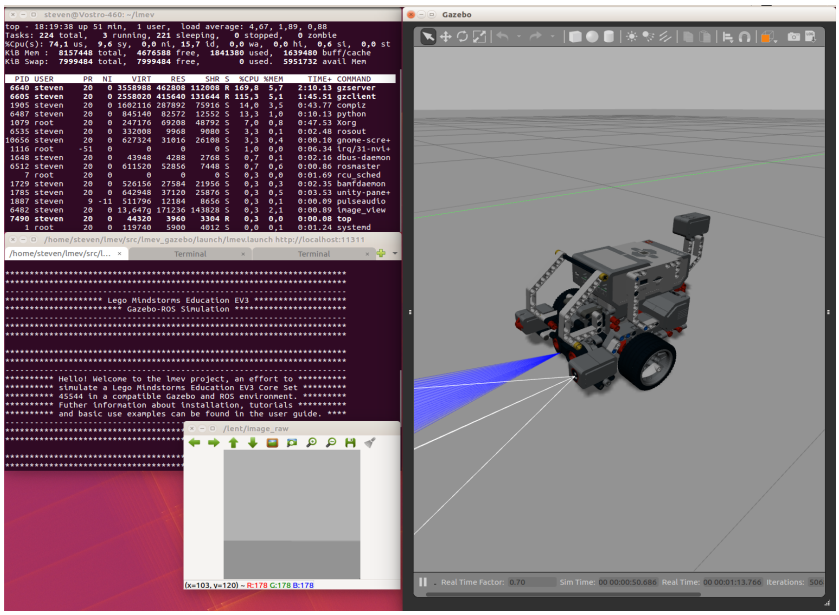



**Figura 4.9:** Carga computacional ocasionada por la simulación en el ordenador Intel-i5. Fuente: Elaboración propia.

Gazebo en su interfaz gráfica, pone a disposición del usuario un factor de tiempo real. Este factor representa una relación entre los segundos simulados y los segundos reales. De tal forma en que si se tiene un factor de 0.5, los acontecimientos que suceden en la simulación están pasando el doble de lento de lo que pasarían en la vida real. Este factor, es evidente que se relaciona directamente con la capacidad de computación que ofrece el procesador del ordenador que ejecuta Gazebo. Ente más capaz sea el ordenador, mayores factores de tiempo real se pueden alcanzar. Si por ejemplo, el computador también esta corriendo otros procesos, como tener el navegador abierto o estar escribiendo un documento de texto, entonces el procesador podrá dedicarle menos atención a Gazebo y por ende el factor de tiempo real disminuye.

Esto es importante para analizar lo que se presenta en la Fig.4.9. En dicha, se muestra la carga a nivel de procesamiento que esta ocupando cada proceso en el computador y se puede observar el factor de tiempo real. Como no hay ningún otro proceso ejecutándose entonces el procesador puede dedicarse únicamente al modelo computacional de simula-

ción y es por esto que utiliza aproximadamente un 350 porciento de procesamiento, (el ordenador tiene 400 porciento en total, 100 en cada núcleo de procesador). Esto a primera instancia supondría que la simulación es bastante exigente, considerando las especificaciones del ordenador, sin embargo esto es erróneo por lo ya explicado. Si se ejecutaran más procesos entonces la carga en el procesador correspondiente a la simulación disminuye, con la condición de reducir su factor de tiempo real a menos del 0.7 que se puede apreciar.

Para dar perspectiva al lector -, según [8], los requerimientos recomendados para trabajar con Gazebo son los mostrados en la Tabla.4.10

| Especificación | Recomendados para Gazebo |
|---|---|
| Sistema operativo | Ubuntu |
| CPU | Intel Core i5 |
| GPU | Dedicada |
| RAM | 6 GB |

**Tabla 4.10:** Especificaciones de *hardware* recomendadas para correr Gazebo. Fuente: Elaboración propia

Note que el ordenador portátil con el que se desarrolló y probó el modelo computacional, en realidad no cumple con las especificaciones recomendadas de Gazebo. Todo el desarrollo del proyecto se llevó a cabo en estos dos ordenadores, usando uno u otro para tareas especificas pero eventualmente en ambos se probaba constantemente la totalidad de las simulaciones. El modelo computacional funciona a la perfección en ambos ordenadores. En el ordenador de escritorio es capaz de trabajar a una velocidad mayor - lo cual es conveniente si se diseñan comportamientos de tiempo muy extenso-, en términos del factor de tiempo real. Mientras que en el ordenador portátil también se puede utilizar en su totalidad la simulación sin ningún inconveniente - a parte de trabajar con factores de tiempo real más bajos. Esto evidencia que el modelo computacional, al correr en el ordenador portátil de baja gamma, cumple con el cuarto y último eje de diseño.

Esto es fundamental, debido a que el desarrollo es para estudiantes y no se puede pretender que cada estudiante tenga una computadora con especificaciones de última generación. Sin embargo, se procede a -de manera conservativa- especificar los requerimientos mínimos de un ordenar que pretenda correr la simulación. Aún sabiendo que se podría intentar ejecutar en un ordenador de menor gamma, como el portátil utilizado. En la Tabla.4.11 se muestran las especificaciones recomendadas para utilizar el modelo computacional de simulación del Lego Mindstorms EV3.

Estos requerimientos, en realidad se apegan mucho a los establecidos por Gazebo. Y si bien son conservativos, en realidad no son fuera de lo cotidiano para un estudiante de ingeniería - lo cual es lo que finalmente se busca. Cualquier computador ingenieril de no más de 3 años de antigüedad, puede ejecutar el modelo computacional sin problema alguno. Por lo que, el modelo computacional en simulación del Lego Mindstorms EV3 cumple con el cuarto y último eje de diseño.

| Especificación | Recomendados para el modelo computacional |
|---|---|
| Sistema operativo | Ubuntu 16.04 LTS |
| CPU | Intel Core i5 x4 (o semejante) |
| GPU | Tarjeta gráfica dedicada de 1 GB |
| RAM | 6 GB |

**Tabla 4.11:** Especificaciones de *hardware* recomendadas para correr el modelo de simulación. Fuente: Elaboración propia

## 4.6 Alcance educativo

Los resultados y análisis aquí presentados, indican un desarrollo del modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3 funcional, práctico y fiable. Sin embargo, su verdadero éxito depende de si satisface los alcances educativos que se propusieron en un inicio. Pues, se debe recordar que el verdadero problema es que los estudiantes tienen un tiempo de trabajo insuficiente e ineficiente con las prácticas de curso en las que utilizan el *kit* robótico, al no tener manera de avanzar con la misma cuando no cuentan con dicha plataforma en físico. Entonces, se analizan los alcances educativos producto de los resultados obtenidos.

1. Los estudiantes pueden trabajar en sus prácticas de curso sin dependencia de los demás grupos de trabajo. Cada grupo puede trabajar de manera simultánea con su propio modelo computacional, aun cuando otro grupo se encuentra utilizando el *kit* físico. Esto resulta en un aumento en el tiempo de trabajo disponible para desarrollar la práctica.

2. Los estudiantes pueden ahora avanzar con su práctica del curso desde su lugar de habitación, realizando las simulaciones necesarias del comportamiento del sistema robótico para la misma. Esto debido a que, el modelo computacional al ser poco exigente a nivel computacional, se puede instalar el modelo computacional en su ordenador ingenieril y utilizarlo sin problema desde cualquier lugar. Esto resulta en un aumento en el tiempo de trabajo disponible para desarrollar la práctica.

3. Los estudiantes, al tener disponibles todas las herramientas del modelo y un comportamiento semejante al real, pueden realizar aproximaciones previas al sistema robótico que les ayude a formar experiencia sobre el uso y manejo del mismo. Debido a que con el modelo computacional pueden validar el flujo lógico de su algoritmo, el estudiante puede darse cuenta de qué ha hecho mal y qué debe corregir, para cuando realice la práctica con el sistema en físico cometa menos errores que comprometan negativamente el estado del *kit*. De esta manera, se reducen los daños ocasionados por la inexperiencia del usuario y se conserva el buen estado del sistema robótico. Además, se evitan paros indeseable en los tiempos de trabajo.

4. El modelo de simulación, al haber sido desarrollado siempre con la mentalidad de poder ser configurable y adaptable a las necesidades del usuario, ofrece la posibilidad de experimentar con el *kit* y simular las prácticas de curso con anterioridad. Esto permite a los estudiantes realizar la práctica con conocimiento previo, y preparados para las eventuales situaciones que ameriten la práctica. Los estudiantes pueden entonces, sacar más provecho al tiempo de uso con el *kit* en físico y enfocarse en los objetivos de las prácticas académicas.

5. El modelo de simulación computacional del sistema robótico Lego Mindstorms EV3 desarrollado representa un aporte intelectual a la comunidad científica y tecnológica pues, no se han realizado esfuerzos con anterioridad para implementar una simulación de este *kit* didáctico. El modelo computacional de simulación para el *kit* robótico no solo solventa las complicaciones actuales, si no que también mejora la calidad didáctica del curso de robótica y enriquece la experiencia y aprendizaje académico del estudiante - al poder ser altamente configurable y fácil de usar. Abre un abanico de posibilidades de estudio, investigación y experimentación para cualquier futuro usuario y sirven a fortalecer los objetivos del curso actual.

# Capítulo 5

# Conclusiones

El modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3, ha evidenciado cumplir con sus diversos ángulos de funcionalidad. Muestra ser una solución tangible, que ataca concretamente los objetivos de su desarrollo. No solo a nivel técnico sino a nivel educacional. Se pueden anotar las siguientes conclusiones sobre el desarrollo llevado a cabo, considerando primordialmente los objetivos del trabajo planteados pero también, sobre los detalles y ejes analizados en el capítulo anterior.

Primero, el modelo computacional desarrollado dispone de los componentes de carácter esencial y de aprovechamiento del usuario que representan una fiel herramienta de práctica para el uso del kit robótico Lego Mindstorms EV3. El modelo tridimensional del sistema robótico Lego Mindstorms EV3 diseñado se asemeja a su contra parte real y el entorno de simulación diseñado responde a las características físicas del entorno real, permitiendo la configuración y adaptación deseada por el usuario. Esto quiere decir, que efectivamente se ha diseñado un modelo tridimensional del Lego Mindstorms EV3 y se ha habilitado su entorno de simulación en Gazebo. Segundo, el modelo computacional desarrollado modela fielmente los comportamientos básicos de la plataforma robótica real. El comportamiento de los sensores responde al descrito por el bloque programático. Y los actuadores permiten funcionalidades típicas de movimiento de la plataforma. El modelo computacional tiene sentido físico en su interacción con el entorno programático. Y además, mediante la interconexión programática con ROS, el usuario dispone de este entorno de simulación de manera sencilla y práctica. Las condiciones de uso y manejo de las herramientas desarrolladas se apegan a su contra parte real. Tercero, la guía de usuario desarrollada pone a entera disposición del usuario toda la información e instrucciones necesarias para el aprovechamiento del modelo computacional. Siempre dirigida a los estudiantes, ejemplificando y explicando los elementos de uso común en sus prácticas de curso. Como la lectura de mediciones y envío de órdenes en algoritmos complejos, y el uso de las demás herramientas del entorno de simulación. Además, los desarrollos programáticos se encuentran bien documentados para posibilitar y facilitar su futura actualización y modificación en función de las necesidades del usuario. Cuarto - y último, el modelo computacional desarrollado abre un abanico de oportunidades educativas y cumple con su objetivo final de

presentarse ante los estudiantes como una herramienta para las prácticas de la asignatura de robótica del grado de informática de la UDC en las que utilizan el kit robótico Lego Mindstorms EV3. Ahora los estudiantes pueden aprovechar de mejor manera el tiempo dedicado a dichas prácticas, al utilizar el modelo computacional de simulación desarrollado y sus herramientas. Antes, no existía tal herramienta digital, pero ahora existe y esta disponible para los estudiantes.

Sin embargo, cabe también rescatar un par de recomendaciones sobre el desarrollo del modelo computacional de simulación del Lego Mindstorms EV3. La primer recomendación es mejorar el desempeño de los sensores simulados, específicamente el sensor de color y el sensor ultrasónico. Si bien estos cumplen en su mayoría con los sentidos prácticos de la simulación, sería ideal que la medición del sensor de color sea fiel a la medición de su semejante real y que el comportamiento del sensor ultrasónico para superficies inclinadas modele de mejor forma el comportamiento real. Esto en realidad supone más un reto a nivel de las herramientas *software* utilizadas, pero eventualmente, se podrían realizar esfuerzos propiamente en el modelo para cada vez más acercar lo simulado a lo real. La segunda recomendación consiste en la interconexión del bloque programático del Lego Mindstorms EV3 con ROS. Actualmente, los estudiantes luego de validar su algoritmo en la simulación, deben ahora programarlo en la plataforma robótica real. Esto se realiza mediante el *software* RobotC el cual permite programar la plataforma real mediante el lenguaje C y algunas herramientas propias del *software*. Sin embargo, sería ideal que el estudiante luego de que haya terminado de trabajar con la simulación pueda programar la plataforma robótica real de la misma manera. Esto es, con el lenguaje de programación Python y con las herramientas que ofrece ROS en la simulación. Con esto, se conseguiría un flujo de trabajo mucho más eficiente y versátil, en el cual el estudiante se vería beneficiado.

Finalmente, cabe rescatar el hecho de que el estudiante antes no contaba con ninguna herramienta para realizar una aproximación al sistema robótico. Pero ahora, -como desarrollo o producto final del presente trabajo- el estudiante dispone de toda una herramienta digital en sus manos, fiel a su semejante real y adaptable a sus necesidades.

# Bibliografía


[1] F. P. Beer, R. Johnston, and P. J. Cornwell. *Mecánica vectorial para ingenieros. Dinámica*. Mc Graw Hill, 2010.

[2] Blender. Blender [online]. 2017 [visitado el 14 de marzo de 2018]. URL https://www.blender.org/.

[3] Cailliau.org. A classic brick [online]. s.f. [visitado el 14 de marzo de 2018]. URL https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313.

[4] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[5] Universidade da Coruña. Presentación de la universidad [online]. 2017 [visitado el 14 de marzo de 2018]. URL https://www.udc.es/sobreUDC/.

[6] Grupo Integrado de Ingeniería. Grupo integrado de ingeniería [online]. 2017 [visitado el 14 de marzo de 2018]. URL http://www.gii.udc.es/.

[7] International Organization for Standardization. Iso/pas 17506:2012 [online]. 2012 [visitado el 14 de marzo de 2018]. URL https://www.iso.org/standard/59902.html.

[8] Open Source Robotics Foundation. Gazebo [online]. 2017 [visitado el 14 de marzo de 2018]. URL http://gazebosim.org/.

[9] Open Source Robotics Foundation. Ros [online]. 2017 [visitado el 14 de marzo de 2018]. URL http://www.ros.org/.

[10] Open Source Robotics Foundation. Sdf [online]. 2017 [visitado el 14 de marzo de 2018]. URL http://sdformat.org/.

[11] LEGO. 31313 mindstorms ev3 [online]. 2014 [visitado el 14 de marzo de 2018]. URL https://www.cailliau.org/Lego/Dimensions/zMeasurements-en.xhtml.

[12] LEGO. Ev3 hardware developer kit [online]. 2014 [visitado el 14 de marzo de 2018]. URL https://education.lego.com/en-us/support/mindstorms-ev3/developer-kits.

[13] LEGO. Lego mindstorms education ev3 building instructions [online]. 2014 [visitado el 14 de marzo de 2018]. URL https://education.lego.com/en-us/support/mindstorms-ev3/building-instructions.

[14] LEGO. Lego digital designer [online]. 2015 [visitado el 14 de marzo de 2018]. URL http://ldd.lego.com/es-ar/.

[15] LEGO. Lego mindstorms education ev3 core set [online]. 2017 [visitado el 14 de marzo de 2018]. URL https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set-/5003400.

[16] LeoCAD.org. Leocad [online]. 2017 [visitado el 14 de marzo de 2018]. URL http://www.leocad.org/.

# Apéndice A

# Guía de usuario

La guía de usuario se puede encontrar como archivo adjunto al presente documento. Esto debido a una cuestión de formato, versatilidad y facilidad para el estudiante.

# Guía de usuario

## 1 Bienvenida

Bienvenido al *modelo computacional de simulación del sistema robótico Lego Mindstorms EV3*. Este modelo computacional tiene todos los elementos necesarios para poder simular de manera fiable cualquier comportamiento que se desee implementar en la plataforma robótica real. Con dicho desarrollo, es ahora fácil desarrollar las prácticas de la asignatura de robótica sin la necesidad de utilizar el Lego Mindstorms EV3 en físico. Y se abren muchos caminos de alcance educativo e investigativo.

En la presente guía se pueden encontrar tres herramientas para el usuario:

- Un manual de instalación de las herramientas software necesarias para la utilización y aprovechamiento del modelo computacional.
- Un manual de uso básico de los diferentes elementos del modelo computacional que provea el conocimiento necesario para el manejo del mismo en breve tiempo.
- Ejemplos sencillos sobre el comportamiento del sistema robótico en donde se detallan los pormenores de lectura de sensores y envío de órdenes a los actuadores de la configuración del kit.

Además, el usuario encontrará en los códigos de programación disponibles, explicaciones y comentarios a más bajo nivel con el fin de proveer una comprensión de los ejemplos y uso de la simulación más profunda.

## 2 Manual de instalación

### 2.1 Requerimientos software y hardware

A continuación, en la Tabla 1 se muestran las especificaciones computacionales recomendadas para utilizar el modelo computacional de simulación del Lego Mindstorms EV3.

| Especificación | Recomendación para el modelo computacional |
|---|---|
| OS | Ubuntu 16.04 LTS |
| CPU | Intel Core i5 x4 (o semejante) |
| GPU | Tarjeta gráfica dedicada de 1 GB |
| RAM | 6 GB |
| Disco Duro | 1 GB de espacio libre |

Tabla 1. Requerimientos computacionales para el modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3. Fuente: Elaboración propia.

### 2.2 Instrucciones de instalación

Para el aprovechamiento del modelo computacional es necesario instalar las siguientes herramientas:

- ROS Kinetic Kame
- Gazebo 7.x
- lmev package

El proceso de instalación es muy sencillo. Para instalar ROS Kinetic Kame, se deben seguir las instrucciones del siguiente enlace.

http://wiki.ros.org/kinetic/Installation/Ubuntu

Al seguir dichas instrucciones, se instalará ROS y a la vez la versión más reciente de Gazebo 7. Adicionalmente, se debe instalar el paquete “gazebo_ros” siguiente las instrucciones “Pre-Built Debians” del siguiente enlace.

http://gazebosim.org/tutorials?tut=ros_installing

Finalmente, cabe nada más descargar la carpeta “lmev” puesta a disposición por el profesor y colocarla en el directorio “/home”.

# 3 Manual de uso

Antes de iniciar con lo que respecta propiamente con el modelo computacional en cuestión, es recomendable que el usuario cuente con conocimientos previos de las herramientas que se utilizan en la simulación. Para esto, se deben seguir los siguientes tutoriales. Al seguir dichos tutoriales, se garantiza que el usuario este en la capacidad de entender y manejar el modelo computacional de simulación a un nivel básico pero suficiente.

- Contar con un conocimiento básico del sistema operativo Linux. Con lo que respecta a comandos y manejo desde terminales. Se puede encontrar un tutorial básico en el siguiente enlace. No es necesario realizarlo todo, pero es una buena guía de futura referencia.
  http://www.ee.surrey.ac.uk/Teaching/Unix/

- Contar con un conocimiento básico del funcionamiento del software Gazebo. Con lo que respecta a manejo de interfaz y las plataformas robóticas. Se puede encontrar un tutorial básico en el siguiente enlace. Es necesario realizar los primeros tres tutoriales de la categoría de principiantes y recomendable realizar hasta el sexto tutorial para intermedios.
  http://gazebosim.org/tutorials

- Contar con un conocimiento básico del funcionamiento del software ROS. Con lo que respecta a sus conceptos básicos y comandos. Se puede encontrar un tutorial básico en el siguiente enlace. Es necesario realizar el tutorial 4, 5, 6 y 12 de la categoría de principiantes y recomendable realizar del 1 al 13 de la misma categoría.
  http://wiki.ros.org/ROS/Tutorials

## 3.1 Inicialización de la simulación

Una vez la carpeta “lmev” contenida en el directorio “/home”, se debe abrir una terminal (Ctrl+Alt+T). En dicha terminal, se debe correr una única vez el siguiente comando:

```
echo "source ~/lmev/devel/setup.bash" >> ~/.bashrc
```

Dicho comando sirve para habilitar el paquete ROS en cualquier futura terminal que se abra. Posteriormente, para inicializar la simulación se deben ejecutar los siguientes comandos:

```
cd lmev
./run_lmev.sh
```

Se debe recordar que si se presiona TAB se autocompletan los comandos en la terminal. Al correr este último comando, se abrirá una nueva terminal con tres pestañas, se inicializará Gazebo en pausa, se cargara la plataforma robótica en el mundo de simulación y se abrirá una pequeña ventana que muestra la visualización de la cámara.
La primer pestaña de la nueva terminal es la responsable de ejecutar roscore y por ende de inicializar todos los elementos ROS necesarios para la simulación, incluyendo el ROS Master. Además, da una pequeña bienvenida en la que se encuentra información sobre el paquete desarrollado. La segunda pestaña de la nueva terminal es la responsable de iniciar la visualización de la cámara que posee la plataforma robótica. El sensor de color es simulado mediante una cámara, por lo que es imperativo mantener tanto esta pestaña como la ventana abierta en todo momento que se utilice el modelo computacional. La tercera pestaña de la nueva terminal muestra en pantalla los valores actuales de todos los sensores presentes en la plataforma robótica simulada. Solo se actualizarán los valores mientras la simulación en Gazebo no esté en pausa y es solo con el fin de visualización. Más adelante se retomará esta pestaña.

Con lo que respecta a la ventana de Gazebo inicializada, se muestra una interfaz gráfica de usuario común para este software y un entorno de simulación básico diseñado. Al igual que la plataforma robótica simulada en el centro del mundo de simulación, lista para su uso.

### 3.2 Control de la simulación

Para el control del comportamiento de la plataforma robótica, existen dos acercamientos. El primer acercamiento es mediante un control manual por terminal. Este consiste en enviar y visualizar datos de la plataforma robótica en una terminal para mayormente familiarizarse con las herramientas de la simulación o para pequeñas pruebas de debug. El segundo

acercamiento es mediante un control programático por un nodo ejecutable de Python. Este último es especialmente útil para cuando se deseen diseñar comportamientos complejos y que posteriormente se deseen trasladar a la plataforma robótica real.

### 3.2.1 Control por terminal

Inicialice la simulación

```
cd lmev
./run_lmev.sh
```

En la interfaz gráfica de Gazebo, presione el botón de “Play”, esto habilitará el movimiento de la plataforma robótica, y el funcionamiento de los sensores – los cuales puede comprobar en la tercer pestaña de la nueva terminal. A partir de este momento, Gazebo está utilizando sus motores de cálculo y gráficos para calcular y mostrar dinámicamente un comportamiento de la plataforma robótica fiel a su semejante real. Presione ahora el botón de “Pausa” para tratar primero con algunos otros aspectos de la simulación.

En general, a nivel de usuario el control que se desea es poder realizar una lectura de las mediciones de los sensores y un poder enviar órdenes a los actuadores de la plataforma robótica en simulación. Este proceso se lleva a cabo mediante la comunicación y uso de nodos y topics en el entorno ROS. El usuario, al haber realizado los tutoriales propuestos, debe estar familiarizado con estos conceptos y comandos de ROS, al igual que sus principios de funcionamiento.

En la terminal ejecute el siguiente comando.

```
rostopic list
```

Se enlistará, una serie de topics activos que ROS Master está manejando. De los cuales importan los siguientes:

- /Arm_Motor/vel_cmd
- /Color_Sensor/Ambient
- /Encoder_Arm_Sensor

- /Encoder_Left_Wheel_Sensor
- /Encoder_Right_Wheel_Sensor
- /Gyroscope_Sensor
- /Left_Wheel_Motor/vel_cmd
- /Right_Wheel_Motor/vel_cmd
- /Touch_Sensor
- /Ultrasonic_Sensor
- /cmd_vel

Cada topic de estos maneja información que recibe o envía de un nodo de procesamiento ROS. Aquellos topics que contengan en su nombre la palabra “Sensor” son resultado de un nodo publicador, el cual constantemente envía mensajes por medio de ellos. Mientras que aquellos topics que contengan en su nombre la palabra “cmd” son topics que al recibir un mensaje, envían dicha información a un nodo subscritor para realizar algún tipo de procesamiento.

Para visualizar desde la terminal, la medición de un sensor de la plataforma robótica en simulación, se utiliza el siguiente comando. Considerando que la simulación se encuentre corriendo, es decir en “Play”.

```
//rostopic echo “Nombre del topic”
rostopic echo /Touch_Sensor
```

De esta manera, el usuario estará visualizando el valor actual de la medición del sensor táctil simulada. Puede detener su impresión en pantalla con Ctrl+C y luego consultar otro sensor como por ejemplo:

```
rostopic echo /Gyroscope_Sensor
```

Así, el usuario puede consultar manualmente la medición de cualquier sensor en cualquier momento. En la Tabla 2, se muestra la documentación programática necesaria para cada topic tipo Sensor.

| Nombre del topic | Funcionalidad | Tipo de mensaje | Unidades físicas |
|---|---|---|---|
| /Color_Sensor/Ambient | Indica la medición del sensor de color en modo ambiente | std_msgs/int16 | % |
| /Encoder_Arm_Sensor | Indica la medición del encoder del motor del brazo actuador | std_msgs/int16 | ° |
| /Encoder_Left_Wheel_Sensor | Indica la medición del encoder del motor del motor de la rueda izquierda | std_msgs/int16 | ° |
| /Encoder_Right_Wheel_Sensor | Indica la medición del encoder del motor del motor de la rueda derecha | std_msgs/int16 | ° |
| /Gyroscope_Sensor | Indica la medición del sensor giroscopio | std_msgs/int16 | ° |
| /Touch_Sensor | Indica la medición del sensor táctil | std_msgs/int16 | - |
| /Ultrasonic_Sensor | Indica la medición del sensor ultrasónico | std_msgs/float64 | cm |

Tabla 2. Especificación programática de los topics para los sensores de la plataforma robótica en simulación. Fuente: Elaboración propia.

Para enviar órdenes desde la terminal a los actuadores de la plataforma robótica simulada, se utiliza el siguiente comando. Se pueden enviar aun cuando la simulación se encuentre en pausa y su efecto se verá cuando se corra la simulación.

```
//rostopic pub “Nombre del topic” *Presionar dos veces TAB*
rostopic pub /Left_Wheel_Motor/vel_cmd std_msgs/Float64 "data: 0.0"
```

Luego el usuario puede manipular el argumento “data: 0.0” estableciendo cualquier número que desee, siempre y cuando cumpla con su debido formato. Al darle enter, la plataforma robótica en la simulación de Gazebo empezará a moverse acorde. Se puede detener el envío de datos con Ctrl+C pero usualmente es necesario especificar un valor igual a cero si desea que el movimiento se detenga. Así, el usuario puede manipular el estado de cada actuador

en cualquier momento. En la Tabla 3, se muestra la documentación programática necesaria para cada topic tipo cmd.

| Nombre del topic | Funcionalidad | Tipo de mensaje | Unidades físicas | Máximo valor |
|---|---|---|---|---|
| /Arm_Motor/vel_cmd | Define una velocidad angular al motor del brazo actuador | std_msgs/Float64 | rad/s | 27 rad/s |
| /Left_Wheel_Motor/vel_cmd | Define una velocidad angular al motor de la rueda izquierda | std_msgs/Float64 | rad/s | 18 rad/s |
| /Right_Wheel_Motor/vel_cmd | Define una velocidad angular al motor de la rueda derecha | std_msgs/Float64 | rad/s | 18 rad/s |
| /cmd_vel | Define una velocidad lineal y angular al movimiento simultáneo de ambas ruedas | geometry_msgs/Twist | m/s para la velocidad lineal y rad/s para la velocidad angular | Velocidad lineal: 0.513 m/s<br>Velocidad angular: 0 rad/s<br>---<br>Velocidad lineal: 0 m/s<br>Velocidad angular: 7.71 rad/s |

Tabla 3. Especificación programática de los topics para los motores de la plataforma robótica en simulación. Fuente: Elaboración propia.

Se podrá notar, que el topic /cmd_vel funciona ligeramente diferente a los demás. Dicho topic es encargado de movilizar la plataforma robótica mediante el movimiento simultáneo de ambas ruedas, semejante a la función “Move Tank”. Para esto, el usuario debe especificar un

mensaje tipo Twist. A modo de ejemplo, se puede ejecutar (emplee la funcionalidad de TAB para llegar a dicho comando):

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
  x: -0.1281
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.925"
```

Al ejecutar dicho comando, la plataforma empezará a describir una trayectoria circular – Para detener la plataforma, puede ejecutar el mismo comando pero con todos los valores en 0.0. El usuario puede entonces especificar un valor en la componente x del mensaje y en la componente z del mensaje. La primera corresponde a la velocidad lineal a la que se moverá la plataforma robótica y la segunda corresponde a la velocidad angular en el eje z a la que girara la plataforma. Note que estos dos son los únicos parámetros configurables, pues corresponden a los grados de libertad de la plataforma robótica. También, es conveniente que cuando se utilice este topic, no se utilicen los otros dos topics responsables de mover las ruedas de manera individual. Pero esto no quiere decir que no se pueda.

#### 3.2.2 Control por script

Para controlar la plataforma robótica mediante un script el usuario debe crear un archivo ejecutable de extensión .py. Dicho script es propiamente un código de programación escrito en Python, capaz de utilizar todas las herramientas que dicho lenguaje ofrece. Este script en realidad funciona como un nodo en el entorno ROS, el cual se subscribe a los topics de los sensores y publica en los topics de los actuadores. Lo único relevante para el modelo computacional es las bibliotecas que se deben agregar y las líneas de código encargadas de lectura y envío de datos. Cabe rescatar que por cuestiones de rendimiento y eficiencia es conveniente realizar el diseño programático del comportamiento deseado mediante una clase en Pyhton.

Bibliotecas necesarias:

```
# ROS libraries
import roslib
import rospy

# ROS messages
from std_msgs.msg import Int16
from std_msgs.msg import Float64
from std_msgs.msg import Float32
from geometry_msgs.msg import Twist
```

La forma más común y versátil para la lectura de la medición de un sensor de la plataforma robótica simulada se realiza mediante la siguiente línea de código:

```
//variable_sensor=rospy.wait_for_message('Nombre del topic sensor', Tipo de mensaje)
Color_Sensor=rospy.wait_for_message('Color_Sensor/Ambient', Int16)
```

Esto ocasionara que la variable “Color_Sensor” almacene el siguiente mensaje que se publique en topic “Color_Sensor/Ambient”, el cual corresponde a la medición más reciente del sensor simulado. Cabe rescatar que lo que se almacena en la variable es el mensaje, por lo que si se quiere utilizar propiamente el valor de la medición, ya sea para realizar comparaciones u operaciones, se debe accesar de la siguiente manera:

```
Color_Sensor.data
```

La forma más común y versátil para el envío de órdenes a los actuadores de la plataforma robótica simulada se realiza mediante las siguientes instrucciones.

```
//Publicador= rospy.Publisher('Topic de actuador', ‘Tipo de mensaje’, queue_size=0)
Mover_Brazo= rospy.Publisher('/Arm_Motor/vel_cmd', Float64, queue_size=0)
```

Esta instrucción se encarga de inicializar el topic, por lo que es importante ejecutar dicha línea en las funciones de inicialización del algoritmo. Luego, a lo largo del algoritmo, para

enviar una orden en concreto, se sigue el siguiente comando. Esto ocasionará un movimiento de la articulación correspondiente acorde, siempre y cuando “Data” cumpla con el formato debido.

```
//Publicador.publish(“Data”)
Mover_Brazo.publish(-0.5)
```

Prácticamente, estos métodos son la versión programática del rostopic echo y pub. Es altamente recomendable que el usuario estudie los scripts llamados “Main Controller”, “Target Motor”, “Target Tank” y “Validation” presentes en la carpeta “lmev”. Y que tome estos como fuente de ejemplificación de las herramientas aquí explicadas y como base para sus futuros diseños programáticos.

Para ejecutar un script, se debe abrir una nueva terminal y ejecutar el siguiente código:

```
//rosrun lmev_gazebo “Nombre del script”.py
rosrun lmev_gazebo Validation.py
```

### 3.3 Otros elementos de interés

El usuario además de solo saber cómo controlar la plataforma robótica, es de utilidad saber manejar y entender algunos otros componentes presentes en el paquete “lmev”. En la dirección “/home/user/lmev/src/lmev_gazebo”, el usuario encontrará 7 carpetas de las cuales interesa el estudio de 4. Para ejemplificar el uso de estas carpetas la mejor forma es que el usuario estudie los pequeños códigos que contienen y entienda los comentarios adjuntos para rápidamente aprender a realizar las modificaciones que necesite.

#### 3.3.1 models

La carpeta llamada models es la carpeta donde se contienen todos los elementos tipo modelos descritos en el formato SDF en Gazebo y todos los archivos adicionales para la descripción de la misma. Por ejemplo, en esta carpeta se encuentra el modelo lmev, que corresponde a la descripción de la plataforma robótica Lego Mindstorm EV3 y todas las piezas tridimensionales construidas (archivos .dae). Note que en esta carpeta se puede

agregar cualquier otro modelo que el usuario desee incluir en la simulación y sus respectivos meshes. Estos modelos pueden ser guardados directamente desde la construcción gráfica de Gazebo sin necesidad de diseñar un código SDF. Es decir, aquí el usuario puede almacenar la descripción de cualquier otro modelo que considere necesario para su simulación.

#### 3.3.2 worlds

La carpeta llamada worlds es la carpeta donde se van a contener todos los elementos tipo mundo descritos en el formato SDF en Gazebo. Por ejemplo, en esta carpeta se encuentra el mundo lmev_world, que corresponde a la descripción del mundo de simulación básico. Note que en esta carpeta se podría agregar cualquier otro mundo que el usuario desee simular. Ahora, si se quiere simular el comportamiento de la plataforma robótica pero con una pista de carreras distinta o en el aula de trabajo, basta solo con cambiar el archivo .world utilizado. De esta manera, se puede realizar simulaciones con distintas condiciones de manera eficiente.

#### 3.3.3 launch

La carpeta llamada launch es la carpeta donde se van a contener instrucciones para la ejecución de distintos nodos de ROS. Por ejemplo en esta carpeta se encuentra el archivo lmev.launch, que se encarga de ejecutar un nodo cuya funcionalidad es inicializar Gazebo y cargar el mundo de simulación especificado, un otro nodo cuya funcionalidad es cargar en el mundo de simulación el modelo lmev en una posición especificada. Note que en esta carpeta se podría agregar cualquier otro archivo launch que ejecute diversos nodos en diverso orden, según como el usuario desee en su simulación.

#### 3.3.4 scripts

La carpeta llamada scripts es la carpeta donde se van a contener los algoritmos programáticos que describen un comportamiento de movimiento y sistema sensorial de la plataforma robótica ensamblada. Por ejemplo en esta carpeta el estudiante puede agregar archivos de extensión .py o .cpp (provenientes de Python o C++, respectivamente) que dentro de sus líneas de código hayan instrucciones de lectura de sensores, una lógica programática y luego instrucciones de control de actuadores de la plataforma robótica en función al algoritmo. Es decir, cada archivo de estos, funciona como un nodo individual en ROS. Prácticamente los archivos de esta carpeta se pueden comparar con los archivos que

se generan al diseñar un comportamiento del Lego Mindstorm EV3 en su software de programación oficial o de RobotC.

# 4 Ejemplos de uso

Para ejemplificar el uso y funcionalidades del modelo computacional de simulación para el Lego Mindstorms EV3, se pusieron en disposición cuatro comportamientos programáticamente diseñados. El estudiante, al estudiar y probar estos cuatro comportamientos, puede comprender cómo utilizar la simulación de una manera eficiente y así adaptarse a sus herramientas en el menor tiempo posible. La creación de estos ejemplos, se ve también como una oportunidad de desarrollar funcionalidades en la plataforma robótica, pues como se verá a continuación, son comportamientos comunes de la misma. En seguida, una guía de uso para los cuatro ejemplos.

Antes de iniciar, cabe rescatar que si desea terminar la simulación, la mejor forma de hacerlo es primero cerrando la ventana de Gazebo y luego cerrar la terminal en donde se encuentra corriendo roscore.

## 4.1 Main_Controller

Este algoritmo es el ejemplo más simple, escrito en el lenguaje Python. Ejemplifica la lectura de las mediciones de los sensores simulados y da una primera idea de cómo se diseña un comportamiento básico.

1. Entre al directorio “/home/user/lmev/src/lmev_gazebo/scripts” y abra el archivo Main_Controller.
2. Observe la estructura de la clase y de la función main().
3. Observe en la función Show_Measures() cómo se almacenan las mediciones de cada sensor.
4. Observe el otro método para realizar la lectura de los sensores.
5. Abra una terminal e inicialice la simulación y presione “Play”.

   ```
   cd lmev
   ./run_lmev.sh
   ```

6. En la tercera pestaña de la nueva terminal podrá ver las mediciones de cada sensor cada 2 segundos.

### 4.2 Target_Motor

Este algoritmo, escrito en el lenguaje Python, ejemplifica una forma alternativa de realizar la lectura de las mediciones de los sensores. Además, se ejemplifica como enviar órdenes al motor del brazo actuador. Note, que eventualmente este algoritmo habilita la funcionalidad de mover el brazo actuador a un ángulo específico. A partir de aquí, además de servir como ejemplo, el usuario dispone de dicho comportamiento como nueva herramienta en la simulación.

1. Entre al directorio "/home/user/lmev/src/lmev_gazebo/scripts" y abra el archivo Target_Motor.
2. Observe el flujo lógico de la programación. Ayúdese con los comentarios.
3. Note la línea 23 y 53. Estas son las encargadas de enviar las ordenes de movimiento al motor.
4. Note la línea 26. Aquí se crea un topic el cual el usuario puede acceder desde la terminal y especificar un ángulo deseado.
5. Abra una terminal, inicialice la simulación y presione "Play".
   ```
   cd lmev
   ./run_lmev.sh
   ```
6. En la misma terminal, ejecute el siguiente comando:
   ```
   rosrun lmev_gazebo Target_Motor.py
   ```
7. Abra una nueva terminal y ejecute el comando:
   ```
   rostopic list
   ```
   El usuario podrá notar la incorporación de un nuevo topic "/Arm_Motor/target_cmd".
8. Ejecute el siguiente comando y note lo que pasa con la plataforma robótica en simulación.
   ```
   rostopic pub /Arm_Motor/target_cmd std_msgs/Int16 "data: -40"
   ```
9. Cuando termine el movimiento, fíjese en la pestaña de mediciones de sensores y note el valor del sensor encoder del brazo.
10. Puede tratar con diferentes valores de ángulo o puede tratar de modificar la velocidad a la que el brazo actuador se mueve modificando la variable en el script.

### 4.3 Target_Tank

Al igual que el ejemplo anterior, el algoritmo Target_Tank está escrito en el lenguaje Python y no solo encuentra funcionalidad como ejemplo, si no que también permanece como futura herramienta disponible para el usuario. Este comportamiento permite especificar un número de rotaciones por realizar por las ruedas de la plataforma robótica. Se fortalece el control por terminal, la lectura y envío de datos, el uso del movimiento sincronizado de las ruedas y la creación de mensajes ROS.

1. Entre al directorio “/home/user/lmev/src/lmev_gazebo/scripts” y abra el archivo Target_Tank.
2. Observe el flujo lógico de la programación. Ayúdese con los comentarios.
3. En la línea 51 y 63 puede repasar como realizar la lectura de los sensores de manera simple.
4. La línea 24 inicializa el topic del actuador por usar. Y en la línea 60 y 67 se envía la orden de movimiento.
5. Note la línea 34. Aquí se crea de manera programática un mensaje tipo “Twist” y se inicializa su valor.
6. Abra una terminal, inicialice la simulación y presione “Play”.
   ```
   cd lmev
   ./run_lmev.sh
   ```
7. En la misma terminal, ejecute el siguiente comando:
   ```
   rosrun lmev_gazebo Target_Tank.py
   ```
8. Abra una nueva terminal y ejecute el comando:
   ```
   rostopic list
   ```
   El usuario podrá notar la incorporación de un nuevo topic “/Move_Tank_Rev/target_cmd”.
9. Ejecute el siguiente comando y note lo que pasa con la plataforma robótica en simulación.
   ```
   rostopic pub /Move_Tank_Rev/target_cmd std_msgs/Int16 "data: 10"
   ```
10. La plataforma robótica se moverá 10 revoluciones de las ruedas hacia adelante. Cuando termine el movimiento, fíjese en la pestaña de mediciones de sensores y note el valor del sensor encoder de las ruedas.
11. Puede tratar con diferente número de revoluciones, a una diferente velocidad de avance. ¿Cuánto debería la plataforma desplazarse hacia adelante? ¿Cuánto tiempo

debería tomarle completar esas revoluciones a esa velocidad? ¿Qué sucede si se modifica el mensaje Twist para ahora tener también velocidad angular?

### 4.4 Validation

Este último ejemplo pretende utilizar todas las funcionalidades básicas del modelo computacional desarrollado. Consiste en que la plataforma, al haber un nivel de luz ambiente alto, se mueva diez revoluciones hacia adelante. Luego, tomando en cuenta la medición del sensor giroscopio, girar noventa grados. Una vez ahí, lee la medición del sensor ultrasónico y si existe un objeto cerca de la plataforma entonces baja el brazo actuador. Este ejemplo pretende que el usuario se familiarice con lo que se podría considerar un diseño de algoritmo semejante al que tiene que implementar en las prácticas de clase, en donde utilice gran número de sensores, envíe diferentes órdenes a los motores y en dónde programe un comportamiento lógico deseado. Además, motiva al usuario a utilizar las herramientas de la interfaz gráfica de Gazebo.

1. Entre al directorio “/home/user/lmev/src/lmev_gazebo/scripts” y abra el archivo Target_Tank.
2. Observe el flujo lógico de la programación. Ayúdese con los comentarios.
3. Abra una terminal, inicialice la simulación y presione “Play”.
   ```
   cd lmev
   ./run_lmev.sh
   ```
4. En la misma terminal, ejecute el siguiente comando:
   ```
   rosrun lmev_gazebo Validation.py
   ```
5. Observe con atención que sucede en la plataforma robótica en simulación. Mire la medición del sensor de color en modo ambiente en la tercera pestaña.
6. En la ventana de Gazebo, inserte un cubo al mundo. Eso se logra presionando el símbolo de cubo en el panel superior de la interfaz gráfica y luego haga click en el mundo. Puede cambiar la ubicación del cubo con la herramienta de traslación en el mismo panel. Coloque el cubo aproximadamente a 2m en el eje x y 2m en el eje y. Se puede ayudar observando la “pose” del cubo en el panel izquierdo.
7. Ahora en la ventana de Gazebo, inserte una luz tipo “Point Light”. Esta se encuentra en el panel superior, es un dibujo semejante a un sol. Presiónelo e haga click en algún sitio cercano a la plataforma robótica y frente a ella. Observe con atención. (Este tipo de iluminación es conveniente para un comportamiento de buscador de luz).

8. Cuando el movimiento haya terminado. Puede repetir los pasos del 1 al 7, pero ahora inserte el cubo a una distancia más cercana, por ejemplo 1m en el eje x y 2m en el eje y. ¿Ahora qué sucede cuando la plataforma robótica gira?
9. El usuario puede ahora experimentar a gusto, modificando las variables en el algoritmo y modificando dinámicamente los modelos en la interfaz gráfica de Gazebo.