ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

DISEÑO, CONTROL Y SIMULACIÓN, MEDIANTE EL USO DEL ENTORNO ROS/GAZEBO, DE UN SISTEMA MULTI-AGENTE CONFORMADO POR PLATAFORMAS TURTLEBOT3 ENFOCADO A MANTENER UNA FORMACIÓN LÍDER SEGUIDOR (TOMO 1)

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN ELECTRÓNICA Y AUTOMATIZACIÓN

RODRIGO ALEJANDRO AYALA PAVÓN

rodrigo.ayala@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

patricio.cruz@epn.edu.ec

DMQ, febrero 2022

CERTIFICACIONES

Yo, Rodrigo Alejandro Ayala Pavón declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

RODRIGO ALEJANDRO AYALA PAVÓN

Certifico que el presente trabajo de integración curricular fue desarrollado por Rodrigo Alejandro Ayala Pavón, bajo mi supervisión.

Ing. PATRICIO JAVIER CRUZ DÁVALOS, PhD.
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como los productos resultantes del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Rodrigo Alejandro Ayala Pavón

Nathaly Gabriela Romero Calle (Colaboradora)

Ing. Patricio Javier Cruz Dávalos, PhD.

DEDICATORIA

El trabajo, esfuerzo y perseverancia que ha implicado el desarrollo de este trabajo va dedicado a:

A mis padres que siempre supieron apoyarme en mi trayecto académico y me han educado con el ejemplo de esfuerzo y gallardía. Gracias a ellos he podido alcanzar una meta más.

A mis hermanos, por su cariño y por todas las enseñanzas que han sabido darme y por siempre traer alegría a mi vida.

Finalmente, quiero dedicar esta tesis a Gabriela por su amor incondicional y a todos esos compañeros que nunca me abandonaron ni en los momentos más difíciles.

AGRADECIMIENTO

Mi más grande agradecimiento a la Escuela Politécnica Nacional por enseñarme que para alcanzar la meta se debe persistir y aprender de los fracasos para ser mejores.

A mi tutor Ing. Patricio Cruz, PhD. y al Ing. Diego Maldonado, Mgs. por el conocimiento y la guía que me han sabido brindar a lo largo de todo este trabajo.

Finalmente, quiero agradecer al Team ROS por hacer de este proceso algo gratificante y enseñarme que “Si quieres llegar rápido ve solo, pero si quieres llegar lejos, ve acompañado”

ÍNDICE DE CONTENIDO

RESUMEN

Las innovaciones en el área de la tecnología han desembocado en el crecimiento y desarrollo que está teniendo la robótica en la actualidad. Uno de los campos que más destaca es la robótica móvil, caracterizada por el movimiento autónomo de plataformas que cuenta con la suficiente inteligencia para reaccionar y tomar decisiones en función de la información que perciben del ambiente que los rodea. Justamente uno de los más utilizados es el robot móvil de tracción diferencial por las facilidades, especialmente en cuanto a su controlabilidad. Por esta razón, el presente trabajo emplea la versión virtual de la plataforma comercial TurtleBot3, ya que su comportamiento se lo puede simular y comprobar en los softwares de código abierto ROS y Gazebo.

Es así como la aplicación desarrollada se centra en simular un entorno libre de obstáculos para el manejo por un usuario de un TurtleBot3 virtual. El cual pueda escoger entre las dos versiones disponibles de este robot, así también como su modo de operación, ya sea mediante teclado del computador o por medio de la aplicación ROS Mobile. También se emplea el paquete turtlebot3_autorace_2020, del cual se utiliza el modo de conducción autónoma del robot y su pista para realizar diferentes pruebas con el robot virtual y las opciones de operación. De modo que, el entorno diseñado y las opciones de movimiento de las dos versiones del Turtlebot3 virtual son un aporte necesario para proceder con el segundo tomo del trabajo desarrollado en conjunto.

PALABRAS CLAVE: TurtleBot3, ROS Mobile, Movimiento por teclado, ROS, Gazebo

ABSTRACT

Technological innovations have led to the growth and development of robotics. One field that stands out the most is mobile robotics. It is characterized by the autonomous movement of platforms that have enough intelligence to react and take actions based on the information they perceive from their surrounding environment. In particular, one of the most popular platforms is the differential drive mobile robot principally because of its controllability. For this reason, this work uses the virtual version of the commercial platform TurtleBot3 since its behavior can be simulated in the open source software ROS and Gazebo.

Thus, this application focuses on simulating an obstacle-free environment for the command by a user of a virtual TutleBot3. The user can choose between the two available versions of the robot, as well as its operation mode either by computer keyboard or through the application ROS Mobile. The turtlebot3_autorace_2020 package is also used to take advantage of its autonomous driving mode and the developed track to perform different tests with the virtual robot and the implemented operation options. Therefore, the designed environment and the movement options of a both versions of the TurtleBot3 are a fundamental contribution to proceed with the second volume of the total project developed together.

KEYWORDS: TurtleBot3, ROS Mobile, Keypad Commanded Motion, ROS, Gazebo

1 INTRODUCCIÓN

Este documento es el primer tomo del trabajo desarrollado en conjunto por los autores Rodrigo Ayala y Gabriela Romero. El mismo es la base para lo detallado en el segundo tomo [1], por lo que se sugiere revisarlo para comprender de mejor manera lo desarrollado como parte del Proyecto de Trabajo de Integración Curricular conformado por ambos tomos. Esto es respecto al entorno diseñado y las opciones de movimiento implementadas para un robot virtual TurtleBot3 en los softwares ROS y Gazebo.

La robótica tiene varios antecedentes históricos que abarcan desde sistemas hidráulicos empleados por la cultura griega hasta historias de ficción de Isaac Asimov [2]. Hoy en la actualidad un mundo sin robótica sería inimaginable por todos los beneficios que representa no solo para la industria, medicina o milicia, sino también para actividades de la vida cotidiana. Los robots, en cualquier área que sean utilizados tienen como finalidad ofrecer mayor productividad, aumentar flexibilidad, mejorar el cumplimiento de tareas y reducción de precios. Aunque esto último, puede variar dependiendo del tipo de robot que se desee emplear.

Existen robots manipuladores como brazos robóticos, generalmente utilizados en producción industrial o en medicina con múltiples brazos robóticos (también llamados robots antropomórficos). De manera similar, existen robots móviles, que se subdividen según su movimiento, es decir, acorde a la forma en la que se desplazan. Puede ser mediante ruedas (robots tipo diferencial, tipo carro, etc.), patas (robots zoomórficos), alas (robots aéreos) y finalmente los robots humanoides o androides [3].

Respecto a los robots móviles que emplean ruedas, el desafío que representa controlarlos se evidencia precisamente en su movimiento. Si bien se pueden realizar diferentes controladores de trayectoria o posición, el presente trabajo se centra en permitir a un usuario mover libremente a un robot en un entorno libre de obstáculos previamente diseñado. En particular, se hace uso de robots móviles de tracción diferencial por las facilidades que presenta en cuanto a su controlabilidad, basado en su modelo matemático.

En términos generales, para el desarrollo de una aplicación robótica es necesario utilizar un software especializado, como lo es ROS (Robot Operating System) [45]. Este software es una plataforma de código abierto que posee varias librerías y herramientas que facilitan la comunicación a través de un sistema llamado nodos y tópicos. Además, se lo emplea junto con Gazebo, el cual al ser un simulador de robótica 3D de código abierto que permite diseñar entornos con todas las condiciones físicas de un mundo real, crear robots desde cero o integrar modelos comerciales, entre otras opciones.

De modo que este trabajo de titulación hace uso de los softwares antes mencionados junto con la plataforma TurtleBot3 [54], ya que se encuentra respaldada por ROS y se usa en varios campos educativos y de investigación por ser un robot móvil de tracción diferencial de fácil programación. Esto implica una integración sencilla con Gazebo, para lograr que un usuario pueda comandar su movimiento.

1.1 OBJETIVO GENERAL

Cabe indicar que, dado que este es el primer tomo del trabajo desarrollado, se presenta el objetivo general acorde al plan aprobado, pero enfocado a lo detallado en este tomo y el mismo es: Desarrollar y simular utilizando ROS/Gazebo un entorno libre de obstáculos para el control de la plataforma TurtleBot3 por un usuario.

1.2 OBJETIVOS ESPECÍFICOS

Cabe indicar que, dado que este es el primer tomo del trabajo desarrollado, se presentan los objetivos específicos acorde al plan aprobado, pero enfocados a lo detallado en este tomo.

- Investigar la utilización de la plataforma ROS y simulación en Gazebo para el control del robot virtual TurtleBot3 por parte de un usuario.
- Implementar un entorno libre de obstáculos para el robot virtual TurtleBot3 a través de las herramientas ROS y Gazebo.
- Implementar la programación necesaria para que un usuario pueda dirigir al robot virtual TurtleBot3.
- Comprobar el funcionamiento del control del robot virtual TurtleBot3 por parte de un usuario, sobre todo considerando acciones bruscas que pueden ser comandadas por el mismo.

1.3 ALCANCE

Cabe indicar que, dado que este es el primer tomo del trabajo desarrollado, se presentan los alcances acorde al plan aprobado que están relacionados con lo detallado en este tomo.

- Se realiza un estudio del robot de tracción diferencial TurtleBot3 donde se deberá tomar en cuenta sus restricciones no holonómicas y características principales para establecer un control que envíe directamente la velocidad lineal y angular al robot.
- Se revisa el entorno ROS en cuanto a su estructura y comunicación entre publicadores y subscriptores mediante nodos-tópicos y su conexión con Gazebo para la simulación de entornos integrando el robot TurtleBot3 para entender su funcionamiento y aprovechar sus herramientas.
- Se diseña un entorno de trabajo libre de obstáculos utilizando el software Gazebo que permita simular las condiciones físicas del robot TurtleBot3 con el fin de verificar el movimiento del robot.
- Se realiza pruebas del manejo de la plataforma TurtleBot3 por parte de un usuario en el entorno de Gazebo desarrollado.

1.4 MARCO TEÓRICO

1.4.1 ROBOTS MÓVILES TERRESTRES

Actualmente, la robótica móvil es uno de los campos de investigación científica que más está creciendo. Debido a sus habilidades, los robots móviles pueden substituir a los humanos en muchos campos, como por ejemplo en aplicaciones de vigilancia, operaciones de rescate de emergencia, patrullaje, reconocimiento, automatización industrial, construcción, entretenimiento, guías en museos, servicios personales, intervención en ambientes extremos, asistencia médica, entre otras [6].

Este tipo de robots cuentan con una plataforma mecánica que posee un sistema de locomoción, lo que le permite navegar a través de un ambiente de trabajo establecido, además de dotarles de cierto nivel de autonomía para trasladarse. La autonomía no se refiere únicamente a cuestiones energéticas sino también a la capacidad del robot de planificar, sensar y ejecutar acciones necesarias para cumplir con sus objetivos, con poca o sin nada de participación de un ser humano. Por lo que, generalmente, cada robot ocupa su propio sistema de control para que se puedan desenvolver ya sea en ambientes estructurados o no estructurados [7].

Los robots móviles tienen diferentes características que les permite tener una amplia clasificación. Si se analiza el sistema de locomoción, un robot móvil terrestre, puede moverse con: patas, ruedas, alas o cadenas. Por otra parte, si se investiga acerca de la

disposición de sus ruedas se tendrá una vasta clasificación que dependerá del número de ruedas y su disposición en el robot [8].

A lo largo de este trabajo se desarrolla el estudio y control del movimiento de robots móviles terrestres con sistema de locomoción y disposición de tres ruedas, dos de tracción diferencial y una tercera rueda como punto de contacto. Por lo que el tipo de robot que se va a detallar en la siguiente sección corresponde a un robot móvil de tracción diferencial.

1.4.1.1 Robot Móvil de Tracción Diferencial

La tracción diferencial o tracción y dirección en un mismo eje se logra con motores independientes en las ruedas de un mismo eje, y "ruedas locas" en el resto de los ejes, como se observa en la Figura 1.1. Este modelo representa una construcción sencilla, pero se limita a radios de giro del tamaño del vehículo. Además, para que el control de este tipo de robots sea simple, los motores deben ser de características idénticas [9].

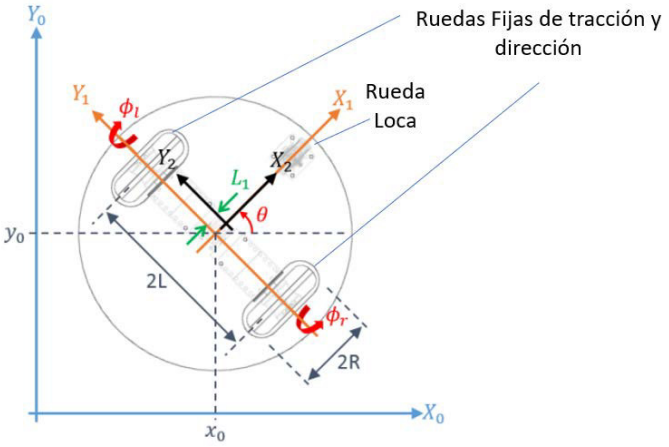


Figura 1.1 Esquema robot de tracción diferencial [9]

En la Figura 1.1 se observa el esquema de un robot móvil de tracción diferencial con su sistema de coordenadas $\sum_0(X_0, Y_0)$, $\sum_1(X_1, Y_1)$ y $\sum_2(X_2, Y_2)$ que facilitan percibir la orientación y el desplazamiento del robot móvil. Además, se definen los parámetros:

- $2L$: representa la distancia de separación entre las dos ruedas de tracción.
- $2R$: es el diámetro de las ruedas.
- L_1: la distancia del centro de masa al eje de rotación de las ruedas.

Restricción No holonómica

De igual importancia, se debe analizar si el robot móvil es o no holonómico. Un robot holonómico es capaz de desplazarse en cualquier dirección, mientras que, en un robot no

holonómico su movimiento se verá restringido por los grados de libertad controlables que posea. Un robot móvil de tracción diferencial posee un total de 3 grados de libertad para describir su posición y movimiento sobre un plano, estos están expresados por la posición en (x, y) y el ángulo de rotación θ [9]. Por otro lado, los grados de libertad controlables hace referencia a los grados de libertad dados las ruedas o elementos de locomoción, por esto, para el robot de tracción diferencial se tienen dos grados de libertad controlables. Dicho brevemente, la restricción no holonómica se refiere al movimiento que un robot no puede hacer. En el caso del robot de tracción diferencial, se desplazará solo adelante o atrás mas no de forma lateral, como se observa en la Figura 1.2 [10].

Figura 1.2 Representación de restricción no-holonómica de robots móviles de tracción diferencial [Fuente propia]

Revisados los componentes del esquema del robot móvil de tracción diferencial y su restricción no holonómica, es necesario ahora detallar la cinemática de esta plataforma. En la siguiente sección se resume su desarrollo matemático.

1.4.1.2 Cinemática Del Robot Móvil De Tracción Diferencial

La cinemática estudia el movimiento de sistemas mecánicos sin considerar las fuerzas que afecten dicho movimiento. Para el modelado del robot móvil de tracción diferencial, se representa la velocidad del robot en función de la velocidad de sus ruedas y sus parámetros geométricos [11]. El análisis se lo realiza en función de la Figura 1.2, fijando un sistema de referencia fijo $\sum_0(X_0, Y_0)$, seguido a esto se forma otro sistema $\sum_1(X_1, Y_1)$, atado al robot, para hallar los componentes de traslación y la orientación del robot móvil respecto al sistema de referencias $\sum_0$. En base a esto, considerando un robot móvil de tracción diferencial no holonómico que se mueve en una superficie plana horizontal, sus ecuaciones cinemáticas se pueden describir como:

$$\begin{aligned} \dot{x}_0 &= v\cos(\theta) \\ \dot{y}_0 &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{1. 1}$$

Donde $p = [x_0, y_0]^T$ es la posición del robot, θ corresponde a la orientación del robot, v y ω representan la velocidad lineal y angular, respectivamente.

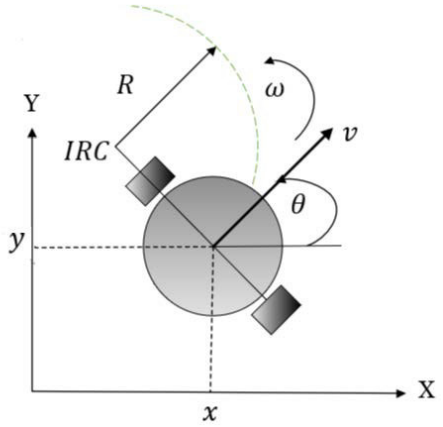


Figura 1.3. Representación de IRC en robot móvil de tracción diferencial [Fuente propia]

Teniendo en cuenta las características de un robot móvil de tracción diferencial y su modelo matemático, se debe notar que existe una rigidez entre las dos ruedas generando movimientos pequeños como: traslaciones longitudinales y rotaciones alrededor del punto central del eje de las ruedas. Sin embargo, es posible desplazar al robot de mejor manera al combinar los movimientos de traslación y rotación. De esta forma, se genera el Centro de Rotación Instantáneo (IRC), representado en la Ecuación (1.2), que permite rotar al robot en cualquier punto que esté en la línea del eje de las ruedas, como se muestra en la Figura 1.3. Por consiguiente, pese a sus restricciones no holonómicas, es posible emplearlo para diferentes aplicaciones, por su simplicidad en cuanto a número de ruedas y el control que se puede desarrollar para su desplazamiento [12].

$$IRC = \big(x - Rsin(\theta), y + Rcos(\theta)\big) \tag{1.2}$$

Hasta el momento se ha revisado sobre los tipos de robots móviles en función de la disposición de las ruedas, el modelo cinemático de un robot de tracción diferencial y sus restricciones no holonómicas. Se considera ahora oportuno, detallar sobre uno de los robots comerciales con las características anteriormente mencionadas, por lo que se detalla acerca de la plataforma TurtleBot.

1.4.2 PLATAFORMA TURTLEBOT

TurtleBot fue creado en 2010 en Willow Garage por Melonee Wise y Tully Foote. La plataforma como tal es un kit robótico que se encuentra respaldado por Robot Operating

System (ROS) y Point Cloud Library (PCL), esta última es una librería de código abierto con algoritmos que ayudan en el procesamiento de geometría tridimensional como en la visión artificial [13]. La plataforma TurtleBot se desarrolla con el propósito de ser fácil no sólo de utilizar, sino de construir y ensamblar ya que el hardware de montaje posee una base móvil con un sensor de distancia 2D / 3D y un SBC (single board computer). Además, permite acoplar diferentes sensores extras en caso de ser necesario.

Los robots TurtleBot han evolucionado tanto que han pasado por tres generaciones o familias. En un inicio, 2010, se tenía al TurtleBot original que consistía básicamente en una batería, un giroscopio y un sensor Kinect, contaba también con la posibilidad de añadir más sensores, pero hoy en día se encuentra descontinuado. Después en 2012, aparece la familia del robot TurtleBot2 con cuatro modelos que están representados en la Figura 1.4. Esta familia se caracteriza por poseer una batería que requiere de menos corriente que la del robot original. Finalmente, ya en 2017 se encuentra la familia TurtleBot3 que cuenta con tres placas modulares: Burger, Waffle y Waffle Pi. Su mejora radica en la personalización del robot, reducir aún más la corriente de las baterías, implementar un LIDAR de 360 grados, una Raspberry, entre otras características [14].

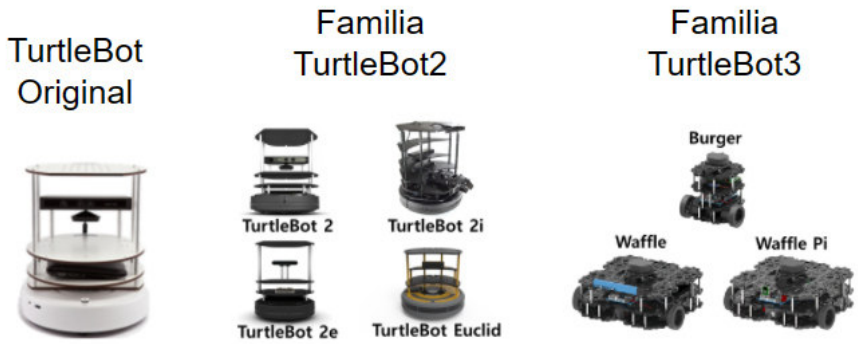


Figura 1.4. Evolución de plataforma TurtleBot [14].

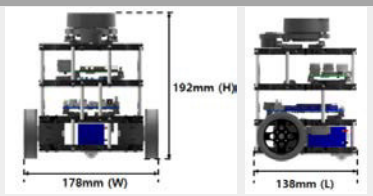
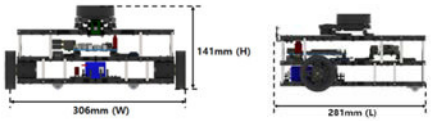
En el presente proyecto se trabaja con la última versión de la plataforma cuyas características de detallan con mayor profundidad en la siguiente sección.

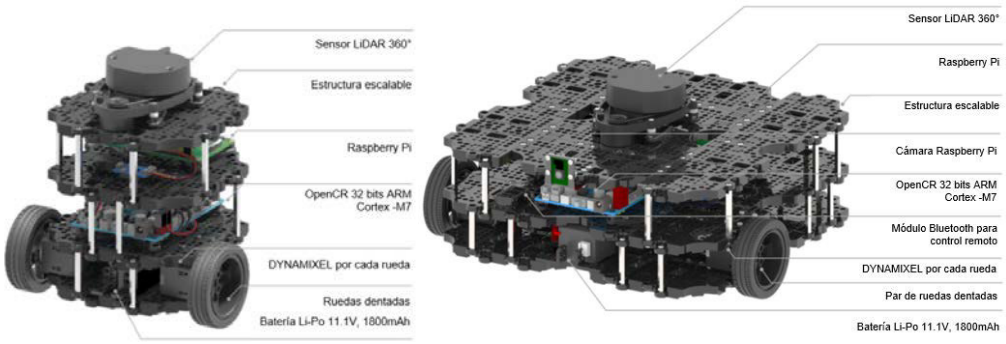
1.4.2.1 Modelos TurtleBot3

Para este proyecto se emplea la plataforma TurtleBot3 en sus modelos Burger y Waffle pi, ya que, en el segundo tomo, se los utilizará para realizar la formación líder seguidor [1]. En la Tabla 1.1 se encuentran las dimensiones físicas de cada modelo. De esta forma se puede notar que el modelo Waffle Pi es más ancho y bajo, mientras que el modelo Burger es más angosto y alto. En la Figura 1.5 se pueden observar los modelos de los robots junto

con sus principales características. Para ambos modelos, su cara frontal es la que está más próxima a las llantas del robot [15].

Tabla 1.1. Dimensiones de los modelos TurtleBot3 (modificado de [15]).

Modelo	Dimensiones	Imagen
Burger	Largo: 138 [mm] Ancho: 178 [mm] Altura: 192 [mm] Peso: 1 [Kg]	
Waffle Pi	Largo: 281 [mm] Ancho: 306 [mm] Altura: 141 [mm] Peso: 1.8 [Kg]	



(a) (b)

Figura 1.5. (a) modelo Burger (b) modelo Waffle pi [Fuente propia]

Características principales:

- **LiDAR 360 (LDS - 01):** es un scanner láser de 360° que está montado en la parte superior del robot. Este sensor transmite un láser rotativo, el cual se refleja al chocar con obstáculos cercanos. Usando los tiempos de reflexión se puede determinar la presencia de un obstáculo.

- **Raspberry Pi**: el TurtleBot3 cuenta con una computadora de placa única Raspberry Pi 3. La cual, desde su introducción ha ganado una popularidad increíble en investigación debido a su tamaño reducido, bajo costo y uso de energía. Esto es muy útil en aplicaciones que requieren una computadora, pero no una gran cantidad de poder de procesamiento. En el TurtleBot3, la Raspberry se encarga de leer la

información de los sensores y se comunica con una computadora “master” la cual hace el verdadero procesamiento.

- **OpenCR**: debajo de la Raspberry Pi, el hardware OpenCR es una tarjeta de control basada en Arduino Uno. La tarjeta contiene una unidad de medición inercial (IMU) con un acelerómetro de tres ejes, giroscopio y magnetómetro. Su función principal es conectar a la Raspberry Pi con los motores y sensores. Adicionalmente, la tarjeta provee conexiones de energía a todos los componentes [40].

- En el nivel inferior, se encuentran los motores Dynamixel X series y la batería LiPo de 11.1 V de litio. Tanto el modelo Burger como el Waffle pi tienen una configuración diferencial, lo que significa que cuentan con 2 servomotores independientes, uno en cada rueda y una rueda caster para su estabilidad.

- **SOFTWARE:** el sistema operativo con el que cuenta la Raspberry Pi es Ubuntu MATE 16.04, la cual corresponde a una versión más ligera de la distribución Ubuntu de Linux. [16]

El robot móvil de tracción diferencial TurtleBot3 ha llegado a mejorar tanto que puede ejecutar algoritmos SLAM para localización, mapeo de entornos y también existe la posibilidad de poder controlarlo de forma remota mediante una laptop, un joystick o un teléfono celular con sistema operativo Android.

Todas estas aplicaciones forman parte de las ventajas de emplear esta plataforma robótica, pero el beneficio de más impacto para el presente trabajo es la facilidad con la que se puede incorporar, simular y controlar a través de los softwares de ROS y Gazebo.

1.4.3 ROS

ROS o Robot Operating System es un framework de código abierto orientado al desarrollo de aplicaciones en robótica y ofrece herramientas y librerías para el desarrollo de estos. Su éxito está basado en su filosofía que no es inventar la rueda y se apoya en otros proyectos como pueden ser OpenCV, Python, Gazebo, etc [17], [18].

Entre sus principales características se destaca su capacidad de conectar el sistema operativo con los dispositivos hardware; así como también, el control de dispositivos a bajo nivel, la implementación de funciones comúnmente usadas y su sistema de mensajería entre procesos [18]. Todas las capacidades mencionadas anteriormente hacen de ROS una plataforma que permite el desarrollo de una gran cantidad de aplicaciones. Extendiendo así su uso al campo de la investigación, productos comerciales, educación y

como centro de entretenimiento tanto en el sector académico como en el sector privado [45].

En sus inicios, ROS comenzó bajo el nombre de Switchyard en el año de 2007 y su popularidad ha provocado el desarrollo de varias distribuciones. Una distribución es un conjunto de paquetes que tienen diferentes versiones, como lo es, por ejemplo, Linux con la distribución Ubuntu. Cada distribución de ROS se lanza al público siguiendo tres reglas [19]:

1. El tiempo de lanzamiento se basa en los recursos disponibles y su necesidad.
2. Todas las versiones de ROS1 son de largo plazo por lo que se da soporte para 5 años.
3. Las versiones de ROS deberán dejar de prestar soporte para la distribución de EOL (End of Life) de Ubuntu

A lo largo de los años ROS1 ha venido desarrollando 13 distribuciones, por lo que el nuevo proyecto al que se apunta es ROS2 en donde el objetivo principal es expandirse a demás sistemas operativos y ser compatible con Ubuntu Xenial, OS X El Capitan y Windows 10. Las distribuciones más recientes son: ROS Morenia Melódica, ROS Tortuga boba lunar y ROS Noetic Ninjemys [13], De manera que ROS cuenta con una comunidad global donde los desarrolladores y usuarios contribuyen para mejorar el software y no requiere de ningún licenciamiento de alguna empresa comercial, como es el caso de MATLAB o LabVIEW [20].

1.4.3.1 Comunicación

ROS utiliza una topología de red peer-to-peer. Al ejecutar ROS los sistemas pueden estar formados por varios procesos, mejor conocidos como nodos. Los nodos se encargan de realizar el cálculo del sistema y se comunican entre sí, intercambiando mensajes. Estos están conformados como estructuras que almacenan datos tipificados. La comunicación entre nodos puede ser de dos formas:

- **Sincrónica:** este tipo de comunicación se la conoce también como servicio. Los servicios son muy parecidos a las funciones que se emplean en los lenguajes de programación tradicionales. Un servicio se define mediante un nombre (tipo string) y un par de mensajes: solicitud y respuesta. Sólo un nodo puede proporcionar un servicio de un nombre específico [21].

- **Asíncrona:** este tipo de comunicación es más conocido por el nombre tópico. Como se puede observar en la Figura 1.6, los tópicos son flujos de datos publicados por un nodo, denominado publicador, de forma que otros nodos, llamados oyentes (listeners) o subscriptores, se pueden suscribir a este tópico. Contrario a los servicios, los nodos oyentes no se pueden suscribir al tópico para comunicarse con el nodo publicador (Publisher). En la comunicación asíncrona varios nodos pueden ser publicadores o suscriptores simultáneamente a un mismo tópico.

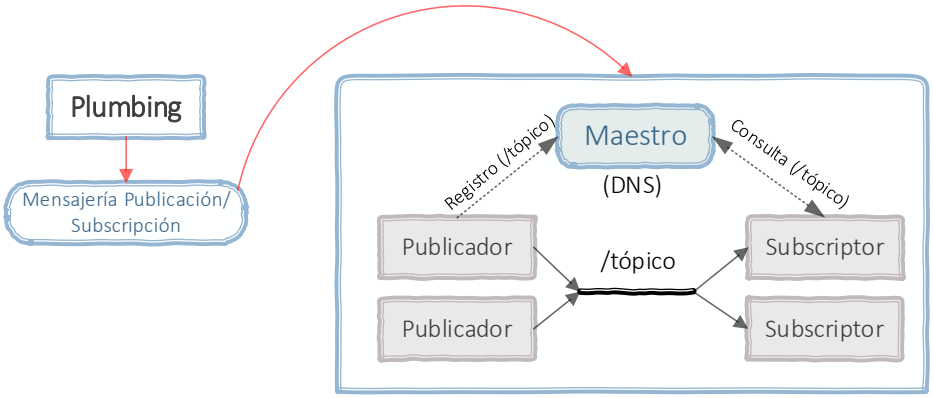


Figura 1.6. Estructura de mensajería de ROS [17]

ROS es mucho más que un conjunto de librerías que sólo proporcionan un mecanismo y un protocolo de comunicación. Los nodos se desarrollan mediante un sistema de construcción elaborado sobre la plataforma de código abierto CMake que ayuda a crear y controlar la ejecución y la creación de archivos como paquetes, tan característicos de ROS.

En resumen, la finalidad que presta ROS es facilitar el desarrollo de múltiples robots de diferentes categorías mediante la simulación de navegación, mapeo y visión computacional todo implementado en un mismo ambiente, haciendo uso de la integración de librerías que posee como OpenCV o Gazebo. Estas librerías como tal permiten que el intercambio de información sea más sencillo. Como lo es ROS con Gazebo ya que concede la simulación de todo un sistema robótico, con diferentes opciones para desarrollar todo tipo de ambientes, incorporar modelos de robots comerciales, etc.

1.4.4 GAZEBO

Los simuladores físicos permiten el desarrollo de la gran mayoría de la investigación de la robótica. Estos simuladores proveen un ambiente que permite a los usuarios acceder a una gran variedad de robots sin el peligro potencial de dañarlos o romperlos. En determinados casos, la simulación podría correr más rápido que en la vida real, lo cual es especialmente

importante para enfoques basados en el aprendizaje; además, no se requiere atención física para reestablecer un entorno [22].

Uno de los simuladores físicos más popular por su interacción con ROS, es Gazebo. El cual, está en la capacidad de simular robots móviles terrestres, manipuladores e incluso robots aéreos, permitiendo así, el desarrollo de un extenso número de aplicaciones [22]. Gazebo es un simulador 3D orientado a sistemas multi-robots, el cual, tiene la capacidad de emular las características dinámicas y cinemáticas de cada robot, así como el de sus sensores. Además, permite la simulación de objetos físicamente plausibles que conforman el ambiente de simulación [23].

El ambiente de simulación en Gazebo está potenciado por un motor de gráficas robusto y de alta calidad. Basa su arquitectura en el motor de simulación ODE (Open Dynamics Engine) lo que permite representar entornos realistas (por ejemplo: texturas de alta calidad, sombras e iluminación) [24]. Además, al ser un software de código abierto es gratis y existe una gran comunidad que brinda soporte para el desarrollo de aplicaciones sobre esta plataforma [45].

El carácter colaborativo y universalista de ROS-Gazebo, a través de su paquete *gazebo_ros_pkgs* integrado a ROS en el 2009 con la simulación PR2 por Jhon Hsu, contribuye a la popularidad de este simulador y simplifica el proceso de probar el sistema de control a través de simulación y transferirlo a un sistema físico. Así también, Gazebo ofrece la capacidad de importar modelos de los archivos de la Universal Robot Description Format, como se observa en la Figura 1.7. Sin embargo, es importante considerar que Gazebo en sí no brinda la funcionalidad de planificación de movimiento. En su lugar, su estrecha integración con ROS permite el uso de los planificadores de caminos desarrollados en ROS [22].

Figura 1.7. Formación de robots esquema líder seguidor [Fuente propia]

Es necesario notar que la organización Open Source Robotics Foundation (OSRF) apoya el desarrollo de software libre para usarlo dentro de la robótica, ya sea en educación, investigación o producción. OSRF es una organización sin fines de lucro donde sus dos proyectos principales son ROS y Gazebo, mismos que son herramientas principales para desarrollar este trabajo.

Finalmente es relevante mencionar que tanto ROS como Gazebo corren sobre el sistema operativo Linux dentro del cual es común la utilización de archivos bash por sus siglas en inglés (Bourne-again Shell). Este es un lenguaje utilizado en la plataforma Linux para la ejecución de comandos que causan acciones dentro del terminal. Es importante el conocimiento y uso de estos archivos ya que permite al usuario automatizar tareas repetitivas en Linux, facilitando la ejecución de aplicaciones por parte del usuario, ahorrando tiempo y esfuerzo. Para su funcionamiento se debe crear un 'script' el cual es un archivo de texto con extensión .sh donde se escriben los comandos que se van a ejecutar de forma secuencial y al correrlo, realizar todas las instrucciones especificadas. [25]

2 METODOLOGÍA

En este capítulo se presenta la información sobre el desarrollo del entorno virtual, en cuanto disposición de carpetas y archivos en ROS y Gazebo, así también como datos importantes sobre la plataforma TurtleBot3 y cómo se usa al robot de manera de virtual. También, se describe el modo de operación del robot a través del teclado del computador y de otra forma alternativa de comandar el movimiento del robot a través del uso de una aplicación móvil llamada ROS Mobile. De modo que todo este trabajo sirve como base para expandir el manejo al de varios robots, que conformarán el sistema multi-agente del segundo tomo [1] del trabajo de integración curricular desarrollado en conjunto.

2.1 ENTORNO VIRTUAL

El entorno virtual se desarrolla dentro del software ROS, en su última distribución número 13 llamada Noetic Ninjemys y el programa Gazebo con la versión 11. Estos a su vez se ejecutan dentro del sistema operativo Linux Ubuntu con la versión 20.04. Se recomienda revisar el Anexo I, para tener más detalles sobre la instalación y tutoriales sugeridos sobre estos programas.

2.1.1 COMPONENTE EN ROS

El primer paso para el desarrollo del presente trabajo en ROS es entender su arquitectura, la cual está conformada por tres niveles: nivel de Filesystem (sistema de archivos), nivel de computation graph (gráfico de cálculo) y nivel de comunidad [15]. Estos niveles se encuentran detallados en el Anexo II y son importantes para entender la organización y distribución de archivos dentro del espacio de trabajo de ROS.

2.1.1.1 Organización de archivos

El espacio de trabajo o ROS Workspace corresponde a una carpeta la cual contiene el código ROS y su creación se detalla en los ROS Tutorials [26]. Cuando es creada, dentro de ella se generan tres carpetas: src, build y devel. Tal como se puede observar en la Figura 2.1 y cuyo contenido se describe en el Anexo III.

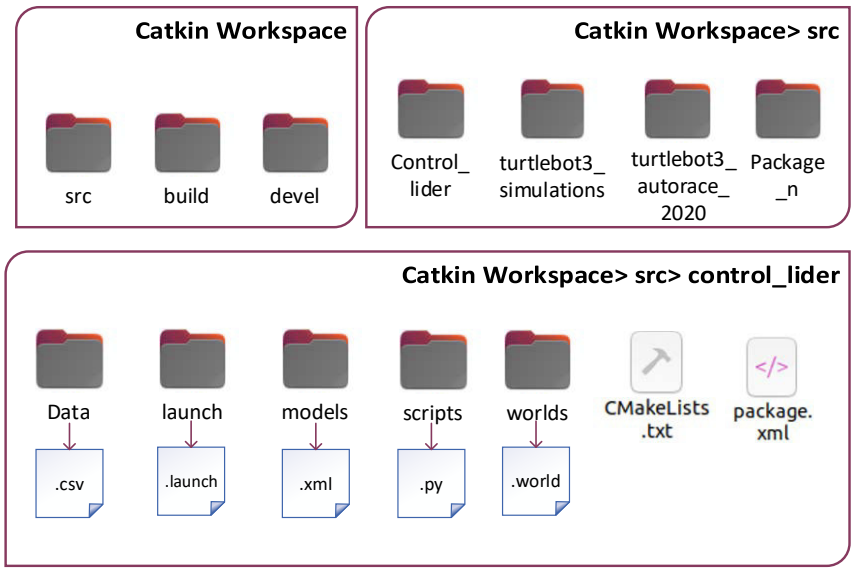


Figura 2.1 Espacio de trabajo de ROS [Fuente propia]

En la Figura 2.1 se observa el espacio de trabajo que se ha creado para el actual proyecto, en cual se encuentra el paquete control_lider construido desde cero y los paquetes de uso público turtlebot3_simulations y turtlebot3_autorace_2020 instalados desde los tutoriales de la página web ROBOTIS e manual [16]. En el paquete creado para el desarrollo de este primer trabajo se encuentran diferentes carpetas dispuestas de la siguiente manera.

- **Data:** se guardan los datos de la posición y orientación del TurtleBot3 mismos que son utilizados en la sección de resultados para obtener las gráficas de trayectoria de los modelos del TurtleBot3.
- **Launch**: contiene archivos que permiten lanzar múltiples nodos y configurar el servidor de parámetros. Una mayor explicación sobre los archivos .launch creados se encuentran en la Sección 2.1.3

- **Models**: contiene archivos .config y .sdf que describen los modelos de elementos físicos desde una pared hasta un robot que son utilizados dentro del entorno de simulación.
- **Scripts**: son archivos Python que contienen en este caso el código para controlar al TurtleBot3 a través de teclado.
- **Worlds**: contiene los archivos .world que describen las características y configuraciones de la pista de Autorace que se emplea en la Sección 2.2.3, para realizar diferentes pruebas respecto al manejo del robot y también se encuentra el entorno virtual desarrollado para el segundo tomo del trabajo en conjunto.

Toda la documentación, paquetes y scripts desarrollados tanto para este tomo como para el siguiente se pueden encontrar en la plataforma GitHub [27]. Para más detalles por favor revisar el Anexo IV.

Como se explicó en la Sección 1.4.3.1, dentro de ROS la comunicación se puede dar de forma síncrona o asíncrona, siendo la primera la que se escogió para el desarrollo de la aplicación presentada tanto en el primero como en el segundo tomo del trabajo en conjunto [1]. Así, en la siguiente sección se revisa más a fondo la comunicación asíncrona o comunicación por tópicos en ROS, pero enfocada desde lo requerido en el presente proyecto.

2.1.1.2 Nodos y tópicos empleados

ROS fue concebido para trabajar en un entorno distribuido. Para el intercambio de datos se tiene las siguientes opciones: tópicos, servicios y acciones, como se observa en la Figura 2.2. Tanto en este como en el siguiente tomo el intercambio de datos entre nodos se realiza a través de tópicos, puesto que al hacerlo de esta forma no se requiere de una configuración adicional en el archivo CMakeLists.txt, como se explica en el Anexo III. Para conocer cómo realizar el intercambio de datos se recomienda revisar los tutoriales de la página oficial de ROS [26].

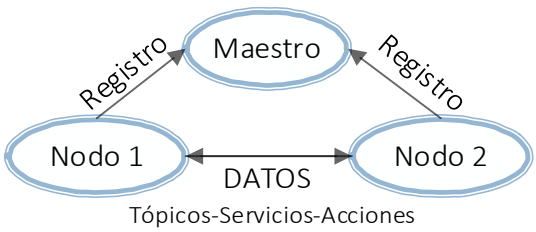


Figura 2.2. Red ROS [Fuente propia]

La programación de nodos dentro de ROS se la puede realizar utilizando varios lenguajes. Sin embargo, los más usados son C++ y Python. Siendo este último el que se escogió para el desarrollo de este proyecto, el cual dispone de una librería cliente pura para ROS llamada rospy. Esta librería interactúa rápidamente con los tópicos, servicios y parámetros del programa permitiendo probar de manera ágil algoritmos dentro del software, ya que muchas de las herramientas de ROS son escritas en esta librería para aprovechar diferentes capacidades que ayudan en la comunicación y tratamiento de datos [28].

Sabiendo que se va a emplear la plataforma TurtleBot3 en la siguiente sección se detallan los tópicos de esta y el tipo de mensajes. De esta forma se identifica la información que se puede publicar o recibir.

2.1.1.3 TÓPICOS TURTLEBOT3

Los tópicos se pueden dividir en tópicos de subscripción recibidos por el TurtleBot3 y tópicos de publicación transmitidos desde el TurtleBot3, como se observan en la Tabla 2.1 y 2.2, respectivamente. Si bien no es necesario conocer todos los tópicos de subscripción y publicación, se considera una buena práctica aprender cómo usarlos.

Dentro de la Tabla 2.1 se debe prestar especial atención al tópico /cmd_vel el cual es bastante útil para controlar al robot, ya que a través de este se pueden dar instrucciones de avanzar, retroceder y realizar giros a la izquierda o derecha.

Tabla 2.1. Tópicos de Subscripción recibidos por el TurtleBot3 [29].

Nombre del tópico	Tipo de Mensaje	Función
motor_power	std_msgs/Bool	Enciende o apaga el torque de los motores Dynamixel de las llantas del TurtleBot3
reset	std_msgs/Empty	Reinicia los datos de odometría y de la unidad de medida de inercia IMU
sound	turtlebot3_msgs/Sound	Sonido de pitido de salida
cmd_vel	geometry_msgs/Twist	Controla la velocidad de traslación y rotación del robot en m/s y rad/s respectivamente

Algunos de los tópicos de publicación son los tópicos como ‘joint_states’, ‘sensor_state’, ‘odom’, ‘version_info’ y ‘tf’. Dentro de estos destaca ‘odom’ para la información de odometría, el cuál recopila información a través de encoders.

Tabla 2.2. Tópicos de publicación transmitidos desde el TurtleBot3 [29].

Nombre del tópico	Tipo de mensaje	Función
sensor_state	turtlebot3_msgs/SensorState	Tópico que contiene el valor de los sensores montados en el TurtleBot3
battery_state	sensor_msgs/BatteryState	Contiene el estado y el voltaje de la batería
scan	sensor_msgs/LaserScan	Tópico que contiene los valores de escaneo del LiDAR 360 montado en el TurtleBot3 tales como ángulo máximo y mínimo, y una matriz con todo el rango de visión.
imu	sensor_msgs/Imu	IMU o unidad de medida de inercia, es un tópico que contiene la posición del robot basado en el sensor de aceleración y giroscopio
odom	nav_msgs/Odometry	Incluye la información de odometría del TurtleBot3 basado en el encoder y el IMU
tf	tf2_msgs/TFMessage	Contiene la transformada de coordenadas como base_footprint y odom
join_states	sensor_msgs/JointState	Revisa la posición (m), velocidad (m/s) y el esfuerzo (N*m) cuando las ruedas son consideradas como articulaciones
diagnostics	diagnostic_msgs/DiagnosticArray	Contiene la información de autodiagnóstico

version_info	turtlebot3_msgs/VersionInfo	Contiene la información de software, hardware y firmware del TurtleBot3
cmd_vel_rc100	geometry_msgs/Twist	Este tópico se publica cuando el controlador basado en Bluethooth, RC100, se conecta al control de velocidad (m/s) y a la velocidad angula (rad/s) del robot móvil
camera	sensor_msgs/Image	Este tópico envía toda la información respecto a la cámara incorporada en la plataforma TurtleBot3

Cabe señalar que el modelo Burger no posee una cámara, pero al poseer una estructura modular se puede añadir una. Lo presentado en esta sección corresponde a todos los tópicos a los cuales se tendría acceso en caso de disponer del robot físico; sin embargo, varios de estos tópicos también se presentan en el robot virtual. Por lo que, al simular dentro de una plataforma como Gazebo, se tendrá a disposición los tópicos mostrados en la Sección 2.1.4.

2.1.2 COMPONENTE EN GAZEBO

En el simulador de Gazebo se tiene la interfaz mostrada en la Figura 2.3, la cual cuenta con cuatro secciones principales: escena, panel izquierdo o paleta, panel derecho y barra de herramientas.

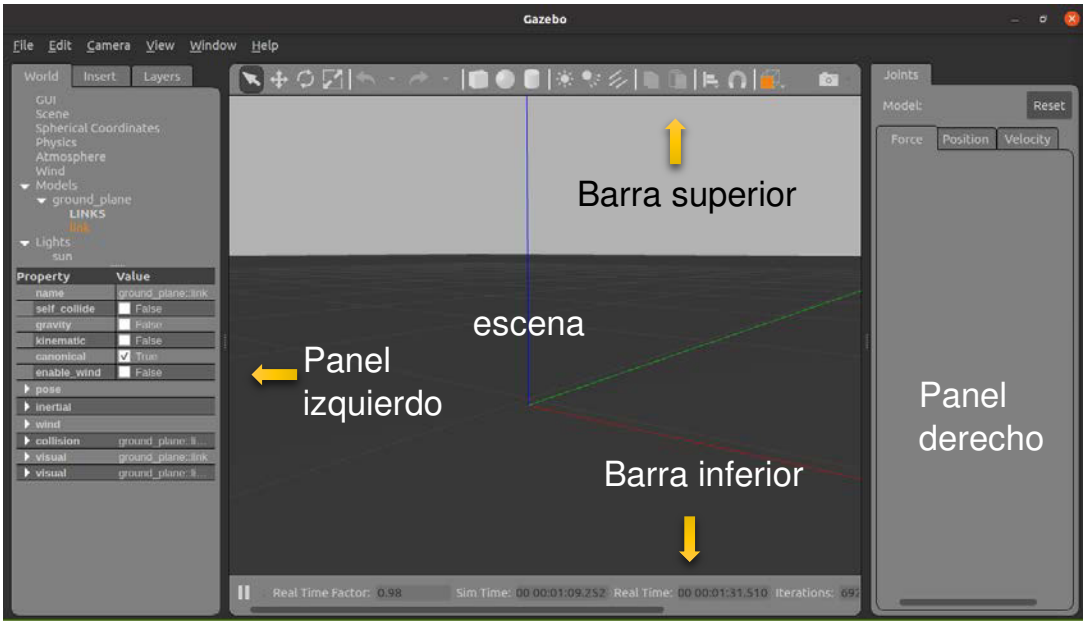


Figura 2.3. Interfaz de Gazebo [Fuente propia]

Se debe mencionar que en Gazebo se puede crear desde cero el entorno de simulación o si se desea un robot. Si este es el caso, se recomienda visitar la página oficial de Gazebo Tutorials [30], para tener más detalles. De manera general se procede a describir los archivos principales que se necesita para el modelamiento de un ambiente o robot [31]:

- **Archivo *nombre*.sdf:** este archivo se lo realiza en lenguaje XML y contiene elementos que describen físicamente al ambiente o robot como, por ejemplo: disposición de actuadores, sensores, descripción general del robot, inercia, masa, amortiguamiento, etc., del TurtleBot3. Este tipo de archivos se lo guardan generalmente en la carpeta Models, descrita anteriormente en la Sección 2.1.1.1

- **Archivo *nombre*.config:** al igual que el anterior archivo se lo realiza en lenguaje XML y se lo utiliza para definir información sobre los desarrolladores del proyecto y la dirección del archivo .sdf. Al igual que el anterior archivo, se lo guarda dentro de la carpeta Models.

- **Archivo *nombre*.world:** contiene la descripción de leyes físicas y demás características dentro del ambiente como la viscosidad, gravedad, fricción, superficies de contacto, entre otras. Los archivos .world se los guarda en la carpeta Worlds, dentro del paquete creado, tal como se indicó en la Sección 2.1.1.1

Para el desarrollo del entorno libre de obstáculos se añadió una escena como cielo con nubes y un suelo tipo pavimento. En el archivo .world se configuró dentro del código una

parte para la escena de cielo y otra para el tipo de piso, de la forma indicada en las Figuras 2.4 y 2.5

```
<!-- CIELO-->
<scene>
      <ambient>0.4 0.4 0.4 1</ambient>
      <background>0.7 0.7 0.7 1</background>
      <shadows>true</shadows>
      <sky>
        <clouds>
          <speed>15</speed>
        </clouds>
      </sky>
    </scene>
```

Figura 2.4. Código empleado para implementar el cielo [Fuente propia]

```
<!-- SUELO -->
  <include>
    <uri>model://my_ground_plane1</uri>
  </include>
```

Figura 2.5. Código empleado para implementar el tipo de suelo [Fuente propia]

La modificación del piso o suelo comienza habilitando la opción de “Mostrar archivos ocultos” en la sección de Carpeta personal, tal como se muestra en la Figura 2.6, para poder entrar en la carpeta .gazebo.

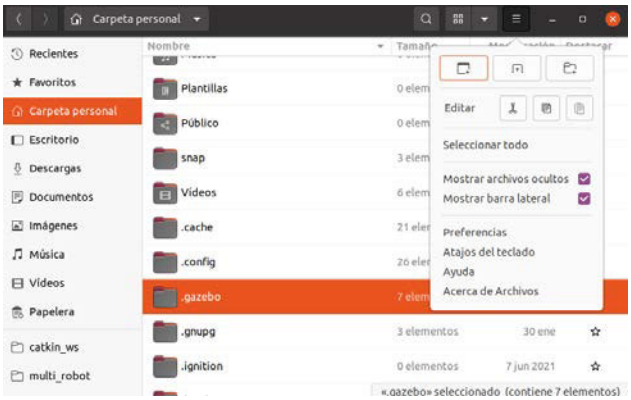


Figura 2.6. Acceso a carpeta .gazebo [Fuente propia]

Luego se debe navegar dentro de esta carpeta siguiendo la ruta: .gazebo\models\my_ground_plane1. Al llegar a la carpeta my_ground_plane1 se tiene dos carpetas más: scripts y textures. (véase en la Figura 2.7)

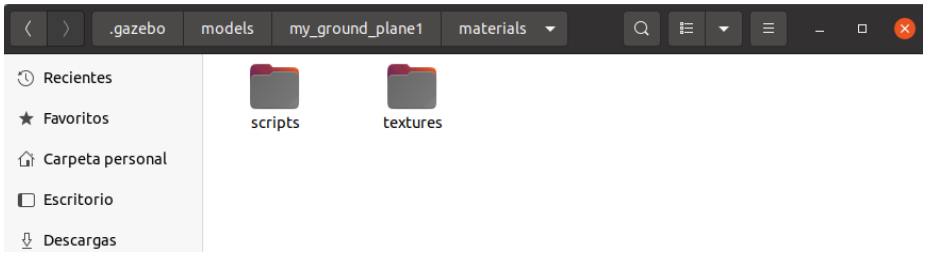


Figura 2.7. Acceso a carpeta my_ground_plane1 [Fuente propia]

Al entrar en carpeta Scripts, se encuentra el código repeated.material que se lo utiliza con el fin de modificar el tipo de imagen que se desea como suelo y las dimensiones, en la Figura 2.8, se puede observar que el nombre de la imagen ocupada está subrayada en color naranja

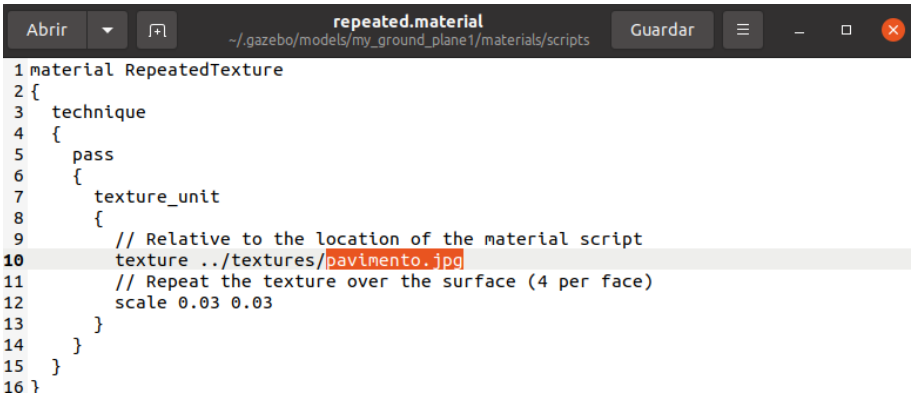


Figura 2.8. modificación de archivo repeated.material [Fuente propia]

Finalmente, para la configuración del tipo de suelo dentro de la carpeta textures se ocupó la imagen pavimento.jpg. Además, es posible aumentar diferentes imágenes tal como se observa en la Figura 2.9.

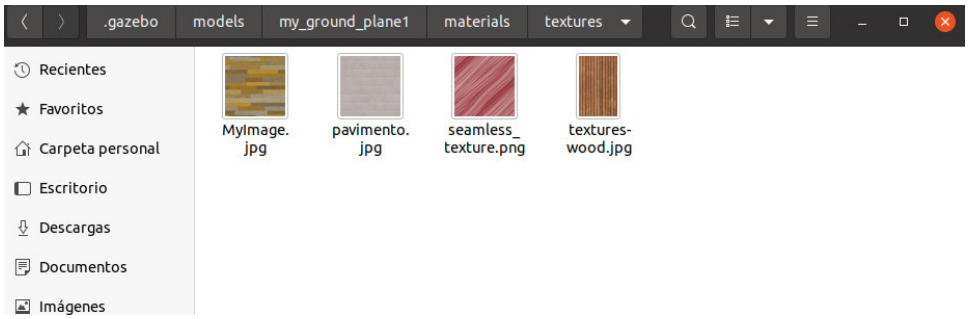
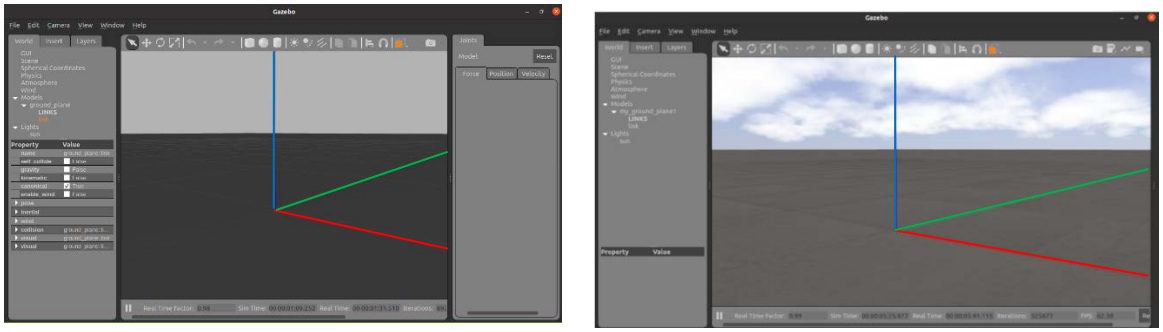


Figura 2.9. Imágenes en carpeta textures [Fuente propia]

Dentro de esta sección, se presentó la escena por default de Gazebo (véase la Figura 2.3), la cual para efectos de comparación también puede apreciarse en la Figura 2.10(a). Realizando las configuraciones mencionadas se obtiene el entorno de simulación libre de

obstáculos que se será utilizado para el segundo tomo del trabajo desarrollado en conjunto, visto en la Figura 2.10(b).



(a) (b)

Figura 2.10. Entorno de simulación en Gazebo (a) por defecto y (b) desarrollado [Fuente Propia]

En ambos entornos de simulación se observa que se tiene un sistema de referencia estático propio de Gazebo, donde el eje “x” está representado por el color rojo, el eje “y” en verde y el eje “z” en azul. Este sistema se lo utiliza como referencia para obtener datos de posición del TurtleBot3. Cabe señalar que el TurtleBot3 posee su propio sistema de referencia y podrá ser visto en la Sección 2.1.2.

Dado que este proyecto utiliza un modelo de robot comercial, no se debe crear uno desde cero en Gazebo ya que se emplea la plataforma TurtleBot3, la cual, al ser respaldada por ROS, tiene incluidos todos los archivos necesarios para ocuparlos directamente en el simulador.

2.1.3 ROBOT VIRTUAL TURTLEBOT3

Lo primero que se debe realizar es la instalación del paquete turtlebot3 y del paquete turtlebot3_msgs, lo cual se encuentra detallado en el apartado correspondiente al inicio rápido, de la página web ROBOTIS e manual [16]. Estos paquetes son útiles tanto para la simulación como para el manejo del robot real. Luego, dentro de la misma página, se debe revisar el apartado correspondiente a simulación y seguir las instrucciones para instalar el paquete turtlebot3_simulations.

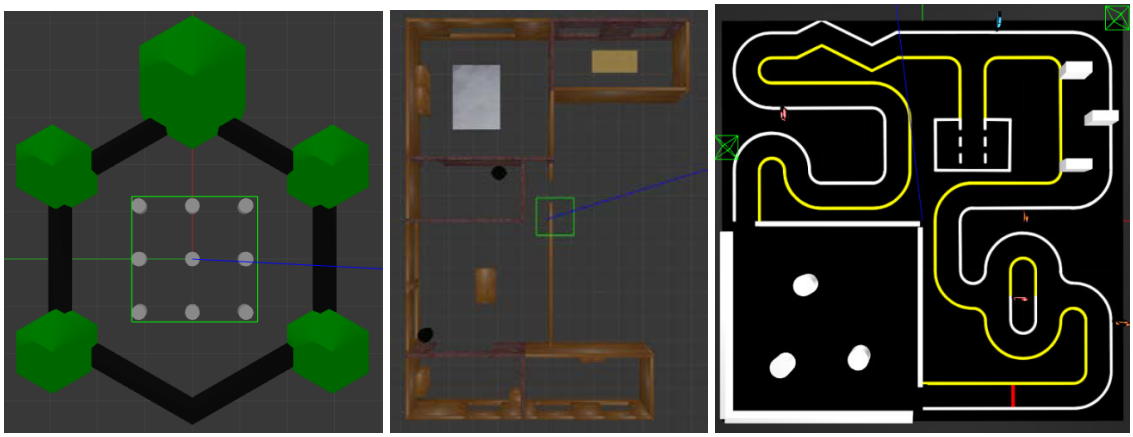
El paquete turtlebot3_simulations contiene al modelo 3D del robot TurtleBot3 de forma virtual a través de los siguientes archivos.

- **urdf/:** (United Robot Description Format) es un formato de lenguaje XML que se emplea para describir todo lo que corresponde al aspecto físico del robot, es decir, contiene la representación xacro de diferentes partes del robot, principalmente se

encuentra todo en cuanto a su estructura y los sensores que posee dependiendo de su placa modular.

- **meshes/:** contiene archivos tipo mesh (.dae, .3ds, .stl) que se utilizan para propiedades de colisión y la respectiva visualización en 3D.

Además, el paquete contiene mundos ya construidos en Gazebo, como los mostrados en la Figura 2.11, que pueden ser usados. También cuenta con ejemplos para ser ejecutados a través de archivos .launch para observar el funcionamiento y posibles aplicaciones de la plataforma.



(a) (b) (c)

Figura 2.11. Mapas proporcionados por el paquete turtlebot3_simulations en Gazebo (a) turtlebot3_world (b) turtlebot3_house (c)turtlebot3_autorace_2020

Basándose en los ejemplos del paquete instalado así como de la información que se muestra en ROS Answers [32], se crean archivos .launch propios de este trabajo, especificados en la Figura 2.12 , los cuales se encargan de lanzar el mundo creado en conjunto con la plataforma TurtleBot3

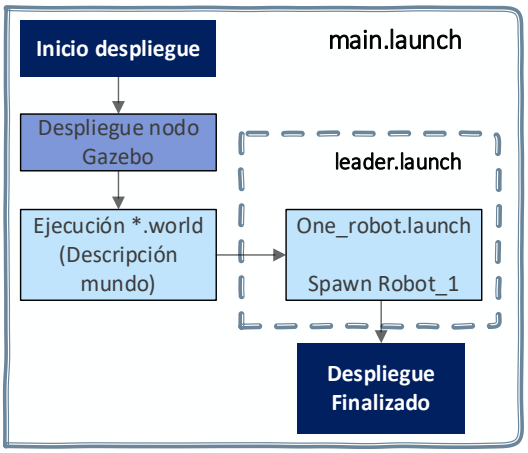


Figura 2.12. Ejecución del archivo main.launch

En la Figura 2.12 se muestra gráficamente cómo funciona el archivo llamado ‘main.launch’, el cual realiza el despliegue del nodo en Gazebo y ejecuta el archivo .world que se le especifique. En el caso de este trabajo, este es un entorno libre de obstáculos con un piso y un cielo diferente al original. Seguido a esto, se llama a otro archivo llamado ‘leader.launch’ en el cual se ingresan los valores de la posición en la que se desea que aparezca el robot para finalmente llamar al archivo ‘One_robot.launch’ que hace el ‘spawn’ del robot en la posición especificada. El ‘spawn’ del robot se refiere al lanzamiento del modelo URDF, explicado anteriormente, dentro del entorno de simulación. Con esto, se da fin al despliegue y se tiene la plataforma TurtleBot3 lista para su uso dentro del entorno de simulación virtual, como se observa en la Figura 2.13.

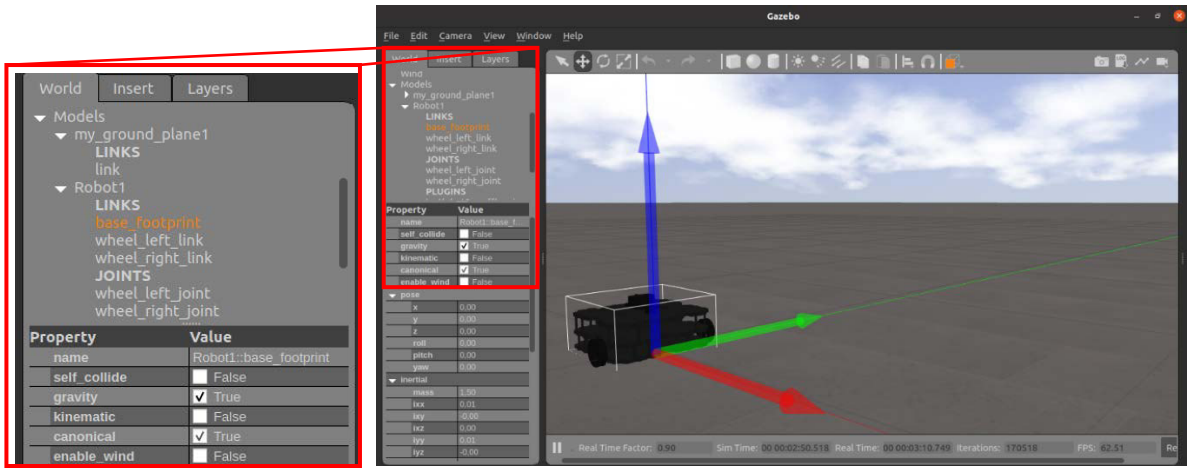


Figura 2.13. Visualización de elementos de TurtleBot3 [Fuente propia]

Los elementos de modelamiento se pueden observar en el panel izquierdo de Gazebo y de forma ampliada a la izquierda de la Figura 2.13. Estos a su vez significan:

- **Model:** el modelo se refiere al nombre del robot. En este elemento se encuentran archivos que detallan la física y geometría de los elementos modelados.
- **Link:** este elemento se usa con el fin de describir características visuales y de inercia, con lo que se define puntos de contacto y demás visualización.
- **Joint:** describe las características estáticas y dinámicas de las articulaciones del robot así también como los límites, ejes de rotación, tipo de articulación, entre otras.
- **Sensor:** el elemento es usado para obtener las propiedades básicas que definen al sensor lo que permite relacionar a los plugin para obtener la información necesaria del entorno virtual.

Para escoger el modelo TurtleBot3 que se desea que aparezca en el entorno de simulación se debe ejecutar previo al archivo 'main.launch' uno de los comandos presentados en la Figura 2.14:

```
$ export TURTLEBOT3_MODEL=burger
$ export TURTLEBOT3_MODEL=waffle
$ export TURTLEBOT3_MODEL=waffle_pi
```

Figura 2.14. Comandos para exportar el modelo del TurtleBot3 [Fuente propia]

Si bien en este tomo se trabaja únicamente con un robot, Para el siguiente tomo [1] se deben desplegar varios robots dentro del entorno simulación, haciendo necesario la creación de otro archivo .launch, que de una forma esquemática se presenta en la Figura 2.15.

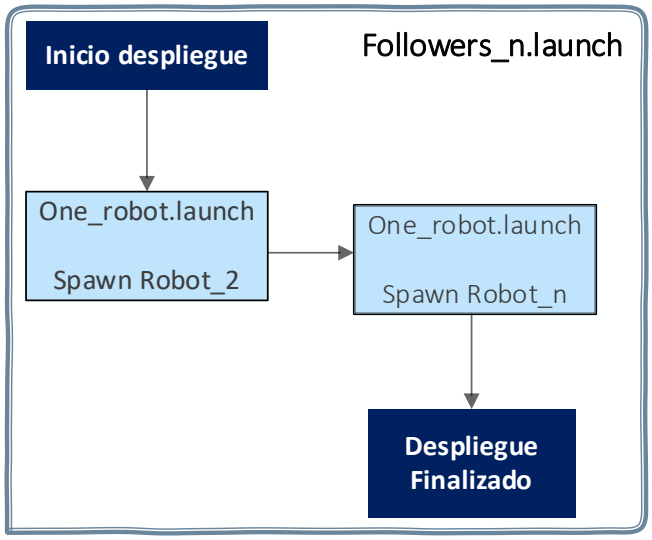


Figura 2.15. Ejecución del archivo Followers_n.launch

En la Figura 2.15 se observa que en el despliegue ya no se incluye el nodo de Gazebo puesto que este archivo se lanza después de haber ejecutado el 'main.launch' por lo que ya se tiene el entorno de simulación. Pasando así directamente a llamar al archivo 'One_robot.launch' el cual hace el 'spawn' del robot en la posición que le sea indicado y así sucesivamente dependiendo del número de robots que se desee que aparezcan. Para visualizar estos resultados referirse al segundo tomo [1] del trabajo en conjunto.

Conviene destacar que los nombres utilizados para los archivos como 'leader.launch' y 'Followers_n.launch' se debe a que en el siguiente tomo [1], donde serán usados, se implementa un algoritmo de formación líder seguidor.

2.1.4 TÓPICOS DE SIMULACIÓN

Cuando se utiliza la plataforma TurtleBot3 dentro del entorno de simulación Gazebo, se le debe asignar un nombre bajo el cual se puede diferenciar el robot que aparece en el entorno virtual. En el caso de este tomo se ha decidido nombrarlo como robot1 y para el caso del tomo 2 [1], en el cual se tienen varios robots dentro del entorno de simulación, se les asignará el nombre de robot$n$, donde n representa el número del robot y $n = 1$ corresponderá al robot líder.

Asumiendo que ya se tiene al robot TurtleBot3 dentro del entorno de simulación y se desea comprobar qué tópicos se tienen disponibles, se puede ejecutar el comando ($rostopic list) el cual listará todos los tópicos disponibles como se observa en la Figura 2.16. Conviene señalar que en esta imagen también aparecen los tópicos relacionados a ROS y Gazebo.

```
$rostopic list

/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/robot1/camera/parameter_descriptions
/robot1/camera/parameter_updates
/robot1/camera/rgb/camera_info
/robot1/camera/rgb/image_raw
/robot1/camera/rgb/image_raw/compressed
/robot1/camera/rgb/image_raw/compressed/parameter_descriptions
/robot1/camera/rgb/image_raw/compressed/parameter_updates
/robot1/camera/rgb/image_raw/compressedDepth
/robot1/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/robot1/camera/rgb/image_raw/compressedDepth/parameter_updates
/robot1/camera/rgb/image_raw/theora
/robot1/camera/rgb/image_raw/theora/parameter_descriptions
/robot1/camera/rgb/image_raw/theora/parameter_updates
/robot1/cmd_vel
/robot1/imu
/robot1/joint_states
/robot1/odom
/robot1/scan
/rosout
/rosout_agg
/tf
```

Figura 2.16. Tópicos disponibles de Gazebo, TurtleBot3 y ROS resaltando los que generalmente se usan para el control del TurtleBot3 [Fuente Propia].

En la Figura 2.16 se muestran todos los tópicos al simular la plataforma TurtleBot3 los cuales se dividen en tres grandes categorías:

- /**gazebo:** representan a los tópicos que gazebo utiliza para simular al TurtleBot3 en el entorno desarrollado.
- /**robot1:** representa los tópicos disponibles en simulación del TurtleBot3, varios de estos se pueden visualizar con mayor detalle en la Sección 2.1.1.3
- /**rosout:** corresponde a los tópicos de ROS
- /**tf:** corresponde a las coordenadas de transformación y es uno de los conceptos más usados cuando se describe las partes del robot, así como también los obstáculos y los objetos.

En el caso del presente trabajo, se tienen acceso a todos los tópicos vistos en la Figura 2.1; sin embargo, para controlar al robot TurtleBot3 a través del teclado y la aplicación de teléfono ROS Mobile, se publica y subscribe a los tópicos subrayados en amarillo correspondientes al de velocidad y odometría respectivamente. El tópico 'cmd_vel', se observa en la Tabla 2.3 y permite escribir la velocidad lineal y la velocidad angular haciendo referencia a tres ejes de coordenadas (x, y, z).

Por otro lado, el tópico 'odom', que también se muestra en la Tabla 2.3, permite conocer la posición del robot lo cual es útil para obtener resultados dentro de este tomo y es fundamental para escribir el algoritmo de control del segundo tomo [1] del trabajo en conjunto.

Tabla 2.3. Contenido de los tópicos /cmd_vel y /odom del TurtleBot3 en Gazebo [Fuente Propia].

Comandos y resultados	
$ rostopic echo /robot1/cmd_vel	**$ rostopic echo /robot1/odom**

linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
	header:
	  seq: 14967
	  stamp:
	  secs: 615
	  nsecs: 421000000
	  frame_id: "robot1_tf/odom"
	child_frame_id: "robot1_tf/base_footprint"
	
	pose:
	  pose:
	
	  position:

	x: 0.8611752835200949 y: -1.7394009652472868 z: -0.0010087799728917962 orientation: x: -2.2054075320693868e-05 y: 0.0015938387000223471 z: 0.013619112394434758 w: 0.9999059850652979 covariance: [1e-05, 0.0, 0.0, 0.0, …] twist: twist: linear: x: 1.2679912416382112e-06 y: 2.889235798207622e-06 z: 0.0 angular: x: 0.0 y: 0.0 z: 1.8388716956434736e-05 covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, …]

Puede llamar la atención que en el tópico ‘cmd_vel’ para el control de velocidad, tanto lineal como angular, se deban escribir tres variables $(\boldsymbol{x, y, z})$. Esto se debe a que los tópicos en ROS utilizan un formato de mensajes específico, para el caso de ‘cmd_vel’ es el geometry_msgs/Twist, como se explicó anteriormente en la Tabla 2.1 de la Sección 2.1.1.3. El formato de los mensajes resulta genérico y se pueden utilizar en otro tipo de robots.

Igualmente, se debe recordar que la velocidad lineal representa el cambio de posición de un objeto con el tiempo en una ruta rectilínea $(\boldsymbol{m/s})$, siendo así la velocidad lineal en el eje x el desplazamiento del robot hacia adelante, en el eje y el desplazamiento de forma lateral y en el eje z el desplazamiento de forma vertical, considerando que se toma como referencia el eje de coordenadas atado al robot presentado en la Figura 2.17, mismo que se encuentra colocado en el frente del TurtleBot3 para sus dos modelos.

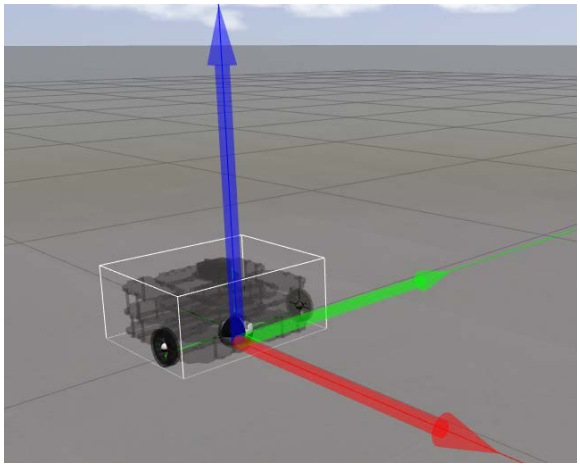


Figura 2.17. Ejes de coordenadas atado al TurtleBot3 [Fuente propia]

Para un robot móvil de tracción diferencial, como el TurtleBot3, únicamente se envía el comando de velocidad lineal correspondiente al eje “x”, puesto que en el eje “y” no sería posible debido a la restricción no holonómica del robot. Asimismo, no se pueden enviar comandos de velocidad lineal correspondientes al eje “z” ya que no se trata de un robot móvil aéreo que pueda elevarse.

De manera similar, se debe tener presente que la velocidad angular es una medida de velocidad de rotación y se lo puedes describir como el ángulo girado por una unidad de tiempo (rad/s)**.** Siendo así la velocidad angular en “x” equivalente al roll, la velocidad angular en “y” equivalente al pitch y finalmente la velocidad angular en “z” igual al yaw, considerando que se toma como referencia el eje de coordenadas atado al robot, lo cual se observa de mejor manera en la Figura 2.18.

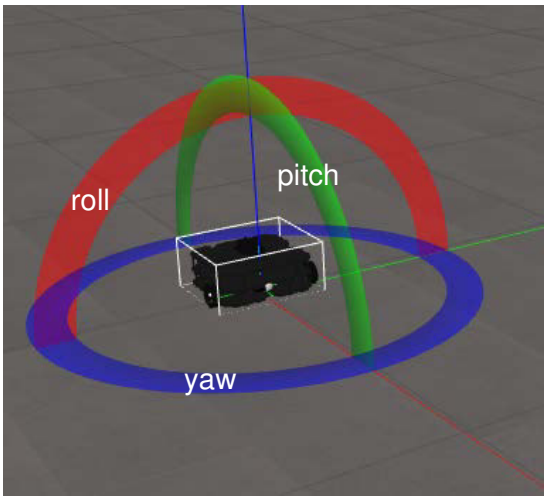


Figura 2.18. Ejes de rotación atados al TurtleBot3 [Fuente propia]

Estos ángulos de rotación roll, pitch y yaw se utilizan generalmente en la aviación o en robots móviles aéreos. Sin embargo, para un robot móvil de tracción diferencial que se mueve en una superficie plana horizontal, únicamente se puede enviar el comando de rotación respecto al eje "z", es decir, el ángulo yaw que representa el giro del robot.

El tópico /odom, cuyo contenido se describió en la Tabla 2.3, puede acceder a datos de posición y orientación del robot. Por un lado se tiene a la posición representada por tres coordenadas (x,y,z) referidas al sistema de referencia de Gazebo, como se observa en la Figura 2.10. Mientras que, la orientación está dada por el sistema de coordenadas (x,y,z,w) lo cual indica que la orientación está representada en cuaterniones.

Los cuaterniones son números hipercomplejos de cuatro componentes, una componente real y tres imaginarias [33], tal como se ve en la Ecuación (2.1). Estos son bastante empleados para representar rotaciones dentro de la robótica, y ROS también así lo hace, esto debido a su notación compacta y la rapidez con la que se realizan sus cálculos mediante matrices. Además, la forma del cuaternion está libre del bloqueo del cardán o los problemas de velocidad que se presentan en el método de Euler de los vectores roll, pitch y yaw [29].

$$q = w + x * \boldsymbol{i} + y * \boldsymbol{j} + z * \boldsymbol{k} \tag{2.1}$$

Donde w, x, y, z son números reales, mientras que $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ son unidades imaginarias que se definen mediante un sistema de igualdades.

El uso de cuaterniones no es intuitivo, ya que no se describe la rotación en tres ejes (x,y,z) como se lo hace con los ángulos de roll, pitch y yaw, que son comúnmente usados. Disponiendo así por conveniencia de funciones que conviertan valores de Euler a cuaterniones y viceversa. Para este fin, en el presente trabajo se importó la función "euler_from_quaternion" de la librería de Python "tf.transformations" [34].

2.2 MOVIMIENTO DEL ROBOT

Como se observó en la sección anterior, a través del tópico 'cmd_vel' se pueden dar instrucciones al robot de avanzar, retroceder y realizar giros a la izquierda o derecha. Las posibilidades que se tiene para escribir sobre este comando son extensas, ya que se puede utilizar dispositivos remotos para su manejo como el Joystick del PS3 o de la XBOX 360. En el caso del presente trabajo, se emplea el teclado de una computadora y un teléfono celular, con la aplicación ROS Mobile.

Así también, se puede diseñar una ley de control como es el caso del tomo 2 del trabajo en conjunto, en el cual el algoritmo creado se encarga de asignar estos parámetros de velocidad al robot. Del mismo modo, existen paquetes de uso público que incluyen conducción autónoma, evasión de obstáculos y navegación, cuyas leyes controlan principalmente los parámetros de velocidad [16].

2.2.1 MOVIMIENTO POR TECLADO

Teniendo en cuenta los tópicos en los que se va a enfocar, para mover el robot se realizan dos cosas importantes: inicialización del nodo para publicar en el tópico de velocidad y la lectura del teclado de modo que se tenga las teclas dispuestas de la siguiente forma:

- a: aumenta la velocidad lineal en pasos de $+0.01\ m/s$
- w: disminuye la velocidad lineal en pasos de $-0.01\ m/s$
- d: aumenta la velocidad angular en pasos de $+0.01\ rad/s$
- x: disminuye la velocidad angular en pasos de $-0.01\ rad/s$
- s o espacio: detiene al robot reiniciando la velocidad lineal y angular a 0

En la Figura 2.19, se puede apreciar el diagrama de flujo del algoritmo que emplea el teclado del computador para mover al robot.

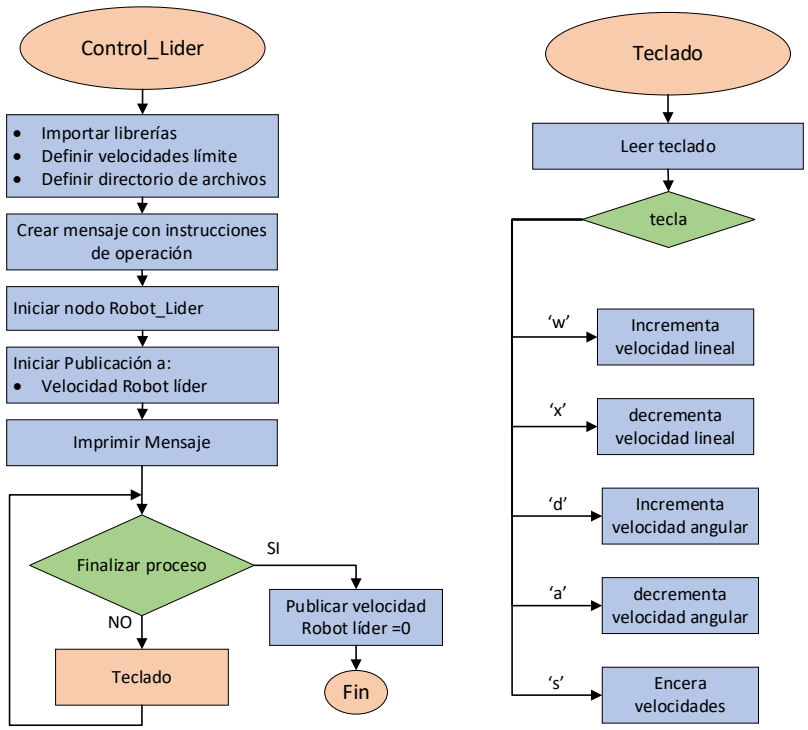


Figura 2.19. Diagrama de flujo del movimiento del robot mediante teclado [Fuente propia]

Una vez implementado el algoritmo de control de movimiento del robot TurtleBot3 mediante teclado, se ejecutan los nodos correspondientes al control del robot y al entorno de simulación Gazebo, como se explicó en la Sección 2.1.2. Obteniendo así el diagrama de nodos y tópicos que se muestra en la Figura 2.20 o en la Figura 2.21. La diferencia entre ambas imágenes se debe a que para el caso de la Figura 2.20 se tiene activada la opción Debug en la ventana rqt_graph, haciendo visible el nodo /rosout el cual permite visualizar los nodos correspondientes a ROS. Si bien este diagrama es más completo se desactivará de aquí en adelante la opción Debug, tanto para este tomo como para el siguiente [1], sobre todo por cuestión de espacio, repetición del nodo base /rosout en todos los diagramas y con el objetivo de aumentar la facilidad de distinguir entre los nodos y tópicos que se está utilizando.

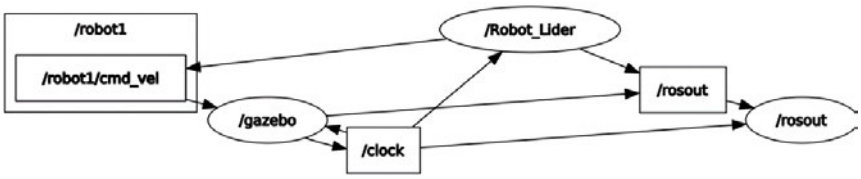


Figura 2.20. Diagrama de nodos y tópicos del movimiento del Robot Turtlebot3 Waffle Pi mediante teclado con la opción de Debug activada [Fuente propia]

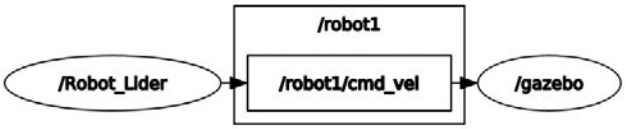


Figura 2.21. Diagrama de nodos y tópicos del movimiento del Robot Turtlebot3 Waffle Pi mediante teclado [Fuente propia]

Tanto en la Figura 2.20 como en la Figura 2.21 se aprecia el nodo /Robot_Lider que corresponde al script de Python creado para publicar sobre el tópico /robot1_cmd_vel el cual envía esta información a Gazebo para poder observar a través de simulación el movimiento que realiza el TurtleBot3.

Una vez se han desplegado los nodos se abre el entorno de simulación Gazebo y en la terminal aparece un mensaje con instrucciones para el manejo del TurtleBot3 Waffle Pi. A través de esta terminal se ingresan comandos de velocidad lineal y angular y al hacerlo se puede observar como el robot se mueve libremente en el entorno de simulación. Sin embargo, para comprobar qué tan bien llegan estos comandos al robot se abre una nueva ventana rqt_topic la cual permite monitorear todos los tópicos disponibles, en este caso se centra en el tópico de velocidad, en el cual se comprueba que efectivamente lo que se

ingresa por teclado le está llegando al TurtleBot3, como se lo puede apreciar en la Figura 2.22.

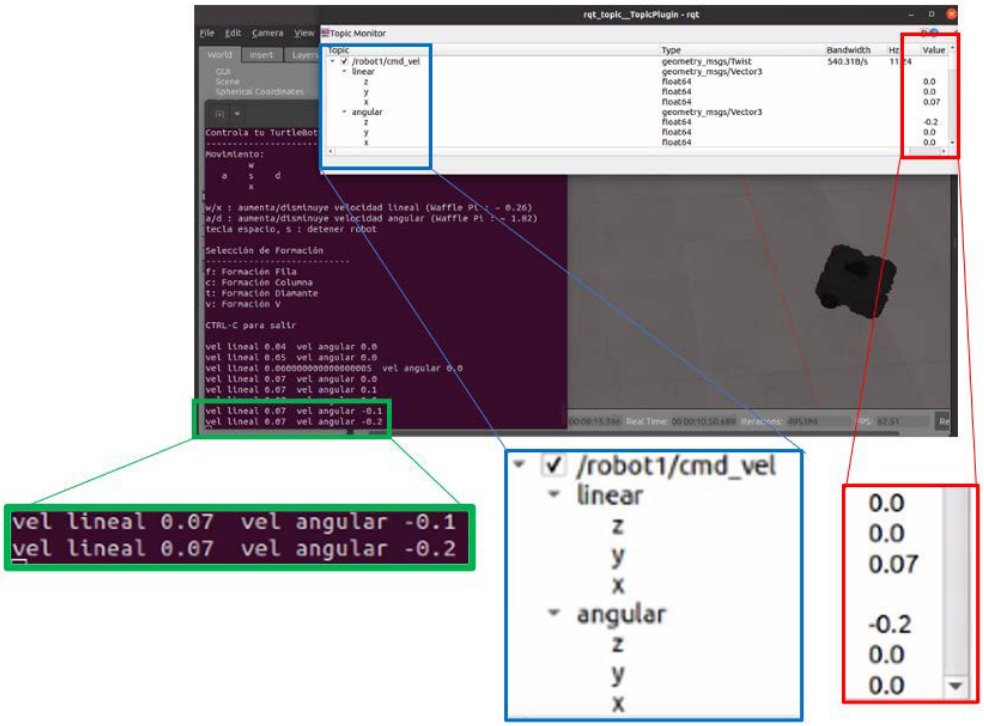


Figura 2.22. Movimiento del Robot Turtlebot3 Waffle Pi mediante teclado en el entorno creado en Gazebo [Fuente Propia]

2.2.2 MOVIMIENTO POR ROS MOBILE

Sabiendo que existe aplicaciones dentro de ROS innovadoras, como lo es ROS Mobile, se propone utilizar esta aplicación, habilitada para Android, con el fin de proporcionar un control dinámico para el robot de modo que el usuario pueda conectarse a través de su teléfono inteligente y dirija el movimiento del robot. Para instalar la aplicación ROS Mobile, en un smartphone Android se deberá dirigir a la aplicación de Play Store, y en el buscador de esta aplicación escribir ROS Mobile e instalar, véase la Figura 2.23.

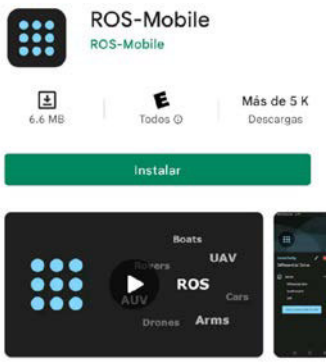


Figura 2.23. Aplicación ROS Mobile en Play Store [Fuente Propia]

Con la aplicación ya instalada, al abrirla se tiene una introducción sobre las características que posee y cómo implementarlas, mostradas mediante un video tutorial. Actualmente, no es posible cambiar el idioma dentro de la aplicación por lo que el idioma por defecto se mantiene en inglés. En la Figura 2.24 se muestran las capturas de pantalla de la introducción de la aplicación.

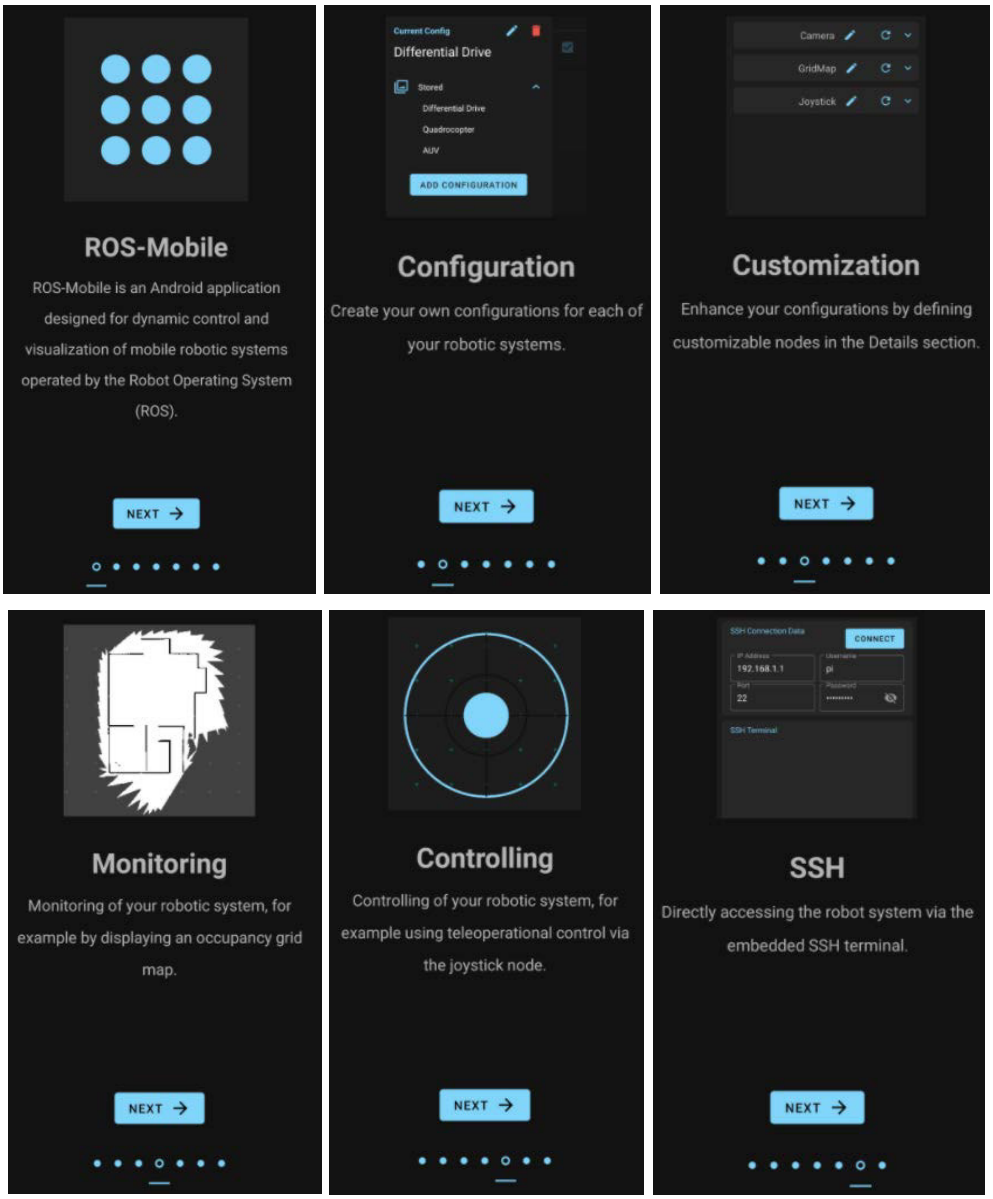


Figura 2.24. Introducción a ROS Mobile parte 1 [Fuente Propia]

Para empezar con la configuración que se desee realizar, se puede personalizar el nombre y luego seleccionar START CONFIGURATION tal como se observa en la Figura 2.25

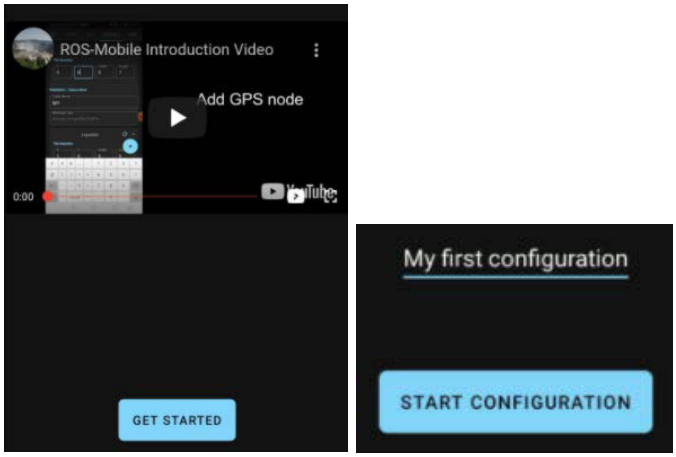


Figura 2.25. Introducción a ROS Mobile parte 2 [Fuente Propia]

A modo de resumen se presenta a continuación las características de la aplicación y los detalles de estos. ROS Mobile cuenta con cuatro secciones o pestañas dispuestas en la parte superior: MASTER, VIZ, DETAILS y SSH.

En la sección MASTER se tiene la configuración de conexión de la aplicación ROS Mobile con ROS. Para que logren conectarse es necesario que los dispositivos con el software ROS (computadora) y la aplicación ROS Mobile (smartphone) estén conectados a la misma red de Wifi. Luego, se deberá conocer el Master URL y el Master Port, estos se muestran resaltados de color amarillo en la Figura 2.26 y en la Figura 2.27, al ejecutar los comandos $ifconfig y $roscore, respectivamente. Además, el código o proyecto que se desee manejar con la aplicación deberá estar corriendo desde la computadora y, por último, verificar que los nombres de los tópicos de ROS con los de la aplicación coincidan.

```
gabriela@gaby:~$ ifconfig
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.218  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 2800:bf0:2a3:12eb:4174:ab55:7add:9e20  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::5431:9a1a:169b:e095  prefixlen 64  scopeid 0x20<link>
        inet6 2800:bf0:2a3:12eb:2302:7f80:5711:99af  prefixlen 64  scopeid 0x0<global>
        ether a4:4c:c8:38:0a:be  txqueuelen 1000  (Ethernet)
        RX packets 246996  bytes 290792046 (290.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 123784  bytes 32229993 (32.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Bucle local)
        RX packets 5182  bytes 703705 (703.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5182  bytes 703705 (703.7 KB)
```

Figura 2.26. Valor obtenido de Master URL en la terminal [Fuente Propia]

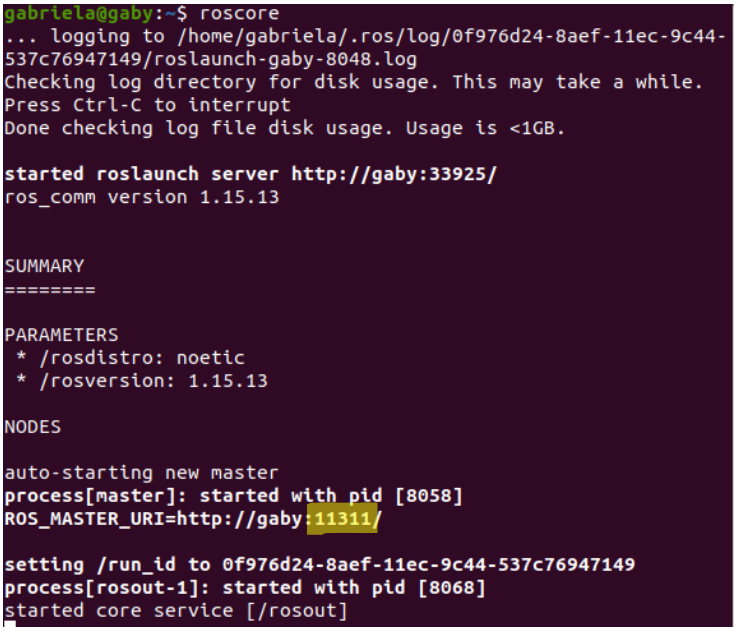


Figura 2.27. Valor obtenido de Master port en la terminal [Fuente Propia]

Los valores configurados, se muestran a continuación en la Tabla 2.4 y se ve su ingreso en la aplicación en la Figura 2.28.

Tabla 2.4. Configuración de parámetros en sección MASTER [Fuente Propia].

Parámetro	Descripción	Valor
Master URL	Dirección IP del computador donde se ejecuta ROS	192.168.100.218
Master port	Se lo encuentra como ROS MASTER URI y es una configuración para que los nodos ubiquen al maestro	11311
Network SSD	Nombre de red wifi	NETLIFE-PASTORA
Device IP adress	Dirección IP del teléfono con la aplicación ROS Mobile	192.168.100.160

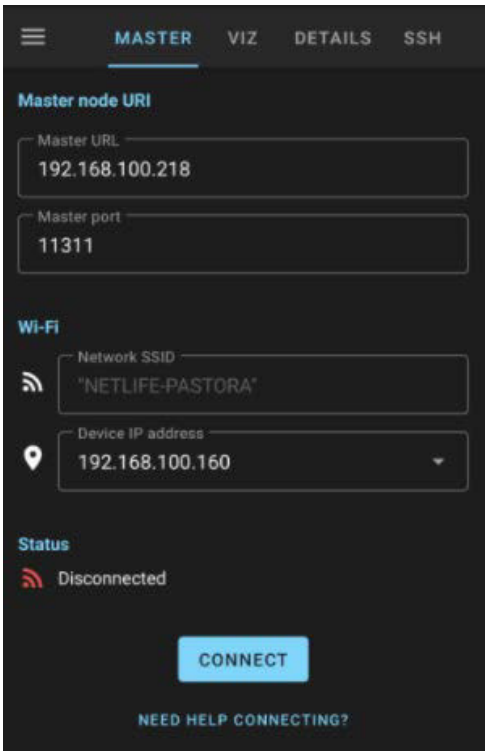


Figura 2.28. Configuración sección MASTER en ROS Mobile [Fuente Propia]

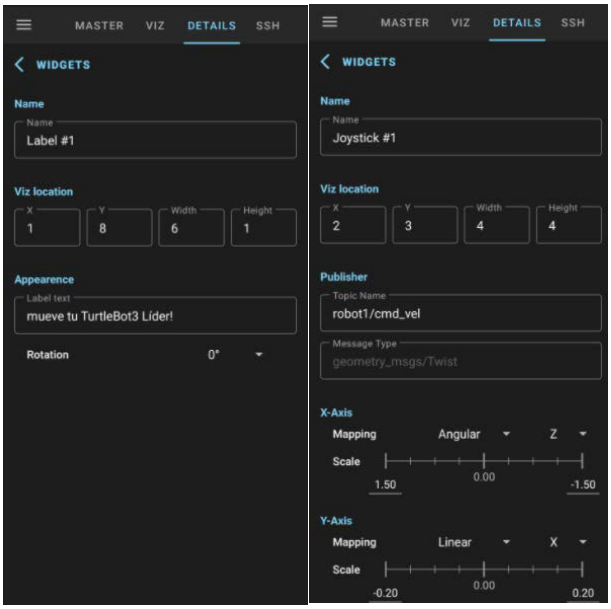
Luego, se muestra en la Figura 2.29 la sección VIZ, en esta no se debe hacer ninguna configuración ya que es el panel en donde se podrá añadir diferentes herramientas de la siguiente sección DETAILS.

Figura 2.29. Vista de sección VIZ en ROS Mobile [Fuente Propia]

En la sección DETAILS se pueden añadir diferentes herramientas o widgets, como: cámara, botón, etiquetas, joystick, GPS, entre otras opciones. En esta ocasión se escogen las herramientas: Label y Joystick. La configuración de la herramienta Label y Joystick se puede ver en la Figura 2.30 y los valores dispuestos se modificaron acorde a la Tabla 2.5.

Tabla 2.5. Configuración de parámetros en sección DETAILS [Fuente Propia].

Parámetro	Descripción	Label	Joystick
Name	nombre de la herramienta	Label #1	Joystick #1
Viz location	configuración de posición, ancho y largo de la herramienta en la sección VIZ	X=1 Y=7 Width=6 Height=1	X=2 Y=3 Width=4 Height=4
Apearence	edición del título y su rotación	mueve tu TurtleBot3 Líder! Rotación: 0°	No aplica
Publisher	configuración del tópico con el que se desea comunicar	No aplica	robot1/cmd_vel
X-Axis	Movimiento horizontal del Joystick digital. En este caso se escogió la modificación de velocidad angular	No aplica	Mapping: Angular-Z Scale: 1.50 [rad/s] a -1.50 [rad/s]
Y-Axis	Movimiento vertical del Joystick digital. En este caso se escogió la modificación de velocidad angular	No aplica	Mapping: Linear-X Scale: -0.20 [m/s] a 0.20 [m/s]



(a) (b)

Figura 2.30. Configuración sección DETAILS (a) Label y (b) Joystick en ROS Mobile [Fuente Propia]

A diferencia del movimiento del robot mediante teclado, al usar ROS Mobile únicamente se ejecuta el nodo de Gazebo en el computador el cual proporciona el entorno de simulación y el robot en cuestión. Seguido a esto, se debe seguir la configuración de la aplicación en el Smartphone, mostrada en la Sección 2.2.2, para desplegar el nodo de ROS Mobile que envía los comandos de velocidad, llamado /robot1/cmd_vel, el cual tiene el mismo nombre del tópico en el que publica como se puede apreciar en la Figura 2.31.

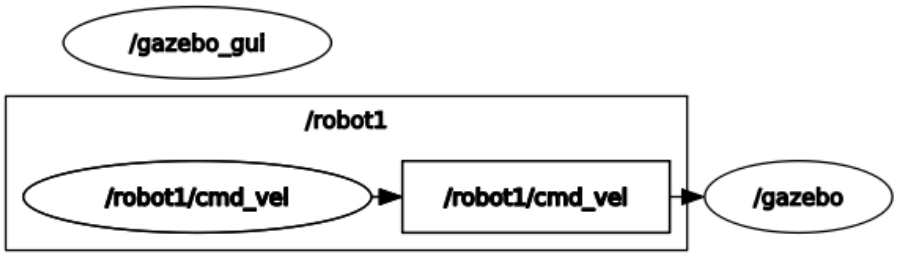


Figura 2.31. Diagrama de nodos y tópicos del movimiento del Robot Turtlebot3 Waffle Pi mediante la aplicación ROS Mobile

En este caso, cabe tener en cuenta que al utilizar la aplicación ROS Mobile para el manejo del robot TurtleBot3 se dispone de un Joystick en la pantalla del teléfono (véase la Figura 2.29), por lo que no se configura la velocidad lineal y angular como tal. Sino más bien depende de la forma como el usuario manipule el Joystick virtual.

2.2.3 PAQUETE CONDUCCIÓN AUTÓNOMA PARA EL TURTLEBOT3

El paquete *Autorace* se encuentra disponible en el ROBOTIS e manual [16] en la sección *Autonomouos Driving.* Siguiendo las instrucciones de la página web mencionada, se pueden descargar el paquete y las dependencias adicionales que este necesita para su ejecución. Se debe considerar que para su uso se debe haber descargado previamente el paquete turtlebot3_simulations como se muestra en la Sección 2.1.3.

El paquete *Autorace* tiene el propósito de desarrollar una aplicación de conducción autónoma, la cual utiliza las cámaras que vienen incorporadas en el robot para detectar la línea amarilla y la línea blanca de la pista, como se observa en la Figura 2.32, y a través de estas identificar el camino y comandar su movimiento. Para un buen desempeño se deben calibrar las cámaras como se indica en el mismo tutorial de la página de ROBOTIS e manual [16].

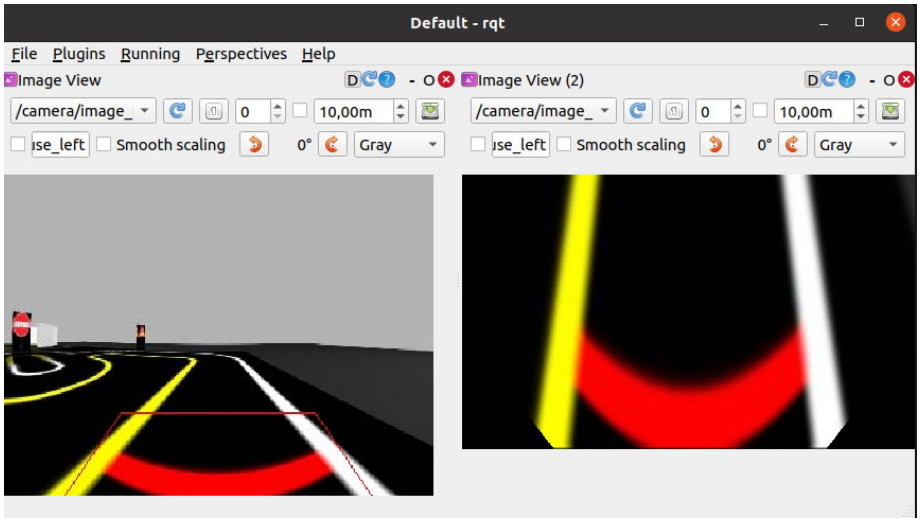
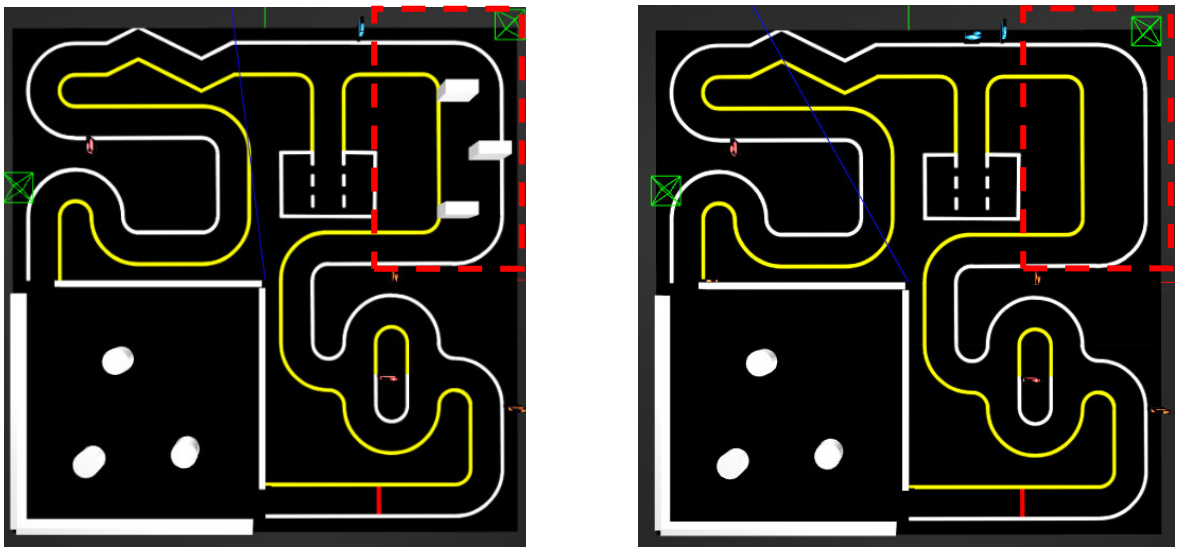


Figura 2.32. Vista de la cámara del robot TurtleBot3 Burger en el mapa turtlebot3_autorace_2020

Al ejecutar la aplicación de conducción autónoma se despliegan varios nodos, cuyo diagrama junto a los tópicos se lo puede apreciar en el Anexo V y el TurtleBot3 inicia su trayectoria a través del mapa.

En el presente trabajo se ha modificado la pista turtlebot3_auorace_2020 mostrada en la Figura 2.33 (a) eliminando los obstáculos que se encuentra dentro del recuadro en rojo en la esquina superior derecha obteniendo así la Figura 2.33 (b) para realizar las pruebas que se muestran en la Sección 3.



(a) (b)

Figura 2.33. (a) pista turtlebot3_autorace_2020 (b) pista turtlebot3_autorace_2020 modificada

Por otro lado, para poder capturar la trayectoria que sigue el robot con conducción autónoma se crea un nodo adicional que se subscriba al tópico de odometría del TurtleBot3 y que almacene estos datos en un archivo .csv, el cual en conjunto con la librería Matplotlib de Python permita graficar la trayectoria generada como se observa en la Figura 2.34.

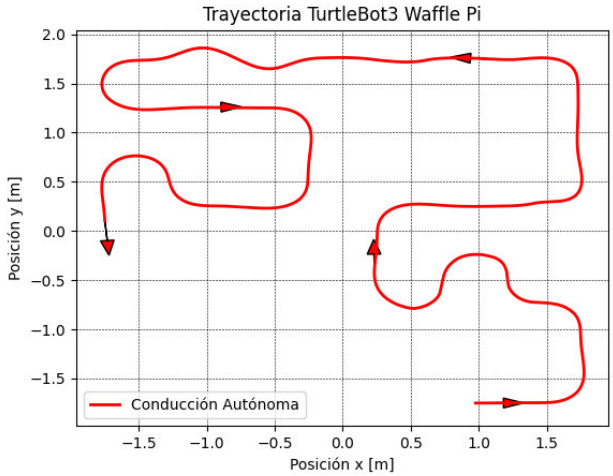


Figura 2.34. Trayectoria realizada por el algoritmo de conducción autónoma.

De esta forma, se puede comparar la trayectoria trazada con el control de conducción autónoma mostrada en la Figura 2.34 respecto a las trayectorias generadas por el manejo de un usuario mediante el teclado del computador o la aplicación de teléfono ROS Mobile.

2.3 CREACIÓN DE EJECUTABLES

Para facilitar la ejecución de la aplicación desarrollada se crea archivos bash, como se explicó en la Sección 1.4.4. Estos archivos permiten crear menús que serán utilizados tanto en este tomo como en el segundo del trabajo desarrollado en conjunto [1].

El diagrama de flujo del menú correspondiente al tomo uno se lo puede observar en la Figura 2.35. En esta gráfica se muestra como el usuario a través del archivo menu.sh puede escoger el modelo del robot TurtleBot3 y la opción para ejecutar el ejemplo de Autorace a través de Autónomo.

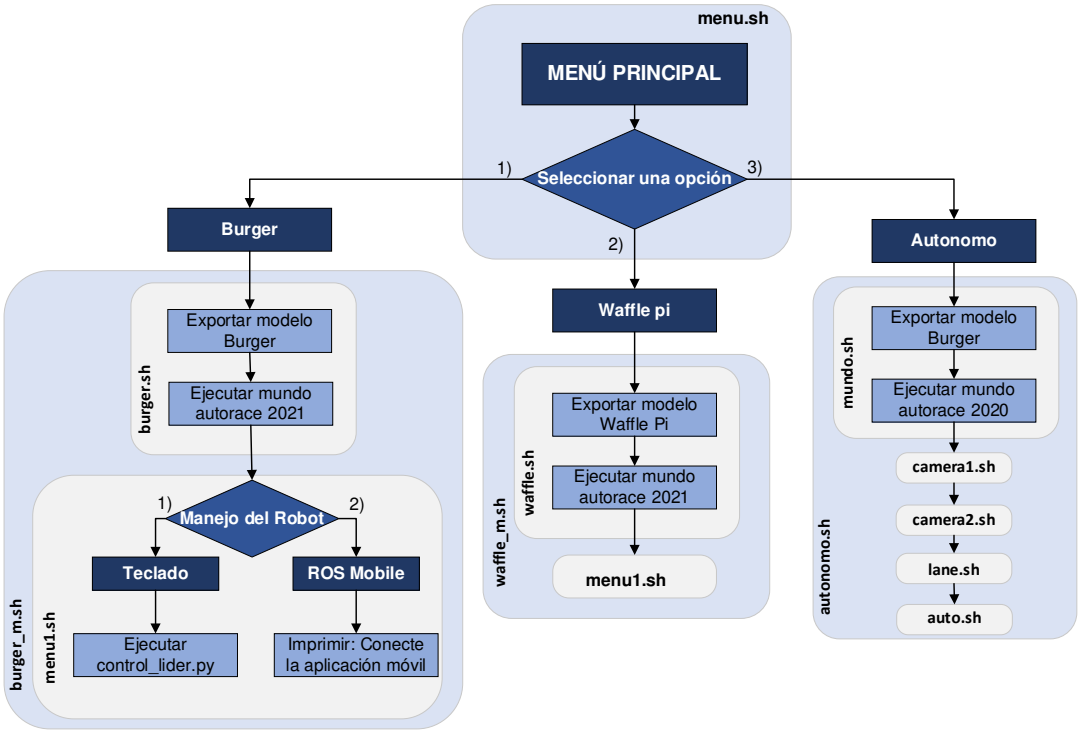


Figura 2.35. Diagrama de flujo del menú desarrollado para tomo 1 [Fuente propia]

En el caso de seleccionar el modelo Burger o Waffle Pi, se ejecuta un archivo model.sh que exporta el modelo del robot y lanza un archivo .launch que contiene a la pista modificada turtlebot3_autorace_2021 y que también realiza el 'spawn' del robot exportado. Seguido a esto se ejecuta el archivo menu1.sh que permite escoger la forma de conducción de la plataforma.

En el caso de seleccionar la opción de Autónomo, se ejecutan varios archivos .sh que a su vez contienen archivos .launch necesarios para el funcionamiento de la aplicación de conducción autónoma. El manual de usuario correspondiente se puede encontrar en Anexo VI.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

Para comprobar el manejo de la plataforma TurtleBot3 por un usuario utilizando el teclado del computador y la aplicación de teléfono ROS Mobile, se procede a realizar pruebas con los modelos Burger y Waffle Pi, a través de la pista modificada del turtlebot3_autorace_2020 presentada en la Sección 2.2.3, y que se puede observar en las Figuras 3.1 (a) con el modelo Burger y 3.1 (b) con el modelo Waffle Pi. Las pruebas y resultados de esta sección se lo puede observar en los videos del ANEXO VII.

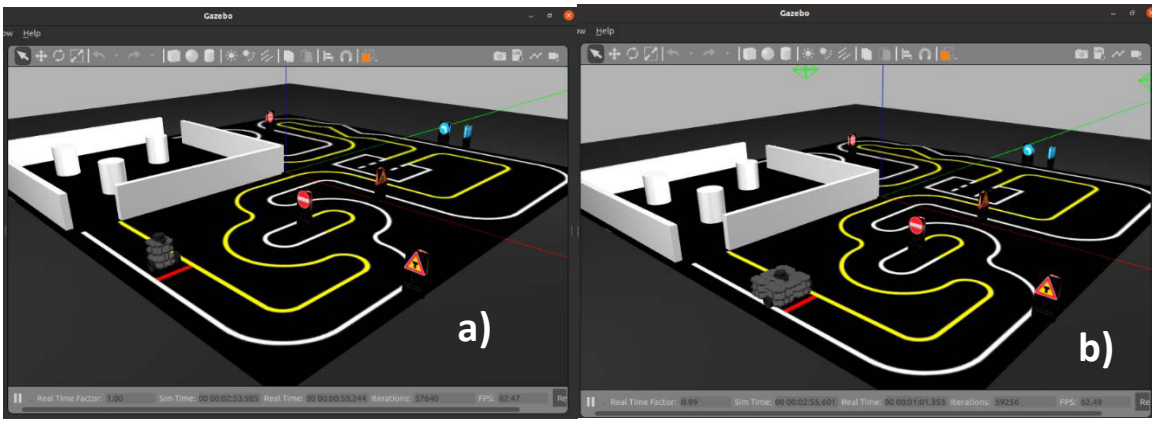


Figura 3.1. Mapa turtlebot3_autorace_2020 en el entorno de simulación Gazebo con (a) modelo Burger y (b) modelo Waffle Pi

Esta pista contiene curvas cerradas que obligan al usuario, en determinados momentos, a realizar movimientos bruscos con el robot siendo un excelente escenario para realizar pruebas.

Para poder comparar los resultados obtenidos en las diferentes pruebas, cuando se hace uso del teclado para la operación del robot TurtleBot3 se lleva al robot a una velocidad lineal de 0.05m/s y posterior a esto únicamente se varía la velocidad angular para completar la ruta marcada. En la aplicación de teléfono ROS Mobile no es posible preconfigurar una velocidad lineal fija. Sin embargo, se trató de llevar a cabo la prueba emulando lo más posible las condiciones de la conducción por teclado para poder realizar una comparación justa.

3.1.1 PRUEBA USUARIO 1

Al manejar la plataforma TurtleBot3 en sus modelos Burger y Waffle Pi a través de la pista turtlebot3_autorace_2020 mediante el teclado del computador y la aplicación de teléfono

ROS Mobile, se obtiene la Figura 3.2. En esta se pueden observar las distintas trayectorias realizadas por el manejo del usuario comparadas con la trayectoria realizada por el algoritmo de control de conducción autónoma, el cual se asume que representa una conducción adecuada a través de la pista.

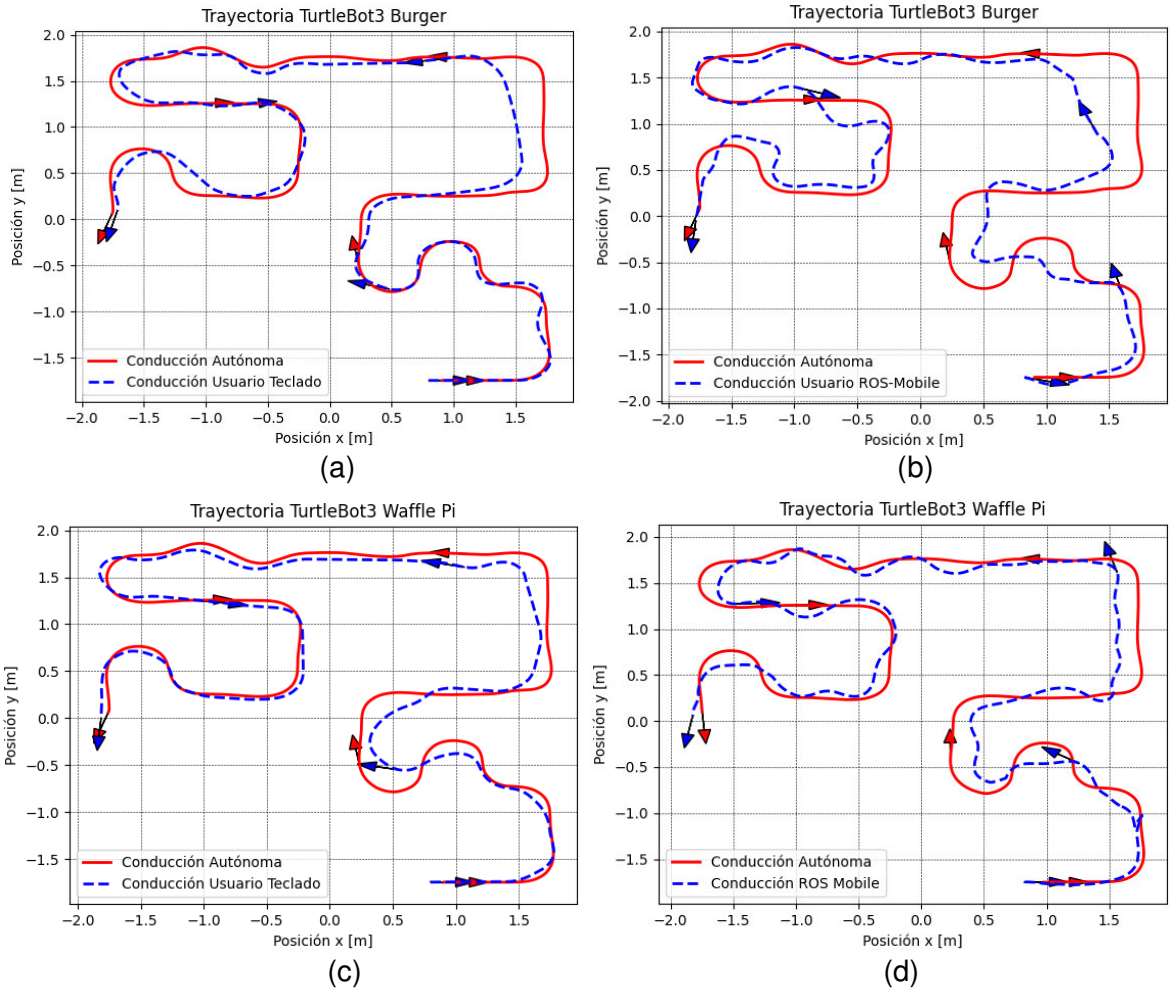


Figura 3.2. Trayectorias realizadas por la conducción del usuario 1

En la Figura 3.2 se observa de manera gráfica que tan bien condujo el usuario a través de la pista con el modelo Burger haciendo uso del teclado del computador y la aplicación ROS Mobile, como se muestra en la Figura 3.2 (a) y 3.2 (b) respectivamente. Asimismo, se puede observar la conducción realizada con el modelo Waffle Pi usando el teclado (Figura 3.2 (c)) y la aplicación ROS Mobile (Figura 3.2(d)).

En términos generales, la conducción mediante la aplicación ROS Mobile presentó mayor dificultad que la conducción a través de teclado, ya que para el caso de ROS Mobile es muy difícil mantener una velocidad lineal constante, por la dinámica propia del Joystick virtual, y en determinados casos fue necesario detener al robot para corregir su trayectoria. Asimismo, al usar la aplicación ROS Mobile se debe considerar que se necesita una buena

conexión a internet, ya que, si no se dispone de la misma, los comandos de velocidad que se envíen desde el teléfono llegarán con retardo al robot o en el peor de los casos no le llegarán.

Por otro lado, al comparar la conducción entre los modelos de la Plataforma TurtleBot3 se determina que el modelo Waffle Pi tiene mayor restricción de giro debido a las dimensiones físicas de este modelo, como se muestra en la Sección 1.4.2.1.

Se debe considerar que las pruebas obtenidas en esta sección al ser realizadas por un humano están sujetas a diversos factores como la experiencia que se tenga con la plataforma. Por esta razón, en la siguiente sección se muestra la conducción realizada por un usuario externo que no tenía gran experiencia manejando la plataforma TurtleBot3.

3.1.2 PRUEBAS USUARIO 2

Bajo las mismas condiciones del usuario 1, es decir velocidad lineal 0.05m/s para la conducción por teclado, y realizando las mismas pruebas, se obtiene la Figura 3.3.

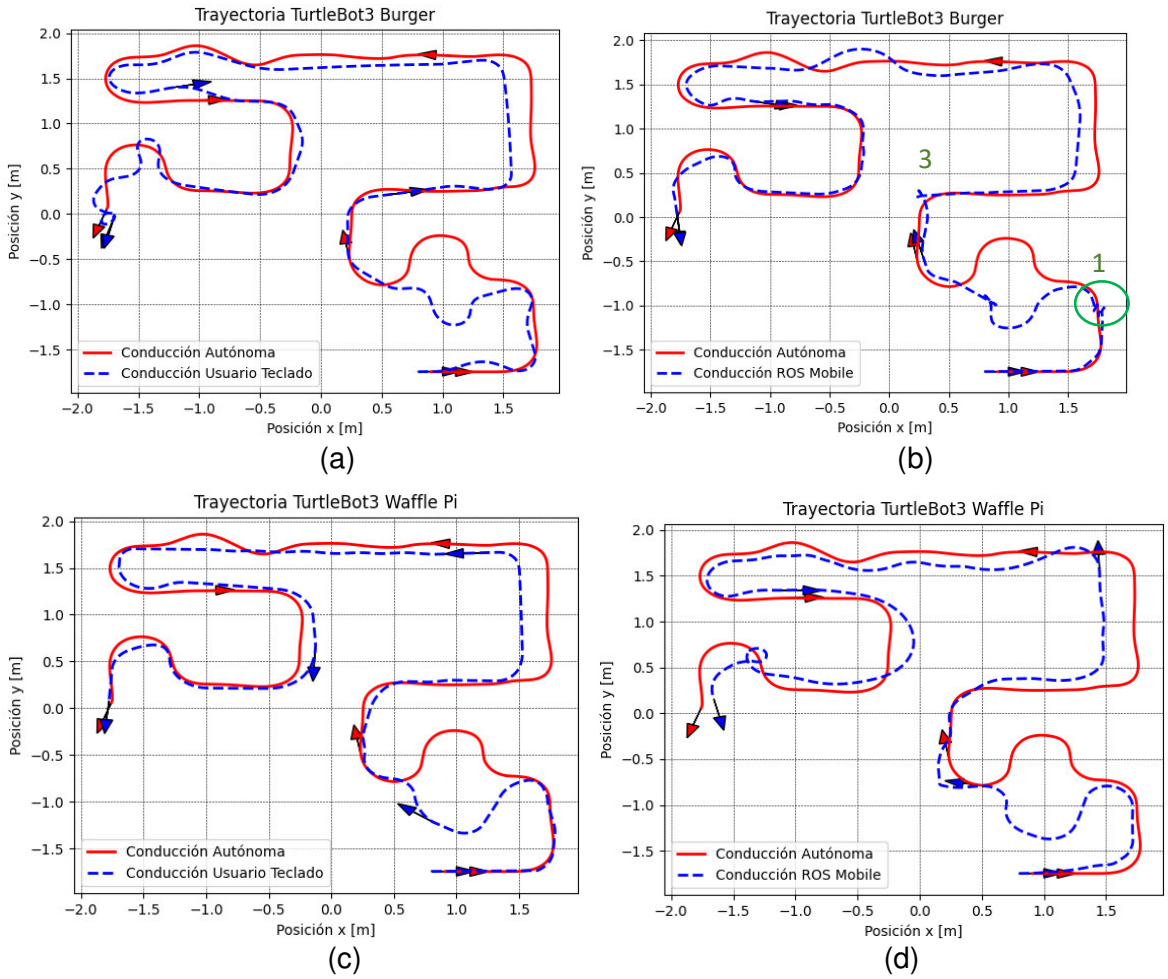


Figura 3.3. Trayectorias realizadas por la conducción del usuario 2

Los resultados que se observan en la Figura 3.3 son similares en la conducción por teclado a los obtenidos por el Usuario 1. Sin embargo, es preciso señalar que en la Figura 3.3 (b), correspondiente a la conducción del modelo Burger utilizando la aplicación ROS Mobile, se ha encerrado en un círculo el lugar donde el Usuario 2 colisionó con una señal de tránsito de la pista, a causa de una mala conducción.

Ciertamente resulta difícil distinguir el choque presentado en la Figura 3.3 (b) por lo que, para presentar esto de una manera más gráfica, se procede a sacar capturas de pantalla correspondientes a la conducción del usuario 2 con el modelo Burger mediante ROS Mobile, obteniendo la secuencia mostrada en la Figura 3.4.

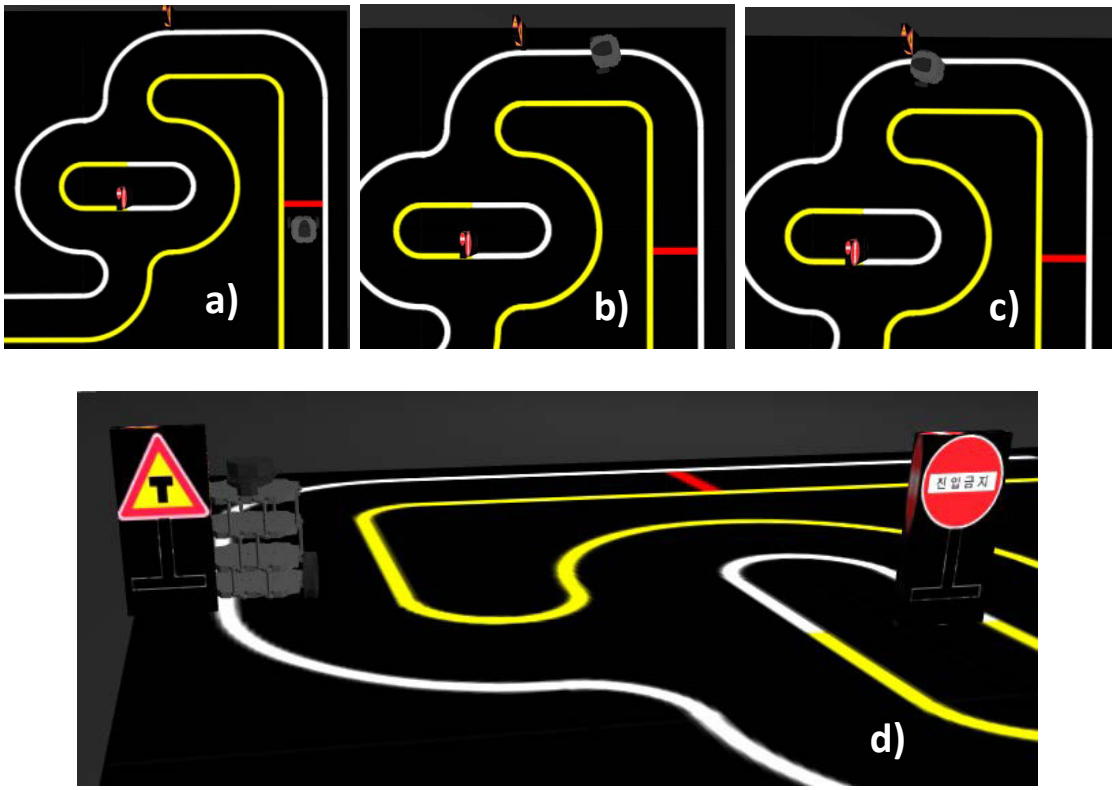


Figura 3.4. Choque de la plataforma TurtleBot3 en Gazebo [Fuente Propia]

En la Figura 3.4 (a) el robot se encuentra en la posición de partida, luego en la Figura 3.4 (b) se puede observar que el robot empieza a perder pista después de la primera curva realizada, para finalmente en la Figura 3.4 (c) colisionar contra una señal de tránsito. La colisión se puede observar de una mejor forma en la Figura 3.4 (d), donde se observa que el modelo Burger ha chocado con la rueda derecha haciendo que ya no pueda avanzar a pesar de tener velocidad lineal y angular obligando al usuario a retroceder para evitar el obstáculo.

Cabe indicar, que no se contabilizan los errores generados por la conducción de la plataforma TurtleBot3 por un usuario, ya que no se podía realizar una comparación justa puesto que en las pruebas realizadas se tienen un número diferente de datos que depende del tiempo que demora el usuario en recorrer la pista. Además, el objetivo del trabajo es proporcionar la plataforma para el manejo de un robot TurtleBot3 y que sirve de base para la implementación del sistema multi-agente con control de formación, que se detalla en el Tomo 2 [1]. Lo referente a que un usuario pueda conducir la plataforma, presentado en este tomo, es más por fines explicativos del entorno de simulación y su versatilidad, así como los pasos requeridos para contar con la escena conformada por cualquiera de los dos modelos disponibles del robot TurtleBot3 virtual.

3.2 CONCLUSIONES

- Gazebo es un simulador que considera las características cinemáticas y dinámicas de un robot, además de tener la capacidad de simular sus sensores, sumado a esto su facilidad de conexión con ROS lo hacen ideal para la simulación de plataformas robóticas tan necesarios a la hora de investigación y desarrollo de algoritmos de control, como el que se presenta en el siguiente tomo.
- El uso del entorno ROS/Gazebo permite la simulación de una gran cantidad de aplicaciones robóticas haciendo uso de varias plataformas como la del TurtleBot3, que permitieron en este trabajo implementar la manipulación del movimiento de sus modelos virtuales incluyendo sensores y actuadores.
- Se concluye que, utilizando el entorno de Gazebo, a través de código se pueden crear nuevos escenarios que difieran del original en aspectos sencillos como el tipo de piso o el cielo como fue el caso del mapa libre de obstáculos creado; así también permite la creación, modificación y uso de mapas más complejos que incluyan varios elementos que pueden ser exportados desde repositorios, como es el caso de la pista mostrada en la sección de resultados
- La programación necesaria para que un usuario pueda dirigir al robot virtual TurtleBot3 se simplifica al utilizar la herramienta ROS, la cual por su estructura de mensajería de nodos y tópicos permite fácilmente subscribirse y publicar comandos de velocidad, controlando así directamente la velocidad lineal y angular del robot y por ende su movimiento.

- De los resultados obtenidos en las pruebas de manejo de la Plataforma TurtleBot3 a través de una pista de pruebas, se determina que se tiene una mejor maniobrabilidad al utilizar el teclado del computador que con la aplicación ROS Mobile; de manera similar, se tiene mayor facilidad para realizar giros con el modelo Burger debido a que es más ligero y tiene un menor radio que el modelo Waffle Pi.

3.3 RECOMENDACIONES

- Como se explicó la transferencia de datos entre nodos en ROS se puede realizar a través de tópicos, servicios y acciones; sin embargo, si no se tiene mucha experiencia en la plataforma se recomienda hacerlo a través de tópicos, ya que es el método más sencillo.

- Los nodos en ROS se pueden programar en varios lenguajes como C++ y Python; sin embargo, si no se conoce ninguno de estos, se recomienda aprender Python ya que, aparte de ser uno de los lenguajes de programación más usados, su librería rospy facilita la creación de nodos publicadores y subscriptores

- Se recomienda revisar los tutoriales disponibles de la plataforma TurtleBot3 disponibles en la página oficial

- Para la simulación de comportamientos robóticos es recomendable utilizar software capaz de considerar las condiciones no únicamente cinemáticas sino también dinámicas de un robot como es el caso de Gazebo; ya que es una forma posible en la se puede observar el frenado brusco, o choques que se podrían ocasionar al implementar los algoritmos de control diseñados.

- Se debe considerar el estudio de las restricciones no holonómicas para el desarrollo del esquema de control líder seguidor presentado en el siguiente tomo [1], así como también las ecuaciones cinemáticas del robot móvil de tracción diferencial, ya que estas características también determinarán las restricciones de la formación.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] Gabriela Romero, “Diseño, control y simulación, mediante el uso del entorno ros/gazebo, de un sistema multi-agente conformado por plataformas turtlebots enfocado a mantener una formación líder seguidor (Tomo 2)” Quito, 2022.

[2] “Robótica/Historia de la robótica - Wikilibros.” https://es.wikibooks.org/wiki/Rob%C3%B3tica/Historia_de_la_rob%C3%B3tica (accessed Jan. 22, 2022).

[3] “Los robots más exitosos de la industria - Master en Industria 4.0 : Universidad de Alcalá - Madrid.” https://www.masterindustria40.com/robots-master-robotica/ (accessed Jan. 22, 2022).

[4] C. Camilo and C. Castañeda, “Ros-gazebo. una valiosa Herramienta de Vanguardia para el Desarrollo de la Robótica,” *Publicaciones e Investigación*, vol. 10, pp. 145–160, Mar. 2016, doi: 10.22490/25394088.1593.

[5] “TurtleBot3.” https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed Dec. 09, 2021).

[6] F. Rubio, F. Valero, and C. Llopis-Albert, “A review of mobile robots: Concepts, methods, theoretical framework, and applications:,” *https://doi.org/10.1177/1729881419839596*, vol. 16, no. 2, Jan. 2019, doi: 10.1177/1729881419839596.

[7] M. A. R. Serrano and E. A. Bricaire, “Coordinación de movimiento para sistemas multiagente heterogéneos,” *Memorias del XVI Congreso Latinoamericano de Control Automático*, 2014, [Online]. Available: http://amca.mx/memorias/amca2014/media/files/0209.pdf

[8] U. El, E. Líder-Seguidor, P. Anthony, and A. Purisaca, “Control de robots móviles autónomos en formación usando el esquema líder-seguidor,” *Pontificia Universidad Católica del Perú*, May 2021, Accessed: Jan. 22, 2022. [Online]. Available: https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/18953

[9] H. Cruz, “Monografia robot movil.” [Online]. Available: https://www.academia.edu/11182541/Monografia_robot_movil

[10] D. Daniel and C. Salas, “Teleoperación con realimentación de fuerza aplicada a un robot móvil con restricción no-holonómica,” Nov. 2016, Accessed: Jan. 22, 2022. [Online]. Available: http://bibdigital.epn.edu.ec/handle/15000/16876

[11] D. E. H. Sánchez, J. Ramón, E. Cuenca, C. C. Sánchez, J. Fernando, and R. Cortés, “‘Diseño, construcción y modelo dinámico de un robot móvil de tracción diferencial aplicado al seguimiento de trayectorias,’” 2017.

[12] “(PDF) Modelamiento cinemático y odométrico de robots móviles: aspectos matemáticos.” https://www.researchgate.net/publication/233603982_Modelamiento_cinematico_y_odometrico_de_robots_moviles_aspectos_matematicos (accessed Jan. 22, 2022).

[13] S. Boucher Research Assistantship Report and R. Canosa, “Obstacle Detection and Avoidance Using TurtleBot Platform and XBox Kinect,” 2011.

[14] “About - TurtleBot.” https://www.turtlebot.com/about/ (accessed Jan. 22, 2022).

[15] R. Amsters and P. Slaets, “Turtlebot 3 as a Robotics Education Platform,” *Advances in Intelligent Systems and Computing*, vol. 1023, pp. 170–181, Apr. 2019, doi: 10.1007/978-3-030-26945-6_16.

[16] “TurtleBot3.” https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation (accessed Jan. 27, 2022).

[17] “ROS: The ROS Ecosystem,” *Open Robotics*. 2021. [Online]. Available: https://www.ros.org/blog/ecosystem/

[18] J. M. O’kane, “A Gentle Introduction to ROS”, Accessed: Dec. 28, 2021. [Online]. Available: http://www.cse.sc.edu/~jokane

[19] “Distributions - ROS Wiki.” http://wiki.ros.org/Distributions (accessed Jan. 22, 2022).

[20] C. Fairchild and T. L. Harman, “ROS robotics by example : bring life to your robot using ROS robotic applications”, Accessed: Jan. 22, 2022. [Online]. Available: https://www.researchgate.net/publication/312279333_ROS_ROBOTICS_BY_EXAMPLE

[21] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, “Rosbridge: ROS for Non-ROS Users,” *Springer Tracts in Advanced Robotics*, vol. 100, pp. 493–504, 2017, doi: 10.1007/978-3-319-29363-9_28.

[22] J. Collins, S. Chand, A. Vanderkop, and D. Howard, “A review of physics simulators for robotic applications,” *IEEE Access*, vol. 9, pp. 51416–51431, 2021, doi: 10.1109/ACCESS.2021.3068769.

[23] “Gazebo.” http://gazebosim.org/ (accessed Dec. 09, 2021).

[24] A. Nicolás, I. Sanjuán, T. Directores, R. Aragüés, M. Gonzalo, and L. Nicolás, “Simulation of soft bodies in ROS and Gazebo,” 2021.

[25] Giner Joaquín, “Programación en Bash – Programación#Bioinformática.” https://blogs.upm.es/estudiaciencia/programacion-en-bash/ (accessed Feb. 10, 2022).

[26] “ROS/Tutorials - ROS Wiki.” http://wiki.ros.org/ROS/Tutorials (accessed Jan. 20, 2022).

[27] “GitHub - raaptoy/Leader_Follower_Formation_TurtleBot3 at master.” https://github.com/raaptoy/Leader_Follower_Formation_TurtleBot3/tree/master (accessed Jan. 31, 2022).

[28] “rospy - ROS Wiki.” http://wiki.ros.org/rospy (accessed Jan. 04, 2022).

[29] P. YoonSeok, C. HanCheol, and L. TaeHoon, *ROS Robot Programming*. Seoul: ROBOTIS Co.,Ltd., 2017.

[30] “Gazebo : Tutorial : Beginner: GUI.” https://gazebosim.org/tutorials?cat=guided_b&tut=guided_b2 (accessed Jan. 22, 2022).

[31] J. Pablo Rojas Bustos, “ENTORNO VIRTUAL PARA LA SIMULACIÓN DE UN ROBOT MÓVIL SOBRE EL FRAMEWORK ROS UNIVERSIDAD PILOTO DE COLOMBIA FACULTAD INGENERIA MECATRÓNICA,” 2015.

[32] “Multiple robots simulation and navigation - ROS Answers: Open Source Q&A Forum.” https://answers.ros.org/question/41433/multiple-robots-simulation-and-navigation/ (accessed Jan. 24, 2022).

[33] by H. Alemi Ardakani and T. J. Bridges, “Review of the 3-2-1 Euler Angles: a yaw-pitch-roll sequence”.

[34] “transformations — tf 0.1.0 documentation.” http://docs.ros.org/en/jade/api/tf/html/python/transformations.html (accessed Feb. 17, 2022).

[35] “Packages - ROS Wiki.” http://wiki.ros.org/Packages (accessed Jan. 04, 2022).

5 ANEXOS

ANEXO I. Instalación de ROS y Gazebo

ANEXO II. Arquitectura de ROS

ANEXO III. Información general sobre organización de archivos

ANEXO IV. Repositorio digital

ANEXO V. Diagrama de nodos Autorace

ANEXO VI. Manual de Usuario

ANEXO VII. Link a Videos demostrativos

En los Anexos para referenciar las páginas web se hace uso de hipervínculos los cuales se distinguen por estar subrayados y con color celeste.

ANEXO I. Instalación ROS-GAZEBO

Instalación ROS

Para la instalación de Ubuntu 20.04 se lo puede hacer mediante el tutorial que se presenta en Instalación Ubuntu 20.04. Una vez instalado el sistema operativo se realizará la instalación de ROS con la distribución Noetic, siguiendo los pasos indicados en las páginas oficiales de ROS Instalación ROS Noetic.

Para la familiarización con el entorno se recomienda seguir los tutoriales ya sea de la página oficial ROS Tutorials , o mediante la plataforma The Construct, que ofrecen tutoriales de robótica junto con ROS y Gazebo. Sin embargo, se debe tomar en cuenta que no todos los cursos que ofrecen son gratuitos.

Al terminar de instalar los programas junto con realizar los tutoriales sugeridos, es posible crear y controlar desde cero un robot.

Instalación Gazebo

En este trabajo se utiliza la versión 11 del software Gazebo, ya que esta versión esta oficialmente integrada y soportada para varias distribuciones de ROS. Además, cuenta con una gran cantidad de tutoriales y mejoras respecto a otras versiones Instalación Gazebo .

Para proceder con la instalación del simulador es necesario conocer los requisitos del sistema:

- computadora con unidad de procesamiento gráfico o GPU (Se recomienda el uso de tarjetas Nvidia).
- CPU con al menos Intel I5.
- En cuanto a espacio libre se recomienda por lo menos 500 MB de espacio libre en el disco.
- Ubuntu Trusty o superior.

Para su instalación se recomienda seguir las instrucciones de la página oficial de Gazebo Tutorials . Una vez instalado, se lo podrá utilizar presionando Alt+F2, luego escribir “Gazebo” y presionar Entrar. Como se ha mencionado anteriormente se recomienda seguir los tutoriales o cursos de las páginas oficiales de ROS, Gazebo y The Construct.

ANEXO II. Arquitectura de ROS

Nivel de Filesystem (archivos de sistema): Este nivel está compuesto de herramientas de ROS para el manejo del código fuente, construcción de instrucciones y definición de mensajes, la organización de estos archivos de sistema se puede observar en la Figura II.1

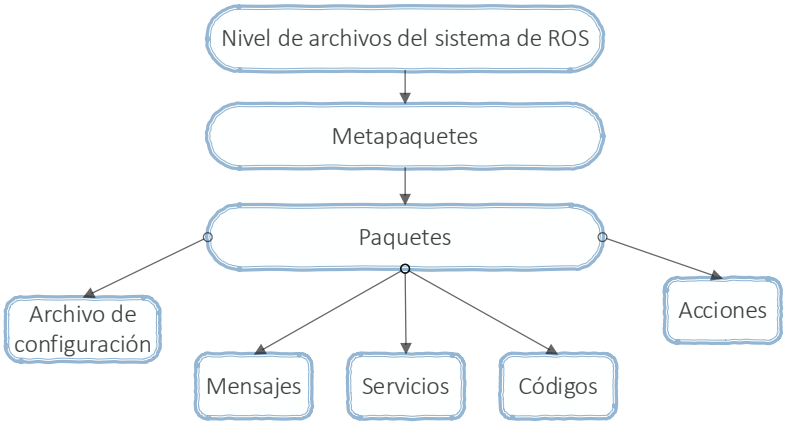


Figura II.1 Nivel de archivos del sistema de ROS

Nivel de computation graph (gráfico de cálculo): Este nivel abarca el intercambio de mensajes entre nodos o la red conocida como Peer2Peer y sus elementos se detallan en la Figura II.2.

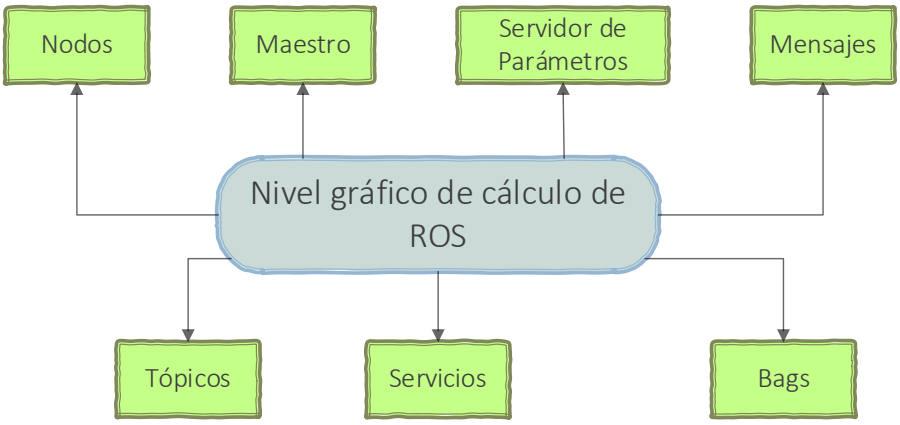


Figura II.2 Nivel computacional gráfico de ROS

Nivel de comunidad: ROS al ser un software de código abierto cuenta con varios servicios para toda la comunidad como: repositorios de código, foro oficial ROS-Wiki el cual proporciona documentación y tutoriales, un sitio de preguntas y respuestas ROS Answers y un blog de ros.org que proporciona actualizaciones periódicas acerca de las distribuciones o versiones de ROS.

ANEXO III. ESPACIO DE TRABAJO DE ROS

Src: todo el trabajo se lo realiza dentro de esta carpeta la cual contiene el código fuente. Este es el espacio en el cual se puede clonar, crear y editar código para los paquetes que se desean construir. En este caso, el paquete control_lider mostrado en la Figura III.1.

Build: este es el lugar en el cual se llama al CMake para construir los paquetes en el espacio fuente. Así también, se puede encontrar archivos intermedios y de Cache. Esta carpeta no se la va a manipular.

Devel: aquí se encuentran todos los objetos construidos y de igual manera no se manipula esta carpeta para el desarrollo de la aplicación del presente trabajo.

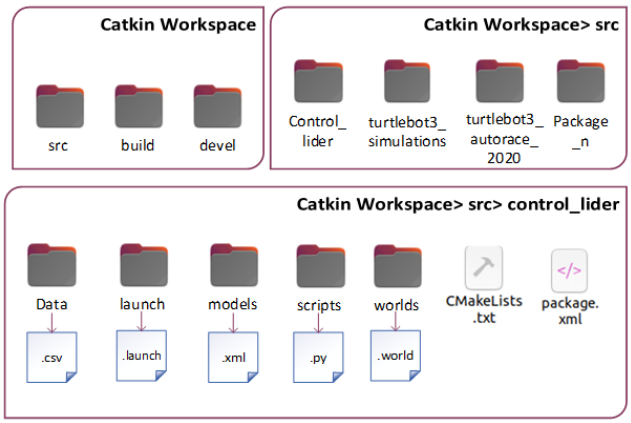


Figura III.1 Organización de carpetas en ROS

Paquetes de ROS: Se conoce que la organización del software ROS se da a través de paquetes. Los paquetes contienen los servicios, mensajes, nodos y archivos de configuración. Su objetivo es proveer un software que pueda ser fácilmente reutilizado, es así como existen paquetes de uso público [35] como el turtlebot3_simulations o turtlebot3_autorace_2020, mostrados en la Figura III.1. La creación de paquetes se puede encontrar en los tutoriales de ROS [26]. Junto con la creación del paquete se crean dos archivos el CMakeLists.txt y el package.xml.

- **CMakeLists.txt:** Es un archivo que se lo debe configurar previo a la creación de servicios y acciones.
- **Package.xml:** Es el encargado de definir las propiedades del paquete, como por ejemplo autor, número de versión, nombre del paquete y otras dependencias.

ANEXO IV. Repositorio digital:

El anexo presentado en esta sección se puede encontrar de igual manera en el segundo tomo del trabajo desarrollado en conjunto.

Dado que se ocupa softwares de código abierto como ROS y Gazebo. El proyecto desarrollado en conjunto por los autores Rodrigo Ayala y Gabriela Romero, se encuentra en el repositorio digital GitHub.

Para acceder a este repositorio se debe dirigir al siguiente link dando Ctrl+click, sobre las letras que están subrayadas y en azul:

Link: GitHub - raaptoy/Leader_Follower_Formation_TurtleBot3 at master

Al entrar al link, se encontrará con la Figura IV.1:

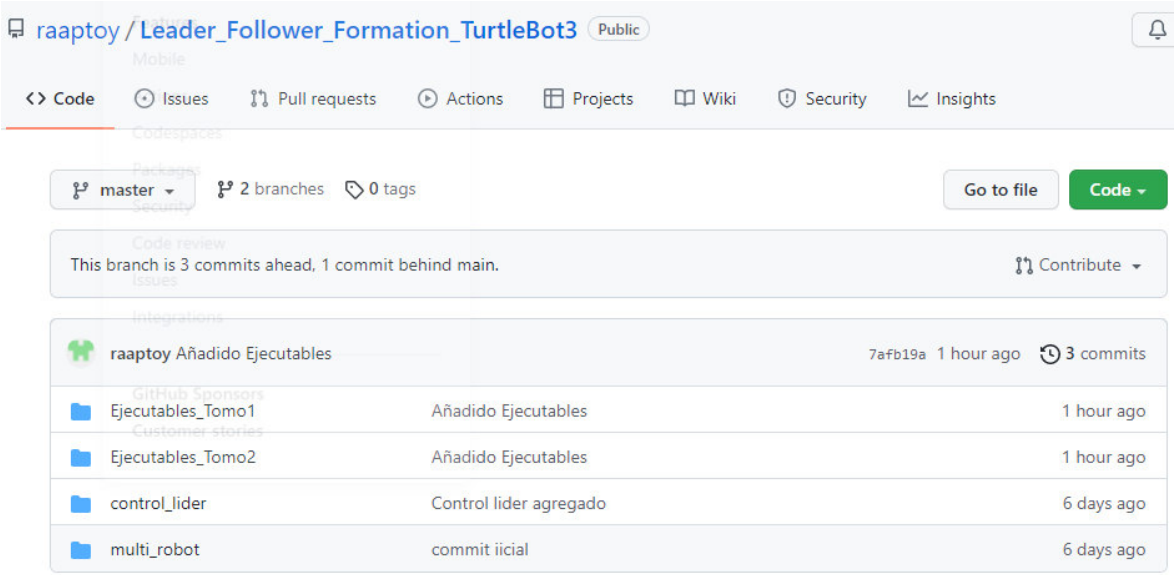


Figura IV.1. Imagen general del repositorio desarrollado

En este repositorio se encuentra los paquetes desarrollados para el primer y segundo tomo, así también como los ejecutables.

Primer Tomo:

- Paquete: control_lider
- Ejecutables: Ejecutables_Tomo1

Segundo Tomo:

- Paquete: multi_robot
- Ejecutables: Ejecutables_Tomo2

Los paquetes almacenan todas las carpetas necesarias para la ejecución del proyecto, tal como se observa en la Figura IV.2. En un inicio se tienen las mismas carpetas para los dos paquetes. Se tienen: Data, launch, models, scripts, worlds, CMakeLists.txt, package.xml.

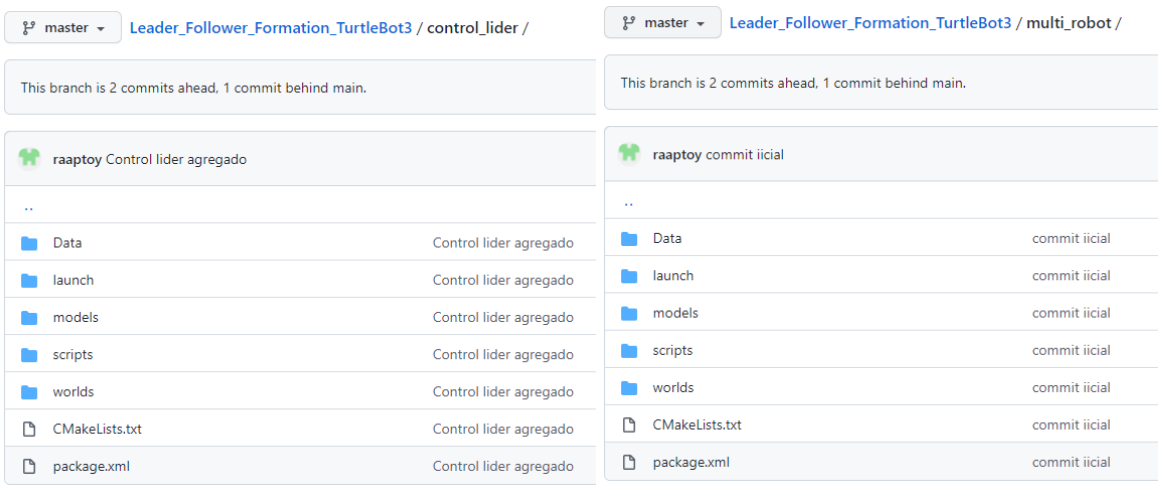


Figura IV.2. Carpetas internas de los paquetes del tomo 1 y tomo2

En el repositorio se puede entrar a cada una de las carpetas para ver que contienen por ejemplo en las carpetas Data, de cada uno de los tomos almacenan archivos .csv utilizados para plotear las gráficas de trayectoria, de ángulo y distancia. La carpeta scripts almacenan los códigos generados para cada tomo, como se puede observar en la Figura IV.3:

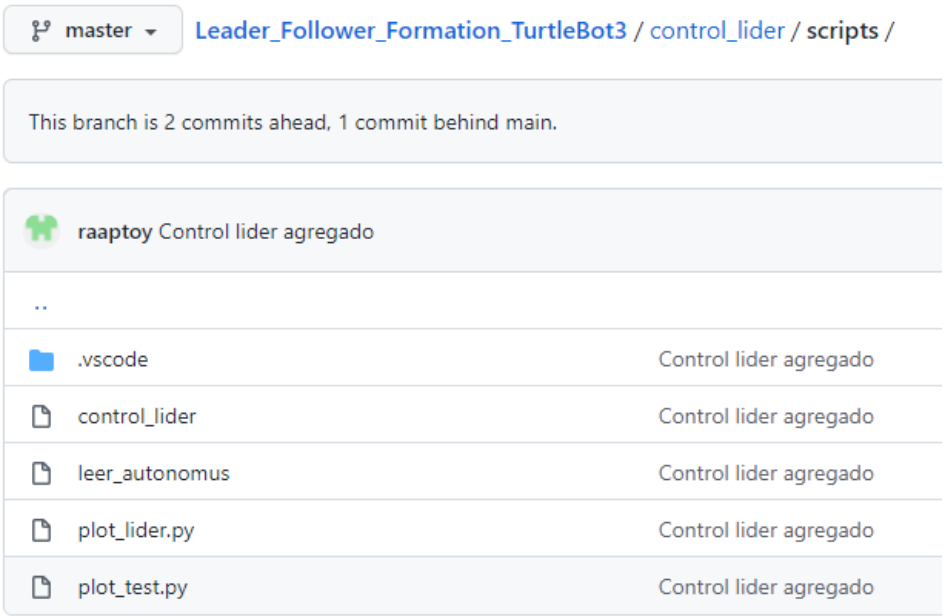


Figura IV.3. Scripts desarrollados para el tomo1 en el paquete control_lider

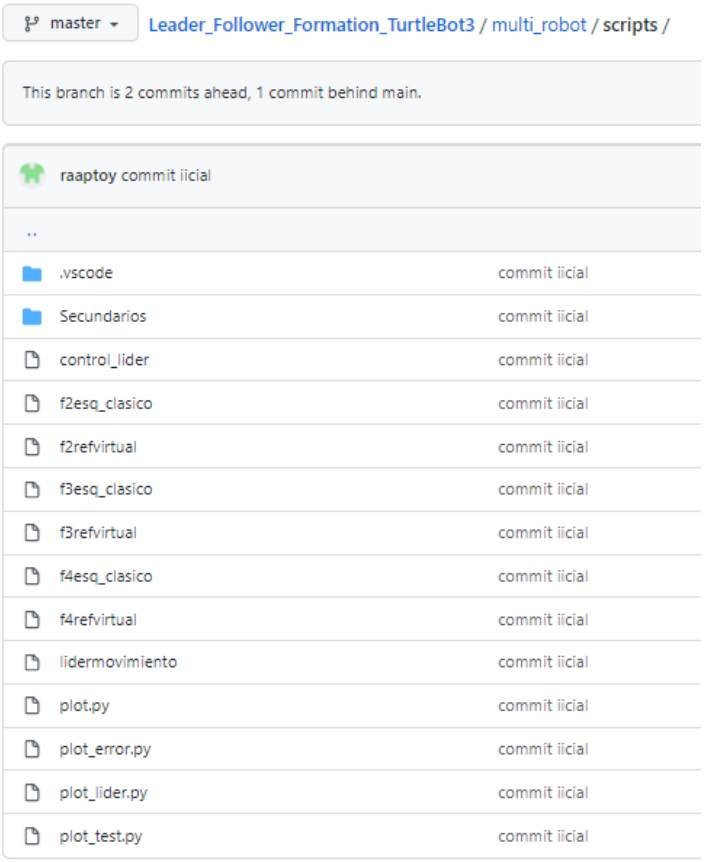


Figura IV.4. Scripts desarrollados para el tomo1 en el paquete multi_robot

Finalmente, en la carpeta de ejecutables, se encuentran todos los archivos bash (.sh) que ayudan a ejecutar los proyectos, se los puede observar en las Figuras IV.4 y Figura IV.5.

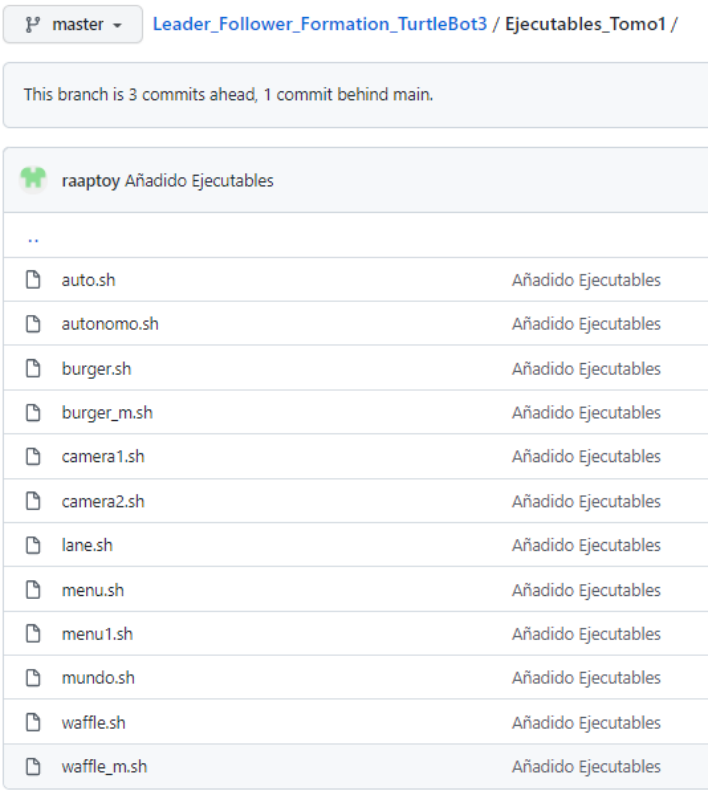


Figura IV.5. Archivos ejecutables desarrollados para el tomo 1

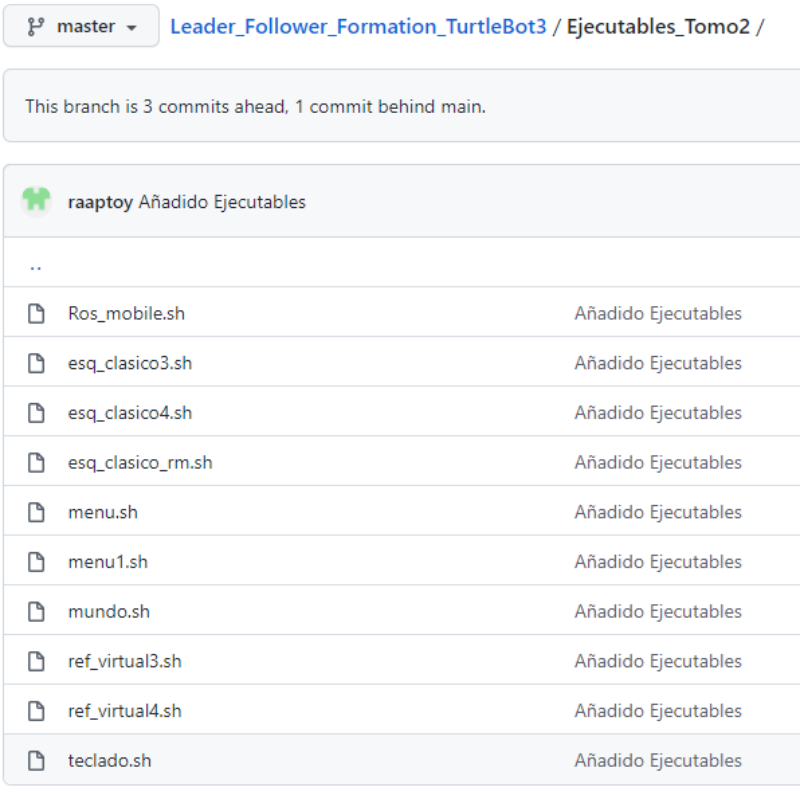


Figura IV.6. Archivos ejecutables desarrollados para el tomo 2

ANEXO V. Diagrama de nodos Autorace

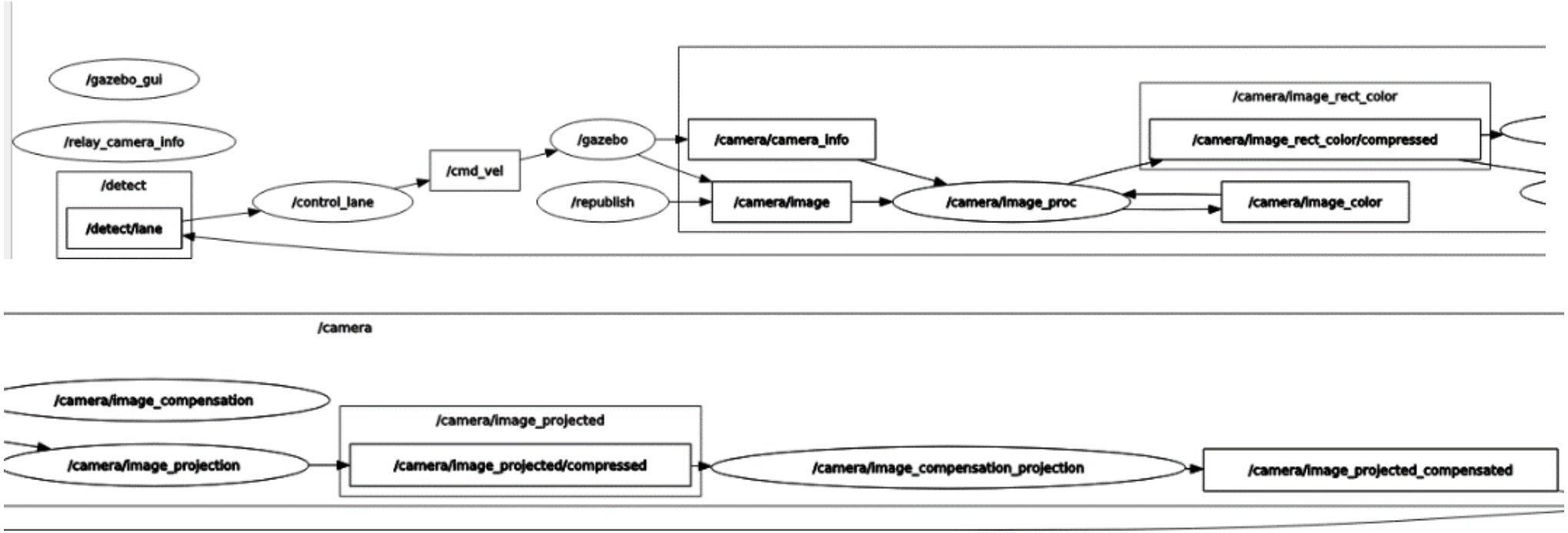


Figura V.1 Diagrama de nodos y tópicos del movimiento del Robot Turtlebot3 Burger con conducción autónomo

ANEXO VI. Manual de Usuario:

El anexo presentado en esta sección se puede encontrar de igual manera en el segundo tomo del trabajo desarrollado en conjunto.

El manual de usuario presentado a continuación servirá de ayuda para poder ejecutar el trabajo realizado en el primer y segundo tomo del trabajo desarrollado en conjunto.

Requisitos del sistema

- Sistema operativo: Linux Ubuntu 20.04 LTS
- ROS: Noetic Ninjemys
- Gazebo: V11.0.0

Paquete desarrollado en ROS

Con todos los requisitos del sistema, para poder ejecutar el programa desarrollado en su computador se recomienda clonar el paquete de ROS del repositorio:

repositorio: GitHub - raaptoy/Leader_Follower_Formation_TurtleBot3 at master

En el repositorio se encontrará con la carpeta multi_robot, esta deberá copiarse en catkin workspace, con lo que deberá tener el siguiente path:

/home/USER/catkin_ws/src/multi_robot

Se entiende por **USER** al nombre del usuario que se ha puesto en el sistema operativo Linux.

Una vez copiado el paquete, es necesario ejecutar los siguientes mandos, dentro de la terminal:

```
roscd
catkin make
```

Ejecución

Para asegurar que todos los datos se guarden es necesario modificar el usuario o USER del directorio, dentro de los scripts de Python. Así también como los paths dentro de los archivos de la carpeta bash. Como, por ejemplo, en un archivo bash el dato que se debe cambiar se lo subraya en color amarillo y el user está subrayado en color rojo, esto se lo puede observar en la Figura VI.1. y en la Figura VI.2

master ▾ Leader_Follower_Formation_TurtleBot3 / Ejecutables_Tomo2 / ref_virtual4.sh

raaptoy Añadido Ejecutables

1 contributor

16 lines (15 sloc) | 699 Bytes

```
1  #!/bin/sh
2  roscd
3  source ./devel/setup.bash
4  export TURTLEBOT3_MODEL=burger
5  #Lanzo Robots Diamante
6  roslaunch multi_robot followers4.launch
7  #Lanzo el control del robot 1 (Teleoperación)
8  gnome-terminal --tab --title="Control Lider" --command="rosrun multi_robot lidermovimiento 'cd /etc; ls; $SHELL'"
9  #Follower2
10 gnome-terminal --tab --title="Seguidor 2" --command="rosrun multi_robot f2refvirtual  'cd /etc; ls; $SHELL'"
11 #Follower3
12 gnome-terminal --tab --title="Seguidor 3" --command="rosrun multi_robot f3refvirtual  'cd /etc; ls; $SHELL'"
13 #Follower4
14 gnome-terminal --tab --title="Seguidor 4" --command="rosrun multi_robot f4refvirtual  'cd /etc; ls; $SHELL'"
15 cd /home/rodrigo/Escritorio/Ejecutables
16
```

Figura VI.1. Archivo bash que contiene directorio a cambiar

master ▾ Leader_Follower_Formation_TurtleBot3 / multi_robot / scripts / f2esq_clasico

raaptoy commit iicial

1 contributor

Executable File | 199 lines (162 sloc) | 7.45 KB

```
1  #!/usr/bin/env python3
2  import time
3  import sys
4  import rospy
5  from geometry_msgs.msg import Twist
6  from nav_msgs.msg import Odometry
7  from threading import Thread
8
9  # Other Imports
10 import numpy as np
11 import math as m
12 from tf.transformations import euler_from_quaternion, quaternion_from_euler
13
14 max_linear_velocity = 0.2
15 max_angular_velocity = 2.8
16
17 LOG_FILE_DIR = '/home/rodrigo/catkin_ws/src/multi_robot/Data'
```

Figura VI.2. Archivo bash que contiene directorio a cambiar

Una vez se hayan hecho todos los cambios realizados. En la carpeta Ejecutables del tomo 1 o tomo 2 dar click derecho, abrir en una terminal. Seguidamente se seguirán los siguientes pasos

Tomo 1:

Al abrir la carpeta Ejecutables_Tomo1, en la terminal generada dar el comando

```
bash menu.sh
```

Se podrá visualizar el menú principal como se ve en la Figura VI.3

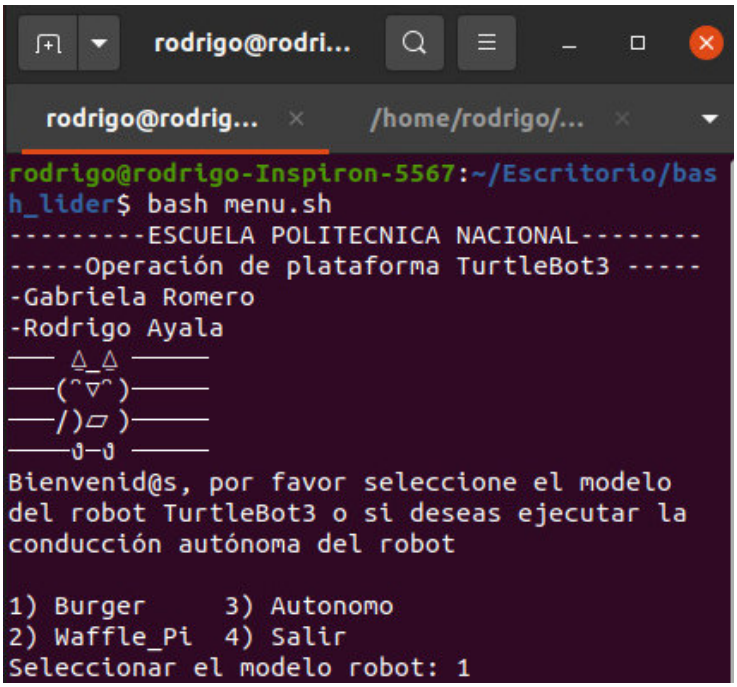


Figura VI.3. Menú principal del tomo 1

Al seleccionar la opción Burger o Waffle_Pi se despliega el menú secundario mostrado en la Figura VI.4, de aquí al escoger la opción Teclado se ejecutará el script del control de movimiento del robot a través del teclado. Si por el contrario se escoge ROS Mobile, se imprime un mensaje que indica al usuario que se conecte mediante la aplicación móvil.

En caso de seleccionar la opción de Autonomo, se ejecuta todos los nodos necesarios para visualizar la conducción autónoma del robot.

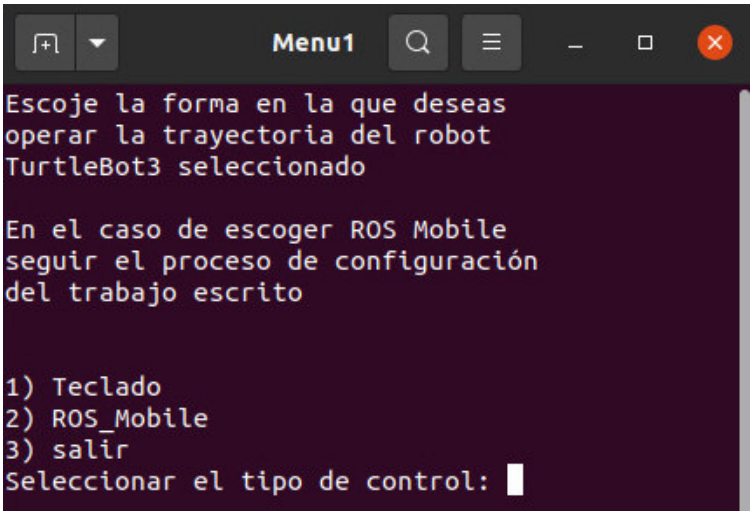


Figura VI.4. Menú secundario del tomo 1

Descripción de terminales

- **Terminal 1:** ejecución de menú principal donde se podrá escoger el modelo del robot, véase en la Figura VI.3.
- **Terminal 2:** despliegue de Roscore junto con Gazebo y el modelo del robot como se muestra en la Figura VI.5.
- **Terminal 3:** despliegue de menú secundario (véase en la Figura VI.4), en este se permite seleccionar el modo de operación del robot por teclado o por la aplicación ROS Mobile. Si se escoge la opción de teclado, en esta misma terminal se ejecutará el script necesario. En el caso de ROS Mobile, se mostrará un mensaje indicando que se conecte con la aplicación.

Si en el menú principal se selecciona Autónomo, se tienen de igual forma a las terminales 1 y 2 y luego se despliega lo siguiente:

- **Terminal 3:** despliegue del archivo launch que contiene la configuración de: camera 1 intrínseca. Ver la Figura VI.7(a)
- **Terminal 4:** despliegue del archivo launch que contiene la configuración de camera 2 extrínseca. Ver la Figura VI.7(b)
- **Terminal 5:** despliegue del archivo launch que contiene el script llamado Lane. Ver la Figura VI.7(c)
- **Terminal 6:** despliegue del archivo launch que contiene el script llamado control_lane. Ver la Figura VI.7(d)

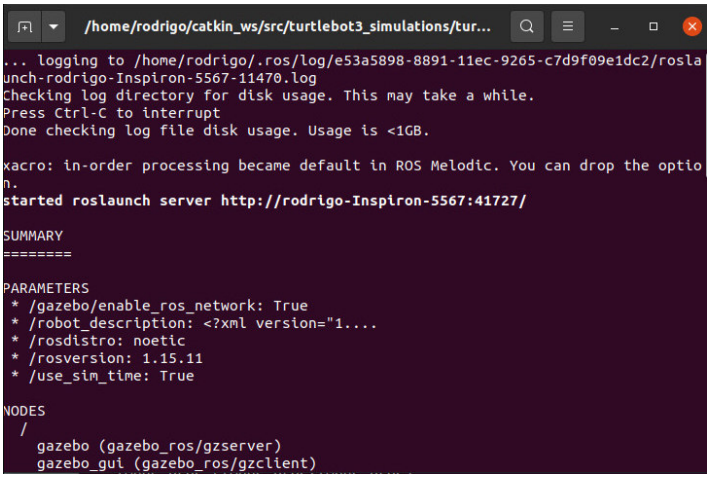


Figura VI.6. Despliegue del robot en el mundo de Gazebo en el tomo 1

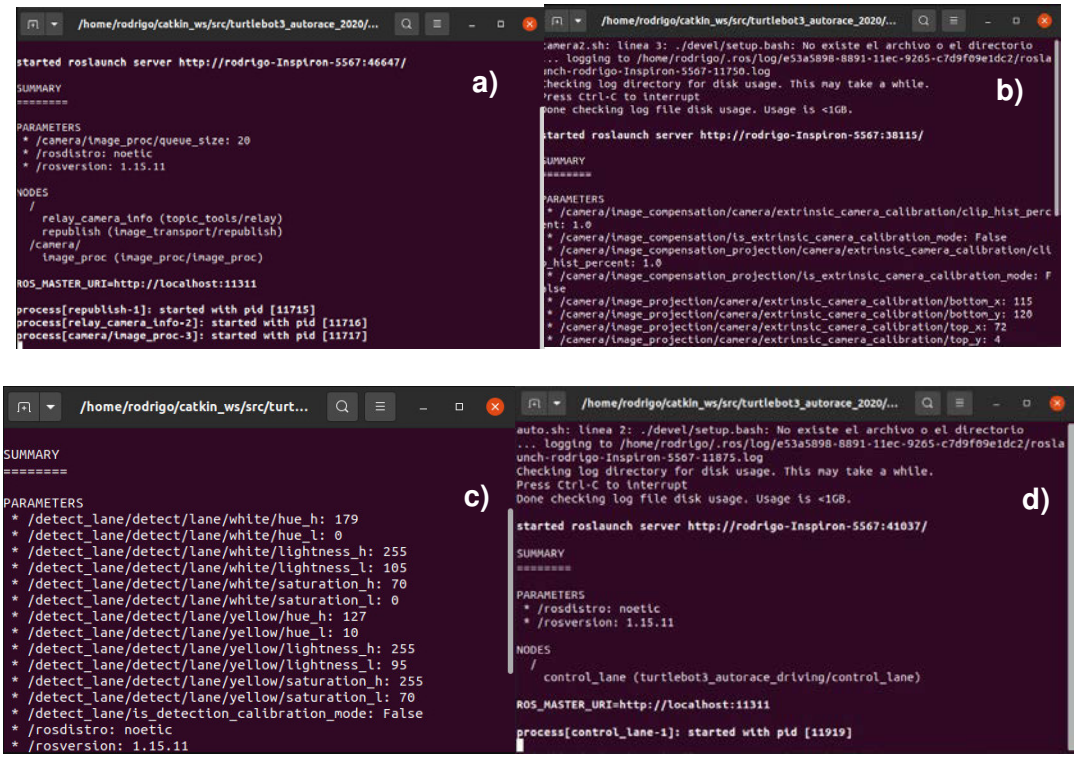


Figura V.7. Terminales generadas con la opción Autónomo en el tomo 1

En la Figura V.8 se presenta el resultado de la simulación del robot TurtleBot3 modelo Waffle pi con operación por teclado y en la Figura VI.9, se aprecia la simulación del modelo Burger al seleccionar el modo de operación Autonomo en el menú principal.

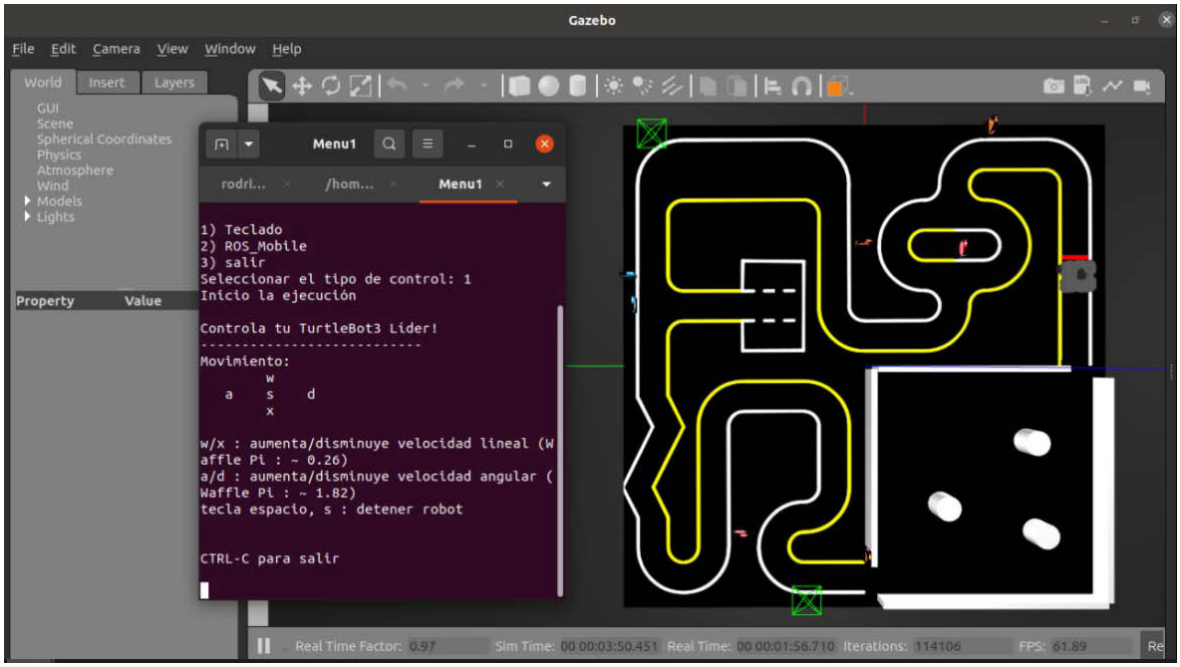


Figura VI.8. Resultado de simulación con operación por teclado del robot TurtleBot3 modelo Waffle pi

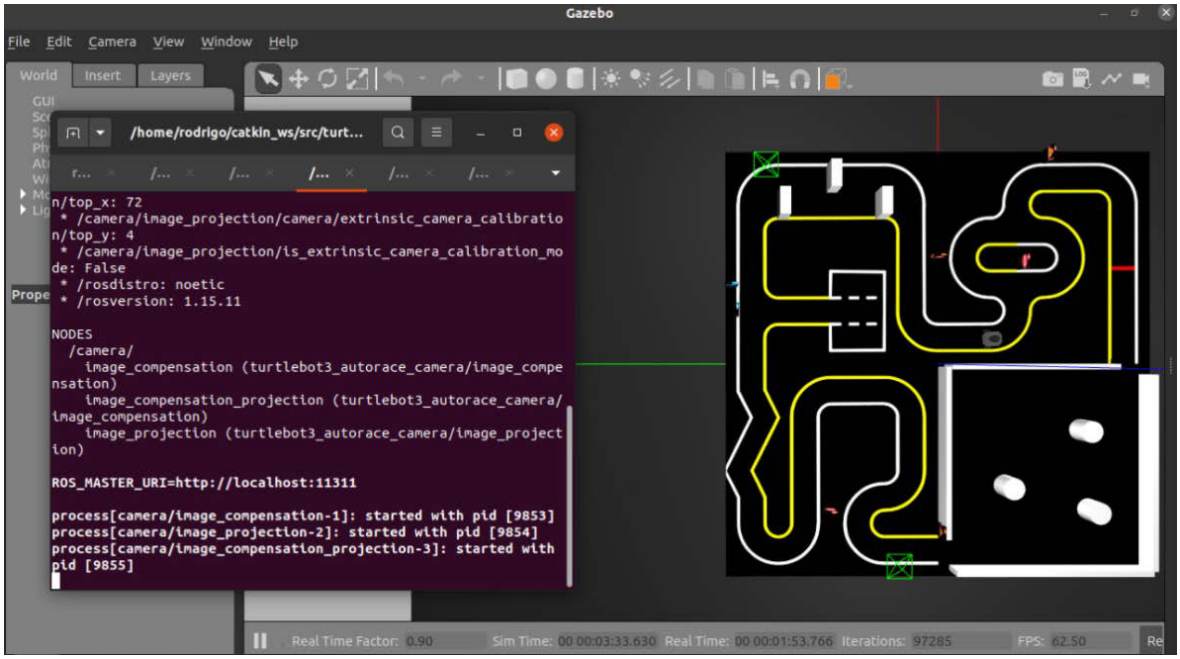


Figura V.9. Resultado de simulación de opción Autónomo del robot TurtleBot3 modelo Burger

Tomo 2

En la terminal generada dar el comando

```
bash menu.sh
```

Aparecerá entonces una gráfica del menú principal del tomo 2 mostrado en la Figura VI.10.

Figura VI.10. Menú principal del tomo 2

Al ejecutar la opción Teclado se ejecutará el entorno de Gazebo con el robot líder TurtleBot3 Waffle pi y seguidamente se abrirá otra terminal mostrando el menú secundario, se lo podrá realizar en la Figura VI.11.

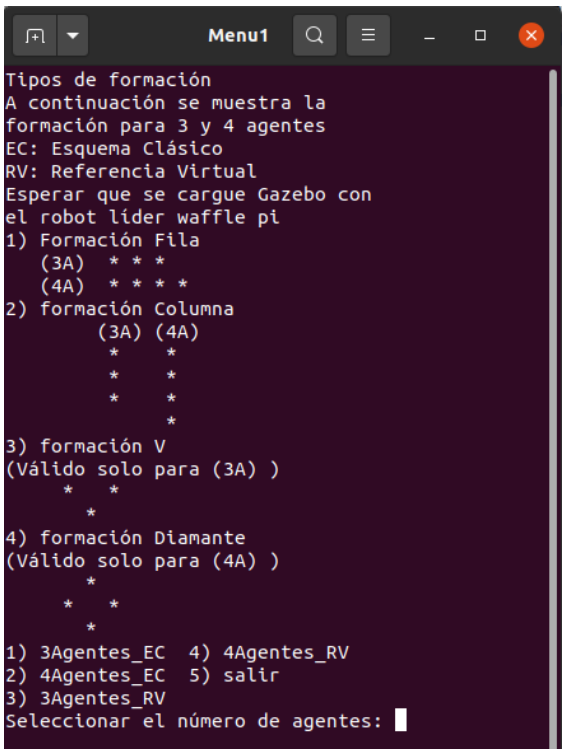


Figura VI.11. Menú secundario del tomo 2

Luego se podrá elegir cualquier opción mostrada en el menú secundario y se desplegarán las demás terminales. A continuación, se presenta una breve descripción de las terminales que se generan:

- **Terminal 1:** ejecución de menú principal donde se podrá escoger el movimiento del robot líder, véase en la Figura VI.10.
- **Terminal 2:** despliegue de Roscore junto con Gazebo y el modelo del robot líder como se muestra en la Figura VI.12(a).
- **Terminal 3:** despliegue de menú secundario (véase en la Figura VI.11), en este se permite seleccionar el tipo de formación, número de agentes y tipo de algoritmo. También se lanza el modelo de los robots seguidores para que aparezcan en Gazebo como se lo aprecia en la Figura VI.12(b).
- **Terminal 4:** en caso de seleccionar el movimiento del robot líder por teclado se ejecuta el scrip que realiza el movimiento del robot líder a través del teclado, tal como se observa en la Figura VI.12(c). En caso de seleccionar el movimiento por ROS_Mobile, se omite esta terminal.
- **Terminal 5,6 y/o 7:** en el caso de escoger esquema clásico, se muestra información de distancia en [m] y ángulo deseado en [rad] y los valores reales del seguidor junto con el tipo de formación que se está realizando en ese momento. Para el algoritmo de referencia virtual se aumenta la información de distancia y ángulo de la referencia como se observa en la Figura VI.12(d).

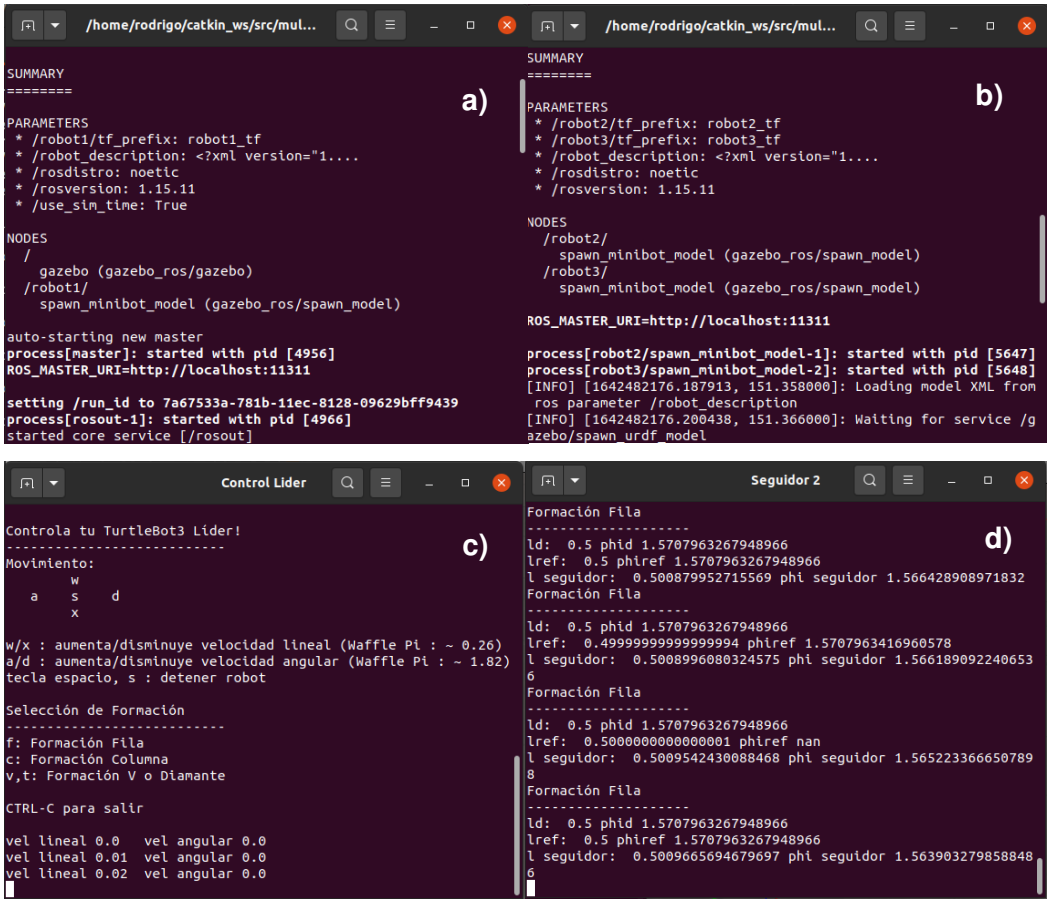


Figura VI.12. Despliegue de terminales del tomo 2

Para terminar la simulación del proyecto se deberá cerrar las terminales generadas escribiendo en cada una de ellas Ctrl+C.

En la Figura VI.13 y Figura VI.14, se pueden observar los resultados obtenidos de la simulación en el software Gazebo junto con las terminales.

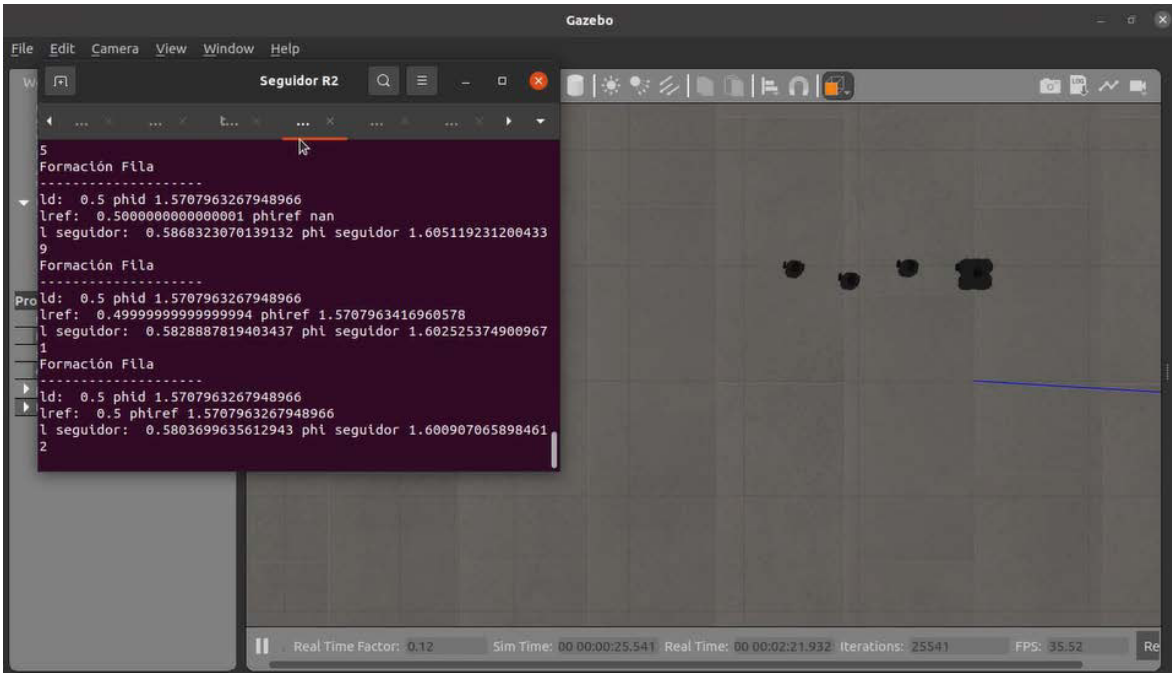


Figura VI.13. Resultado de simulación en formación fila con muestra de terminal del seguidor R2

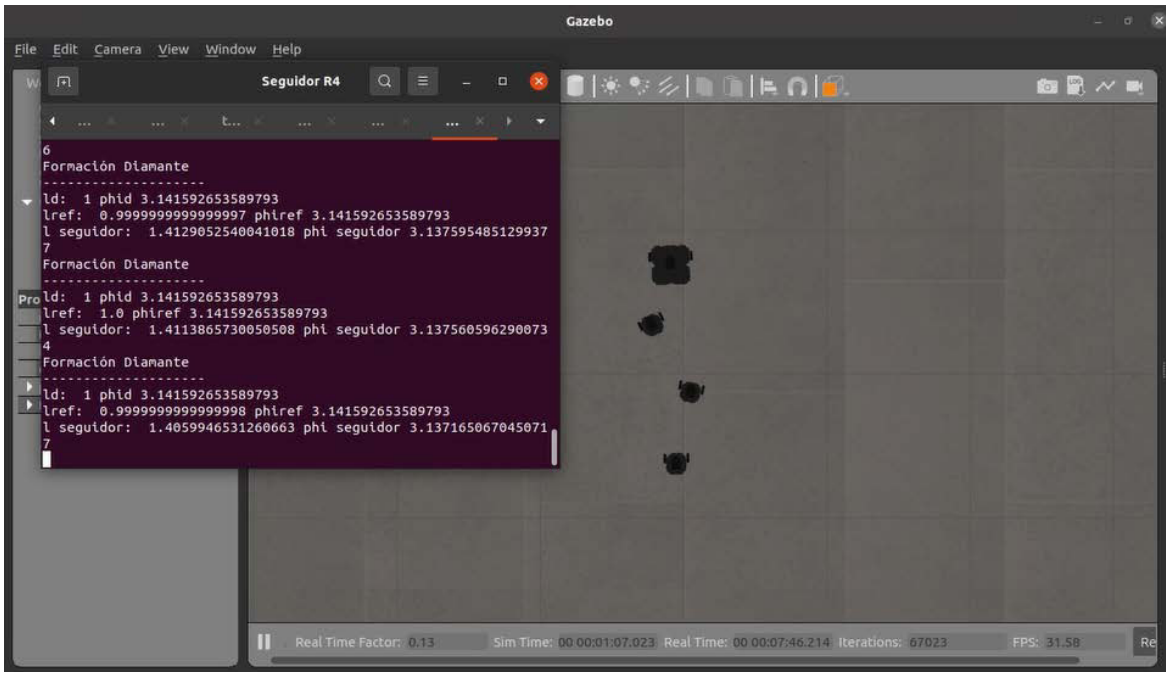


Figura VI.14. Resultado de simulación en formación columna con muestra de terminal del seguidor R4

ANEXO VII. Link a Videos demostrativos

El anexo presentado en esta sección se puede encontrar de igual manera en el primer tomo del trabajo desarrollado en conjunto.

Los videos principales que confieren a los resultados obtenidos del primer y segundo tomo se encuentran en el siguiente link:

Link: Leader-Follower Multi-Agent System - YouTube

En un inicio se tienen los videos de los resultados del tomo 1, con la descripción:

- Video 1: comparación del control de movimiento por teclado y ROS Mobile del robot TurtleBot3-Burger
- Video 2: comparación del control de movimiento por teclado y ROS Mobile del robot TurtleBot3-Waffle pi
- Video 3: ejecución de la conducción autónoma del paquete Autorace

Para el segundo tomo se tiene lo siguiente:

- Video 4: comparación de algoritmos con referencia virtual y esquema clásico para la realización de la formación en fila
- Video 5: comparación de algoritmos con referencia virtual y esquema clásico para la realización de la formación en columna
- Video 6: comparación de algoritmos con referencia virtual y esquema clásico para la realización de la formación en V
- Video 7: comparación de algoritmos con referencia virtual y esquema clásico para la realización de la formación en diamante
- Video 8: comparación de algoritmos con referencia virtual y esquema clásico para la realización de diferentes cambios de formación