ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO, CONTROL E IMPLEMENTACIÓN DE SISTEMAS ROBÓTICOS AVANZADOS

APLICACIÓN DE ALGORITMOS CON CERTIFICADOS DE BARRERA PARA PREVENIR COLISIONES EN SISTEMAS TIPO ENJAMBRE

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

GABRIEL EDUARDO MENA MARTÍNEZ

gabriel.mena@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD

patricio.cruz@epn.edu.ec

DMQ, octubre 2022

CERTIFICACIONES

Yo, Gabriel Eduardo Mena Martínez, 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.

GABRIEL EDUARDO MENA MARTÍNEZ

Certifico que el presente trabajo de integración curricular fue desarrollado por Gabriel Eduardo Mena Martínez, 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 el producto resultante 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.

GABRIEL EDUARDO MENA MARTÍNEZ

ING. PATRICIO JAVIER CRUZ DÁVALOS, PHD

MGS. DIEGO MALDONADO

DEDICATORIA

A mi familia, por su apoyo incondicional durante toda mi vida, por impulsarme para ser una mejor persona y profesional, este logro es para ustedes.

AGRADECIMIENTO

A Dios, por la oportunidad que me ha dado de estudiar en una prestigiosa institución, por su acompañamiento en cada paso, por la sabiduría y las oportunidades.

A mi familia, por su apoyo incondicional durante toda mi vida, por su comprensión, paciencia y compañía, por creer en mi potencial y siempre impulsarme a ser una mejor persona y profesional.

A mis padres, por sus enseñanzas, apoyo, paciencia y amor que ha forjado la persona que soy, gracias por siempre creer en mí.

A mis directores, por su guía, sus enseñanzas, su paciencia y su apoyo durante todo el desarrollo de este trabajo de integración curricular.

A la Escuela Politécnica Nacional y todos mis profesores, quienes han sido parte de mi crecimiento personal y profesional, por todos los conocimientos impartidos, por todas las experiencias vividas y por todo su apoyo y paciencia.

A Lorraine, por ser un apoyo incondicional durante toda la carrera, dentro y fuera de la Universidad, así como en mi vida, día a día.

A mis amigos, por todas las experiencias que compartimos a lo largo de los años.

ÍNDICE DE CONTENIDO

RESUMEN

Este trabajo de integración curricular plantea la elaboración de un sistema de robots tipo enjambre en un entorno simulado que consta de un robot explorador y dos seguidores, así como de obstáculos fijos preestablecidos y de una interfaz de usuario que permite ingresar obstáculos en cualquier coordenada dentro del entorno y en cualquier momento de la simulación. Se tiene como objetivo aplicar algoritmos con certificados de barrera para prevenir que los robots colisionen entre sí o con obstáculos del ambiente. De este modo cada uno ejecuta acciones individuales que, en conjunto, contribuyen a que el sistema tenga un funcionamiento adecuado. Para lo cual, se implementan algoritmos de prevención de colisiones y de consenso en los tres robots, con los cuales se identifican objetos cercanos y se evaden posibles colisiones con los mismos gracias al uso de un sensor láser de proximidad para el primer algoritmo y al cálculo de la distancia y la comunicación constante entre los agentes para el segundo. Además, el robot explorador posee como sensor una cámara que junto con algoritmo de detección de colores le permite diferenciar entre los obstáculos, que son de color rojo y blanco, y el objetivo, que es de color verde. Una vez que el robot explorador encuentra el objetivo, envía sus coordenadas a los seguidores, quienes inician su recorrido hacia las mismas. Tras realizar varias pruebas de funcionamiento del entorno simulado, se concluye que los algoritmos utilizados son eficaces y que, por ende, el sistema tipo enjambre funciona correctamente.

PALABRAS CLAVE: Sistemas Tipo Enjambre, Prevención de Colisiones, Consenso, Certificados de Barrera, ROS, Gazebo.

ABSTRACT

This final degree project proposes the implementation of a swarm robotic system in a simulated environment consisting of an explorer robot and two followers. It also has preset obstacles and a user interface that allows new obstacles to be included at any time and in any coordinate within the environment. The objective is to apply algorithms with barrier certificates to prevent robot collision with their neighbors and with the obstacles. Therefore, each robot executes individual actions that, together, contribute to the optimal functioning of the system. To achieve these goals, collision avoidance and consensus algorithms are programmed for the three robots so nearby objects can be identified and possible collisions with them may be avoided thanks to the use of a proximity laser sensor for the first algorithm and the distance calculation and constant communication between agents for the other. In addition, the explorer robot has a camera and a color detection algorithm that allows it to differentiate between obstacles, which are red and white, and the target of the project that is green. Once the explorer robot finds the target, it sends its coordinates to the followers that start their movement towards them. After carrying out various tests of the simulated environment, we conclude that the implemented algorithms are effective and therefore, the swarm robotic system works correctly.

KEYWORDS: Swarm Robotics, Collision Avoidance, Consensus, Barrier Certificates, ROS, Gazebo.

1. INTRODUCCIÓN

Acorde a [1], el objetivo final de la robótica es el diseño, la creación y la programación de robots que sean autónomos; es decir, que puedan colaborar con los seres humanos sin que tengan que ser supervisados. Para ello, es necesario que los robots puedan interpretar y asimilar su entorno, de modo que optimicen la ejecución de las tareas. Existen varios métodos para lograr estos objetivos, pero la mayoría de ellos resultan ser muy complejos, muy costosos o no se cuenta con la tecnología necesaria para utilizarlos [1].

Por ello, una solución que resulta ser más práctica y económica es el uso de sistemas de enjambres computarizados, mismos que permiten la existencia de un comportamiento global del sistema que esté basado en la auto organización de los individuos que lo componen. Esto es posible gracias al control descentralizado y a la comunicación entre agentes [2]. Otra característica de los enjambres de robots es el uso de algoritmos con el fin de replicar los comportamientos observados en la naturaleza [3], como los de las poblaciones de hormigas, los panales de abejas, los cardúmenes de peces y la alineación de las aves en vuelo [4]. Así, los sistemas tipo enjambre poseen un gran potencial para resolver problemas como el control de tráfico vehicular, de maquinaria o de procesos, la exploración, el mapeo, búsqueda y rescate, entre muchos otros.

El presente proyecto de integración curricular alinea sus objetivos a los mencionados en el párrafo anterior y plantea la elaboración de un entorno simulado e interactivo con el usuario; de modo que el mismo pueda colocar diferentes obstáculos en el escenario mientras que robots móviles, que funcionan como un sistema tipo enjambre, ya se encuentran en el mismo ejecutando tareas asignadas; de esta manera deben evadir posibles colisiones y decidir entre varios caminos para realizar las tareas designadas. Para lograrlo, se trabajará con el entorno ROS - GAZEBO, en el que se desarrollará la aplicación de prevención de colisiones deseada.

Así, el entorno mostrará obstáculos tanto predeterminados como creados por el usuario, mismos que serán evadidos por al menos un robot móvil explorador y un obrero. La diferencia entre ambos radica en que el robot explorador se moverá primero por todo el escenario en búsqueda de un elemento objetivo, detectando los obstáculos que encuentre en su camino y evitándolos. Una vez que el robot explorador haya encontrado el objetivo comunicará su ubicación al robot obrero, quien empezará a moverse por el escenario evitando los obstáculos que encuentre en su camino hasta llegar a la ubicación del objetivo y cumplir con su función.

1.1 OBJETIVO GENERAL

Aplicar algoritmos con certificados de barrera para prevenir colisiones en sistemas tipo enjambre.

1.2 OBJETIVOS ESPECÍFICOS

1. Realizar una revisión bibliográfica sobre los sistemas de enjambres de robots, los algoritmos con certificados de barrera para la prevención de colisiones y el uso del entorno ROS – GAZEBO.

2. Diseñar un escenario en ROS – GAZEBO que conste de obstáculos fijos preestablecidos pero que también permita al usuario interactuar con el entorno, ingresando obstáculos en cualquier momento y ubicación.

3. Diseñar al menos un robot diferencial móvil explorador y un obrero e implementar un algoritmo descentralizado de prevención de colisiones y un algoritmo de consenso en cada uno.

4. Realizar la simulación de todo el sistema integrado en el software ROS – GAZEBO, ejecutando el algoritmo descentralizado para una adecuada prevención de colisiones y el algoritmo de consenso por parte de los robots explorador y obrero.

5. Evaluar los resultados obtenidos, en especial en cuanto a la aplicación del algoritmo descentralizado de prevención de colisiones y del algoritmo de consenso.

1.3 ALCANCE

- Se realiza una revisión bibliográfica sobre estudios similares, conceptos relevantes, algoritmos y aplicaciones relacionados con sistemas robóticos tipo enjambre. Dicha revisión se lleva a cabo para conocer el tema, sus aplicaciones, beneficios y limitaciones.

- Se recopila información correspondiente al software ROS – GAZEBO y los algoritmos de prevención de colisiones, con el fin de conocer sus principales herramientas y aplicaciones.

- Se diseña un entorno virtual en el software ROS – GAZEBO para un enjambre de al menos un robot diferencial móvil explorador y un obrero.

- Se diseñan al menos tres robots móviles terrestres: un explorador y dos obreros.

- Se diseña una interfaz gráfica de ROS – GAZEBO, misma que muestra obstáculos fijos y dinámicos, estos últimos pueden ser ingresados por el usuario en cualquier momento y ubicación, de modo que las configuraciones de obstáculos sean diferentes cada vez.
- Se implementa un algoritmo de consenso y un algoritmo descentralizado de prevención colisiones de los robots entre sí y con objetos del entorno.
- Se realiza la simulación de todo el sistema integrado, ejecutando acciones de transmisión de información entre los robots explorador y obrero.
- Se comprueba el correcto funcionamiento del sistema tipo enjambre, del algoritmo de consenso y algoritmo descentralizado de prevención de colisiones, mediante la ejecución de pruebas en el entorno de simulación desarrollado.

1.4 MARCO TEÓRICO

1.4.1. SISTEMAS ROBÓTICOS TIPO ENJAMBRE

1.4.1.1. Definición

En la actualidad, la investigación del desarrollo y aplicaciones de los sistemas tipo enjambre es uno de los campos de estudio más relevantes dentro de la robótica, pues este analiza cómo la coordinación entre un gran número de robots relativamente sencillos permite obtener sistemas inteligentes colectivos gracias a la interacción de los agentes entre sí y con el entorno [5].

Acorde a [5], la robótica de enjambres surge a partir de la bioinspiración; es decir, se basa en el comportamiento organizado de insectos como hormigas, termitas, avispas, abejas y peces, mismos que son considerados como *insectos sociales* dada su capacidad de interrelacionarse y de auto organizarse. Así, cada miembro del grupo cumple una función específica que, cuando se junta con las funciones del resto de miembros, permite crear un sistema colectivo inteligente que hace que las colonias funcionen adecuadamente.

De la misma manera, los sistemas multi – agente están conformados por varios robots que poseen una función específica y que, cuando interactúan entre sí y con el entorno, crean una inteligencia colectiva que surge del comportamiento global del enjambre, mismo que es capaz de llevar a cabo tareas complejas, que están fuera de las capacidades de un robot individual [5].

Además, la programación y autonomía de los sistemas tipo enjambres se basa en la creencia de que los individuos se vuelven más inteligentes a medida que interactúan con sus grupos sociales; por ejemplo, un individuo que ha crecido aislado de la sociedad nunca

alcanzará el nivel de inteligencia y habilidades que posee un individuo que se encuentra en constante interacción con otros a lo largo de su vida. Esto demuestra la importancia de permitir que los agentes robóticos sean capaces de *aprender en la marcha*, de modo que puedan regular su comportamiento acorde a los requerimientos del entorno [6].

Entonces, el objetivo principal de la robótica aplicada a los sistemas multi – agente es crear y programar robots autónomos, que sean capaces de ayudar a los seres humanos en diferentes tareas sin que se requiera demasiada supervisión. Para ello, los agentes robóticos tendrán que estar programados para interpretar su entorno, tomar decisiones y adaptar su comportamiento en tiempo real, de modo que se optimice la ejecución de las tareas asignadas al enjambre [1].

1.4.1.2. Características

La robótica de enjambres destaca tres características importantes de los grupos de insectos mencionados anteriormente [5], estas son: robustez, flexibilidad y escalabilidad. Cuando se aplican dichas características en la robótica, se pueden enfatizar aspectos como el control descentralizado, el uso de información local, la aparición de un comportamiento global y la robustez del sistema.

Pero no todos los grupos de robots pueden ser considerados como enjambres o como sistemas multi – agente; para serlo, deben cumplir con una serie de características, mismas que se detallan a continuación:

- ***Simplicidad:*** El hardware de los robots pertenecientes a un sistema tipo enjambre, así como la función que debe cumplir cada robot, deben ser simples, pues estas características son las que permiten que los robots aprendan y se adapten rápidamente a las necesidades del entorno [6]. Los sistemas multi – agente centran su atención en el uso de varios robots sencillos en lugar de un solo robot completo, de modo que las capacidades de los individuos son limitadas, pero es la cooperación entre todos los miembros del grupo la que permite que se lleven a cabo tareas complejas de una forma eficiente [7].
- ***Autonomía:*** El objetivo de los sistemas multi – agente es que los robots miembros del mismo no estén controlados central ni remotamente por algún otro agente; es decir, se busca que los robots interactúen, tomen decisiones y cumplan las tareas designadas mediante un control descentralizado y que sea el conjunto de acciones individuales el que permita al sistema alcanzar una inteligencia global. Sin embargo,

podría existir un robot *líder* que guíe las acciones o establezca el camino que deberán seguir otros agentes [5].

- ***Control del sistema:*** El objetivo de la robótica de enjambres es que el sistema multi – agente posea un control descentralizado para su funcionamiento, de modo que los robots sean autónomos y adapten su comportamiento a las necesidades del entorno gracias al aprendizaje en tiempo real [2].

- ***Comunicación:*** Ya sea que el sistema multi – agente posea a un robot líder y varios seguidores o que no exista una jerarquía entre los mismos, la comunicación es un factor primordial para que el sistema funcione de forma óptima. En el caso del sistema líder – seguidor, por ejemplo, la comunicación permite que el líder transmita información sobre el entorno y el objetivo a los seguidores, de modo que los mismos puedan elegir una ruta determinada o cumplir una función acorde a la información que reciban. Existen tres formas de comunicación entre los agentes, mismas que se dan gracias a la interacción con el medio ambiente, a través de sensores o de comunicación [6].

- ***Bioinspiración:*** La robótica de enjambres está inspirada por la observación de sistemas biológicos de insectos sociales, que son capaces de organizarse para que cada miembro del grupo social cumpla con una función y que la sumatoria de funciones permita que el sistema funcione [5], como lo muestra la Figura 1.1.

Figura 1.1. Bioinspiración como base para los sistemas multi – agente [8]

1.4.1.3. Aplicaciones

Dada la simplicidad de los robots miembros del enjambre y al trabajo en equipo de os mismos, es posible ejecutar tareas complejas en varios campos de acción; la siguiente tabla presenta ejemplos sobre estos.

Tabla 1.1. Aplicaciones de los sistemas de robots tipo enjambre [Basado en 7]

Tareas / Aplicación	Descripción
Explorar una región	Los diferentes robots miembros del enjambre pueden ser utilizados para explorar un entorno en su totalidad, mapearlo, tomar datos y realizar un seguimiento del estado de la región explorada, cumpliendo el objetivo en menor tiempo del que lo cumpliría un solo robot.
Identificar un objetivo en el entorno	Los enjambres de robots también pueden sr utilizados para encontrar una fuente de recursos, buscar un objeto en específico, recuperar objetos o salvar a personas y seres vivos que se encuentren perdidos o atrapados. Los robots son de gran utilidad en aquellos entornos en los que se dificulta el acceso de un equipo de rescatistas humanos.
Actividades de riesgo	Los sistemas multi – agente pueden ser utilizados para actividades que resultarían ser muy peligrosas para un ser humano. Además, los sistemas son robustos, tolerantes a fallos y cuentan con un gran número de robots, por lo que la pérdida o averío de un robot no representa una pérdida importante para todo el sistema.
Variación en la cantidad de agentes participantes	Existen proyectos en los que es necesario variar la cantidad de agentes participantes para realizar una actividad en específico, dependiendo de las necesidades de la actividad realizada. Si se trabaja con seres humanos, sería poco ético variar la cantidad de individuos a voluntad, pues la estabilidad de los mismos resultaría afectada. Sin embargo, los sistemas multi – agente son escalables, por lo que se pueden agregar o quitar robots miembros sin que esto suponga un inconveniente.
Redundantes	Los sistemas tipo enjambre está diseñados para cumplir con tareas que requieran redundancia, pues todos los robots son

	homogéneos y tienen la capacidad de repetir procedimientos las veces que sean necesarios, sin que exista una probabilidad de fallo por redundar, como sucede con los seres humanos.

Todo lo mencionado da cabida a una serie de ventajas que poseen los sistemas de robots tipo enjambre. Entre las más destacadas se encuentran [7]:

- ***Robustez y tolerancia a fallos:*** Los sistemas tipo enjambre pueden seguir funcionando a pesar de que uno de sus agentes no funcione adecuadamente o deje de realizar su función, esta es una ventaja significativa tanto en costos económicos como en eficiencia del sistema.
- ***Escalabilidad:*** La cantidad de agentes que intervienen en un proceso puede ser aumentada o disminuida a conveniencia, según las necesidades del proceso y sin que esto afecte al funcionamiento global del sistema.
- ***Paralelismo:*** Esto quiere decir que un grupo de robots que trabaja en conjunto puede llevar a cabo una tarea compleja, que ha sido descompuesta en sus tareas más sencillas, en lugar de que esta sea realizada por un único robot, que resulta ser más complejo y costoso.
- ***Menores costos:*** Resulta menos costoso elaborar un sistema de múltiples robots relativamente sencillos a elaborar un único robot complejo; además, resulta menos costoso sustituir o eliminar a una parte del sistema cuando presenta fallos o deja de ser necesaria, que eliminar a todo el sistema en su totalidad.

1.4.2. COORDINACIÓN DE SISTEMAS TIPO ENJAMBRE

1.4.2.1. Algoritmos de consenso

Como se mencionó anteriormente, el objetivo de un sistema de enjambre es lograr que el mismo funcione de forma óptima gracias al cumplimiento efectivo de actividades individuales por parte de cada miembro del grupo. Para lograr dicho objetivo y dado que el control del sistema es descentralizado, es necesario que exista comunicación constante entre los agentes, más aún cuando se trabaja con un modelo líder- seguidor, en el que el robot líder explora el entorno primero para después comunicarse con los robots seguidores, quienes iniciarán sus funciones, acorde a la información recibida.

Los algoritmos de consenso son los que permiten que exista una comunicación adecuada entre los agentes miembros del sistema, como lo muestra la Figura 1.2. Además de la comunicación entre agentes, este algoritmo puede ser usado para calcular la distancia de los robots entre sí, esto puede tener múltiples aplicaciones en los sistemas tipo enjambre, dado que permite conocer si el robot líder ha llegado a su destino y qué tan distante se encuentra dicho destino de los robots seguidores; así como conocer qué tan distantes se encuentran los robots seguidores entre sí, puesto que pueden aparecer en diferentes posiciones dentro del entorno virtual.

Existe una gran cantidad de algoritmos que pueden ser aplicados en los sistemas tipo enjambre, por lo que la selección de uno de ellos dependerá de su complejidad, la aplicación del sistema, el entorno en el que se encuentran los agentes y de la cantidad de robots miembros del enjambre. Algunos de los algoritmos de consenso que se enfocan a diferentes áreas de aplicación de los sistemas tipo enjambre en robótica son [9]:

- ***Consenso para Rendezvous:*** Consiste en utilizar un algoritmo que permita que todos los agentes miembros del sistema se reúnan en un solo lugar dentro del entorno, tras haber tomado una decisión de forma colectiva [10].
- ***Consenso para fusionar datos:*** La transmisión de información entre agentes suele ser compleja debido a la gran cantidad de datos e información recopilada por cada individuo. Por ello, es necesario que los agentes lleguen a un consenso para la fusión de sus datos y la posterior toma de decisiones con base en los datos globales ya fusionado. Además, ningún agente está encargado de tomar una decisión para todo el sistema, pero cualquier agente puede iniciar el proceso de fusión de datos y toma de decisiones en cualquier momento [11].
- ***Consenso distribuido con el uso de grafos de comunicación dirigida:*** Se utiliza un grafo de comunicaciones que está conectado con nodos estrechamente relacionados entre sí, pero el consenso es realizado de forma distribuida [12].

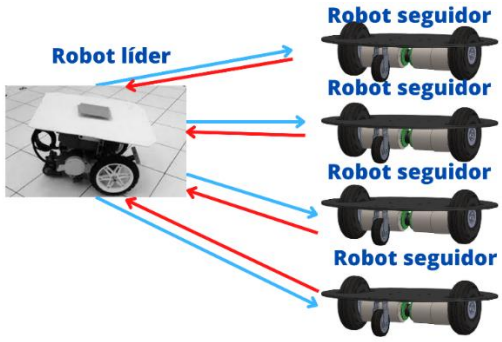


Figura 1.2. Funcionamiento del algoritmo de consenso para la toma de decisiones [Fuente propia]

1.4.2.2. Algoritmos de prevención de colisiones

La prevención de colisiones es fundamental dentro de diversas aplicaciones robóticas, pues una oportuna prevención disminuye costos de mantenimiento, pérdidas monetarias y materiales e incrementa la eficiencia del sistema. Para lograr prevenir las colisiones de los agentes miembros de un sistema entre sí o con objetos del entorno, es necesario asegurarse de que ninguna área del entorno esté ocupada por más de un objeto a la vez, de modo que se previenen las colisiones basándose en la información geométrica de un objeto y su posición en el espacio [13], información que es detectada por los agentes robóticos gracias al uso de sensores.

Si se desea evitar que los robots colisionen entre sí o con objetos del entorno, es primordial el uso de sensores de proximidad y, en algunos casos, de cámaras que permitan detectar a otros agentes u objetos y adaptar en tiempo real el comportamiento del robot, de modo que este tome la decisión de cambiar su dirección y posición de destino al instante. A este método se le conoce como *enfoque pre colisión* [12], pues evita que una colisión se produzca al detectarla de antemano.

Algunos de los algoritmos de prevención de colisiones cuentan con certificados de barrera, los cuales son medios utilizados para controlar o prevenir la colisión del robot con otros agentes o con otros objetos presentes en el sistema. Algunos algoritmos que se pueden utilizar para la prevención de colisiones en un sistema se muestran a continuación; ciertos algoritmos como RAPID, consultas de proximidad y I-COLLIDE utilizan certificados de barrera [12].

Algunos algoritmos que se pueden utilizar para la prevención de colisiones en un sistema son [12]:

- ***RAPID:*** También se conocen como *árboles de cajas envolventes orientadas* (OBBTree, por sus siglas en inglés) y son utilizados para detectar colisiones con objetos que poseen un movimiento rígido; por lo general, este modelo se aplica en modelos que son poligonales.

- ***QuOSPO:*** Es un algoritmo de detección de colisiones que se basa en QuOSPO, el cual es un bloque que posee orientaciones cuantiadas y primarias y utiliza la técnica del volumen envolvente para la detección de objetos en el entorno.

- ***HYCODE:*** Este método para la detección de colisiones es aplicado en un espacio bidimensional donde se encuentran imágenes. Este algoritmo distribuye la carga computacional hacia el flujo de trabajo dado por el hardware gráfico, de modo que

evita caer en un cuello de botella producido por la gran cantidad de información que se recopila para prevenir una colisión.

- ***Consultas de Proximidad:*** Por lo general, los algoritmos de detección de colisiones reportan cero, una o varias colisiones acorde a la información recopilada a partir del entorno; sin embargo, cuando se aplica el algoritmo de consultas de proximidad, se hace viable la adquisición de información relevante adicional sobre los objetos del entorno, como la distancia entre objetos y la profundidad de penetración en el objeto en caso de producirse un impacto. El resultado obtenido tras la aplicación de este algoritmo puede ser aproximado, exacto o booleano

- ***I-COLLIDE:*** es una técnica de prevención de colisiones en dos niveles y consiste en *podar* varios pares de objetos acorde con el uso del método de cajas envolventes; así, detecta la posible colisión entre los pares de una manera exacta dada su geometría y posición.

1.4.3. ROS – GAZEBO

Los algoritmos de consenso y de prevención colisiones descritos anteriormente, pueden ser implementados en los agentes robóticos a ser creados e implementados en el sistema operativo ROS-GAZEBO.

1.4.3.1. Definición

El Sistema Operativo Robótico - ROS (Robot Operating System) es un sistema operativo de código abierto BSD que es ampliamente utilizado por los desarrolladores de software que crean aplicaciones con robots. Una de las ventajas más significativas de este sistema operativo es que provee una abstracción del hardware, varios controladores para los dispositivos, librerías, ejemplos preestablecidos, herramientas de visualización, mensajes para la comunicación, administración de paquetes y entorno dinámico e interactivo con el usuario [14]. ROS es compatible con Ubuntu 14.04, el cual es un sistema operativo que se basa en GNU / Linux, es un software libre que funciona en su propio entorno, denominado *Unity,* es capaz de compilar paquetes de datos de cualquier tipo, incluso para versiones antiguas del sistema [14].

El presente proyecto de integración curricular trabaja con ROS – Gazebo, el cual permite simular diferentes prototipos de robots en entornos de simulación que ya estén establecidos previamente o bien, que sean diseñados en su totalidad por el programador del entorno [15]. Todo esto permite que la simulación de los robots sea eficiente y exacta

y que los entornos puedan ser aplicados tanto en interiores como en exteriores, incluso si los mismos son complejos [9].

Gazebo es un simulador dinámico en tres dimensiones que puede trabajar con varios robots a la vez. Sus simulaciones son precisas y eficientes y permite trabajar con una variedad de robots, sensores y objetos en entornos que pueden ser simples o complejos y que se encuentran tanto en el interior como en el exterior, como se aprecia en la Figura 1.3. El realismo de Gazebo permite al usuario percibir, en tiempo real, a las interacciones entre los objetos de un entorno determinado, así como identificar si el objeto es rígido; así como sus características geométricas y de posición [16]. Además, el acceso a Gazebo es gratuito y sus interfaces gráficas son de alta calidad [9], lo que permite que las simulaciones se aproximen efectivamente a la realidad.

Figura 1.3. Entorno de simulación en ROS - Gazebo [17]

1.4.3.2. Aplicaciones

En la actualidad, el uso de simulaciones robóticas es primordial para el correcto diseño, implementación y desempeño de varios proyectos en la industria y en la vida cotidiana. Estas disminuyen significativamente los costos de los proyectos, pues evitan la creación de prototipos físicos que podrían presentar errores de diseño o armado o que quedarían obsoletos tras las pruebas de desempeño de los robots. Asimismo, permiten que los procesos sean más eficientes y económicos, dado que las pruebas de funcionamiento y el diseño de los robots se realizan en tiempo real y en poco tiempo, de modo que el producto final será adecuado para el proyecto.

Un simulador como ROS – Gazebo permite probar la eficiencia y uso adecuado de algoritmos en robots, entornos de simulación realistas y entrenamiento de los robots

mediante el uso de inteligencia artificial (AI) [9]. Así, existen varias aplicaciones de ROS Gazebo, entre las más importantes destacan:

- ROS posee dos niveles de trabajo, el primero trabaja con archivos o paquetes del sistema, mismos que contienen nodos que permiten que exista un proceso determinado para el funcionamiento del sistema. El segundo, por su parte, utiliza comandos útiles para las gráficas computacionales, mismos que facilitan la comunicación entre los nodos mencionados anteriormente [16].
- Creación de plataformas y sistemas que hacen posible llevar a cabo una operatividad eficaz de los robots [16].
- Acceso al trabajo de otros programadores, a través del repositorio virtual GitHub.
- Simulación de entornos y robots tanto sencillos como complejos.
- Pruebas en tiempo real, obtención de datos e información sobre los procesos realizados.
- Detección de las coordenadas y características geométricas de los robots y objetos del entorno virtual, de modo que las lecturas sobre los mismos son precisas, como se observa en la Figura 1.4; todo esto permite que la implementación en físico de los proyectos sea eficaz, económica y viable [15].

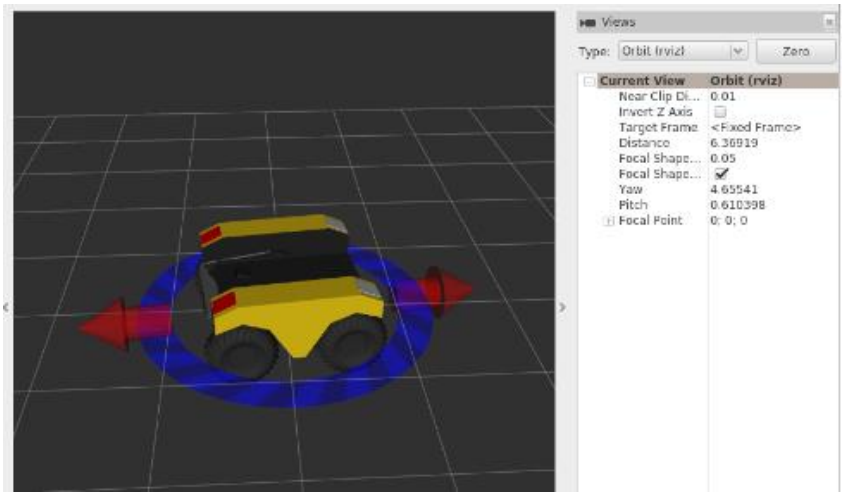


Figura 1.4. Detección de características geométricas en ROS – Gazebo [17]

1.4.3.3. Estructuración de archivos .xacro utilizando el formato URDF

URDF *(United Robotics Description Format)* es un lenguaje popular en el entorno de ROS y se utiliza para modelar a los robots y colocarlos en dicho entorno con el fin de realizar simulaciones y análisis con los mismos, dependiendo del objetivo del proyecto en el que se trabaje. Para hacerlo, se utilizan macros de gramática XML, lo que permite que se

estructuren archivos .xacro que, al tener como formato del código o instrucciones al lenguaje URDF, beneficia considerablemente al diseño, creación y desarrollo de los robots, pues genera estructuras legibles y sencillas dentro de ROS [18].

Para diseñar un robot utilizando URDF es necesario conocer los componentes del mismo; en la mayoría de los casos los robots se componen en su mayor parte de uniones y enlaces, denominados *links* y estructurados en forma de árbol, como lo plantea la gramática XML. Estos enlaces poseen propiedades o componentes básicos como: propiedades de masa, de inercia, colores y tipos de articulaciones, estas últimas pueden ser traslacionales o giratorias y le permiten al robot moverse en diferentes direcciones o girar sobre su propio eje. Los enlaces, a su vez, poseen subcomponentes como los inerciales y los visuales según se necesite [18]. Un ejemplo de la estructura general del lenguaje URDF se visualiza en la Figura 1.5, donde se pueden ver algunas propiedades como inercia y visualización de la estructura del robot que se está creando, en esta última se pueden observar generalidades como la geometría y detalles del material como su color u otras características que el diseñador desee implementar.

```
<robot name = "linkage">
        <link name = "root link">
                <inertial>
                        ...
                </inertial>
                <visual>
                        <geometry>
                                ...
                        </geometry>
                        <material>
                                <color rgba = "1 0 0 1" />
                        </material>
                </visual>
        </link>
        ...
</robot>
```

Figura 1.5. Estructura general de parametrización en lenguaje URDF.

A partir de la estructuración del lenguaje URDF que se mencionó anteriormente se plantea la estructura general de un archivo .xacro, que se puede observar en la Tabla 1.2 y que se utilizará como base para el desarrollo de este trabajo de integración curricular en el diseño de los agentes.

Tabla 1.2. Estructura general de un archivo xacro [18]

Parte	Ejemplo base de código
Estructuración del archivo xacro	<?xml version="1.0" ?> <robot name="ejemplo" xmlns:xacro="https://www.ros.org/wiki/xacro" >
Inicio: link	<link name="${name}"><link name="link_chassis">
Propiedades: Inercia, colisión y visualización, entre otros.	<inertial> <mass value="5"/> <origin rpy="0 0 0" xyz="0 0 0.1"/> <inertia ixx="0.03" ixy="0" ixz="0" iyy="0.10 iyz="0" izz="0.106"/> </inertial> <collision name="collision_chassis"> <geometry> <cylinder length="0.08" radius="0.3"/> </geometry> </collision> <visual> <origin rpy="0 0 0" xyz="0 0 0"/> <geometry> <cylinder length="0.08" radius="0.3"/> </geometry> <material name="blue"/> </visual>
Cierre: link	</link>

Como se observa, para la estructuración que se realiza con base en los códigos URDF es primordial la esquematización de las propiedades; así como considerar los niveles de los contenidos, secciones o subsecciones para cada propiedad. Esto es debido a que la presencia de niveles en los códigos establece una jerarquía de inicio o cierre de las etiquetas o links de trabajo.

Por otro lado, cada uno de los archivos que se ejecutan dentro de ROS se conocen como *nodo* y trabaja en conjunto con los tópicos, que son los encargados de transmitir y recibir datos tras haberse suscrito a los nodos mencionados. El trabajo en conjunto de los nodos y los tópicos permite crear redes de comunicación entre cada nodo y el nodo maestro *Gazebo* dentro del mundo virtual.

También se utiliza el formato URDF para la creación del parámetro que permite que se implementen sensores en los robots, esto se ejemplifica en la Figura 1.6 para la implementación de un sensor cámara en un robot. Sin embargo, utilizando la misma base se pueden implementar otros tipos de sensores, como los sensores láser de proximidad, que serán utilizados más adelante en el diseño de los robots para el presente trabajo de integración curricular.

```
<robot>
  ... robot description ...
  <link name="sensor_link">
    ... link description ...
  </link>

  <gazebo reference="sensor_link">
    <sensor type="camera" name="camera1">
      ... sensor parameters ...
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        ... plugin parameters ..
      </plugin>
    </sensor>
  </gazebo>

</robot>
```

Figura 1.6. Implementación de sensores en un robot [19]

2. METODOLOGÍA

En el presente capítulo se describe el proceso para el desarrollo de la aplicación en ROS - Gazebo, con base en la unificación del sistema tipo enjambre; para ello, se utilizan algoritmos con certificados de barrera para la prevención de colisiones, basándose en la revisión de la literatura, que fue detallada en la sección anterior.

Dicha aplicación permite que se produzca una interacción de tres agentes robóticos, cuyo diseño y proceder fueron inspirados en el comportamiento de las hormigas, mismas que trabajan en conjunto para el alcance de sus objetivos. El primer agente se denomina *robot explorador* y, para este proyecto, cuenta con un sensor de proximidad y una cámara para poder detectar y reconocer el objetivo. Una vez que el robot explorador encuentra el objetivo en el entorno envía una señal a dos agentes denominados *robots obreros*, estos, a su vez, detectan la señal enviada por el robot explorador y la siguen con el fin de dirigirse hacia el objetivo.

Mientras los tres robots se encuentran ejecutando las tareas asignadas, los algoritmos de evasión de obstáculos con certificados de barrera se encuentran activos en todo momento, permitiendo que el desarrollo de la aplicación sea adecuado y se eviten potenciales colisiones de los agentes robóticos entre sí y con objetos del entorno.

El enfoque del presente trabajo de integración curricular es cualitativo-cuantitativo, dado que no solo analiza las cualidades de los agentes robóticos y del entorno diseñado en ROS

– Gazebo, sino que también analiza la eficiencia de la aplicación acorde al comportamiento tipo enjambre de los robots. Adicionalmente, el tipo de trabajo realizado es de carácter exploratorio, descriptivo y explicativo, pues se realizan una serie de pruebas y se ejecuta varias veces la aplicación con el fin de analizar su efectividad; justamente, se describen los resultados obtenidos con base en el comportamiento de los agentes y de su interrelación con el entorno simulado.

Para el desarrollo de la aplicación en ROS – Gazebo se sigue una serie de pasos, que abarcan desde la creación del entorno y de los robots, hasta la implementación de los algoritmos de consenso y de prevención de colisiones, la implementación de una interfaz gráfica que permita al usuario ingresar obstáculos en el entorno y la ejecución de la aplicación. Todos estos pasos se detallan en las siguientes subsecciones.

2.1. SISTEMAS MULTIAGENTES Y ENTORNOS VIRTUALES

Se inicia con el diseño del entorno virtual dentro de ROS – Gazebo, así como con el diseño y desarrollo de los agentes robóticos que fueron descritos anteriormente. El proceso para estos diseños se detalla a continuación.

2.1.1. DISEÑO DEL ENTORNO

Para el diseño del entorno se trabaja con el software de Gazebo, el cual se ejecuta dentro del sistema operativo Linux. Dentro de este sistema se trabaja únicamente con el terminal, mismo que permite que se ejecuten diferentes programas, entre ellos se encuentran Rviz y ROS, los cuales que se utilizan en este trabajo de integración curricular para el diseño y ejecución de la aplicación. El uso de dicha terminal permite además que se utilicen archivos xacro, URDF, python, entre otros.

Para acceder al software es necesario ejecutar en el terminal el comando denominado *Gazebo;* después, se accede al apartado de edición y se selecciona la opción *building editor*, de modo que sea posible utilizar las diferentes herramientas del menú de opciones. Estas, a su vez, permiten implementar paredes, muros, ventanas, puertas o escaleras en el entorno; así como la edición de colores o estructuras en las diferentes secciones que han sido creadas individualmente dentro del entorno de simulación.

Dentro de la opción *building editor* se encuentran dos pantallas, la pantalla superior que se utiliza para la edición en dos dimensiones (2D); mientras que la visualización de los objetos editados se presenta en la pantalla inferior, que trabaja en tres dimensiones (3D), como se visualiza en la Figura 2.1. Si se desea ingresar a las opciones de cada objeto que haya sido colocado en el entorno, por ejemplo, de una pared u obstáculo, se lo debe realizar

mediante la opción *inspector*. Esta permite cambiar las configuraciones de posición inicial o final de los objetos, así como manipular su largo, ancho, opacidad, color, textura, entre otros.

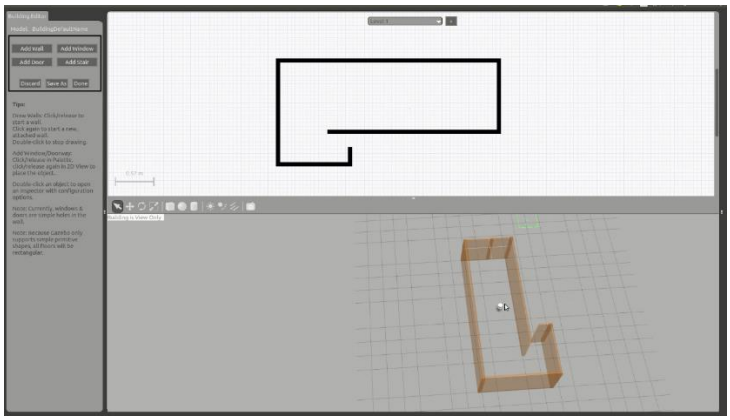


Figura 2.1. Interfaz ROS building editor [20]

Una vez terminado el diseño del entorno virtual, se procede a guardar el modelo creado en la misma opción de *building editor*; tras hacerlo, el sistema de Gazebo regresará a su interfaz principal, en la que se guardará el archivo deseado con la extensión *.world*. De esta forma, es posible diseñar diferentes entornos virtuales con los que se puede interactuar o seguir modificando a lo largo del tiempo. Para ejecutar cualquiera de estos internos se debe utilizar la terminal e introducir en la misma el código *roslaunch DirectorioArchivo Launch.*

Para el presente trabajo de integración curricular se ha diseñado un entorno conformado, inicialmente, por paredes o muros que delimitan tanto a los alrededores del entorno cuadrado como a secciones internas del mismo, como se muestra en la Figura 2.2, a continuación.

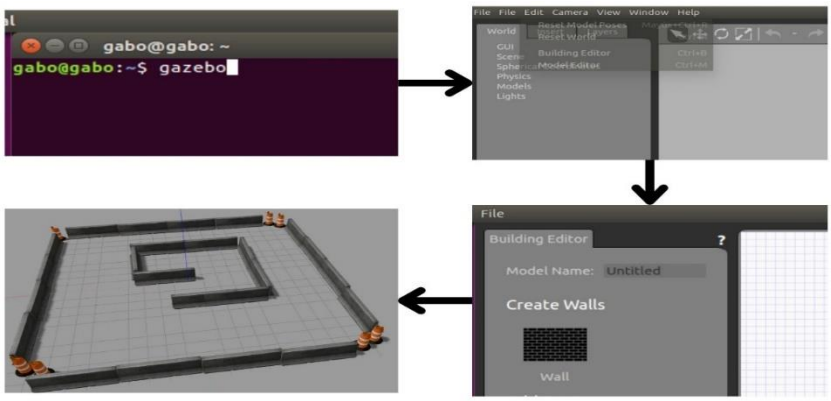


Figura 2.2. Creación del entorno simulado en ROS – Gazebo [Fuente propia]

2.1.2. DISEÑO DE LOS ROBOTS

Como se mencionó anteriormente, para este trabajo de integración curricular se han diseñado tres agentes robóticos, uno de los cuales es un robot explorador, el cual examina el entorno hasta encontrar el objetivo para, posteriormente enviar una señal a los otros dos, que actúan como robots seguidores para que se dirijan hacia la posición del objetivo. Este modelo se denomina *líder – seguidor* y tiene múltiples aplicaciones tanto en la vida cotidiana como en la industria. Por ejemplo, este tipo de aplicación resulta ser de suma utilidad en la búsqueda y rescate de víctimas de desastres naturales. El robot explorador sería el encargado de buscar a las víctimas que han quedado atrapadas entre los escombros o en lugares de difícil acceso para los rescatistas humanos y, una vez que este encuentre a una víctima, se detendría en esa ubicación y enviaría una señal a los robots seguidores; mismos que pueden llevar implementos como medicina o alimentos a las víctimas o, incluso, podrían rescatarlas. Además, los rescatistas humanos podrían conocer el estado de la víctima gracias a la cámara que, en este caso, se asume que posee el robot explorador.

Todos los agentes robóticos del sistema multi-agente propuesto son robots móviles diferenciales; para ello, se utiliza el controlador driver de ROS – Gazebo, denominado *dif_driver_controller*. Este permite controlar a los sistemas de ruedas motrices diferenciales mediante la generación de comandos de velocidad; mismos que están encargados del control tanto de la velocidad lineal como de la velocidad angular de cada agente [21]. Sin embargo, existen diferencias entre los agentes explorador y seguidores que no sólo radican en las actividades que se les han sido asignadas y en que el robot explorador posee una cámara, sino también en su robustez, su velocidad y el tipo de sensores que posee cada tipo de robot.

Así, el robot explorador tiene un diseño más robusto dado que es más alto y más largo que los robots seguidores, pues tiene una altura de 0.08 metros y un radio de 0.5 metros, mientras que los robots seguidores tienen una altura de 0.05 y un radio de 0.3 metros, pero la velocidad del robot explorador es menor; esto es debido a que el robot explorador, al ser el primero en moverse por el entorno, necesita ser más lento de modo que identifique claramente los objetos de su entorno. Además, es más robusto dada la bioinspiración, pues las hormigas líderes, en las que se inspira este diseño, por lo general son más robustas que las hormigas obreras. Del mismo modo, a pesar de que todos los robots cuentan con sensores láser de proximidad que les permiten detectar objetos cercanos u otros robots en el entorno, la diferencia radica en que el robot explorador también posee un sensor cámara

que permite al usuario visualizar el entorno a la vez que el robot se moviliza por el mismo. Estas diferencias se detallan en las siguientes subsecciones.

2.1.2.1. Robot Explorador

Como se mencionó anteriormente, la idea de crear sistemas de robots tipo enjambre surge a partir de la bioinspiración, siendo uno de sus mayores representantes los enjambres de hormigas; en estos existen hormigas consideradas como líderes o exploradoras, pues son las encargadas de guiar al grupo para el logro de los objetivos y de explorar los entornos para asegurarse de que sean seguros para el resto del enjambre o para encontrar los caminos más adecuados a seguir por el mismo. Las hormigas exploradoras comunican sus hallazgos y dan las órdenes al resto de la colmena mediante el uso de feromonas, gracias a las mismas, es posible que hormigas que se encuentren a una distancia considerable puedan encontrar su camino y dirigirse a la ubicación de la hormiga líder [22].

El funcionamiento del sistema tipo enjambre planteado en el presente trabajo de integración curricular es similar al mencionado en el párrafo anterior, con la diferencia de que se trabaja con robots y el funcionamiento del mismo se da gracias a los algoritmos implementados, mismos que se explican en la Sección 2.2, donde se aplican los mismos principios que usan los súper organismos naturales. Es así que se ha diseñado un robot explorador, que es más alto y largo que los seguidores y que será el encargado de recorrer el entorno virtual en busca del objetivo, que es una esfera de color verde claro.

Para lograrlo, el robot líder o explorador cuenta con un sensor láser de proximidad, que le permite detectar y evitar objetos, obstáculos u otros robots que se encuentren cerca de este robot en el entorno virtual, los casos para la detección y toma de decisiones al respecto serán detallados en la Sección 2.3.1. Además, este robot también posee un sensor cámara; es decir, posee una cámara colocada en la parte delantera de su cabecera, de modo que se puede visualizar, en una pantalla diferente a la del entorno, el recorrido del robot y los objetos u obstáculos cercanos al mismo; así, el usuario podrá conocer el camino y sus componentes sin necesidad de encontrarse presente en el lugar y gracias a la ayuda del sensor cámara colocado en el robot. El detalle de funcionamiento de este sensor se encuentra más adelante en la Sección 2.2.2.

2.1.2.2. Robots Obreros

Recordando que el sistema tipo enjambre planteado en el presente trabajo de integración curricular basó su inspiración en los enjambres de hormigas, se puede relacionar a los robots seguidores u obreros con las hormigas obreras que son miembros de una colmena;

estas son más pequeñas que las hormigas exploradoras, pero son más rápidas y ágiles que las mismas.

Es necesario que el diseño de los robots seguidores u obreros siga el mismo patrón dado que ya no será necesario que los mismos exploren el entorno con detenimiento en busca del objetivo, sino que sólo se deben dirigir hacia las coordenadas del objetivo, dadas por el robot explorador, mientras evitan los obstáculos con los que se encuentren dentro del entorno. Por este motivo, los robots seguidores cuentan con un sensor láser de proximidad, de modo que puedan detectar y evadir a los objetos del entorno, pero no cuentan con el sensor cámara, pues ya no se necesita visualizar el camino que recorren los mismos para llegar al objetivo. Así, las diferencias principales entre los robots explorador y los obreros radica tanto en su largo, altura y velocidad, como se mencionó anteriormente, y en el sensor cámara, donde el primer tipo de robot la posee y el segundo tipo no. El funcionamiento autónomo del sistema con los robots explorador y obreros diseñados se explicará detalladamente más adelante, en la Sección 2.4.

El diseño de los robots empleado en este trabajo utiliza el modelo cinemático, mismo que sirve para calcular a la velocidad angular de las llantas derecha e izquierda del robot de tracción diferencial; dichos valores están relacionados con el controlador *driver cinemático*. A partir de la Figura 2.3, es posible detallar las ecuaciones matemáticas que permiten el uso del controlador diferencial, mismas que se presentan a continuación [21].

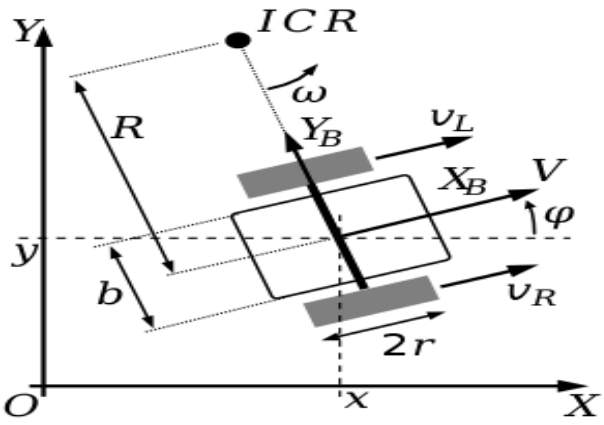


Figura 2.3. Establecimiento de variables para el modelo del robot diferencial [21]

En esta figura se presentan las siguientes variables:

v - velocidad lineal

ω - velocidad angular

X y Y - sistema de referencia inercial o global

X_B y Y_B - sistema de referencia atado al agente

φ - ángulo del robot con respecto al sistema de referencia inercial

r - radio de las ruedas

b - ancho del agente

ICR - centro de rotación instantánea

v_L, v_R - velocidad de contacto con el suelo ruedas izquierda y derecha respectivamente

ω_L, ω_R - velocidad angular ruedas izquierda y derecha respectivamente

Para empezar, se deben calcular las velocidades de contacto con el suelo de las ruedas derecha e izquierda de los agentes. Esto se realiza mediante el uso de fórmulas matemáticas [21], dadas a continuación:

$$\omega \left(R + \frac{b}{2} \right) = v_R \tag{2.1}$$

$$\omega \left(R - \frac{b}{2} \right) = v_L \tag{2.2}$$

Al resolver las ecuaciones (2.1) y (2.2) se obtienen las ecuaciones para la velocidad angular ω, que es la velocidad a la que rota el agente, y para la distancia R entre el Centro de Rotación Instantánea (ICR) y el centro del robot, como se evidencia en las ecuaciones (2.3) y (2.4).

$$\omega = \frac{V_R - V_L}{b} \tag{2.3}$$

$$R = \frac{b}{2} * \frac{V_R + V_L}{V_R - V_L} \tag{2.4}$$

A partir de las ecuaciones (2.3) de la velocidad angular y (2.4) de la distancia se obtiene la ecuación (2.5) de la velocidad instantánea, que se calcula en el punto medio entre las ruedas del robot.

$$V = \omega R = \frac{V_R + V_L}{2} \tag{2.5}$$

La ecuación anterior se encuentra en función de las velocidades tangenciales $\mathrm{V_R}$ y $\mathrm{V_L}$, tanto de la rueda izquierda como de la derecha del robot, las cuales se muestran en las ecuaciones (2.6) y (2.7), respectivamente.

$$V_R = r\omega_R \tag{2.6}$$

$$V_L = r\omega_L \tag{2.7}$$

Utilizando la ecuación (2.5), que hace referencia a la velocidad instantánea de las ruedas derecha e izquierda del robot diferencial, se establecen las siguientes ecuaciones, que representan el movimiento del robot y donde x_0, y_0 representan a la posición inicial del robot, x_f, y_f a su posición final y t al tiempo.

$$x_f = x_0 + Vcos(\varphi)t \tag{2.8}$$

$$y_f = x_0 + Vsen(\varphi)t \tag{2.9}$$

Para el cambio de orientación de los robots se utiliza la ecuación (2.3), donde la velocidad angular permite el giro del robot; la nueva orientación de dicho robot se representa con la ecuación (2.10).

$$\varphi_f = \varphi_0 + \omega t \tag{2.10}$$

Con el uso de las tres últimas ecuaciones se establece la aproximación de la resolución cinemática del robot diferencial, lo cual se observa en la ecuación (2.11), donde $\dot{x}$, $\dot{y}$ y $\dot{\varphi}$ representan a la variación de movimiento en los ejes *x* e *y,* y a la orientación, respectivamente, las mismas dependen de la velocidad tangencial y de la velocidad angular, con las cuales se controla tanto el giro del robot respecto a su eje como la velocidad lineal con la que el mismo se desplazará en el entorno.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} cos(\varphi(t)) & 0 \\ sen(\varphi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{2.11}$$

Las variaciones de las dos velocidades de control mencionadas generan diferentes casos para el movimiento del robot, mismos que se detallan a continuación:

- Si $v(t) = 0$ y $\omega(t) > 0$, el robot gira sobre su eje en sentido horario.
- Si $v(t) = 0$ y $\omega(t) < 0$, el robot gira sobre su eje en sentido antihorario.
- Si $\omega(t) = 0$ y $v(t) > 0$, el robot se moverá en línea recta hacia el frente.
- Si $\omega(t) = 0$ y $v(t) < 0$, el robot se moverá en línea recta hacia atrás.
- Si $\omega(t) = 0$ y $v(t) = 0$ el robot se detendrá.

El análisis matemático descrito anteriormente puede ser unificado con el lenguaje de programación URDF para general los movimientos del robot. Sin embargo, otra forma de

hacerlo es mediante el uso del paquete driver de controlador diferencial de ROS, el cual permite ingresar directamente el tópico de comunicación para las velocidades tangencial y angular. Esto generará posibles casos para el movimiento de los robots, mismos que se describen en la Sección 2.3.1.

Los robots del presente trabajo de integración curricular trabajan con un ángulo de giro de 90°. El valor del ángulo puede variar acorde a la necesidad de cada diseñador, en este caso se ha decidido que sea de 90°, ya sea en sentido horario o antihorario, para que el movimiento del robot siempre sea paralelo a los ejes de coordenadas. Con este ángulo la matriz que representa la aproximación de la resolución cinemática del robot diferencial toma el valor de *1 rad/s* para la velocidad angular y de *0.6 m/s* para la velocidad lineal del movimiento de las ruedas del robot. Dichos valores fueron seleccionados de forma heurística con base en el trabajo realizado por [23], además de los resultados favorables obtenidos para el funcionamiento del controlador drive dentro de ROS – Gazebo.

La información específica para el robot diferencial se distribuye en cuatro archivos tipo xacro, que son detallados a continuación:

- **Structurelaser.xacro:** Cada robot cuenta con un láser que le permite detectar a los objetos del entorno dentro de un rango determinado para así evitar colisionar con los mismos. Por ello, este archivo xacro contiene información sobre la forma, la luminosidad, el difuminado, la estructuración y las dimensiones del láser.
- **Macro.xacro:** Como se mencionó anteriormente, cada robot está formado por dos ruedas, una izquierda y una derecha. Este archivo contiene información sobre características como el origen, la estructura, la geometría, las propiedades y la reacción en caso de colisión de dichas ruedas.
- **Gazebo.xacro:** Este archivo contiene información sobre el color y el plugin para el control diferencial del robot; así como del plugin para el sensor del láser y el plugin para el control de la cámara que posee el robot.
- **Robot.xacro:** Este archivo contiene los links que permiten que se unifique toda la información que se encuentra distribuida entre los archivos xacro enlistados anteriormente; además de información que permite estructurar el chasis o cuerpo del robot, sus dimensiones, su color, su estructura y su reacción en caso de colisión con objetos del entorno o con otros robots.

A continuación, se presenta la estructura de los parámetros utilizados en la configuración del cuerpo del robot; es decir, del chasis, para el cual se han establecido coordenadas de

origen, valores de inercia respecto a los ejes tridimensionales y el largo, radio y masa de la geometría del chasis, siguiendo la estructura mencionada en la Sección 1.4.3.3. Los parámetros especificados fueron elegidos con base en el trabajo de [23] y por medio de prueba y error hasta que se logró que el chasis de los robots obreros tenga una altura de 0.08 metros, un radio de 0.3 metros y una masa de 5 kilogramos. Lo mencionado se observa en la Tabla 2.1. La parametrización completa del resto de archivos .xacro para el diseño del robot se presenta en el Anexo II.

Tabla 2.1. Parametrización de un archivo Robot.xacro. [Fuente propia]

Parte	Parámetros
Estructuración del archivo xacro	Versión: 1 Nombre: ejemplo
Inicio: link	Link chasis
Propiedades: Inercia, colisión y visualización, entre otros.	Masa= 5 Origen= rpy="0 0 0" xyz="0 0 0.1" Inercia= ixx="0.03" ixy="0" ixz="0" iyy="0.10 iyz="0" izz="0.106 Geometría= length="0.08" radius="0.3"
Cierre: link	Link chasis

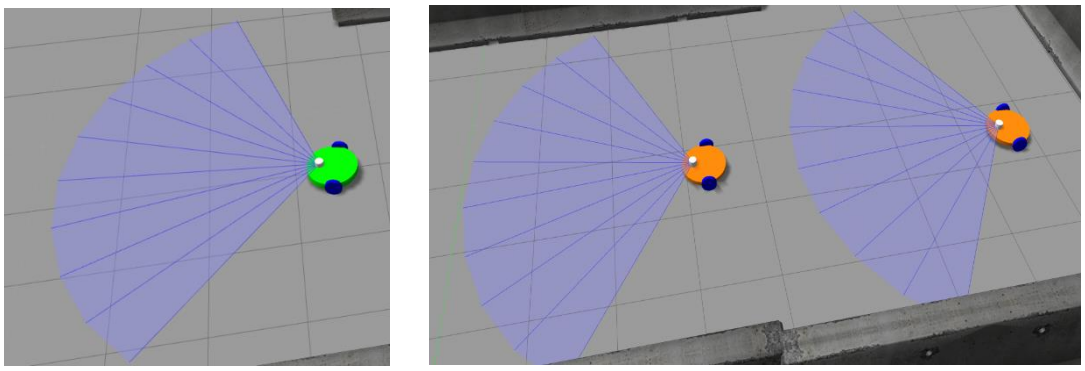
En la siguiente sección se detalla el proceso para la obtención, lectura y procesamiento de los datos.

2.2. OBTENCIÓN, LECTURA Y PROCESAMIENTO DE DATOS

Para poder obtener, leer y procesar los datos se debe trabajar en conjunto tanto con los paquetes de ROS y Rviz como con la interfaz de Gazebo, de modo que se estructuren nodos que se comunican entre sí a través de los diferentes tópicos, como se mencionó en la Sección 1.4.3.3.

Para el presente trabajo de integración curricular se instalaron sensores láser de proximidad en los dos tipos de robots; es decir, tanto en el robot explorador como en los robots seguidores, con el fin de otorgar a cada uno un rango de detección de su entorno y poder prevenir, de esta forma, que los mismos colisionen con objetos del entorno o con otros robots. Gracias a los sensores es posible estructurar un área segura de trabajo para los agentes, mismos que siempre se dirigen hacia adelante, pero que, cuando detectan a algún objeto cambian la dirección de su curso para dirigirse hacia otra sección del entorno. Todo lo mencionado es realizado gracias a los certificados de barrera con los que cuenta

cada robot, concepto que será explicado con mayor detalle en las siguientes sub-secciones. El rango de los sensores para los robots explorador y seguidores son ejemplificados en las Figuras 2.4 (a) y 2.4 (b), respectivamente.



(a) (b)

Figura 2.4. Sensores implementados en el robot explorador (a) y en los robots seguidores (b) [Fuente propia]

Pero la detección de objetos no es el único fin del detector láser implementado en cada agente; sino que también es posible medir la distancia entre el robot y el objeto del entorno que ha sido detectado y la distancia que tienen los robots entre sí. Del mismo modo, el robot explorador cuenta con una cámara, que permite al usuario visualizar el entorno a medida que el robot se moviliza por el mismo; los robots obreros no poseen este sensor. Estas funciones, así como la configuración y parametrización de los sensores láser de proximidad de todos los robots y del sensor cámara del robot explorador se detallan en las siguientes subsecciones.

2.2.1. MEDICIÓN DE DISTANCIA

Para empezar, se instala el sensor láser con el plugin denominado *gazebo_ros_head_hokuyo_controller* y con el directorio o archivo llamado *libgazebo_ros_laser.so*. Posteriormente, se estructuran parámetros que permiten que se genere y utilice dicho sensor, como se muestra en la Tabla 2.2.

Tabla 2.2. Instalación y uso del sensor láser de proximidad [Fuente propia]

Parte	Código
Inicio: link referencia	Sensor laser
Tipo	Tipo = ray
Propiedades	Pose = 0 Visualize = 1 Update rate = 30

	Samples = 10 Resolution = 1
Plugin	plugin name = gazebo_ros_head_hokuyo_controller filename = libgazebo_ros_laser.so topicName = robot/laser/scan frameName = sensor_laser
Cierre: link	Sensor laser

La tabla anterior muestra que la parametrización de propiedades permite que se estructuren las características del sensor láser de proximidad dentro del entorno virtual. Además, es posible generar tópicos, que permiten que los nodos o archivos se comuniquen entre sí, y generar atributos para los campos establecidos. Los parámetros especificados fueron elegidos adecuando lo propuesto en [23], a través de un método heurístico, estableciendo así una frecuencia de actualización de 30 hercios, así como permitiendo la visualización del campo de acción del láser de los robots dentro de la simulación en el mundo virtual. La diferencia entre los parámetros establecidos en el presente trabajo de integración curricular y los propuestos por [23] radica en que los aquí obtenidos son útiles para la simulación del enjambre de robots en el mundo virtual del software Gazebo; mientras que los establecidos por [23] fueron usados para la implementación física de los robots creados para su investigación.

La lectura de la información se realiza mediante el uso de una subrutina que inicializa a un nodo específico y obtiene los datos del mismo a través de un plugin que se encuentra en dicho nodo. Posteriormente, se define una variable global denominada *pub* para permitir la escritura y subscripción del mencionado nodo en la red de comunicación de ROS, como se muestra en la Figura 2.5; dicho nodo de comunicación permite que la información sea transmitida de forma continua acorde a la velocidad de definición de los mensajes. Asimismo, este nodo permite obtener información mediante la lectura del sensor, que envía dicha información al comando de velocidad del controlador diferencial driver mediante el tópico de velocidad *cmd_vel.* Esto permite que se tomen decisiones para prevenir colisiones utilizando, a la vez, un algoritmo, cuyas características serán detalladas más adelante. Entonces, la recopilación y lectura de la información proporcionada por el sensor láser de proximidad es posible gracias a los diferentes plugins que se entrelazan entre sí y se unifican dado el uso de archivos *.xacro*, *.urdf* y *.py.*

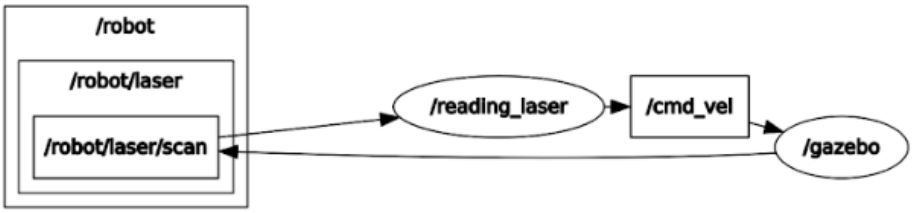


Figura 2.5. Subscripción de un nodo de proximidad en la red de comunicación de ROS [Fuente propia]

El proceso de lectura de información y de procesamiento de datos se encierra en un bucle infinito mediante el código *rospy.spin(),* permitiendo el que el sensor registre continuamente la información de proximidad entre el robot y los objetos del entorno y publique permanentemente dicha información en el terminal de ROS.

2.2.2. VISIÓN ARTIFICIAL

Como se mencionó en la Sección 2.1.2, el robot explorador posee una cámara en la parte frontal de su cabecera; con la misma, el usuario puede observar las imágenes del entorno que son captadas por el robot en forma de video. Esto resulta ser de suma utilidad en aplicaciones como, por ejemplo, en la búsqueda de recursos, pues los usuarios podrán conocer el entorno y registrar los objetos del mismo gracias a las imágenes proporcionadas, de modo que pueden elegir rutas de acción o conocer de mejor manera el escenario en el que deben trabajar.

Para el procesamiento de dichas imágenes se utiliza el paquete y plugin de sensor de cámara proporcionados por ROS, mismos que permitirán no solo obtener los datos mencionados, sino también procesar la información obtenida y hacer que la búsqueda del objetivo sea más eficiente. En el caso del presente trabajo de integración curricular, el objetivo es una esfera de color verde. Para este caso, se registrará información sobre las imágenes del camino del robot explorador hasta encontrar la esfera y de la distancia que tiene el robot con la esfera una vez que la haya encontrado.

Para este fin, se utiliza un algoritmo de detección de color y forma. Este algoritmo se encontrará funcionando constantemente y detectará los colores y formas de cada objeto que ingrese en el rango alcanzado por el sensor láser de proximidad en el entorno virtual. Cuando el robot detecte una forma esférica que tenga el color verde, este se detendrá, pues habrá encontrado su objetivo, y enviará una señal a los robots seguidores para que se dirijan hacia este objetivo. Para la implementación de dicho sensor se utiliza el plugin denominado *camera_controller* junto con el archivo o directorio llamado *libgazebo_ros_camera.so,* para, posteriormente, estructurar parámetros que permitan que

se implemente y utilice el sensor cámara en el robot explorador, como se muestra en la Tabla 2.3.

Tabla 2.3. Instalación y uso del sensor cámara en el robot explorador [Fuente propia]

Parte	Código
Inicio: link referencia	sensor head_camera
Tipo	Type= camera
Propiedades	update_rate = 30.0 camera name = head_camera pose = 0.0 0.21 0.0 0.0 -0.8 -1.570796327 horizontal_fov = 1.3962634 image width = 960 Image height = 600 Format = R8G8B8 Near = 0.02 Far = 300
Plugin	plugin filename= libgazebo_ros_camera.so name= camera_controller alwaysOn = true updateRate = 0.0 cameraName = head_camera imageTopicName = image cameraInfoTopicName = camera_info frameName = head_camera hackBaseline = 0.07 robotNamespace = cameras
Cierre: link	sensor

Para la obtención y lectura de información por parte del sensor cámara se deben iniciar a las diferentes clases a utilizar; es decir, se especifican los rangos de colores, los estados iniciales y las constantes que se utilizarán, como se muestra en la Tabla anterior. Similar al caso anterior, los parámetros especificados fueron elegidos adecuando lo propuesto por [23], estableciendo así un ancho de 960 pixeles, un alto de 600 pixeles y una frecuencia de actualización de 30 hercios. Como se mencionó en la Sección 2.2.1, los parámetros establecidos en el presente trabajo de integración curricular son útiles para la simulación del enjambre de robots en el mundo virtual.

En cuanto al procesamiento de la información obtenida por este sensor, ROS recopila y guarda las imágenes detectadas en forma de mensajes, cambiando las imágenes al formato *OpenCv2* y usando la biblioteca *CvBridge.* Con este procedimiento, las imágenes se convierten al formato *HSV*; es decir, se codifica para su transferencia en forma de datos.

Todo lo mencionado en la parte anterior se estructura en el algoritmo de color. Este algoritmo selecciona los diferentes objetos detectados en el entorno virtual dependiendo de las máscaras que hayas sido establecidas en el formato HSV, permitiendo que el plugin seleccione los colores y formas, de modo que las evite o las detecte como objetivo.

Dicha detección del objetivo por parte del robot explorador es posible mediante el uso de un algoritmo de detección de colores que está implementado en conjunto con la cámara colocada en la parte frontal de la cabecera de dicho robot. Este algoritmo permite que se detecten no solo los colores buscados, sino también las diferentes tonalidades de los colores, según sea el rango que se estructure para la detección de los mismos. En este caso, para el color verde del objetivo, se define una escala HSV de rango mínimo (50, 100, 50) y rango máximo (70, 255, 255). Este tipo de escalas analizan principalmente el tono (H), la saturación (S) y el brillo (V), definiendo así a las diferentes tonalidades de colores en términos de sus componentes o de las variaciones del mismo.

Así como se ha estructurado e implementado el sensor de imagen, también es posible incluir al tag de comunicación denominado *image_to_initial_state,* que permite la subscripción y comunicación de los nodos a la red de ROS; así como la transferencia e impresión de la información, como se observa en la Figura 2.6.

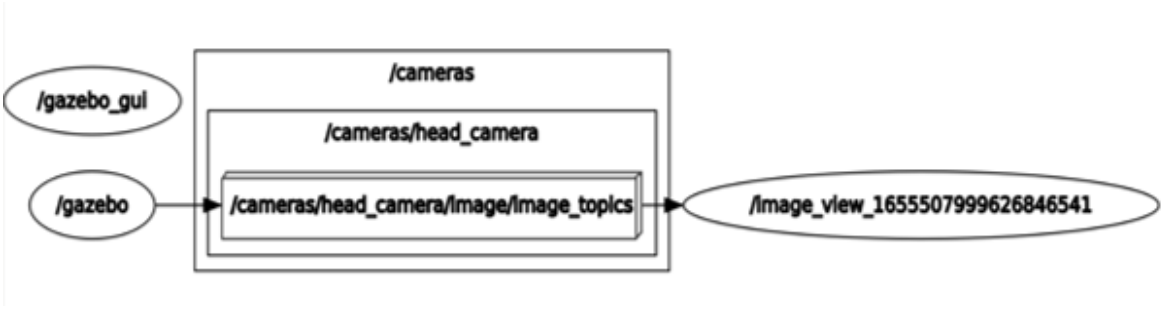


Figura 2.6. Transmisión de la información recopilada por el sensor cámara del robot explorador [Fuente propia]

Cabe mencionar que el número extenso que se visualiza en el recuadro final de la Figura 2.7 presentada anteriormente corresponde a un número aleatorio, mismo que se utiliza únicamente para asignar un nombre al nodo, motivo por el cual puede variar con cada ejecución de la simulación, evitando de este modo la superposición de los nodos.

La visualización de las imágenes en tiempo real y en forma de video se realiza mediante el software Rviz, el cual también permite recopilar datos y detalles a partir de los sensores tipo cámara dentro de ROS – Gazebo. Un ejemplo de la visualización de imágenes se

presenta en la Figura 2.7, en la que el robot explorador detecta al objetivo en el entorno virtual.

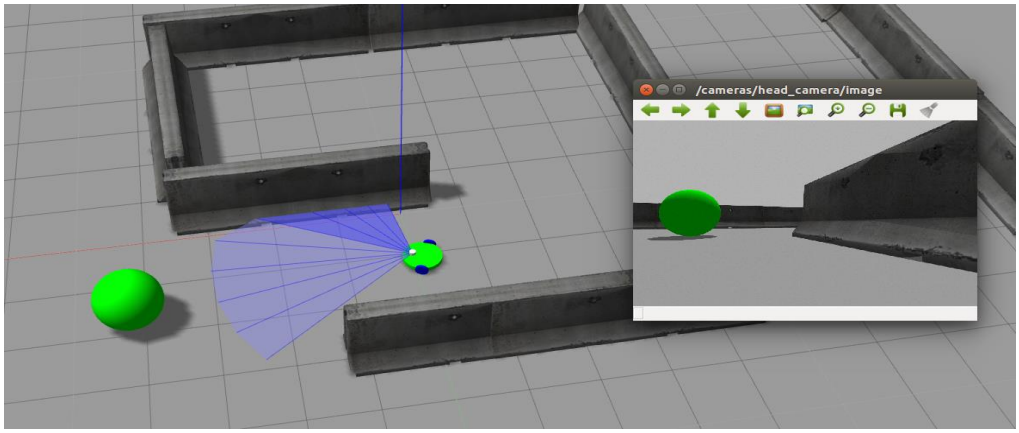


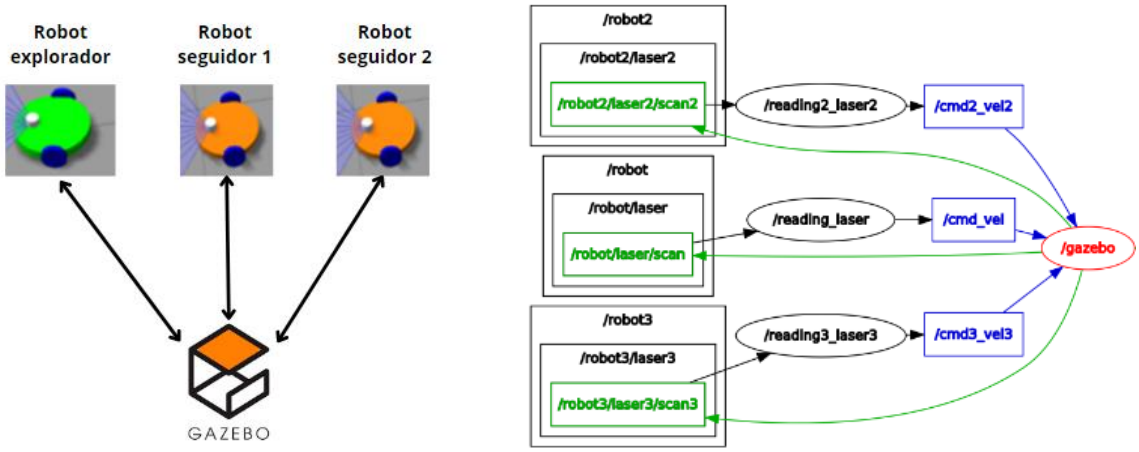
Figura 2.7. Visualización del entorno virtual a través del software Rviz y del sensor cámara del robot explorador [Fuente propia]

2.3. ALGORITMOS CON CERTIFICADOS DE BARRERA

Los algoritmos con certificados de barrera son aquellos que permitan que los agentes miembros del sistema no colisionen entre sí o con otros objetos del entorno, mientras se encuentran trabajando en conjunto, llegando a un consenso sobre el alcance del objetivo planteado. El algoritmo que se utiliza en este proyecto para evitar que el sistema de robots colisione cuando los robots se encuentran unos con otros o con otros objetos del entorno es el algoritmo de prevención de colisiones, mismo que utiliza la metodología de *consultas de proximidad* para permitir que los agentes adquieran información relevante de su entorno. El uso de este algoritmo es primordial dado que el entorno virtual planteado es dinámico; es decir que no solo contiene obstáculos fijos pre establecidos, sino que se permite que el usuario introduzca nuevos obstáculos en una posición del mapa virtual. Así, el uso de este algoritmo permite estructura una barrera al detectar la información mencionada mediante el uso del sensor láser; este sensor es un paquete de ROS, pero se ha programado en el mismo un ángulo total de 120° para la recopilación de datos a partir de objetos que se encuentran alrededor del agente.

Otro algoritmo con certificados de barrera que se utiliza en el presente trabajo de integración curricular es el de consenso, el cual aplica la técnica de *consenso distribuido con el uso de grafos de comunicación dirigida* para permitir que sea posible la comunicación entre los agentes mediante diferentes tópicos que son estructurados en la programación del nodo maestro y los nodos esclavos, como se observa en la Figura 2.8. A continuación,

se detalla el funcionamiento de los algoritmos de prevención de colisiones y de consenso mencionados.



(a) (b)

Figura 2.8. a) Grafos de comunicación dirigida, b) Conexión entre nodo maestro y agentes robóticos. [Fuente propia]

2.3.1. PREVENCIÓN DE COLISIONES

Para que el algoritmo de prevención de colisiones pueda ser implementado y utilizado dentro del entorno de ROS – Gazebo, en primer lugar, se utiliza el nodo denominado *Reading_laser*, que contiene la estructura de dicho algoritmo. Después se definen las variables de suscripción y lectura, que tienen el nombre de *pub* y *sub* respectivamente, estas son variables globales. Tras haber definido las variables se importan las librerías y se definen las constantes para las regiones de acción del láser, mismas que estarán divididas en tres subsecciones o regiones: derecha, izquierda y al frente, donde cada una tiene un rango de acción de 40°; esto es debido a que el rango total de detección del sensor para este algoritmo es de 120°, como se mencionó al inicio de esta sección, por lo que, al dividir equitativamente el rango total de acción entre tres, corresponde a cada región el campo de acción de 40° mencionado.

Entonces, la posición de objetos, obstáculos u otros robots será detectada si los mismos se encuentran en cualquiera de las tres regiones indicadas, generando diferentes casos de detección de objetos y toma de decisiones para el curso a seguir por los robots. Además, se utiliza una velocidad fija, también conocida como *de referencia* o *de consigna,* para los robots, esta es la velocidad lineal, que será conocida en adelante como *vref* y una velocidad

angular que será conocida como *wref*, cuyos signos y valores también varían dependiendo de cada uno de los casos que se detallan a continuación.

Así, la lectura del sensor es obtenida mediante el requisito de lectura del plugin denominado *laser scan*, además del uso de condicionales de acción según la información que ha sido censada; misma que se adquiere de las regiones del frente, de la derecha y de la izquierda del agente. Se utiliza también una distancia de 1.5 metros para cada robot, la cual es la mínima distancia para detectar objetos cercanos al robot; de este modo, dependiendo de la posición del obstáculo en comparación con el robot se pueden obtener diferentes casos, los cuales son:

Caso 1: Cuando no existen obstáculos todas las regiones que se encuentran alrededor del robot (frente, derecha e izquierda) son mayores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *No hay obstáculo*; además, en este caso, el robot se mueve hacia el frente con una velocidad vref y no posee velocidad wref.

Caso 2: Cuando existen obstáculos al frente del robot, la región del frente será menor a la distancia de ruptura, mientras que las regiones de la izquierda y la derecha del robot serán mayores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *De frente*; además, en este caso, el robot rota sobre su eje en sentido horario a una velocidad wref y no posee velocidad vref.

Caso 3: Cuando existen obstáculos a la derecha del robot, la región de la derecha será menor a la distancia de ruptura, mientras que las regiones del frente y de la izquierda del robot serán mayores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *Derecha*; además, en este caso, el robot rota sobre su eje en sentido antihorario a una velocidad wref y no posee velocidad vref.

Caso 4: Cuando existen obstáculos a la izquierda del robot, la región de la izquierda será menor a la distancia de ruptura, mientras que las regiones del frente y de la derecha del robot serán mayores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *Izquierda*; además, en este caso, el robot rota sobre su eje en sentido horario a una velocidad wref y no posee velocidad vref.

Caso 5: Cuando existen obstáculos al frente y a la derecha del robot, la región de la izquierda será mayor a la distancia de ruptura, mientras que las regiones del frente y de la derecha del robot serán menores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *De frente y derecha*; además, en este caso, el robot rota sobre su eje en sentido antihorario a una velocidad wref y no posee velocidad vref. Como

se observa, este caso es similar al caso 3 detallado anteriormente, pero su diferencia radica en que, en este caso, también existen obstáculos al frente del robot, mientras que en el caso 3 sólo hay obstáculos a su derecha.

Caso 6: Cuando existen obstáculos al frente y a la izquierda del robot, la región de la derecha será mayor a la distancia de ruptura, mientras que las regiones del frente y de la izquierda del robot serán menores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *De frente e izquierda*; además, en este caso, el robot rota sobre su eje en sentido horario a una velocidad wref y no posee velocidad vref. Como se observa, este caso es similar al caso 4 detallado anteriormente, pero su diferencia radica en que, en este caso, también existen obstáculos al frente del robot, mientras que en el caso 4 sólo hay obstáculos a su izquierda

Caso 7: Cuando existen obstáculos al frente, a la derecha y a la izquierda del robot, todas las regiones que se encuentran alrededor del robot (frente, derecha e izquierda) son menores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *Derecha, izquierda y frente*; además, en este caso, el robot retrocede y rota sobre su eje en sentido horario con velocidades vref y wref.

Caso 8: Cuando existen obstáculos a la derecha y a la izquierda del robot, la región del frente será mayor a la distancia de ruptura, mientras que las regiones de la derecha y de la izquierda del robot serán menores a la distancia de ruptura. El texto que será publicado en el terminal lleva la inscripción *Derecha e izquierda*; además, en este caso, el robot sigue su camino hacia el frente con una velocidad vref y no rota sobre su eje ya que no ha detectado obstáculos al frente sino únicamente a los costados. Como se observa, este caso es similar al caso 7 detallado anteriormente, pero su diferencia radica en que, en este caso, no existen obstáculos al frente del robot, mientras que en el caso 7 hay obstáculos tanto al frente como a los dos costados del mismo.

La velocidad lineal (*vref*) y angular (*wref*) de referencia para todos los casos son de 0.6 metros por segundos y de 1 radián por segundo respectivamente, pero sus signos y, por ende, su valoración, cambian dependiendo de si el robot rota en sentido horario o antihorario o si no rota en absoluto y sólo sigue un recorrido de frente, todo esto se menciona en cada uno de los casos que fueron detallados.

Tras haber analizado el caso en el que se encuentra el robot, se publican los estados y se carga la información de las variables globales que han sido definidas, permitiendo, de esta forma, que se produzca la escritura y la lectura de los tópicos por parte de los nodos, lo que, a su vez, implica que los nodos se comuniquen entre sí. Finalmente, se crea un lazo

infinito de ejecución del algoritmo mediante el uso del comando *rospy.spin*, el cual no sólo permite la existencia del bucle, sino también hace que sea posible que la lectura de los datos sea constante, de modo que los robots puedan tomar decisiones eficientes a tiempo acorde a cada caso de obstáculo que encuentre, como se visualiza en el diagrama de flujo presentado en el Anexo III.

2.3.2. CONSENSO

Como se mencionó anteriormente, el algoritmo de consenso utilizado en este trabajo de integración curricular utiliza la metodología de comunicación entre nodos; es decir, la comunicación de tópicos que parten del nodo maestro hacia los nodos esclavos. Estos, a su vez, se encuentran solicitando información constantemente sobre el estado en el que se encuentra el robot explorador, con el fin de saber si el mismo ya encontró a la esfera verde, que es el objetivo. Dicha comunicación es ininterrumpida y bidireccional entre los robots explorador y seguidores y permite que los mismos conozcan tanto las coordenadas de ubicación como la distancia con los otros.

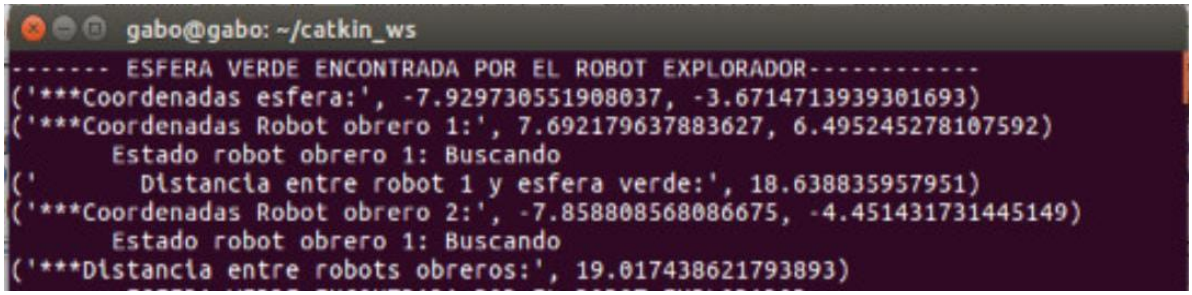
Así, al inicio los robots seguidores reciben constantemente información sobre la ubicación del robot explorador y miden la distancia que tienen con el mismo, mientras que el robot explorador recibe también información sobre la ubicación de los seguidores y calcula la distancia con los mismos; esto es debido a que al inicio el estado de los robots obreros es *en reposo* dentro del entorno, mientras que el robot explorador se encuentra *en movimiento*. Después, una vez que el robot explorador encuentra al objetivo, este se detiene y envía a los robots seguidores información sobre las coordenadas del objetivo mientras recibe información sobre las coordenadas cambiantes en las que se encuentran los seguidores a medida que avanzan por el entorno y mide la distancia que tiene con cada uno de ellos, dado que al recibir la señal del robot explorador los robots seguidores inician su recorrido. A la vez, los robots seguidores reciben las coordenadas enviadas por el robot explorador y las coordenadas del otro robot seguidor, por lo que también miden la distancia que tienen con cada uno de los otros robots. De esta forma, todos los robots trabajan en consenso para cumplir con un objetivo en común mientras intercambian información sobre sus coordenadas y estados *en reposo* o *en movimiento* y calculan la distancia con los otros robots, de modo que, aunque cada robot realiza una acción independiente de las del resto, el sistema, en su conjunto, funciona como un solo agente autónomo y eficiente.

Como se explicó, el cálculo de la distancia entre el robot explorador y los robots obreros y de estos últimos entre sí se realiza como parte del algoritmo de consenso y es importante debido a que conocer constantemente las posiciones de los otros y calcular las distancias

mencionadas permite que los robots obreros se direccionen hacia las coordenadas del objetivo, una vez que sean enviadas por el robot explorador, y determinen si la distancia entre ellos con el objetivo disminuye durante su trayectoria, lo que quiere decir que la misma es adecuada. Además, conocer su distancia con el otro robot seguidor permite determinar cuál de los agentes seguidores llegó primero al objetivo y a qué distancia se encuentra el otro para llegar.

Toda la información sobre la comunicación entre los agentes, que se mencionó en los párrafos anteriores, se visualiza en un terminal asignado para cada robot, dados los algoritmos que poseen cada uno y que también fueron detallados anteriormente. Así, se publican continuamente los datos de las coordenadas y estados de cada agente y de sus distancias con los otros, de modo que se tiene una intercomunicación ininterrumpida de todo el sistema. Esto se observa en las Figuras 2.9. a) y b), que corresponden a la comunicación por parte del robot explorador y de un robot obrero, respectivamente.

a)



b)

Figura 2.9. Ventanas del terminal correspondientes a la información sobre las coordenadas y estado transmitidas entre los robots. a) Robot explorador, b) robot obrero. [Fuente propia]

Como se mencionó, una vez que los robots seguidores han detectado la señal de localización del objetivo enviada por el robot explorador, se inicia el movimiento de los robots seguidores y se ejecutan los algoritmos respectivos. De este modo los robots seguidores se dirigen hacia el objeto de búsqueda, que ya ha sido localizado por el robot explorador; esto se muestra en el Anexo IV.

2.4. TRABAJO AUTÓNOMO DEL SISTEMA TIPO ENJAMBRE

Como se mencionó en la primera sección de este trabajo de integración curricular, un sistema tipo enjambre tiene que ver con la coordinación simultánea de múltiples robots relativamente sencillos, de modo que, en conjunto, los mismos funcionen como un sistema complejo en el que se ejecutan acciones individuales para alcanzar un objetivo en común; esto se da gracias a las constantes interacciones de los robots entre sí y con el entorno [24].

El presente trabajo de integración curricular trabaja con tres robots diferenciales: un explorador y dos seguidores, de modo que el robot explorador inicia el procedimiento al recorrer el entorno virtual en búsqueda del objetivo, mientras evita obstáculos; una vez que el robot explorador encuentra el objetivo, los robots seguidores iniciarán el recorrido para dirigirse a la ubicación del robot explorador. El procedimiento realizado para el diseño de los robots fue detallado anteriormente, en la sección 2.1.2. En las siguientes subsecciones se realiza una descripción detallada del funcionamiento de los robots como parte del sistema tipo enjambre, de modo que sea factible que los mismos contribuyan al funcionamiento autónomo del sistema.

2.4.1. ROBOT EXPLORADOR

Una vez que se haya cargado el entorno virtual y el objeto de búsqueda, el robot explorador inicia su recorrido y se activa el algoritmo de prevención de colisiones mediante el terminal, de modo que, a través del uso del sensor láser de proximidad sea posible detectar obstáculos que puedan encontrase pre-establecidos en el entorno virtual o que aparezcan de forma aleatoria en una ubicación ingresada mediante la interfaz de usuario. Tras haber detectado el obstáculo, el robot cambia su curso y dirección con el fin de evitar dicho obstáculo, de modo que el agente toma decisiones adecuadas constantemente y en tiempo real. Toda la información sobre las coordenadas y el estado del robot explorador, así como de su distancia con los robots seguidores se visualiza continuamente en el terminal en el que fue inicializado el algoritmo de consenso.

Como se mencionó en la Sección 2.1.2, en la que se detallaba el procedimiento realizado para el diseño de los robots, el robot explorador posee una cámara en la parte frontal de su cabecera. Esta cámara se encuentra ligada al algoritmo de detección de colisiones, que puede ser visualizado gracias a la herramienta de Rviz, lo que, a su vez, permite que se visualice el entorno a medida que el robot avance por el mismo. Es así que mediante el trabajo en conjunto de los algoritmos de prevención de colisiones y de detección de colores

se genera no solo una correcta movilidad del robot, sino también hace posible que se detecte de manera eficaz el objetivo buscado que, en este caso, se trata de una esfera de color verde.

Una vez que el robot explorador haya encontrado el objetivo; es decir, una vez que haya detectado el color verde y la forma esférica del objeto, el robot explorador envía a los robots obreros información sobre su estado y sobre las coordenadas de localización del objetivo y los mismos inician su recorrido para dirigirse hacia la ubicación de la esfera localizada. De esta forma, los algoritmos de consenso, de detección de colores y de prevención de colisiones funcionan simultáneamente durante todo el proceso, haciendo que el sistema actúe como un enjambre autónomo.

Además, el robot explorador mide continuamente la distancia que tiene con cada uno de los robots seguidores, de modo que es posible conocer cuál de ellos llegará primero a la ubicación del objetivo y si su recorrido es el adecuado, lo que sucederá cuando la distancia entre los robots seguidores con el robot explorador disminuya gradualmente a medida que los robots obreros se acercan a las coordenadas del objetivo, que fue previamente localizado por el agente explorador. Cabe mencionar que, a diferencia del robot explorador, los robots seguidores no poseen un sensor cámara ni un algoritmo de detección de colores, por lo que su función principal no es en buscar el objetivo, sino que únicamente se dirigen a las coordenadas dadas por el robot explorador.

2.4.2. ROBOTS OBREROS

Los robots obreros no inician su recorrido a la par del robot explorador, sino que esperan en reposo a que el robot explorador haya encontrado el objetivo; una vez que el mismo ha sido hallado, el robot explorador envía sus coordenadas a los robots seguidores, mismas que se transmiten mediante los tópicos de comunicación entre el nodo maestro, que es ROS – Gazebo, y los nodos esclavos, que son los robots seguidores. Así, tras recibir la información mencionada, los robots seguidores se dirigen hacia la ubicación en la que se encuentra el robot explorador, utilizando el algoritmo de prevención de colisiones, el cual también utiliza un sensor láser de proximidad con el fin de obtener datos del entorno y permite que los robots tomen decisiones de cambio de dirección y posición dentro del entorno virtual cuando se encuentran con algún obstáculo en su camino.

Al mismo tiempo se encuentra activo el algoritmo de consenso, gracias al cual es posible que se realice la mencionada comunicación entre los agentes, que es bidireccional; es decir, el robot explorador envía información sobre sus coordenadas y su estado a los robots

seguidores, quienes reciben la información y, a la vez, envían sus coordenadas y estado para que sean recibidas por el robot explorador y por los otros robots seguidores. Además, el algoritmo de consenso de los agentes obreros también permite calcular la distancia de los mismos con el resto de robots, de modo que es posible saber cuál de ellos llega primero a las coordenadas del objetivo, que fueron previamente enviadas por el robot explorador. La comunicación continua entre el robot explorador y los robots seguidores es la que permite que se produzca tanto la recopilación como la escritura y la lectura de datos dentro de ROS, además de controlar el comportamiento en conjunto de los agentes miembros del sistema, para que este funcione como un solo organismo eficaz, autónomo y descentralizado.

2.5. IMPLEMENTACIÓN DE INTERFAZ DE USUARIO

Para el presente trabajo de integración curricular se ha creado una interfaz de usuario utilizando el lenguaje Python dentro de ROS, de esta forma el usuario podrá elegir entre dos diferentes tipos de obstáculos para colocarlos en el entorno virtual, así como establecer las coordenadas de aparición de dichos objetos en los recuadros destinados para el efecto dentro de la interfaz.

Además, se utiliza el paquete TKinter que se utiliza para crear interfaces basándose en el kit de herramientas GIU Tk y de widgets jerarquizados para la programación de dichas interfaces gráficas [25]. En la Tabla 2.4 se puede observar la ejemplificación y parametrización de los widgets mencionados. Cabe mencionar que para la programación de la estructura de la interfaz de usuario se utilizaron empaquetados de información, mismos que permiten la distribución geométrica en forma matricial para el ingreso de los widgets siempre que se especifiquen las filas y las columnas para los mismos.

Tabla 2.4. Widgets utilizados en la creación de la interfaz de usuario. [Fuente propia]

Widget	Detalle	Ejemplo
Tk	Raíz de la interfaz en el cual se ingresarán los widgets	Raíz=tk()
Title	Titulo para la raíz de la interfaz	raiz.title("Interfaz de usuario")
Pack()	Empaquetado de widgets para ir ubicándolos	Principal.pack()
text	Ingreso de texto	Text=”EPN”
fb	Color del texto	Fg=”blue”
font	Formato de letra y tamaño	font=("Arial", 30))

Label	Etiqueta para mostrar texto e imágenes	Label(principal, text="Escuela Politécnica Nacional", fg="blue", font=("Arial", 30))
PhotoImage	Ingreso de imágenes	PhotoImage(“directorio del archivo a cargar”)
command	Nombre de la función a llamar para ejecución de acciones en los botones	command=pares
Button	Ingreso de botones	Button(principal, text="Primer objeto", command=pares, font=("Arial", 20))
textvariable	Variables para ingreso de datos	textvariable=valor1y
Entry	Ingreso de textos para indicar al usuario que debe ingresar datos	Entry(principal, textvariable=valor1y,font=("Arial", 17))
Set	Cambio de textos en comentarios según sea el caso	comentario.set("Objeto ingresado correctamente")
Gird	Posicionamiento matricial del widget según la fila y columna establecida	comentario.grid(row=4, column=1)

La interfaz de usuario está diseñada y estructurada en tres programas principales que tienen la extensión .py, estos se utilizan para generar los códigos y estructuras respectivos, mismos que se detallan a continuación:

Ventana.py: Contiene a los widgets de imágenes, botones, texto, empaquetamiento y posicionamiento de la información principal y del código de llamada para abrir una ventana de la interfaz, que corresponde a uno de los dos obstáculos que el usuario puede ingresar en el entorno virtual, estos se detallan a continuación.

Pares: Contiene a los widgets de imágenes, botones, texto, empaquetamiento y posicionamiento de la información principal y del código de lectura URDF que corresponde a una esfera de color blanco, la cual es uno de los obstáculos que el usuario puede introducir en la interfaz.

Impares: Contiene a los widgets de imágenes, botones, texto, empaquetamiento, posicionamiento de la información principal y del código de lectura URDF que corresponde

a un cilindro de color rojo, el cual es el otro tipo de obstáculo que el usuario puede introducir en la interfaz.

Entonces, el proceso consiste en inicializar a la interfaz de usuario al ejecutar el archivo denominado *ventana.py* en el terminal; una vez que la interfaz se visualiza en la pantalla, el usuario podrá seleccionar una de las dos opciones de obstáculos que se pueden ingresar en el entorno virtual, la esfera blanca o el cilindro rojo, al utilizar el botón que corresponde a cada objeto. La ventana mencionada se puede observar en la Figura 2.10.

Figura 2.10. Ventana principal de la interfaz de usuario. [Fuente propia]

Tras haber seleccionado el obstáculo de preferencia del usuario se genera una nueva ventana que contiene información única sobre la programación y estructura de cada obstáculo; en general, las estructuras de las ventanas de los dos obstáculos son similares entre sí, con diferencia del código de programación cdl botón de ingreso del obstáculo para cada caso. Además, en esta nueva ventana se presentan dos recuadros en los que se solicita al usuario que ingrese las coordenadas x e *y* del entorno en las que desea que aparezca el objeto seleccionado; los valores que el usuario ingresa deben ser números enteros que se encuentren contenidos en el rango de valores *[-8, 8]* inclusive que son las coordenadas correspondientes a los límites en los ejes *x* e *y* del entorno virtual, todo esto se indica al usuario en forma de texto dentro de la ventana. El obstáculo seleccionado ingresará en el entorno virtual cuando el usuario seleccione el botón creado para dicho fin. La ventana especificada se muestra en la Figura 2.11.

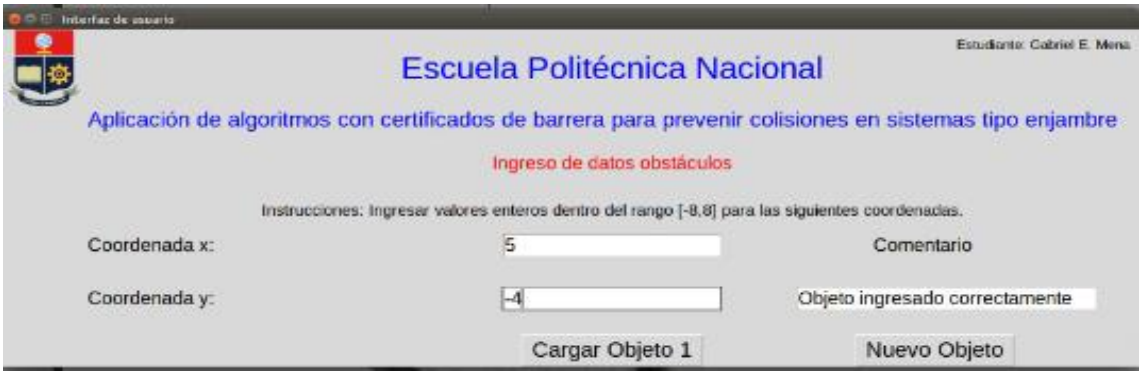


Figura 2.11. Ventana de ingreso de coordenadas para la posición de los obstáculos en el entorno virtual. [Fuente propia]

Dependiendo de las coordenadas ingresadas por el usuario se generan tres posibles mensajes en un cuadro de comentarios para que el mismo los lea, dichos casos y mensajes en respuesta se detallan a continuación:

1. **Coordenada incorrecta:** Este mensaje se muestra en la interfaz cuando el usuario ha ingresado valores que se encuentran fuera del rango establecido para las coordenadas; si esto sucede no se ingresará ningún obstáculo en el entorno virtual y el usuario tendrá que ingresar nuevas coordenadas según las especificaciones que se indican en la ventana de ingreso de objetos, que fue detallada anteriormente.

2. **Objeto ingresado correctamente:** Este mensaje se muestra en la interfaz cuando el usuario ha ingresado valores que se encuentran dentro del rango establecido para las coordenadas; si esto sucede se generará automáticamente el objeto seleccionado, el cual ingresará al mundo virtual en las coordenadas establecidas por el usuario.

3. **Crear nuevo objeto:** Este mensaje se muestra en la interfaz una vez que el obstáculo anterior ha ingresado en el entorno virtual de forma adecuada y en las coordenadas especificadas por el usuario, permitiendo así que el usuario pueda volver a seleccionar un nuevo obstáculo y especificar sus coordenadas de ingreso en el entorno. De este modo, el usuario podrá seguir ingresando la cantidad de objetos que desee en diferentes posiciones del entorno.

Cabe mencionar que para el ingreso de los nuevos obstáculos en el entorno virtual y su compatibilidad con Gazebo se generan nuevos nodos con nombres únicos para cada objeto ingresado; además, para cada uno se establecen todas las características tanto físicas como geométricas utilizando el lenguaje URDF, de esta forma es posible asegurar que los nodos no se suscriban entre sí ni que se eliminen objetos creados anteriormente.

La implementación de la interfaz de usuario en el entorno virtual es apropiada, pues permite a los usuarios interactuar con el entorno sin interferir ni con las funciones de cada robot ni con el funcionamiento general del sistema tipo enjambre; pero, al mismo tiempo, contribuye a demostrar la eficiencia del funcionamiento del sistema, incluso en un entorno cambiante, como el que podría existir en un caso real. De esta forma, se demuestra que los robots son capaces de evadir no solo objetos fijos preestablecidos en el entorno virtual, sino también objetos nuevos e inesperados.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En este capítulo se detallan las pruebas realizadas para la evaluación del correcto funcionamiento del sistema tipo enjambre, de los algoritmos implementados en los robots y de la interfaz de usuario creados para el presente trabajo de integración curricular y se describen los resultados obtenidos en las mismas. Para los primeros cuatro apartados de las mencionadas pruebas se implementa un entorno virtual conformado únicamente por paredes alrededor del escenario a manera de límites del entorno, a este tipo de escenario se lo denomina *entorno virtual básico*. Para las siguientes se añaden nuevos obstáculos, paredes u otros elementos y se cambian ciertas condiciones del escenario según sea necesario. Además, en el último apartado se realiza un análisis de las pruebas anteriores en diferentes escenarios.

3.1. PRUEBAS Y RESULTADOS

3.1.1. PRUEBAS DE INTERFAZ DE USUARIO

Inicialmente se realizó el análisis del correcto funcionamiento de la interfaz de usuario dentro del *entorno virtual básico* mencionado anteriormente; se decidió trabajar en este entorno debido a que, en el mismo, es posible examinar si los obstáculos seleccionados pueden ingresar en el entorno virtual en cualquier momento y posición que el usuario especifique, así como analizar si se puede ingresar una gran cantidad de objetos sin que la interfaz deje de cumplir con su función. Se evaluó además el ingreso de los objetos no sólo al inicio de la simulación, sino también en cualquier momento mientras la simulación se está ejecutando.

Se realizaron diez pruebas de ingreso de los dos tipos de obstáculos permitidos por la interfaz, los cilindros rojos o las esferas blancas, se establecieron de forma aleatoria diferentes posiciones de ingreso de los obstáculos en el entorno virtual y se obtuvieron los resultados presentados en la Tabla 3.1.

Tabla 3.1. Pruebas y resultados de la interfaz de usuario [Fuente propia]

N°	Observación	N° de prueba	Observación
1	Exitoso	**6**	Exitoso
2	Exitoso	**7**	Sobre posición
3	Exitoso	**8**	Exitoso
4	Exitoso	**9**	Movimiento
5	Exitoso	**10**	Exitoso

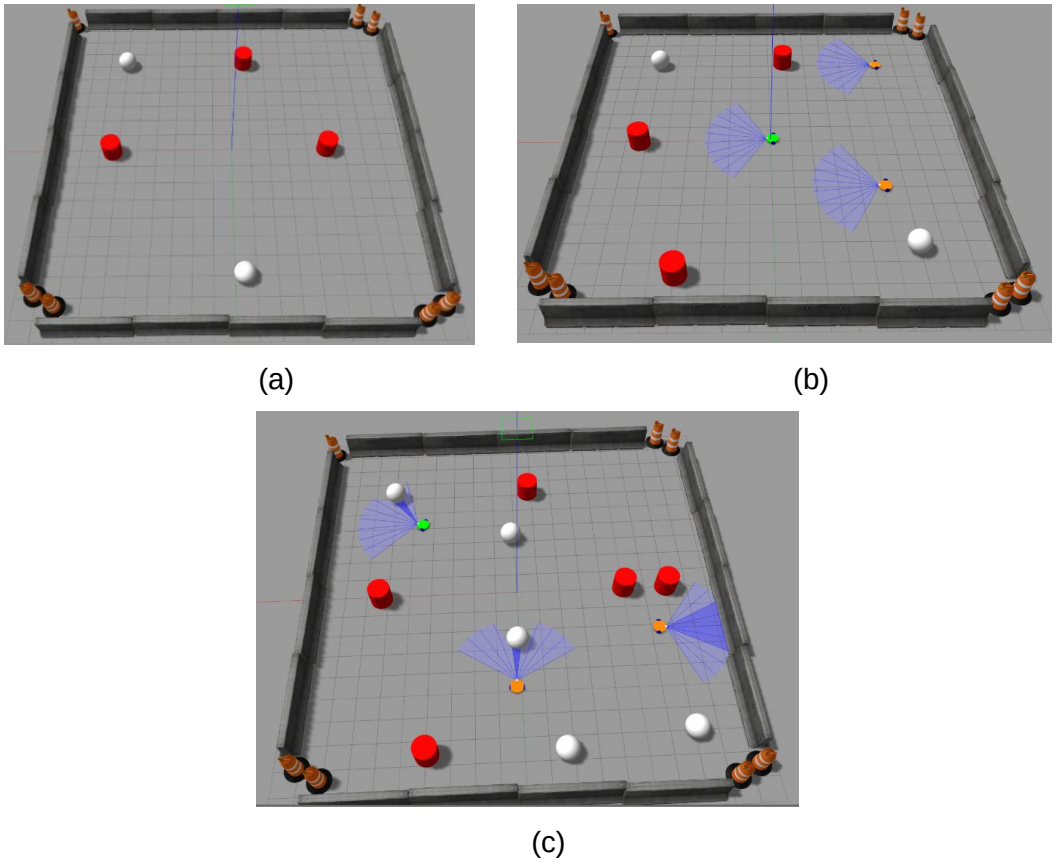
Como se observa, en la séptima prueba se obtuvo como resultado una *sobre posición* de obstáculos, esto se da debido a que no se respetaron las coordenadas físicas de los objetos que fueron previamente ingresados en el entorno virtual y se seleccionaron exactamente sus mismas coordenadas, lo que ocasionó no sólo que el segundo objeto sea creado en la misma posición del otro, sino que se produjo una interacción dinámica en la que los objetos se movían empujando al otro objeto al intentar colocarse cada uno en las coordenadas especificadas que, en este caso, coinciden. Esto se puede solucionar indicando a los usuarios que, en lo posible, eviten colocar dos objetos exactamente en las mismas coordenadas del entorno virtual.

Por su parte, en la novena prueba se obtuvo como resultado un *movimiento* ocasionado por la interacción entre la esfera blanca con el borde del entorno virtual que posee una pequeña rampa, esto ocasiona una interacción dinámica pues los obstáculos colisionan, además, la esfera mueve o empuja al resto de obstáculos previamente ingresados cuando choca con ellos al caer del borde del entorno. Esto se puede solucionar indicando al usuario que no coloque los obstáculos exactamente en las mismas coordenadas de las paredes del contorno del entorno, así como recomendarle que no coloque obstáculos fuera de las coordenadas de trabajo establecidas.

3.1.2. PRUEBAS DEL ALGORITMO EVASOR DE OBSTÁCULOS

Se ejecutaron pruebas para evaluar la efectividad del algoritmo evasor de obstáculos en el *entorno virtual básico* utilizado en la Sección 3.1.1, pero se añadieron en el mismo cinco obstáculos fijos; es decir, con una posición predeterminada antes del ingreso de los robots en el entorno, y, posteriormente, mientras la simulación ya se estaba ejecutando y los robots se encontraban evadiendo correctamente los obstáculos fijos, se ingresaron cinco obstáculos nuevos, variando entre los dos tipos de obstáculos, en posiciones aleatorias del entorno.

Para la evaluación de este algoritmo se colocaron a los tres robots en el entorno: el robot explorador inició su recorrido desde las coordenadas [0, 0] y dos robots seguidores que iniciaron sus recorridos en las coordenadas [-5, 3] y [-6, -6] respectivamente. Los valores de censado de esta prueba se obtuvieron con el sensor láser de proximidad, el cual recopila información sobre la distancia de los robots con los obstáculos y permitió al algoritmo cumplir con su función de forma efectiva, evadiendo así a todos los obstáculos para todos los robots. La Figura 3.1 permite visualizar una ejemplificación del desarrollo de la prueba para este apartado.



(a) (b)

(c)

Figura 3.1. Prueba del algoritmo evasor de obstáculos: a) Diseño inicial del mundo con obstáculos, b) Mundo con obstáculos y robots, c) Funcionamiento del algoritmo evasor con diez obstáculos. [Fuente propia]

Las siguientes subsecciones detallan los resultados obtenidos en las pruebas del algoritmo de evasión de obstáculos y de evasión de otros robots.

3.1.2.1. Evasión de Obstáculos

Esta prueba se realizó cumpliendo con los criterios establecidos en la sección al inicio de esta sección y se repitió veinte veces cambiando las posiciones tanto de los obstáculos fijos como de los ingresados durante la simulación e ingresando a estos últimos en diferentes momentos de la simulación cada vez. La duración de cada simulación para el

análisis fue de cinco minutos. Los resultados obtenidos en esta simulación se presentan en la Tabla 3.2.

Tabla 3.2. Resultados de las pruebas de evasión de obstáculos. [Fuente propia]

N° de prueba	Observación	N° de prueba	Observación
1	Exitoso	**11**	Colisión con cilindro
2	Exitoso	**12**	Exitoso
3	Exitoso	**13**	Exitoso
4	Exitoso	**14**	Exitoso
5	Exitoso	**15**	Exitoso
6	Exitoso	**16**	Colisión con esfera
7	Colisión con esfera	**17**	Exitoso
8	Exitoso	**18**	Exitoso
9	Exitoso	**19**	Exitoso
10	Exitoso	**20**	Exitoso

Como se observa, los resultados obtenidos en la séptima y décimo primer pruebas tuvieron como consecuencia la colisión del *robot seguidor 1* con el obstáculo en forma de esfera y del *robot explorador* con el obstáculo en forma de cilindro. Esto sucedió debido a que el ingreso de las coordenadas de uno de los obstáculos colocado mientras la simulación se estaba ejecutando fue incorrecto, pues, en ambos casos, no se respetó ni la geometría ni la posición de los robots en el entorno en el momento del ingreso del obstáculo, por lo que los obstáculos ingresados aparecieron en la misma posición en la que se encontraban los robots en el mismo instante de tiempo, colisionando con ellos por este motivo. Esto se puede solucionar indicando a los usuarios que, en lo posible, eviten colocar obstáculos exactamente en las mismas coordenadas en las que se encuentra el robot en un momento determinado dentro del entorno virtual.

Asimismo, el *robot seguidor 1* colisionó con el obstáculo en forma de esfera en la décimo sexta prueba, esto se dio debido a las propiedades físicas del obstáculo pues, al ser una esfera, cuando la misma ingresó en el entorno virtual, mientras la simulación se encontraba ejecutándose, esta comenzó a girar y moverse sin control por el entorno, acercándose rápidamente al robot. Cabe destacar que, incluso en este caso, el sensor láser de proximidad detectó a la esfera y el algoritmo funcionó, pues el robot cambió su dirección para evitar el objeto; sin embargo, la esfera avanzaba a una velocidad mayor que el robot y lo impactó desde el costado. Esto se puede solucionar recomendando al usuario ingresar correctamente las coordenadas del obstáculo para evitar que los objetos se superpongan

o choquen iniciando el movimiento descontrolado, en ciertos casos, de los obstáculos en forma de esfera.

3.1.2.2. Evasión de Otros Robots

Para el análisis del funcionamiento del algoritmo de evasión de colisiones entre robots, se utilizó nuevamente el entorno virtual básico que no posee ningún obstáculo preestablecido ni ingresado mientras se ejecuta la simulación y se colocó a los tres robots en el entorno, colocándolos en diferentes coordenadas de inicio para cada nueva prueba, pero respetando la geometría y las posiciones físicas de los otros robots. Para algunas de las pruebas se dejó estático a uno de los robots y se permitió que los otros dos se movilicen por el entorno para evaluar si el algoritmo detecta y evade no sólo a otros robots en movimiento sino también a robots fijos dentro del entorno.

Se realizaron diez pruebas simulando lo mencionado por tres minutos para cada prueba y, en dos de las pruebas se forzó la posible colisión de dos de los robots al cambiar las posiciones de los mismos. En ambos casos el algoritmo detectó al otro robot e intentó evadir la colisión, por lo que todos los resultados fueron exitosos, como se muestra en la Tabla 3.3.

Tabla 3.3. Resultados de las pruebas de evasión de otros robots. [Fuente propia]

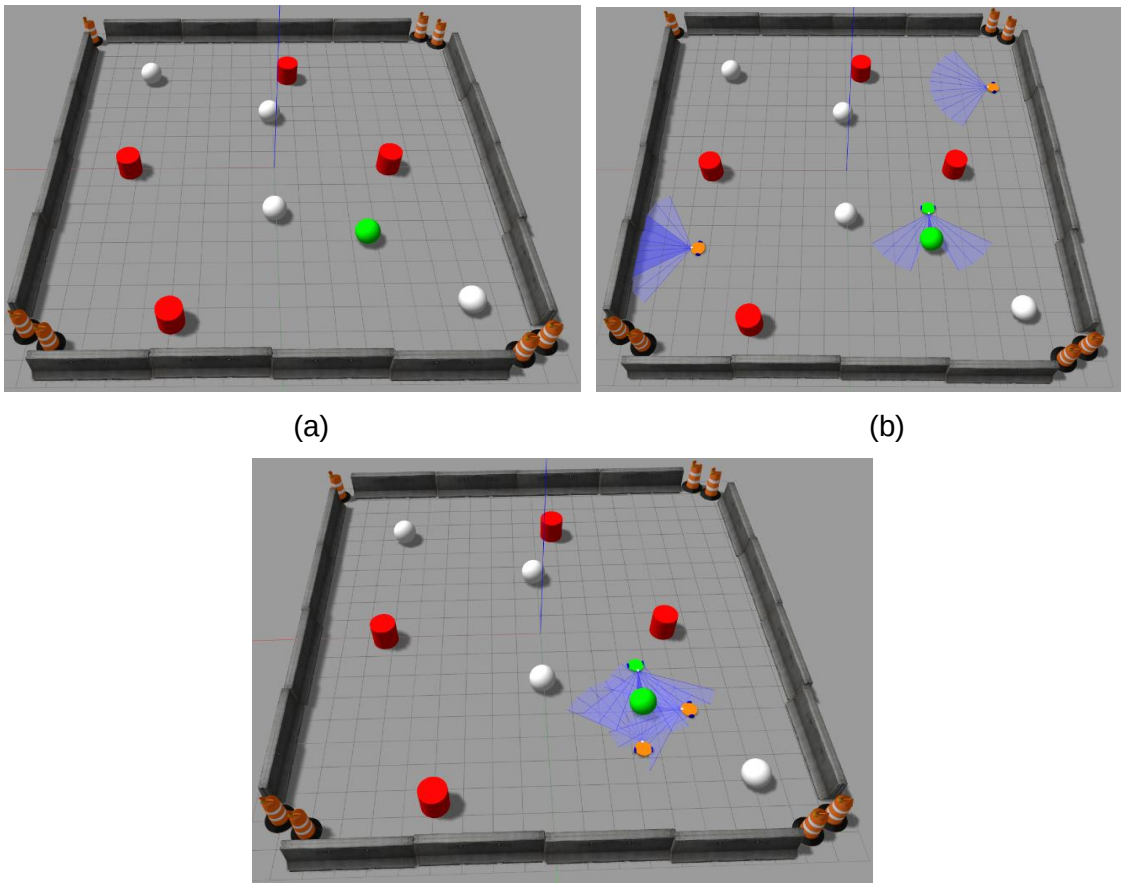
N° de prueba	Observación	N° de prueba	Observación
1	Exitoso	**6**	Exitoso
2	Exitoso	**7**	Exitoso
3	Exitoso	**8**	Exitoso
4	Exitoso	**9**	Exitoso
5	Exitoso	**10**	Exitoso

3.1.3. PRUEBAS DEL ALGORITMO DE CONSENSO

Para poder evaluar la efectividad de este algoritmo primero es necesario evaluar la efectividad del algoritmo detector de colores del sensor cámara que posee únicamente el robot explorador para, a la vez, evaluar si la comunicación entre los robots y la lectura en tiempo real de las coordenadas son apropiadas.

Para la ejecución de las pruebas se ingresaron ocho obstáculos en el entorno virtual, colocándolos en diferentes coordenadas de posición en cada caso, como se sabe, dichos

obstáculos son de color blanco en el caso de la esfera y de color rojo en el caso del cilindro; además, se colocó en diferentes coordenadas de posición del entorno al objetivo, que es una esfera de color verde, para cada prueba. Asimismo, se colocó a los dos robots seguidores y al robot explorador en el entorno, para determinar si el robot explorador detecta el color verde del objetivo y envía las coordenadas de posición del mismo a los seguidores. La Figura 3.2 permite visualizar una ejemplificación del desarrollo de la prueba para este apartado.



(a) (b)

(c)

Figura 3.2. Prueba del algoritmo de consenso: a) Diseño inicial del mundo con obstáculos, b) Detección del objetivo por parte del robot explorador, c) Alcance del objetivo por parte de todos los robots. [Fuente propia]

3.1.3.1. Algoritmo de Detección de Colores

El objetivo final de estas pruebas es la correcta detección del color verde de la esfera objetivo por parte del robot explorador, esto es posible gracias al sensor cámara que dicho robot posee y a su rango de acción. Se realizaron quince pruebas con diferentes condiciones y posiciones de los objetos dentro del entorno, los resultados obtenidos para el algoritmo de detección de color se muestran en la Tabla 3.4.

Tabla 3.4. Resultados de las pruebas del algoritmo de detección de colores. [Fuente propia]

Prueba	Objeto Verde	Objetos Rojo	Objeto Blanco	Observación	Resultado
1	Sí	Sí	Sí	Detección Verde	Exitoso
2	Sí	No	No	Detección Verde	Exitoso
3	No	Sí	Sí	No detección verde	Exitoso
4	Sí	Sí	No	No detección verde	Fallido
5	Sí	No	Sí	Detección Verde	Exitoso
6	No	Sí	No	No detección verde	Exitoso
7	Sí	Sí	Sí	Detección Verde	Exitoso
8	No	Sí	No	No detección verde	Exitoso
9	Sí	Sí	Sí	No detección verde	Exitoso
10	No	No	Sí	No detección verde	Fallido
11	Sí	Sí	No	Detección Verde	Exitoso
12	Sí	Sí	Sí	Detección Verde	Exitoso
13	Sí	No	Sí	Detección Verde	Exitoso
14	No	No	Sí	No detección verde	Exitoso
15	No	Sí	Sí	No detección verde	Exitoso

Tabla 3.5. Porcentajes de error – Detección de colores. [Fuente propia]

Resultados	Cantidad de pruebas realizadas	Porcentaje
Exitosos	13	86.7 %
Fallidos	2	13.3 %
Total	**15**	**100 %**

Como se observa en la Tabla 3.4, la cuarta prueba presenta problemas con la detección del color verde, esto sucedió debido a que únicamente un tercio de la esfera de color verde estaba ubicado dentro del rango de detección del sensor cámara. Del mismo modo, en la décima prueba se presentan problemas causados por la posición de los objetos en el entorno virtual, ya que existen obstáculos que cubren a la esfera verde objetivo y no permiten que el robot explorador la detecte ni se acerque a la misma, pues se produce una *máscara de imagen* que imposibilita el reconocimiento. La Tabla 3.5 muestra que el porcentaje de error es de 13.3 %, lo cual se evitar en las condiciones iniciales del entorno,

colocando adecuadamente el objetivo y los obstáculos en coordenadas factibles para el cumplimiento del propósito.

3.1.3.2. Pruebas de Intercomunicación y Lectura de Coordenadas

Tras haber comprobado el correcto funcionamiento del algoritmo de colores se realizaron pruebas de intercomunicación entre los tres robots y de lectura de coordenadas sobre la posición de cada robot durante la simulación. Para ello, se trabajó con los dos robots obreros y el robot explorador, de modo que fue posible evaluar la efectividad de todo el sistema y se ingresaron en el entorno virtual la esfera objetivo de color verde, seis obstáculos fijos con coordenadas de ingreso al entorno previamente establecidas y diferentes cantidades de obstáculos ingresados, con el uso de la interfaz de usuario, en diferentes momentos mientras se ejecutaba la simulación.

En todos los casos el algoritmo de detección de colores se encontraba activo y funcional. Se busca determinar si existe consenso en el funcionamiento del sistema tipo enjambre; para que esto suceda, los robots tendrán que leer constantemente las coordenadas cambiantes de su ubicación en el entorno y transmitirlas a los otros dos robots, junto con la información sobre su estado *en movimiento* o *en reposo,* de forma ininterrumpida, con el uso del algoritmo de consenso. Además, una vez que el robot explorador encuentra el objetivo, este envía constantemente las coordenadas del mismo a los robots obreros, quienes cambiarán su estado de *en reposo* a *en movimiento* y comenzarán su recorrido hacia las coordenadas dadas por el agente explorador; mientras tanto, los robots obreros envían sus coordenadas entre sí, con el fin de conocer qué robot llegó primero al objetivo, dado que cada robot parte de una posición diferente en el entorno.

Se realizaron quince pruebas con una duración de treinta minutos cada una, cambiando las posiciones iniciales de los robots en cada caso, así como las del objetivo y los obstáculos tanto los preestablecidos como los ingresados mediante la interfaz de usuario. Los resultados obtenidos se muestran en la Tabla 3.6.

Tabla 3.6. Resultados de las pruebas de intercomunicación y lectura de coordenadas. [Fuente propia]

Prueba	Robot explorador detección	Robots obreros detección	Comunicación	Cantidad de obstáculos	Completar la tarea	Resultado
1	Sí	Sí	Exitosa	0	Exitoso	Exitoso
2	Sí	Sí	Exitosa	3	Exitoso	Exitoso

3	Sí	Sí	Exitosa	4	Exitoso	Exitoso
4	Sí	Sí	Exitosa	5	Exitoso	Exitoso
5	No	No	Exitosa	7	Fallido	Fallido
6	Sí	Sí	Exitosa	0	Exitoso	Exitoso
7	Sí	Sí	Exitosa	8	Exitoso	Exitoso
8	Sí	Sí	Exitosa	7	Exitoso	Exitoso
9	Sí	Sí	Exitosa	1	Exitoso	Exitoso
10	Sí	No	Exitosa	6	Fallido	Fallido
11	Sí	Sí	Exitosa	4	Exitoso	Exitoso
12	Sí	Sí	Exitosa	6	Exitoso	Exitoso
13	Sí	Sí	Exitosa	4	Exitoso	Exitoso
14	Sí	No	Exitosa	9	Fallido	Fallido
15	No	No	Exitosa	5	Fallido	Fallido

Tabla 3.7. Porcentajes de error – Intercomunicación y lectura de coordenadas. [Fuente propia]

Resultados	Cantidad de pruebas realizadas	Porcentaje
Exitosos	11	73.3 %
Fallidos	4	26.7 %
Total	**15**	**100 %**

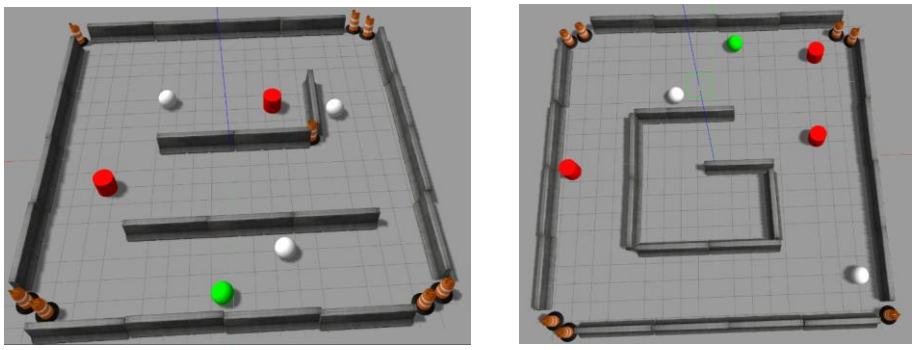
Como muestra la Tabla 3.6, los resultados obtenidos para la quinta y la décimo quinta pruebas fueron fallidos tanto en completar la tarea como en el cumplimiento final del objetivo del análisis, que es encontrar la esfera verde. Esto se produjo debido a que, para empezar, el robot explorador no detectó al objetivo dentro del tiempo destinado para la prueba realizada, por ende, este no transmitió la información de las coordenadas del objetivo a los robots seguidores, quienes no se dirigieron a la ubicación deseada y no se pudo cumplir con la tarea asignada.

Del mismo modo, en la décima y décimo cuarta pruebas el resultado fue fallido; en este caso la problemática sucedió debido a que los robots seguidores no pudieron llegar a las coordenadas de posición del objetivo dentro del tiempo establecido para cada prueba, evitando así cumplir con la tarea asignada. La tabla 3.7 muestra que el porcentaje de error para esta prueba fue de 26.7 %; se puede evitar dichas fallas al dar al sistema más tiempo

de ejecución de las actividades, de este modo se comprobaría si realmente la prueba es fallida o si el tiempo limitado influye en el cumplimiento de los objetivos.

3.1.4. PRUEBAS EN DIFERENTES ESCENARIOS

Para terminar, se realizaron pruebas del funcionamiento global de todo el sistema tipo enjambre de robots dentro de diferentes escenarios para el entorno virtual, de modo que se pruebe la versatilidad de aplicación del presente proyecto no sólo en diferentes escenarios, sino también en diferentes campos de acción. Para hacerlo, se trabajó con diferentes diseños del mundo virtual, mismos que se muestran en la Figura 3.3 y se detallan a continuación.



(a) (b)

Figura 3.3. Mundos virtuales: a) Mundo virtual paralelo, b) Mundo virtual con forma de G. [Fuente propia]

Mundo virtual paralelo: Este escenario cuenta con paredes en todo su contorno con el fin de delimitar a las dimensiones del mismo; además, este escenario posee dos paredes, que son paralelas entre sí, en el centro del escenario, las cuales tienen el rol que cumplen los obstáculos fijos preestablecidos. Además, se ingresan dos cilindros rojos y tres esferas blancas en el escenario como condición inicial, así como la esfera objetivo de color verde claro, todos ellos ubicados en diferentes coordenadas dentro del entorno para cada prueba realizada.

Mundo virtual con forma de *G*: En este escenario también se tienen paredes fijas en todo el contorno del mundo virtual con el fin de delimitar las dimensiones del mismo. En el centro del entorno se cuenta con ocho paredes fijas preestablecidas que forman una letra *G* grande y cumple el rol de obstáculo para los robots. Además, se ingresan tres cilindros rojos y dos esferas blancas como condición inicial de la simulación, así como la esfera objetivo de color verde claro, todos ellos ubicados en diferentes coordenadas dentro del entorno para cada prueba realizada.

Para todas las pruebas se trabajó con los dos robots obreros y el robot explorador y con todos los sensores y algoritmos activos y funcionales. En todos los casos se midió el tiempo que tardan los robots en completar la tarea asignada y se comprobó si la comunicación entre los robots fue adecuada e ininterrumpida, así como el correcto funcionamiento de la interfaz de usuario y del ingreso de nuevos obstáculos en el escenario mientras la simulación se ejecuta. Se ingresaron nuevos obstáculos en el entorno virtual cada dos minutos, lo cual permite que la estructura de los escenarios cambie cada vez. Se realizaron cinco pruebas en cada nuevo mundo virtual, cada una de las cuales tuvo una duración de cuarenta y cinco minutos. Los resultados obtenidos por dichas pruebas se muestran en la Tabla 3.8.

Tabla 3.8. Resultados de las pruebas en diferentes escenarios. [Fuente propia]

N° Prueba	Mundo virtual	Tiempo de simulación	Comunicación	Resultado
1	Paralelo	14	Efectiva	Exitoso
2	Paralelo	16	Efectiva	Exitoso
3	Paralelo	37	Efectiva	Fallido
4	Paralelo	28	Efectiva	Exitoso
5	Paralelo	45	Efectiva	Fallido
6	En forma G	7	Efectiva	Exitoso
7	En forma G	44	Efectiva	Exitoso
8	En forma G	38	Fallida	Fallido
9	En forma G	18	Efectiva	Exitoso
10	En forma G	45	Efectiva	Fallido

Tabla 3.9. Porcentajes de error – Escenario en paralelo. [Fuente propia]

Resultados	Cantidad de pruebas realizadas	Porcentaje
En paralelo		
Exitosos	3	60 %
Fallidos	2	40 %
Total	**5**	**100 %**

Tabla 3.10. Porcentajes de error – Escenario en forma de G. [Fuente propia]

Resultados	Cantidad de pruebas realizadas	Porcentaje
En paralelo		
Exitosos	3	60 %
Fallidos	2	40 %
Total	**5**	**100 %**

Como se observa en la Tabla 3.8, el resultado fue *fallido* para la tercera, quinta, octava y décima prueba realizadas en diferentes escenarios, esto se produjo debido a un exceso en la cantidad de obstáculos ingresados en el entorno virtual, motivo por el cual se limitó el movimiento de los robots en el entorno, creando bucles de decisión de movimiento ya que los agentes quedan atrapados entre los obstáculos y no les es posible cumplir con la tarea asignada; es decir, no se pudo localizar la esfera objetivo dado el movimiento limitado de los robots. Esto demuestra que, mientras más copado de obstáculos se encuentre el entorno, la probabilidad de que los robots cumplan con la tarea asignada disminuye.

Las Tablas 3.9 y 3.10 muestran que, en cada caso, el porcentaje de error es de 40 %; para solucionar esta problemática, se puede recomendar al usuario no copar al entorno virtual de obstáculos para permitir a los robots realizar su trabajo y al alcanzar el objetivo esperado; si bien la interfaz no limita la cantidad de objetos a ingresar, el usuario puede ser consciente de que un entorno sobrecargado de obstáculos no es el ideal para el correcto desempeño del sistema tipo enjambre.

Además, la octava prueba presenta un resultado f*allido* para la comunicación entre los agentes, esto se produjo, principalmente, debido al rendimiento del computador en el que ejecutó la simulación, así como la saturación del mismo, ya que se trabaja directamente con el terminal del sistema y esto puede llegar a saturar al equipo en ciertas ocasiones. Para evitar estas complicaciones se sugiere cerrar otros programas o ventanas irrelevantes para el proyecto que se tengan abiertos durante la ejecución de las simulaciones con el fin de no saturar al equipo, o bien, dar un tiempo de reposo al equipo cada cierto número de pruebas para no afectar a su rendimiento.

3.2. CONCLUSIONES

Tras haber analizado los resultados obtenidos en las pruebas realizadas se concluye que el uso de algoritmos con certificados de barrera en sistemas tipo enjambre es adecuado ya

que se establece un área de trabajo para cada agente robótico, evitando así colisiones y daños en los mismos.

Además, el uso y combinación de lenguajes de programación dentro del software ROS permite que la simulación de los agentes robóticos sea completa en cuanto al uso de diferentes sensores como de proximidad, tipo cámara, así como la interacción de dichos agentes en un entorno dinámico.

También se concluye que, en la robótica, la implementación y utilización de sensores láser de proximidad es importante debido a que este permite que los robots detecten objetos, obstáculos u otros robots cercanos, de modo que puedan re direccionar su curso para evadir la colisión con dichos objetos gracias a algoritmos que se basan en la información proporcionada por dicho sensor.

La aplicación de algoritmos de consenso permite que los robots realicen tareas complejas o estructuradas, dividiéndolas en tareas simples que se asignan por separado a cada uno de los agentes, como se ha demostrado en este trabajo, cumpliendo con los objetivos esperados y permitiendo que exista una comunicación constante e ininterrumpida entre los agentes robóticos miembros del sistema.

En el entorno de ROS – Gazebo, es posible obtener información a partir de la intercomunicación entre los agentes, esto es favorable para cualquier proyecto de robótica dado que es posible realizar cálculos importantes referentes a la distancia entre agentes, estados de los mismos, visualización del entorno, entre otros.

Las diferentes pruebas se realizaron con el fin de evaluar si el funcionamiento del sistema tipo enjambre, tanto en su totalidad como de cada uno de sus componentes, es adecuado, se concluye que se ha cumplido el objetivo general planteado para el presente trabajo de integración curricular. El sistema de robots tipo enjambre funciona de forma eficiente ya que las actividades individuales de cada agente contribuyen para el alcance de los objetivos y para que el sistema funcione como tal. Además, los robots diferenciales evitan las colisiones con obstáculos u otros robots en todos los casos, gracias a los algoritmos con certificados de barrera que fueron implementados en los mismos.

Respecto a la interfaz gráfica, se destaca la importancia de que la misma contenga instrucciones claras y secuenciales sobre los pasos que el usuario debe seguir para ingresar un obstáculo en el entorno a medida que se ejecuta la simulación.

Por último, se concluye que los lenguajes de programación apropiados para el desarrollo de los diferentes componentes del entorno virtual son el lenguaje Python, que sirve para desarrollar la interfaz gráfica, y el lenguaje URDF, que se ha utilizado para la estructuración y parametrización de los robots, lo que contribuye al correcto funcionamiento del sistema como conjunto.

3.3. RECOMENDACIONES

El presente trabajo de integración curricular es una base para futuros alcances de investigación y desarrollo de sistemas de robots tipo enjambre más complejos. Para lograrlo, se recomienda añadir algoritmos de mapeo que sirvan no solo para identificar los objetos del entorno, sino también registrar dicha información, de modo que se obtenga un mapa detallado del entorno virtual, lo cual ofrecerá una serie de oportunidades de aplicación para este tipo de sistemas.

Se recomienda utilizar el sistema tipo enjambres diseñado en este trabajo como base para la creación y desarrollo de redes neuronales, así como de otros métodos de Inteligencia Artificial (AI).

Si se desea replicar este trabajo de integración curricular, se recomienda desarrollar los mundos virtuales utilizando *building editor,* pues el uso de esta herramienta resultó ser sencillo y permitió que el diseño de los entornos sea versátil, ofreciendo infinitas posibilidades de diseño.

Se recomienda utilizar el lenguaje de programación *Python* para el desarrollo de la interfaz de usuario. Del mismo modo, se recomienda utilizar el lenguaje de programación URDF, estructurado dentro de archivos .xacro para la inicialización de nodos y tópicos y para que la interrelación entre los mencionados archivos sea adecuada, todo esto se lo debe realizar dentro de ROS-Gazebo, para lo cual es importante en ROS Gazebo siempre estructurar la inicialización de nodos, tópicos y una correcta interrelación entre los archivos estructurados.

Por último, se recomienda realizar los procedimientos especificados en el presente trabajo de integración curricular de forma sistemática y secuencial para el diseño de los componentes del entorno virtual, de los robots y la correcta implementación de los algoritmos con certificados de barrera para la prevención de colisiones en el sistema.

4. REFERENCIAS BIBLIOGRÁFICAS

[1] M. Rubio, «Desarrollo de comportamientos de enjambre mediante computación evolutiva», 2021.

[2] Y. Mohan y S. G. Ponnambalam, «An extensive review of research in swarm robotics», *2009 World Congr. Nat. Biol. Inspired Comput. NABIC 2009 - Proc.*, pp. 140-145, 2009, doi: 10.1109/NABIC.2009.5393617.

[3] Embention, «Funcionamiento y aplicaciones con UAVs en enjambre», 2019. https://www.embention.com/es/news/funcionamiento-y-aplicaciones-con-uavs-en-enjambre/ (accedido nov. 30, 2021).

[4] A. Aguión, «Inteligencia de enjambre e inteligencia artificial», 2021. https://www.fundacionaquae.org/la-inteligencia-enjambre-y-la-inteligencia-artificial/ (accedido nov. 30, 2021).

[5] R. Solis, «Enjambres de robots y sus aplicaciones en la exploración y comunicación», 2018, [En línea]. Disponible en: https://www.researchgate.net/publication/323302140_Enjambres_de_robots_y_sus_aplicaciones_en_la_exploracion_y_comunicacion.

[6] A. J. C. Sharkey y N. Sharkey, *The application of swarm intelligence to collective robots*. 2006.

[7] M. Luisa y S. Tortosa, «Agentes y enjambres artificiales : modelado y comportamientos para sistemas de enjambre robóticos Agentes y enjambres artificiales : modelado y comportamientos para sistemas de enjambre rob ´», p. 251, 2013.

[8] M. Sabadel, «Robots inspirados en la naturaleza», 2016. https://www.muyinteresante.es/innovacion/articulo/robots-inspirados-en-la-naturaleza-701456834640 (accedido abr. 13, 2022).

[9] J. León, «Simulación de enjambres de robots en labores de exploración para detección de posibles víctimas», p. 111, 2017.

[10] Springer, «Overview of consensus algorithms in cooperative control», *Commun. Control Eng.*, n.º 9781848000148, pp. 3-22, 2008, doi: 10.1007/978-1-84800-015-5_1.

[11] A. Giusti, J. Nagi, L. Gambardella, y G. A. Di Caro, «Cooperative sensing and recognition by a swarm of mobile robots», *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 551-558, 2012, doi: 10.1109/IROS.2012.6385982.

[12] A. Priolo, A. Gasparri, E. Montijano, y C. Sagues, «A decentralized algorithm for balancing a strongly connected weighted digraph», *Proc. Am. Control Conf.*, pp. 6547-6552, 2013, doi: 10.1109/ACC.2013.6580866.

[13] A. Saffioti, F. Pan, y H. Rodríguez, «Estrategia de prevención de colisiones en 3D para colaboración segura hombre-robot, usando el Kinect», *Rev. Iniciación Científica*, vol. 6, n.º 1, pp. 92-97, 2020, doi: 10.33412/rev-ric.v6.1.2619.

[14] TAMPS, «Instalar ROS». https://www.tamps.cinvestav.mx/~arodas/install.html (accedido abr. 11, 2022).

[15] GAZEBOSIM, «Gazebo». http://gazebosim.org/ (accedido abr. 11, 2022).

[16] C. Cuevas, «Vista de Ros-gazebo. una valiosa Herramienta de Vanguardia para el Desarrollo de la Robótica | Publicaciones e Investigación», 2017. https://hemeroteca.unad.edu.co/index.php/publicaciones-e-investigacion/article/view/1593/1940 (accedido abr. 11, 2022).

[17] Clearpath Robotics, «Simulating Warthog — Warthog Tutorials 0.0.2 documentation», 2015. https://www.clearpathrobotics.com/assets/guides/kinetic/warthog/WarthogSimulation.html (accedido jun. 17, 2022).

[18] Programador Click, «Aprendizaje URDF Qué es URDF y cómo entender un archivo URDF», 2022. https://programmerclick.com/article/29081682599/ (accedido jul. 20, 2022).

[19] Gazebo, «Gazebo : Tutorial : Gazebo plugins in ROS», 2022. https://classic.gazebosim.org/tutorials?tut=ros_gzplugins (accedido jun. 19, 2022).

[20] Oreilli, «Building a room in Gazebo - ROS Programming: Building Powerful Robots [Book]», 2022. https://www.oreilly.com/library/view/ros-programming-building/9781788627436/d3689231-b3d0-439f-8ffd-d414528ee5d2.xhtml (accedido jul. 11, 2022).

[21] B. Magyar, «diff_drive_controller - ROS Wiki», 2022. http://wiki.ros.org/diff_drive_controller (accedido jun. 17, 2022).

[22] Holbrook Tate, C. Rebecca, y H. Brian, «Los secretos de un hormiguero | Ask A Biologist», 2021. https://askabiologist.asu.edu/secretos-superorgamismo (accedido jun. 22, 2022).

[23] C. Jiménez, «Desarrollo de un sistema de control de un robot móvil», *Univ. Valladolid*, 2017, Accedido: ago. 13, 2022. [En línea]. Disponible en: https://uvadoc.uva.es/bitstream/handle/10324/25547/TFG-P-644.pdf;jsessionid=C8D606B77683E72A2B1AD3DE0098164F?sequence=1.

[24] HiSoUR, «Robótica de enjambre – HiSoUR Arte Cultura Historia», 2013. https://www.hisour.com/es/swarm-robotics-43163/ (accedido jun. 22, 2022).

[25] D. Salazar, «Interfaz grafica con Tkinter», 2012. https://github.com/elilluminado/Guia-Tkinter/blob/master/Interfaz grafica con Tkinter.wiki (accedido ago. 13, 2022).

5. ANEXOS

A continuación, se presentan los siguientes Anexos:

- Anexo I. Manual de usuario del sistema de robots tipo enjambre
- Anexo II. Parametrización de archivos xacro
- Anexo III. Diagrama de flujo del algoritmo de prevención de colisiones
- Anexo IV. Diagrama de flujo del algoritmo de consenso
- Anexo V. Links de los videos de pruebas y resultados referentes al presente trabajo de integración curricular.

ANEXO I

MANUAL DE USUARIO DEL SISTEMA DE ROBOTS TIPO ENJAMBRE

En este anexo se encuentra el manual de usuario, mismo que detalla el funcionamiento y configuración del sistema de robots tipo enjambres del presente trabajo de integración curricular.

El presente manual se refiere a la programación que fue utilizada para el presente trabajo de integración curricular.

EJECUCIÓN DE LA SIMULACIÓN

Una vez descargados y compilados todos los archivos en el sistema se procede a ejecutar los siguientes comandos en diferentes terminales según se indica a continuación:

1.- En un nuevo terminal del sistema se ejecuta el siguiente comando, el cual se abrirá el mundo virtual paralelo que fue detallado en la sección 3.1.4. y que se visualiza en la Figura I.1.

- *roslaunch my_worlds worldp1.launch*

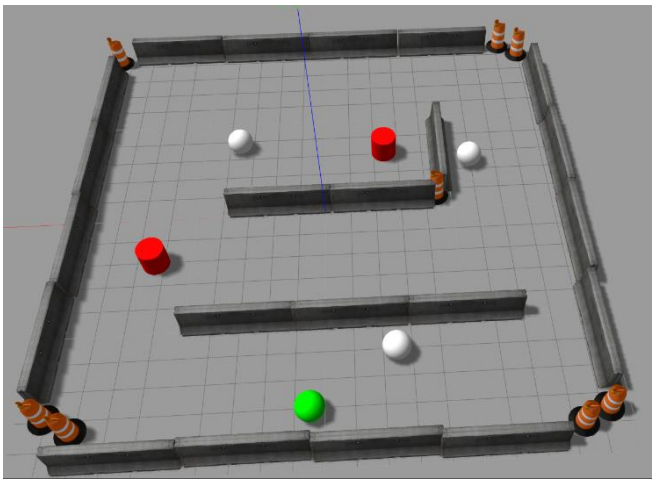


Figura I.1. Condiciones iniciales, mundo paralelo. [Fuente propia]

Posteriormente, se ingresarán los robots en un nuevo terminal con los siguientes comandos:

- *roslaunch robot_description spawn.launch*
- *roslaunch robot_description spawn2.launch*
- *roslaunch robot_description spawn3.launch*

El resultado del ingreso de los robots se observa en la Figura I.2.

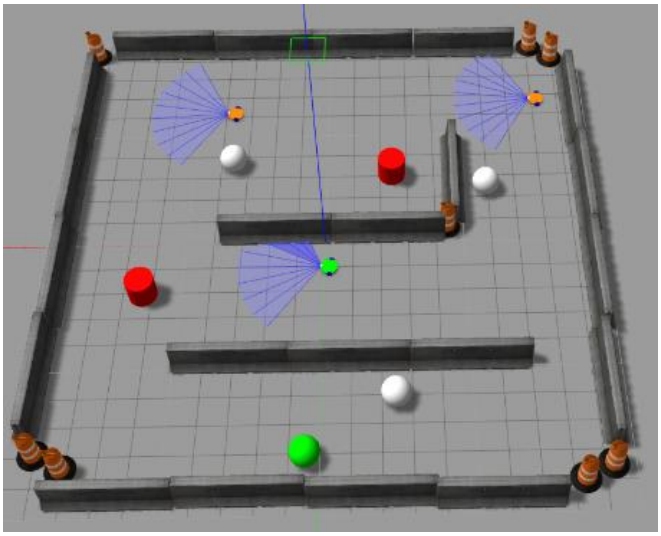


Figura I.2. Ingreso de los robots en el mundo virtual. [Fuente propia]

Después, en un nuevo terminal se procede a abrir la cámara del robot explorador mediante el comando que se detalla a continuación, obteniendo como resultado lo que se muestra en la Figura I.3.

- *rosrun image_view image_view image:=/cameras/head_camera/image*

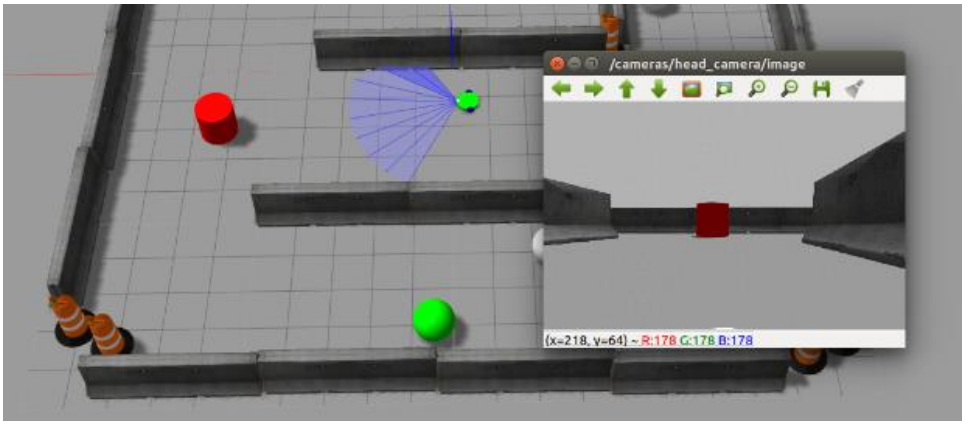


Figura I.3. Vista de la cámara del robot explorador. [Fuente propia]

A continuación, se procede a ejecutar los diferentes algoritmos de evasión de obstáculos y de consenso en cada robot, cada uno de estos comandos, que se presentan a continuación, son ejecutados cada uno en su propio terminal:

Robot explorador:

- *rosrun motion_plan deteccion.py*
- *rosrun motion_plan algoritmo1.py*

Robot obrero 1:

- *rosrun motion_plan algoritmorobot2.py*

Robot obrero 2:

- *rosrun motion_plan algoritmorobot3.py*

Con dicho proceso se inicia el movimiento del robot explorador hasta encontrar el objetivo que, en este caso, es una esfera de color verde; mientras que los robots obreros se encuentran inmóviles a la espera de dicho encuentro, a la vez que todos los robots se comunican constantemente entre sí mediante el nodo maestro Gazebo.

EJECUCIÓN DE LA INTERFAZ DE USUARIO

En un nuevo terminal se procede a ejecutar la pantalla de interfaz de usuario, que permite al usuario interactuar con el entorno virtual al introducir obstáculos en cualquier momento durante la ejecución de la simulación, mediante el comando que se presenta a continuación. El resultado de dicha pantalla se muestra en la Figura I.4.

- *rosrun motion_plan ventana0.py*

Figura I.4. Interfaz de usuario – Selección de obstáculos. [Fuente propia]

Como se observa, los obstáculos que pueden ser ingresados por el usuario son una esfera blanca o un cilindro rojo, el usuario puede seleccionar cualquiera de los dos obstáculos al elegir el botón que corresponde a cada uno. Al hacerlo, se abrirá la pantalla que se muestra en la Figura I.5.

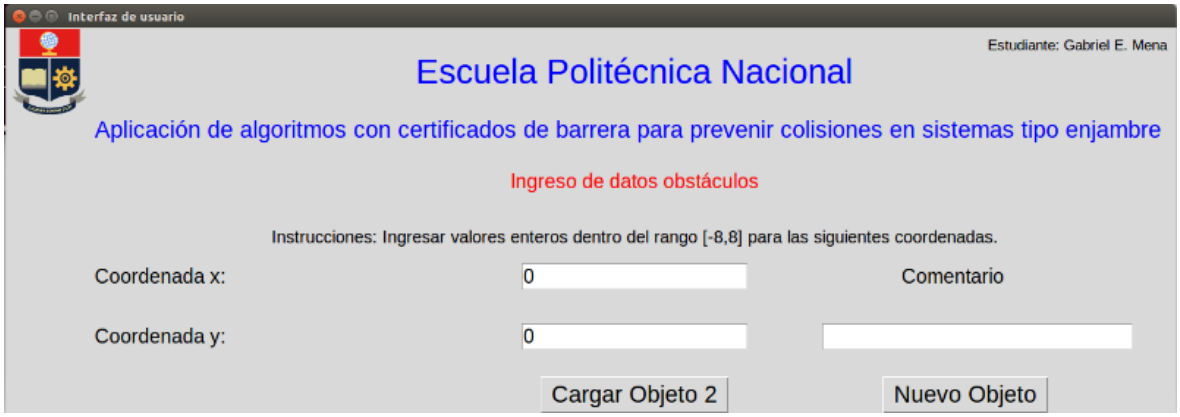


Figura I.5. Interfaz de usuario – Ingreso de coordenadas. [Fuente propia]

Esta pantalla permite al usuario ingresar las coordenadas de aparición del obstáculo dentro del mundo virtual, siempre que las mismas se encuentren dentro del rango [-8, 8] en sus respectivos ejes. En el recuadro de comentario se presentarán tres diferentes casos, que son:

- *Objeto ingresado correctamente: Este mensaje se presenta cuando ha sido exitoso el ingreso del objeto en las coordenadas establecidas por el usuario.*
- *Coordenada incorrecta: Este mensaje aparecerá cuando las coordenadas ingresadas por el usuario no se encuentran dentro del rango establecido para cada eje*
- *Crear nuevo objeto: Este mensaje aparecerá después de que el usuario ha ingresado exitosamente un obstáculo, indicándole que es posible que ingrese uno nuevo, si así lo desea.*

ANEXO II

PARAMETRIZACIÓN DE ARCHIVOS XACRO

En este anexo se encuentran las tablas correspondientes a los tres archivos .xacro cuya parametrización no fue detallada en la sección 2.1.2. del presente trabajo de integración curricular.

Structurelaser.xacro

Tabla II.1. Parametrización de un archivo Structurelaser.xacro. [Fuente propia]

Parte	Parámetros
Estructuración del archivo xacro	Versión: 1 Nombre: Sensor laser Visualización
Inicio: link	material
Propiedades: Inercia, colisión y visualización, entre otros.	ambiente= 0,3 Difuso= 0.7 Especular= 0.01 Emisividad=1
Cierre: link	material

Macro.xacro

Tabla II.2. Parametrización de un archivo Macro.xacro. [Fuente propia]

Parte	Parámetros
Estructuración del archivo xacro	Versión: 1 Nombre: Rueda izquierda y derecha
Inicio: link	Link Name
Propiedades: Inercia, colisión y visualización, entre otros.	Inercia: Masa=”0.2” Origen rpy="0 1.5707 1.5707" xyz="0 0 0" Inercia ixx="0.00052" ixy="0" ixz="0" iyy="0.00052 Colisión Origen rpy="0 1.5707 1.5707" xyz="0 0 0 Largo cilindro="0.04" Radio="0.1" Visualización Origen rpy="0 1.5707 1.5707" xyz="0 0 0" Largo Cilindo="0.04" Radio="0.1”

Cierre: link	Link Name

Gazebo.xacro

Tabla II.3. Parametrización de un archivo Gazebo.xacro. [Fuente propia]

Parte	Parámetros
Estructuración del archivo xacro	Versión: 1 Nombre: robot generalidades
Inicio: link	gazebo
Propiedades: Inercia, colisión y visualización, entre otros.	Referencia="link_chassis" Material=Verde Referencia=”Rueda derecha” Material=Azul Referencia=”Rueda izqueirda” Material=Azul Ingreso del sensor laser Código Ingreso de sensor camara Código
Cierre: link	gazebo

ANEXO III

DIAGRAMA DE FLUJO DEL ALGORITMO DE PREVENCIÓN DE COLISIONES

En este anexo se encuentra el diagrama de flujo que corresponde al algoritmo de prevención de colisiones, cuya estructura y funcionamiento fueron detallados en la sección 2.3.1. del presente trabajo de integración curricular.

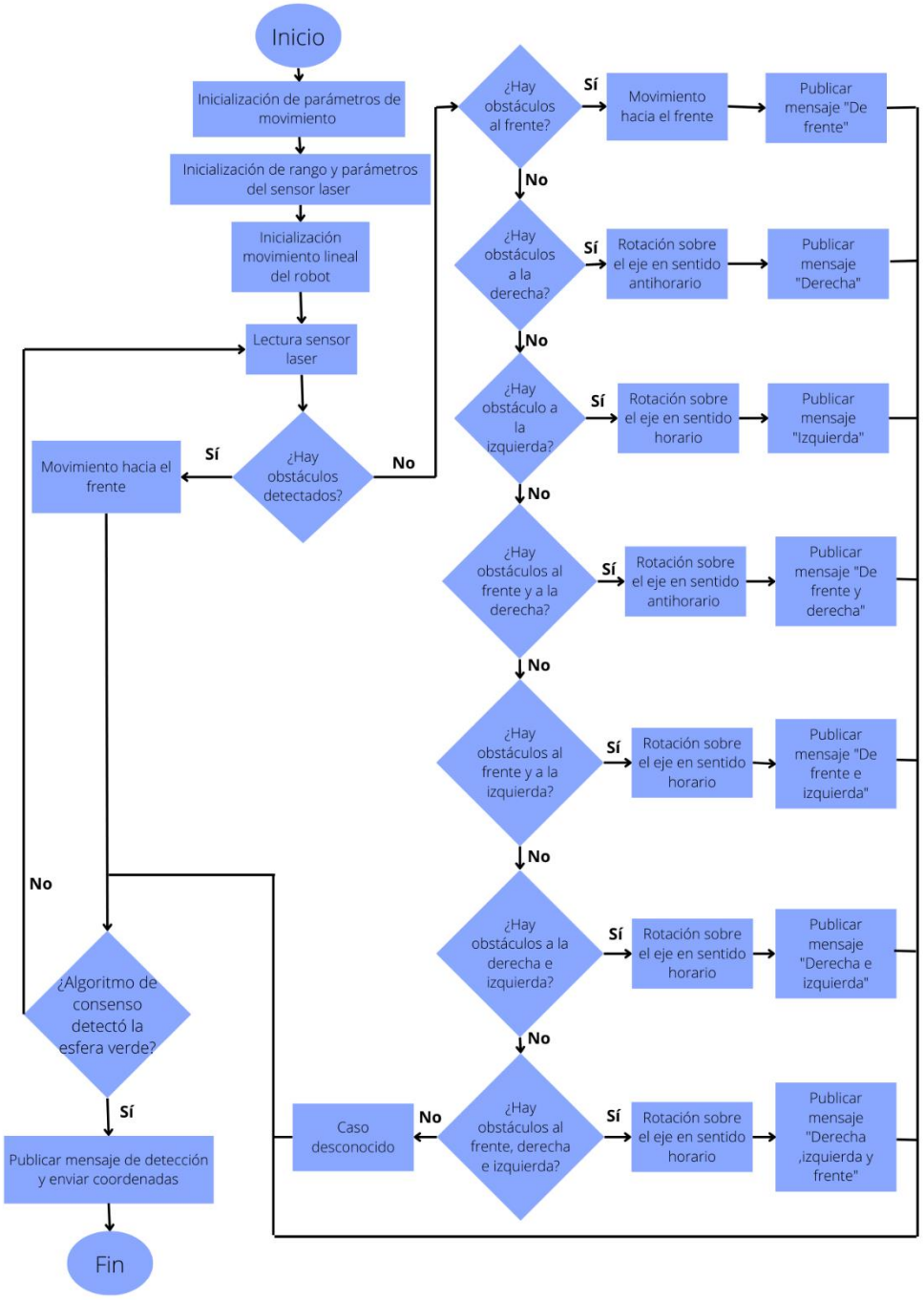


Figura III.6. Diagrama de flujo del algoritmo de prevención de colisiones. [Fuente propia]

ANEXO IV

DIAGRAMA DE FLUJO DEL ALGORITMO DE CONSENSO

En este anexo se encuentra el diagrama de flujo que corresponde al algoritmo de consenso, cuya estructura y funcionamiento fueron detallados en la sección 2.3.2. del presente trabajo de integración curricular.

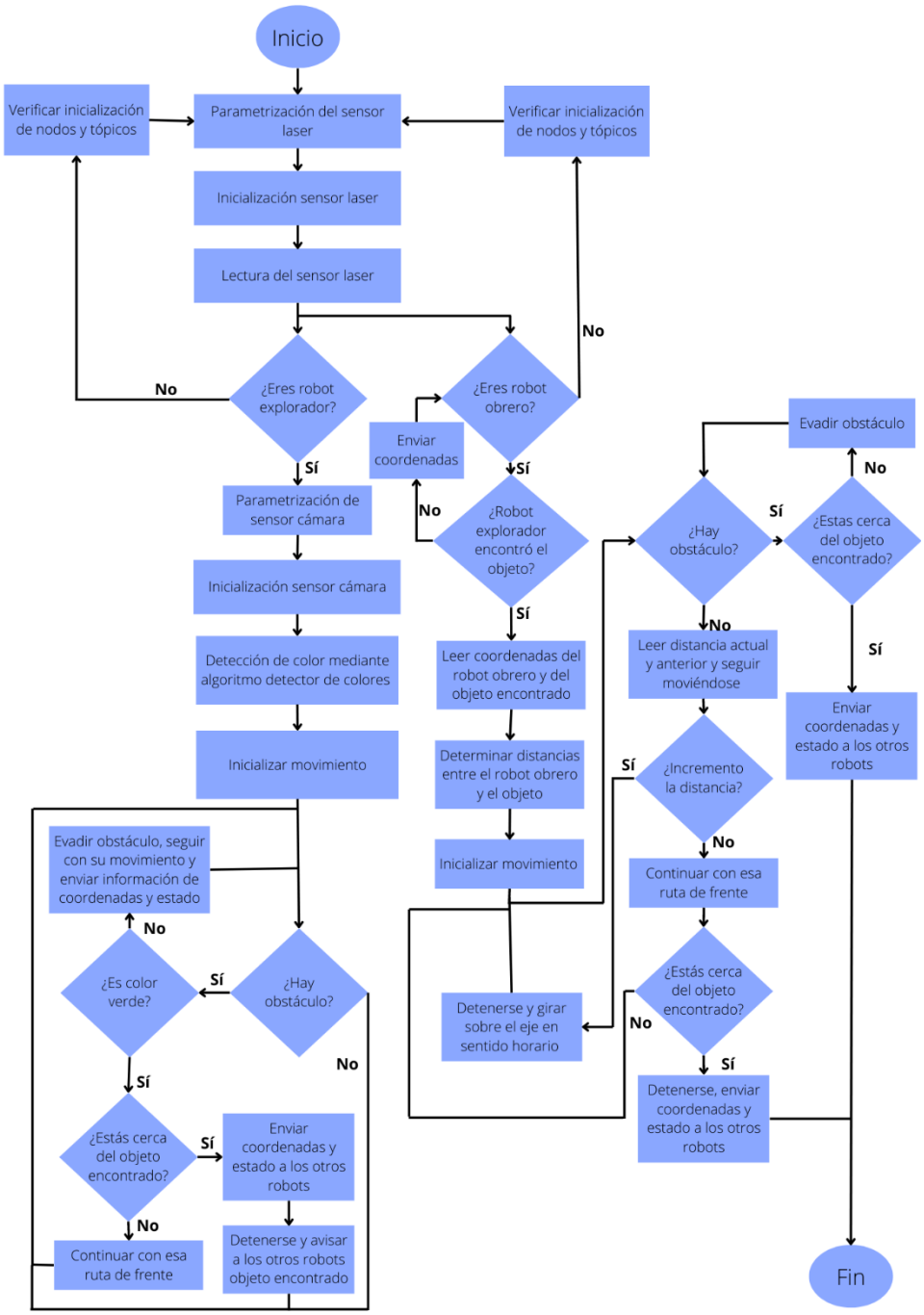


Figura IV.1. Diagrama de flujo del algoritmo de consenso. [Fuente propia]

ANEXO V

LINKS DE LOS VIDEOS DE PRUEBAS Y RESULTADOS

En este Anexo se encuentran los enlaces de los videos correspondientes a las pruebas realizadas en el presente Trabajo de Integración Curricular, de modo que el lector pueda comprender de mejor manera los procedimientos realizados. Asimismo, se encuentra el enlace de GitHub en el que se encuentra este trabajo.

- **Prueba de interfaz de usuario:**

  https://www.youtube.com/watch?v=4An6-6FeMew&list=PLBLhBdbySVdYU-2sOWJHpk6xKFAqO3krr&index=3

- **Prueba del algoritmo de evasión de obstáculos:**

  https://www.youtube.com/watch?v=0s38QUX1j6g&list=PLBLhBdbySVdYU-2sOWJHpk6xKFAqO3krr&index=1

- **Prueba del algoritmo detección de colores y visión artificial:**

  https://www.youtube.com/watch?v=UZPr4JbfzeY&list=PLBLhBdbySVdYU-2sOWJHpk6xKFAqO3krr&index=5

- **Prueba de Intercomunicación y lectura de coordenadas:**

  https://www.youtube.com/watch?v=fJI9VI-3rDw&list=PLBLhBdbySVdYU-2sOWJHpk6xKFAqO3krr&index=2

- **Pruebas en diferentes escenarios:**

  https://www.youtube.com/watch?v=jdKII51I-oo&list=PLBLhBdbySVdYU-2sOWJHpk6xKFAqO3krr&index=4

- **Enlace repositorio en GitHub:**

  https://github.com/GabrielMena1/TIC-EPN