ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

DISEÑO Y SIMULACIÓN DE UN SISTEMA MULTI-ROBOT TIPO ENJAMBRE, MEDIANTE EL USO DEL ENTORNO ROS-MATLAB, ENFOCADO A LA DETECCIÓN Y EVASIÓN DE OBSTÁCULOS E INTER-COLISIONES PARA EJECUTAR LA TAREA DE INTERCAMBIO DE POSICIONES (SWAP POSITIONS).

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

MARCO ALEXANDER ORTEGA RODRÍGUEZ

marco.ortega@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

patricio.cruz@epn.edu.ec

DMQ, Febrero 2022

CERTIFICACIONES

Yo, Marco Alexander Ortega Rodríguez 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.

MARCO ALEXANDER ORTEGA RODRÍGUEZ

Certifico que el presente trabajo de integración curricular fue desarrollado por Marco Alexander Ortega Rodríguez, 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 (los) producto(s) resultante(s) 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.

Marco Alexander Ortega Rodríguez

Ing. Patricio Javier Cruz Dávalos, PhD.

DEDICATORIA

Este proyecto va dedicado a mis padres, abuelitos y familiares que, con su paciencia, esfuerzo y cariño incondicional, me han permitido culminar una etapa más de mi vida.

Así mismo a todos mis amigos y profesores que con sus consejos, enseñanzas y apoyo me han formado no solo como un mejor estudiante, sino como una mejor persona.

Por último, me gustaría dedicar este proyecto a todos los futuros estudiantes e investigadores que se encuentren con este, y el mismo pueda ser usado no solo como una referencia más. Sino sea una base o guía para el desarrollo de la robótica y así mismo una fuente de inspiración para nuevas investigaciones y/o aplicaciones especialmente al integrar las herramientas de ROS y MATLAB.

AGRADECIMIENTO

Mi más grande agradecimiento a mi Alma Mater la Escuela Politécnica Nacional que me ha permitido formarme en sus aulas y me ha enseñado que a partir de dedicación, constancia y esfuerzo es posible lograr cualquier meta o sueño,

Además, que ha sido como mi segundo hogar, en donde he conocido gente maravillosa y forjado grandes amistades, que me han enseñado la verdadera importancia de los buenos amigos y que se han convertido en un soporte de mi vida dentro y fuera de la universidad. Entre ellos un perenne agradecimiento a: Gaby, Martin, Mateo, Samuel, Luis, Rodrigo y todos aquellos que me dieron su confianza y palabras de aliento cuando fue necesario.

Un agradecimiento especial al Doc. Patricio Cruz que con su paciencia, comentarios y sugerencias me ha guiado en todo el proceso para la culminación de este proyecto.

Finalmente, agradezco a mi familia por su comprensión y apoyo constante, de tal manera que logre superarme día con día no solo como profesional, sino en todos y cada uno de los aspectos.

ÍNDICE DE CONTENIDO

RESUMEN

Dentro del campo de la robótica, especialmente en el subcampo de los sistemas multi-agentes, existen múltiples soluciones o propuestas, como los sistemas centralizados, los cuales facilitan el control como un conjunto. Sin embargo, esa misma centralización es una posible vulnerabilidad ante fallos del sistema principal y una limitante ante el escalamiento o reducción del grupo, como es el caso de los sistemas enjambre. Los cuales, al tener el mismo esquema de control en cada uno de sus agentes, facilita su desarrollo y escalabilidad.

El desarrollo de sistemas robóticos tipo enjambre aún es un campo emergente y lleno de retos. Uno de estos es la capacidad de los agentes para la evasión de obstáculos en entornos desconocidos. Por lo cual, para resolverlo, en este trabajo se plantea una posible arquitectura de control basada en subsunción, con el algoritmo VFH+ para la evasión de obstáculos e inter-colisiones entre los agentes. Esto con el objetivo principal de resolver la tarea global del sistema enjambre, que en este caso, es el intercambio de posiciones (Swap Positions). Esta tarea cobra vital importancia en tareas de navegación y exploración especialmente en entornos desconocidos, además, sirve como base para otros comportamientos como la distribución de tareas o formación de patrones.

Justamente para evaluar la aplicación de intercambio de posiciones y la evasión de colisiones, se desarrolla un banco de pruebas virtual para la investigación de sistemas multi-robóticos incorporando las herramientas de ROS, MATLAB y la plataforma robótica TurtleBot3. En este entorno virtual se ejecutan varias pruebas para verificar el funcionamiento de la arquitectura de control planteada, así como la flexibilidad y escalabilidad del sistema enjambre.

PALABRAS CLAVE: Robótica de Enjambre, Intercambio de Posiciones, Evasión de Obstáculos, TurtleBot3, ROS Toolbox.

ABSTRACT

Within the field of robotics, especially in the multi-agent systems research, there are multiple solutions or proposals based on centralized systems that facilitate the group control. However, this centralization is a potential vulnerability to failures of the major system and a limitation to the scaling or reduction of the group, as it is with swarm systems. For swarms, having the same control system in each agent facilitates their development and scalability.

The development of swarm-type robotic systems is still an emerging field with full of challenges. One of these is the agent ability to avoid obstacles in unknown environments. Therefore, to solve it, a possible control architecture based on subsumption is proposed in this work by applying the VFH+ algorithm for obstacle avoidance and agent inter-collisions. The principal global task for the swarm system in this project is the exchange of positions (Swap Positions). This task is crucial for navigation and exploration purposes, especially in unknown environments. Also, it serves as the basis for other behaviors such as task allocation or formation of patterns.

To evaluate the application of Swap Position and the avoidance algorithm, a virtual test bench for multi-robotic systems investigation is developed by incorporating ROS, MATLAB development tools and the TurtleBot3 robotic platform. In this virtual environment, several tests are executed to verify the operation of the proposed control architecture and the flexibility and scalability of the swarm system.

KEYWORDS: Swarm Robotics, Swap Positions, Collision Avoidance, TurtleBot3, ROS Toolbox.

1.INTRODUCCIÓN

La robótica de enjambre (Swarm Robotics) es el campo de estudio dentro del área de la robótica, específicamente una subárea de la inteligencia artificial y sistemas multi-agentes, que se encarga del estudio de la coordinación de una gran cantidad de robots cuya estructura y capacidad de inteligencia son relativamente simples. Este campo toma como principal inspiración el comportamiento observado en múltiples insectos que presentan comportamientos sociales, como hormigas, abejas, termitas, entre otros, así como el comportamiento colectivo de cardúmenes de peces o bandadas de algunas especies de aves. Estos son grandes ejemplos de cómo una gran cantidad de individuos simples pueden interactuar entre sí para crear sistemas con inteligencia colectiva [1]–[3].

Dentro de los sistemas enjambre, el comportamiento colectivo surge de forma orgánica y codependiente de la interacción entre los individuos y en algunos casos, la interacción de los individuos con el ambiente. Además, los sistemas tipo enjambre se caracterizan por ser capaces de cumplir tareas que van más allá de las capacidades individuales de un solo robot [1], [3].

A pesar, de que el concepto de sistemas enjambres dentro de la robótica ha existido desde la década de los 80's, esta rama ha tomado un reciente impulso debido a la evolución de la ingeniería electrónica y su capacidad actual de desarrollar plataformas más pequeñas y mucho más potentes computacionalmente. Además, han tenido un fuerte impacto en este campo, las ventajas que presenta la comunicación wireless actual, así como el desarrollo de otros sistemas basados en inteligencia artificial [2].

Considerando el actual desarrollo en esta área y la importancia dada al comportamiento de cada uno de los individuos, el desarrollo de algoritmos que faciliten o mejoren la autonomía de cada uno de los agentes (robots que conforman el enjambre), es vital para el funcionamiento de este, ya que, si los individuos presentan fallas, el sistema no puede cumplir su objetivo global. Es así como, para lograr ese cierto nivel de autonomía, cada uno de los robots debe tener capacidades de planeación y navegación en el entorno donde se desarrolla el enjambre, con la capacidad detección y evasión de obstáculos [1], [2], [4].

Teniendo en cuenta la importancia que tiene la evasión de obstáculos en el desarrollo de un sistema robótico tipo enjambre, en este trabajo, se plantea la utilización de un algoritmo que evite las colisiones que se pueden presentar en este tipo de sistemas ya sea con su entorno o entre sus agentes (inter-colisiones). Esto, a la vez que se resuelve la tarea principal asignada al enjambre que se enfocará en el intercambio de posiciones de los robots dentro del enjambre (swap positions). Esta tarea requiere que cada uno de los robots

inicie en una ubicación inicial y se le comande a todo el grupo a intercambiar de ubicación entre ellos. Dicha tarea presenta una alta probabilidad de inter-colisiones y de choques con posibles obstáculos estáticos presentes en el entorno de trabajo [5], [6]. Por ende, el presente trabajo consiste, además, en el desarrollo de un entorno de simulación (banco de pruebas virtual) basado en el entorno de ROS, Gazebo y MATLAB® donde se compruebe el funcionamiento del algoritmo de evasión de obstáculos o inter-colisiones desarrollado para un sistema enjambre terrestre formado por robots de tracción diferencial. Los agentes virtuales que conforman el enjambre son creados en base a la plataforma práctica TurtleBot3 [7].

1.1. OBJETIVO GENERAL

Desarrollar y simular un sistema multi-robot tipo enjambre, mediante el entorno de ROS-MATLAB, enfocado a la detección y evasión de obstáculos e inter-colisiones para ejecutar la tarea de intercambio de posiciones (swap positions).

1.2. OBJETIVOS ESPECÍFICOS

- Revisar las características de un sistema robótico tipo enjambre y los algoritmos para la evasión de colisiones en el mismo, además, de familiarizarse con el entorno de ROS, GAZEBO y su interconexión a MATLAB® para el desarrollo del sistema multi-agente.
- Implementar un banco de pruebas conformado por un sistema-robótico tipo enjambre, creado en base a un robot virtual TurtleBot3, y su entorno de trabajo, utilizando las herramientas disponibles en ROS-Gazebo.
- Diseñar el algoritmo para ejecutar la tarea de intercambio de posición considerando la evasión de obstáculos y de inter-colisiones entre los agentes que conforman el sistema enjambre usando las herramientas de MATLAB y su interconexión con ROS.
- Desarrollar las pruebas para corroborar el funcionamiento del algoritmo diseñado y la ejecución de la tarea principal del enjambre dentro de diferentes entornos, así como variando el número de agentes para su posterior análisis de resultados.

1.3. ALCANCE

- Se realiza una revisión bibliográfica acerca del comportamiento y características de un sistema multi-agente tipo enjambre enfocado a robots terrestres, así como de los algoritmos usados para la evasión de obstáculos y las posibles topologías del sistema.
- Se revisan las características y restricciones de movimiento que presenta el robot de tracción diferencial TurtleBot3. Cabe recalcar, que el proyecto no se enfocará en el control a bajo nivel de esta plataforma, es decir, el control de velocidad interno de cada una de las llantas, sino que se enviarán comandos de velocidad lineal y angular a cada uno de los agentes basados en esta plataforma y que conformarán el enjambre.
- Se realiza una revisión y estudio del entorno de ROS y las herramientas que este presenta para el desarrollo, implementación y simulación de sistemas multi-agentes tipo enjambre, así como de la interacción con MATLAB por medio del toolbox de ROS.
- Se diseña y desarrolla el banco de pruebas virtual con las herramientas de ROS-Gazebo, que emulará las características físicas del TurtleBot y su entorno. Además, se diseñarán las adecuaciones necesarias para la intercomunicación con MATLAB, tal que se pueda realizar la detección de obstáculos, a través de la emulación de sensores, y el envío de comandos a los actuadores de cada uno de los agentes.
- Se diseña el algoritmo de control reactivo para la evasión de obstáculos, por lo que será un algoritmo que modificará el comportamiento del robot solamente si existe una posible colisión (con un objeto o con otro agente), de tal manera que se ejecute y priorice la tarea principal del enjambre que será el intercambio de posiciones, esto teniendo en cuenta las restricciones no-holonómicas de los robots.
- Se implementa el sistema multi-robot tipo enjambre dentro del banco de pruebas previamente diseñado, el cual estará conformado de por lo menos 4 agentes y tendrá la posibilidad de ser escalable.
- Se realizan pruebas que llevarán al sistema a situaciones de estrés (cruces de robots, diferentes tipos de obstáculos en un entorno estructurado).
- Se analizan los resultados obtenidos en las distintas pruebas, previamente mencionadas, de tal manera que se determine las capacidades y limitaciones del algoritmo dentro de los distintos entornos.

1.4. MARCO TEÓRICO

1.4.1. ROBÓTICA MÓVIL

La robótica móvil, hace referencia a la subclasificación o rama de la robótica donde los robots poseen una gran capacidad de desplazamiento o un sistema de locomoción que permite la movilización de estos en varios entornos de trabajo, ya sean estructurados o no estructurados, con la finalidad de alcanzar ciertos objetivos de manera autónoma. Para que un robot móvil pueda operar en ambientes, donde se desconoce o se tiene cierta incertidumbre sobre la posición e identificación de objetos u obstáculos, es necesario que la planeación sea capaz de generar trayectorias y guiar al robot por medio de este entorno en función de la información proveniente de sensores, adquiriendo así un cierto nivel de autonomía para su desplazamiento dentro del ambiente de forma segura [8]. Además, un robot móvil se caracteriza por su capacidad de percepción y acción (enlazadas por medio de una lógica de control o conexión inteligente), que definen su comportamiento; es así como, se distinguen ciertas características básicas que debe cumplir para que un vehículo sea considerado un robot móvil autónomo [8]:

- **Percepción:** capacidad del robot para relacionarse con su entorno por medio de su sistema sensorial, esto lleva a la capacidad de análisis y síntesis de la información entregada por los sensores, de tal manera de determinar su localización y si se requiere poder formar mapas de manera global o local del entorno, o caso contrario la actuación del robot [8]. Dentro de los sensores usados se categorizan dos grandes grupos:

  -Propioceptivos: permiten conocer los valores internos del sistema, tales como posición de la plataforma, velocidad de los motores, nivel de batería, etc. Estos sensores pueden ser encoders, potenciómetros, giroscopios, entre otros.

  -Exteroceptivos: estos sensores son usados para obtener datos del entorno, como lo son los sensores infrarrojos, sensores ultrasónicos, LIDARs, etc.

- **Navegación:** una vez se conoce el entorno, bien sea de manera global (se tiene un mapeo de todo el entorno) o local (solo se conoce información actual en base a sensores), el robot móvil debe tener la capacidad de decidir y ejecutar las acciones requeridas en función de su estado actual de tal manera que consiga alcanzar su(s) objetivo(s). Por lo cual, la capacidad de navegación es la que permite la planificación de trayectorias globales (en el caso que se haya mapeado el entorno de manera parcial o completa) o la planificación de trayectorias locales (en el caso de que se presenten obstáculos inesperados - evasión de obstáculos) [8], [9].

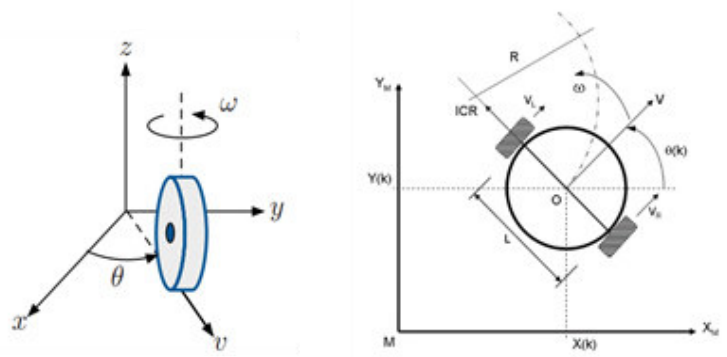
- **Locomoción:** el sistema de locomoción de un robot móvil es uno de los aspectos más importantes en el diseño y es fundamental para su movilidad, ya que no solo es el medio con el cual el robot se moviliza, sino, que afecta fundamentalmente a factores como su maniobrabilidad, controlabilidad, estabilidad, eficiencia, entre otros. Es así como, dependiendo el sistema de locomoción, se determinan las capacidades dinámicas y cinemáticas que tendrá el robot. Dentro de los sistemas de locomoción se destacan los robots de ruedas, de patas, de oruga e híbridos. Los robots de ruedas son especialmente diseñados para un comportamiento terrestre en suelos blandos o duros, además de ser los más usados, principalmente, por su eficiencia, simplicidad y su lógica de control es relativamente menos compleja [8], [9].

1.4.1.1. Robots de Tracción Diferencial

Dentro de los sistemas de locomoción basados en ruedas se destacan los robots de tracción diferencial. Para la representación de estos se considera la modelación tipo uniciclo, tal como se muestra en la Figura 1.1.En esta modelación se tiene como coordenadas base al vector $q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$, las cuales también corresponden a los 3 grados de libertad del robot, que son su posición en el plano (x, y) y la orientación determinada por el ángulo θ [10], [11]. Su modelo cinemático viene dado por:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \qquad (1.1)$$

Donde v y ω representan la velocidad lineal y angular, respectivamente.



(a) (b)

Figura 1.1. (a) Representación uniciclo [10], (b) Esquema Robot de Tracción Diferencial [11].

La representación uniciclo permite simplificar el análisis cinemático de un robot de tracción diferencial, y por ende el desarrollo de sus respectivos controladores; sin embargo, es necesario tener en cuenta sus restricciones no-holonómicas. Esto debido a que, como se muestra en la Figura 1.1, la rigidez de las ruedas limita el movimiento que puede realizar, es decir, los movimientos posibles para el robot son rotaciones respecto al eje central de ruedas y traslaciones en dirección longitudinal del robot. Combinando estos movimientos es posible que el robot realice rotaciones sobre cualquier punto que pertenezca a la línea imaginaria por donde pasa el eje de ruedas del robot, denominado Centro de Rotación Instantáneo (ICR), mostrado en la Figura 1.1. Para este caso, el ICR viene definido por:

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

De la ecuación anterior se conoce que R es el radio de curvatura, mostrado en la Figura 1.1. Es a partir de este análisis que se determina que modificando el ICR en función del tiempo es posible realizar cualquier tipo de trayectorias planas, evitando las restricciones de movimientos de la plataforma móvil [11].

Restricciones Holonómicas:

Para que un sistema robótico sea considerado holonómico debe tener la capacidad de llegar desde cualquier punto inicial a cualquier punto final del sistema de coordenadas usadas. En el caso particular de la plataforma robótica usada en este proyecto, un robot de tracción diferencial, la imposibilidad de realizar movimientos laterales debido a la rigidez de sus llantas, lo clasifica como un sistema con restricciones no-holonómicas.

1.4.2. SISTEMAS MULTI-AGENTE TIPO ENJAMBRE

Una vez revisadas las características que debe poseer un robot autónomo y sus limitantes de locomoción, es necesario conocer como la interacción de múltiples robots permite la formación de sistemas más complejos, como los tipos enjambre.

Los sistemas multi-agentes de tipo enjambre (Swarm) tienen como objeto de estudio la coordinación de grandes grupos de robots (agentes) a través del uso de reglas locales relativamente simples. Estos sistemas se caracterizan por tener un control descentralizado y distribuido lo que dota al sistema de cierta robustez y flexibilidad, esto con el fin de cumplir tareas de alta complejidad respecto a las capacidades de cada uno de los robots [4]. Acorde a [13]: *"El grupo de robots no es solo un grupo. Este tiene algunas características especiales, las cuales se pueden encontrar en los enjambres de insectos, estos son: control descentralizado, falta de sincronización además de miembros simples y (cuasi) idénticos".*

La idea de que un grupo colaborativo de agentes robóticos que presente comportamientos colectivos tiene como principal inspiración el comportamiento de insectos sociales tales como las hormigas, abejas, termitas, además de grupos de aves y peces con comportamientos similares. En estos sistemas, los individuos desconocen la información global del estado de la colonia o enjambre, por lo que no existe un líder que guie a los otros individuos o agentes para cumplir la meta global del conjunto. Es a través del conocimiento distribuido entre todos los agentes que se pueden cumplir las metas que individualmente serían imposibles de lograr para uno solo de los agentes. Es importante destacar, que la capacidad de comunicación entre los agentes se basa en el concepto de localidad, ya que cada uno solo conoce su propio entorno y no la situación completa del enjambre [4], [14], [15].

1.4.2.1. Características Principales

Entre las características que definen a un sistema tipo enjambre se destacan:

Robustez: Debido a que el sistema presenta una lógica distribuida (no se encuentra centralizada), es capaz de realizar o cumplir la tarea objetivo a pesar de que existan perturbaciones en el comportamiento en algunos de sus individuos. Entre estas perturbaciones se pueden tener cambio en el entorno, cambio en el número de obstáculos, o fallos en alguno(s) de los agentes [2], [3].

Flexibilidad: Si bien el sistema en general debe cumplir la tarea objetivo, el control de cada uno de los agentes se lo realiza de manera local, es decir dentro del software de cada uno de los robots. Por lo cual, estos requieren de autonomía para desplazarse y ser capaces de evitar colisiones no solo con su entorno, sino también con obstáculos dinámicos como los otros agentes pertenecientes al enjambre [2].

Escalabilidad: Como ya se mencionó previamente, la ejecución de los algoritmos de control a nivel local (en el software propio del mismo robot) y la lógica distribuida del sistema, permite que cada uno de los agentes pueda comportarse de manera autónoma y por esta razón el sistema es capaz de aumentar o disminuir la cantidad agentes sin comprometer el cumplimiento de la tarea objetivo [2].

A más de estas características, en función de los fundamentos de un sistema enjambre, este debe ser creado de tal manera que los comportamientos colectivos surjan como resultado de las interacciones locales entre agentes y su entorno. Por lo cual, los agentes deberán tener ciertas características para conformar el enjambre:

1. Los robots del enjambre deberán ser independientes, con cierto nivel de autonomía para su desenvolvimiento en el entorno sin depender de sus pares [2].
2. Los robots deberán ser simples (generalmente pequeños y de bajo costo) y no podrán cumplir la tarea del enjambre de forma individual o presentarán dificultades para cumplirla, por lo que el trabajo colaborativo se vuelve indispensable [2], [4].
3. El sistema debe ser homogéneo, es decir, todos los agentes deberán ser iguales [2], [4], [14] .
4. Los robots tienen capacidades locales de comunicación y sensado, lo que asegura que la coordinación del sistema sea distribuida y promueve la escalabilidad del sistema [2], [4].
5. Las normas o reglas que controlen el enjambre serán relativamente simples y ejecutadas individualmente [4].

1.4.2.2. Taxonomía de los Comportamientos Enjambre

Dentro de los algoritmos de enjambre, el objetivo principal es el surgimiento de forma orgánica de un comportamiento colectivo en función del comportamiento individual de cada uno de los agentes, es así como el comportamiento de este se limita al comportamiento local, sin embargo, el comportamiento colectivo da origen a distintas categorías organizadas en la Figura 1.2 [14].

Organización Espacial (Spatial Organization): Estos comportamientos permiten el movimiento de los robots en el ambiente de tal manera que el sistema enjambre quede organizado en el entorno como es el caso de la Agregación (Aggregation) [16], Formación de patrones (Pattern Formation) [17], autoensamblaje (Self-Assembly) [18] o una organización con objetos externos al enjambre (Object Clustering and Assembly) [19].

Navegación (Navigation): Estos comportamientos coordinan el movimiento del enjambre o de los robots en el ambiente [14]. En esta clasificación se destaca actividades como la exploración principalmente en ambientes desconocidos [20], el movimiento coordinado como el comportamiento rebaño (Flocking) [10], intercambio de posiciones (Swap Positions) [5], [6], o en el caso de la interacción de objetos externos al enjambre el transporte colectivo (Collective Transport) [21].

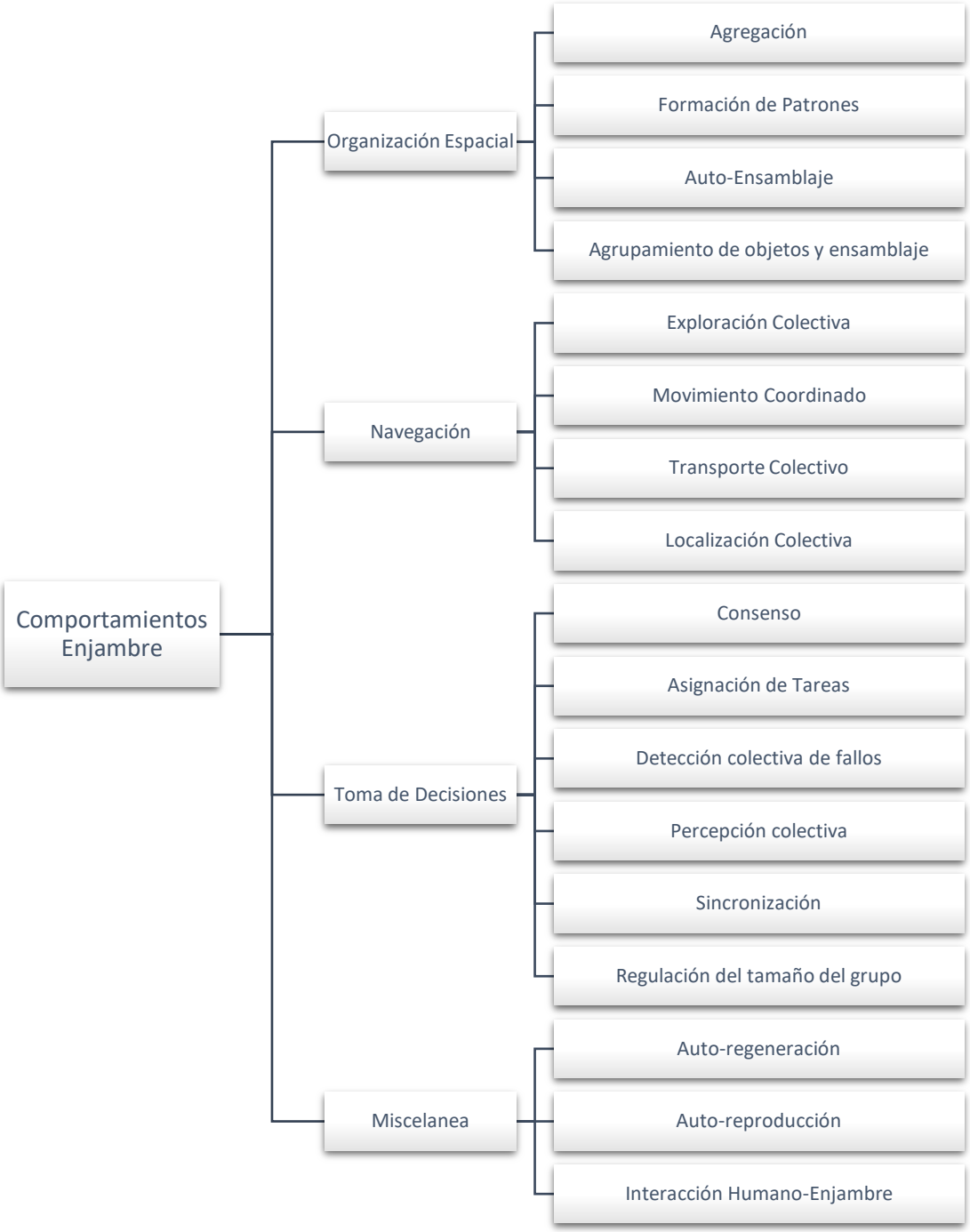


Figura 1.2. Taxonomía de comportamiento Enjambre (realizado en base a [14]).

Toma de Decisiones (Decision Making): Estos se caracterizan por permitir la toma de decisiones o capacidad de elección conjunta en el enjambre en función de los datos de cada uno de los agentes [22].

Miscelánea (Miscellanous): Dentro de esta categoría de comportamientos se encasillan los comportamientos que no encajan en las categorías previamente mencionadas. En esta se destacan Autocuración (Self-Healing), que permite la disminuir el impacto ante la falla

de alguno de los agentes y Auto-reproducción (Self-Reproduction) lo que permite la creación de nuevos robots o replicar patrones creados por algunos individuos del enjambre [23]. Por otra parte, en esta categoría se da énfasis a la interacción humano-enjambre y sus posibles aplicaciones [24] .

1.4.2.3. Topologías Sistema Enjambre

Teniendo en cuenta que una de las principales características de un sistema enjambre es la descentralización de sus agentes, la intercomunicación de estos puede presentar diferentes estructuras o topologías, y en función de estas modificar no solo el comportamiento, sino por ende el desempeño en general del sistema enjambre [25]. Dentro de las posibles topologías de un sistema enjambre se destacan las configuraciones anillo (ring), malla (mesh), caveman, aleatoria (random), entre otras, representadas en la Figura 1.3. Además, se muestra que tipo de topologías deberían ser usadas o son óptimas en función del grado óptimo de conexión entre agentes (k*), la frecuencia de comunicación (ω), y la comparación entre el peso capacidad de las conexiones (ω_{ij}) y la distancia de los agentes dentro de la topología ($|i-j|$) [25]. Este análisis cobra importancia especialmente cuando se cuenta con enjambres con cientos o miles de agentes.

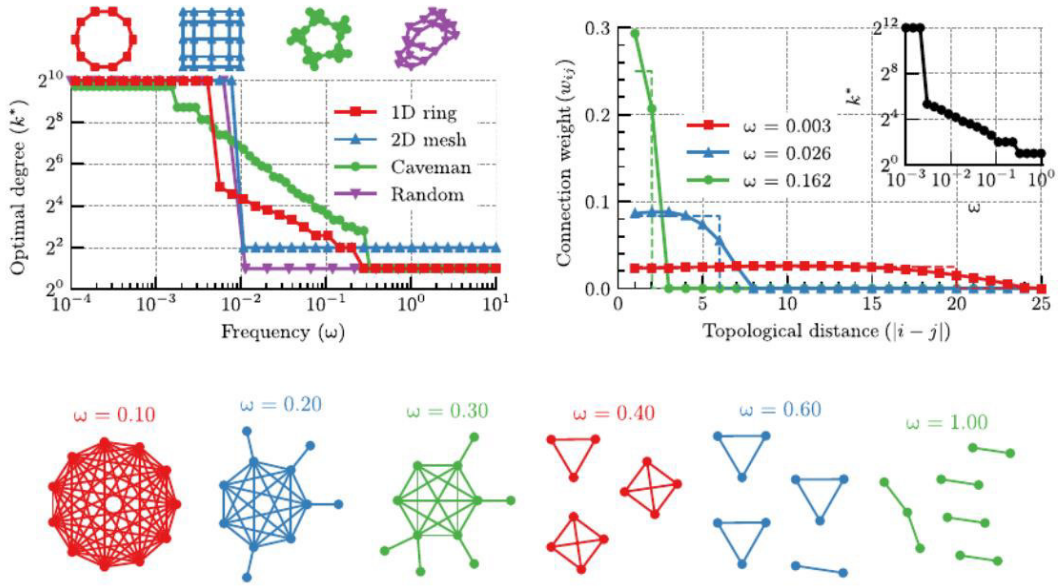


Figura 1.3. Selección de Topologías de Red óptimas para sistemas tipo enjambre para respuesta colectiva [25].

1.4.2.4. Aplicaciones:

Teniendo en cuenta que los sistemas robóticos de tipo enjambre es un campo de investigación relativamente nuevo, su incorporación en la industria no ha sido aceptada

completamente. Por lo que, el enfoque de la investigación de la robótica de enjambre es hacia el desarrollo de plataformas robóticas, donde se pueda analizar y probar distintos algoritmos. Actualmente, estas plataformas de investigación se enfocan en validaciones o pruebas a nivel de laboratorio, es decir con un TRL "technology readiness levels" (nivel de maduración tecnológica) menor o igual a 4, mientras que para una aplicación en entornos reales se requiere TRL mayores a 4 [14], [26]. Sin embargo, el desarrollo de estas plataformas y la investigación asociada ha permitido la paulatina incorporación a la industria de este tipo de sistemas robóticos, como se presenta en [9], [14], abarcando una gran variedad de aplicaciones como las resumidas en la Tabla 1.1.

Tabla 1.1. Aplicaciones y Comportamientos de Enjambres terrestres (basado en [14])

Aplicación	Proyecto	Comportamiento Enjambre
Agricultura	SwarmBot 3.0 [27] y Xaver [28]	• Movimiento coordinado • Distribución de tareas
Emergencia y Rescate	Guardians [29]	• Formación • Exploración colectiva • Movimiento coordinado
Organización de Depósitos (Warehouse)	Amazon (Kiva) [30], [31] y Alibaba [31]	• Movimiento coordinado • Distribución de tareas
Plantas Industriales	Swilt [32]	• Exploración colectiva • Regulación del tamaño del enjambre • Agrupamiento

1.4.3. ALGORITMOS DE EVASIÓN DE OBSTÁCULOS

Una vez revisadas, las principales características, comportamientos y las aplicaciones que debe tener un sistema robótico multi-agente de tipo enjambre, es necesario determinar las características de control que este sistema necesita, es así como, dentro de la robótica móvil y la investigación de esta, existen 4 retos principales o principales líneas de investigación para conseguir un desarrollo óptimo de las posibles aplicaciones con robots terrestres y su integración a sistemas multi-agente. Estos retos se resumen en la Figura 1.4.(a) [9].

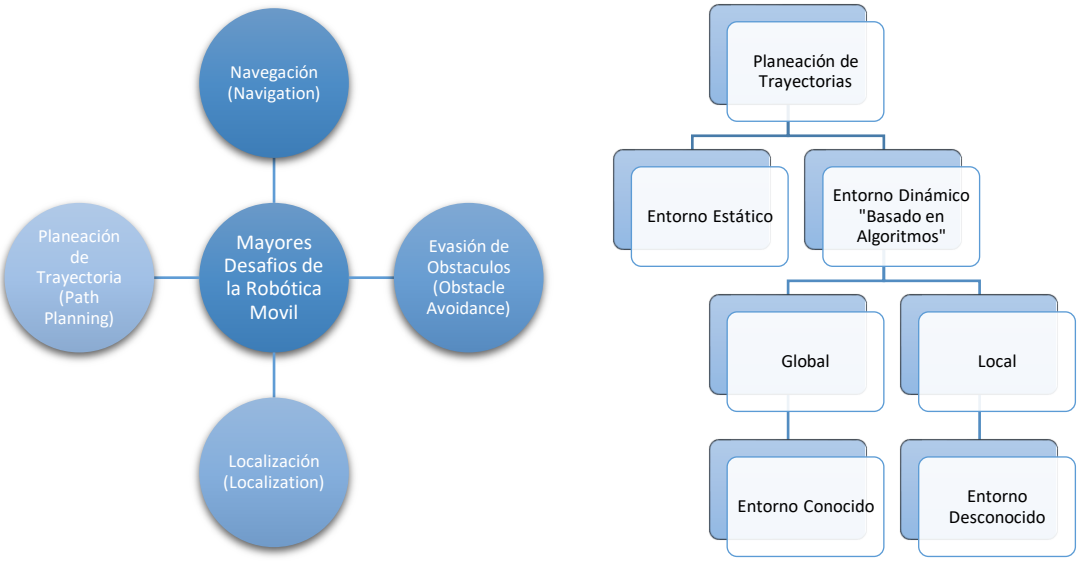


Figura 1.4.(a)Desafíos Robótica Móvil (adaptado de [9]), (b)Clasificación de los métodos de Path Planning (adaptado de [9]).

Dentro de la planeación de caminos o trayectorias (path planning) de robots móviles se puede encontrar distintos métodos en función del tipo de entorno en el cual el robot realice sus actividades. Teniendo siempre en cuenta que el principal objetivo es determinar una trayectoria óptima que lleve al robot desde un punto inicial a uno final dentro del entorno, bien sea previamente conocido o no. Esto a su vez, encaja con la necesidad de la evasión de obstáculos, de tal manera que, el robot pueda llegar a su destino sin ser obstruido por algún obstáculo o un evento de colisión en su trayectoria, especialmente dentro de entornos desconocidos o variables en el tiempo, estas características ubican a los algoritmos de evasión de obstáculos como una parte fundamental dentro de la planeación de trayectorias en entornos dinámicos y desconocidos, con un enfoque local, es decir limitado a los sensores del robot, véase Figura 1.4.(b) [9].

Considerando la aproximación de que la evasión de obstáculos o el algoritmo encargado de esta tarea debe afectar o modificar la trayectoria del robot solo si este interrumpe a la previamente planeada, surge el concepto de que este sea un algoritmo reactivo. Un algoritmo o sistema reactivo hace referencia a la capacidad de mantener interacción constante respecto a su entorno y poder responder a los cambios de este, es así, que debe ser capaz de emitir una acción casi inmediata al momento de recibir la señal de sus sensores o del estado del entorno, véase Figura 1.5. La reactividad permite la posibilidad de acciones rápidas y cruciales en tiempo-real, por lo que no es necesario utilizar reglas relativamente complejas [33], [34].

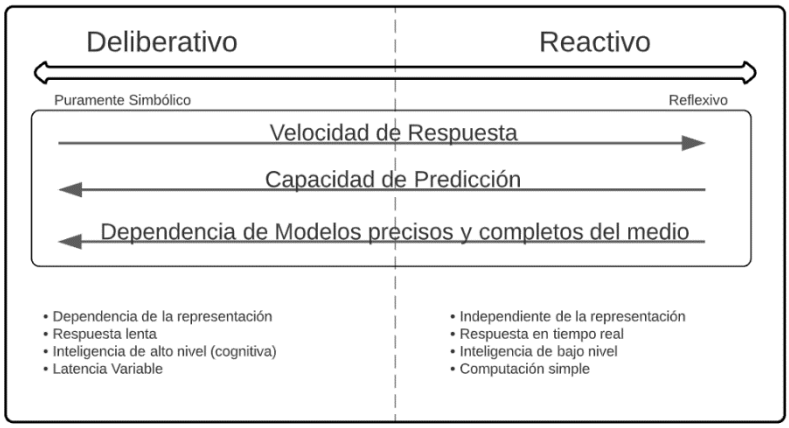


Figura 1.5.Espectro de control robótica móvil (Adaptado de [34]).

Teniendo en cuenta la importancia y las principales características con las debe contar un algoritmo de evasión de obstáculos se han propuesto diferentes resoluciones o aproximaciones que permitan resolver esta problemática entre los que se destacan:

Bug Algorithm: Este algoritmo se caracteriza por ser uno de los más simples, este permite al robot circunvalar el obstáculo y decide el punto más apropiado para continuar con la trayectoria previamente planeada [9], [35], [36]

Virtual Force Field (VFF) Algorithm: En este algoritmo se propone la utilización de una fuerza virtual perpendicular a la superficie del obstáculo, esto permite genera una fuerza de repulsión hacia la dirección opuesta del obstáculo y así evitar la colisión [9], [37].

Hybrid Navigation Algorithm (HNA o NHNA): Este algoritmo se caracteriza por utilizar lo lógica base del algoritmo bug (en la capa de acción reactiva), pero en el caso que la dirección objetivo sea menor de 90º, el algoritmo permitirá acortar la distancia al objetivo. Además, que cuenta con una comunicación asíncrona con la capa de planeación de trayectoria o deliberativa, lo que permite tomar direcciones optimas hacia el punto final de la trayectoria [38].

Vector Field Histogram (VFH): Este algoritmo surge como respuesta ante las limitantes encontradas por el algoritmo VFF, ya que ese algoritmo presenta oscilaciones ante caminos estrechos, es así como la idea del algoritmo VFH se basa en la construcción de un histograma en función de la lectura de los sensores presentes en el robot y así determinar la certeza o probabilidad de que exista un obstáculo en cada uno de los sectores angulares. Una vez determinados los sectores libres y ocupados, se selecciona la dirección deseada considerando además la dirección donde se encuentra el objetivo o trayectoria deseada [39], [40].

1.4.4. MATLAB + ROS

La expansión de la robótica en el ambiente industrial y de investigación en las últimas décadas ha requerido de múltiples plataformas para el desarrollo, prototipado y simulación de soluciones. En esta línea se creó ROS (Robot Operating Systems), una estructura flexible de código abierto para el desarrollo de software de robótica, que presenta un grupo de herramientas, librerías y convenciones, que buscan simplificar la creación de comportamientos robustos y complejos para una variedad de plataformas robóticas [41]. Esta herramienta (framework) no solo presenta herramientas de desarrollo de software, sino que presenta compatibilidad o comunicación directa a distintos entornos de simulación. El más conocido es Gazebo que permite emular las características físicas del sistema robótico incluyendo sensores y actuadores, así como el entorno en donde desarrolla sus actividades [42]. Para el desarrollo de este proyecto se utiliza la distribución ROS Noetic Ninjemys y Gazebo 11.

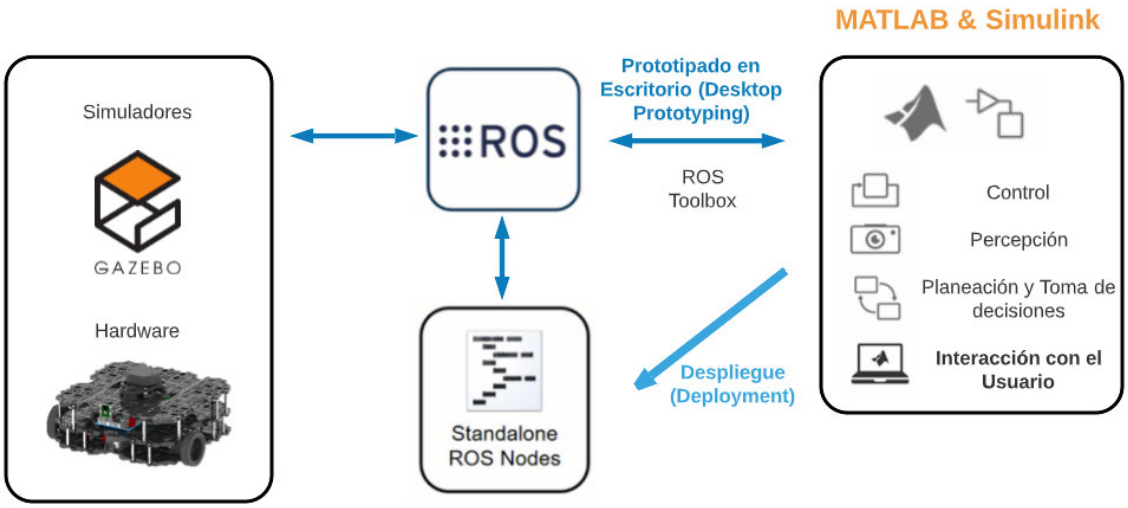


Figura 1.6. Diagrama de Integración MATLAB+ROS+GAZEBO [Fuente Propia]

Por otra parte, MathWorks® ha desarrollado la herramienta de ROS Toolbox [43] que permite establecer una interfaz online entre MATLAB®, Simulink® y el entorno de ROS como se muestra en Figura 1.6. Por lo que se puede incorporar las funciones y herramientas que presentan estos y otros toolboxs de MathWorks® asociados al desarrollo de la robótica. Estas características, sumadas a la integración de un simulador realista como lo es Gazebo, facilita el desarrollo de algoritmos para un sistema robótico virtual que por sus características requeriría de una inversión relativamente alta dependiendo el tipo de plataforma que se utilice y la cantidad de estos. Adicionalmente, por la naturaleza del sistema tipo enjambre, la probabilidad de colisiones en la fase de desarrollo es alta y realizar pruebas directamente con los robots físicos podría terminar en dejar fuera de

servicio a los agentes (robots). Por lo que la implementación o desarrollo puramente en un entorno virtual se presenta como una solución pragmática ante la falta de un banco de pruebas físico, así como de los agentes (robots). En este caso se utiliza la versión 2021A de MATLAB para Linux, por lo que todo el proyecto corre nativamente en Linux (Ubuntu 20.04).

2. METODOLOGÍA

Este capítulo presenta el desarrollo del entorno virtual, así como de la arquitectura de control del sistema enjambre y el algoritmo de evasión de obstáculos. Con este fin se plantea un enfoque mixto (cualitativo y cuantitativo) que permita la exploración, explicación, y la experimentación de distintos casos de comportamiento del sistema enjambre, enfocándose en el desempeño y caracterización del algoritmo de evasión de obstáculos.

2.1. DISEÑO ARQUITECTURA DE CONTROL

2.1.1. ARQUITECTURA DE CONTROL

Dentro de la robótica móvil, una arquitectura de control es la estructura conformada por hardware y software encargada de la adquisición y procesamiento de las señales o información sensorial del entorno, con el fin de analizar y controlar los actuadores del agente robótico [44]. Dentro de las posibles aproximaciones para la construcción de una arquitectura de control se puede destacar la descomposición del problema en pequeños subproblemas. De tal manera, que al resolver independientemente estos problemas se pueda llegar a la solución del problema global [44], [45].

En un sistema enjambre se puede destacar distintas tareas o retos que son necesarios resolver para que se tenga un comportamiento colectivo, estas tareas también son conocidas como niveles de competencia [45]:

- Ejecución de la tarea global (comportamiento enjambre)
- Planeación de trayectoria
- Seguimiento de trayectoria
- Evasión de Obstáculos

Considerando la aproximación de descomponer el problema, se plantea el uso de una arquitectura de subsunción similar a la planteada en [44]–[46]. En esta arquitectura se

organiza o distribuye cada una de las tareas en capas separadas que pueden ejecutarse de forma paralela, pero presentan ciertas interacciones entre ellas.

La arquitectura de subsunción o arquitectura basada en comportamientos planteada por Brooks [45] nace con el objetivo de la ejecución de comportamientos complejos sin la obligatoriedad del uso de controladores o sistemas complejos de IA [45]. Por lo que, se prioriza la simplicidad y robustez para sistemas robóticos autónomos o semiautónomos. Algunas de sus características o aspectos importantes que se deben tener en cuenta para la implementación de esta arquitectura son:

- No existe una modelación del entorno.
- El control no es centralizado
- El comportamiento puede ser modificado o mejorado añadiendo nuevos niveles de competencia [45].

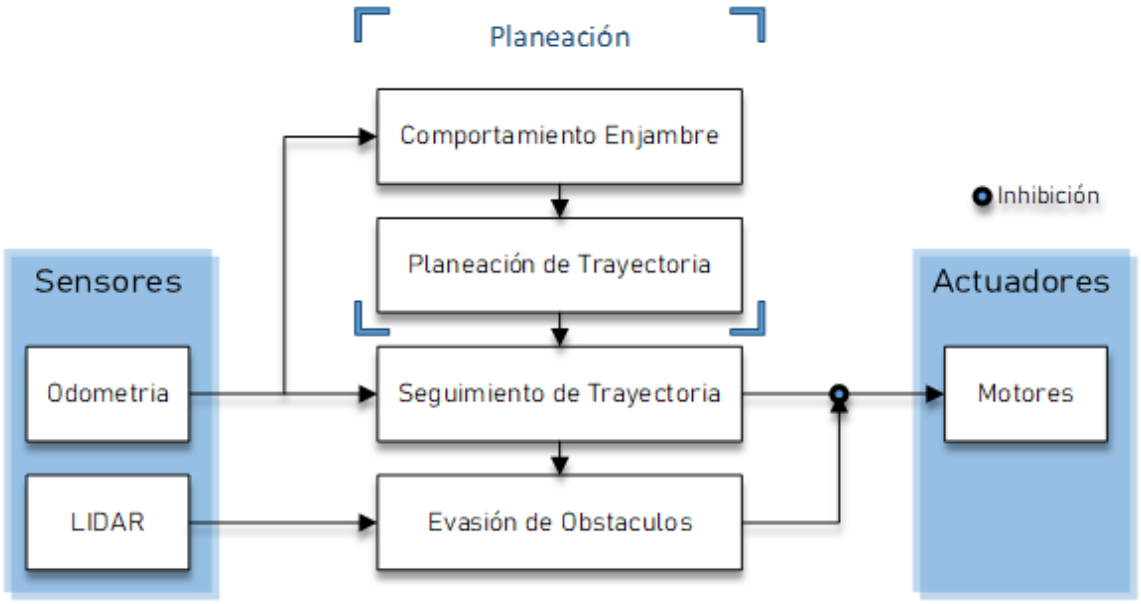


Figura 2.1. Arquitectura de control propuesta para el sistema Enjambre basada en una arquitectura de subsunción [Fuente Propia].

De esta manera, se plantea que para la solución del problema global o que permita el desarrollo del comportamiento enjambre es necesario resolver cada una de las capas o subproblemas presentados anteriormente en la Figura 2.1. Es importante destacar que la acción de control de la capa de Evasión de Obstáculos inhibirá a la del seguimiento de la trayectoria en el caso de que exista un posible caso de colisión, es decir reemplazará la acción de control.

2.1.2. ALGORITMO DE COMPORTAMIENTO ENJAMBRE

Dentro de esta capa de control o nivel de competencia se determina la tarea global del sistema enjambre. De tal manera que a partir de reglas locales (propias de cada agente) se consiga este comportamiento colectivo, entre los comportamientos que el sistema podrá ejecutar se encuentran los detallados previamente en la Sección 1.4.2.3, pero no se ven limitados a los mismos.

En el caso particular de este proyecto la tarea global es el intercambio de posiciones (Swap Positions). Por lo que, al conocer las posiciones iniciales de los agentes y las posibles formaciones de estos, mostradas en el Anexo IV, las trayectorias o reglas de cada uno de los agentes son predefinidas para la formación o prueba que se realice. Es así como, para la tarea específica de (Swap Positions), la separación entre los niveles de control del comportamiento enjambre y la planeación de trayectoria resulta difusa. Por lo que, se podría considerar como nivel de planeación el mostrado en la Figura 2.1, sin embargo, debido a la naturaleza del proyecto como banco de pruebas no debe estar limitado únicamente al cumplimiento de esta tarea. Por lo cual, se considera como 2 capas de control diferentes pero dependientes: comportamiento enjambre y planeación de trayectoria.

La selección del comportamiento, o la modificación de este, estará supeditado a la selección del entorno por parte del usuario, así que este control por parte del usuario estará limitado a un control a nivel de comportamiento, sin embargo, la interfaz permite la modificación de los comportamientos como se detalla en la Sección 3.1.

2.1.3. SEGUIMIENTO DE TRAYECTORIA

Una vez se determina la tarea global y la trayectoria o puntos de interés que deberá recorrer cada uno de los agentes, se debe ejecutar el seguimiento de esta trayectoria. Para poder realizar el seguimiento de la trayectoria se utiliza el algoritmo Pure Pursuit [47]–[49]. Este controlador se encarga del seguimiento de la trayectoria formada desde su posición actual hacia un punto de seguimiento (lookahead point), es decir a un punto a cierta distancia que forme parte de la trayectoria (formada por los puntos de interés o waypoints).

La aproximación Pure Pursuit trabaja calculando la curvatura a la cual se debe mover el agente, desde su posición actual hacia el punto objetivo. Por lo que, lo principal de este algoritmo es la selección del punto objetivo o de seguimiento que pertenezca a la trayectoria y tenga cierta distancia con la posición actual [49]. Esta lógica, que es una

analogía de seguimiento o persecución a un punto que forme la trayectoria, es la que da el nombre al algoritmo.

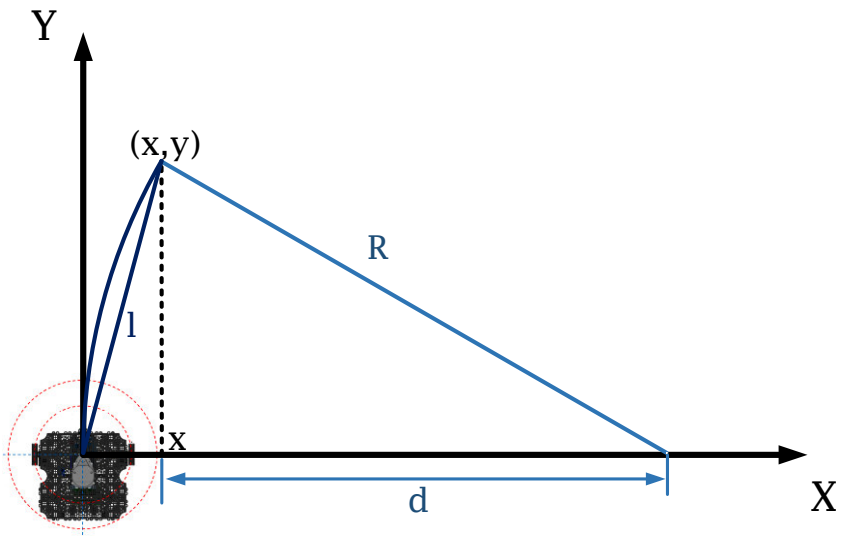


Figura 2.2. Análisis Geométrico del algoritmo Pure Pursuit [Fuente Propia].

A partir del análisis geométrico de la Figura 2.2, se determinan las siguientes ecuaciones:

$$x^2 + y^2 = l^2 \tag{2.1}$$

$$x + d = R \tag{2.2}$$

La Ecuación (2.1) es la de un círculo de radio l centrado en el origen, que describe la ubicación del punto de seguimiento (lookahead point), por lo que la distancia l es conocida como la distancia de seguimiento (lookahead distance). La Ecuación (2.2) describe la relación entre el radio del arco que une el origen y el punto objetivo y la separación del eje x respecto al punto objetivo. Por lo que esta ecuación establece que el radio del arco y la separación en x son independientes y solo dependen de d.

$$\begin{gathered} d = R - x \\ (R-x)^2 + y^2 = R^2 \\ R^2 - 2Rx + x^2 + y^2 = R^2 \\ 2Rx = l^2 \\ R = \frac{l^2}{2x} \quad \text{y} \quad \gamma = \frac{1}{R} = \frac{2x}{l^2} \end{gathered} \tag{2.3}$$

En las Ecuaciones (2.3) se detalla el desarrollo algebraico para la obtención del radio de curvatura deseado en función de las Ecuaciones (2.1) y (2.2). Por lo que se puede observar que la curvatura γ es además la inversa del radio de curvatura R detallado en la Sección 1.4.1.1.

Para el cálculo de la velocidad se considera una velocidad lineal $\boldsymbol{v}$ constante, la cual no puede exceder de 0.26 [m/s] (límite del TurtleBot Waffle Pi, ver el Anexo III). Mientras que la velocidad angular se determina con la Ecuación (2.4), la cual relaciona velocidad lineal y angular con el radio de curvatura.

$$\omega = \frac{v}{R} \tag{2.4}$$

El desarrollo o implementación del control de seguimiento de trayectoria (Path Following) se detalla en el pseudo-código mostrado en la Figura 2.3.

Algoritmo 1: Pure Pursuit

Datos: puntos de interés, posición actual $= [x, y, \theta]'$
Resultado: $v, w, TargetDir, CloseToGoal$
mientras *Distancia al punto de interés final* ≥ 0.5 **hacer**
    Determinar punto de seguimiento ; /* Distancia de 0.5[m] */
    Establecer Velocidad Lineal Deseada v ;
    Calcular TargetDir ; /* Dirección Objetivo */
    Calcular w
    $CloseToGoal \leftarrow 0$
fin
$CloseToGoal \leftarrow 1$; /* Se ha llegado al objetivo */
$v \leftarrow 0$
$w \leftarrow 0$

Figura 2.3. Pseudo-Código del Algoritmo Pure Pursuit [Fuente Propia].

2.1.4. EVASIÓN DE OBSTÁCULOS (OBSTACLE AVOIDANCE)

Para la última capa de control es necesario el desarrollo de un algoritmo dependiente del sensado o detección de los posibles eventos de colisión, sean estas colisiones con obstáculos fijos o inter-colisiones con el resto de los agentes del sistema enjambre.

Para el caso del TurtleBot se cuenta con un sensor LIDAR 360º (Light Detection and Ranging), detallado en el Anexo III, el cual permite detectar y obtener la distancia de obstáculos en el entorno del robot, a partir de la emisión-detección de un láser. En el caso del TurtleBot permite detectar y obtener la distancia en cada uno de los grados de separación angular, por lo que se tiene 360 datos (1 por grado de separación angular). La operación de este sensor se puede observar en la Figura 2.4, en donde cada uno de los datos obtenidos por el LIDAR es representado por un punto, obteniendo la gráfica de la derecha y una visualización 2D del entorno local del robot.

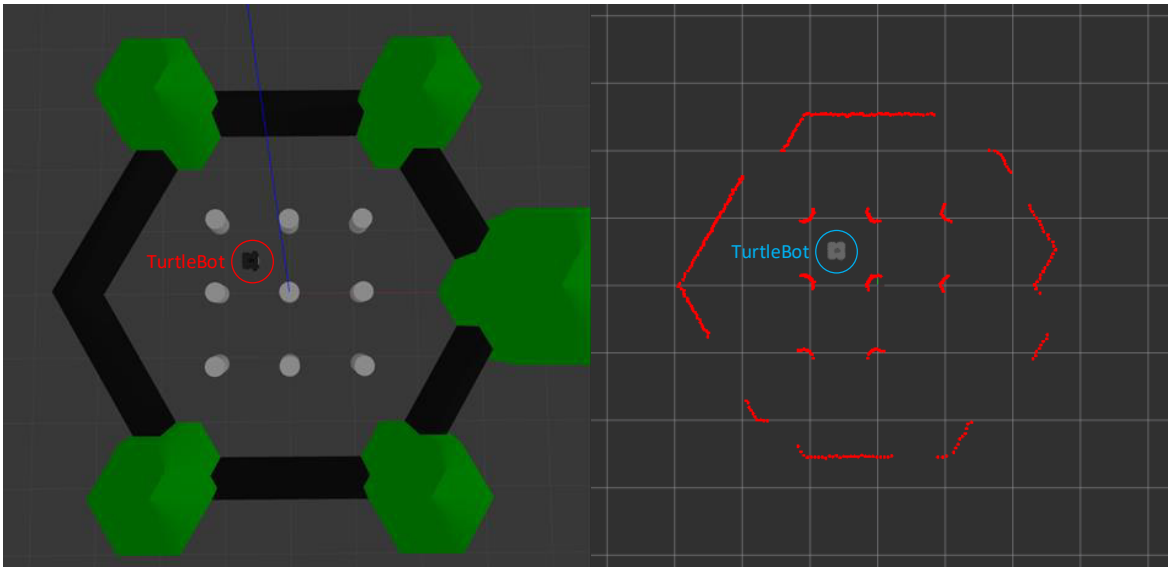


Figura 2.4. Visualización del LIDAR de un entorno de Gazebo por medio de RVIZ [Fuente Propia].

2.1.4.1. VFH+ (Vector Field Histogram)

Teniendo en cuenta las capacidades de detección de obstáculos que brinda el LIDAR 360º, surge la inquietud de cómo abordar el problema de evasión de obstáculos, previamente detallado en la Sección 1.4.3 donde se presentaron varias alternativas o aproximaciones a este reto.

La utilización del algoritmo VFH [39], [40], presenta varias ventajas o características que lo destacan respecto a los otros algoritmos previamente mencionados:

- Permite la detección de obstáculos desconocidos y evita la colisión, a la vez que, corrige la trayectoria de tal manera que se llegue a la trayectoria deseada una vez evadido el obstáculo [39], [40].
- El método VFH es ejecutado de manera local, por lo que no intenta determinar la trayectoria óptima. La determinación de una trayectoria óptima puede ser encontrada solo si se conoce todo el entorno [40].
- A diferencia de otros métodos, este permite la reducción de errores que se pueda generar debido al ruido de los sensores [40]. Esto permite que el algoritmo VFH sea computacionalmente eficiente y robusto. Además, que la respuesta de este es lo suficientemente rápida y confiable para su implementación en robots móviles.

En contraste con los algoritmos basados en VFF (detallado en la Sección 1.4.3, [9], [37]) presentan un solo paso para la reducción de datos, ya que determinan directamente la

dirección y magnitud de la fuerza repulsiva con los datos de los sensores (perdiendo información detallada de la distribución de obstáculos). El algoritmo VFH utiliza 2 etapas de reducción de datos, sin perder información detallada del entorno, la cual puede ser usada posteriormente para el mapeo del entorno [40].

Método VFH:

Como se mencionó previamente la presencia de 2 etapas para la reducción de datos, da como resultado 3 niveles o representaciones de los datos:

a) El nivel más alto de representación mantiene la descripción del entorno observado por el robot. En este nivel se genera un histograma cartesiano 2-D (Histogram Grid), véase la Figura 2.5, que es actualizado en tiempo real con la lectura de los sensores y a partir de este se determina el valor de certeza de que una celda esté activa [40]. En este histograma se considera a cada una de las celdas activas (active cells) como una representación del obstáculo dentro de la ventana del sector de análisis o ventana activa (Active Window). Se debe tener en cuenta que para cada medida del LIDAR se tendrá solo una celda activa, es decir 360 celdas, una por cada medición. Cada una de las celdas activas son asociadas a un vector dirección $\boldsymbol{\beta}$ con origen en el centro del robot, y coinciden con cada uno de los 360º angulares.

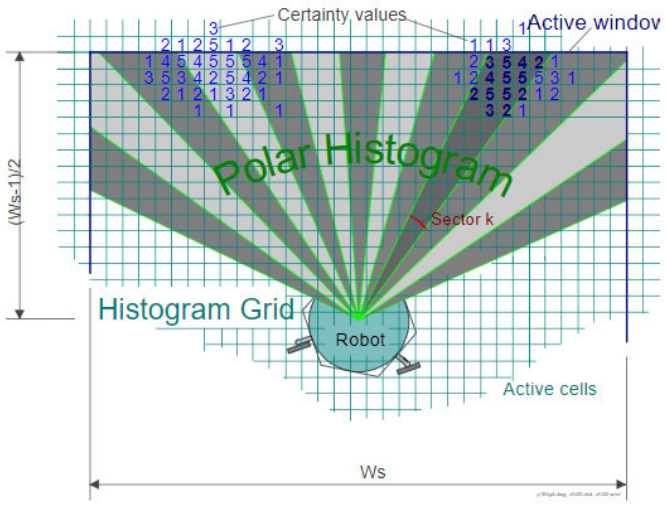


Figura 2.5. Visualización Histogram Grid – Polar Histogram [40]. En esta figura Polar Histogram: Histograma Polar, Histogram Grid: Histograma Cartesiano 2-D, Active Window: Ventana Activa, Active Cells: Celdas Activas, Certainty values : Valores de Certeza

El vector dirección $\boldsymbol{\beta}$ viene dado por:

$$\boldsymbol{\beta}_{i,j} = arctan\left(\frac{y_j - y_o}{x_i - x_o}\right) \quad (2.5)$$

Donde:

x_o, y_o: Coordenadas del centro del robot

x_i, y_j: Coordenadas de la celda activa $C_{i,j}$

Por otra parte, la magnitud $m_{i,j}$ de las celdas activas viene dada por:

$$m_{i,j} = c_{i,j}^2(a - bd_{i,j}^2) \quad (2.6)$$

Donde:

$c_{i,j}$: Valor de certeza de la celda activa $C_{i,j}$

$d_{i,j}$: Distancia desde la celda activa al centro del robot

Se observa que la magnitud de la celda activa es proporcional al cuadrado de la distancia, por lo que las celdas activas más cercanas al robot tendrán un mayor peso.

Para la obtención de las constantes a, b en la Ecuación (2.6) se considera como condición de borde que $m_{i,j}$ sea igual a $c_{i,j}^2$, con lo que a y b son escogidas de acuerdo con la Ecuación (2.7). Siendo w_s el ancho de la región de medición activa observado en la Figura 2.5.

$$a - b\left(\frac{w_s - 1}{2}\right)^2 = 1 \quad (2.7)$$

b) El siguiente nivel o representación de datos, es un histograma polar de 1-D (Polar Histogram), en este se determina la densidad polar de obstáculos (Polar Obstacle Density) en cada uno de los sectores de medición.

Una vez obtenidas las mediciones de todas las celdas activas por sector angular k, se determina la densidad polar del obstáculo por sector angular H_k^p, observada en el histograma polar.

$$H_k^p = \sum_{i,j} m_{i,j} \quad (2.8)$$

Este valor H_k^p forma el histograma visualizado en la Figura 2.6 y por medio de un valor umbral se compara en cada sector para determinar si es un sector libre o con obstáculos.

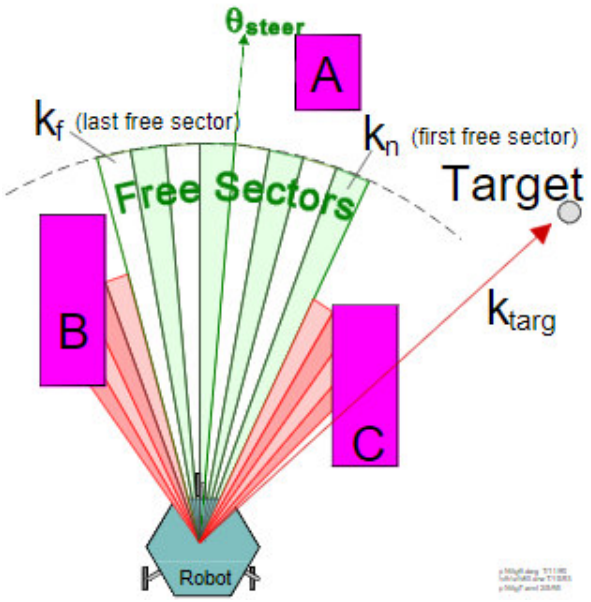


Figura 2.6. Selección de la dirección de giro en función de los sectores libres [40]. En esta figura Free Sectors: Sectores Libres, Target: Punto objetivo del robot

En la Figura 2.6:

θ_{steer}: Dirección de giro, seleccionado por el algoritmo VFH.

k_n , k_f, k_{targ} : primer sector libre, último sector libre, sector objetivo

c) Por último, la representación de la salida del algoritmo selecciona los sectores libres que permitan que el robot pase, se calcula la velocidad y dirección del vehículo a partir de este. Este análisis y solución se ve representado en la Figura 2.6.

Método VFH+:

El algoritmo de VFH+ [50], incorpora nuevas mejoras respecto al método VFH:

a) La incorporación de umbrales de histéresis en el mapa de densidad polar de obstáculos. Estos permiten suavizar la trayectoria planeada [50].

b) Considera el tamaño del robot, por lo que las restricciones no-holonómicas son tomadas en cuenta para la planeación de la trayectoria, además de considerar sectores que son bloqueados por la misma trayectoria del robot [50], [51].

c) Se aumenta el uso de funciones de costo, las cuales permiten cambiar el comportamiento y desempeño en función de estos parámetros. Las funciones de costo ponderan la dirección objetivo, la dirección actual y previa, con las cuales se puede tener un comportamiento, más o menos agresivo respecto al seguimiento de la trayectoria [50], [51].

Una explicación detallada de los niveles de representación de datos y su efecto en el algoritmo VFH+ se puede revisar en el Anexo V.

En el caso del algoritmo VFH+ para el cálculo de la nueva dirección de giro con las nuevas mejoras enumeradas previamente, se tienen 4 etapas de reducción de datos, las cuales son detalladas a continuación.

a) Histograma Polar Primario:

Dentro de esta primera etapa el proceso es casi idéntico respecto al algoritmo VFH, para la obtención del primer histograma polar, formando un mapa local del entorno inmediato del robot, una vez se realiza el análisis explicado en el algoritmo VFH para la lectura del LIDAR mostrado en la Figura 2.7, se obtiene la densidad polar de obstáculos (H_k^p), véase Figura 2.8.

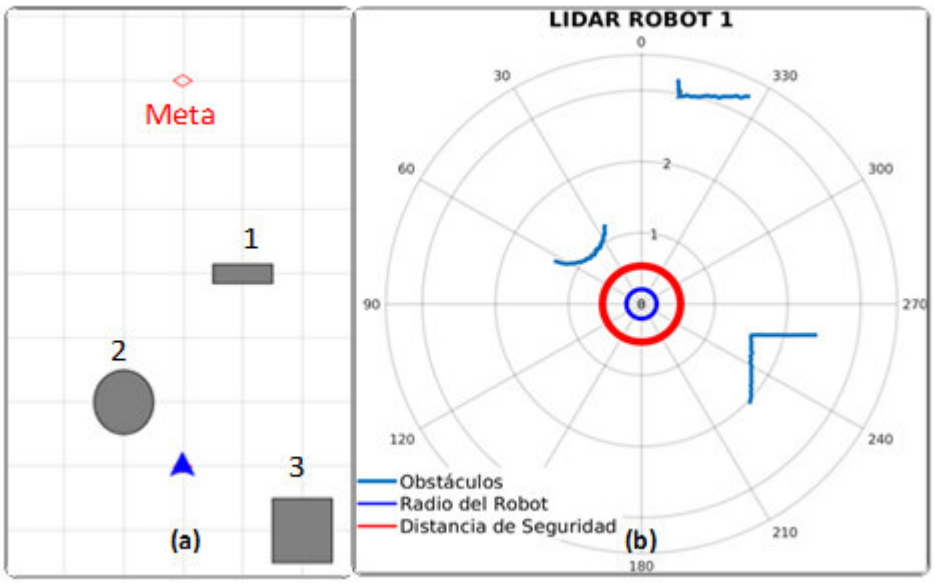


Figura 2.7.(a) Ejemplo del algoritmo VFH+, (b) Representación de la lectura del LIDAR [Fuente Propia].

Se puede observar en la Figura 2.7,que la distancia del obstáculo 1 respecto al robot es bastante significativa. Lo cual se verá representado como una menor magnitud de la densidad polar de obstáculos en comparación a los otros obstáculos, tal como se muestra en la Figura 2.8., alrededor de 340º.

b) Histograma Polar Binario:

De manera general, una trayectoria suave y sin oscilaciones es lo ideal, por lo cual. Para eliminar no solo el ruido de las mediciones sino para evitar cambios bruscos en el estado

de los sectores angulares (libre u ocupado) que podrían generar un comportamiento indeciso en el robot se utiliza 2 umbrales (τ_{high}, τ_{low}), mostrados en la Figura 2.8. Estos son usados como límites de histéresis [50], siguiendo las siguientes reglas:

$$\begin{aligned} H^b_{k,n} &= 1 \qquad si\ H^p_{k,n} > \tau_{high} \\ H^b_{k,n} &= 0 \qquad si\ H^p_{k,n} < \tau_{low} \\ H^b_{k,n} &= H^b_{k,n-1} \qquad en\ otro\ caso \end{aligned} \tag{2.9}$$

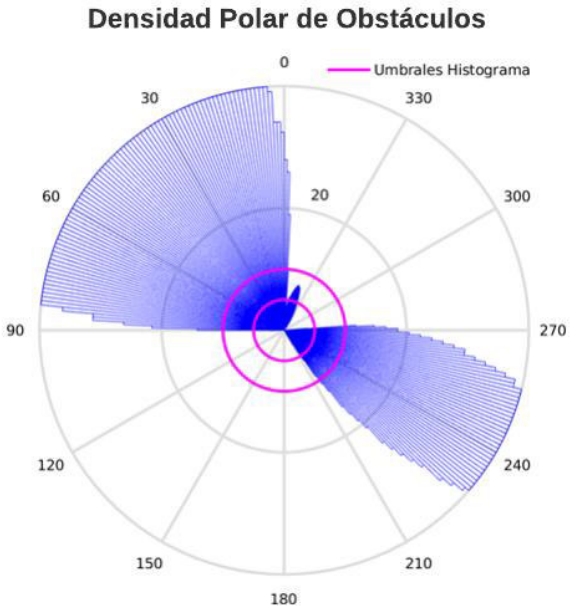


Figura 2.8.Ejemplo de Histograma Polar y la densidad polar de obstáculos [Fuente Propia].

c) Histograma Polar Enmascarado:

Otra mejora respecto al algoritmo VFH, es usar una aproximación simple de la dinámica de un robot móvil. Esta asume que la trayectoria del robot es basada en arcos circulares de curvatura constante y líneas rectas, como se muestra en la Figura 2.9 (a). La trayectoria de máxima curvatura se ve limitada o determinada en función a la velocidad lineal del robot, por lo que a mayor velocidad lineal el radio de giro mínimo aumenta. En el caso particular de los robots de tracción diferencial se puede tener un radio de giro mínimo igual a 0 solamente si la velocidad lineal es igual a 0.

Al considerar las posibles trayectorias del robot y su dependencia a la velocidad de este, es posible determinar qué sectores están bloqueados o no son accesibles por el robot [50]. En el caso de la Figura 2.9 (b), se tiene bloqueado el lado izquierdo superior por el

obstáculo 2 y además se tiene cierta velocidad lineal, por lo que todo el sector izquierdo es bloqueado, ya que no es posible dirigirse al mismo de forma inmediata.

Teniendo en cuenta esta restricción y el histograma binario previamente obtenido, se unen para determinar todas las direcciones de movimiento posibles a la velocidad actual del robot. Formando así el Histograma Polar Enmascarado (Masked Polar Histogram), mostrado en la Figura 2.10.

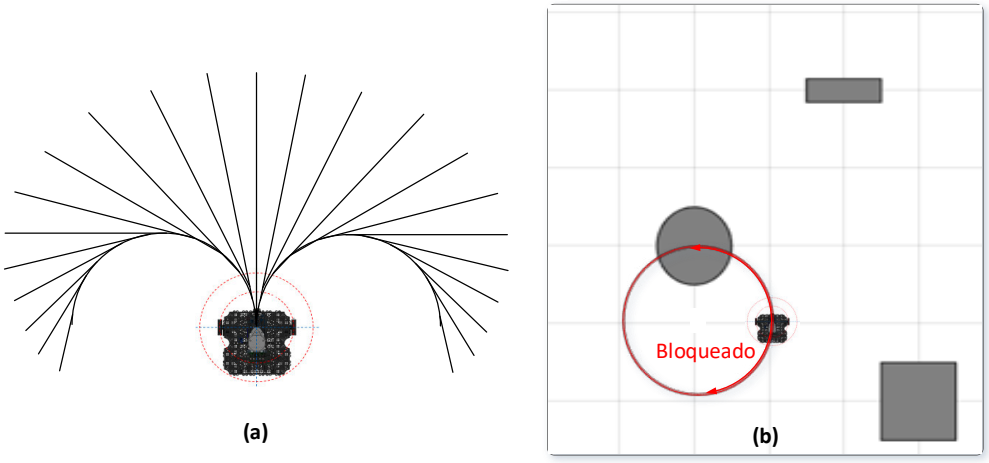


Figura 2.9. (a) Aproximación de las trayectorias considerando la dinámica del robot, (b) Limitación del movimiento según la dinámica del robot [Fuente Propia].

d) Selección de la Dirección de Giro:

Si bien en el histograma polar enmascarado se determinan todas las posibles direcciones que el robot podría tomar, algunas son mejores que otras respecto al objetivo final del agente, el llegar a la meta o trayectoria deseada. Para seleccionar la mejor de estas posibles direcciones, se utiliza la función de costo (g) descrita en la Ecuación (2.10). A partir de la determinación de este en cada una de las direcciones candidatas (c) se selecciona la dirección con un menor costo [50].

$$g(c) = \mu_1 . \Delta(c, k_t) + \mu_2 . \Delta\left(c, \frac{\theta_n}{\alpha}\right) + \mu_3 . \Delta(c, k_{d,n-1}) \quad (2.10)$$

Donde:

$\Delta(c_1, c_2)$: Es una función que calcula la diferencia angular absoluta entre los sectores c_1, c_2 , de tal manera que el resultado sea menor a la mitad del total de sectores angulares (s) en este caso 360, véase Ecuación (2.11).

$$\Delta(c_1, c_2) = \min\{|c_1 - c_2|, |c_1 - c_2 - s|, |c_1 - c_2 + s|\} \quad (2.11)$$

Dentro de la función de costo $g(c)$ descrita en la Ecuación (2.10), el primer término representa el costo asociado a la diferencia entre la dirección candidata (c) y la dirección objetivo (k_t), por lo que mientras mayor sea la diferencia, mayor será el costo y el robot tendrá una dirección más alejada de la dirección objetivo. En el segundo término se representa el costo asociado a la diferencia entre la dirección candidata (c) y la actual orientación del robot (θ_n) ,representada respecto a la resolución de los sectores angulares (α). En este caso mientras mayor sea la diferencia, mayor el cambio de la dirección de giro o nueva dirección de movimiento $(k_{d,n})$. Por último, el tercer término representa el costo asociado a la diferencia entre la dirección candidata (c) y la anterior dirección de giro $(k_{d,n-1})$. De manera general, el primer término se enfoca a la orientación del robot hacia su objetivo, mientras que los otros dos términos consolidan la dirección.

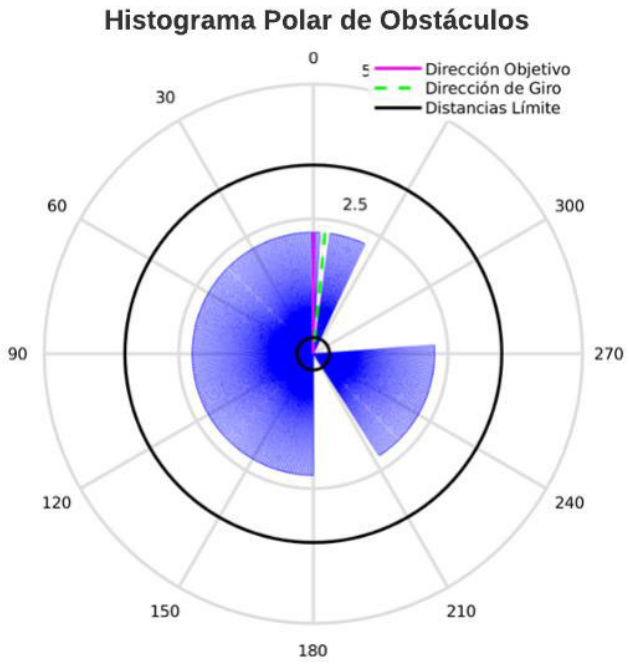


Figura 2.10. Histograma Polar Enmascarado con la dirección de Objetivo y de Giro [Fuente Propia].

La relación de los coeficientes μ_1, μ_2 y μ_3 determinará la suavidad de la trayectoria, así como los comandos de giro y la capacidad del robot de llegar a su meta [50]. Para garantizar el comportamiento de orientarse a la meta se debe cumplir la siguiente condición:

$$\mu_1 > \mu_2 + \mu_3 \tag{2.12}$$

En función de los experimentos desarrollados en [50] se determinó un buen desempeño con los parámetros $\mu_1 = 5$, $\mu_2 = 2$ y $\mu_3 = 2$.

El resultado de la selección de la dirección de giro en función del Histograma Polar enmascarado y de la dirección objetivo se observa en la Figura 2.10.

Una vez analizada la lógica y consideraciones para la aplicación del algoritmo VFH+ para la evasión de obstáculos, existen ciertos parámetros dependientes del tipo de aplicación, tipo de sensor, características físicas del robot, que determinan el desempeño del algoritmo.

Parámetros y Criterios de Sintonización:

- **Número de sectores angulares (NumAngularSectors):** Este parámetro indica el número de sectores en el cual se encontrarán divididos (Polar Obstacle Density), en este caso se encuentra dividido en 360, el cual es el mismo número de mediciones del LIDAR 360º.
- **Distancia Limite (DistanceLimits):** Especifica la distancia mínima y máxima de interés para considerar la evasión de obstáculos como se observa en la Figura 2.11. Se utiliza las distancias límite de [0.3 ,3.5] [m], 0.3 se considera una holgura mínima respecto al radio del robot de 0.22 [m], mientras que la lectura hasta 3.5 [m] permite la corrección suave de la trayectoria evitando así giros bruscos en el caso de obstáculos relativamente lejanos.
- **Radio del Robot (RobotRadius):** El radio asegura que los sectores que son considerados como camino libre sea mayor que el propio robot. Se utiliza el radio propio del TurtleBot3 Waffle PI, que es de 0.22 [m].
- **Distancia de Seguridad (SafetyDistance):** Es un aumento en la distancia que se considera segura para el robot. Se utiliza un rango de seguridad de 0.35 [m]. Este valor fue determinado al realizar pruebas ante inter-colisiones, véase la Sección 3.3.3, ya que, al ser todos los robots exactamente iguales, la detección de cada uno de ellos fue la detección del mismo sensor LIDAR ubicado en la parte superior de los robots, descartando el chasis, por lo que fue necesario aumentar la distancia segura de cada uno de los agentes.
- **Radio de Giro mínimo (MinTurningRadius):** Indica el radio de curvatura mínimo como se muestra en la Figura 2.11, se utiliza de 0.4, considerando aproximadamente 2 veces el radio del robot [m].

- **Umbrales del Histograma (HistogramTresholds):** Estos umbrales permiten el control por histéresis. Las zonas con valores de densidad mayores al umbral máximo se considera un espacio ocupado o 1, mientras que los sectores con densidad menor al umbral inferior son considerados espacio libre o 0. Estos umbrales y su aplicación se detallan en la Ecuación (2.9) ,los umbrales usados son [5 ,10] basado en el ejemplo de cálculo mostrado en [40] y verificado en las pruebas detalladas en la Sección 3.3.

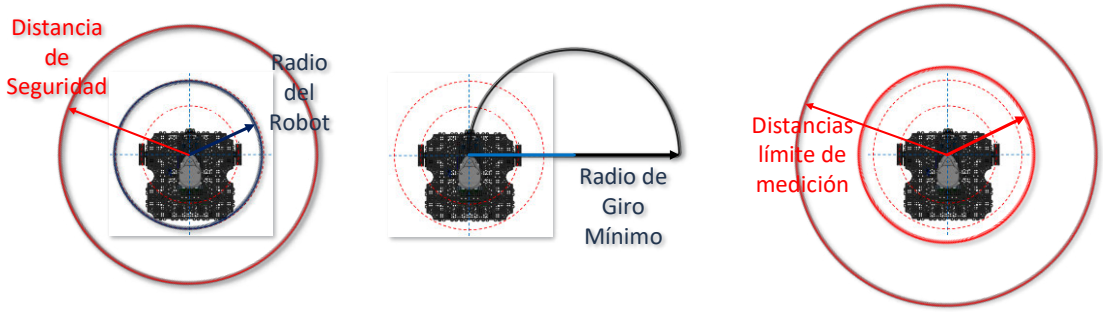


Figura 2.11.Parámetros de Sintonización algoritmo VFH+ [Fuente Propia].

2.2. ENTORNO VIRTUAL

Con el fin de cumplir los objetivos planteados en el proyecto, es vital contar con un software robusto que permita la emulación y simulación de la plataforma robótica, es así como, se parte con la utilización de ROS y su integración a Gazebo, los cuales corren nativamente en Linux. El proceso de instalación, así como la configuración base de estos programas se encuentran en el Anexo I.

2.2.1. ROS

Una vez se cuenta con el framework de ROS [41] y se han seguido los tutoriales iniciales especificados en Anexo I, es necesario crear un paquete (package). En el caso de este proyecto el package creado es el de multi_robot, donde se encontrarán alojados todos los archivos del proyecto en función de su tipo y extensión, por lo que la distribución de carpetas es acorde a lo especificado en el Anexo II.

2.2.2. GAZEBO

Gazebo es un simulador dinámico en 3D con la capacidad de simular de manera eficiente y exacta robots dentro de entornos complejos, así como la estructura y dinámica de estos.

Para lo cual es necesario generar archivos o código que describa estos entornos (archivos con extensión. world) [52]. Por otro lado, para la descripción de los entornos se utiliza el SDFormat (Simulation Description Format) o abreviado SDF. Este es un formato basado en xml que permite describir objetos y ambientes para simulación de robots, visualización y control del entorno. Con este formato es posible describir los aspectos de robots, objetos estáticos y dinámicos, la iluminación, terreno e incluso las interacciones físicas del entorno. Información detallada de la creación de entornos, la interacción con estos, así como documentación del formato SDF se encuentran en el Anexo III.

TurtleBot

Esta es la plataforma robótica estándar dentro de ROS [7], teniendo un enfoque principalmente en educación, investigación y prototipado de productos. Actualmente se encuentra en su tercera versión TurtleBot3, el mismo está enfocado en la reducción de tamaño y costo respecto a sus predecesores [7]. La selección de esta plataforma para el desarrollo de un sistema multi-robótico de tipo enjambre se basa principalmente en:

- Es la plataforma robótica de código abierto más popular para educación e investigación, por lo que su hardware, firmware y software son de libre acceso [7].
- Tiene un costo reducido dentro de los robots equipados con un sensor tipo LIDAR 360º.

Para poder usar esta plataforma robótica dentro de la simulación de Gazebo se requiere su modelo URDF (Universal Robot Description Format). Este modelo es un conjunto de archivos de tipo xml que caracterizan la estructura, así como la dinámica del robot, y la integración de sensores y actuadores, como en la Figura 2.12. Para la descarga e integración de este robot virtual en Gazebo por favor refiérase al Anexo III.

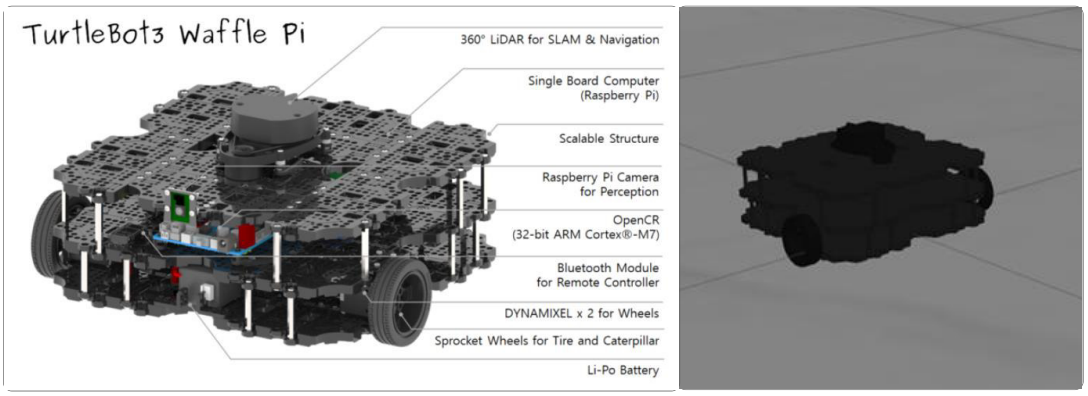


Figura 2.12. Plataforma Robótica TurtleBot3 Waffle Pi (Real-Virtual) [Fuente Propia].

Para la recepción y envíos de mensajes entre el modelo virtual en Gazebo y los nodos de ROS se tienen los siguientes tópicos:

Tabla 2.1. Tópicos usados en el control del robot virtual [Fuente Propia].

Tópicos	Lectura/Escritura	Tipo de Mensaje	Datos
robotn /odom	Lectura	nav_msg/Odometry	Posición y Orientación del robot
robotn/scan	Lectura	sensor_msgs/LaserScan	Medición Lidar
robotn/cmdvel	Escritura	geometry_msgs/Twist	Velocidad Linear y Angular

***n** indica el número del robot.

Una explicación más detallada acerca de los tipos de mensajes previamente mencionados, así como sus características pueden ser encontrados en la ROS Wiki referenciado en el Anexo I.

En el caso del tópico odom, los datos de posición (x, y) son obtenidos directamente, mientras que para la orientación del robot es necesario la conversión del cuaternión a ángulos de Euler, y de estos extraer el ángulo yaw correspondiente a θ en el modelo del robot [53]. Los cuaterniones, al igual que los números imaginarios, son una extensión de los números reales que facilitan las representaciones de rotaciones en el espacio y por ende abstraer rotaciones y traslaciones del modelo 3D (robot) con relativa facilidad y reducción en el costo computacional. Una explicación más detallada se encuentra en el Anexo III.

Para el tópico scan, es necesario extraer los datos de rangos y ángulos asociados, por lo que cada ángulo tendrá un valor del LIDAR asociado, en este caso se tiene 360 sectores angulares, por lo que se tiene 360 datos. Por último, para el envio de los comandos de velocidad en el tópico cmdvel se envía velocidad lineal asociada al eje x del modelo del robot y la velocidad angular asociada al eje z.

Obstáculos virtuales

Una parte fundamental para el desarrollo del banco de pruebas, son los distintos tipos de obstáculos que se encontrarán en el mismo, véase la Figura 2.13. Para la inserción de estos, se utilizan modelos preexistentes o creados a partir de xml, que son ubicados en el archivo ". world" correspondiente. Ejemplos de inserción de obstáculos se encuentra en el Anexo III. En el caso de este proyecto se utilizarán distintos tipos de obstáculos como

cubos, cilindros y paredes, los cuales formarán el ambiente estructurado como se muestra en la Figura 2.13.

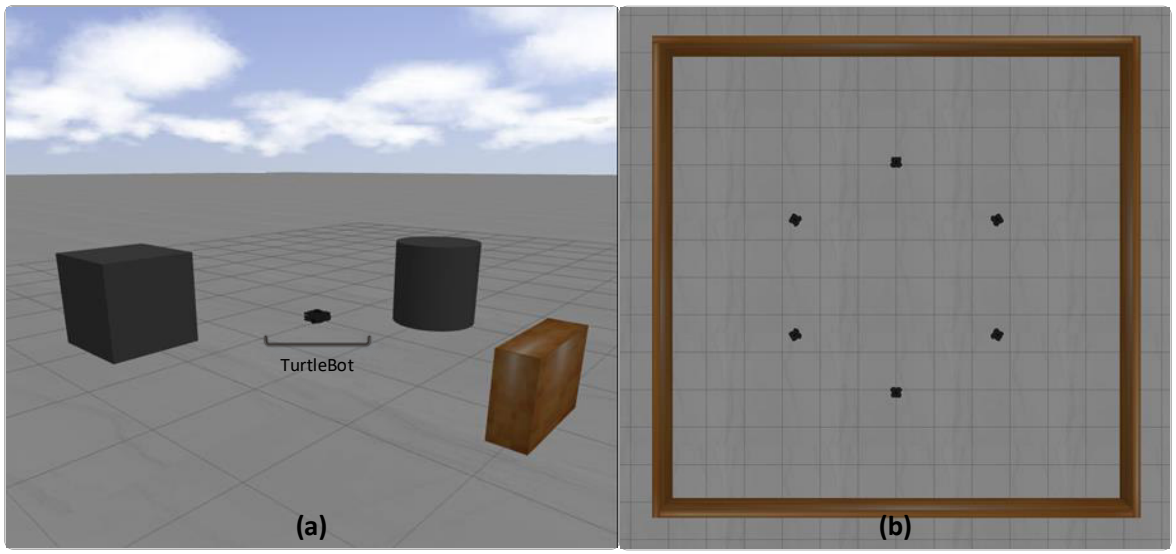


Figura 2.13. (a)Vista Isométrica de mundo desarrollado en Gazebo con obstáculos típicos, (b) Vista superior del entorno con múltiples robots [Fuente Propia].

Ejecución:

Una vez se tiene el diseño de los mundos, los robots con sus respectivos tópicos en las ubicaciones deseadas, se genera los ejecutables o ". launch" que permitirán la ejecución y despliegue de todos los componentes descritos previamente. En el caso de los robots se requiere hacer el proceso de "Spawn". Este permite cargar el modelo URDF en el mundo previamente desarrollado, así como asociar los tópicos correspondientes al mismo, además de determinar la posición y orientación inicial de cada uno de los agentes o robots que se encuentren en el entorno virtual. La implementación del proceso de "Spawn" se detalla en el Anexo III.

La ejecución o despliegue del entorno de Gazebo y de los robots virtuales se lo realiza con la ejecución de los archivos *. launch como se indica en la Figura 2.14.

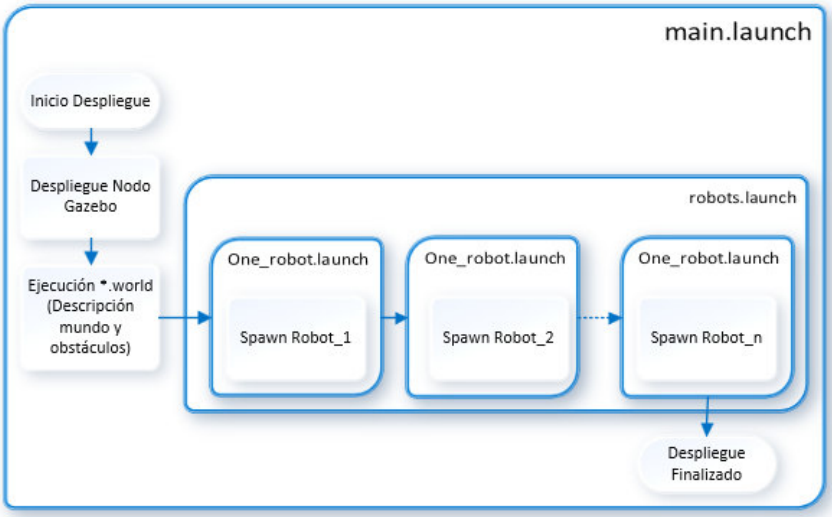


Figura 2.14. Diagrama de ejecución de los archivos ejecutables para el despliegue del banco de pruebas [Fuente Propia].

2.2.3. MATLAB+ SIMULINK

Comunicación con ROS

Los modelos de Simulink® (archivos .slx) con ROS Toolbox permiten la conexión directa de la red de ROS ya sea en el caso de una máquina remota, usando una máquina virtual o en el caso de este proyecto completamente en Linux. La conexión con el entorno de ROS se realiza en la inicialización del modelo de Simulink® cada vez que se ejecuta el mismo. La configuración de la dirección de la red de ROS se puede acceder en el modelo de Simulink® en: **Simulation>ROS Toolbox>ROS Network.**

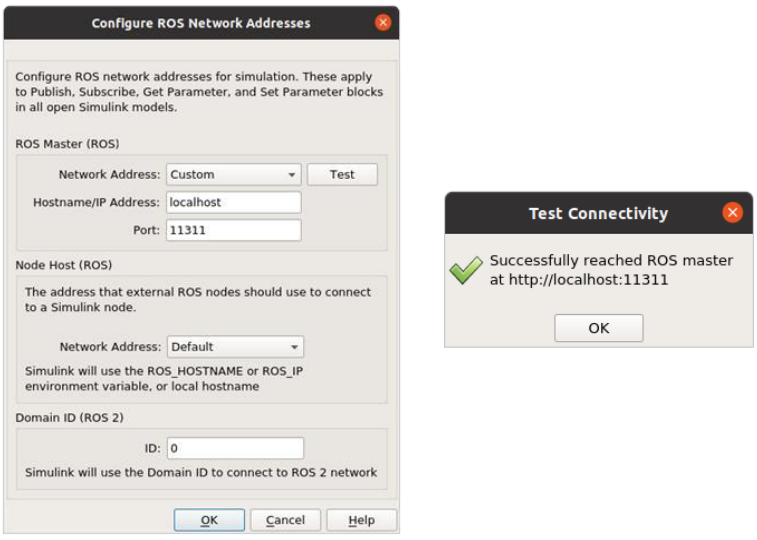


Figura 2.15. Configuración y prueba de conectividad a la red ROS [Fuente Propia].

Es importante la verificación de la conexión de MATLAB® y especialmente Simulink® con la red de ROS, como se muestra en la Figura 2.15. ya que con esta se puede conocer los tópicos disponibles para la publicación y subscripción de estos dentro del entorno de desarrollo de Simulink®. Por otra parte, cada modelo de Simulink® está asociado a un nodo único en ROS, este nodo es creado al inicio de cada simulación y eliminado al finalizar la misma. El nombre del nodo creado es <nombre_del_modelo>_<#random> como se muestra en la Figura 2.16. La adición de este número aleatorio permite evitar conflictos en la generación de los nodos.

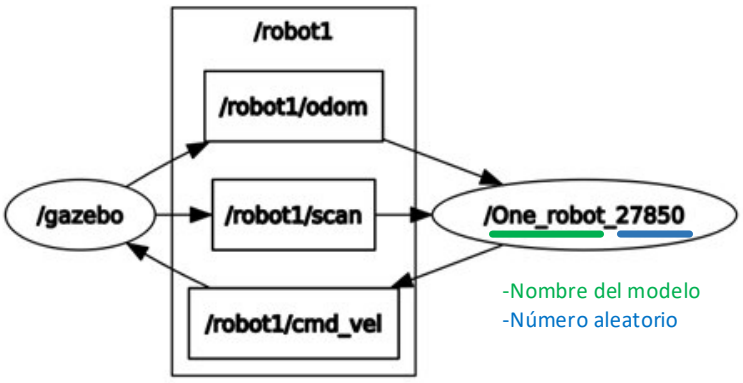


Figura 2.16. Verificación de conexión y creación del nodo de Simulink® en el entorno de ROS [Fuente Propia].

Configuración de mensajes

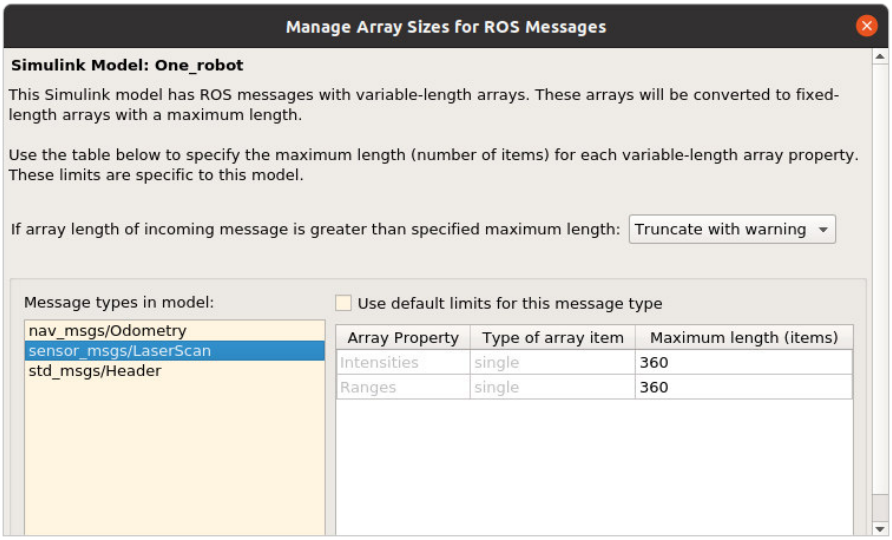


Figura 2.17. Configuración de mensajes en Simulink® [Fuente Propia].

Dependiendo los tipos de mensajes que se tengan como subscriptores (lectura de datos), es necesario configurarlos para que acepten el tamaño deseado, en este caso se modificó el tipo sensor_msgs/LaserScan a 360 para obtener la lectura de los 360º como se observa en la Figura 2.17. Otros mensajes que deberían ser modificados es la lectura de la cámara en el caso de que sea necesario su uso. Esta ventana se puede acceder en el entorno de Simulink® en **Simulation>ROS Toolbox>VariableSizeMessages.**

2.3. IMPLEMENTACIÓN DE LA ARQUITECTURA DE CONTROL

Para la implementación de la arquitectura de control previamente propuesta, se utiliza una estructura escalable como se muestra en la Figura 2.18. Teniendo el mismo esquema de control para cada uno de los agentes que conforman el sistema enjambre, modificando solamente el número de robot y los tópicos de lectura y escritura para su conexión son su homónimo en Gazebo.

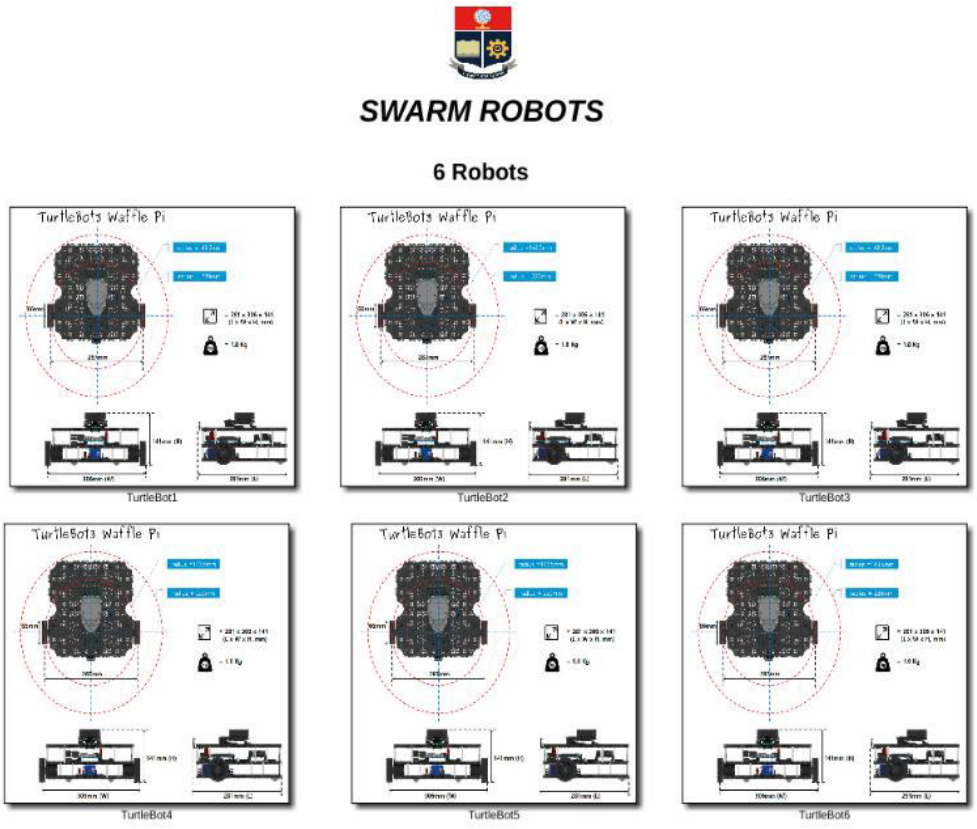


Figura 2.18. Implementación de Simulink® para 6 Agentes [Fuente Propia]

Dentro de cada uno de los bloques de control para los robots se tiene el esquema mostrado en la Figura 2.19 y detallado en el Anexo VI. Este se conforma por 4 subsistemas que cumplen las siguientes tareas:

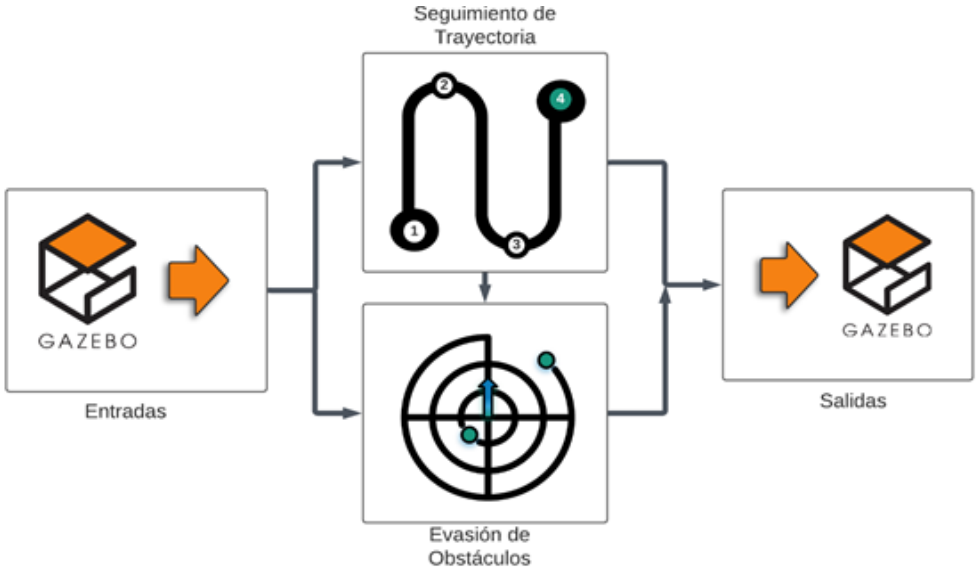


Figura 2.19.Esquema de Control en Simulink [Fuente Propia].

Entradas/Salidas:

Estos subsistemas se enfocan en la lectura y escritura de los tópicos correspondientes, explicados en la Sección 2.2.2, así como su interpretación y procesamiento para poder ser usado por Si otros subsistemas.

Dentro del subsistema enfocado a la lectura de los tópicos, es necesario crear los subscriptores necesarios para los tópicos de scan y odom del robot explicados en la Sección 2.2.2. y mostrados en la Figura 2 del Anexo VI. Estos bloques pueden ser encontrados en la librería propia de ROS Toolbox de Simulink®. Mientras que para la escritura en el tópico cmdvel se genera un mensaje tipo Twist y se asocia la velocidad lineal y angular, tal como se describe en la Sección 2.2.2. y se muestra en la Figura 2 del Anexo VI.

Seguimiento de Trayectoria:

Dentro del subsistema de seguimiento de trayectoria o path following se aplica el algoritmo de control pure pursuit, descrito en la Sección 2.1.3, así como la detección de llegada a la meta o fin de trayectoria.

Evasión de Obstáculos:

En este subsistema se implementa el algoritmo VFH+ detallado en la Sección 2.1.4.1.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En esta sección se presenta inicialmente un breve resumen de la interfaz gráfica desarrollada y la sintonización del algoritmo de seguimiento de trayectoria. Posteriormente se detallan los resultados de las pruebas realizadas para la verificación del desempeño del sistema enjambre y la arquitectura de control planteada. Finalmente se exponen las conclusiones y recomendaciones del trabajo realizado.

3.1. INTERFAZ DE EJECUCIÓN DEL SISTEMA ENJAMBRE

Con el objetivo de mejorar la comprensión del usuario acerca del comportamiento enjambre y el sistema robótico en general, además del envío de comandos e interacción con el mismo, se desarrolla una interfaz gráfica. Esta permite la selección de los mundos generados en Gazebo y los archivos de MATLAB y Simulink® asociados a cada uno de los mundos. La interfaz gráfica se observa en la Figura 3.1 y su funcionamiento/ejecución es explicado en el Anexo VII.

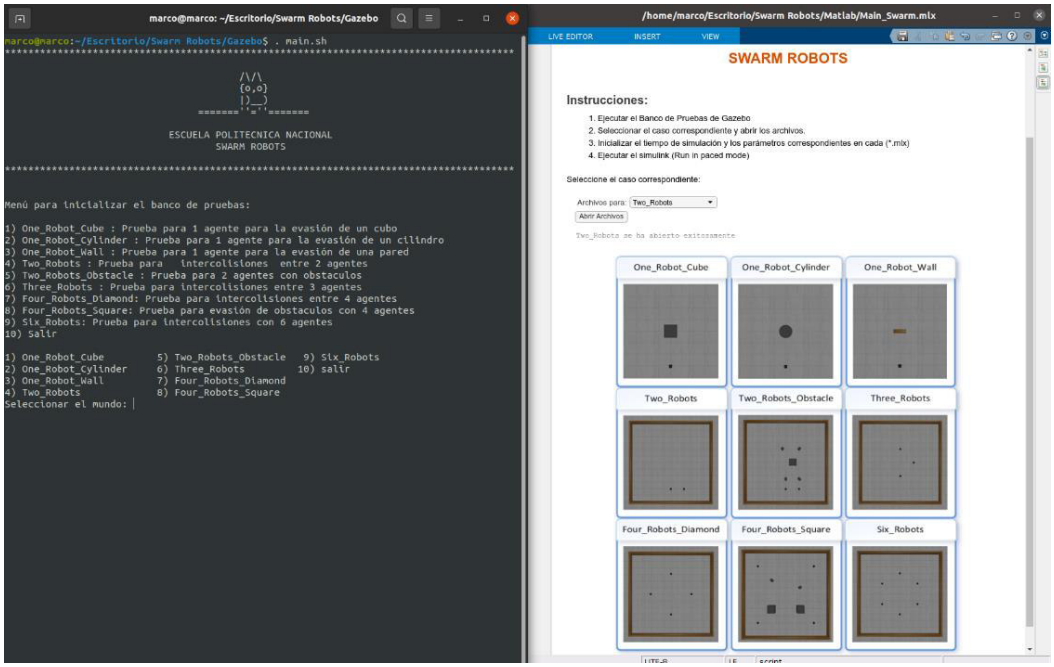


Figura 3.1. Interfaz Gráfica Swarm Robots [Fuente Propia].

3.2. SINTONIZACIÓN ALGORITMO PURE PURSUIT

Con el fin de implementar el algoritmo de seguimiento de trayectoria (Pure Pursuit), explicado en la Sección 2.1.3, es necesario realizar su calibración y verificar su comportamiento, independientemente de las otras capas de control, detalladas en la arquitectura propuesta en la Sección 2.1.1.

El punto de anticipación l, explicado en la Sección 2.1.3, es el parámetro de sintonización o calibración que permite que el robot siga la trayectoria de manera fluida con una velocidad lineal constante como se observa en la Figura 3.2.

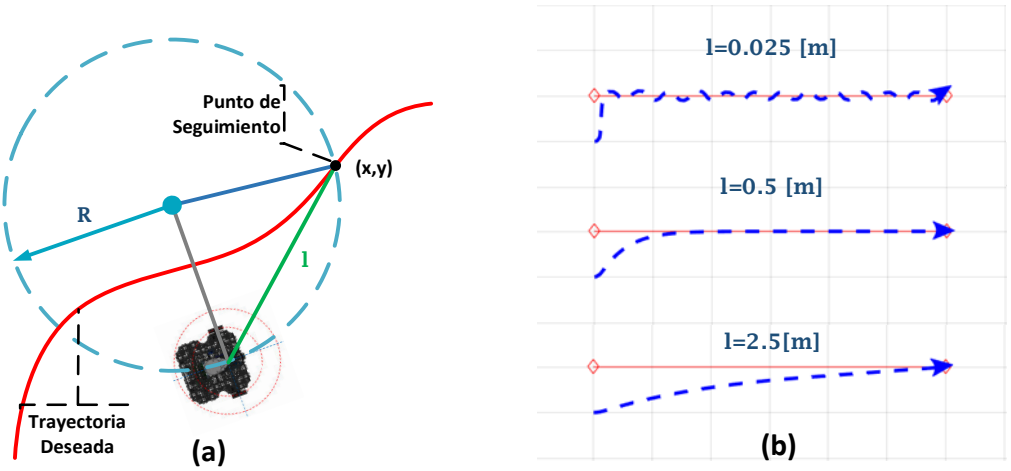


Figura 3.2.(a) Explicación geométrica del algoritmo Pure Pursuit, (b) Comparación distancia de seguimiento algoritmo Pure Pursuit [Fuente Propia].

Para determinar el valor de l , se comparó el desempeño del algoritmo para varios valores de la distancia de seguimiento, como se observa en la Figura 3.2.(b). En esta se evidencia que para valores pequeños se tienen oscilaciones, mientras que para valores relativamente grandes se tiene una trayectoria demasiada suave y por ende el robot demora en converger al punto deseado. Se puede observar además que el algoritmo permite el seguimiento en cualquiera de los casos y la variación del parámetro funciona como un factor de amortiguamiento, de manera similar a un sistema dinámico de segundo orden [47]–[49]. Por lo que se determina que el valor de l en este caso particular es 0.5 [m]. Esta distancia demostró un buen desempeño para el seguimiento de trayectorias como se observa en la Figura 3.2.(b), sin embargo, no se ve limitado al mismo.

3.3. RESULTADOS

Aquí se presentan las diferentes pruebas realizadas para comprobar el desempeño del sistema, así como para determinar las capacidades y limitaciones de cada uno de los agentes y de su comportamiento colectivo. Las pruebas realizadas están enfocadas en:

- Análisis del comportamiento individual de un agente ante distintas situaciones de colisión (tipos de obstáculos).
- Análisis del comportamiento individual y colectivo ante inter-colisiones y colisiones múltiples.

- Verificación de la integridad de los agentes en un entorno altamente dinámico (alta probabilidad de colisiones) ya sea con el entorno o con otros robots pertenecientes al enjambre.
- Verificación del comportamiento colectivo del sistema enjambre, manteniendo las capacidades individuales de planeación, navegación y evasión de obstáculos.

Ejemplos de ejecución de cada una de las pruebas son mostradas en el Anexo VIII.

3.3.1. PRUEBAS CON 1 AGENTE

Estas pruebas tienen como objetivo el análisis y validación de la arquitectura de control a nivel de uno solo de los agentes, especialmente en la capa de control reactiva (evasión de obstáculos). En esta prueba se verifica y analiza la respuesta de un solo agente ante distintos tipos de obstáculos, en particular se utilizan los siguientes tres:

- un cubo de 1x1x1 [m]
- un cilindro de 0.5 [m] de radio y 1 [m] de altura.
- una pared de 1 [m] de ancho y 0.3 [m] de profundidad.

En la Figura 3.3. se detalla la trayectoria planeada y el efecto que tiene un cubo como obstáculo en la trayectoria real del robot. En la parte inicial (a), se puede observar que al reconocer inicialmente el obstáculo tiende a girar a la izquierda, sin embargo, al alejarse demasiado de la trayectoria planeada, además considerando la existencia de espacio libre en el lado derecho, corrige la trayectoria como se muestra en (b). Es importante destacar que no existe una predisposición a ninguno de los lados (izquierda o derecha) respecto a una situación simétrica como la de las pruebas, sino que la dirección de giro es determinada por la función de costo detallada en la Sección 2.1.4.1.

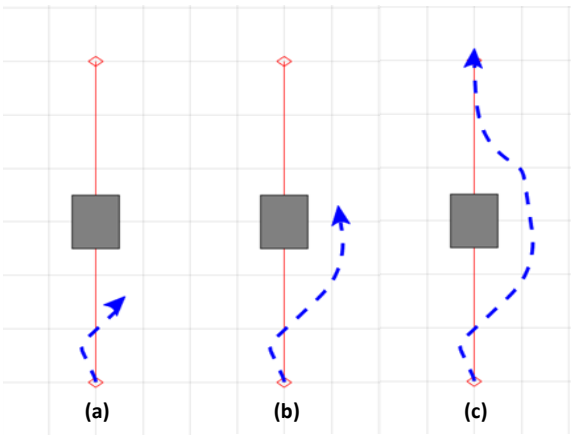


Figura 3.3. Trayectoria del Robot para la evasión del cubo [Fuente Propia].

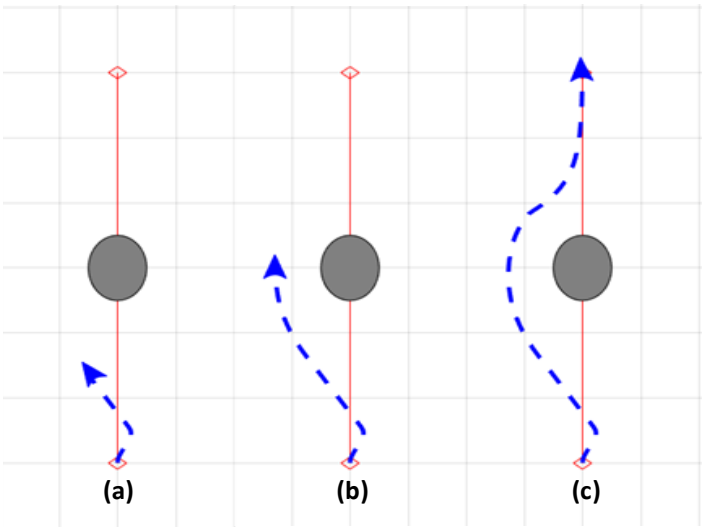


Figura 3.4.Trayectoria del Robot para la evasión del cilindro [Fuente Propia].

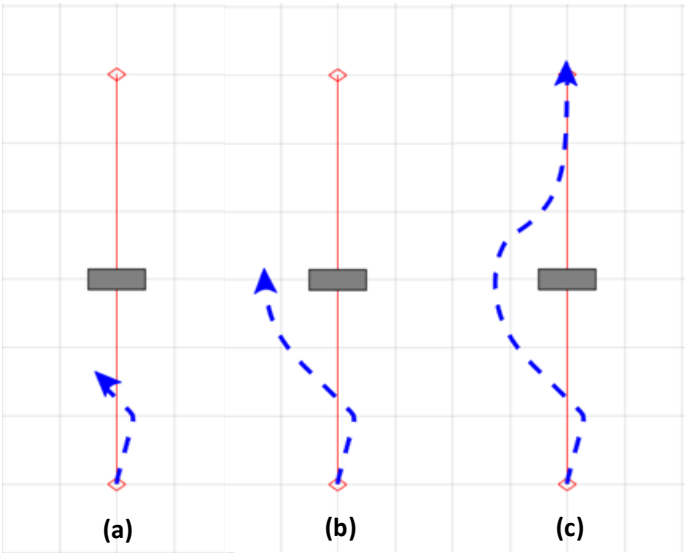
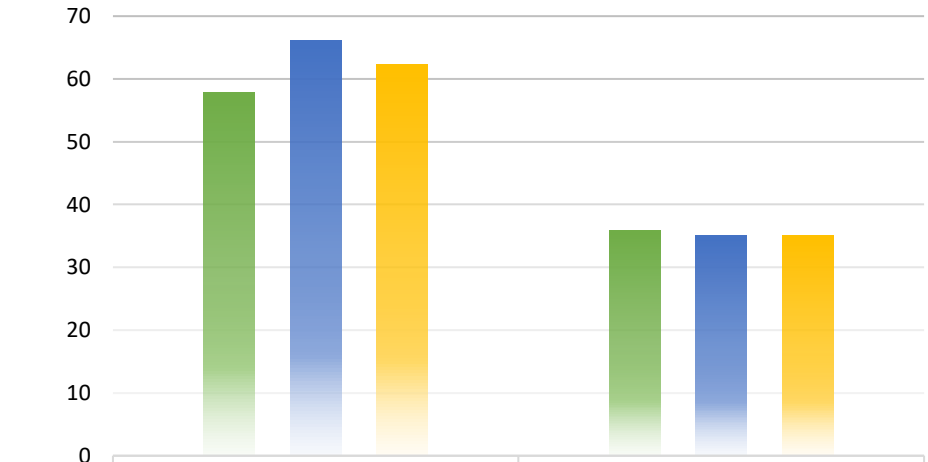


Figura 3.5. Trayectoria del Robot para la evasión de pared [Fuente Propia].

De manera similar las pruebas realizadas para los obstáculos cilíndrico y tipo pared (véase Figura 3.4 y Figura 3.5) tienen un comportamiento similar al mostrado al tipo cubo, con ligeras diferencias respecto a la trayectoria y por ende en el tiempo total de trayecto. Se observa en la Figura 3.6, que el tiempo de trayecto es mayor mientras mayor es el área del obstáculo, como es el caso de comparar el cilindro y el cubo.

Por otra parte, se observa una respuesta más suave o “redondeada”, en función de la curvatura del obstáculo y por ende una mayor distancia de separación respecto al obstáculo. Por lo que se concluye que la arquitectura planteada y por ende cada una de sus capas de control cumple con brindar la autonomía básica necesaria para un agente, ejecutando la planeación, seguimiento de trayectoria y la evasión de obstáculos.



	Distancia Mínima [cm]	Tiempo de Trayecto [s]
Cubo	57,88	35,94
Cilindro	66,13	35,04
Pared	62,32	35,07

Figura 3.6. Comparación desempeño para los distintos tipos de obstáculos [Fuente Propia].

Además, en la evasión del obstáculo para el caso de la Figura 3.5.(a). Se corrobora que el al detectar el obstáculo, como se muestra en la Figura 3.7, se ejecuta una corrección relativamente suave de la trayectoria para evitar giros bruscos una vez se tengan colisiones inminentes con el obstáculo.

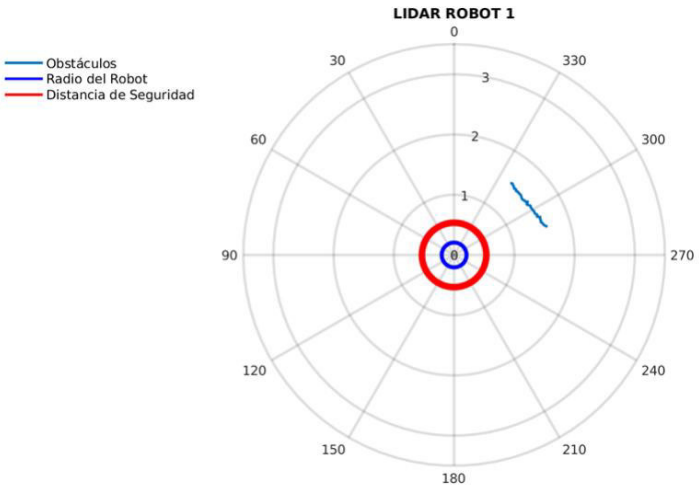


Figura 3.7. Lectura LIDAR para la detección del obstáculo (pared) [Fuente Propia].

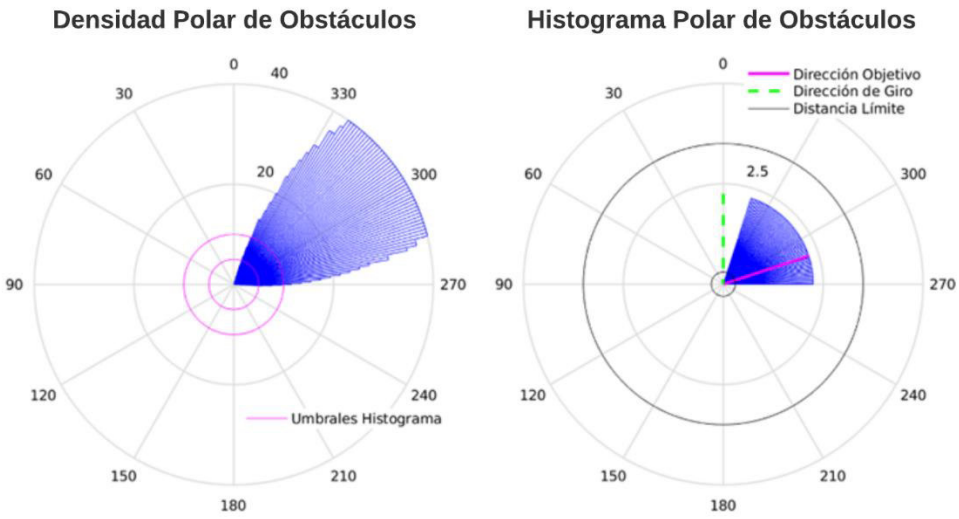


Figura 3.8. Ejemplo de ejecución algoritmo VFH+ [Fuente Propia].

Así mismo, al analizar la ejecución del algoritmo VFH+ para el mismo instante de tiempo de la Figura 3.5.(a), se verifica en la

Figura 3.8. que la sintonización y criterios tomados para la ejecución de los algoritmos pure pursuit y VFH+ permiten la corrección de la trayectoria a un sector libre. Para lo cual, como se indicó anteriormente, se está considerando no solo la dirección objetivo del robot, sino las características físicas y dinámicas del robot, explicadas previamente en la Sección 2.1.4.1. Además, en la

Figura 3.8., se observan el proceso de reducción de datos que permiten la obtención de la dirección de giro.

3.3.2. PRUEBAS CON 2 AGENTES

Estas pruebas tienen como objetivo el análisis y validación de la arquitectura de control a nivel de sistema, especialmente en la evasión de inter-colisiones. En esta prueba se verifica y analiza la respuesta de dos agentes realizando la tarea de intercambio de posiciones.

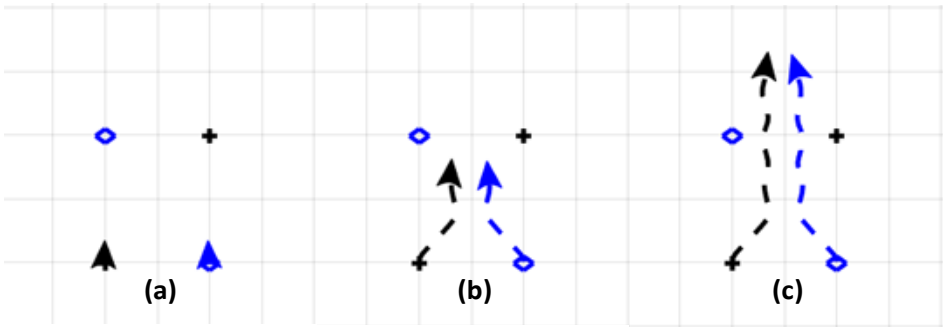


Figura 3.9. Trayectoria de los 2 agentes (simetría) [Fuente Propia].

Una vez teniendo la arquitectura funcional para uno de los agentes, se realiza el escalamiento a 2 agentes. En la Figura 3.9 se puede observar la primera prueba realizada con 2 agentes, como se muestra en la misma debido a que ambos agentes son prácticamente clones uno del otro, las potenciales inter-colisiones resultan en acciones de control casi idénticas. Lo que da como resultado la oscilación que se observa, sin que ninguno de los 2 agentes pueda llegar a su objetivo.

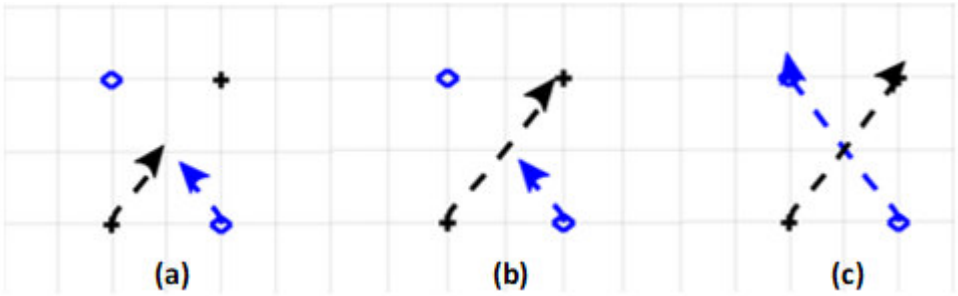


Figura 3.10. Trayectoria de los 2 agentes (asimetría) [Fuente Propia].

Con el fin de solucionar este inconveniente de simetría se plantea añadir cierta asimetría con la que los agentes resuelvan el problema. Por lo cual se plantea que en el caso de los robots asociados a un # impar, en este caso (azul 1), se detenga si la distancia entre los agentes está bajo un umbral en este caso (1 [m]), con esta lógica implementada se obtienen el resultado de la Figura 3.10.

Una vez realizado esto, en la prueba de la Figura 3.11, se observa un comportamiento fluido y dinámico de los agentes para la evasión de cada uno de los obstáculos con el fin de llegar a la meta respectiva de cada agente. En la Figura 3.11.(c), se muestra una ligera desviación de los agentes, lo cual cambia la postura final de estos. Este cambio de orientación se debe exclusivamente a la existencia de la pared, la cual es obviamente considerada como un obstáculo próximo. Este comportamiento destaca que una vez que el agente llega a su objetivo, este se detiene sin importar su orientación. Esta característica,

hace notar que el algoritmo VFH puede ser afectado por la presencia de obstáculos masivamente superiores al tamaño del robot, por lo que la trayectoria puede ser afectada al encontrarse cerca de las paredes que limitan el entorno o banco de pruebas.

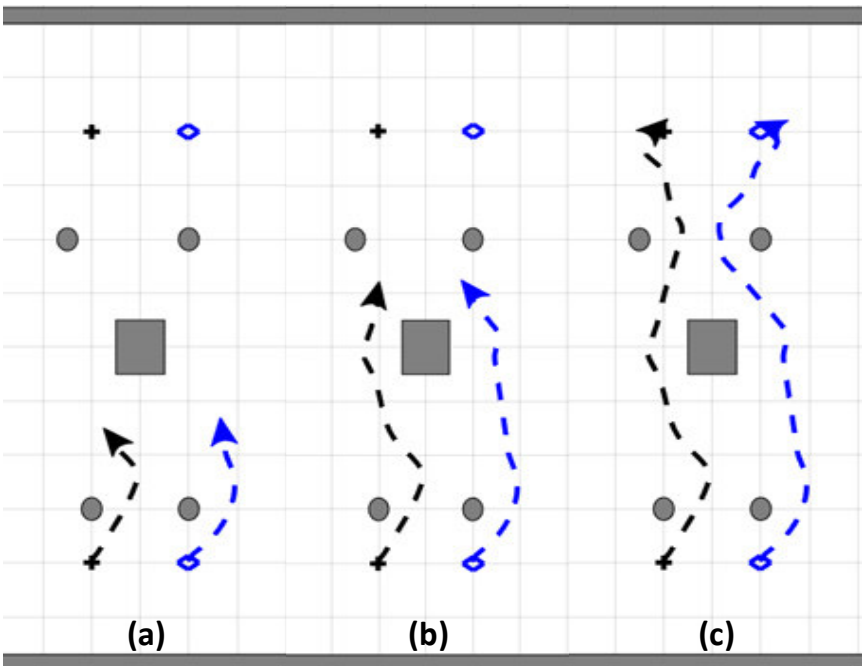


Figura 3.11. Trayectoria de los 2 agentes para múltiples obstáculos [Fuente Propia].

En función de las pruebas realizadas se observó que el comportamiento de los agentes se ve afectado por cualquier tipo de obstáculo considerando la distancia al mismo, pero solo para determinar la dirección de giro que evitaría la colisión inminente, sin tener en cuenta el fin de la trayectoria. Dando como resultado, movimientos o evasiones innecesarias, tal como se observa en el último tramo de la Figura 3.11.

3.3.3. PRUEBA CON 3 AGENTES

Para los casos de 3 o más agentes se desarrolla la prueba mostrada en la Figura 3.12. La cual se basa en el intercambio de posiciones de los puntos antipodales (puntos diametralmente opuestos de un círculo), lo cual permitirá el escalamiento y verificación del sistema a cualquier número de agentes.

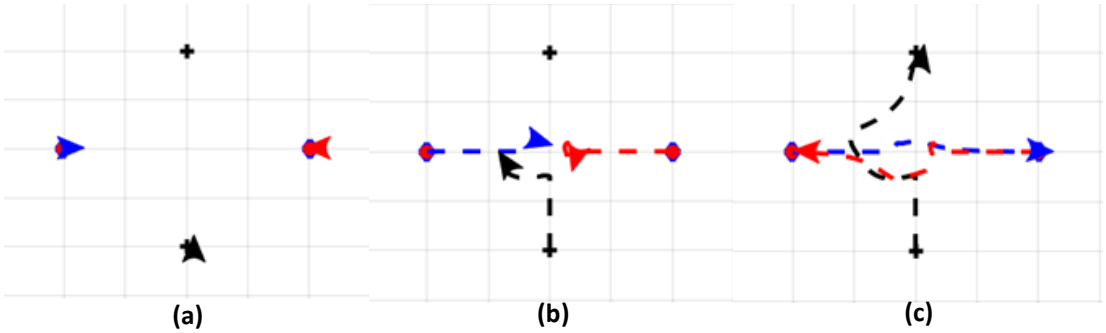


Figura 3.12. Trayectorias para 3 agentes para el intercambio de posiciones [Fuente Propia].

En el caso de los 3 agentes podemos corroborar la autonomía de cada uno de ellos buscando la dirección libre de obstáculos más rápida o cercana a la dirección objetivo como se plantea en el desarrollo del algoritmo VFH+.

3.3.4. PRUEBAS CON 4 AGENTES

Previamente en la prueba con 2 agentes, se mencionó que la alta simetría o semejanza entre agentes podría generar conflictos en ciertos casos, sin embargo, estos son casos muy particulares y bajo condiciones ideales solo obtenidas en un entorno virtual. Además, al observar el comportamiento de un grupo más extenso y con alta simetría respecto a las trayectorias de cada uno los agentes, como el de la Figura 3.13, se tiene que individualmente cada agente es capaz de llegar su objetivo, sin la necesidad de una generación de asimetría por software. A la vez, se demuestra un comportamiento colectivo propio de un sistema enjambre, sin un control centralizado como se había planteado en la Sección 1.4.2.

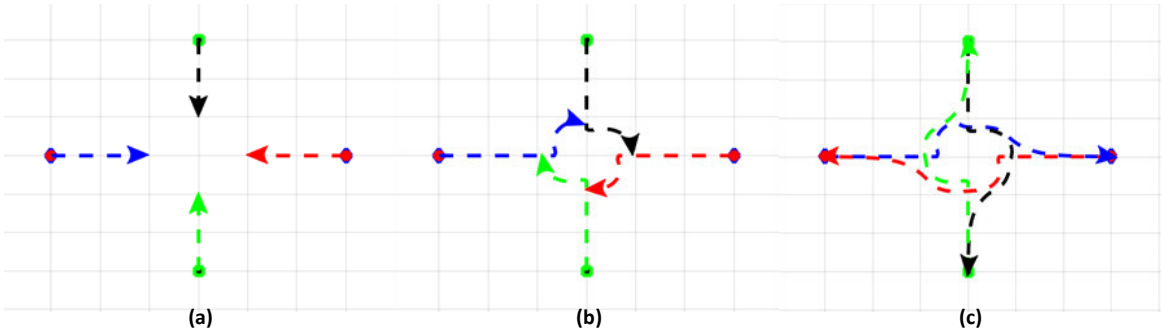


Figura 3.13. Trayectorias para 4 agentes para el intercambio de posiciones antipodales [Fuente Propia].

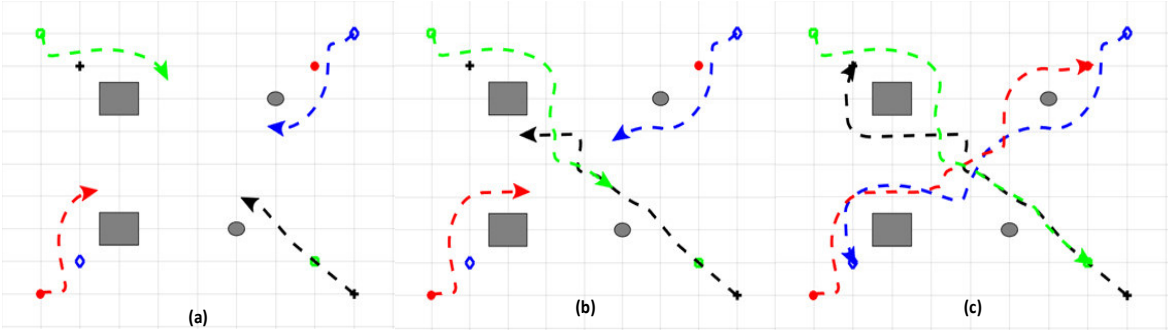


Figura 3.14.Trayectorias para 4 agentes para el intercambio de posiciones en formación cuadrada [Fuente Propia].

Por otra parte, en la prueba mostrada en la Figura 3.14. tenemos una situación con alta probabilidad de colisiones, especialmente inter-colisiones, lo que lleva a rutas relativamente complicadas, o no son las más eficientes; sin embargo, la tarea siempre es completada satisfactoriamente. Por lo que, para el caso mostrado en la Figura 3.14, se plantea una lógica similar a la asimetría planteada con 2 agentes (véase Figura 3.10), en donde se detienen ciertos agentes una vez que se determina una colisión próxima, como es el caso de los agentes rojo y azul en la Figura 3.14.(b). Estos se detienen, permitiendo el paso del resto de agentes del enjambre, lo que reduce considerablemente la cantidad de inter-colisiones del sistema y facilita que los agentes negro y verde tengan una trayectoria más rápida o eficiente respecto al sistema sin esta asimetría.

3.3.5. PRUEBA CON UN MAYOR NÚMERO DE AGENTES

Una vez verificado el sistema para 4 agentes se verifica la escalabilidad con más agentes, en este caso 6 y se realiza la prueba planteada en la Figura 3.15. En la misma, se puede observar un comportamiento similar al del cado para 4 agentes, donde estos rotaban sobre un eje imaginario sin la necesidad de un control centralizado, solo con su comportamiento individual, que genera una coordinación colectiva bastante llamativa.

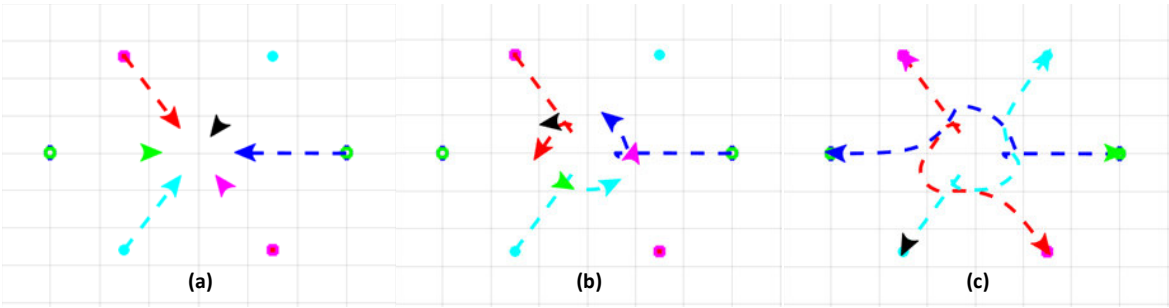


Figura 3.15.Trayectorias para 6 agentes para el intercambio de posiciones antipodales [Fuente Propia].

Con el fin de cuantificar cuan efectiva es la arquitectura de control planteada en una situación con alta probabilidad de colisiones, como la planteada en la Figura 3.15, se compara la tasa de éxito de la prueba en 20 repeticiones de esta. Los resultados obteniendo se presentan en las Figura 3.16. y Figura 3.17. En las mismas se considera que la prueba fue un éxito del 100 % si todos los agentes no tuvieron ninguna colisión y llegaron a su objetivo, por lo que en el caso de colisiones la tasa decaerá. Esto se puede observar en la tercera prueba realizada, donde la mitad de los agentes tuvieron colisiones. Cabe aclarar, que, si bien se hace referencia a colisiones, las mismas, no fueron más que roces que se determinó con la distancia mínima entre los agentes (véase Figura 3.17), además que en todos los casos los agentes si lograron llegar a la meta.

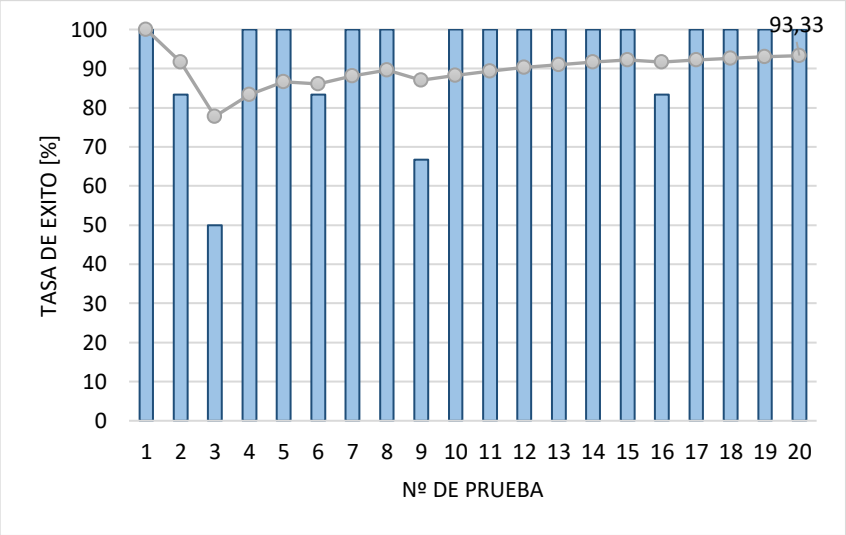


Figura 3.16. Tasa de éxito de la arquitectura de control [Fuente Propia].

Entonces se determina que la tasa de éxito que tiene el sistema para este tipo de colisiones es de alrededor del 93.33% para esta prueba, el cual se considera un gran logro respecto a la arquitectura de control planteada y el algoritmo de evasión de obstáculos implementado en el sistema enjambre.

A la par de las pruebas realizadas, se analizaron los datos de:

- **Tiempo extra [s]:** El cual indica el tiempo adicional en segundos que se demoran los agentes en llegar a su objetivo respecto al tiempo que le toma a un agente recorrer el mismo trayecto sin obstáculos. Se observa en la Figura 3.17 que en promedio les toma 9.59 [s] extra a los agentes en sortear a sus pares. Además, se observa una mayor desviación hacia arriba, valores que indican que en la mayoría de los casos se tomó más tiempo, llegando a alrededor de 23.64 [s], esto debido a que en los casos donde

hubo roces o colisiones los robots necesitaron formar otras trayectorias con mayor recorrido que las mostradas en la Figura 3.15.

- **Velocidad Promedio [m/s]:** Se determina en función del módulo de velocidad lineal y angular, y nos da un indicativo de la suavidad del movimiento de los agentes, ya que al tener una velocidad lineal constante de 0.2 [m], la variabilidad de velocidad angular se refleja en un aumento de la velocidad promedio total. Se observa en la Figura 3.17 que no existe mucha variabilidad en la velocidad de los robots, esto principalmente a la velocidad lineal constante de 0.20 [m/s], sin embargo, se pueden ver contadas excepciones (mostradas como puntos) que tienen una alta variabilidad de la velocidad angular.

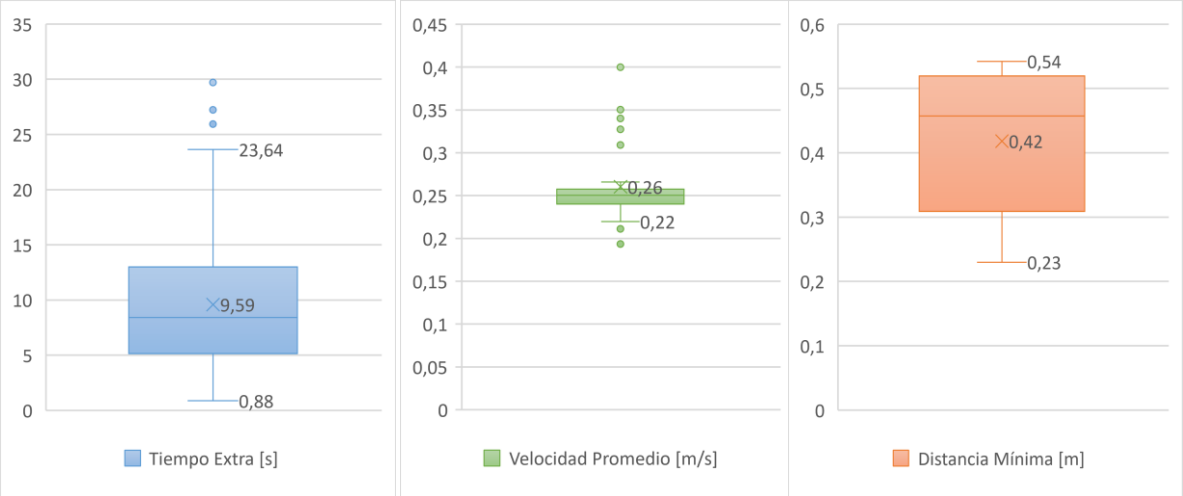


Figura 3.17.Gráfico de cajas y bigotes de las pruebas realizadas [Fuente Propia].

- **Distancia Mínima [m]:** Este dato permite evaluar el éxito de los agentes para cumplir su tarea, ya que si se tiene una distancia mínima menor al de radio del agente (0.22 [m]) se considera una colisión. En este caso se dio una holgura (0.25 [m]) para que se pueda determinar no solo colisiones, sino roces entre los agentes. Tal como se observa en la Figura 3.17, la cantidad de colisiones y roces fue mínima.

3.4. CONCLUSIONES

- Al investigar las capacidades y características que poseen los sistemas tipo enjambres, se observó que existen muchos comportamientos colectivos y que estos presentan aplicaciones en ámbitos muy variados como la agricultura, rescate, exploración, entretenimiento, entre otros, lo que los está constituyendo en una excelente opción para futuras aplicaciones en la industria.

- Tras el uso de la plataforma robótica TurtleBot, se comprueba que, debido a su variedad de sensores y modularidad, permite una gran versatilidad para aplicaciones multi-robóticas no solamente limitadas a sistemas enjambre, sino para una gran mayoría de aplicaciones que requieran robots de tipo terrestre.
- Al usar y analizar las herramientas presentes en ROS y su interacción con programas de terceros como lo son Gazebo y MATLAB se determina que cumple su función como un entorno de simulación robusto y confiable para el prototipado, simulación y desarrollo de aplicaciones robóticas.
- Se verifica el uso del algoritmo VFH+ y su incorporación en la arquitectura de control del sistema enjambre, encargándose de la evasión de obstáculos propios del entorno, además de prevenir inter-colisiones entre los agentes, mientras permite la ejecución del comportamiento enjambre.
- Tal como se pudo comprobar en las pruebas realizadas, los sistemas robóticos de tipo enjambre son capaces de resolver tareas complejas, a partir del uso de reglas locales. En este caso la arquitectura de control basada en subsunción le da el nivel de autonomía necesario para que, a partir del comportamiento individual o independiente de cada uno de sus agentes robóticos, de como respuesta un comportamiento organizado, colectivo y distribuido.
- Considerando las pruebas realizadas, se verifica que la respuesta para obstáculos fijos es suave y sin cambios bruscos en la trayectoria; sin embargo, para obstáculos dinámicos como lo son los propios agentes vecinos tiende a una respuesta relativamente brusca o volátil, (justamente por el alto dinamismo de los obstáculos), lo que da como resultado que su respuesta no sea 100% confiable.

3.5. RECOMENDACIONES

- Una vez se ha comprobado el funcionamiento del sistema multi-robótico en un entorno simulado, se recomienda su despliegue con los robots reales, lo que permitirá comprobar el funcionamiento de la arquitectura de control y sus posibles limitaciones, así como del comportamiento enjambre en un entorno real y proponer a futuro posibles aplicaciones de este sistema. Por otra parte, se pondrá a prueba las capacidades de ROS y MATLAB para su ejecución en Hardware.
- A partir de las pruebas realizadas se observó que, si bien los agentes logran cumplir su objetivo de llegar a la posición deseada (x, y), su postura es determinada en función de

la trayectoria que tomo para llegar a ese punto. Por lo cual se recomienda la incorporación de un control de orientación que permita que el robot tome cualquier ángulo deseado y esto a su vez permita el desarrollo de tareas más complejas por parte de todo el sistema, como la formación de patrones.

- Como ya se mencionó previamente, para la evasión de los obstáculos se parte de un conocimiento local basado en el sensor LIDAR y sin información previa del entorno, lo que podría llevar al robot a situaciones sin salida dependiendo del entorno. Se recomienda que a la vez que los robots pasen por el entorno se genere un mapa de este (SLAM) y con el mismo planear y ejecutar una trayectoria óptima. Lo cual formaría una capa de control superior a la planeación de trayectorias dentro de la arquitectura planteada, para este fin se recomienda el algoritmo VFH*.
- En este proyecto se plantea el uso de una plataforma robótica de robots de tracción diferencial, sin embargo, debido a la naturaleza del sistema enjambre en donde puede existir una gran cantidad de colisiones, además que, las restricciones de movimiento (no-holonómicas) de los agentes representan una limitante en las posibles direcciones de evasión, se recomienda analizar o desarrollar un sistema enjambre basado en robots omnidireccionales, los cuales eliminan esta limitante.
- Teniendo en cuenta la respuesta brusca o no completamente eficiente para obstáculos dinámicos, se recomienda el análisis o implementación de otros algoritmos de evasión de obstáculos que consideren no solamente la dinámica del robot en cuestión, sino adicionalmente la variación con el entorno. Por lo cual el algoritmo no sería necesariamente reactivo y sus requerimientos computacionales serían mayores, para este fin se recomienda el algoritmo VFH+D.

4. REFERENCIAS BIBLIOGRÁFICAS

[1] M. Luisa and S. Tortosa, “Agentes y enjambres artificiales: modelado y comportamientos para sistemas de enjambre robóticos,” 2013.

[2] P. G. Faria Dias, M. C. Silva, G. P. Rocha Filho, P. A. Vargas, L. P. Cota, and G. Pessin, “Swarm Robotics: A Perspective on the Latest Reviewed Concepts and Applications,” *Sensors 2021, Vol. 21, Page 2062*, vol. 21, no. 6, p. 2062, Mar. 2021, doi: 10.3390/S21062062.

[3] A. R. Cheraghi, S. Shahzad, and K. Graffi, “Past, Present, and Future of Swarm Robotics,” *Lecture Notes in Networks and Systems*, vol. 296, pp. 190–233, Sep. 2021, doi: 10.1007/978-3-030-82199-9_13.

[4] I. Navarro and F. Matía, “An Introduction to Swarm Robotics,” *ISRN Robotics*, vol. 2013, pp. 1–10, Sep. 2013, doi: 10.5402/2013/608164.

[5] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning”, Accessed: Dec. 07, 2021. [Online]. Available: https://sites.google.com/view/drlmaca.

[6] D. Xu, X. Zhang, Z. Zhu, C. Chen, and P. Yang, “Behavior-based formation control of swarm robots,” *Mathematical Problems in Engineering*, vol. 2014, 2014, doi: 10.1155/2014/205759.

[7] “TurtleBot3.” https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed Aug. 22, 2021).

[8] il Bambino, “[PDF] Una Introducción a los Robots Móviles. Il Bambino - Free Download PDF,” 2008. https://silo.tips/download/una-introduccion-a-los-robots-moviles-il-bambino (accessed Nov. 23, 2021).

[9] M. B. Alatise and G. P. Hancke, “A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods,” *IEEE Access*, vol. 8, pp. 39830–39846, 2020, doi: 10.1109/ACCESS.2020.2975643.

[10] R. Martínez-Clark, C. Cruz-Hernández, J. Pliego-Jimenez, and A. Arellano-Delgado, “Control algorithms for the emergence of self-organized behaviours in swarms of differential-traction wheeled mobile robots,” 2018, doi: 10.1177/1729881418806435.

[11] C. de I. de la U. D. F. J. de Caldas, “Revista Tecnura,” *Tecnura*, vol. 6, no. 12, pp. 19–30, Jan. 2003, doi: 10.14483/22487638.6131.

[12] S. M. Lavalle, “Chapter 13 Differential Models”, Accessed: Dec. 06, 2021. [Online]. Available: http://planning.cs.uiuc.edu/

[13] G. Beni, “From Swarm Intelligence to Swarm Robotics,” *Lecture Notes in Computer Science*, vol. 3342, pp. 1–9, 2004, doi: 10.1007/978-3-540-30552-1_1.

[14] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, “Swarm Robotic Behaviors and Current Applications,” *Frontiers in Robotics and AI*, vol. 7, p. 36, Apr. 2020, doi: 10.3389/FROBT.2020.00036/BIBTEX.

[15] “How termite mounds get their shape.” https://www.seas.harvard.edu/news/2019/02/how-termite-mounds-get-their-shape (accessed Dec. 07, 2021).

[16] D. Shah and L. Vachhani, “Swarm Aggregation Without Communication and Global Positioning,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 886–893, Apr. 2019, doi: 10.1109/LRA.2019.2893413.

[17] M. Coppola, J. Guo, E. Gill, and G. C. H. E. de Croon, “Provable self-organizing pattern formation by a swarm of robots with limited knowledge,” *Swarm Intelligence*, vol. 13, no. 1, pp. 59–94, Mar. 2019, doi: 10.1007/S11721-019-00163-0/FIGURES/16.

[18] N. Aubert-Kato *et al.*, “Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017,” vol. 8, 2017, doi: 10.1145/3071178.

[19] J. Werfel, K. Petersen, and R. Nagpal, “Distributed Multi-Robot Algorithms for the TERMES 3D Collective Construction System”.

[20] S. Zhang, R. Pohlmann, T. Wiedemann, A. Dammann, H. Wymeersch, and P. A. Hoeher, “Self-Aware Swarm Navigation in Autonomous Exploration Missions,” *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1168–1195, Jul. 2020, doi: 10.1109/JPROC.2020.2985950.

[21] S. Berman, Q. Lindsey, M. S. Sakar, V. Kumar, and S. C. Pratt, “Experimental study and modeling of group retrieval in ants as an approach to collective transport in swarm robotic systems,” *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1470–1481, 2011, doi: 10.1109/JPROC.2011.2111450.

[22] M. Vigelius, B. Meyer, and G. Pascoe, “Multiscale Modelling and Analysis of Collective Decision Making in Swarm Robotics,” *PLOS ONE*, vol. 9, no. 11, p. e111542, Nov. 2014, doi: 10.1371/JOURNAL.PONE.0111542.

[23] Y. S. Dai, M. Hinchey, M. Madhusoodan, J. L. Rash, and X. Zou, “A prototype model for Self-Healing and self-reproduction in swarm robotics system,” *Proceedings - 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC 2006*, pp. 3–10, 2006, doi: 10.1109/DASC.2006.10.

[24] J. Alonso-Mora, R. Siegwart, and P. Beardsley, “Human - Robot swarm interaction for entertainment: From animation display to gesture based control,” *ACM/IEEE*

International Conference on Human-Robot Interaction, p. 98, 2014, doi: 10.1145/2559636.2559645.

[25] D. Mateo, N. Horsevad, V. Hassani, M. Chamanbaz, and R. Bouffanais, "Optimal network topology for responsive collective behavior," *Science Advances*, vol. 5, no. 4, 2019, doi: 10.1126/SCIADV.AAU0999/FORMAT/PDF.

[26] J. Miguel Ibañez de Aldecoa Quintana, "NIVELES DE MADUREZ DE LA TECNOLOGÍA. TECHNOLOGY READINESS LEVELS. TRLS".

[27] "SwarmFarm Robotics | Robotic Agriculture." https://www.swarmfarm.com/ (accessed Aug. 23, 2021).

[28] "Project Xaver | Fendt Future Farm | Fendt World - Fendt." https://www.fendt.com/int/xaver (accessed Aug. 23, 2021).

[29] M. S. Couceiro, D. Portugal, R. P. Rocha, and A. Araújo, "Fostering human–robot cooperative architectures for search and rescue missions in urban fires," *https://doi.org/10.1177/0037549720964548*, vol. 97, no. 3, pp. 177–194, Nov. 2020, doi: 10.1177/0037549720964548.

[30] D. Jain and M. Y. Sharma, "Adoption of next generation robotics: A case study on Amazon," *A Case Research Journal*, vol. III, 2017.

[31] O. Salzman and R. Stern, "Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems," 2020.

[32] "Swarm Intelligence Layer to Control Autonomous Agents." https://swilt.aau.at/ (accessed Aug. 23, 2021).

[33] Aguilar Jose, Rios Addison, Hidrobo Francisco, and Cerrada Mariela, *Sistemas MultiAgentes y sus Aplicaciones en Automatización Industrial*. 2013. Accessed: Dec. 08, 2021. [Online]. Available: https://www.researchgate.net/publication/320935195_Sistemas_MultiAgentes_y_sus_Aplicaciones_en_Automatizaci_on_Industrial

[34] R. C. Arkin, *Behaviour-Based Robotics (Intelligent Robotics and Autonomous Agent Series)*. MIT Press, 1998. Accessed: Dec. 08, 2021. [Online]. Available: https://books.google.com.ec/books?hl=en&lr=&id=mRWT6alZt9oC&oi=fnd&pg=PR11&dq=robotics&ots=451scfQ7qE&sig=MjizqFNWwsqaTEuGh-s7J6wEb54&redir_esc=y#v=onepage&q=robotics&f=false

[35] K. Shahzad, S. Iqbal, and P. Bloodsworth, “Points-Based Safe Path Planning of Continuum Robots,” *https://doi.org/10.5772/60857*, vol. 12, Jul. 2015, doi: 10.5772/60857.

[36] H. T. Nguyen and H. X. Le, “Path planning and Obstacle avoidance approaches for Mobile robot,” *International Journal of Computer Science Issues*, vol. 13, no. 4, pp. 1–10, Sep. 2016, doi: 10.20943/01201604.110.

[37] G. Li, A. Yamashita, H. Asama, and Y. Tamura, “An efficient improved artificial potential field-based regression search method for robot path planning,” *2012 IEEE International Conference on Mechatronics and Automation, ICMA 2012*, pp. 1227–1232, 2012, doi: 10.1109/ICMA.2012.6283526.

[38] Y. Zhu, T. Zhang, J. Song, and X. Li, “A new hybrid navigation algorithm for mobile robots in environments with incomplete knowledge,” *Knowledge-Based Systems*, vol. 27, pp. 302–313, Mar. 2012, doi: 10.1016/J.KNOSYS.2011.11.009.

[39] W. J. Yim and J. B. Park, “Analysis of mobile robot navigation using vector field histogram according to the number of sectors, the robot speed and the width of the path,” *International Conference on Control, Automation and Systems*, pp. 1037–1040, Dec. 2014, doi: 10.1109/ICCAS.2014.6987943.

[40] by J. Borenstein, Y. Koren, and S. Member, “THE VECTOR FIELD HISTOGRAM-FAST OBSTACLE AVOIDANCE FOR MOBILE ROBOTS,” *IEEE Journal of Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[41] “ROS.org | About ROS.” https://www.ros.org/about-ros/ (accessed Aug. 22, 2021).

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

[43] “ROS Toolbox - MATLAB.” https://la.mathworks.com/products/ros.html (accessed Aug. 18, 2021).

[44] J. P. Briot, T. Meurisse, and F. Peschanski, “Architectural Design of Component-Based Agents: A Behavior-Based Approach,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4411 LNAI, pp. 71–90, 2006, doi: 10.1007/978-3-540-71956-4_5.

[45] R. A. Brooks, “A Robust Layered Control System for A Mobile Robot,” *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986, doi: 10.1109/JRA.1986.1087032.

[46] G. Alonso *et al.*, “Sistema robótico autónomo para la exploración y construcción de mapas en entornos estructurados,” *Investigación e Innovación en Ingenierías, ISSN-e 2344-8652, Vol. 8, Nº. 1 (enero-Julio), 2020, págs. 69-84*, vol. 8, no. 1, pp. 69–84, 2020, doi: 10.17081/invinno.8.1.3593.

[47] “Pure Pursuit Controller - MATLAB & Simulink.” https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html (accessed Jan. 05, 2022).

[48] W. J. Wang, T. M. Hsu, and T. S. Wu, “The improved pure pursuit algorithm for autonomous driving advanced system,” *2017 IEEE 10th International Workshop on Computational Intelligence and Applications, IWCIA 2017 - Proceedings*, vol. 2017-December, pp. 33–38, Dec. 2017, doi: 10.1109/IWCIA.2017.8203557.

[49] M. Samuel *et al.*, “A Review of some Pure-Pursuit based Path Tracking Techniques for Control of Autonomous Vehicle,” *International Journal of Computer Applications*, vol. 135, no. 1, pp. 975–8887, 2016.

[50] I. Ulrich and J. Borenstein, “VFH+: Reliable obstacle avoidance for fast mobile robots,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1572–1577, 1998, doi: 10.1109/ROBOT.1998.677362.

[51] “Vector Field Histogram - MATLAB & Simulink.” https://www.mathworks.com/help/nav/ug/vector-field-histograms.html (accessed Jan. 05, 2022).

[52] “Gazebo.” http://gazebosim.org/ (accessed Aug. 22, 2021).

[53] “Topics in Quaternion Linear Algebra | Princeton University Press books | IEEE Xplore.” https://ieeexplore.ieee.org/document/9452663 (accessed Jan. 24, 2022).

5. ANEXOS

De manera general para las páginas web referenciadas en los anexos se utiliza hipervínculos los cuales se encuentran subrayados y con color celeste.

ANEXO I. Instaladores Programas:

Instalación Ubuntu 20.04:

Para la instalación y configuración del sistema operativo Linux se recomienda seguir el siguiente tutorial, es importante destacar, que el SO puede correr en una máquina virtual.

Instalación ROS:

Una vez se tiene corriendo el sistema operativo de Linux, se instala ROS por medio de los pasos descritos en la ROS WIKI para la distribución de Noetic.

Instalación recomendada (Linea de comando):

```
sudo apt install ros-noetic-desktop-full
```

Familiarización con ROS:

Al completar la instalación de ROS, se recomienda seguir los tutoriales para la familiarización del entorno de Linux y ROS, así como la creación del área de trabajo (Workspace).

Comandos útiles ROS:

Si bien dentro de los tutoriales de ROS se puede encontrar los comandos que pueden ser usados para controlar ROS por medio de la línea de comandos, es necesario tener a la mano algunos comandos que son los más usados al momento de trabajar con ROS:

```
roscore
```

Permite iniciar el nodo principal de ROS, es vital para correr el resto de los nodos.

```
rosrun [nombre del archivo a correr sin extensión]
```

Ejecuta los scripts de los distintos paquetes de ROS, es necesario la creación de los ejecutables previamente.

```
roslaunch [nombre del archivo. launch]
```

Ejecuta los archivos con extensión. launch, lo cual facilita la generación de múltiples nodos.

```
roscd
```

Cambia el directorio a la carpeta del área de trabajo de ROS.

```
rqt_graph
```

Abre una ventana con una visualización de los nodos y tópicos de ROS, así como la interacción entre ellos.

```
rqt_topic
```

Abre una ventana con la lista de todos los tópicos disponibles en ROS, así como los mensajes actuales de los mismos.

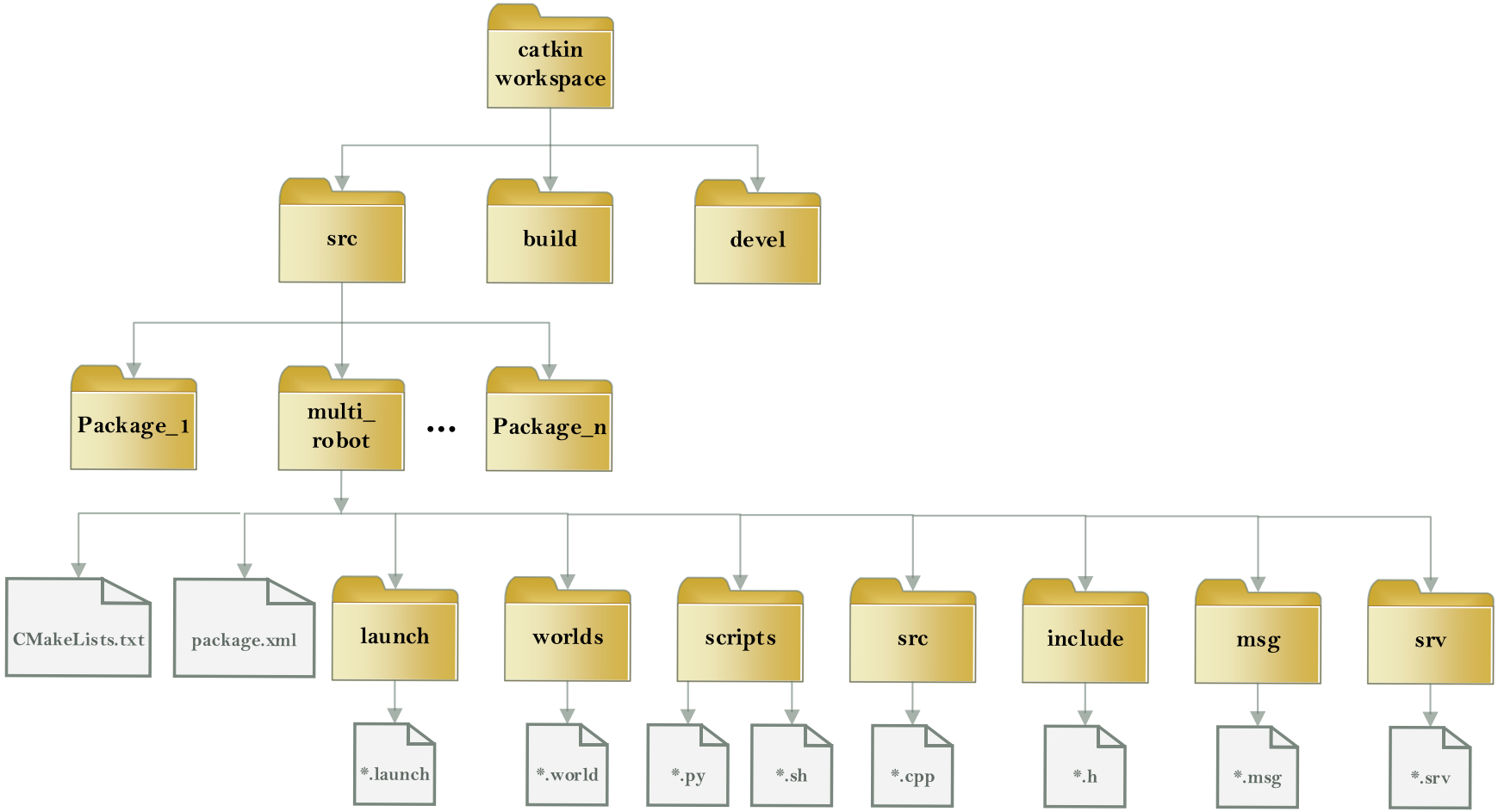
```
source ./devel/setup.bash
```

Este comando debe ser ejecutado al abrir una nueva ventana del terminal mientras se está en la carpeta de trabajo (catkinws), permite la ejecución de archivos bash con los comandos de ROS.

Instalación Gazebo:

El simulador de Gazebo puede ser instalado desde su página oficial, siguiendo los pasos indicados en la misma. En este proyecto se utiliza Gazebo 11.0.0.

ANEXO II. Distribución Carpetas Workspace



ANEXO III. Documentación ROS

Gazebo:

- Documentación Tutoriales: Documentación para familiarizarse con el entorno de Gazebo, así como la explicación a bajo y alto nivel de su funcionamiento e interacción con modelos.
- Documentación SDF :Documentación del formato SDF con el cual se desarrolla los archivos. world, que describen el ambiente en Gazebo.
- Video Tutoriales: Explicación detallada de ejemplos base en Gazebo.
- Plataforma de Aprendizaje: Plataforma Web para aprender conceptos básicos de ROS y su integración con Gazebo.
- Cuaterniones: Explicación y aplicaciones de los cuaterniones.

Turtlebot3:

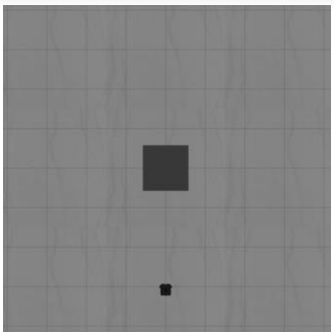
- TurtleBot3: Página oficial de la plataforma robótica.
- Especificaciones :Características físicas y de hardware de la plataforma robótica.
- Instalación Package: Instalación detallada de la instalación del package “TurtleBot3 simulation”, para su integración con Gazebo. Este cuenta con ejemplos de uso del Turtlebot.
- Turtlebots Spawning: Explicación y código para la generación o spawn del robot o robots en Gazebo con sus respectivos tópicos de sensores y actuadores.

ROS Toolbox:

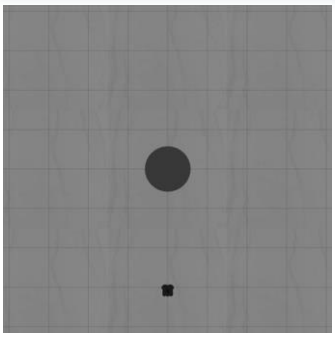
- Guia: Guia de usuario de ROS Toolbox.

ANEXO IV. Banco de Pruebas

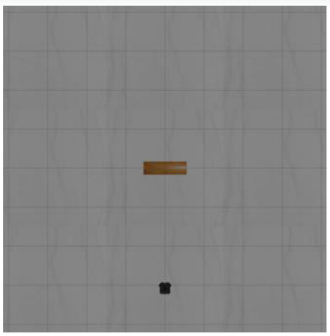
One_Robot_Cube



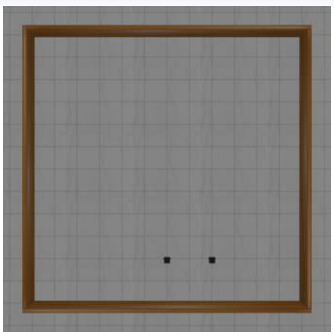
One_Robot_Cylinder



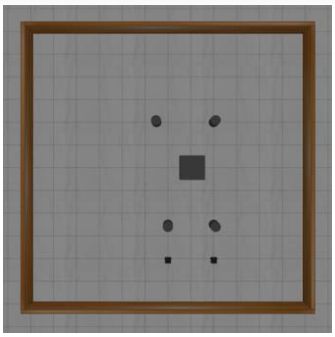
One_Robot_Wall



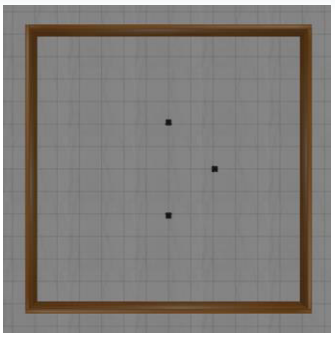
Two_Robots



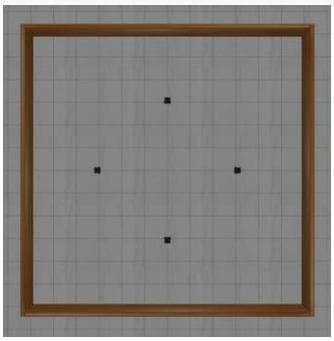
Two_Robots_Obstacle



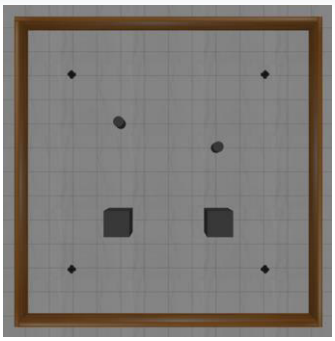
Three_Robots



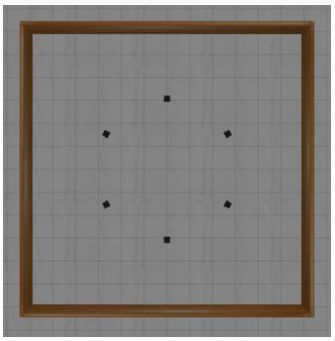
Four_Robots_Diamond



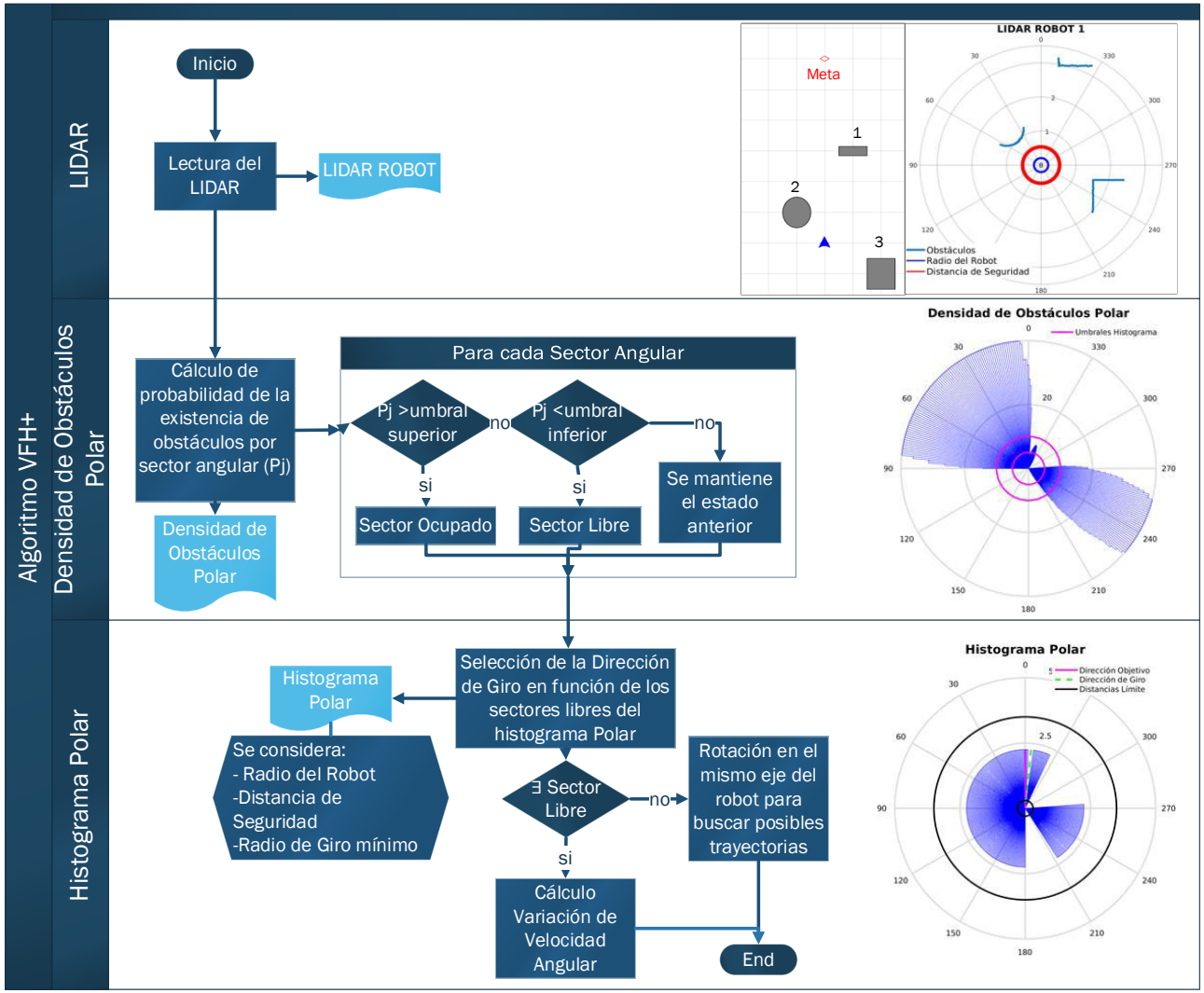
Four_Robots_Square



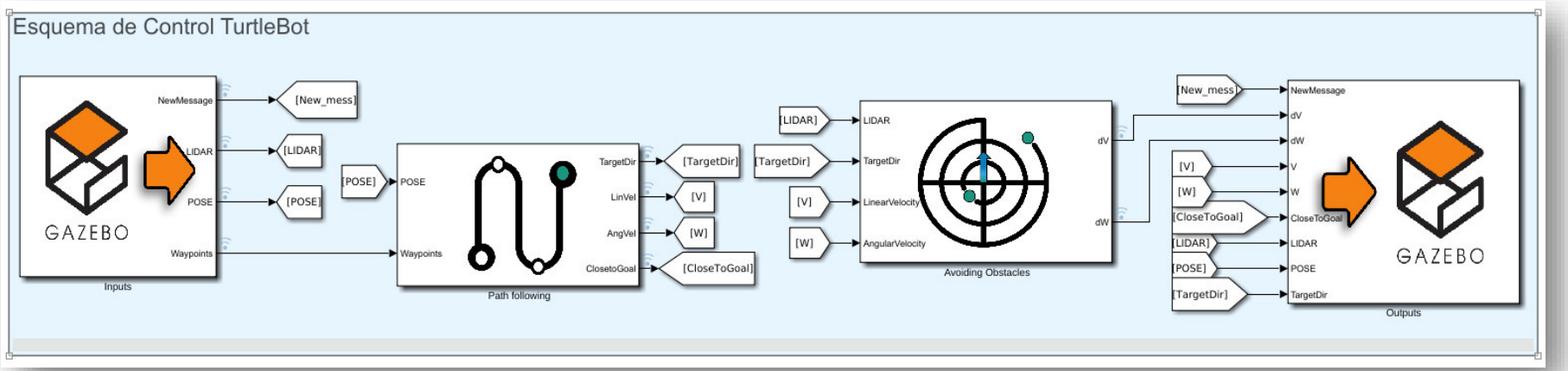
Six_Robots

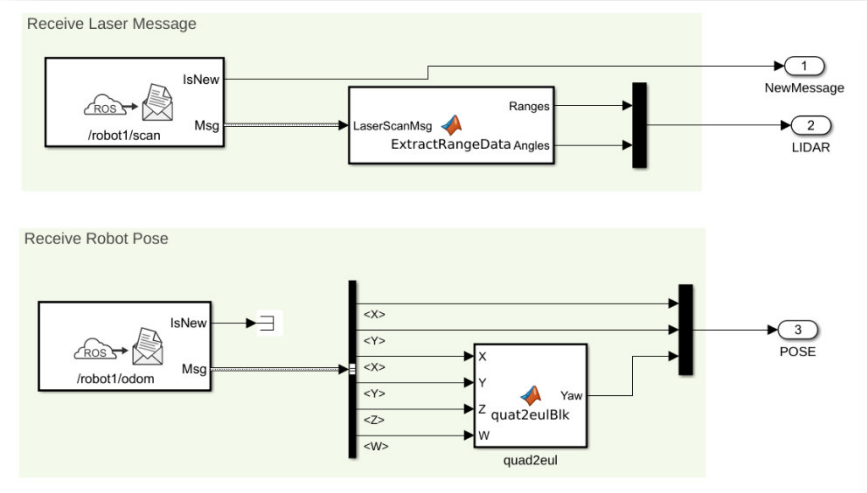


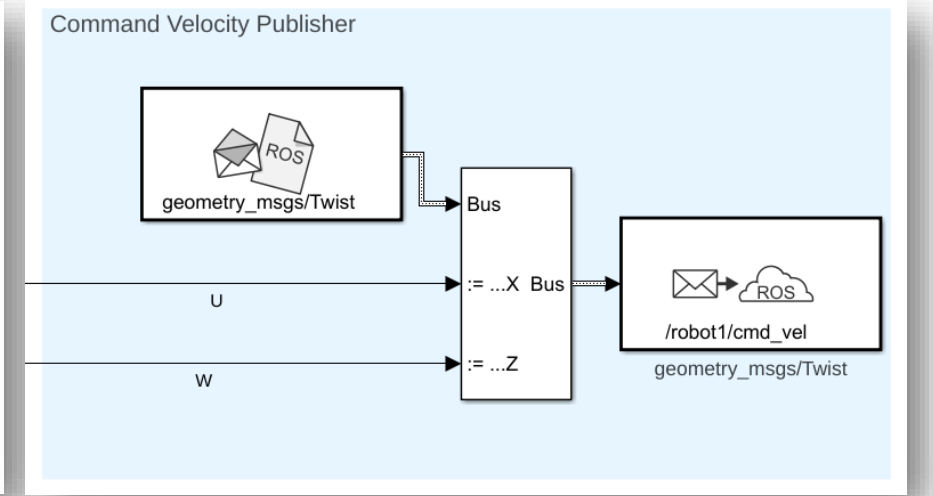
ANEXO V. Diagrama De Flujo Algoritmo VFH+



ANEXO VI. Implementación Del Esquema De Control







ANEXO VII. Manual de Usuario

➢ Requisitos del Sistema:

Software	Versión
Sistema Operativo	Linux Ubuntu 20.04 LTS
ROS	Noetic Ninjemys
Gazebo	11.0.0
MATLAB	2021A +

Además, dentro de Matlab será necesario contar como mínimo con:

- Simulink
- Robotics System Toolbox
- ROS Toolbox

➢ Copia del Package de ROS:

Al verificarse la instalación de todos los requisitos previamente mencionado, es necesario clonar o copiar el package del repositorio:

https://github.com/MarcoOrtega0/Swarm_Robotics

Dentro de este repositorio se tiene la carpeta multi_robot, la cual deberá ser copiada en el catkin workspace:

/home/USER/catkin_ws/src/multi_robot

Por otra parte, las carpetas de Gazebo y Matlab son copiadas en una carpeta en el escritorio de tal manera:

/home/USER/Escritorio/Swarm\ Robots

*USER es el nombre de usuario de Linux, por lo que debe ser modificado para cada computadora, y en los ejecutables dentro de la carpeta Gazebo

Se recomienda la compilación del paquete después de copiarlo, ejecutando las siguientes líneas de comandos:

```
roscd

catkin make
```

Para añadir los modelos de obstáculos diseñados para el banco de pruebas se copia el contenido de la carpeta models en multi_robot, dentro de la carpeta base de Gazebo, la cual se encuentra en: ~/ **.gazebo/models.** Sin estos modelos, los mundos de Gazebo aparecerán únicamente con los robots.

➢ Ejecución:

Para la ejecución del banco de pruebas se sigue el siguiente diagrama de flujo:

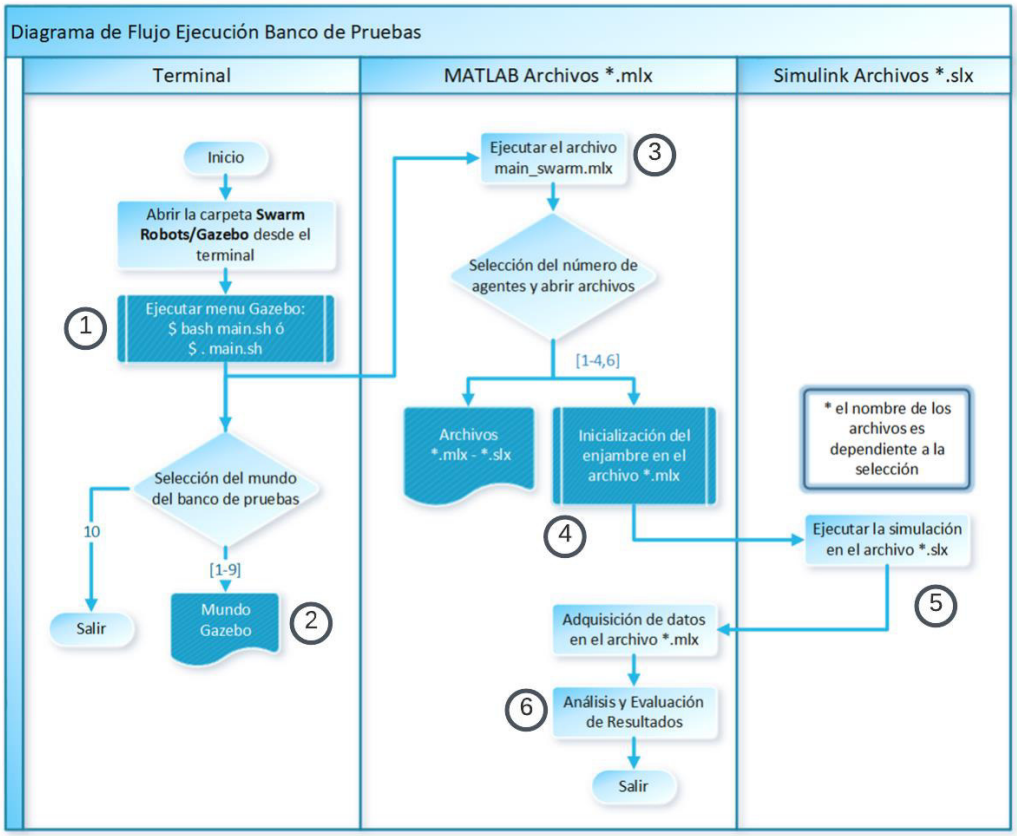


Figura VII.1. Diagrama de Flujo para la ejecución del Banco de Pruebas

1) Para iniciar, se parte ejecutando el menú de Gazebo por medio del comando:

```
bash main.sh
```

El cual abre el menú mostrado en la Figura VII.2, este nos permite seleccionar cualquiera de los mundos previamente desarrollados en el entorno de Gazebo.

2) Como ejemplo se abre el segundo mundo mostrado en la Figura VII.2. Dentro del mismo se podrá utilizar las herramientas de Gazebo para cambiar la visualización del entorno.

3) A la par, se abre el archivo Main_Swarm.mlx dentro de la carpeta de MATLAB mostrada en la Figura VII.3. En este se seleccionará el caso en función del número de robots existentes en el entorno de Gazebo seleccionado.

4) Una vez abierto el mundo de Gazebo y los archivos. mlx y .slx correspondientes. Se inicializa los controladores y el comportamiento enjambre en el archivo. mlx correspondiente como se muestra en la Figura VII.4.

5) Inicializado el sistema, se tienen las variables iniciales para la ejecución del archivo .slx. Es de vital importancia que estén todas las variables, caso contrario el modelo de Simulink dará error al momento de su ejecución. Caso contrario se puede correr el modelo directamente como se muestra en la Figura VII.5.

Se recomienda verificar la conexión del modelo de Simulink y el entorno de ROS si es la primera vez ejecutando el banco de pruebas.

6) Al finalizar la ejecución del archivo de Simulink, bien sea de manera manual o con el tiempo de simulación seleccionado en la inicialización, se procede a la adquisición y análisis de resultados como se muestra en la Figura VII.6.

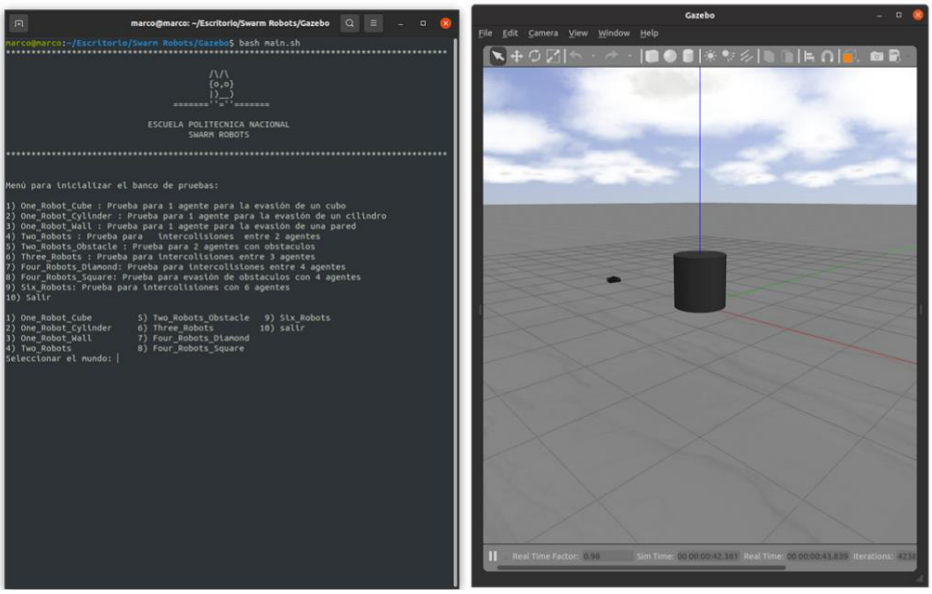


Figura VII.2. Menú para la selección del entorno y Ejemplo de este en Gazebo

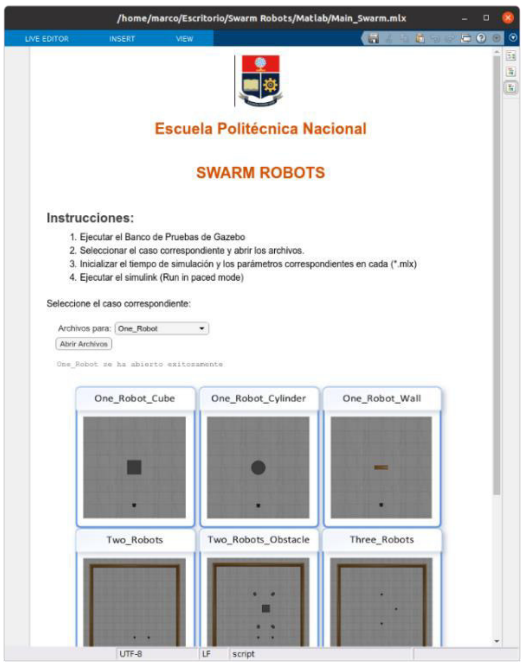


Figura VII.3. Menú para seleccionar los archivos de control del banco de pruebas.

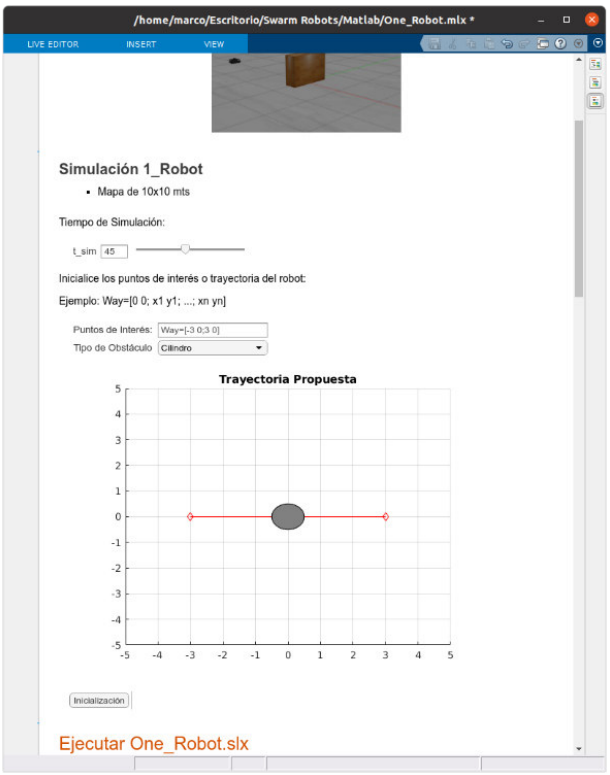


Figura VII.4.Ejemplo de Inicialización

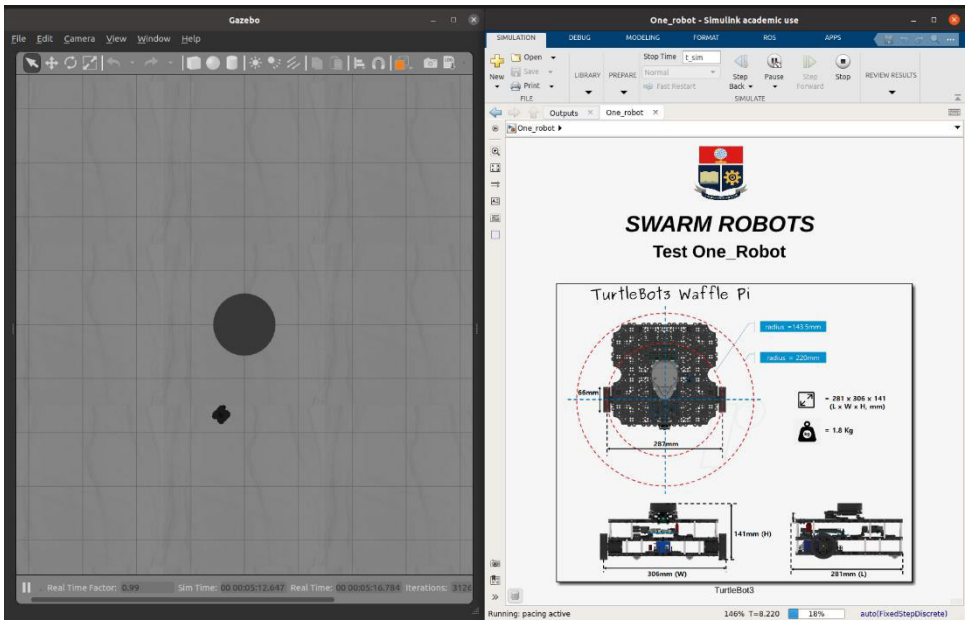


Figura VII.5. Ejemplo de Ejecución

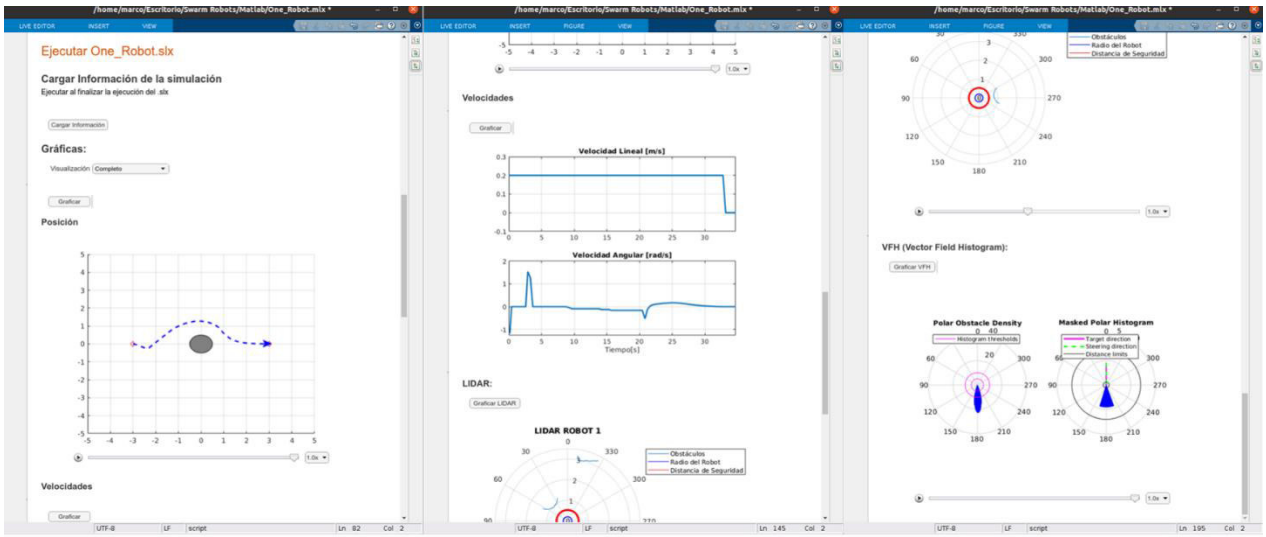


Figura VII.6. Ejemplo de Adquisición y Análisis de Resultados

En resumen, se selecciona el mundo del banco de pruebas y los archivos de Matlab-Simulink en función del número de agentes

Una vez abiertos los documentos y el banco de pruebas se realizan los siguientes pasos:

1. Inicialización, comportamiento enjambre (Archivo *.mlx)
2. Ejecución (Simulación en run paced mode) (Archivo *.slx)
3. Una vez finalizada la prueba adquirir y analizar los datos de simulación (Archivo *.mlx).

- **Modificación:**

Cualquier parte del entorno o arquitectura de control, así como la visualización de datos son modificables:

- Banco de Pruebas: Package multi_robot
- Arquitectura de control: Simulink
- Planeación y Visualización de datos: MATLAB. mlx

ANEXO VIII. Videos de Funcionamiento

Lista de reproducción: Swarm Robotics - YouTube

- Ejemplo de Ejecución Banco de Pruebas:

Prueba Ejecución del Banco de Pruebas - YouTube

- Pruebas para 1 Agente:

Pruebas para 1 Agente - YouTube

- Pruebas para 2 Agentes:

Pruebas para 2 Agentes - YouTube

- Pruebas para 3 Agentes:

Prueba para 3 Agentes - YouTube

- Pruebas para 4 Agentes:

Pruebas para 4 Agentes - YouTube

- Pruebas para 6 Agentes:

Prueba para 6 Agentes - YouTube

- Verificación de otros Comportamientos Enjambre:

Verificación Comportamientos Enjambre - YouTube