ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO, DESARROLLO E INTEGRACIÓN DE EQUIPOS AUXILIARES PARA EL CONTROL Y/O MONITOREO DE PLATAFORMAS ROBÓTICAS

UTILIZACIÓN DEL SISTEMA DE RECONOCIMIENTO DE GESTOS EMPLEADO EN EL PROYECTO DE INVESTIGACIÓN PIGR-19-07 PARA EL COMANDO DE UN SISTEMA MULTI-AGENTE

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

CINTHYA FERNANDA ORBE MONCAYO

cinthya.orbe@epn.edu.ec

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

patricio.cruz@epn.edu.ec

DMQ, octubre 2022

CERTIFICACIONES

Yo, CINTHYA FERNANDA ORBE MONCAYO 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.

CINTHYA FERNANDA ORBE MONCAYO

Certifico que el presente trabajo de integración curricular fue desarrollado por CINTHYA FERNANDA ORBE MONCAYO, 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.

CINTHYA FERNANDA ORBE MONCAYO

ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

DEDICATORIA

A mis padres, por el apoyo que me han demostrado en todas mis decisiones y el orgullo que sienten de mí por cada pequeño logro, orgullo impulsado por su amor por mí. Como hija siempre hay mucho por agradecer, por eso les doy esta pequeña parte de mí que ha involucrado un proceso lleno de esfuerzo y paciencia.

A mis hermanos, por vivir esta vida conmigo y ser los amigos que no se elige, pero que más amas. Por estar todas las etapas que recuerdo de mi vida y seguir estando, por su preocupación constante y su compañía.

A toda mi familia, en especial a mis abuelitos, los que desde el cielo me cuidan en cada paso, y los que aún los tengo en mi vida y con los que quiero compartir cada una de mis alegrías. A mis tíos y primos, qué con su cariño han sido mi combustible para seguir metiéndole ñeque y proponerme metas nuevas.

A mí, porque solo yo puedo ser quien me dirija y en un futuro me haga la persona que quiero ser.

AGRADECIMIENTO

En primer lugar, quiero agradecer por la vida y salud, porque sin ella no habría podido llegar hasta aquí. Todo mi proceso académico hasta el momento lo agradezco a mis padres, por siempre buscar las mejores opciones tanto en calidad como entorno, y por aceptar con dolor en el alma que me aleje de ellos para continuar estudiando lo que elegí. Gracias por su continuo cariño y hacerme saber que son mi lugar de apoyo y descanso cuando me siento cansada o con dudas. Los amo mucho.

A la Escuela Politécnica Nacional, que ahora lleva un gran lugar en mi corazón, por haber sido mi hogar durante tantos años, por haberme visto crecer en sus aulas, pasillos, canchas, cafetería, y todos los lugares donde cree recuerdos y aprendí cosas valiosas de mis profesores y amigos, tanto para mi vida profesional como personal.

Quiero agradecer también a mis amigos, a Carlita que solo veía en vacaciones, pero nunca dejamos de preocuparnos una por la otra, a mis amigas con las que viví mientras estuvimos lejos de nuestras casas: a Mela, Cami y Diani, por ser el apoyo mutuo en una nueva ciudad y haber creado una fuerte amistad, gracias por convertirse en mis hermanas. Por último, a aquellos con los que la vida me juntó y elegí pasar durante esta etapa, mis compañeros y amigos, quienes me acompañaron durante esta travesía y una de mis más memorables experiencias, mi más grande cariño a todos, en especial a mis confidentes Vaneza, Steven Matías y Pablito, gracias por su cariño incondicional. Gracias infinitas a Gaby, Marco y Rodrigo por ayudarme a definir mi camino hacia el final de este capítulo mediante este trabajo.

Mi más grande agradecimiento al Ing. Patricio Cruz, PhD., por su gran guía, paciencia y consejos brindados en todo este proceso, por ser uno de los mejores maestros que me pude encontrar y compartir toda su experiencia para hacer de este un buen trabajo. De igual manera a quienes forman parte del proyecto PIGR-19-07, por brindarme una cálida bienvenida e incluirme en las actualizaciones del proyecto, además de mostrar interés con sus preguntas y sugerencias.

A todas aquellas personas que sin saberlo formaron una parte especial para mi crecimiento: familia, amigos, maestros, compañeros, gracias a ustedes estoy aquí.

ÍNDICE DE CONTENIDO

CERTIFICACIONES I
DECLARACIÓN DE AUTORÍA II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
RESUMEN VII
ABSTRACT VIII
1 INTRODUCCIÓN 1
1.1 OBJETIVO GENERAL 2
1.2 OBJETIVOS ESPECÍFICOS 2
1.3 ALCANCE 2
1.4 MARCO TEÓRICO 3
1.4.1 SISTEMA DE DETECCIÓN DE GESTOS DEL PROYECTO PIGR-19-07 3
1.4.1.1 Sensor Myo Armband 4
1.4.1.2 Software de Reconocimiento 5
1.4.2 SISTEMA MULTI-AGENTE ROBÓTICO 6
1.4.3 ROBOT DE TRACCIÓN DIFERENCIAL - TURTLEBOT3 8
1.4.4 PLATAFORMAS DE SIMULACIÓN ROBÓTICAS 11
1.4.4.1 ROS-Gazebo 12
1.4.4.2 ROS Toolbox 13
1.4.4.3 Máquina Virtual 14
2 METODOLOGÍA 16
2.1 SISTEMA MULTI-AGENTE ROBÓTICO 17
2.1.1 IMPLEMENTACIÓN DE LA ESCENA 17
2.1.2 INTERFAZ DE COMUNICACIÓN ENTRE MATLAB Y GAZEBO 21
2.2 INTEGRACIÓN CON EL SISTEMA DE RECONOCIMIENTO DE GESTOS 22
2.2.1 LINEAMIENTOS DE COMANDOS DEL SISTEMA MULTI-AGENTE 24
2.2.1.1 Acciones mediante comandos del sistema de reconocimiento de gestos 25
2.2.1.2 Acciones mediante comandos de la IMU del sensor Myo Armband 26
2.2.2 INTERFAZ DE COMUNICACIÓN ENTRE INSTANCIAS DE MATLAB 28
2.3 CONTROL DE FORMACIONES 30
2.3.1 SISTEMAS DE COORDENADAS DE UBICACIÓN 30
2.3.2 ALGORITMO DE CONTROL DE FORMACIÓN 33

2.3.3 IMPLEMENTACIÓN DEL ALGORITMO EN SIMULINK 36
2.4 DESARROLLO DE LA INTERFAZ GRÁFICA 38
2.4.1 INTERFAZ EN LA PLATAFORMA DE SIMULACIÓN ROBÓTICA 39
2.4.2 INTERFAZ EN MATLAB 40
2.4.2.1 Prueba con 1 agente 41
2.4.2.2 Comando del Sistema Multi-Agente 41
3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES 42
3.1 PRUEBAS Y RESULTADOS 42
3.1.1 PRUEBAS CON 1 AGENTE 43
3.1.1.1 Sintonización del algoritmo Pure Pusuit 44
3.1.1.2 Condicionalidad de la velocidad lineal al ángulo 46
3.1.1.3 Movimiento comandado por la IMU 50
3.1.2 PRUEBAS DE CONTROL DE FORMACIÓN 51
3.1.2.1 Pruebas sin el Sistema de Reconocimiento 53
3.1.2.2 Pruebas con el Sistema de Reconocimiento 58
3.2 CONCLUSIONES 60
3.3 RECOMENDACIONES 62
4 REFERENCIAS BIBLIOGRÁFICAS 62
5 ANEXOS 66
ANEXO I 67
ANEXO II 72
ANEXO III 74
ANEXO IV 75
ANEXO V 82
ANEXO VI 86
ANEXO VII 88
ANEXO VIII 91

RESUMEN

El proyecto de investigación grupal PIGR-19-07 se basa en el reconocimiento de gestos, mismo que ha sido desarrollado empleando el sensor MYO Armband y MATLAB bajo el sistema operativo Windows. Para comprobar su aplicabilidad, se lo emplea, a través del presente proyecto, para el comando de cambio de formaciones de un sistema multi-agente robótico.

Este sistema se lo implementa en un entorno virtual, para lo cual se escoge la plataforma de simulación robótica basándose en los agentes robóticos y modo para comandarlos. En particular, el simulador Gazebo facilita la simulación del robot TurtleBot3, modelo del agente del sistema, y utiliza la plataforma ROS para su comando, mismo que está optimizado para el sistema operativo Linux, por lo cual la plataforma robótica se implementa en una máquina virtual. Para la integración con el reconocimiento de gestos se establecen lineamientos de las acciones que se realizan con los comandos obtenidos tanto de gestos como de la IMU del sensor MYO. El control de formación se basa en el seguimiento de cada agente hacia su posición, para lo cual se utiliza ROS Toolbox, herramienta que permite una fácil comunicación con el nodo maestro de ROS y Simulink. Por lo que, en la máquina virtual se desarrolla una interfaz para la ejecución de la escena multi-agente y en MATLAB otra para el control de las acciones comandadas. Las pruebas realizadas se basan en las formaciones realizadas para verificar el algoritmo de control implementado empleando el sistema integrado de reconocimiento de gestos, cuyos resultados se centran en el éxito en el comando del sistema multi-agente mediante el gesto.

PALABRAS CLAVE: Reconocimiento de Gestos, MYO Armband, Sistema multi-agente, Control de formación, ROS Toolbox, TutleBot3.

ABSTRACT

The research project PIGR-19-07 has developed hand gesture recognition algorithms that are designed by employing the MYO Armband sensor and MATLAB on Windows operating system. To verify its applicability, this project uses this system to command the change of the formation shapes of a robotic multi-agent system.

This system is implemented in a virtual environment, so the robotic simulation platform is chosen based on the robotic agents and the way they are commanded. The Gazebo simulator facilitates to launch and work with virtual TurtleBot3 robots as system agents and uses the ROS platform for their command, which is optimized for Linux, so the robotic platform is implemented in a virtual machine. The guidelines for the integration with the hand gesture recognition system are established by taking into consideration the actions to be performed with the commands obtained from both: gestures and the IMU of the MYO sensor. The control is based on the tracking of each agent towards its position, so the ROS Toolbox available in MATLAB-Simulink is used for this implementation. This tool allows the communication with the ROS master node using blocks available for Simulink models. An interface for the execution of the multi-agent scene is developed in the virtual machine and another one in MATLAB for the control of the commanded actions. The carried-out tests are based on verifying the performance of the implemented formation algorithm working together with the hand-gesture recognition system and their results focus on the success of commanding the multi-agent system.

KEYWORDS: Hand-gesture Recognition, MYO Armband, Multi-Agent System, Formation Control, ROS Toolbox, TurtleBot3.

1 INTRODUCCIÓN

El proyecto PIGR-19-07 “Reconocimiento de Gestos de la Mano Usando Señales Electromiográficas e Inteligencia Artificial y su Aplicación para la Implementación de Interfaces Humano-Máquina y Humano-Humano.”, se basa en la creación y/o mejoramiento de los algoritmos para el reconocimiento de gestos utilizando el sensor Myo Armband y las señales que este proporciona [1]. Mediante el sistema desarrollado se reconocen cinco gestos de la mano, los cuales son: Wave In (mano en ángulo de 90° hacia adentro), Wave Out (mano en ángulo de 90° hacia afuera), Fist (puño), Pinch (pellizco) y Open (mano abierta) [1]. Actualmente dicho proyecto se encuentra en una fase en la cual se ha logrado este objetivo y se ha visto la necesidad de buscar su aplicabilidad en diversos campos. Uno de estos es el planteado en el presente trabajo, que se enfoca en emplearlo para el comando de un sistema robótico multi-agente, teniendo como antecedentes su empleo en manipuladores robóticos y plataformas móviles, utilizando ya sea simuladores o prototipos de equipos, respectivamente [2], [3].

El sistema multi-agente que se plantea permitirá observar la factibilidad de uso del sistema de reconocimiento de gestos en un entorno robótico con una plataforma más compleja, como lo es un sistema compuesto por varios robots. En este sentido, en el presente trabajo, no solo se establece una escena multi-agente, en específico, la misma permitirá mostrar la adaptabilidad de comunicación del sistema de gestos con robots optimizados en un sistema operativo distinto, a través de un entorno virtual. El desarrollo del sistema de control y su interconexión con el reconocimiento de gestos del Proyecto de Investigación PIGR-19-07 dará una visión de las potenciales aplicaciones, basándose en el control de formaciones, ya que se podrá comprobar que sistemas robóticos complejos, como lo es un sistema multi-agente, responden a los comandos que el reconocimiento de gestos puede ofrecer como referencia.

Un sistema multi-agente se puede definir como el comando de grupos de robots a través de reglas para lograr cumplir tareas que necesitan desarrollarse en conjunto. Este grupo tiene la característica de coordinarse de una manera distribuida y descentralizada, además de estar compuesto por miembros casi idénticos [4]. Para la definición del entorno multi-agente se plantea la utilización de una plataforma de simulación que utiliza el sistema operativo robótico (ROS, por sus siglas en inglés) para implementar la escena de varios robots en un entorno de trabajo. Varias aplicaciones que permiten realizar esto son CoppeliaSim, Gazebo, Webots, SwarmBot3D, entre otras, mismas que tienen distintos fines dentro de la simulación robótica [4].

Por ende, este trabajo se centra en la implementación de un sistema multi-agente, el mismo que tendrá como objetivo lograr distintas configuraciones de formaciones de los agentes que lo conforman, a través del comando mediante el acoplamiento con el sistema de reconocimiento de gestos. Además, se plantea el movimiento en conjunto del grupo de robots, por lo cual en el componente también intervienen las señales inerciales que también entrega el sensor Myo Armand, aparte del comando por gestos que otorga el programa de reconocimiento de gestos del Proyecto de Investigación PIGR-19-07.

1.1 OBJETIVO GENERAL

Utilizar el sistema de reconocimiento de gestos empleado en el proyecto de investigación PIGR-19-07 para el comando de un sistema multi-agente.

1.2 OBJETIVOS ESPECÍFICOS

1. Revisar bibliografía relacionada a los sistemas multi-agente, así como de los agentes que los pueden conformar, los simuladores que permitan la creación de este tipo de escena y del sistema de reconocimiento de gestos basado en el sensor Myo Armband desarrollado en el Proyecto de Investigación PIGR-19-07.
2. Seleccionar el simulador robótico más adecuado para el diseño e implementación de la escena y el sistema multi-agente.
3. Definir un algoritmo de control para la formación del sistema multi-agente e implementarlo en MATLAB, el cual se conecte con la escena virtual desarrollada en el simulador.
4. Integrar el sistema de reconocimiento de gestos al sistema multi-agente virtual para el comando del mismo, y desarrollar una interfaz en MATLAB que permita la visualización del gesto reconocido y el estado del sistema multi-agente.
5. Realizar varias pruebas y analizar los resultados para comprobar un adecuado sistema de control del sistema multi-agente virtual con el comando del sistema de reconocimiento de gestos.

1.3 ALCANCE

- Se realiza una revisión bibliográfica sobre el sistema de reconocimiento de gestos y el sensor Myo Armband, utilizado en el desarrollo del Proyecto de Investigación PIGR-19-07, esto ya que son una parte esencial para el comando

del sistema propuesto tanto por los gestos como por las señales inerciales que se pueden obtener de este sensor.

- Se realiza una revisión bibliográfica acerca de sistemas multi-agente, en particular conformados por agentes terrestres; así como de los simuladores robóticos disponibles, de los cuales se escoge el de mayores comodidades para desarrollar la escena multi-agente virtual y comunicarse con MATLAB, ya sea mediante código o utilización de un Toolbox disponible.
- Se define un sistema multi-agente virtual, con robots terrestres como agentes, con un mínimo de 3 agentes, y definiendo la actuación, tanto en formación y movimiento, del sistema dependiendo el comando que se le dé, empleando el sensor Myo Armband.
- Se define y diseña el algoritmo de control, para el control de formación, para implementarlo a través de MATLAB mediante una interfaz de comunicación hacia la plataforma de simulación de la escena virtual que permitirá la adquisición y transmisión de datos entre ellos.
- Se diseña una interfaz en MATLAB, para obtener una visualización del estado del sistema multi-agente, del gesto reconocido, y del controlador.
- Se integra el sistema de reconocimiento de gestos como aquel que da los comandos para las diferentes acciones del sistema multi-agente y se evalúa el funcionamiento del sistema en conjunto para verificar el seguimiento de los lineamientos predefinidos.
- Se realizan varias pruebas para observar la integración del sistema multi-agente con el sistema de reconocimiento de gestos, además de obtener resultados que muestren errores de posición, velocidad e índices de desempeño del sistema multi-agente, cuyos resultados pueden ser visualizados en el simulador y la interfaz gráfica en MATLAB.

1.4 MARCO TEÓRICO

1.4.1 SISTEMA DE DETECCIÓN DE GESTOS DEL PROYECTO PIGR-19-07

El desarrollo de los sistemas de detección de gestos es un campo de interés para el desarrollo del presente trabajo. Si bien existen algunas formas para su detección tales como utilizar reconocimiento a través de imagen [5], guantes u otros tipos de sensores, en este caso se utiliza un brazalete que consta de sensores electromiográficos [1]. Las señales

que producen este tipo de sensores se basan en la contracción de los músculos, por lo cual estas señales son las que se someten al proceso dentro del patrón para reconocimiento de los gestos.

El proyecto de investigación grupal PIGR-19-07 “Reconocimiento de Gestos de la Mano Usando Señales Electromiográficas e Inteligencia Artificial y su Aplicación para la Implementación de Interfaces Humano-Máquina y Humano-Humano.”, ha desarrollado algoritmos donde se utilizan las señales electromiográficas del sensor Myo Armband [1]. Por tanto, en esta sección se abarca información acerca de este dispositivo tipo brazalete que hace la función del hardware del sistema de reconocimiento de gestos, y del software desarrollado por el proyecto PIGR-19-07.

1.4.1.1 Sensor Myo Armband

El sensor Myo Armband es un dispositivo para el reconocimiento de gestos distribuido por su empresa creadora North (en sus inicios Thalmic Labs) a partir del 2014. Actualmente sus ventas y soporte han parado desde octubre de 2018 y su empresa de origen North fue adquirida por parte de Google en junio de 2020, sin embargo, aún es posible el emplear el control basado en gestos que posee [6]. En la Figura 1.1 se puede encontrar una imagen de referencia del sensor Myo Armband.

Figura 1.1 Sensor Myo Armband [1]

Si bien el producto ha sido descontinuado aún es posible su uso y obtener información del mismo en base a fuentes de trabajos que también lo emplean y compilaron su información, como en [1], [7]. Algunas de sus características son:

- Dispone de 8 electrodos diferenciales de superficie
- Dispone de una IMU (acelerómetro, magnetómetro y giroscopio en x, y, z)
- Frecuencia de medición del EMG: 200 Hz
- Frecuencia de medición de la IMU: 50 Hz
- Cuantiza cada muestra con 8 bits
- Cuenta con comunicación Bluetooth.
- Batería recargable de ion litio de larga duración

- Realimentación háptica a través de vibraciones.
- Se ajusta al antebrazo de forma fácil gracias a sus bandas elásticas. Permite un perímetro del antebrazo en un rango de 19 a 34 cm.

El proyecto de investigación en sus programas ha incluido la manera de utilizar el sensor Myo Armband para poder realizar la correcta adquisición de datos y realizar las pruebas con el usuario colocándose correctamente el brazalete. En su programa para adquisición de datos podemos apreciar la Figura 1.2 dentro de la información que posee el usuario para colocar correctamente el sensor.

Figura 1.2 Colocación del sensor Myo Armband [1]

1.4.1.2 Software de Reconocimiento

Para el desarrollo del software para el reconocimiento de gestos (HGR, por sus siglas en inglés), independientemente del método que se utiliza, se siguen cuatro etapas. Las señales electromiográficas (con sus siglas EMG) que se obtiene de los sensores del Myo Armband distribuidos alrededor del antebrazo del usuario se tratan mediante estas cuatro etapas, implementadas en el sistema desarrollado en el proyecto PIGR-19-07. Las mismas son listadas a continuación [1]:

1) Extracción de la señal útil
2) Preprocesamiento
3) Extracción de características “features”
4) Clasificación

Antes de la ejecución del patrón de reconocimiento de gestos es necesario pasar por el escenario de adquisición de datos, el cual está formado por el sensor y el envío de las señales a través de Bluetooth. Para la correcta toma de datos el programa realiza una sincronización utilizando el gesto Wave Out, y en base a esta medición se ajustan los datos

que se obtienen del sensor. Para más información acerca de las distintas etapas se puede consultar las referencias [1], [8], [9].

El desarrollo del sistema toma en cuenta las variaciones intrapersonales, en el mismo sujeto, e interpersonales, entre varios sujetos, para volverse un sistema apto a cualquier usuario. Entre algunos de los resultados se tiene una precisión del 97% en una prueba a 12 usuarios de distinto género y edad entre 16 a 51 años [1]. Los gestos que se pueden reconocer mediante el uso son los cinco mencionados anteriormente: Wave In (mano en ángulo de 90° hacia adentro), Wave Out (mano en ángulo de 90° hacia afuera), Fist (puño), Pinch (pellizco) y Open (mano abierta) [1]. Además de las señales IMU que si bien no están procesadas en el sistema de reconocimiento es posible utilizarlas dependiendo de la aplicación.

1.4.2 SISTEMA MULTI-AGENTE ROBÓTICO

Una parte fundamental del trabajo es los sistemas multi-agente ya que es hacia donde se enfoca la integración del sistema de reconocimiento de gestos mencionado en la anterior sección. En base a los cinco gestos que reconoce el sistema y los datos proporcionados por la IMU del Myo Armband, se plantea el escenario del sistema multi-agente robótico para el desarrollo del componente del trabajo.

Una manera de entender un sistema multi-agente robótico es como un grupo que actúa en actividades donde se requieren habilidades sofisticadas o de precisión. El comportamiento del mismo se desarrolla en base a estímulos dados por el ambiente y sus capacidades. En estos sistemas el interés en su desarrollo ha pasado hacia una perspectiva global donde su diseño es cada vez más complejo y adaptable a requerimientos del usuario, tomando como partida y requerimiento inicial la misión principal asignada [10].

Los sistemas multi-agente se manejan en un entorno de arquitecturas basadas en agentes. El concepto de agentes es una manera de implementar unidades de autonomía en cada parte del sistema para que se pueda cooperar en resolver tareas más complejas. Acorde a [10]: "*Agentes* puede referirse a la descripción lógica de robots autónomos, o componentes funcionales, o facultades de un robot." Por lo que un sistema multi-agente se puede plantear como el conjunto de varios robots como agentes, así como un solo robot que tenga distintas facultades a las cuales se les pueda denominar como agentes para cooperar hacia un mismo objetivo, esto se ilustra en una imagen representativa en la Figura 1.3, en la cual se denota la colaboración hacia un solo fin.

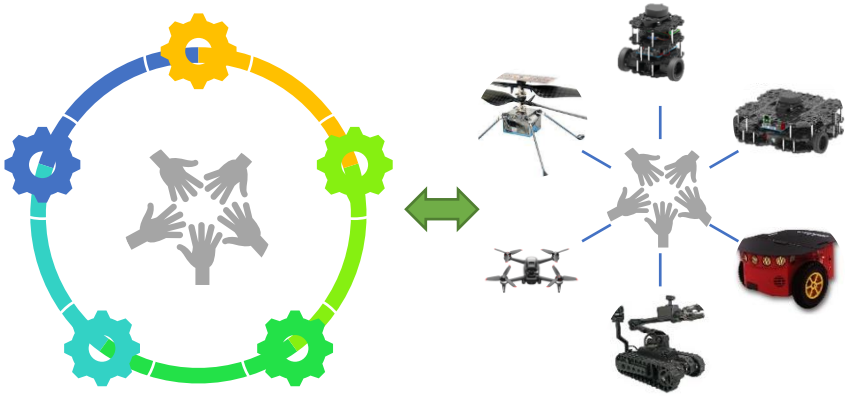


Figura 1.3 Relación entre agentes [Fuente Propia]

Un sistema multi-agente puede estar conformado por agentes diferentes entre sí, o agentes casi idénticos. En este sentido, en el presente trabajo se enfoca hacia un sistema conformado por robots casi idénticos, los cuales pueden conformar un sistema tipo enjambre robótico. Como se menciona en [4], un enjambre robótico se caracteriza por coordinar grandes grupos de robots relativamente simples mediante el uso de reglas locales. La taxonomía, es decir clasificación, que se puede dar a los grupos de robots, se basa en el tipo de características que puede tomar un sistema, y existen diversas formas de estructurar esa taxonomía según las diversas literaturas. Aquella que se presenta por niveles es planteada por Iocchi et al. en [4], la cual se presenta en la Figura 1.4, en la cual además se puede observar en un tono más oscuro el tipo de características que toma un sistema tipo enjambre robótico (swarm-robotic system). El nivel de cooperación indica la tarea en común que poseen los agentes, el nivel de conocimiento acerca de si el robot sabe de la existencia de otros, el tercer nivel es acerca de la coordinación donde se diferencia la profundidad de la relación con el ambiente, y en el último nivel el cual es la organización se identifica si será un sistema centralizado, con un robot comandando a los demás, o uno descentralizado, en el cual los robot tiene la autonomía que se espera alcanzar en este tipo de sistemas robóticos [4].

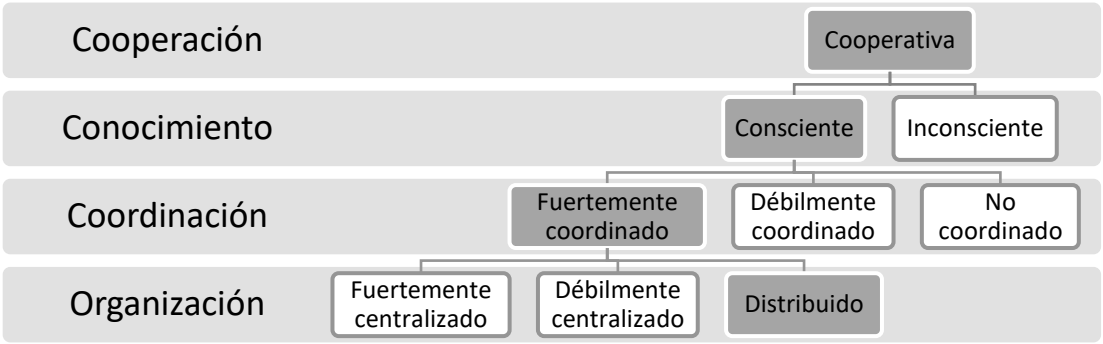


Figura 1.4 Taxonomía de un sistema multi-agente adaptado en base a [4]

Con base en lo planteado, para el presente trabajo se establece el uso de robots terrestres como agentes para el sistema. Dentro de estos robots se posee los robots con locomoción de tracción diferencial, que son el tipo de robot terrestre más común; a continuación, se amplía este concepto.

1.4.3 ROBOT DE TRACCIÓN DIFERENCIAL - TURTLEBOT3

Un robot de tracción diferencial es aquel que presenta su sistema de locomoción en base a dos ruedas, y son aquellos que no pueden cambiar su dirección de movimiento sin evitar la realización de giros, es decir no puede realizar movimientos laterales por la restricción de sus llantas. Este tipo de restricción se denominan no-holonómica, es decir, la existencia de una restricción que impide tener la capacidad de llegar desde cualquier punto inicial a otro cualquiera final del sistema de coordenadas [11], [12].

Dentro de los posibles robots comerciales de tracción diferencial en el presente trabajo se utilizan robots TurtleBot3 en forma de enjambre donde existe cooperación, conocimiento uno de otros, coordinación entre ellos y una organización distribuida [4]. Por lo que las tareas que puede resolver un enjambre no se basan en un solo individuo para completarlas, sino una coordinación entre agentes lo que requiere, por ejemplo, un control de formaciones.

TurtleBot es una plataforma robótica estándar de bajo costo y con un software de código abierto. El primer modelo fue creado por Melonee Wise y Trully Foote de Willow Garage en 2010 y ha estado a la venta desde 2011 [13]. En la actualidad se ha desarrollado la versión 4 de la familia de TurtleBots, estando la primera y segunda versión de las series discontinuadas. Tomando en cuenta lo nuevo que es actualmente el lanzamiento de la cuarta serie, en el presente trabajo se enfoca al uso del TurtleBot3.

TurtleBot3 es un robot pequeño, accesible, programable y basado en el sistema operativo robótico (ROS) para su uso en la educación, investigación, entretenimiento o desarrollo de prototipos de producto. Esta versión de esta plataforma robótica presenta una reducción del tamaño y precio, sin sacrificar la eficiencia y calidad, al mismo tiempo que se ofrece la capacidad de expansión al personalizarlo [13]. Por otro lado, los TurtleBot3 son robots móviles terrestres robustos, disponen de sensor de 360° y vienen en varios modelos, en la Figura 1.6 se puede visualizar algunas de sus características generales. Adicionalmente, esta plataforma móvil robótica está optimizada para ser usado con ROS/Linux, por lo que es necesario evaluar la necesidad de interconectar con MATLAB(Windows), donde está desarrollado el sistema de gestos, o revisar la posibilidad de trabajar con la versión que dispone ROS para Windows [14].

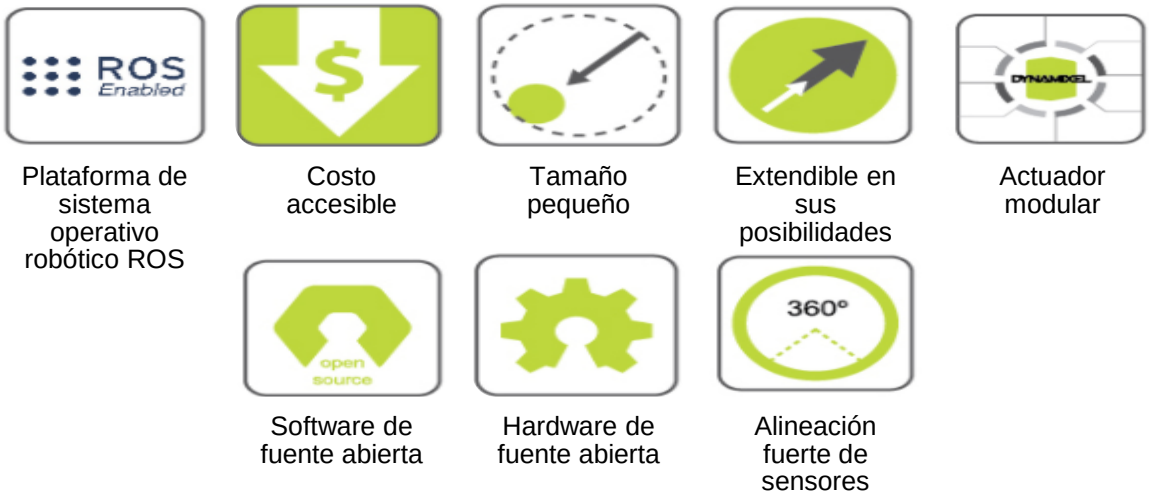


Figura 1.6 Características del TurtleBot3, adaptado en base a [15]

Los TurtleBot3 vienen en distintos posibles modelos, los cuales se diferencia por su construcción, lo cual afecta a las características de uso que puede tener cada uno de ellos. Entre los principales aspectos que se diferencian entre ambos son los actuadores, el SBC (Single Board Computer) que es el circuito principal, y los sensores. Los distintos modelos son: Burger y Waffle Pi, en la Figura 1.7 se puede observar imágenes de cada tipo de TurtleBot3 [15]. Los detalles como las dimensiones y especificaciones físicas de las características de cada modelo se pueden encontrar dirigiéndose hacia la referencia [15].

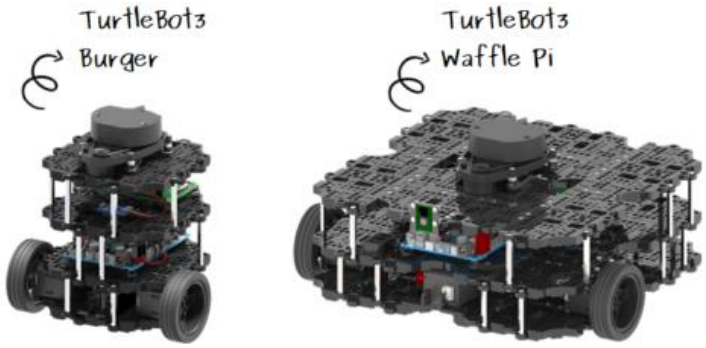


Figura 1.7 Tipos de robot TurtleBot3 [15]

En la Tabla 1.1 se pueden apreciar los datos relevantes del modelo Burger, el cual se utiliza en el presente trabajo, y que son necesarios tomar en cuenta para una aplicación correcta respecto a los límites que se deben considerar dentro del controlador.

Tabla 1.1. Datos del TurtleBot3 Burger

Tipo	Unidad	Valor
Velocidad lineal máxima	[m/s]	0,22
Velocidad angular máxima	[rad/s]	2,84
Radio exterior	[mm]	105

A continuación, se abarca el análisis de la cinemática que posee un robot del tipo tracción diferencial, tomando en cuenta sus características de movimiento en el plano terrestre y dimensiones que influyen en el mismo.

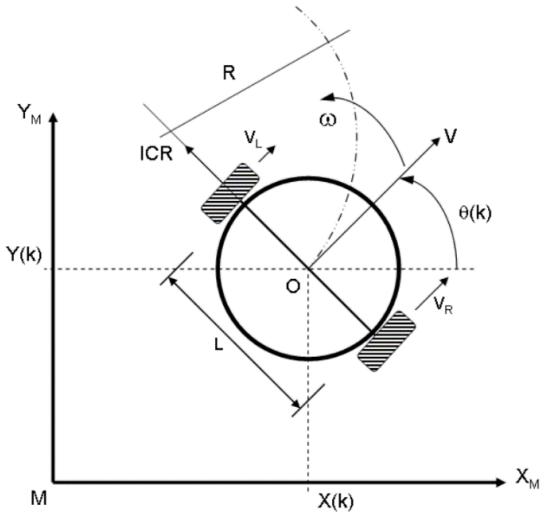


Figura 1.5 Esquema de Robot de Tracción Diferencial [12]

En la Figura 1.5 se muestra el esquema en el que se basa el modelado de este tipo de robot, donde se tienen como coordenadas base para un robot al vector $q = (q_1 \quad q_2 \quad q_3)^T = (x \quad y \quad \theta)^T$, que representan los tres grados de libertad del robot, siendo estos su posición en el plano (x, y) y la orientación es determinada por el ángulo θ con respecto al eje *X* [11]. En base a estas variables, el modelo cinemático del robot de tracción diferencial se define de la siguiente manera:

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

Donde v representa la velocidad lineal del centro de rotación del robot, y ω su velocidad angular. Además, es necesario conocer el valor del radio r de las llantas, así como también la separación transversal L que hay entre ambas. El radio permite relacionar la velocidad lineal y la angular de cada llanta como se denota en las Ecuaciones (1.2):

$$\begin{aligned} v_R &= \omega_R . r \\ v_L &= \omega_L . r \end{aligned} \tag{1.2}$$

Mientras que la distancia L permite obtener la velocidad angular del robot en función de la velocidad de cada llanta, lo cual se muestra en la Ecuación (1.3), y tomando en cuenta la Ecuación (1.2), se puede expresar la ecuación en función de las velocidades angulares en lugar de las lineales al aplicarse a la velocidad de cada llanta, en caso de requerirse que se exprese en función de esas variables.

$$\omega = \frac{v_R - v_L}{L} \tag{1.3}$$

Además la velocidad también se puede expresar como el promedio de las velocidades lineales de cada llanta, lo cual se expresa en la Ecuación (1.4), con lo cual se tiene un modelo cinemático inicial referenciado en el mismo robot.

$$v = \frac{v_R + v_L}{2} \tag{1.4}$$

Debido a que como se mencionó posee restricciones no-holonómicas, se introduce el concepto de Centro de Rotación Instantáneo (IRC), en base al cual se determina el punto central en el que se realizan las rotaciones y se ve los puntos a los que puede llegar, siendo que el ICR viene definido por:

$$ICR = (x - Rsin(\theta), y + Rcos(\theta)) \tag{1.5}$$

Siendo R el radio de curvatura instantáneo de la trayectoria del robot y relativo al punto medio del eje de las ruedas. Si su valor es cero indica que el robot está girando en su propio centro [12]. Su expresión viene definida por:

$$R = \frac{L}{2}\frac{v_R + v_L}{v_R - v_L} \tag{1.6}$$

Teniendo en cuenta las Ecuaciones (1.1), (1.3), (1.4) y la equivalencia que establece la Ecuación (1.2) se presenta el modelo en base a las velocidades angulares de cada llanta de la siguiente manera:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} r/2\cos(\theta) & r/2\cos(\theta) \\ \mathrm{r}/2\sin(\theta) & r/2\sin(\theta) \\ r/L & -r/L \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{1.7}$$

Una manera de comprobar lo que se realiza antes de implementarlo es la utilización de simuladores, por lo cual el trabajo se centra en la utilización de la versión virtual del robot TurtleBot3 en un sistema multi-agente planteado dentro de una plataforma de simulación.

1.4.4 PLATAFORMAS DE SIMULACIÓN ROBÓTICAS

El presente trabajo se va a realizar en un entorno virtual, por lo tanto, es fundamental la elección de una plataforma robótica que se adapte al sistema multi-agente, en base al robot de tracción diferencial TurtleBot3 que será el que conforme los agentes del mismo, y sea capaz de comunicarse con el sistema de reconocimiento de gestos para el comando. Las plataformas de simulación robóticas se han desarrollado con distintos fines cada una, por lo que hay una variedad para analizar cuál es la mejor según los requisitos de uso que se le dará, un punto clave es aquel en el que el comando de los agentes sea intuitivo y fácil

de realizar. Un simulador bien diseñado permite probar de manera rápida algoritmos, diseñar robots, probar regresiones de actuación y entrenar sistemas de inteligencia artificial usando escenarios realistas [16].

Entre algunos de los simuladores que se pueden encontrar están CoppeliaSim, Gazebo, Webots, Microsoft Robotics Studio, SwarmBot3D, entre otros, las cuales son diferentes en aspectos como sistema operativo recomendado, simulación en 2D o 3D, el número de robots que puede manejar al mismo tiempo en una simulación, entre otros aspectos como gratuidad, licencias y costos [4].

La plataforma que se utiliza en el presente trabajo es Gazebo debido a las características del mismo para su funcionamiento en el entorno de ROS. Este es un punto importante en la decisión ya que es por medio de este sistema que se comunicará con los robots de tracción diferencial descritos en la sección anterior, TurtleBot3.

1.4.4.1 ROS-Gazebo

Con la mentalidad de crear una herramienta que permita la creación de más innovaciones en la robótica de manera rápida se creó el Sistema Operativo Robótico (ROS, por sus siglas en inglés), el cual es un conjunto de librerías de software y herramientas de fuente abierta que ofrecen ser una plataforma estándar poderosa para el desarrollo de proyectos robóticos [14]. A medida que avanza el tiempo también lo hace el desarrollo de la tecnología, es así que existen varias versiones de ROS que están disponibles a la vez. Entre las recomendadas en este momento se encuentran Ros Noetic Ninjemys, ROS Foxy Fitzroy y ROS Galactic Geochelone, los dos últimos siendo recomendados ya para últimas versiones de ROS2, mientras que ROS Noetic lo es para ROS1. Por tanto, se ve conveniente el uso de esa versión, y se toma en cuenta que trabaja en el sistema operativo Ubuntu Linux [17]. En la Figura 1.8 se puede visualizar lo mencionado anteriormente en referencia a las versiones actuales que recomienda ROS en su página oficial en el momento.

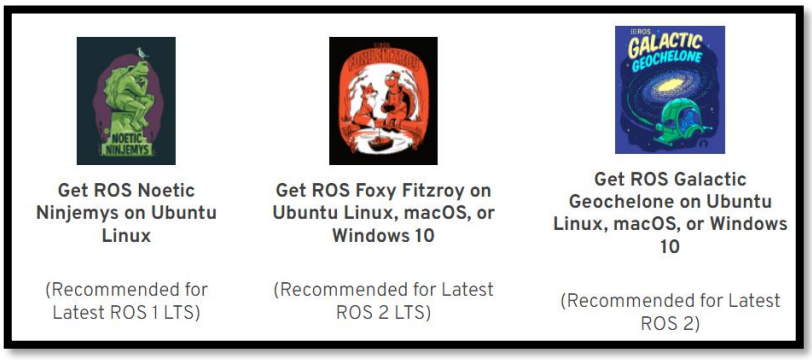


Figura 1.8 Versiones recomendadas de ROS [17]

Una de las plataformas de simulación que trabaja mediante ROS y en la cual se puede encontrar la inclusión del robot TurtleBot3 es Gazebo. Este entorno ofrece la posibilidad de simular grandes grupos de robots, controlarlos y probarlos ya sea en entornos del interior o exterior. Además, se considera como un sistema robusto con gráficos de alta calidad e interfaces convenientes para la programación al disponer de librerías de fuente abierta que encapsulan todo lo esencial [16].

Gazebo presenta una manera simple de integrar el modelo del robot TurtleBot3 en base a tutoriales, cuyos datos resumidos se encuentran en el ANEXO I, y además se puede codificar de manera simple el entorno con los agentes identificados en el mismo. En la Figura 1.9 se puede observar un ejemplo del entorno de la plataforma Gazebo en el sistema operativo Ubuntu Linux.

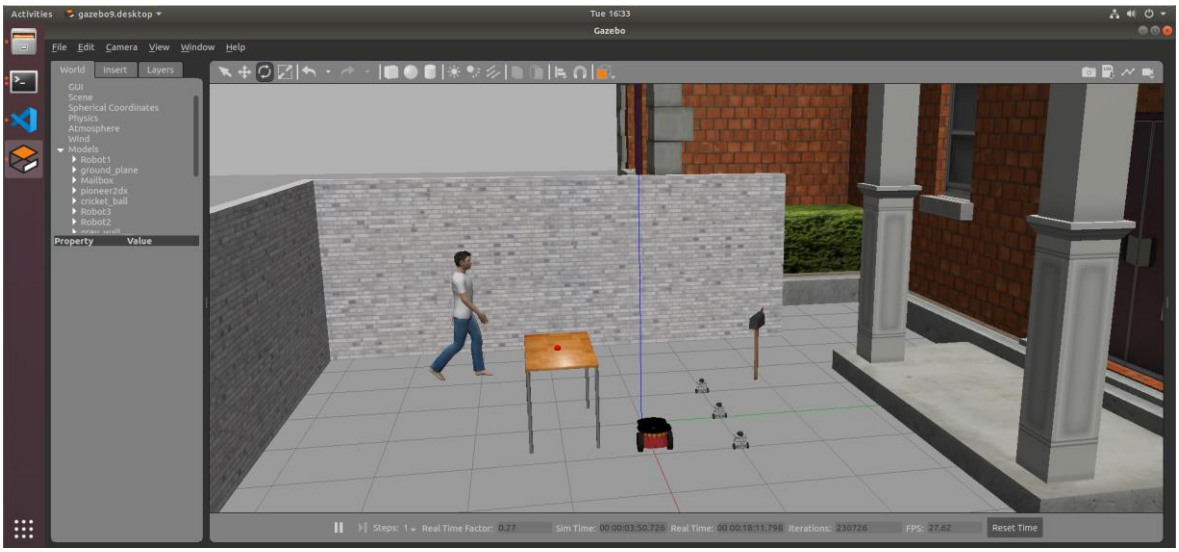


Figura 1.9 Entorno de Gazebo [Fuente Propia]

1.4.4.2 ROS Toolbox

El utilizar la configuración ROS-Gazebo presta varias ventajas en la simulación del sistema multi-agente al tener varias prestaciones relacionadas a TurtleBot3. Tomando en consideración la integración que debe realizarse con MALTAB para realizar el control e interconectar con el sistema de reconocimiento de gestos, se incluye en el presente trabajo la utilización de una de las herramientas que ofrece MathWorks: ROS Toolbox, entre otras relacionadas con la robótica.

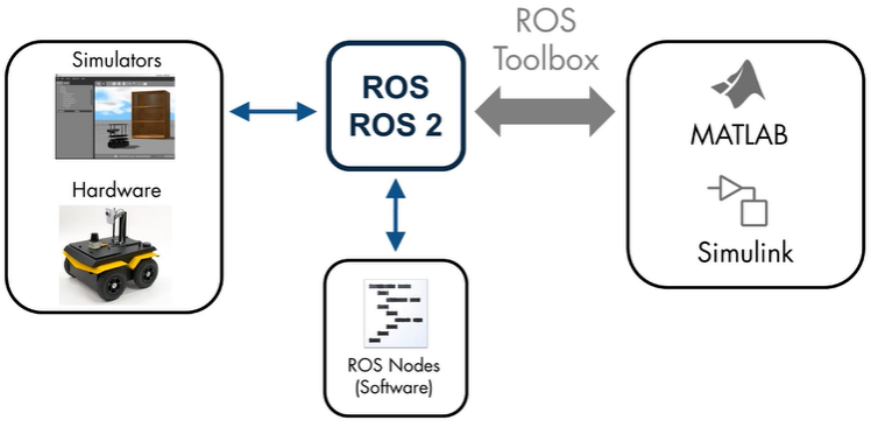


Figura 1.10 Esquema de una red ROS interconectada con ROS Toolbox [18]

ROS Toolbox es una herramienta que proporciona una interfaz para conectar MATLAB y Simulink con ROS, permitiendo crear una red de nodos ROS. Un esquema de una red de nodos ROS en conjunto con la toolbox se puede observar en la Figura 1.10. La toolbox incluye funciones de MATLAB y bloques de Simulink con los cuales se puede importar, analizar y exportar datos, además de la conexión con una red ROS en tiempo real con lo cual se permite la comunicación vía mensajes tanto de envío como recepción [18], mediante lo cual se puede realizar la conexión con la plataforma de simulación Gazebo donde se encuentra el sistema multi-agente.

1.4.4.3 Máquina Virtual

Para el presente trabajo se necesita ejecutar MATLAB dentro del sistema operativo Windows, en el cual está realizado el sistema para la detección de gestos, mientras que la plataforma de simulación robótica Gazebo trabaja con ROS optimizado en el sistema operativo Linux. Tomando esto en cuenta, se establece la implementación de una máquina virtual en la cual se ejecute la plataforma de simulación, siendo posible de esta manera que esta se ejecute en el sistema operativo Linux, mientras MATLAB se ejecuta en Windows.

Una máquina virtual, o VM por sus siglas en inglés, es una implementación de software para emular una máquina física, es decir toma recursos físicos de la máquina para trabajar sin embargo se es capaz de manipular como algún otro tipo de archivo. Los archivos fundamentales de una máquina virtual son *el archivo de configuración*, en el cual se definen los recursos como cantidad de procesadores o memoria que utilizará al encenderse, y *el o los archivos de disco* en los cuales se encuentran los archivos del sistema operativo, sus aplicaciones y datos [19].

Para el manejo de los archivos de una máquina virtual y su ejecución se han desarrollado aplicaciones que permiten la ejecución de la máquina con sus características, sea cual sea su sistema operativo, tal como si se corriera una aplicación [20]. En esta línea se tiene a VMware Workstation, el cual corre bajo un sistema operativo Windows o Linux, siendo este el sistema operativo host de la máquina donde se ejecutará la VM, pudiendo el sistema operativo de esta ser distinto ya que no son conscientes uno de otro [19]. Por tanto, una de las ventajas que podemos encontrar en el uso de este tipo de máquinas es que poseen sus propias configuraciones, incluso en cuanto al sistema operativo en el cual trabajan. Una característica importante es la capacidad de movilizar el estado de la máquina al importarla, por lo que se virtualizan todos sus datos y se puede llevar hacia otro equipo siempre que esta cuente con los recursos físicos que va a consumir la máquina virtual en el momento de encenderla.

En la Figura 1.11 se puede observar una representación de ambas máquinas que operan en un sistema operativo distinto y las actividades que cada una de ellas realiza. La comunicación entre las dos partes se realiza a través de toolbox de MATLAB, esto y más detalles acerca de la VM utilizada se abarcan en el Capítulo 2 de Metodología.

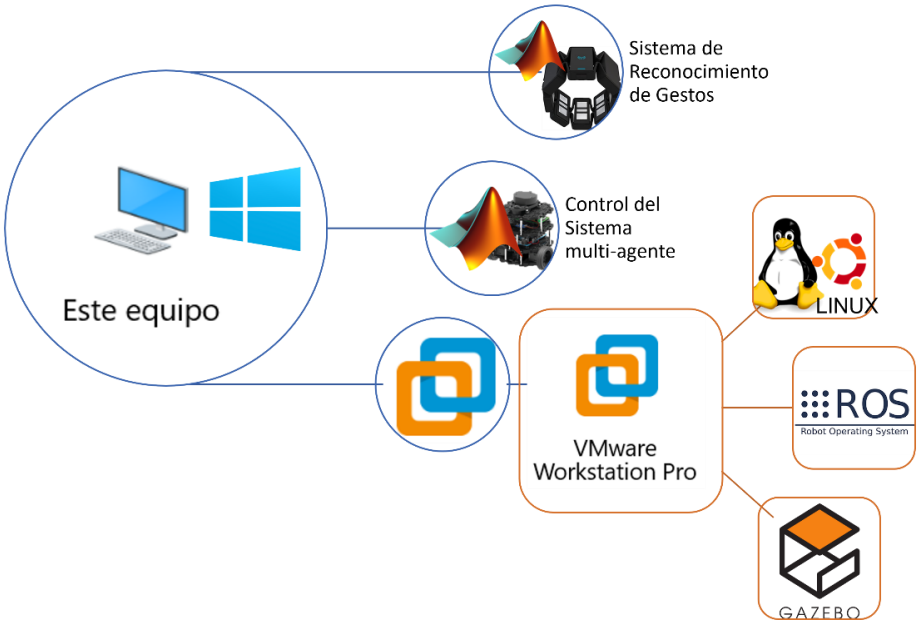


Figura 1.11 Representación de la máquina principal junto a la máquina virtual [Fuente Propia]

2 METODOLOGÍA

En el presente capítulo se abarcan secciones clave para el desarrollo del presente trabajo de titulación. Tomando en cuenta la información que se recopiló previamente de literatura disponible en el marco teórico, presentado en la Sección 1.4, se establecen las bases para el desarrollo del trabajo tomando en cuenta las siguientes partes: el sistema de reconocimiento de gestos (HGR), el control del sistema multi-agente y su implementación en la plataforma de simulación. En la Figura 2.1 se muestra el diagrama que representa el esquema propuesto para este trabajo.

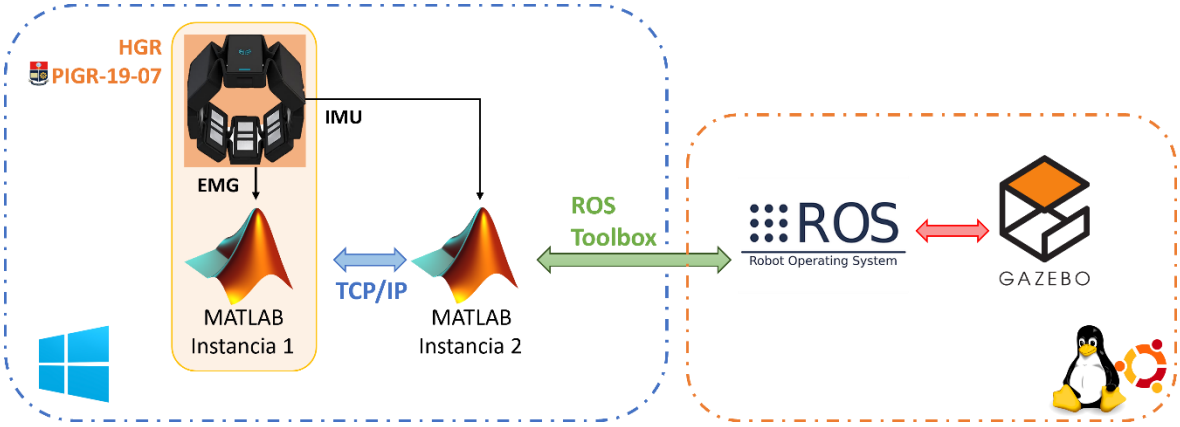


Figura 2.1 Diagrama de la aplicación [Fuente Propia]

Describiendo con mayor detalle lo expuesto en el diagrama de la Figura 2.1, en el recuadro con fondo naranja, se observa el sistema de reconocimiento de gestos (HGR) del proyecto PIGR-19-07, conformado por el sensor Myo Armband (hardware) y la aplicación desarrollada en MATLAB (software), que se detalla en la Sección 1.4.1. Para relacionarse con el control para el sistema multi-agente, se posee una interfaz que comunica la Instancia de MATLAB del sistema de reconocimiento (1) con la Instancia donde se desarrolla el control (2), la cual también procesa las señales inerciales IMU del sensor Myo Armband. Posteriormente, para la comunicación con la plataforma de simulación se hace uso del ROS Toolbox de MATLAB que permite establecer conexión con la plataforma ROS de la máquina virtual, donde se encuentra la escena multi-agente en el simulador Gazebo. Por ende, los temas que aborda el presente capítulo se dividen en la implementación del sistema multi-agente robótico, la integración con el sistema de reconocimiento de gestos, con la configuración de las interfaces de comunicación, el control de formaciones del sistema multi-agente mediante el sistema de reconocimiento de gestos, y finalmente, la interfaz para el usuario.

2.1 SISTEMA MULTI-AGENTE ROBÓTICO

El sistema multi-agente robótico que se plantea está conformado por la escena dentro de la plataforma robótica y su control a través de la integración con el sistema de reconocimientos de gestos del Proyecto de Integración PIGR-19-07 que se describió previamente en la Sección 1.4.1.

2.1.1 IMPLEMENTACIÓN DE LA ESCENA

Se recomienda empezar con la implementación de una máquina virtual funcional que cumpla los requerimientos para que trabaje la plataforma de simulación. Una máquina virtual se la puede crear con la ayuda de la aplicación VMware, que se mencionó anteriormente en la Sección 1.4.4.2, para lo cual se deben tener en cuenta los archivos principales que esta necesita: la imagen del sistema operativo base en el que se trabaje, las aplicaciones que se requiera instalar, y demás recursos que sean necesarios. O bien se puede descargar una máquina funcional de una página confiable y realizarle ajustes a la necesidad, de esta manera se asegura la funcionalidad y se ahorra tiempo de instalaciones.

En el presente trabajo se optó por personalizar la máquina virtual dispuesta dentro de la documentación de ayuda que ofrece MathWorks, la distribución que posee la máquina provista puede ser actualizada al ser un sitio oficial que debe mantenerse actualizado. La máquina virtual tiene un sistema operativo Ubuntu 20.04.3 LTS (Focal Fossa), la versión más actualizada. Puede trabajar en múltiples plataformas, es decir el sistema operativo principal del computador puede pertenecer a Windows, Mac o Linux. Dentro de los softwares instalados en la máquina descarga están ROS 2 Foxy, ROS Noetic, el simulador robótico Gazebo en su versión 11.0.0 y ejemplos de mundos para simular con TurtleBot3 [21]. De lo mencionado, en el presente trabajo se utiliza ROS Noetic y Gazebo 11.0.0, además, se instala Visual Studio Code para tener un mejor acceso a los archivos y una adaptación rápida al entorno y organización de los archivos. En la Figura 2.2 se puede observar una representación de lo anteriormente descrito.

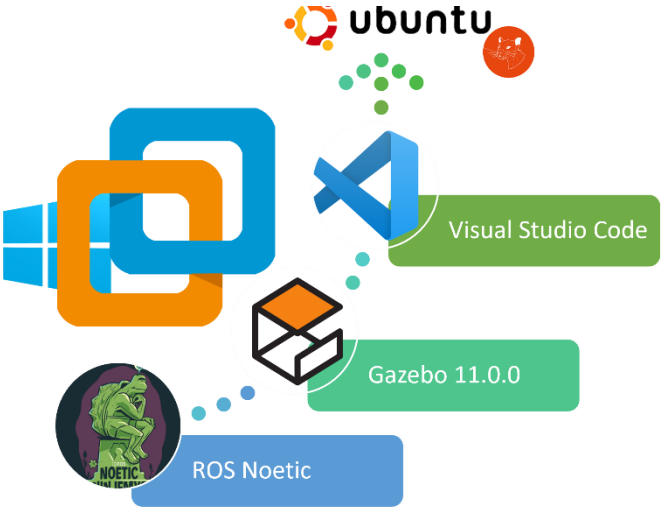


Figura 2.2 Distribución de la máquina virtual [Fuente Propia]

Una vez se posee el entorno para trabajar, se procede a desarrollar la escena del sistema multi-agente. Para esta se plantea un espacio libre de obstáculos donde solo se coloquen los agentes del sistema ya que el enfoque para el presente trabajo es el control de formación, por lo cual no se ve necesario la colocación de un obstáculo para cumplir el objetivo. La cantidad de agentes robóticos en la escena es de tres TurtleBot3 Burger, este número es la base para dar al sistema multi-agente varias posibles formaciones.

Para que sea posible ejecutar la escena multi-agente se siguen las instrucciones clave que se posee en la *Compilación de Comandos de Tutoriales ROS-Gazebo* en el Anexo I y se obtiene la estructura de los archivos necesarios que se observa en la Figura 2.3, los archivos que se muestran son aquellos que se programan para luego ejecutarlos. Al plantearse una escena libre de obstáculos, el mundo de Gazebo como tal se encontrará vacío, por ende, luego del comando que permite el inicio de *empty_world.world* en los ejecutables principales, se incluye a los archivos que tienen la programación para lanzar los robots dentro del mismo.

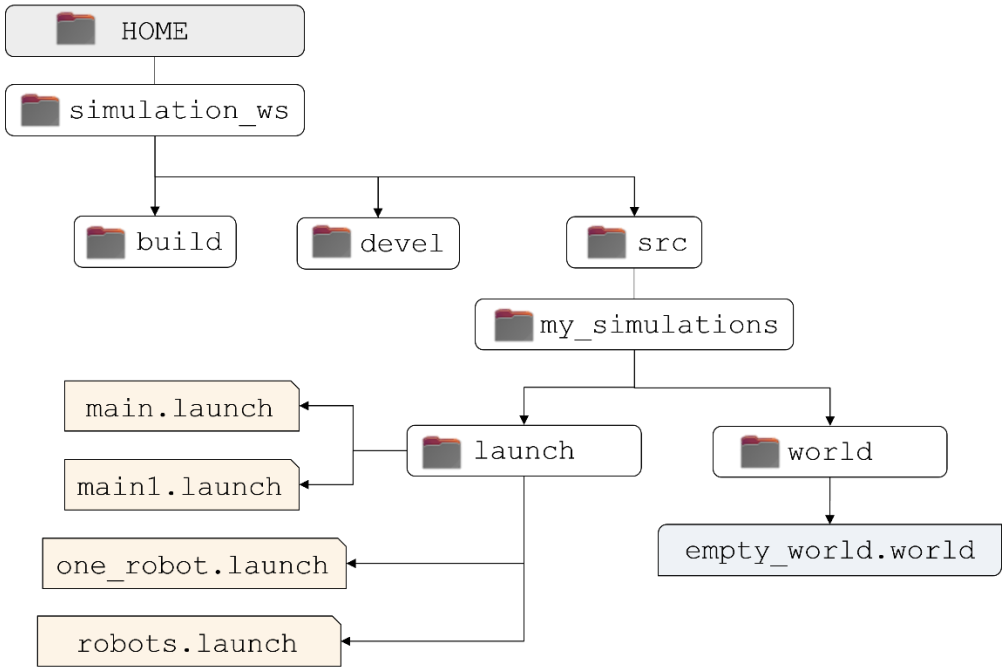


Figura 2.3 Estructura de archivos desarrollados para la implementación de la escena multi-agente [Fuente Propia]

El archivo *main1.launch* se utiliza para ejecutar el mundo y colocar solamente un robot, mientras que en *main.launch* se realiza la colocación de los tres robots TurtleBot3 Burger que se propone en el presente trabajo. La siguiente explicación se realizará tomando como base el archivo *main.launch* ya que permite observar la utilización de todos los archivos creados. El orden de lanzamiento de los archivos *.launch*, tomando en cuenta cómo van conteniéndose entre sí, es de la siguiente manera: **main→robots→one_robot**. A continuación, se muestran los códigos implementados para cada uno de ellos y se resalta los lugares donde se van relacionando, además del lugar donde se inicia el mundo vacío en Gazebo dentro del archivo *main.launch*.

main.launch

```
<launch>
  <param name="/use_sim_time" value="true" />
  <!-- start world -->
  <node name="gazebo" pkg="gazebo_ros" type="gazebo"
   args="$(find my_simulations)/world/empty_world.world" respawn="false"
output="screen" />
  <!-- include our robots -->
  <include file="$(find my_simulations)/launch/robots.launch"/>
</launch>
```

robots.launch

```
<launch>
  <param name="robot_description"
    command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_burger.urdf.xacro" />

  <!-- BEGIN ROBOT 1-->
  <group ns="robot1">
    <param name="tf_prefix" value="robot1_tf" />
    <include file="$(find my_simulations)/launch/one_robot.launch" >
      <arg name="init_pose" value="-x 0.5 -y 0.5 -z 0.0 -Y 0.0" />
      <arg name="robot_name"  value="Robot1" />
    </include>
  </group>

  <!-- BEGIN ROBOT 2-->
  <group ns="robot2">
    <param name="tf_prefix" value="robot2_tf" />
    <include file="$(find my_simulations)/launch/one_robot.launch" >
      <arg name="init_pose" value="-x 0.0 -y 0.5 -z 0.0 -Y 1.7" />
      <arg name="robot_name"  value="Robot2" />
    </include>
  </group>

   <!-- BEGIN ROBOT 3-->
  <group ns="robot3">
    <param name="tf_prefix" value="robot3_tf" />
    <include file="$(find my_simulations)/launch/one_robot.launch" >
      <arg name="init_pose" value="-x -0.5 -y 0.5 -z 0.0 -Y 3.4" />
      <arg name="robot_name"  value="Robot3" />
    </include>
  </group>
</launch>
```

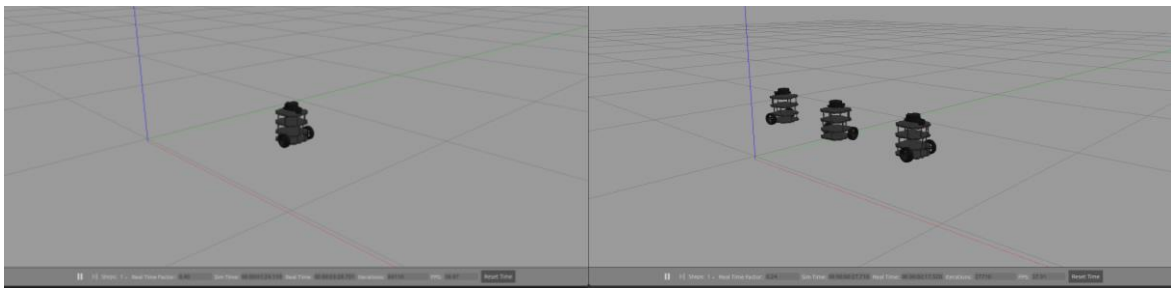
one_robot.lauch

```
<launch>
    <arg name="robot_name"/>
    <arg name="init_pose"/>

    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
     args="-urdf -model $(arg robot_name) $(arg init_pose) -param
/robot_description"
     respawn="false" output="screen" />

    <node pkg="robot_state_publisher" type="robot_state_publisher"
          name="robot_state_publisher" output="screen"/>
</launch>
```

Como se observa en el código de *robots.launch*, cada vez que se requiera colocar un robot es necesario crear un “grupo”, con lo cual se crea un nodo, y esto permite que cada robot tenga sus tópicos bajo su propio nombre y sea posible comunicarse con el robot que se coloca, además para cada uno se puede definir la posición y orientación inicial. En la Figura 2.4 se observa las escenas que se muestran en Gazebo al ejecutar los archivos *main1.launch* y *main.launch*, respectivamente en las partes (a) y (b) de la figura.



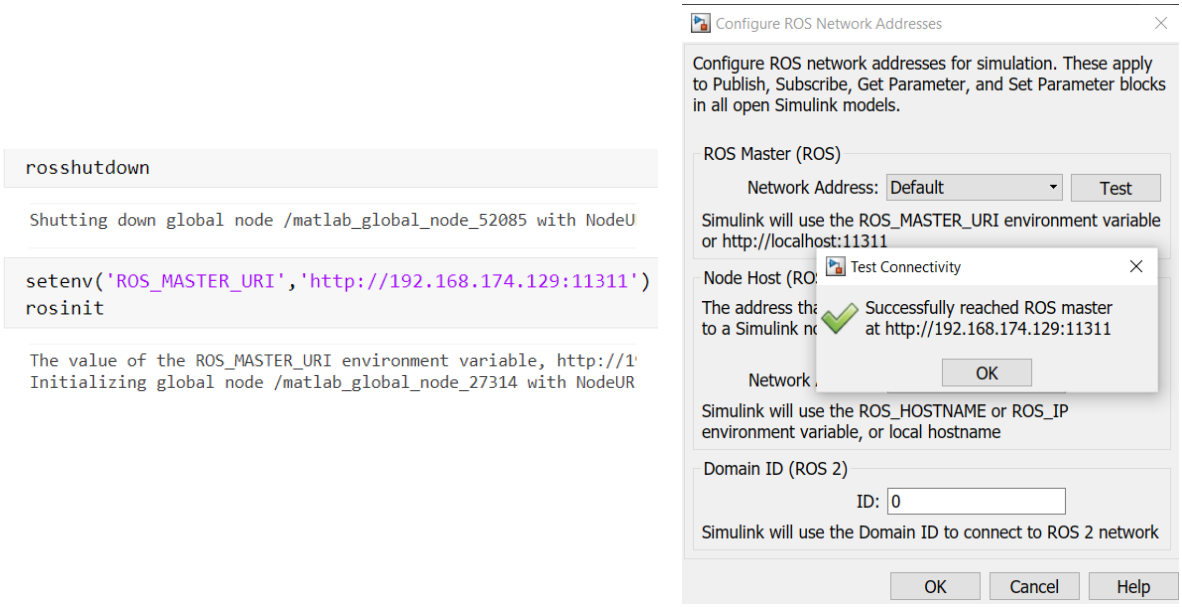
(a) (b)

Figura 2.4 Escenas ejecutadas en Gazebo con (a) un robot, y (b) tres robots TurtleBot3 Burger [Fuente Propia]

2.1.2 INTERFAZ DE COMUNICACIÓN ENTRE MATLAB Y GAZEBO

Para configurar la comunicación entre el control para el sistema multi-agente y su respectiva escena en el simulador Gazebo en la máquina virtual, se hace uso de la herramienta ROS Toolbox. El programa de control se realiza dentro de Simulink de MATLAB, en donde se puede acceder a las utilidades que presenta la Toolbox, como lo es la configuración de la dirección de red de ROS, con la cual se mantendrá la comunicación con la plataforma ROS que posee la máquina virtual. Dentro de dicha configuración se

incluyen variables Default, las cuales se pueden modificar mediante líneas de código, de esa manera se actualizan los valores con los que realiza la conexión del modelo de Simulink.



(a) (b)

Figura 2.5 Configuración de red de ROS en (a) líneas de código y (b) modelo de Simulink [Fuente Propia]

En la Figura 2.5 (a) se pueden observar las líneas de código para configurar las variables que se usan por defecto en la conexión con la red ROS y en la Figura 2.5 (b) la ventana de configuración en el modelo de Simulink a que se accede siguiendo: **Simulation→Prepare→ROS Network**, junto con la prueba de conexión que se puede realizar en ella. Cabe mencionar que el maestro de ROS de la máquina virtual estará activado siempre y cuando se esté ejecutando la escena en el simulador Gazebo, con aquello y la configuración mencionada en MATLAB se obtiene una conexión exitosa.

2.2 INTEGRACIÓN CON EL SISTEMA DE RECONOCIMIENTO DE GESTOS

El sistema de reconocimiento de gestos (HGR) que se utiliza en el presente trabajo es el desarrollado en el Proyecto de Investigación PIGR-19-07, el mismo que envía sus resultados de gesto de dos maneras: por números mediante la variable `ges` o a través de una cadena de caracteres o string mediante la variable `newGesture`. La relación de cada variable con los gestos disponibles se observa en la Figura 2.6.

Gesto / Variable						
newGesture	waveOut	waveIn	fist	open	pinch	noGesture
ges	1	2	3	4	5	6

Figura 2.6 Relación entre el gesto y las variables de resultado del sistema de HGR [22]

Además, se utiliza la señal inercial IMU que se puede obtener del brazalete Myo Armband. En este caso, para definir el ingreso de comandos se toma como base lo desarrollado en [23], donde se establecen límites en los ángulos de Euler que se obtienen como parte de la IMU para definir las zonas en las que se está ejecutando cada comando. Los límites y comandos generados en base a los ángulos de Euler se muestran en la Tabla 2.1 y la Figura 2.7.

Tabla 2.1 Comandos generados a partir de la IMU del Myo Armband [23]

Ángulos de Euler	Superior	Inferior	Límites
Pitch	UP	DOWN	$\pm 45°$
Yaw	RIGHT	LEFT	$\mp 45°$
Roll	FORWARD	BACKWARD	$\pm 60°$

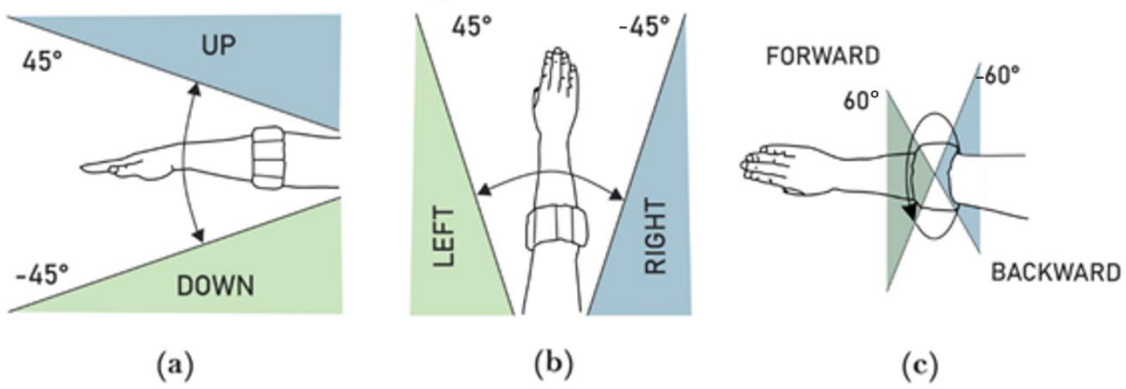


Figura 2.7 Comandos generados a partir de la IMU del Myo Armband (a) UP y DOWN, (b) LEFT y RIGHT, (c) FORWARD y BACKWARD [23]

Debido a los cálculos y procesamiento que es necesario al ejecutar el sistema HGR se presenta un tiempo de retado de 0.5 hasta 1 segundos entre la realización del gesto y su reconocimiento [22]. Por tanto, al ser distintas las velocidades de ejecución entre el sistema

HGR y el control del sistema multi-agente, se los ejecuta de manera independiente en distintas instancias de MATLAB y se los integra mediante una interfaz de comunicación.

2.2.1 LINEAMIENTOS DE COMANDOS DEL SISTEMA MULTI-AGENTE

Para integrar el sistema multi-agente con el sistema de reconocimiento de gestos se requieren establecer los posibles comandos que se pueden obtener tanto de gestos como en relación a la IMU del Myo Armband. Estos se dividen en la definición de acciones en base a los comandos para el control de formación y para el seguimiento de trayectoria como grupo. Por ende, se utilizan los resultados de los gestos para comandar el tipo de formación, y los comandos generados por la IMU para el movimiento del sistema multi-agente.

Para organizar la navegación entre las acciones que se requieren realizar con el sistema multi-agente se definen tres niveles: **Menú Inicial**, **Control de formación** y **Movimiento de trayectoria grupal**. En Menú Inicial se selecciona lo que se quiere realizar con el sistema multi-agente, entre cambiar hacia otra formación o mover la actual, lo que implica el siguiente nivel al que se dirige. Para el Control de formación se definen seis opciones en cuanto a la formación que se desea tenga el sistema multi-agente. Tomando en cuenta que el sistema posee 3 agentes, estas seis opciones son las siguientes:

1.- Columna
2.- Fila
3.- Formación V
4.- Formación V inversa
5.- Formación L
6.- Formación L inversa

En cuanto al Movimiento de trayectoria grupal, no se poseen opciones sino se plantea el movimiento de toda la formación del sistema multi-agente en función a los comandos que se obtienen de la IMU del sensor. En la Figura 2.8 se puede apreciar la organización de los niveles de selección que se han descrito anteriormente, junto a las opciones de cada uno.

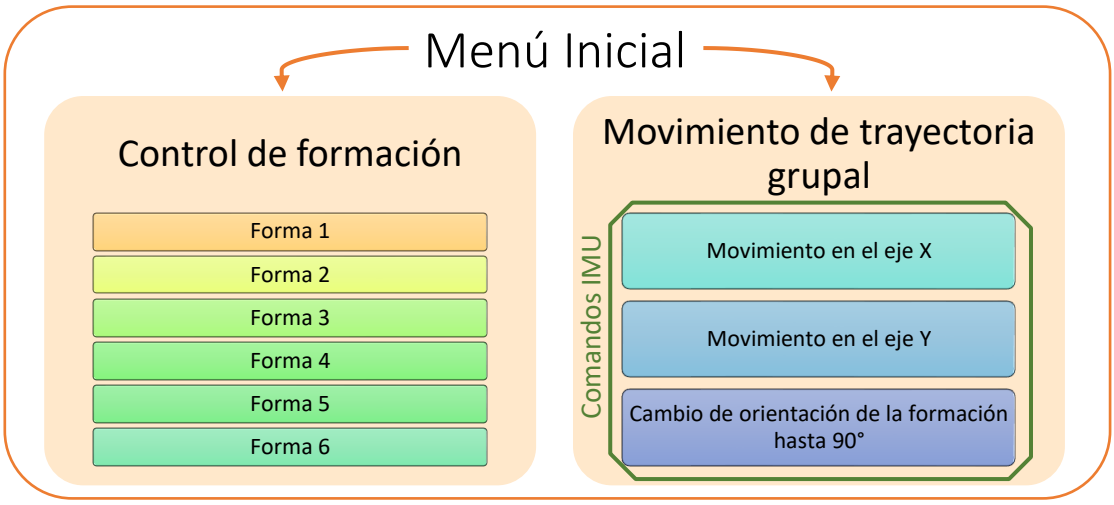


Figura 2.8 Niveles y opciones de selección del sistema [Fuente Propia]

2.2.1.1 Acciones mediante comandos del sistema de reconocimiento de gestos

El sistema de detección de gestos (HGR) se emplea para definir acciones generales de navegación entre niveles y dentro de los mismos. Por ende, se dispone de cinco comandos que se pueden relacionar con una acción dentro de cada nivel, cuatro de ellos se utilizan para la navegación, selección y retroceso en las opciones, mientras que el gesto Pinch se utiliza para una acción específica dentro del Control de formación.

Los gestos Wave Out y Wave In se utilizan para cambiar la opción que se va a seleccionar, el primero cambia hacia la opción a la derecha y el otro hacia la de izquierda. En el nivel Menú Inicial las opciones entre las que se navega son Control de formación y Movimiento de trayectoria grupal. En el nivel Control de formación las opciones que se dispone son desde la formación 1 hasta la formación 6 que se mencionaron anteriormente. En el tercer nivel, al no tener opciones, estos gestos no se utilizan.

Luego de elegir la opción que se desea aplicar en el nivel que se encuentre, para aceptarla y seleccionarla se utiliza el gesto Fist. Esto es aplicable en los niveles de Menú Inicial y Control de formación ya que son aquellos que poseen opciones. El gesto Open tiene la misma utilidad en cualquiera de los niveles que se encuentre, la cual es regresar hacia el nivel anterior, se ve la necesidad de esta acción para poder navegar entre los niveles y corregir la selección en caso de ingreso erróneo por error humano. Además, en el nivel de Control de formación, se utiliza el gesto Pinch para realizar un cambio en la distancia que tendrán los agentes entre sí, una vez se realiza se acercan o alejan entre ellos. En la Figura 2.9 se observa un resumen de las acciones mencionadas, cada una relacionada a un gesto que se obtienen como comandos por parte del sistema de reconocimiento de gestos.

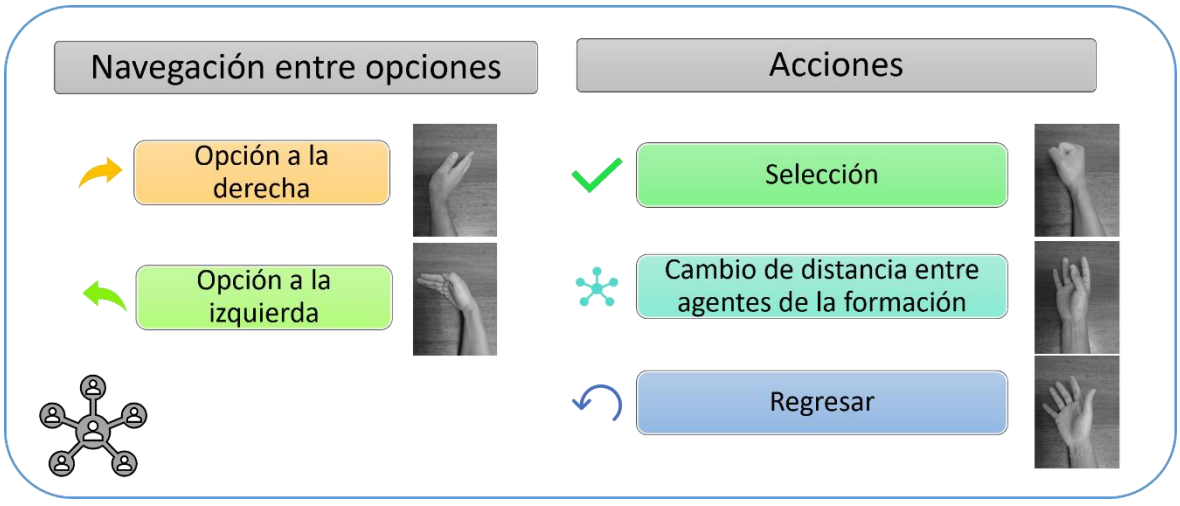


Figura 2.9 Acciones comandadas por el sistema HGR [Fuente Propia]

2.2.1.2 Acciones mediante comandos de la IMU del sensor Myo Armband

El único nivel en el cual se utiliza los comandos que se obtienen de las señales inerciales (IMU) del sensor es en Movimiento de trayectoria grupal. Los comandos UP, DOWN, LEFT y RIGHT se emplean para establecer la dirección de movimiento de la formación en el plano X-Y, mientras que a través de los comandos FORWARD y BACKWARD se cambia la orientación de la formación. Los movimientos del antebrazo que afectan a las señales IMU se expresan con ángulos de Euler, por tanto, en la Figura 2.10 se observan las acciones que comanda la IMU en Movimiento de trayectoria grupal junto a aquel que los causa y a qué ángulo de Euler está relacionado, lo cual se menciona al inicio de la Sección 2.2.

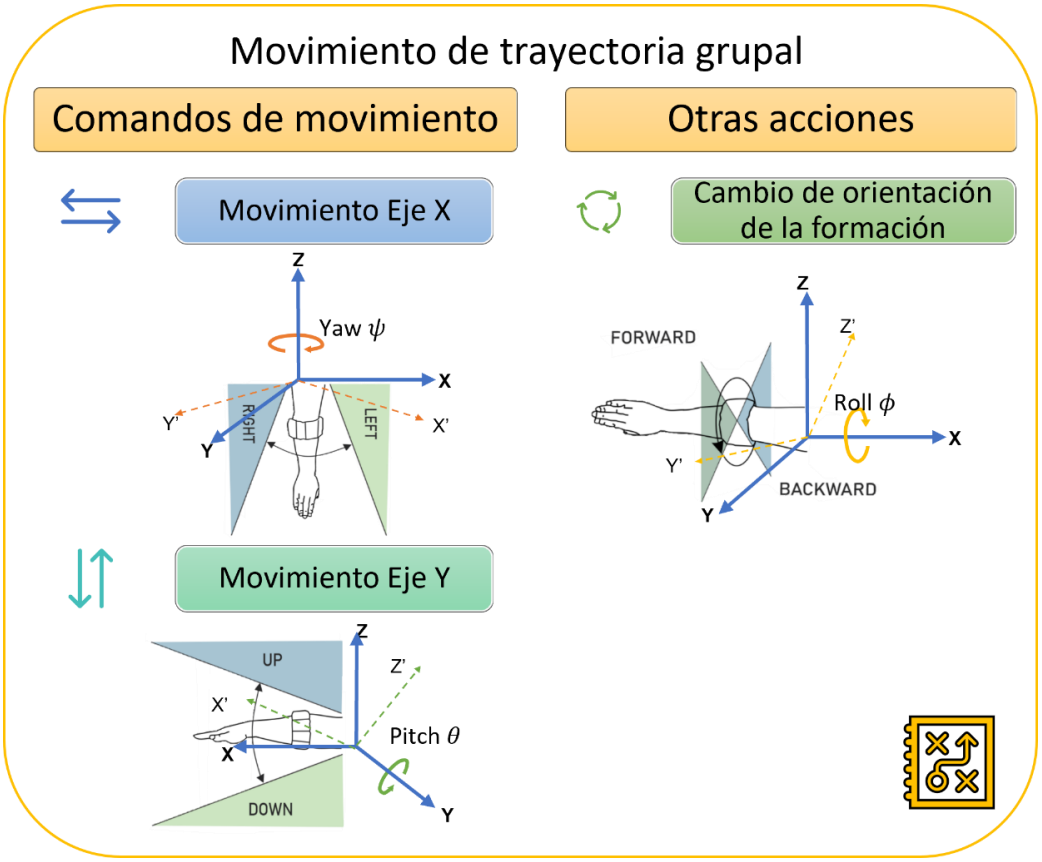
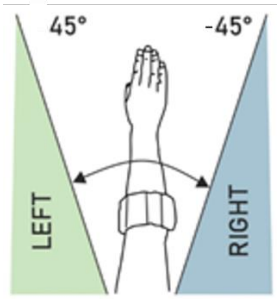
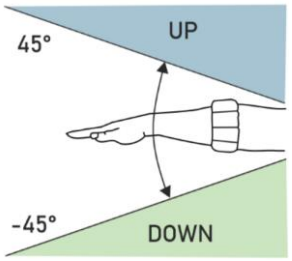
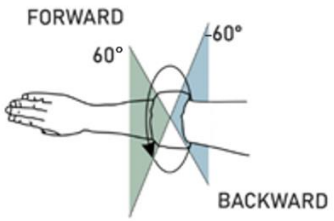


Figura 2.10 Acciones comandadas por la IMU del sensor Myo Armband [Fuente Propia]

En la Tabla 2.2 se muestra la relación del comando con cada acción, por lo que se observa el nivel donde se aplican las acciones que se describen y el comando mediante el cual se ejecuta dicha acción. En esta tabla se resume lo explicado en detalle a lo largo de esta sección respecto a los lineamientos establecidos. Para una mejor comprensión de los lineamientos planteados, en el Anexo II se incluye un diagrama de flujo donde se encuentra en detalle la funcionalidad del sistema HGR con el multi-agente.

Tabla 2.2 Comandos para las acciones de cada nivel

<table>
<tr><td>Nivel
Acción</td><td>Menú Inicial
Control de formación</td><td>Nivel
Acción</td><td>Movimiento de trayectoria grupal</td></tr>
<tr><td rowspan="2">Navegación entre las opciones a la derecha</td><td>WAVE OUT</td><td rowspan="2">Movimiento en el Eje X</td><td>Condiciones del ángulo Yaw</td></tr>
<tr><td>  </td><td> 
</td></tr>
<tr><td rowspan="2">Navegación entre las opciones a la izquierda</td><td>WAVE IN</td><td rowspan="2">Movimiento en el Eje Y</td><td>Condiciones del ángulo Pitch</td></tr>
<tr><td>  </td><td> 
</td></tr>
<tr><td rowspan="2">Selección</td><td>FIST</td><td rowspan="2">Cambio de orientación de la formación</td><td>Condiciones del ángulo Roll</td></tr>
<tr><td>  </td><td> 
</td></tr>
<tr><td>Nivel
Acción</td><td>Control de formación</td><td>Nivel
Acción</td><td>TODOS</td></tr>
<tr><td rowspan="2">Cambio de distancia entre agentes de la formación</td><td>PINCH</td><td rowspan="2">Regresar</td><td>OPEN</td></tr>
<tr><td>  </td><td>  </td></tr>
</table>

2.2.2 INTERFAZ DE COMUNICACIÓN ENTRE INSTANCIAS DE MATLAB

Como se mencionó anteriormente, la velocidad de ejecución del sistema HGR es lenta en comparación a la necesaria en el control del sistema multi-agente, por tanto, es necesario que funcionen de manera independiente, empleando dos instancias de MATLAB. Mientras los datos del sistema HGR se obtienen con retardos de entre 0.5 y 1 segundos, los comandos e información referentes al sistema multi-agente son enviados y recibidos más rápido y de manera simultánea para mantener un control adecuado, un valor promedio tomado en cuenta la memoria que usa el computador para el intercambio de datos es de 2 milisegundos.

En la Figura 2.11 se muestra un esquema en el cual se observa lo que se realiza en cada Instancia de MATLAB, las cuales se comunican a través del protocolo TCP/IP para el cual se configuran líneas de código para definir el cliente, el servidor y aquellas de envío y lectura de datos.

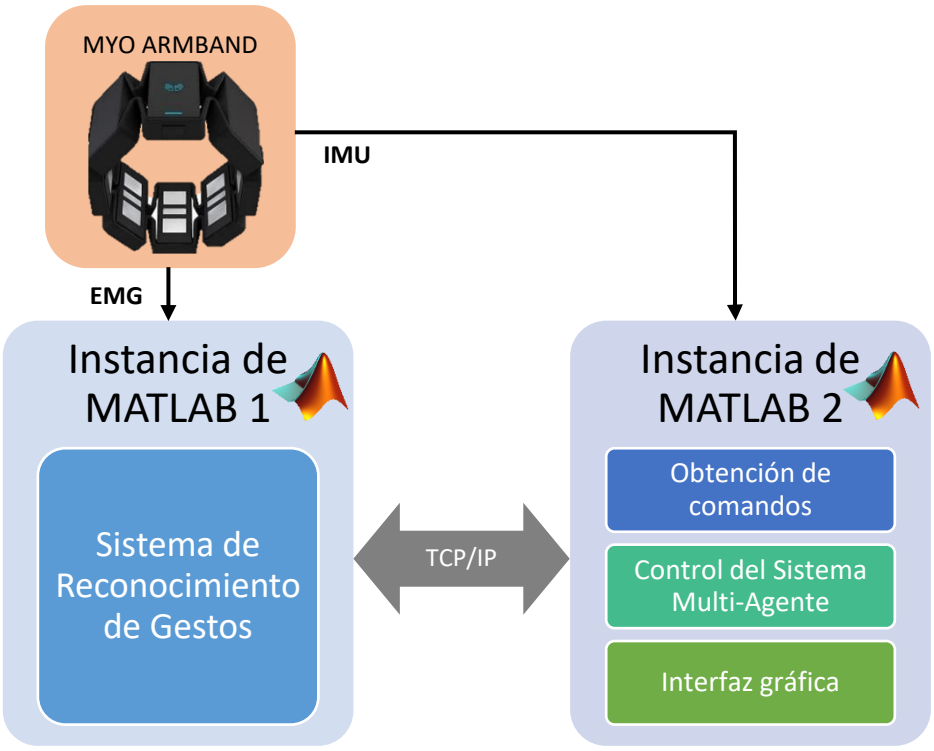


Figura 2.11 Esquema de comunicación y detalles de actividades en las instancias de MATLAB [Fuente Propia]

Dentro del protocolo TCP/IP para comunicar las dos instancias de MATLAB que se utiliza se establece el cliente y servidor como se muestra en la Tabla 2.3. En este caso, ya que ambas instancias se encuentran en el mismo computador, el cliente se conectará a "localhost", mientras que el servidor se establece para que cualquiera se pueda conectar

para tener un código general, de esta manera si se ejecuta las Instancias de MATLAB en distintos computadores, lo único que es necesario cambiar es la dirección de conexión del cliente, esto se deja como dato para futuros proyectos que se basen en este trabajo.

Tabla 2.3 Configuración para uso del protocolo TCP/IP

	Instancia	**Acción**
Cliente	Instancia 1 de MATLAB – Sistema HGR	Envío de dato de gesto
Servidor	Instancia 2 de MATLAB – Sistema de Control	Lectura de dato de gesto

El sistema de reconocimiento es el cliente en el protocolo de comunicación, el cual se encarga de enviar el dato del gesto que se ha realizado. Para colocar la información en el servidor es necesario que un cliente esté conectado, el servidor por sí solo no puede manejar información [24]. Para ello se especifica en el cliente el dato que se requiere enviar, el cual se manda hacia el servidor para que se pueda disponer de la información en el sistema de control. El código necesario en el cliente es el siguiente, donde `DATA` es aquello que se enviará.

```
client = tcpclient("localhost",3000)
client =
  tcpclient with properties:

             Address: 'localhost'
                Port: 30000
    NumBytesAvailable: 0

  Show all properties, functions

write(client,DATA,"uint8")
```

En el servidor al definir la dirección IP como “0.0.0.0” implica que aceptará el intento de conexión de cualquier cliente que provenga de la misma u otra máquina [24]. Al estar el servidor en el sistema de control, lo que se busca es leer el gesto realizado, por lo que si hay datos en el servidor que se hayan incluido por parte de un cliente, en este caso el sistema de reconocimiento, se los lee. El ejemplo de código para realizar lo descrito es el que se muestra a continuación, donde `data` es aquello que se recibirá.

```
%Servidor abierto a cualquiera en el puerto 3000
server = tcpserver("0.0.0.0",3000)
server =
  TCPServer with properties:
```

```
        ServerAddress: "0.0.0.0"
           ServerPort: 3000
            Connected: 0
        ClientAddress: ""
           ClientPort: []
    NumBytesAvailable: 0

  Show all properties, functions
```

```
if server.NumBytesAvailable~=0
data=read(server,server.NumBytesAvailable,"uint8");
end
```

2.3 CONTROL DE FORMACIONES

Para establecer el control de formaciones se plantean las formas que se podrán realizar, las mismas que se enumeraron anteriormente en la Sección 2.2.1, y se presentan en la Figura 2.12, donde se muestra el plano base respecto al origen en el cual se encuentra un líder virtual que actuará como centro para cada formación. Además, se incluye los robots TurtleBot3 Burger, que serán los agentes del sistema.

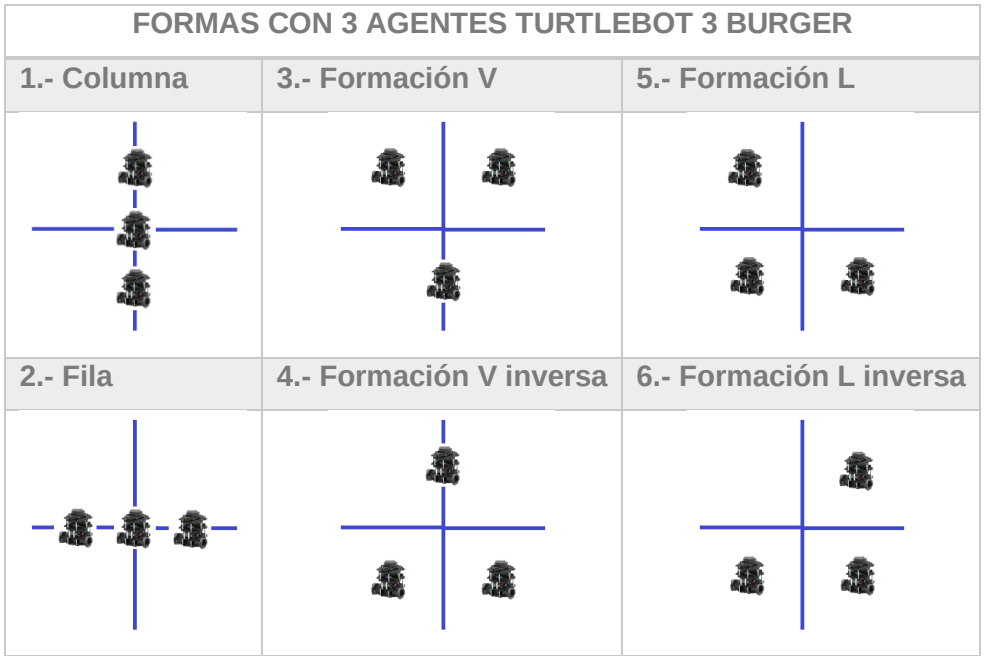


Figura 2.12 Formas disponibles para el sistema multi-agente [Fuente Propia]

2.3.1 SISTEMAS DE COORDENADAS DE UBICACIÓN

Para poder retroalimentar el control es necesario conocer la ubicación de los agentes, para lo cual se emplean las utilidades de ROS Toolbox que se explicarán en la Sección 2.3.3. En este caso se lee la odometría de cada robot, a través de la cual se obtiene la pose de cada uno, que está conformada por la posición y la orientación. La pose completa que se

recibe de la odometría conseguida incluye la posición en los ejes X, Y y Z, y la orientación representada en manera de Cuaterniones (Quat, abreviado de su nombre en inglés) , el cual se expresa en forma de un vector $Quat = [w\ x\ y\ z]$. La unidad de los cuaterniones es el método más robusto para representar la orientación, debido a que es más compacto, numéricamente estable y eficiente en su representación como matrices de rotación [25].

El estado de la pose que se necesita para el algoritmo de control incluye la ubicación en los ejes X y Y, para conseguir el estado en el plano del piso ya que al ser un robot terrestre no es posible elevarlo, y la orientación del mismo plano, lo cual expresa el ángulo Euler Yaw, por tanto, es necesario realizar un procesamiento de datos. En la Figura 2.13 se observa el esquema para obtener el estado de la pose de un robot. Acorde al mismo, se procede con la lectura donde se obtiene la pose del robot completa y se realiza el procesamiento de los datos para conseguir el estado de la pose del robot, como se requiere para el desarrollo del algoritmo de control.

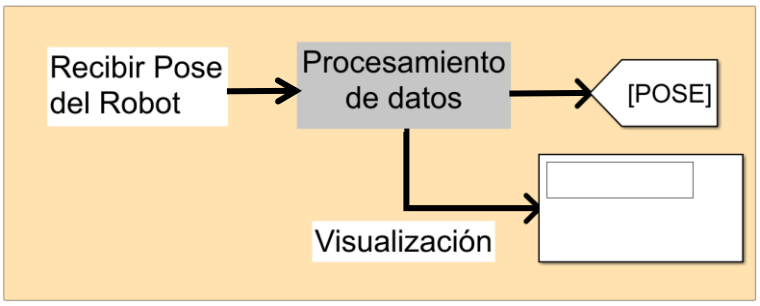


Figura 2.13 Diagrama de lectura de pose de un agente robótico terrestre [Fuente Propia]

Para poder utilizar la orientación como un ángulo que describa hacia donde observa el frente del robot, en el procesamiento se transforma las coordenadas de orientación que se reciben desde la unidad de Cuaterniones hacia el sistema de ángulos Euler. Para esto se emplea el bloque disponible en la librería de ROS Toolbox que permite transformar las coordenadas, para la cual hay que tomar en cuenta el ingreso correcto de los datos, es decir cumplir con el orden del vector $Quat = [w\ x\ y\ z]$, siendo este el dato que se obtiene de cada TurtleBot3. Además, cuando se utiliza el sistema de ángulos Euler es importante considerar el orden de las rotaciones ya que la interpretación cambia dependiendo de ello, el orden que se estará utilizando es ZYX, por lo que en términos de los ángulos se compone de yaw (ψ), pitch (θ) y roll (ϕ), por lo que el último paso en el procesamiento para obtener el ángulo es tomar el primer valor del vector conseguido de los ángulos Euler (ψ, θ, ϕ) [25]. En la Figura 2.14 (a) se puede observar la forma de rotación que se da dependiendo de cada ángulo para comprender su relación con los ejes.

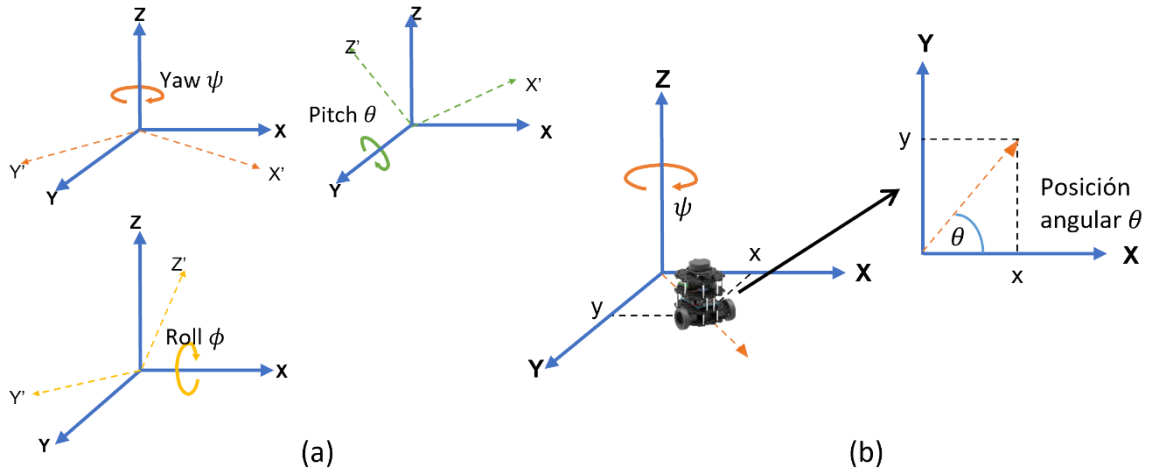


Figura 2.14 (a) Rotación de la orientación de las coordenadas dependiendo de cada ángulo de Euler (b) Representación de las coordenadas de un agente robótico terrestre en base a sus tres grados de libertad [Fuente Propia]

En la Figura 2.14 (b) se observa un agente en el plano y los estados necesarios para la implementación del control, que corresponden a los valores de las coordenadas base de un robot en sus tres grados de libertad $q = (x \ y \ \theta)^T$, como se mencionó en la Sección 1.4.3. En el procesamiento de los datos se realiza un cambio de coordenadas a partir de los Cuaternarios a los ángulos de Euler, y tomando en cuenta la aplicación en el presente trabajo, de los ángulos disponibles se utiliza el Yaw para determinar la orientación del robot, ya que describe la rotación respecto al eje Z y por tanto se determina la posición angular que tiene el robot terrestre en el plano X-Y. Definiéndose de esta manera el estado de la pose para cada robot de la siguiente manera:

$$Pose\_i = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} \tag{2.1}$$

Donde:

i es el número de agente robótico, $i \in N, i = 1,2,3$

$Pose\_i$ es la pose con los tres grados de libertad del agente robótico i

x_i es la posición en el eje X del robot i [m]

y_i es la posición en el eje Y del robot i [m]

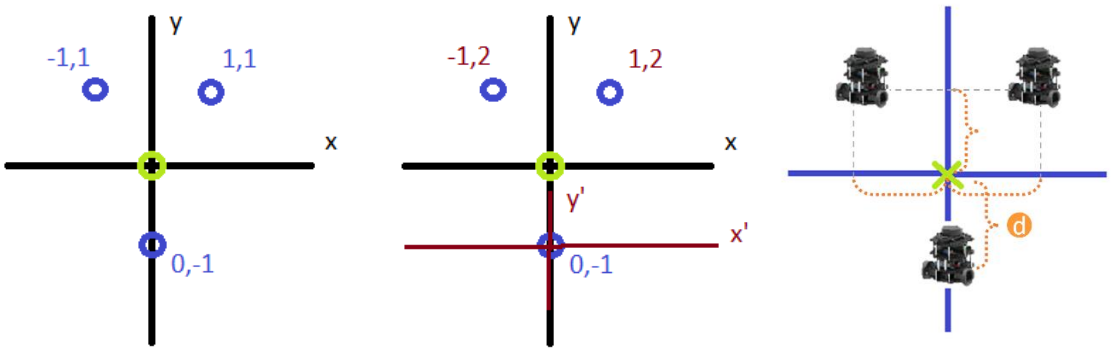
θ_i es la posición angular del robot i [rad]

2.3.2 ALGORITMO DE CONTROL DE FORMACIÓN

Este control se realiza una vez se ha escogido la formación que se desea conformen los 3 agentes, mediante la navegación entre las opciones y selección con el gesto Fist (puño). Para el seguimiento de cada agente a su posición designada se utiliza como referencia un plano, en cuyo origen se ubica un Líder Virtual, el mismo que no es un robot TurtleBot, sino la base para la formación de los agentes robóticos. El seguimiento de los agentes se puede realizar de dos maneras:

- **Modo 1:** Seguimiento de 3 agentes a un Líder Virtual
- **Modo 2:** Seguimiento de 2 agentes a un agente Líder Seguidor

En el primer caso todos toman sus posiciones relativas respecto al Líder Virtual que se establece como origen, mientras que en el segundo modo solo uno de los agentes basa su posición en el Líder Virtual y los otros dos agentes las basan en él, lo cual lo convierte en un agente Líder Seguidor. Un ejemplo de la diferencia de ambos modos se puede observar en la Figura 2.15, en donde el círculo color verde es el Líder Virtual, los de color azul son los agentes robóticos y se ubica al agente Líder Seguidor siendo el origen del plano rojo en la parte (b) de la figura. Además, los modos de formación que se plantean son conocidos como topologías tipo árbol dentro de la teoría de los grafos, que a la vez es una topología importante debido a que conecta a todos sus elementos con el menor número de caminos posibles [26].



(a) (b) (c)

Figura 2.15 Referencias para posición de los agentes, (a) Modo 1 (b) Modo 2. (c)Distancia *d* en cada eje para cada agente para su posición respecto al Líder Virtual [Fuente Propia]

El primer paso que se realiza antes de empezar la simulación empleando Simulink y Gazebo, es definir los estados de posición de cada robot relativos al Líder Virtual, ubicados a cierta distancia en cada eje respecto al Líder Virtual como se indica en la parte (c) de la Figura 2.15, por lo que se establece los siguientes parámetros:

$$h_i = [rx_i \quad ry_i] \rightarrow h = \begin{bmatrix} rx_1 & ry_1 \\ rx_2 & ry_2 \\ rx_3 & ry_3 \end{bmatrix} \tag{2.2}$$

$$l = [x \quad y] \tag{2.3}$$

Donde:

i es el número de agente robótico, $i \in N, i = 1,2,3$

h_i es la posición relativa del agente i respecto al Líder Virtual

rx_i es la posición relativa en el eje X del agente robótico i [m]

ry_i es la posición relativa en el eje Y del agente robótico i [m]

h es la unión de los vectores de las posiciones relativas de los agentes respecto al Líder Virtual

l es la posición del Líder Virtual en el plano X-Y en *metros*

Una parte importante en el control una vez se tiene las posiciones a las cuales tiene que llegar cada agente, es el seguimiento de la trayectoria hasta llegar a ese punto. El algoritmo de control implementado para el seguimiento de cada agente hacia el punto correspondiente de la formación es **Pure Pursuit** [27]. Este algoritmo se basa en el seguimiento de rutas de un punto hacia otro, los cuales se denominan *Waypoints*, y utiliza la pose conformada por el vector de coordenadas base que se ha descrito en secciones anteriores, donde se tiene posición y orientación del robot. Con lo mencionado anteriormente calcula la velocidad a la que tiene que moverse para trazar una trayectoria y lograr llegar al punto de destino deseado, por lo general busca mantener una velocidad lineal constante y calcula la velocidad angular necesaria para lograr el objetivo.

Para definir el *Waypoint* de cada robot se debe tomar en cuenta el mundo que se está ejecutando, el modo que se ha elegido para el seguimiento de los robots, y la posición actual de cada uno de ellos. Por ende, en el Algoritmo 2.1 se plantea un pseudo-código de lo que se implementa para conseguir cada *Waypoint*, siendo $PosXi$ y $PosYi$ la ubicación actual en plano X-Y de cada respectivo agente.

Algoritmo 2.1: Determinación del Waypoint

Si el mundo es de un robot **Entonces**

    Waypoint1 ← [PosX1 PosY1; x+rx1 y+ry1]

SiNo Si el mundo es multi-agente en Modo 1 **Entonces**

    Waypoint1 ← [PosX1 PosY1; x+rx1 y+ry1]

    Waypoint2 ← [PosX2 PosY2; x+rx2 y+ry2]

Waypoint3 ← [PosX3 PosY3; x+rx3 y+ry3]

SiNo Si el mundo es multi-agente en Modo 2 **Entonces**

Waypoint1 ← [PosX1 PosY1; x+(rx1-rx2) y+(ry1-ry2)]

Waypoint2 ← [PosX2 PosY2; x+rx2 y+ry2]

Waypoint3 ← [PosX3 PosY3; x+(rx3-rx2) y+(ry3-ry2)]

/ / Tomando como el agente Líder Seguidor al robot 2

SiNo

Waypoint1 ← [PosX1 PosY1; PosX1 PosY1]

Waypoint2 ← [PosX2 PosY2; PosX2 PosY2]

Waypoint3 ← [PosX3 PosY3; PosX3 PosY3]

FinSi

Las variables a establecer para realizar el control son la velocidad lineal máxima que se desea del robot, la cual por lo general se mantiene constante en ese valor, la velocidad angular máxima del robot, y el parámetro **LookAheadDistance**. Este último es aquel con el que se sintoniza el control ya que dependiendo del valor se puede obtener una respuesta suave o rápida, teniendo en cuenta el riesgo de inestabilidad, por lo que en cada aplicación se lo puede reajustar. En la Figura 2.16 se puede observar los posibles resultados en la trayectoria al cambiar este valor entre valores pequeños y grandes.

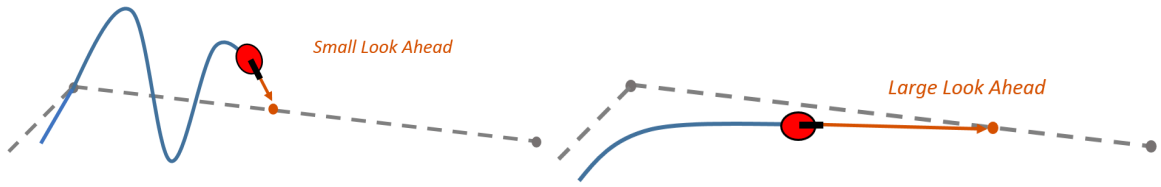


Figura 2.16 Control Pure Pursuit con distintos valores en el parámetro LookAheadDistance [27]

Para mejorar el control y evitar la realización de rutas largas, se implementa un control condicional en la velocidad lineal que calcula el algoritmo Pure Pursuit. Se propone condicionar el valor de la velocidad lineal al ángulo que expresa la dirección de donde se encuentra el punto meta, por ende, en caso de que el ángulo sobrepase el 15% (0.5 rad) del ángulo máximo que se puede tener respecto a la meta (3.14 rad), el robot se mantiene con una velocidad lineal del 1% para centrarse en el giro del robot, caso contrario avanza con su velocidad lineal máxima. Las ventajas de la implementación de este control se observan en la ruta que realiza el robot, y en el tiempo que tardarán las formaciones en realizarse. Las pruebas y resultados que comprueban lo mencionado se realizan y presentan en el siguiente Capítulo.

2.3.3 IMPLEMENTACIÓN DEL ALGORITMO EN SIMULINK

Para poder implementar el algoritmo es necesario conocer de qué manera se pueden conseguir los datos necesarios como es la odometría, de la cual se puede conocer la posición y orientación del robot. Para ello se hace uso de las utilidades de ROS Toolbox que dispone Simulink en MATLAB. Para colocar los bloques que sean necesarios se debe abrir la Librería de Simulink y buscar en la siguiente dirección: **Simulink Library Browser→ROS Toolbox→ROS**, en esta se puede encontrar los bloques para configurar u obtener datos de la red ROS. En la Figura 2.17 se puede observar en la parte superior la figura del bloque y en la inferior la ventana de configuración correspondiente a cada uno, respectivamente para lectura y escritura.

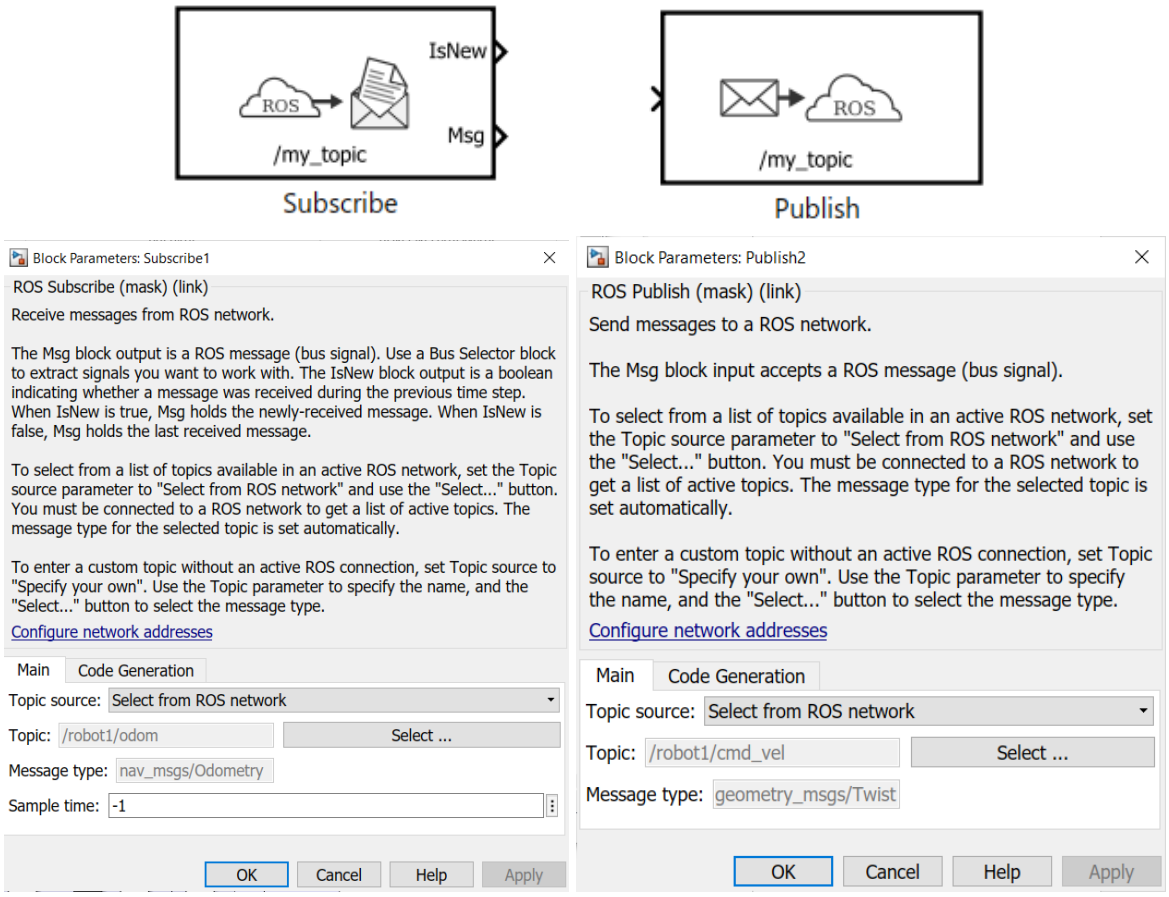
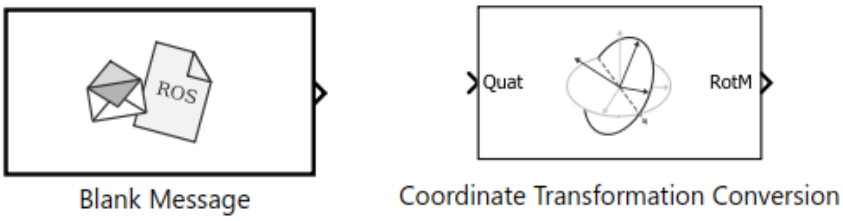


Figura 2.17 Lectura y envío de datos en tópicos de ROS [Fuente Propia]

Otros bloques que se utilizan para la implementación en el modelo de Simulink son aquellos que se muestran en la Figura 2.18. En la parte (a) se presenta el bloque **Blank Message** con el cual se puede introducir un bus de mensaje ROS en blanco que ayuda a establecer el tipo de mensaje que se está enviando. Mientras que el bloque de la parte (b) sirve para realizar una trasformación de coordenadas a elección, en este caso se puede utilizar para obtener la orientación en representación de ángulos Euler y poder utilizar el ángulo Yaw.



(a) (b)

Figura 2.18 Bloques de ROS Toolbox para (a) establecer un mensaje y (b) realizar cambio de coordenadas [Fuente Propia]

En el modelo de Simulink el primer paso es definir los **Waypoints** para cada uno de los robots. El punto de partida se irá actualizando con la retroalimentación que se obtiene al leer la posición de cada robot con ROS Toolbox, mientras el punto final de destino se establece en base a lo definido en la selección de la forma y el modo de seguimiento de los agentes. El esquema general del modelo que se implementa en Simulink se puede observar en la Figura 2.19, lo que se ha descrito hasta el momento abarca el área celeste *Waypoints Input*.

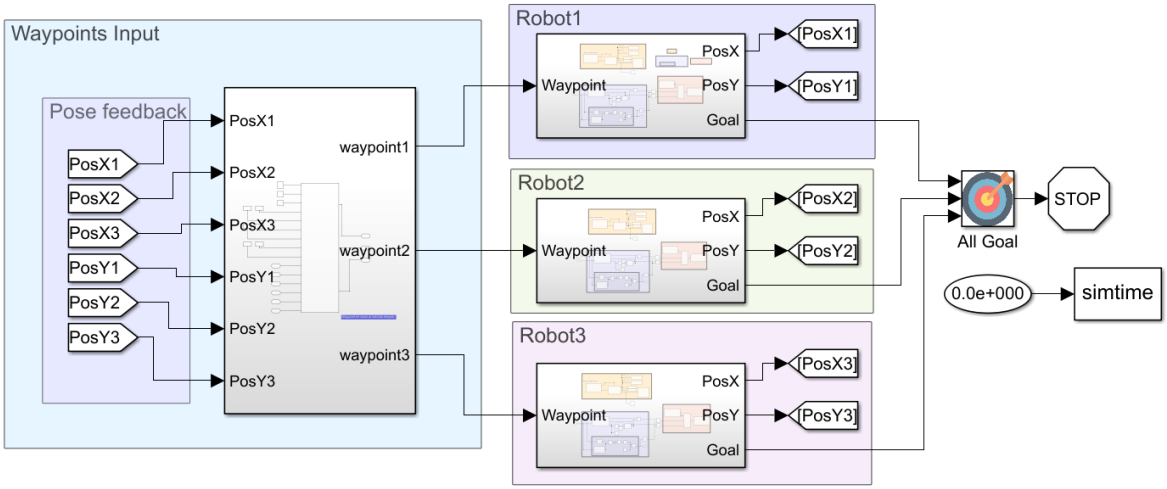


Figura 2.19 Implementación general en el modelo de Simulink [Fuente Propia]

El esquema de control para cada agente robótico se presenta en la Figura 2.20. Dentro del bloque superior se realiza la lectura y procesamiento de los datos para conseguir la pose del robot, la cual se utiliza en el control de seguimiento. El bloque de color morado es en el que se encuentra el control, por lo cual en él se aplica el algoritmo Pure Pursuit, se ajustan sus parámetros de seguimiento y se calcula la velocidad que debe tener el robot para conseguir el objetivo. Dentro del control se ha incluido el cómputo para verificar cuando se encuentra cerca a la meta establecida e identificar el momento cuando ya no es necesario

seguir ejecutando la simulación, además del condicionamiento a la velocidad lineal que se menciona en la última parte de la sección anterior. Por último, se posee el bloque en el cual se realiza la escritura o publicación de los valores de velocidad tanto lineal como angular que se obtiene de la parte del control para el robot.

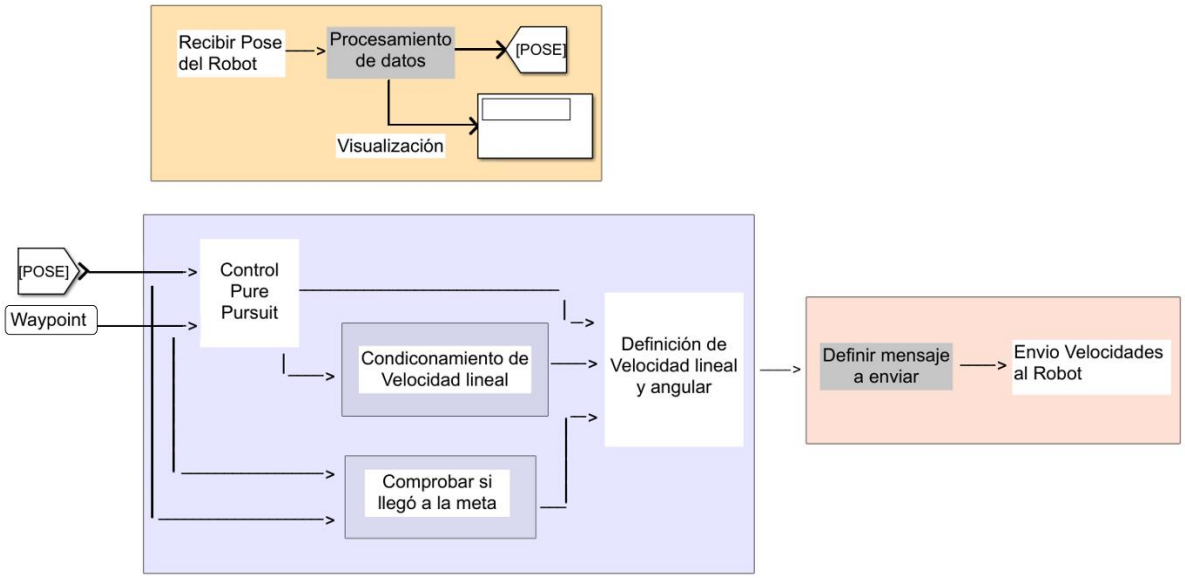


Figura 2.20 Diagrama esquemático de control para mover cada agente a su posición [Fuente Propia]

Como se puede observar en la parte derecha de la Figura 2.19, como parte final del modelo, se posee un bloque en el cual se reúnen las señales que indican el logro del objetivo de todos los robots. Este bloque permite el paso hacia el fin de la simulación ya que se ha terminado de realizar la formación, y a la vez se obtiene el tiempo que ha tardado la simulación el cual refleja lo que ha tardado en realizar el sistema la acción pertinente. Los diagramas esquemáticos presentados en esta Sección son para facilitar su explicación. Los diagramas detallados se los puede encontrar en el Anexo III.

2.4 DESARROLLO DE LA INTERFAZ GRÁFICA

Una interfaz gráfica es esencial para poder establecer el vínculo con el usuario y es mediante la cual se retroalimenta del estado en el que se encuentra el sistema. En el presente trabajo se ha desarrollado la interfaz haciendo uso de los **Live Script** que posee MATLAB [28]. La organización de la interfaz se relaciona a los niveles que posee el sistema, mismos que se detallaron en la Sección 2.2.1, y agregando la opción de una sección de la interfaz destinada a cuando se realicen pruebas con un solo agente. Además, en la máquina virtual se ha establecido un menú inicial para que el usuario pueda ejecutar la escena multi-agente de manera sencilla, así como cerrarla. A continuación, se detalla en

primer lugar la interfaz que se realiza para la plataforma de simulación, y posteriormente la que se realiza en MATLAB en conjunto con los niveles del sistema y la alternativa de realizar pruebas con un agente.

2.4.1 INTERFAZ EN LA PLATAFORMA DE SIMULACIÓN ROBÓTICA

Para obtener una interfaz dentro de la máquina virtual se desarrolla archivos tipo *bash*, los cuales interpretan líneas de código y los resultados se pueden observar al ejecutarlo en un terminal o consola en el entorno de Linux que se está trabajando. El comando que se debe ingresar para ejecutar un archivo se muestra a continuación:

```
user@ubuntu:~$ . archivo.sh
```

Donde `archivo` es el nombre que se le dio al archivo tipo *bash* que se quiere ejecutar. Tomando esto en cuenta, se desarrolla un archivo que actúe como el principal y contenga a los demás que se deban ejecutar según los requerimientos. En la Figura 2.21 se muestra la organización dada a los diferentes archivos, y en la Figura 2.22 se muestra el resultado de la interfaz desarrollada.

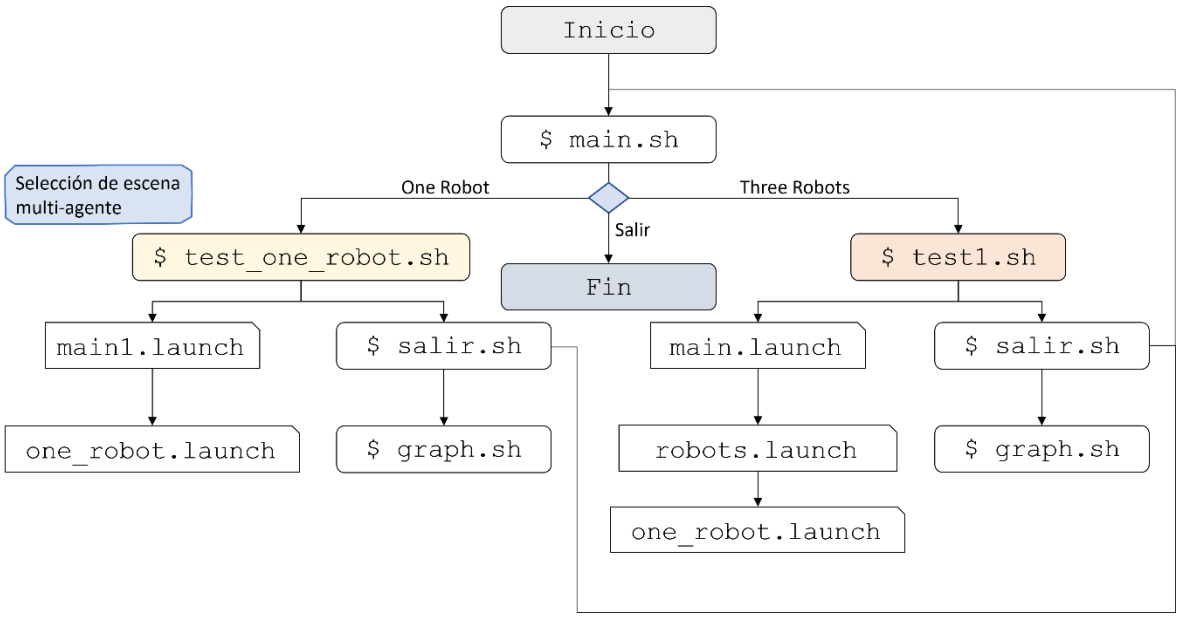


Figura 2.21 Organización de los archivos para ejecutar la interfaz de la plataforma de simulación [Fuente Propia]

(a) (b)

Figura 2.22 Interfaz final desarrollada, (a) su menú principal y (b) la ventana emergente para finalizar la escena ejecutada [Fuente Propia]

2.4.2 INTERFAZ EN MATLAB

Dado que esta interfaz es aquella que tiene como objetivo principal ser de guía para el usuario, en esta sección se describen pasos para utilizarla. En la parte inicial de la interfaz se realiza la conexión con el nodo maestro de ROS del simulador e inicialización de las variables a utilizar en el sistema, y se dispone de una elección acerca de la escena abierta en la plataforma de simulación, ya sea la escena con 3 robots o uno solo. La organización que se menciona se observa en la Figura 2.23, donde en base a la escena abierta se puede ir hacia el sistema multi-agente, en el cual se podrán observar los niveles en los que se ha dividido para su comando, o una sección destinada a realizar pruebas con un agente.

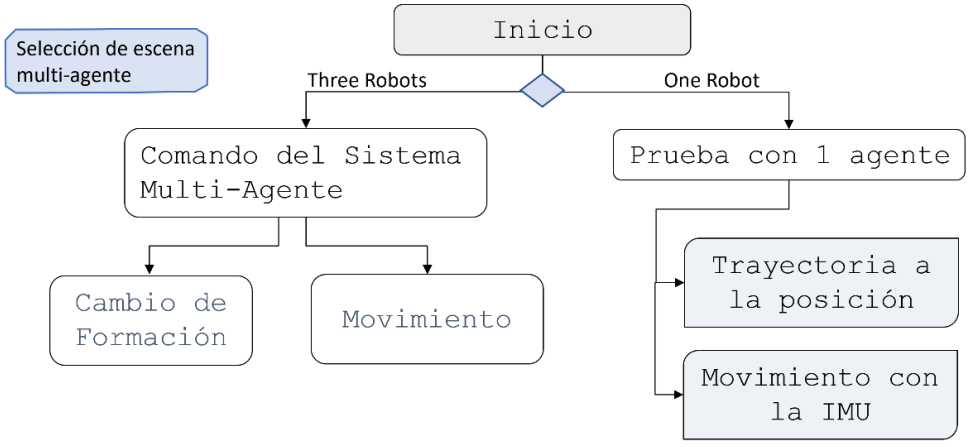


Figura 2.23 Organización de la interfaz desarrollada en MATLAB [Fuente Propia]

2.4.2.1 Prueba con 1 agente

En esta parte de la interfaz se especifica las posibles pruebas que se puede realizar con un agente, las mismas que se observan en la Figura 2.23 y son:

- Prueba de control de trayectoria para llegar a una posición
- Prueba de movimiento de un agente con la IMU (señales inerciales) del sensor Myo Armband

Se dispone de hipervínculos para moverse con mayor facilidad entre las dos opciones, y pasos a seguir para realizar las pruebas correctamente en cada caso. El objetivo de la prueba de trayectoria hacia una posición es comprobar el control realizado para que un robot avance hacia su frente al momento de dirigirse hacia su posición meta, evitando grandes giros en su trayectoria recorrida, para lo cual primero cambia su orientación en el mismo sitio en caso de que la dirección de avance hacia la meta pase de 0.5 radianes. Esto parte del control condicional descrito en la última parte de la Sección 2.3.2-Algoritmo de Control de Formación. Mientras que, mediante la prueba de movimiento se realiza una comprobación simple del funcionamiento de movimiento en base a los lineamientos establecidos con la IMU de sensor Myo Armband.

2.4.2.2 Comando del Sistema Multi-Agente

El nivel de **Menú Inicial** es aquel que se muestra al abrir esta sección. Antes de dirigirse a cualquiera de los otros niveles, en esta parte se selecciona el modo de seguimiento que se utiliza para el control. Para la operación del sistema se puede utilizar el sistema de reconocimiento de gestos o el ingreso de comandos de referencia; por tanto, antes de continuar es necesario ejecutar en la otra instancia de MATLAB, ya sea el programa del sistema HGR o aquel para enviar comandos de referencia.

Siguiendo los lineamientos establecidos, permite seleccionar al usuario la siguiente etapa a la que se quiere dirigir, basándose en la acción que quiere realizar con el sistema multi-agente, ya sea **Control de formación** para realizar un cambio a la misma, o **Movimiento de trayectoria grupal** para desplazar la formación como un todo.

La Figura 2.24 muestra la parte de la interfaz cuando se ingresa a la sección principal de comando del sistema multi-agente que representa el **Menú Inicial**, y en el Anexo IV, dentro del Manual de Usuario, se encuentran las figuras de cada parte de la organización de la interfaz, que se muestra en la Figura 2.23.

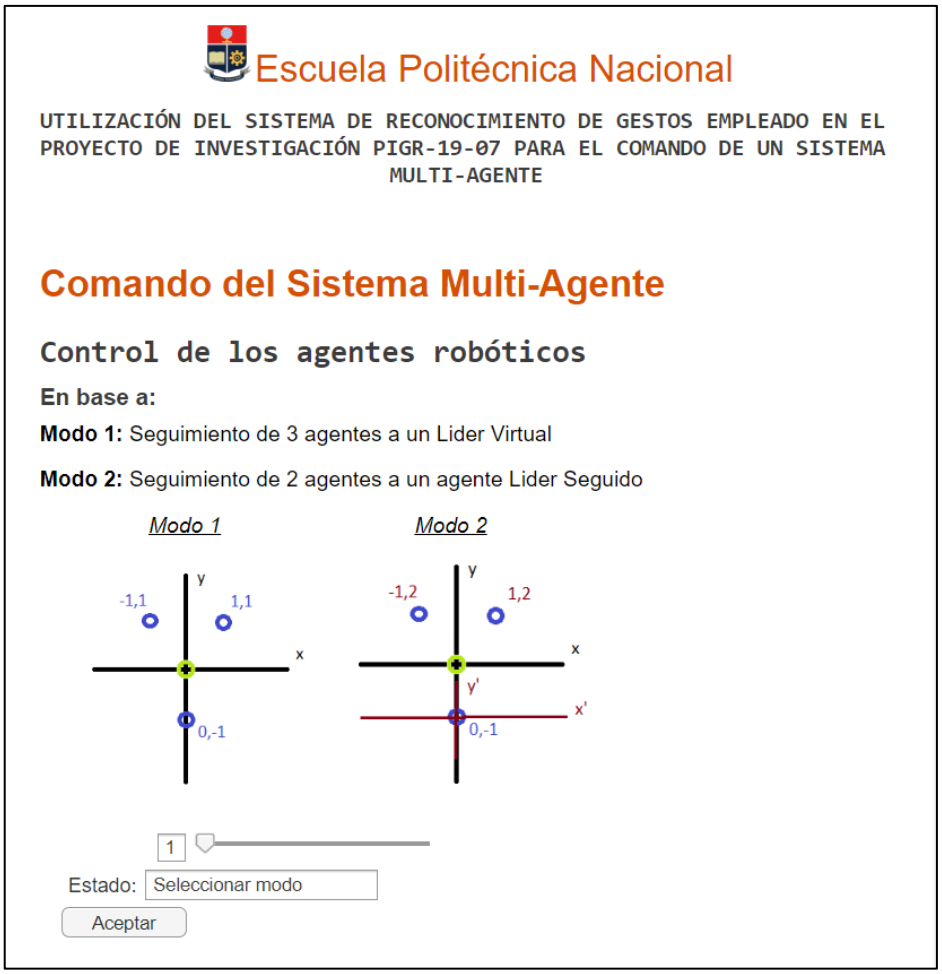


Figura 2.24 Imagen representativa de una sección de la interfaz desarrollada en MATLAB [Fuente Propia]

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 PRUEBAS Y RESULTADOS

Para comprobar el desempeño del sistema se propone la realización de varias pruebas, las mismas que se enfocan en lo siguiente:

- Análisis del movimiento de un agente robótico, lo cual incluye la sintonización de parámetros del algoritmo de control.
- Análisis de los cambios de formaciones comandados por referencias ingresadas o el sistema de reconocimiento de gestos.
- Verificación de la integración del sistema de reconocimiento de gestos a través del control de cambio de formaciones.

- Verificación del seguimiento de trayectoria de la formación como un todo al utilizar la IMU del sensor Myo Armband.

3.1.1 PRUEBAS CON 1 AGENTE

En las pruebas con 1 agente es necesario tomar en cuenta los siguientes detalles:

- la pose inicial que representa el punto de partida del robot al abrir la escena en el simulador,
- la distancia respecto al Líder Virtual que se plantea para el agente robótico, y
- los tipos movimientos que serán necesarios para mover el agente, tomando en cuenta los cambios de formación que se dan en el sistema multi-agente.

Para la representación de las poses y posiciones relativas de cada agente se utilizan respectivamente la Ecuación (2.1) y la Ecuación (2.2), en las cuales se especifica cada uno de sus elementos y sus unidades. Cuando se abre la escena para la prueba con un robot, el TurtleBot3 aparece en la posición [0.5 , 0.5] en el plano X-Y y su frente se encuentra hacia la dirección positiva del eje X del plano que muestra la escena en el simulador, lo cual expresado en términos de la Ecuación (2.1) es dado por: $Pose\_1 = [0.5, 0.5, 0.00]^T$. La posición de referencia respecto al Líder Virtual se expresa usando la Ecuación (2.2) siendo: $h_1 = [0.0\ , 0.5]$, lo cual indica que el agente se encontrará por encima en el eje Y de la posición en la que se encuentre el Líder Virtual. Los movimientos necesarios en base a las formaciones establecidas para el presente trabajo, son para desplazarse en el sentido de los ejes o en direcciones inclinadas, las mismas que se observan en la Figura 3.1, estableciendo la relación de estas direcciones con una proporción *x:y*, los ángulos que se muestran son tomados desde la premisa que el robot se encuentre con una posición angular de 0 radianes inicialmente. Siendo que se aplica el control condicional descrito en la Sección 2.3.2, el robot avanza con su velocidad máxima si el ángulo del objetivo de destino es menor a $|0.5|$ radianes, por tanto, si el ángulo es mayor a eso se centrará en girar, por lo cual es posible realizar los movimientos mostrados con la menor distancia de desvío a la trayectoria recta.

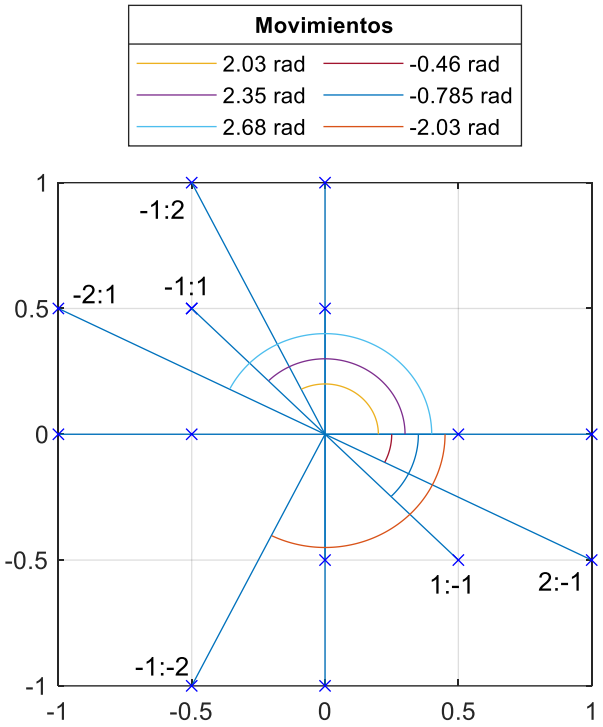


Figura 3.1 Tipos de direcciones de movimiento posibles al realizar un cambio de formación [Fuente Propia]

A continuación, se presenta las pruebas realizadas para la sintonización del algoritmo encargado de seguimiento de trayectoria, y una vez obtenido ese resultado, se procede con las pruebas realizadas al control condicional implementado en la velocidad lineal, siendo esto parte de lo explicado en la Sección 2.3.2-Algoritmo de Control de Formación. Además, se realizan las pruebas de movimiento con los comandos obtenidos de la IMU para probar el comando en un solo agente.

3.1.1.1 Sintonización del algoritmo Pure Pusuit

Con el fin de implementar el algoritmo de seguimiento de trayectoria Pure Pursuit, es necesario sintonizar el parámetro LookAheadDistance que necesita calibrarse para verificar que su valor sea adecuado para la aplicación. Tomando en cuenta las condiciones iniciales en la escena, se establece los puntos de destino de prueba para la sintonización: P1 [1.0 , 0.5] que representa una mata a 45° y P2 [1.5 , 0.5] que implica 26.56° de inclinación de la dirección, lo cual cambia el punto de la posición final del robot en cada caso, estos valores se toman como base para la prueba de sintonización inicial ya que más adelante en lugar de girar a un objetivo que se encuentra a más de 90° de su frente, a la vez que avanza a velocidad lineal constante, se cambiará primero la orientación del robot, por tanto estos ángulos serán las direcciones más significativas en el comando del sistema.

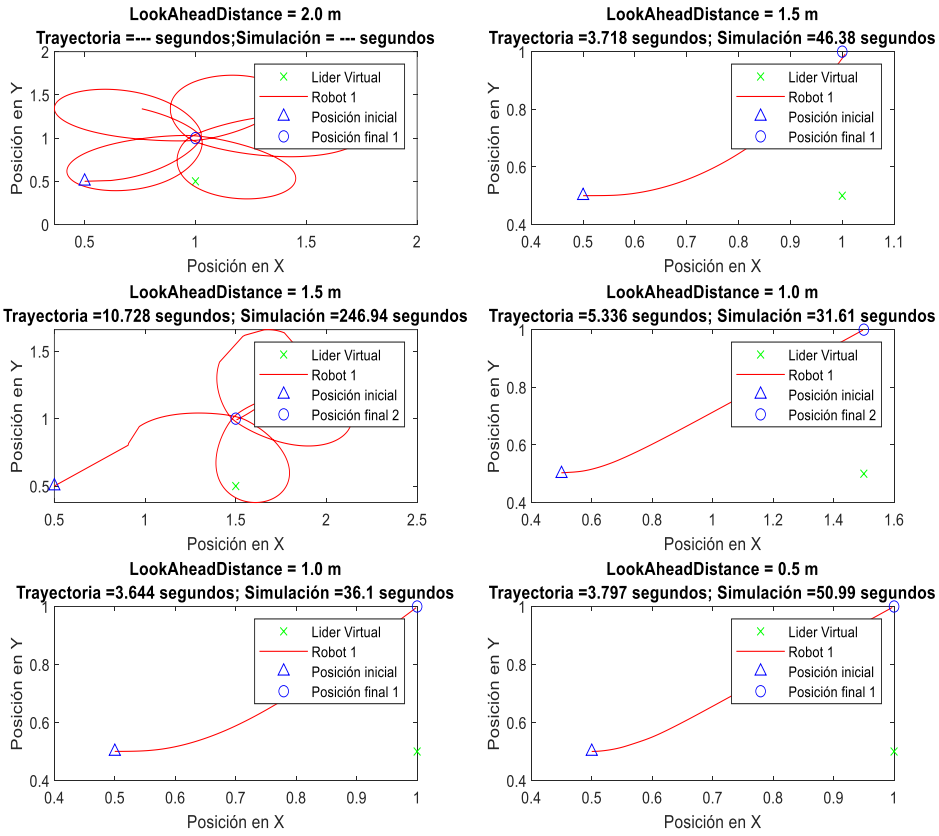


Figura 3.2 Trayectoria del robot hacia una respectiva posición final basada en el valor del parámetro LookAheadDistance [Fuente Propia]

En la Figura 3.2 se observa los resultados de las pruebas realizadas para la calibración del parámetro LookAheadDistance. En la primera imagen se observa que no logra llegar al punto destino a 45° por lo cual se descarta el valor de 2.0 m para el parámetro. Al rebajar el valor a 1.5 m se puede observar una respuesta adecuada, sin embargo, al cambiar el punto de destino al segundo que evoca el ángulo de dirección de 26.56°, se puede ver nuevamente la realización de una trayectoria no conveniente, por tanto, se cambia el valor a 1.0 m. Con este valor se observa una respuesta favorable, y, por ende, se prueba en el punto de destino P1 (45°) para comprobar su funcionamiento. Al observar un resultado aceptable se prueba con rebajar el valor del parámetro a 0.5 m, sin embargo, si bien hay un cambio en la manera de la trayectoria se toma en cuenta que se demora más en lograrlo, por tanto, se decide mantener el valor en 1.0 m. Para una mejor verificación de la elección del parámetro se prueba con otros puntos elegidos en base a las direcciones de movimientos que se muestran en la Figura 3.1 sin considerar un cambio de orientación previo, y los resultados se muestran en la Figura 3.3, en los mismos que la trayectoria asemeja a la mejor situación que se puede obtener.

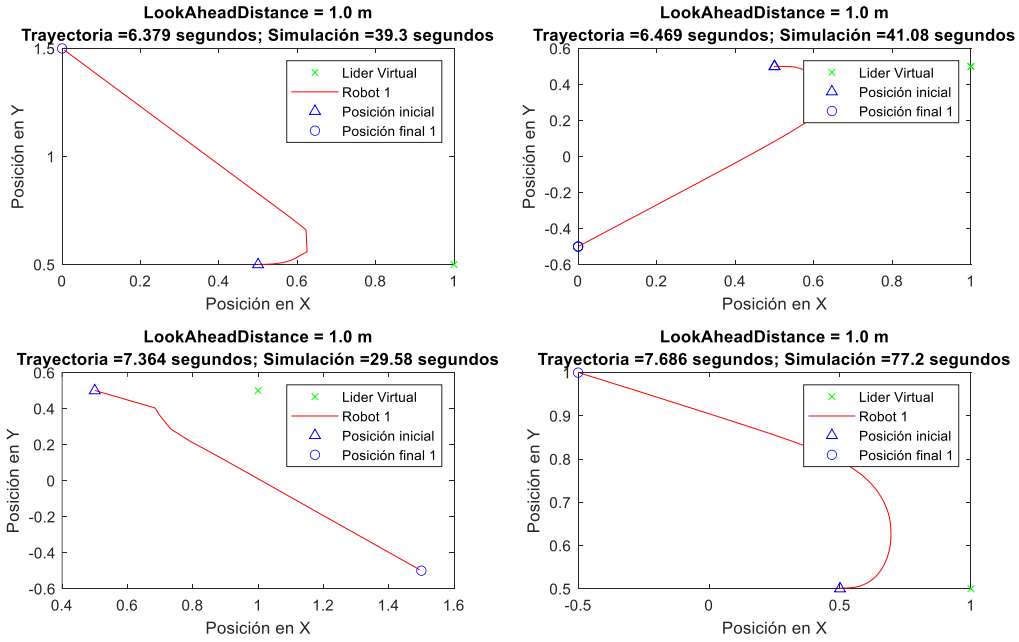


Figura 3.3 Verificación de la trayectoria hacia direcciones en distintos cuadrantes durante la sintonización del parámetro LookAheadDistance [Fuente Propia]

3.1.1.2 Condicionalidad de la velocidad lineal al ángulo

Como parte del algoritmo de control que se detalla en la Sección 2.3.2, se menciona la condicionalidad que tiene la velocidad lineal con el ángulo de la dirección del punto meta. En este sentido, se considera importante mostrar las ventajas de implementarlo a través de pruebas en las mismas condiciones cuando no existe la condicionalidad y cuando se la aplica. Se propone también, la aplicación de un control proporcional como opción al condicionamiento de la velocidad lineal, para mostrar cómo se determina que el control condicional es más ventajoso en cuanto a la ruta y tiempo de llegada del robot, basado en los datos obtenidos.

Se toma en cuenta la pose inicial y la relación de distancia con el Líder Virtual planteada como referencia para la posición meta en cada prueba a realizar. Para las pruebas se ingresa la ubicación para el Líder Virtual, se empieza realizando pruebas con distintos puntos de destino y luego con uno solo para comprobar que en varias pruebas se mantiene resultados cercanos. Para seleccionar el mejor método se basa en la relación tiempo-trayectoria, para determinar qué tipo de control cumple una buena relación entre ambos valores, se define como trayectoria buena una que no debe recorrer grandes distancias para los giros y el mejor tiempo a aquel que tenga una variación positiva respecto al tiempo que hace sin condicionalidad, o en caso de presentar variaciones negativas, aquella más cercana a 0. Por lo que, en la Tabla 3.1 se muestran parámetros de “Trayectoria” y

“Tiempo”, el que se basa en el valor de “Variación %”. En base a los resultados gráficos de cada prueba como los que se muestran en la Figura 3.4, se asigna un valor de 0, 1 o 2 a “Trayectoria”, estableciendo 0 como el menos beneficioso, tomando en cuenta que lo que se considera mejor es una trayectoria donde avance directo hacia su frente. En caso de considerar un empate entre las trayectorias de distintos casos que se realizan en una misma prueba, es decir la diferencia en las trayectorias es casi imperceptible, se da el mismo valor considerando asignarlo desde el menor número, caso contrario se asigna 2 al más beneficioso y 1 a aquella intermedia. Para la calificación del parámetro “Tiempo” se califica con valores de -1 a 1, si “Variación %” es positiva toma valor positivo, caso contrario negativo, dando un valor de 0 cuando la variación es menor de alrededor 1%. Para mejor presentación en la tabla los tipos de control de nombran de la siguiente manera:

* Caso 1: Movimiento lineal y angular en conjunto
* Caso 2: Movimiento lineal condicionado por el ángulo
* Caso 3: Movimiento lineal proporcional al ángulo

Para el Caso 1 no se analiza el parámetro “Tiempo” ya que el tiempo que realiza la prueba sin condicionalidad es tomado como base para calcular “Variación %” y definir el valor del parámetro. Por tanto para el análisis se considera como 0 de ser necesario asignar un valor al resultado de “Tiempo” del Caso 1.

Tabla 3.1 Resultados de prueba de control de seguimiento con condicionamiento en la velocidad lineal a distintos puntos

Punto de destino [x , y]	**Caso 1** Tiempo [seg]	Trayectoria	**Caso 2** Tiempo [seg]	Variación %	Tiempo	Trayectoria	**Caso 3** Tiempo [seg]	Variación %	Tiempo	Trayectoria
Prueba 1 [2.0 , -1.0]	8,477	0	8,568	-1,07	0	1	9,207	-8,61	-1	0
Prueba 2 [2.0 , 1.0]	8,554	0	8,597	-0,50	0	1	9,416	-10,08	-1	1
Prueba 3 [1.5 , -1.0]	6,802	0	7,004	-2,97	-1	1	7,249	-6,57	-1	0
Prueba 4 [2.5 , 1.0]	10,489	0	10,452	0,35	0	0	11,5	-9,64	-1	0
Prueba 5 [1.0 , 1.0]	5,638	0	6,022	-6,81	-1	1	6,248	-10,82	-1	1
Prueba 6 [0.0 , -1.0]	6,278	1	6,389	-1,77	-1	2	7,055	-12,38	-1	0

Prueba 7 [-0.5 , 1.0]	8,039	0	8,066	-0,34	0	2	8,232	-2,40	-1	1
Prueba 8 [1.5 , 0.0]	4,707	0	4,725	-0,38	0	0	5,142	-9,24	-1	0
Prueba 9 [0.5 , 1.0]	5,642	0	5,581	1,08	1	2	5,784	-2,52	-1	1
Prueba 10 [-0.5 , 0.0]	11,314	0	8,675	23,33	1	2	9,312	17,69	1	1
Puntuación		1		1,09	-1	12		-5,46	-8	5

En base a los datos obtenidos que se muestran en la Tabla 3.1, se puede decir que aquel que presenta una trayectoria más ventajosa es el movimiento lineal condicionado por el ángulo (Caso 2), mismo que de igual manera presenta en promedio una variación de tiempo positiva, lo cual indica que el agente robótico tarda menos tiempo en llegar a la meta. Si bien el parámetro “Tiempo” muestra valores negativos, eso indica que en ocasiones se demora más en realizar estos casos que el primero que no posee condicionalidad alguna, sin embargo en el promedio de “Variación %” para el Caso 2 tiene un valor positivo, por lo que mantiene un buen puntaje en cuanto al tiempo.

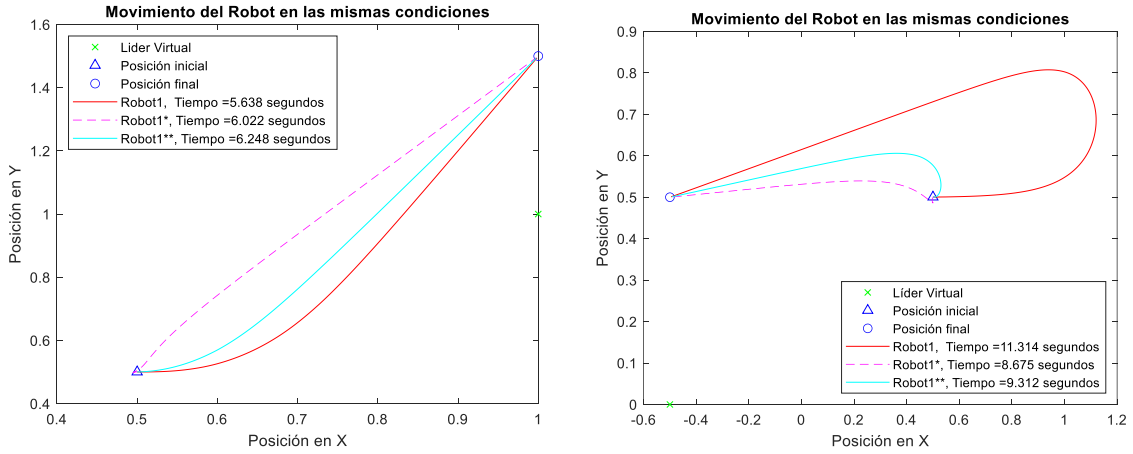


Figura 3.4 Resultados de las pruebas: Prueba 5 [1.0 , 1.0] y Prueba 10 [-0.5 , 0.0]

En la Figura 3.4 se puede observar los resultados de las pruebas 5 y 10, donde la trayectoria roja representa el Caso 1, la magenta el Caso 2 y la de color cyan el Caso 3, de esta manera se puede ver un ejemplo del análisis para los resultados de trayectoria. Si se requiere observar las imágenes de las otras pruebas, estas se presentan en el Anexo V.

Para comprobar la validez y precisión de los resultados se realizan varias pruebas con el mismo punto de destino, en este caso el de la Prueba 10 realizada anteriormente, los resultados obtenidos se muestran en la Tabla 3.2. Al basarse en el último punto de prueba

anterior, el resultado de trayectoria se mantendrá, asegurando la mejor trayectoria al Caso 2 de movimiento condicionado por el ángulo, por tanto, el análisis a continuación solamente se basa en el tiempo.

Tabla 3.2 Resultados de prueba de control de seguimiento con condicionamiento en la velocidad lineal a un mismo punto

Punto de destino [-0.5 , 0.0]	**Caso 1** Tiempo [seg]	**Caso 2** Tiempo [seg]	Variación %	Tiempo	**Caso 3** Tiempo [seg]	Variación %	Tiempo
Prueba 1	9,669	8,734	9,67	1	9,591	0,81	0
Prueba 2	11,432	9,294	18,70	0	9,896	13,44	1
Prueba 3	11,238	9,432	16,07	1	9,989	11,11	1
Prueba 4	9,513	9,169	3,62	1	9,593	-0,84	0
Prueba 5	11,431	9,053	20,80	1	8,466	25,94	1
Puntuación			13,77	4		10,09	3

En la Figura 3.5 se muestra los resultados del parámetro “Variación %” de las Tablas 3.1 y 3.2, en forma de gráficos de cajas, para observar la tendencia de los resultados, mientras mayor sea indica que el método es más rápido que usar el algoritmo de seguimiento Pure Pursuit sin condicionalidad agregada. El primer punto a destacar en ambos casos es que utilizar un condicionamiento en base al ángulo meta (Caso 2) da mejores resultados que una proporcionalidad de velocidad lineal en base al ángulo (Caso 3).

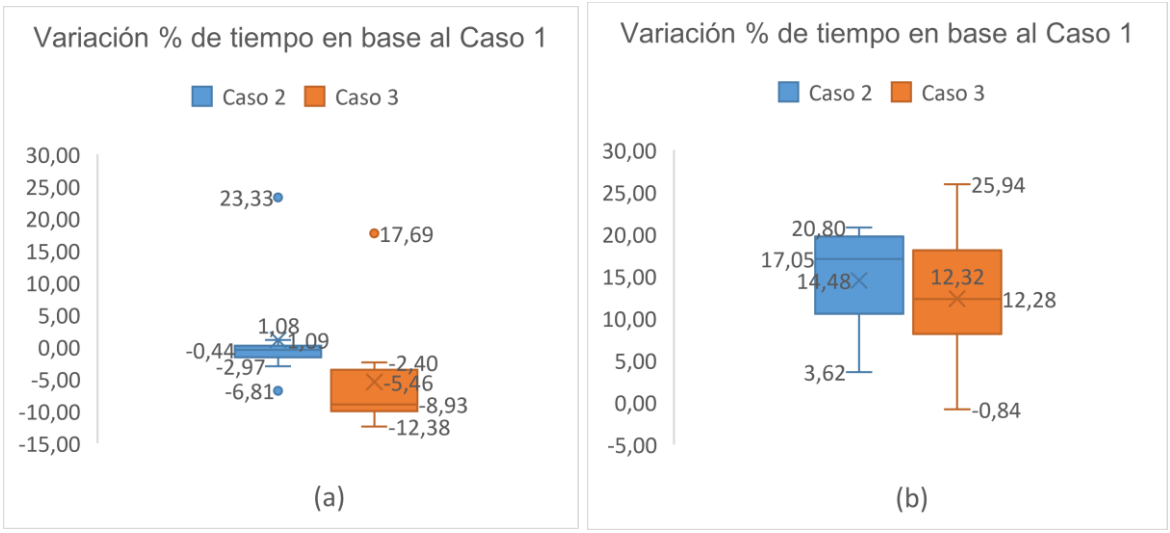


Figura 3.5 Variación % de tiempo al realizar pruebas (a) a distintos puntos de destino (b) con el mismo punto de destino [Fuente Propia]

3.1.1.3 Movimiento comandado por la IMU

Se propone una prueba inicial con un solo agente para probar el movimiento del mismo basado en comandos obtenidos de la IMU del sensor Myo Armband. Para poder utilizar las señales de la IMU es necesario identificar los ángulos de Euler que proporciona y los valores de los límites que se establecen dentro de los lineamientos para usarlos como comandos. En la Figura 3.6 se puede observar un ejemplo de las pruebas realizas para determinar los ángulos límites para definir los comandos. En el Anexo VI se encuentran las figuras que contienen las demás pruebas realizadas, que se resumen en la Tabla 3.3.

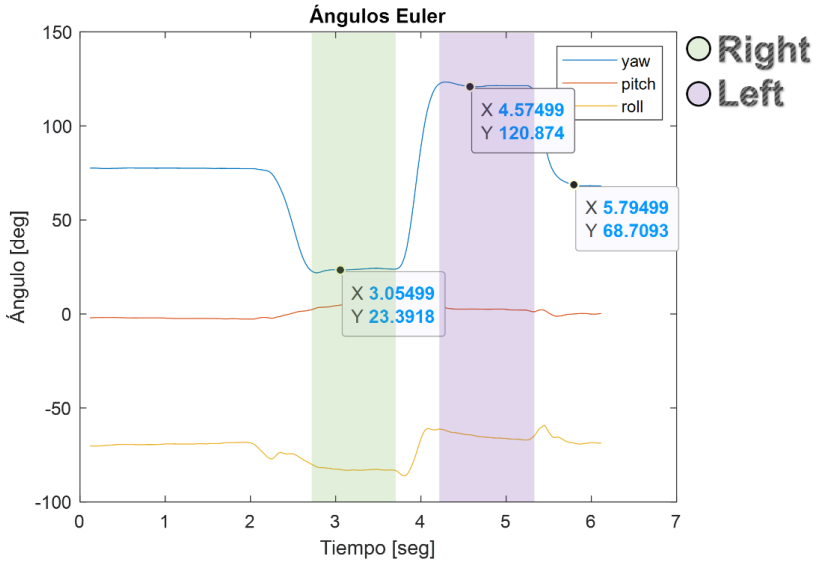


Figura 3.6 Movimiento realizado para obtener los límites del ángulo Yaw [Fuente Propia]

Tabla 3.3 Ángulos de los límites para definir los comandos que se obtienen de la IMU

DATOS DE PRUEBAS [deg]					**Límites para el procesamiento [deg]**	**Límites de movimiento [deg]**
YAW	**P1**	**P2**	**P3**	**P4***	**Resultados**	
RIGHT	23.39	17.41	23.12	-8.34	17	-43
0	68.71	61.029	52.16	33.1437	60	0
LEFT	120.87	93.64	103.876	68.11	100	40
PITCH	**P1**	**P2**	**P3**	**P4**	**Resultados**	
UP	61.52	55.22	38.61	53.62	50	45
0	5.30	4.63	-1 a 7.08	1.14 a 4.57	5	0
DOWN	-58.76	-31.84	-38.03	-48.42	-40	-45

ROLL	P1	P2	P3	P4	Resultados	
FORWARD	40.65	38.242	27.99	8.65	30	64
0	-31.9	-34.92	-24.83	-34.71	-34	0
BACKWARD	-102.88	-105.73	-81.51	-82.75	-95	-61

* Prueba realizada con el frente 45° Right

La última columna en la Tabla 3.3 indica los límites de movimiento, siendo estos ángulos los límites para el usuario tomando la posición de descanso como 0 y la realización del comando cuando los pasen, estos al relacionarse con los ángulos para el procesamiento crean un plano de referencia fácil de intuir para el usuario. Una vez adjuntado el procesamiento necesario para el comando por los ángulos Euler de la IMU, se observa el movimiento del robot, obteniendo como resultado lo que se presenta en la Figura 3.7. De lo que se puede analizar, cuando la distancia de movimiento es más pequeña hay mayor riesgo de trayectorias con curvas.

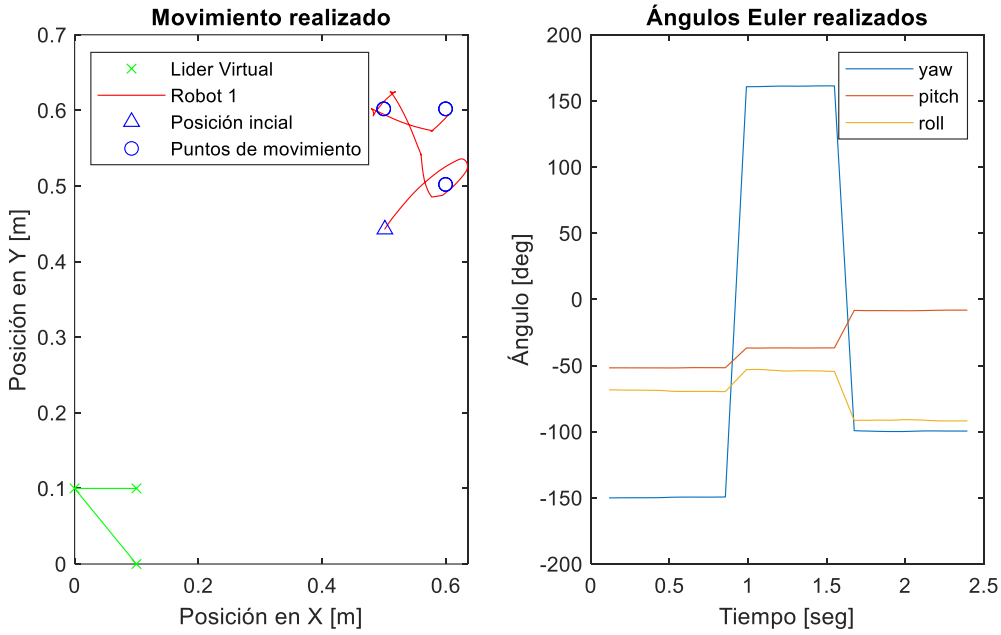


Figura 3.7 Prueba de movimiento de un agente en base a comandos de la IMU del sensor Myo Armband [Fuente Propia]

3.1.2 PRUEBAS DE CONTROL DE FORMACIÓN

Cuando se ingresa al control de formación para el sistema multi-agente, los comandos se obtienen ya sea del archivo para envío de comandos de referencia o del sistema de reconocimiento de gestos. A continuación, se plantean pruebas donde se observan los siguientes datos relevantes: el error de la formación final, la velocidad promedio de cada robot para llegar a su posición y el tiempo de simulación a la formación.

La primera de las pruebas es la realización de cada formación a partir de una misma posición para evaluar en las mismas condiciones. La segunda prueba es establecer un patrón de cambio de formaciones para evaluar cómo van de una hacia la otra. Mientras que, en la prueba en la cual se emplea el sistema de reconocimiento de gestos se analiza también la tasa de éxito al realizar la segunda prueba varias veces de la misma manera para comprobar la fiabilidad de la integración.

La posición inicial de cada robot, basándose en la Ecuación (2.1), al abrir la escena son las siguientes:

$$Pose\_1 = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.0 \end{bmatrix} \; Pose\_2 = \begin{bmatrix} 0.0 \\ 1.0 \\ 1.7 \end{bmatrix} \; Pose\_3 = \begin{bmatrix} -0.5 \\ -0.5 \\ 3.14 \end{bmatrix}$$

Para la realización de las pruebas se muestra primero las formaciones con la identificación de cada robot (véase Figura 3.3), con lo cual se puede ver los movimientos que tendrán que realizar para cambiar a cada una de ellas. Otro dato a tomar en cuenta, es que el valor de la distancia *d* que se toma en cada eje para establecer las posiciones relativas de cada agente, es inicialmente de 1.0 m.

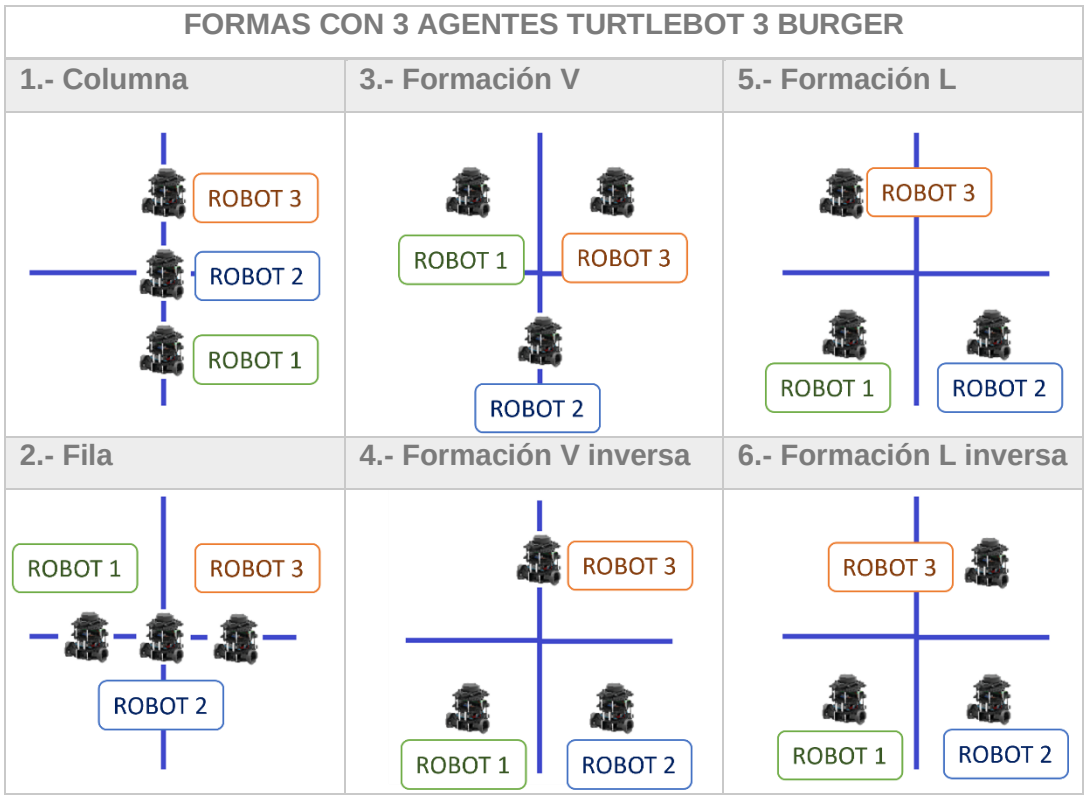


Figura 3.8 Posición de cada agente en las formaciones [Fuente Propia]

3.1.2.1 Pruebas sin el Sistema de Reconocimiento

A partir de las condiciones iniciales de la escena especificadas, se realiza cada una de las formaciones y los resultados se encuentran: en la Tabla 3.3 aquellos utilizando el Modo 1 de seguimiento, y en la Tabla 3.4 utilizando el Modo 2, mismos que fueron descritos en la Sección 2.3.2. Las figuras de las trayectorias realizadas en cada prueba y gráficas de la velocidad se encuentran en el Anexo VII.

Tabla 3.3 Resultados de formación utilizando el Modo 1 de seguimiento

Formación	Error	V1 [m/s]	V2 [m/s]	V3 [m/s]	Tiempo simulación ROS [segundos]
		Velocidad promedio [m/s]			Tiempo real [segundos]
1.- Columna	0,032	0,21872	0,20066	0,21119	11,969
		0,210			80
2.- Fila	0,064	0,20651	0,1705	0,17985	13,948
		0,186			93
3.- Formación V	0,008	0,19192	0,19228	0,17723	15,757
		0,187			106
4.- Formación V inversa	0,024	0,19653	0,19999	0,21509	14,846
		0,204			101
5.- Formación L	0,053	0,200	0,20647	0,16311	17,030
		0,190			172
6.- Formación L inversa*	0,066	0,19547	0,21552	0,22114	9,227
		0,211			105

* A partir de una posición inicial diferente

En la Tabla 3.3 se posee una observación con respecto a la prueba realizada a la Formación L inversa (6), en esta se indica que se realizó con otras condiciones. Esta situación se presenta debido a que al realizar la prueba con las mismas condiciones que las demás, resultó en un choque de los agentes, resultando en la Figura 3.9, esto debido a que para realizar lo indicado se pone a prueba los límites que presenta el trabajo ya que la aplicación se centra en control de formación sin evasión, y por tanto si necesitan cruzar entre ellos se presentan problemas al ser esta una situación extrema.

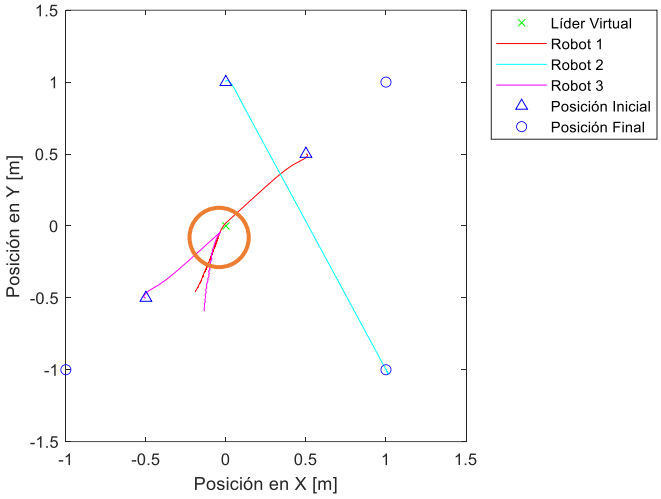


Figura 3.9 Trayectoria realizada al comandar la Formación L inversa (6) en el escenario inicial planteado que conduce a una colisión [Fuente Propia]

Tabla 3.4 Resultados de formación utilizando el Modo 2 de seguimiento

Formación	Error	V1 [m/s]	V2 [m/s]	V3 [m/s]	Tiempo simulación ROS [segundos]
		Velocidad promedio [m/s]			Tiempo real [segundos]
1.- Columna	0,024	0,19297	0,1681	0,19229	17,729
		0,184			195
2.- Fila	0,020	0,18714	0,17951	0,20224	17,882
		0,190			193
3.- Formación V	0,046	0,19904	0,18237	0,21447	29,814
		0,199			334
4.- Formación V inversa	0,046	0,20706	0,21131	0,20905	19,675
		0,209			236
5.- Formación L	0,046	0,20566	0,21184	0,20407	19,525
		0,207			204
6.- Formación L inversa	0,014	0,19709	0,20159	0,20164	26,851
		0,200			256

En las tablas expuestas con los datos de los resultados se observa el error de formación, el cual es la suma de los errores de distancia que deben tener 2 agentes con respecto al sobrante, de esta manera se indica la cantidad en la que la formación no se logra con exactitud.

En algunos casos, debido a que en el presente trabajo no se abarca evasión de obstáculos o colisión como se mencionó anteriormente, cuando las trayectorias que necesitan 2 o más agentes para llegar a su posición se cruzan, es posible que choquen si coincide el momento en que el robot se encuentre allí. Sin embargo, esto se observó en mayor medida al utilizar el Modo 2, es por esta razón que las pruebas en las que se produce un choque sin posibilidad de evasión, como se muestra en la Figura 3.11, se para la simulación (mediante el comando Open) y es por ello que aparecen valores de error mayores en esos casos al estar lejos de lograr la formación debido a escenarios que no se adaptan a la aplicabilidad del presente trabajo, debido a que el alcance no contempla la evasión, se deja como futura implementación un algoritmo de control para evasión que se acople al del presente trabajo. Al observar el comportamiento de los agentes en la simulación, se puede decir que el Modo 2 es más útil cuando se trata de seguimiento, pero no para ubicarse en una formación, por tanto, es un método válido dependiendo la situación de la escena.

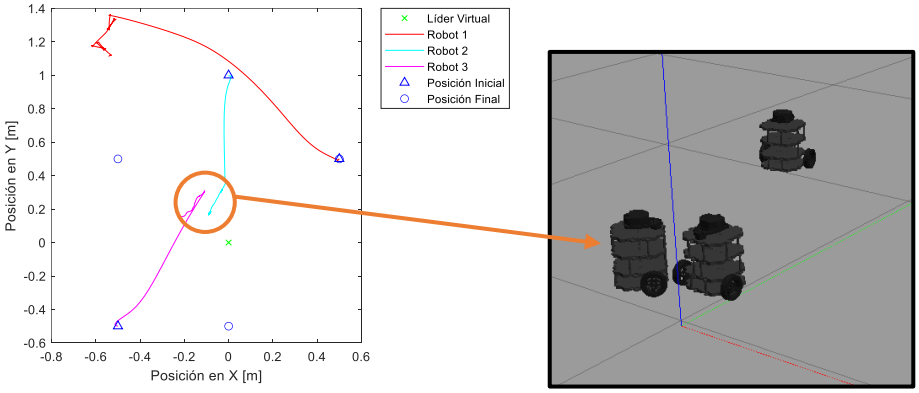


Figura 3.11 Choque de agentes en su trayectoria hacia su posición [Fuente Propia]

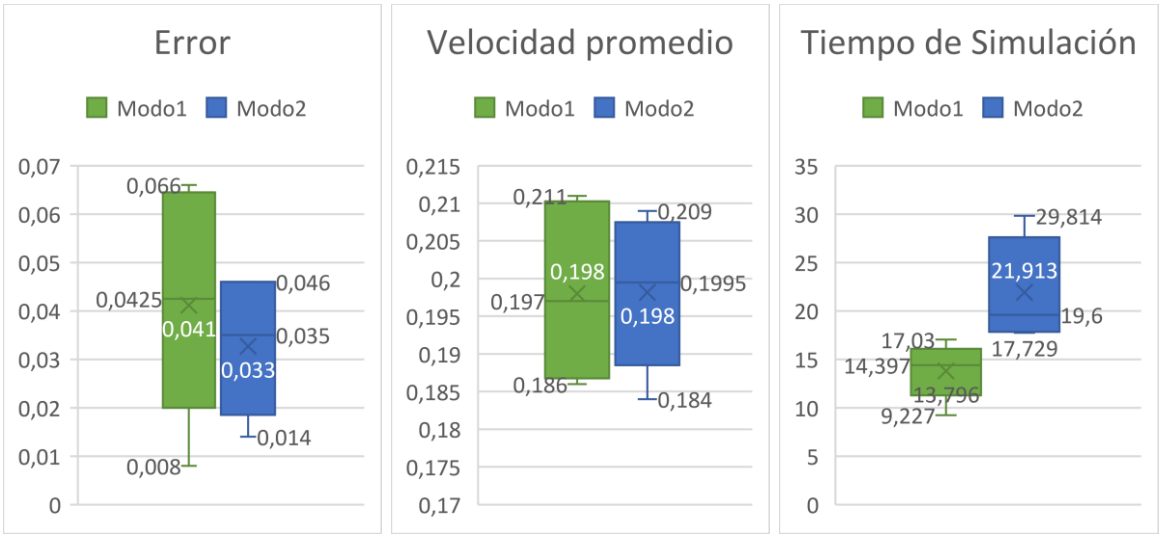


Figura 3.10 Gráficos de cajas y bigotes de las pruebas realizadas [Fuente Propia]

En base a los resultados obtenidos, se analiza los valores del error de formación, velocidad promedio del sistema multi-agente y el tiempo que se tardó en realizar la forma en la plataforma de simulación, en la Figura 3.10 se observan gráficas de los resultados mencionados y se presenta un análisis de las mismas:

- **Error [m]:** Su valor varía entre 0.008 [m] a 0.066 [m], lo cual es un indicador de que, dependiendo de la escena, si bien no hay un 100% en exactitud, el error que acumulan en conjunto es mucho menor al radio de uno de los robots TurtleBot3 Burger de 0.105 [m], por tanto, la formación se logra apreciar correctamente. El Modo 2 presenta un máximo de error menor en 2 [cm], sin embargo, pueden ocurrir eventos no deseados como los choques que se detalló anteriormente.
- **Velocidad promedio [m/s]:** En este caso, la variación que se presente con respecto a la velocidad lineal de 0.22 [m/s], la cual se asigna cuando el robot solo tiene que avanzar, indica la incidencia de la velocidad angular en la prueba, que está presente de mayor manera cuando se gira para cambiar la orientación del robot. El valor promedio para ambos modos es de 0,198 [m/s], sin embargo, el Modo 1 posee valores más cercanos a 0.22 [m/s], por lo cual en este modo es necesario menos tiempo de giro en algunas escenas.
- **Tiempo de Simulación [segundos]:** Este parámetro es el tiempo conseguido mediante ROS del simulador y no el tiempo real, si se lo quiere consultar este dato se encuentra en las tablas de los resultados. En este parámetro se observar una clara diferencia en cada Modo de seguimiento, siendo el Modo 2 aquel con mayores valores que concuerdan con las trayectorias más largas que presentan en sus resultados, para lo cual se hace referencia a la Figura 3.11. La variación que presenta en los tiempos que tarda usando el Modo 1 no es amplia y tiende a un valor de 14.4 [segundos] para las formaciones con distancia de 1.0 [m], siendo poco más de la mitad de lo que tarda con el Modo 2 de 21.91 [segundos], por lo cual por trayectoria y tiempo se observan mejores resultaos con el Modo 1.

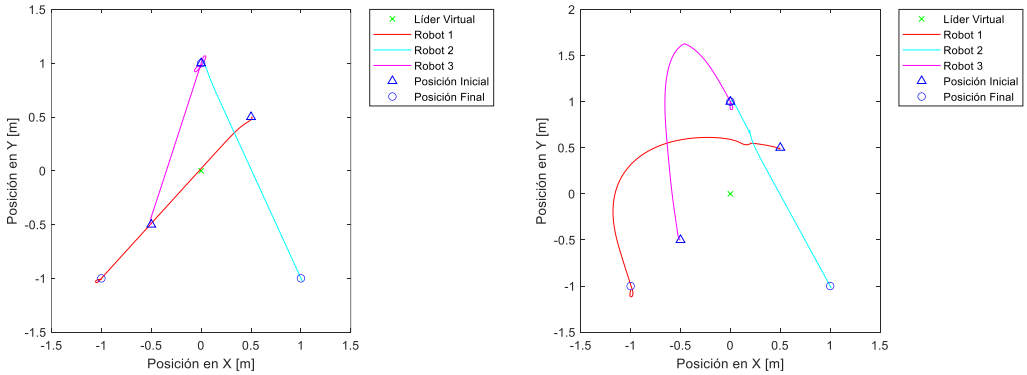


Figura 3.11 Formación V inversa (4) realizada aplicando el modo de seguimiento: (a) Modo 1, (b) Modo 2 [Fuente Propia]

Para la segunda prueba se plantea realizar los cambios de formación en el siguiente orden:

- Columna (1) con una distancia entre agentes de 1.0 m
- Formación V (3) con una distancia entre agentes de 1.0 m
- Formación L inversa (6) con una distancia entre agentes de 0.5 m

El orden que se ha planteado se lo realizó analizando de qué manera se logra visualizar un cambio en las formaciones. De igual manera que en la prueba anterior se obtienen los errores de la formación, sin embargo, en esta prueba para cada formación se parte desde distintas posiciones por la secuencia planteada. Tomando en cuenta el análisis realizado en la prueba anterior, se utiliza el Modo 1 para el seguimiento al momento de la ubicación en las formaciones.

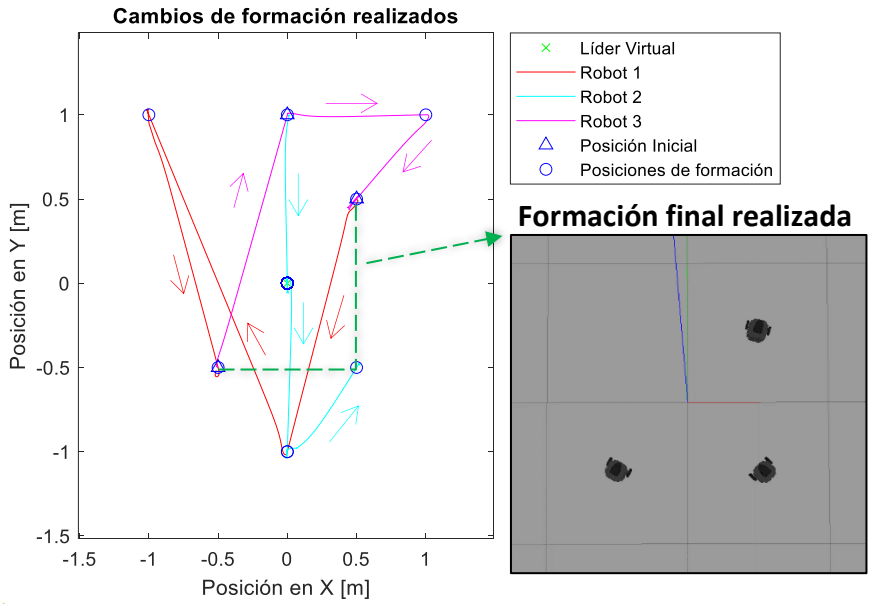


Figura 3.12 Trayectorias de cambios de formación realizados [Fuente Propia]

Tabla 3.5 Resultados de formación siguiendo un orden de cambio

Formación	Error	V1 [m/s]	V2 [m/s]	V3 [m/s]	Tiempo simulación ROS [segundos]	
		Velocidad promedio [m/s]			Tiempo real [segundos]	
1.- Columna	0,022	0,21477	0,18333	0,21625	10,161	
		0,205			71	
3.- Formación V	0,030	0,19525	0,14398	0,22001	15,325	
		0.186			121	
6.- Formación L inversa	0,087	0,20033	0,16151	0,18297	10,562	
		0,182			75	
Promedio	0,0463	0,191			*Total*	36,048

En la Tabla 3.5 se encuentran los resultados obtenidos de la secuencia de cambios realizada en la prueba, se realiza un promedio de los errores y las velocidades obtenidos en cada formación realizada, y para el tiempo se suma los tiempos de simulación para conocer el tiempo que tomó realizar toda la secuencia de cambios. El error de 0.0463 [m] es cercano al valor promedio de error utilizando el Modo 1, mientras que la velocidad de 0.191 [m/s] indica que hay influencia de la velocidad angular por cambio de orientación de los agentes, pero esto no sucede en una gran cantidad de tiempo y mejora el tiempo que tarda en realizarse la formación, lo cual se refleja en que para realizar los 3 cambios tomó alrededor de 36 segundos siguiendo las trayectorias que se muestran en la Figura 3.12.

3.1.2.2 Pruebas con el Sistema de Reconocimiento

Para realizar esta prueba se plantea realizar la misma secuencia descrita en la Sección anterior, para lo cual en esta parte lo importante es qué gestos son los que se tiene que realizar para lograrlo, lo cual se describe en la Tabla 3.6, asumiendo que se encuentra dentro del menú de Control de Formación.

Tabla 3.6 Comandos para realizar las acciones de la prueba

Acción	**Gesto**
Empezar el cambio	Fist
Cambiar a Columna (1) - Seleccionar	Fist
Navegar hacia Formación V (3)	Wave Out
	Relax
	Wave Out
Seleccionar	Fist

Cambiar la distancia a 0.5 m	Pinch
Navegar hacia Formación L inversa (6)	Wave Out
	Relax
	Wave Out
	Relax
	Wave Out
Seleccionar	Fist
Culminar con estos cambios	Open

La prueba mencionada anteriormente se realiza en varias ocasiones para analizar la tasa de éxito de los comandos adquiridos. En base a los comandos mencionados en la Tabla 3.6 para la secuencia es necesario realizar 11 gestos sin contar los momentos en los cuales no se realiza ningún gesto, o los movimientos necesarios para volver a las condiciones iniciales de la prueba.

Tabla 3.7 Resultados de pruebas a la secuencia planteada

Prueba	Número de Gestos	% Éxito	Gesto incidente	Número de Gestos sin el más incidente	% Éxito
1	15	73,33	Pinch	11	100,00
2	14	78,57	Open	13	84,62
3	29	37,93	Pinch	20	55,00
4	20	55,00	Pinch	15	73,33
5	16	68,75	Pinch	12	91,67
6	12	91,67	Wave In	12	91,67
7	14	78,57	Pinch	12	91,67
8	21	52,38	Pinch	15	73,33
9	13	84,62	Pinch	11	100,00
10	18	61,11	Pinch	16	68,75
Gesto más incidente			Pinch	**% Éxito**	83,00

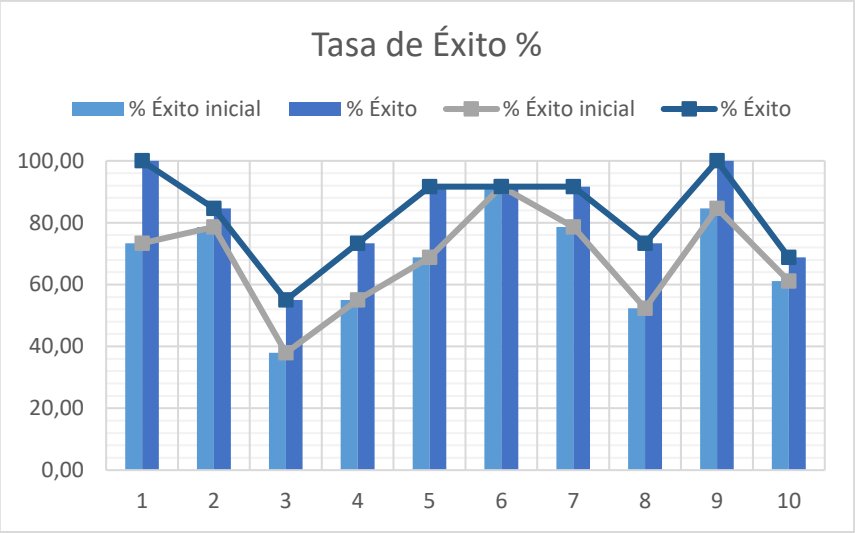


Figura 3.13 Tasa de éxito porcentual de integración [Fuente Propia]

Los resultados obtenidos se pueden observar en la Tabla 3.7, donde se listan el número de gestos que se reconoció en el proceso de la secuencia mencionada en la Tabla 3.6. Como resultado de ello se observó que al mover la muñeca hubo más incidencia a que se reconozca el comando del gesto Pinch cuando no se lo realizó, por ello se observa también la tasa de éxito que se obtiene sin la incidencia del gesto comandado cuando no se lo requería. El porcentaje de éxito promedio es de 83,00%, y las variaciones de este valor en las pruebas se lo puede observar en la Figura 3.13, donde se puede ver que la mayoría de las pruebas presentan un porcentaje de éxito alto cercanos al 70%.

3.2 CONCLUSIONES

- El desarrollo de algoritmos para el reconocimiento de gestos es un campo que está en desarrollo, por lo cual se observa el avance en los trabajos publicados en lo referente a métodos, como lo es el análisis en base a señales electromiográficas, y número de gestos reconocidos. Esto se observa en lo que se expresa en la literatura del proyecto PIGR-19-07, al tener una precisión en su sistema del 81.2%. Por tanto, las aplicaciones que se desarrollan en base a comandos obtenidos de estos sistemas, también van adquiriendo mejores rendimientos y mayores posibilidades.

- Los softwares de simulación robótica están en constante actualización, por lo que cada vez hay mayores posibilidades de aplicación en los mismos. Por ejemplo, gracias a la plataforma ROS se ha vuelto mucho más intuitivo y conveniente el desarrollo de proyectos robóticos. Por esto, MATLAB posee una herramienta llamada ROS Toolbox, que ofrece una interfaz para comunicarse con una red ROS externa de manera sencilla al configurar sus parámetros.

- Al utilizar el TutleBot3 como el modelo de agente robótico, se observa su facilidad de control al basarse en el sistema operativo ROS y su movimiento depende de las velocidades lineal y angular. Tomando en cuenta estas características se eligió el simulador Gazebo donde es conveniente crear una escena con este robot. De esta manera se implementó una máquina virtual que corra Linux para la plataforma robótica, esto por la optimización de ROS para este sistema operativo.
- El control de formación se basa en el seguimiento de cada agente hacia su posición designada, por lo cual las partes principales son la asignación de esas posiciones y el algoritmo para comandarlos hacia ellas. Los métodos de seguimiento pueden variar por el modo en que se organicen los agentes para llegar a su meta, en el presente trabajo se observó que utilizar un solo punto como referencia, siendo el rol del Líder Virtual en el Modo 1, asegura mayor rapidez en la formación y los errores pueden variar a valor un poco más altos, pero no considerados excesivos, logrando realizar la formación de la manera más conveniente.
- Al aplicarse el algoritmo Pure Pursuit se obtiene las velocidades tanto lineal como angular que necesita cada agente para lograr un determinado cambio de posición, sin embargo, se observó que para el control de formación, al aplicar condicionamientos extra como el aplicado en la velocidad lineal, se pueden obtener mejores resultados que involucren la relación trayectoria-tiempo.
- Desarrollar interfaces para la ejecución del sistema ayudan a guiar al usuario para un uso adecuado y retroalimentación de las acciones que se están realizando. MATLAB ofrece distintas maneras en las que se puede lograr el objetivo que tiene una interfaz de usuario, por lo que se ha aprovechado lo que proporciona con su Live Script e incluso en sus modelos de Simulink.
- Al realizar las pruebas con el ingreso de comandos de referencia se logra verificar la correcta implementación del control del sistema multi-agente y la comunicación entre todas las partes del sistema creado en el presente trabajo. El error máximo final en las formaciones realizadas es de alrededor de 0,06 m, valor que es la suma de los errores en las distancias entre los agentes, el mismo que es menor al radio de un solo agente por lo cual se considera que se mantiene la formación.
- Como se comprueba gracias a las pruebas realizadas, la integración con el sistema de reconocimiento de gestos fue factible, sin embargo, debido a la alta necesidad de procesamiento, el ejecutarlo en conjunto con el sistema de control provoca una

ralentización a la máquina, lo cual se observa en la diferencia entre tiempos de simulación y tiempo real.

3.3 RECOMENDACIONES

- Al empezar a realizar cualquier tipo de aplicación, se recomienda comenzar probando la efectividad de lo que se realiza paso a paso, como en el caso de una aplicación a un sistema multi-agente, comprobar el comando sobre solo uno de los agentes, de esta manera se asegura tener bases sólidas para la aplicación que se desarrolle.
- Se pudo notar que las capacidades de los equipos son importantes para ejecutar los programas necesarios para este tipo de trabajos, por lo cual es necesario equipo con suficiente memoria libre para evitar que se cuelgue la máquina en medio de pruebas que se realicen y por ende se refleje en los resultados. Se recomienda utilizar como mínimo una máquina con las especificaciones que se exponen en las *Características del Sistema* en el Manual de Usuario en el Anexo IV.
- Si bien el presente trabajo se centró en el control de formaciones y el movimiento como un conjunto, todo esto se realizó en un entorno libre de obstáculos en el simulador, por lo cual se puede plantear escenarios distintos y enfocarse en la evasión del sistema multi-agente.
- Todo lo realizado hasta el momento se basó en un entorno virtual, sin embargo, los TurtleBot son robots comerciales por lo que un siguiente paso, luego de comprobar la funcionalidad en una plataforma de simulación, es implementarlo en robots físicos.
- Debido a que cada parte utilizada en este trabajo está en constante actualización, se recomienda que, si se usan versiones más actualizadas que las expuestas, se realice una comparación de las sintaxis que se utilizan para evitar algún inconveniente o si se presentara alguno.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] J. A. Ordóñez Flores et al., «A New Methodology For Pattern Recognition Applied To Hand Gestures Recognition Using EMG. Analysis Of Intrapersonal And Interpersonal Variability,» *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM),* pp. 1-6, 2021.

[2] A. Chico et al., «Hand Gesture Recognition and Tracking Control for a Virtual UR5 Robot Manipulator,» *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM),* pp. 1-6, 2021.

[3] R. Romero et al., «Hand Gesture and Arm Movement Recognition for Multimodal Control of a 3-DOF Helicopter».

[4] I. Navarro y F. Matía, «An Introduction to Swarm Robotics,» *ISRN Robotics,* vol. 2013, p. 1–10, 2013.

[5] J. Alonso-Mora, R. Siegwart y P. Beardsley, «Human - robot swarm interaction for entertainment,» de *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, Mar. 2014.

[6] North Canada Devices ULC, «Myo Armband,» 2020. [En línea]. Available: https://support.bynorth.com/myo/. [Último acceso: Mayo 2021].

[7] J. Zea, Implementación de un Sistema de Clasificación de Gestos del Brazo Humano Utilizando Myo Armband para Mando a Distancia de un Brazo Robótico de 3GDL, Quito: EPN, 2017.

[8] J. Ordóñez, Implementación de un algoritmo para el reconocimiento de gestos de la mano en tiempo real, usando señales electromiográficas, Quito: EPN, 2021.

[9] D. Cárdenas, Diseño e implementación de una arquitectura multi agente para comunicar conocimientos de aprendizaje por refuerzo y mejorar el comportamiento de los agentes, Quito: EPN, 2020.

[10] M. Cossentino, L. Sabatucci y A. Chella, «A possible approach to the development of robotic multi-agent systems,» de *IEEE/WIC International Conference on Intelligent Agent Technology. IAT.*, Halifax, NS, Canada, 2003.

[11] J. B. Martínez Morales, *Seguimiento en formación con evasión de obstáculos para un sistema de múltiples agentes no holónomos,* Centro de Investigación en Matemáticas, A.C., 2019.

[12] G. Bermudez, «Modelamiento cinemático y odométrico de robots móviles: aspectos matemáticos,» *Tecnura,* vol. 12, pp. 19-30, 2003.

[13] ROBOTICS e-Manual, «TurtleBot3: Overview,» [En línea]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[14] ROS, «ROS.org,» [En línea]. Available: http://wiki.ros.org/ROS/Introduction. [Último acceso: 2022].

[15] ROBOTIS e-Manual, «TurtleBot3: Features,» [En línea]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/.

[16] Open Robotics, «GAZEBO,» [En línea]. Available: https://gazebosim.org/home. [Último acceso: Abril 2022].

[17] ROS, «ROS.org: Getting Started,» [En línea]. Available: https://www.ros.org/blog/getting-started/#. [Último acceso: Mayo 2022].

[18] MathWorks, «ROS Toolbox,» [En línea]. Available: https://la.mathworks.com/products/ros.html. [Último acceso: Mayo 2022].

[19] N. Solop, «Que es una maquina virtual – Parte 1,» Wetcom: Blog, [En línea]. Available: https://www.wetcom.com/blog/blog-1/post/que-es-una-maquina-virtual-parte-1-204. [Último acceso: Mayo 2022].

[20] N. Solop, «Que es una maquina virtual – Parte 2 – tipos de hypervisores,» Wetcom: Blog, [En línea]. Available: http://www.wetcom.com/blog/blog-1/post/que-es-una-maquina-virtual-parte-2-tipos-de-hypervisores-203#wrap. [Último acceso: Mayo 2022].

[21] MathWorks, «Virtual Machine Installation Instructions: ROS 2 Foxy and Gazebo,» [En línea]. Available: https://la.mathworks.com/support/product/robotics/ros2-vm-installation-instructions-v6.html. [Último acceso: Mayo 2022].

[22] R. A. Romero Paredes, Diseño e implementación de una plataforma multi – rotor de tres grados de libertad comandada por medio de gestos realizados con la mano., Quito: EPN, 2022.

[23] A. P. Chico Godoy, Integración de un sistema de reconocimiento de gestos de la mano basado en el sensor Myo Armband con el simulador de robots Coppeliasim para el control de un manipulador virtual de 6 grados de libertad., Quito: EPN, 2022.

[24] MathWorks, «Communicate Using TCP/IP Server Sockets,» [En línea]. Available: https://la.mathworks.com/help/instrument/communicate-using-tcpip-server-sockets.html. [Último acceso: Junio 2022].

[25] Zivid AS, «Position, Orientation and Coordinate Transformations,» [En línea]. Available: https://support.zivid.com/v2.4/reference-articles/position-orientation-coordinate-transform.html. [Último acceso: Junio 2022].

[26] W. Ren, «Consensus strategies for cooperative control of vehicle formations,» *IET Control Theory Appl.,* vol. 1, nº 2, p. 505– 512, March 2007.

[27] MathWorks, «Pure Pursuit Controller,» [En línea]. Available: https://www.mathworks.com/help/releases/R2021b/robotics/ug/pure-pursuit-controller.html. [Último acceso: Junio 2022].

[28] MathWorks, «Create Live Scripts in the Live Editor,» [En línea]. Available: https://la.mathworks.com/help/matlab/matlab_prog/create-live-scripts.html?lang=en. [Último acceso: Julio 2022].

5 ANEXOS

ANEXO I. Compilación de Comandos de Tutoriales ROS-Gazebo

ANEXO II. Diagrama de flujo de interconexión entre el Sistema HGR y el Multi-Agente

ANEXO III. Diagrama esquemático del control implementado en Simulink

ANEXO IV. Manual de Usuario

ANEXO V. Pruebas con 1 agente

ANEXO VI. Movimientos para la obtención de comandos de la IMU

ANEXO VII. Pruebas de cambio de formación con comando de referencia

ANEXO VIII. Videos de Funcionamiento

ANEXO I

Compilación de Comandos de Tutoriales ROS-Gazebo

Creación del Paquete "Creating the ROS Package"

Para poder ejecutar cualquier cosa utilizando ROS, es necesario crear un "ROS package". Para eso se necesita abrir un terminal o consola. En ROSDS, se puede abrir un terminal haciendo clic en "Tools -> Shell".

Primero se debe crear un espacio de trabajo "workspace". Para el ejemplo a desarrollarse en el presente documento, se llamará ~/simulation_ws. En la consola se coloca lo siguiente:

```
user@ubuntu:~$ mkdir ~/simulation_ws/src -p
```

Luego se compila el "workspace" vacío creado

```
user@ubuntu:~$ source /opt/ros/YOUR_ROS_DISTRO/setup.bash
user@ubuntu:~$ source /usr/share/gazebo/setup.sh
user@ubuntu:~$ cd ~/simulation_ws/
user@ubuntu:~/simulation_ws$ catkin_make
```

Ahora, se crea el "ROS package", el cual en el presente ejemplo toma el nombre de "my_simulations"

```
user@ubuntu:~/simulation_ws$ source
~/simulation_ws/devel/setup.bash
user@ubuntu:~/simulation_ws$ cd ~/simulation_ws/src
user@ubuntu:~/simulation_ws/src$ catkin_create_pkg my_simulations
```

Con esto, la estructura más simple que se puede conseguir de un paquete se debe ver de la siguiente manera:

```
my_simulations/
        CMakeLists.txt
        package.xml
```

Ahora, dentro del paquete creado "my_simulations", se crea una carpeta para **launch** y una para **world**, donde se encontrarán los archivos a lanzar para su ejecución.

```
user@ubuntu:~/simulation_ws/src$ cd my_simulations
user@ubuntu:~/simulation_ws/src/my_simulations$ mkdir launch world
```

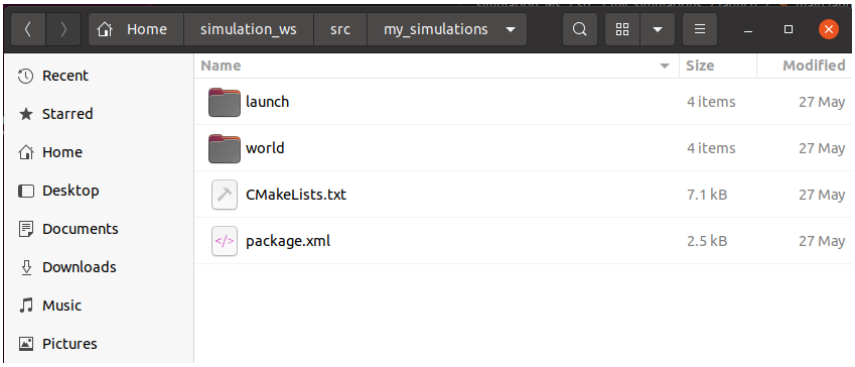


Figura I.1 Directorio my_simulations/

Dentro de la carpeta **launch**, para el presente ejemplo, se crea un archivo llamado "my_world.launch":

```
user@ubuntu:~/simulation_ws/src/my_simulations$ touch
launch/my_world.launch
```

Dentro del cual se coloca el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8" ?>
<launch>
        <!-- overwriting these args -->
        <arg name="debug" default="false" />
        <arg name="gui" default="true" />
        <arg name="pause" default="false" />
        <arg name="world" default="$(find
my_simulations)/world/empty_world.world" />

        <!-- include gazebo_ros launcher -->
        <include file="$(find gazebo_ros)/launch/empty_world.launch">
                <arg name="world_name" value="$(arg world)" />
                <arg name="debug" value="$(arg debug)" />
                <arg name="gui" value="$(arg gui)" />
                <arg name="paused" value="$(arg pause)" />
                <arg name="use_sim_time" value="true" />
        </include>
</launch>
```

Para colocar el contenido, se puede abrir el archivo en un editor de código mediante "Tools -> IDE" o dentro del programa Visual Code Studio, que posee la máquina virtual que se utiliza en el trabajo relacionado al presente documento.

Luego, se crea un archivo llamado "empty_world.world" dentro de la carpeta **world**:

```
user@ubuntu:~/simulation_ws/src/my_simulations$ touch
world/empty_world.world
```

En ese archivo, se coloca el siguiente contenido para tener un escenario vacío:

```
<?xml version="1.0" ?>
<sdf version="1.5">
      <world name="default">
            <!-- A global light source -->
            <include>
                  <uri>model://sun</uri>
            </include>
            <!-- A ground plane -->
            <include>
```

```
            <uri>model://ground_plane</uri>
        </include>
    </world>
</sdf>
```

Si se ha seguido los pasos hasta el momento, se debe tener la estructura que se muestra dejado del código necesario:

```
user@ubuntu:~/simulation_ws/src$ tree
.
|-- CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake
`-- my_simulations
    |-- CMakeLists.txt
    |-- launch
    |   `-- my_world.launch
    |-- package.xml
    `-- world
        `-- empty_world.world
3 directories, 5 files
```

Si no se observa aquello y se muestra un mensaje acerca de que no lo tiene instalado, se puede realizar las respectivas instalaciones o se puede comprobar mediante varios comandos que se mencionarán más adelante.

Cargar y ejecutar un mundo Gazebo usando ROS "Loading the Gazebo world using ROS"

Con lo anterior se creó el espacio de trabajo y el paquete, una vez hecho eso se puede ejecutar los archivos a través del siguiente código:

```
user@ubuntu:~$ source ~/simulation_ws/devel/setup.bash
user@ubuntu:~$ roslaunch my_simulations my_world.launch --screen
```

Variables del Entorno "*ENVIRONMENT VARIABLES*"

```
user@ubuntu:~/simulation_ws/src$ printenv | grep ROS
```

```
user@ubuntu:~$ printenv | grep ROS
ROS_VERSION=1
ROS_PYTHON_VERSION=3
ROS_PACKAGE_PATH=/home/user/simulation_ws/src:/opt/ros/noetic/share
ROSLISP_PACKAGE_DIRECTORIES=/home/user/simulation_ws/devel/share/common-lisp
ROS_IP=192.168.174.129
ROS_ETC_DIR=/opt/ros/noetic/etc/ros
ROS_MASTER_URI=http://192.168.174.129:11311
ROS_ROOT=/opt/ros/noetic/share/ros
ROS_DISTRO=noetic
user@ubuntu:~$
```

Figura I.2 Variables del entorno de ROS

También se puede utilizar comandos específicos, como el siguiente:

```
user@ubuntu:~$ echo $ROS_PACKAGE_PATH
/home/user/simulation_ws/src:/opt/ros/noetic/share
```

Navegar en el Sistema de Archivos de ROS "Navigating the ROS Filesystem"

Usando las herramientas de línea de comandos como **roscd**, **rosls**, y **rospack.** Para ubicar un paquete, en este ejemplo *roscpp,* se utiliza de la siguiente manera:

```
$ rospack find roscpp
```

La forma del comando **roscd** que se utiliza para verificar el directorio donde se encuentra un determinado paquete es el siguiente:

```
$ roscd roscpp
```

Además, permite cambiar el directorio (change directory) directamente de un paquete o stack (pila de información). La estructura para su manejo es la siguiente:

```
$ roscd <package-or-stack>[/subdir]
```

Para poder observar el directorio en el cual está trabajando la consola en ese instante se utiliza el comando Unix:

```
$ pwd
        YOUR_INSTALL_PATH/share/roscpp
```

El comando **rosls** permite ver los archivos que contiene un paquete por su nombre. Ejemplo:

```
$ rosls roscpp
        cmake launch package.xml  src
```

Si se quiere observar una lista de todos los paquetes actualmente instalados, se puede usar la función de completar con la tecla de tabulación de la siguiente manera:

```
$ rosls <<< ahora aplastar la tecla TAB dos veces >>>
```

Conceptos de Nodos de ROS "Understanding ROS Nodes"

Nodos: Un nodo es un ejecutable que usa ROS para comunicarse con otros nodos.
Mensajes: Datos tipo ROS usados para suscribir o publicar en un tópico.
Tópicos: Los nodos pueden publicar mensajes en un tópico, así como suscribirse a uno para recibir mensajes.
Maestro: Servidor principal para ROS (por ejemplo, ayuda a los nodos a encontrarse entre ellos)

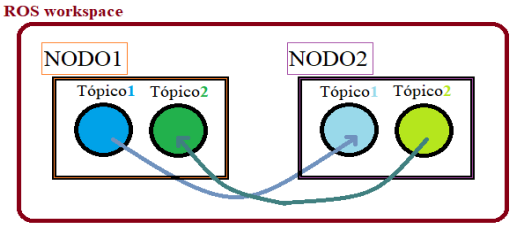


Figura I.3 Relación de nodos y tópicos

A continuación, se detallan unos comandos que son útiles para ver el estado de los nodos y tópicos.

El comando **rosout** cierra el nodo maestro de ROS que pueda estar abierto. El comando **roscore** (Master + rosout + parameter server) es, por el contrario, lo primero que se debe correr cuando se usa ROS para inicializar un nodo maestro de ROS. El comando **rosrun** permite ejecutar un nodo que se encuentre dentro de un paquete con el nombre de ambos.

```
$ rosrun [package_name] [node_name]
```

El comando **rosnode list** permite observar todos los nodos activos actualmente, mientras el comando **rostopic list** permite observar todos los tópicos suscritos o publicados. El comando **rosnode info** devuelve información acerca de un nodo en específico.

```
$ rosnode info /rosout
```

Para observar de mejor manera los nodos y tópicos que se encuentran ejecutando, se puede hacer uso del comando **rqt_graph.** Si no lo tiene instalado ejecute los siguientes comandos:

```
$ sudo apt-get install ros-YOUR_ROS_DISTRO-rqt
$ sudo apt-get install ros-YOUR_ROS_DISTRO-rqt-common-plugins
```

En un nuevo terminal se ingresa:

```
$ rosrun rqt_graph rqt_graph
```

Figura I.4 Arquitectura del entorno de ROS en rqt_graph

La herramienta **rostopic** permite obtener información de los tópicos de ROS, y en su opción de ayuda se puede encontrar sub-comandos que están disponibles.

```
$ rostopic -h
```

```
rostopic bw     display bandwidth used by topic
rostopic echo   print messages to screen
rostopic hz     display publishing rate of topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

```
$ rostopic echo [topic]
```

Ahora, tras el último comando se activa la subscripción hacia un tópico para mostrarlo en pantalla. El nodo que se crea indirectamente se puede observar en color rojo y se observa que ambos nodos están suscritos al mismo tópico [turtle1/command_velocity].

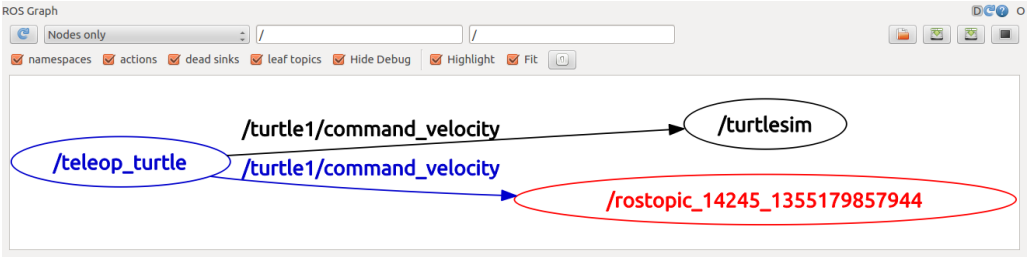
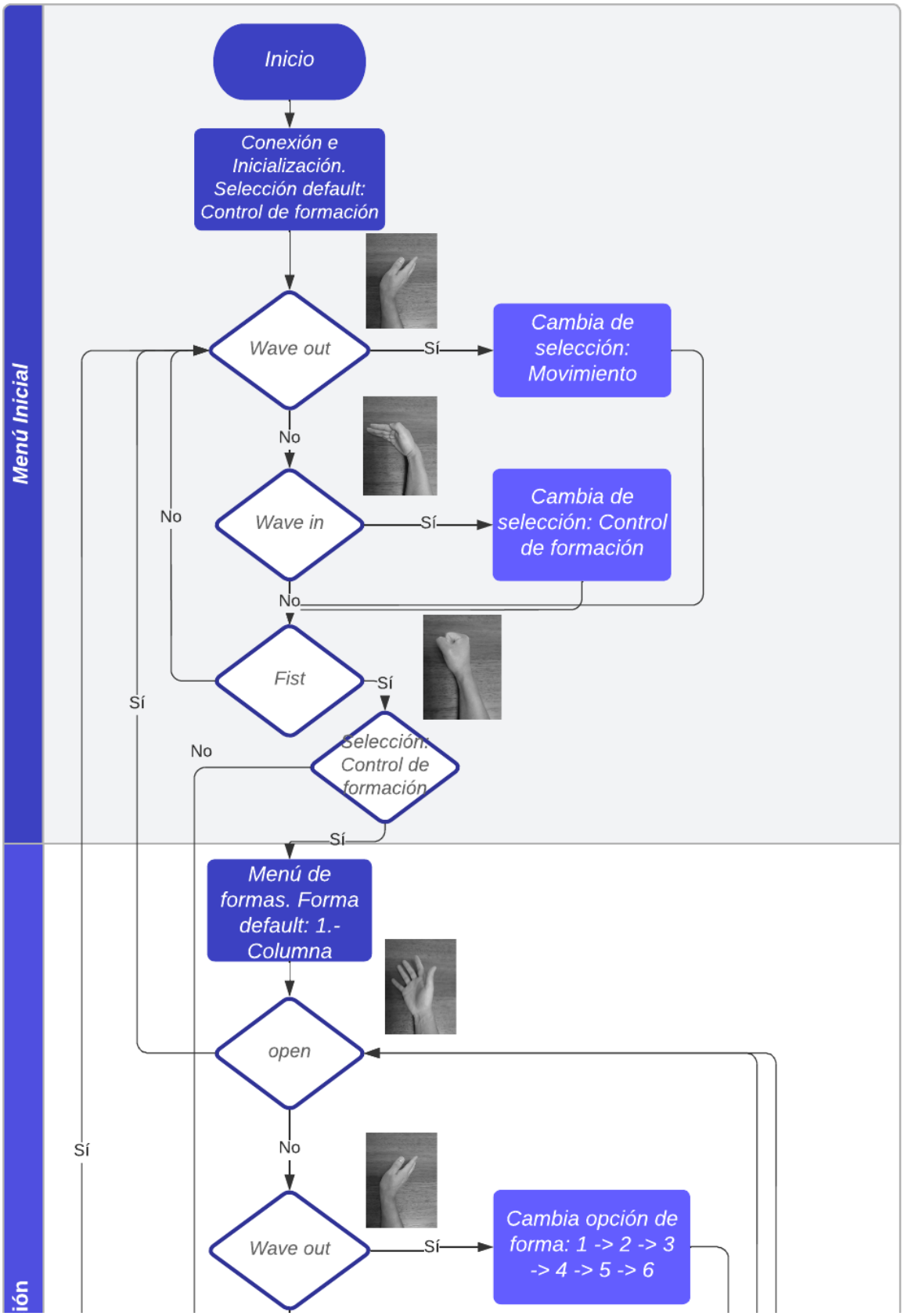
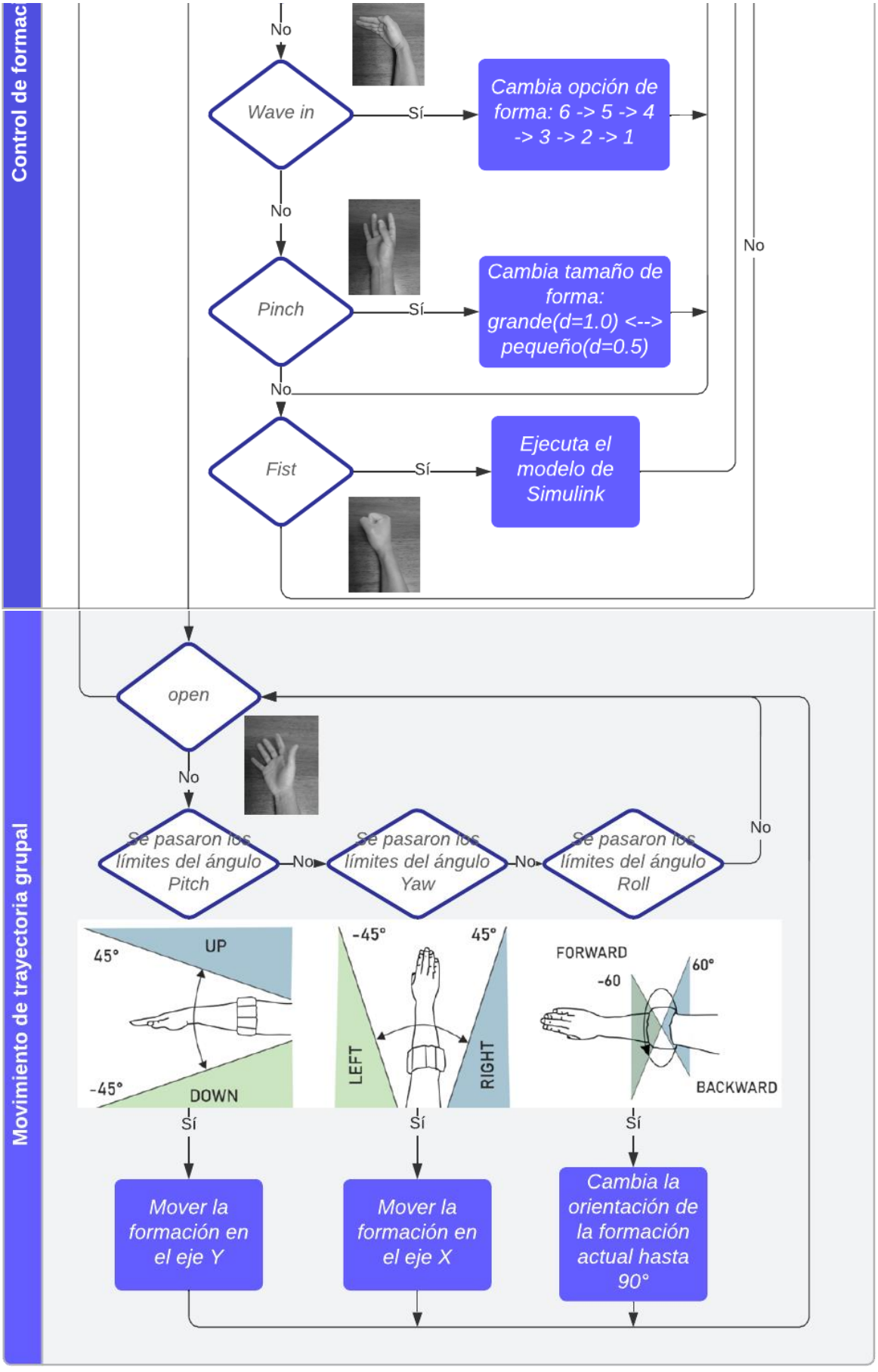


Figura I.5 Arquitectura del entorno de ROS actualizada al ejemplo

ANEXO II

Diagrama de flujo de interconexión entre el Sistema HGR y el Multi-Agente




Control de formac
No
Wave in
Sí
Cambia opción de forma: 6 -> 5 -> 4 -> 3 -> 2 -> 1
No
Pinch
Sí
Cambia tamaño de forma: grande(d=1.0) <--> pequeño(d=0.5)
No
Fist
Sí
Ejecuta el modelo de Simulink
No
Movimiento de trayectoria grupal
open
No
Se pasaron los límites del ángulo Pitch
No
Se pasaron los límites del ángulo Yaw
No
Se pasaron los límites del ángulo Roll
No
45°
UP
-45°
DOWN
-45°
45°
LEFT
RIGHT
FORWARD
-60
60°
BACKWARD
Sí
Sí
Sí
Mover la formación en el eje Y
Mover la formación en el eje X
Cambia la orientación de la formación actual hasta 90°

ANEXO III

Diagrama esquemático del control implementado en Simulink

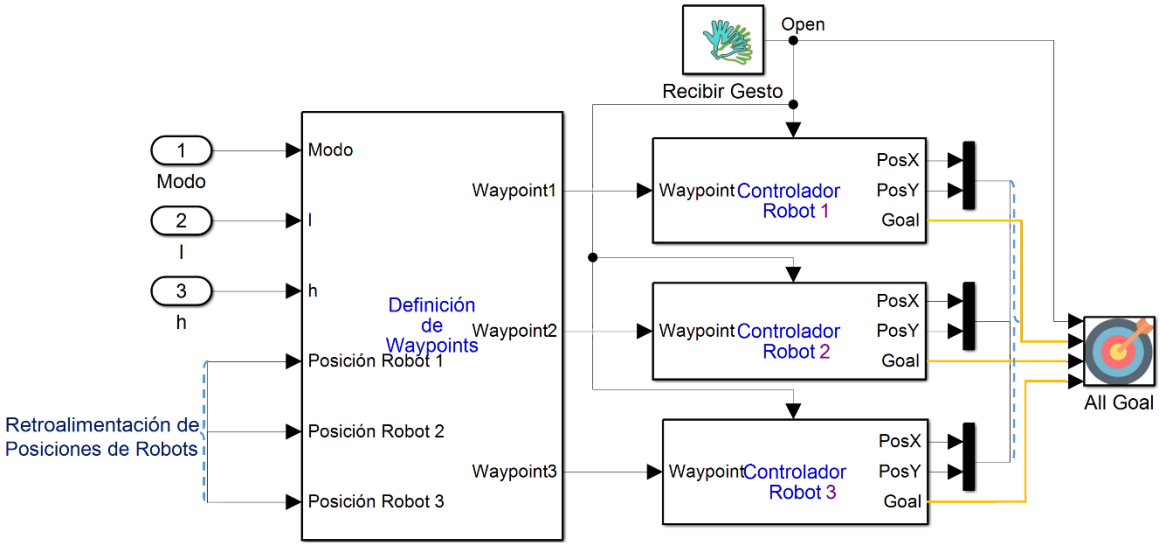


Figura III.1 Diagrama de control general del sistema

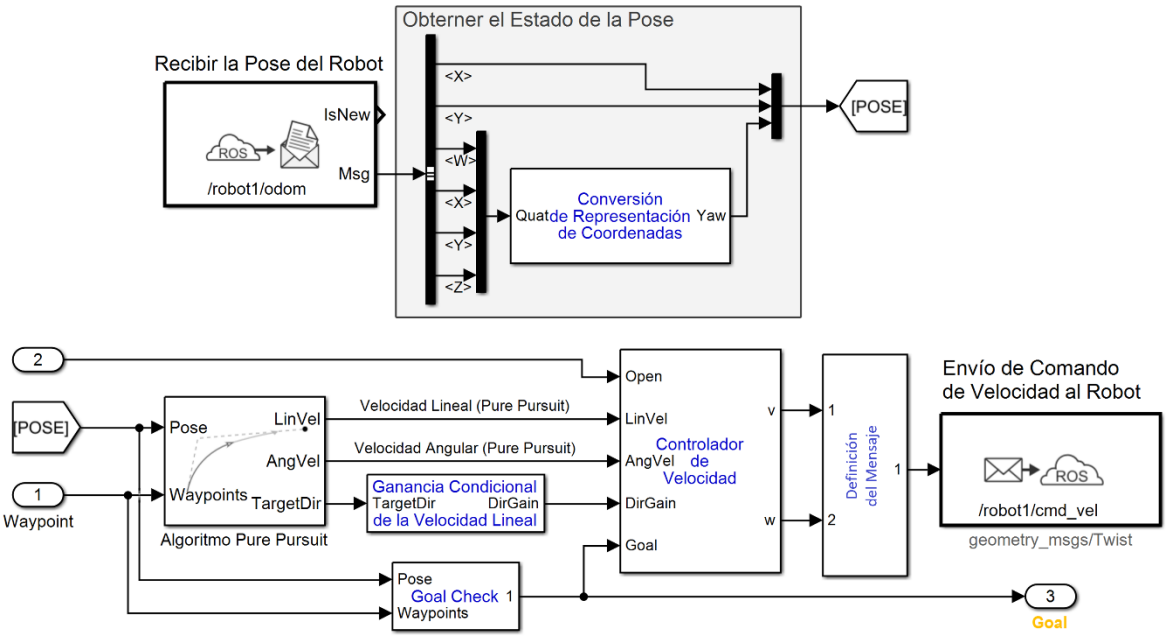


Figura III.2 Diagrama de control de cada Robot para el seguimiento a su meta

ANEXO IV

Manual de Usuario

Requisitos:

Sistema Operativo	Windows 10
MATLAB	2021b +
VMware Workstation	VMware® Workstation 16 Pro. Ver 16.1.2

Las características mínimas que debe poseer el Sistema son:

Procesador	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz
Memoria RAM	8,00 GB
Memoria Disco duro	1 TB de estado sólido

Es necesario instalar de la versión 2021b en adelante. Incluir Simulink en la instalación, además de los siguientes requisitos en Add-On:

- Image Processing Toolbox
- Image Acquisition Toolbox
- Statistics and Machine Learning Toolbox
- Signal Processing Toolbox
- Computer Vision Toolbox
- Deep Learning Toolbox
- Instrument Control Toolbox
- Robotics System Toolbox
- ROS Toolbox

Para utilizar el Sistema de Reconocimiento de Gestos (HGR) del Proyecto de Investigación PIGR-19-07 es necesario seguir unos pasos para su instalación y correcto uso, para lo cual se puede referir a la “Guía de Instalación y Uso de Software de Reconocimiento HGR de Myo con Matlab” que se encuentra como Anexo C del Trabajo de Titulación “Diseño e implementación de una plataforma multi – rotor de tres grados de libertad comandada por medio de gestos realizados con la mano.”, de Ricardo Romero, disponible en http://bibdigital.epn.edu.ec/handle/15000/22245. Adicional a los pasos presentados es necesario reemplazar la carpeta “GeneralHGR” por la que se encuentra en la siguiente dirección en el enlace que se presenta a continuación: Trabajo de Integración Curricular-Orbe Fernanda → 0 PIGR-19-07 Sistema HGR, en el mismo se puede encontrar de igual manera las descargas requeridas.

La instalación de la máquina virtual se la realiza descargando el archivo comprimido en Trabajo de Integración Curricular-Orbe Fernanda → 1 Virtual Machine, el cual se recomienda descomprimir en el Disco (D:) del computador para no llenar la memoria del otro disco. La máquina virtual presenta los siguientes softwares necesarios para el sistema implementado:

Sistema Operativo	Linux Ubuntu 20.04.3 LTS (Focal Fossa)
Distribución ROS	Noetic
Gazebo	11.0.0
Visual Studio Code	1.67.2

Para abrir la máquina virtual se utiliza VMware Workstation, en donde en su pantalla principal se selecciona “Open a Virtual Machine” para buscar el archivo descomprimido, y se elige que se ha copiado esta máquina virtual. Lo siguiente es asignar los recursos que se le darán, como referencia se ha asignado 2GB de memoria y 1 procesador.

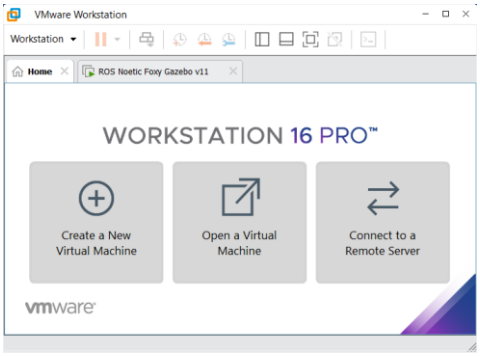


Figura IV.1 Pantalla principal de VMware Workstation

1) Lo primero que se debe realizar es encender la Máquina Virtual y dentro de ella abrir el **ROS Noetic Terminal** en donde se ingresa la siguiente línea: `. main.sh`. Esto también se encuentra indicado dentro del archivo **Read me** ubicado en el escritorio.
   Utilizar el menú que se abre para ejecutar la escena que se desea abrir en Gazebo.

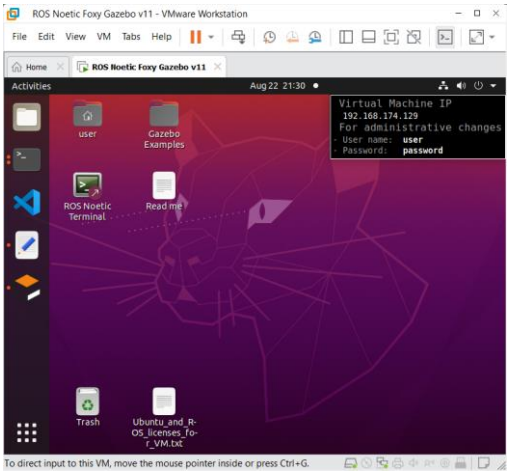


Figura IV.2 Escritorio de la Máquina Virtual

2) Abrir una ventana de MATLAB, y abrir la pantalla de inicio de la interfaz, la cual se encuentra en el archivo **Main.mlx**, y se puede observar en la Figura IV.3, en ella se ingresa la dirección IP que posee la máquina virtual en su computador, y luego de seguir los pasos que se mencionan en ella se Conecta el sistema.

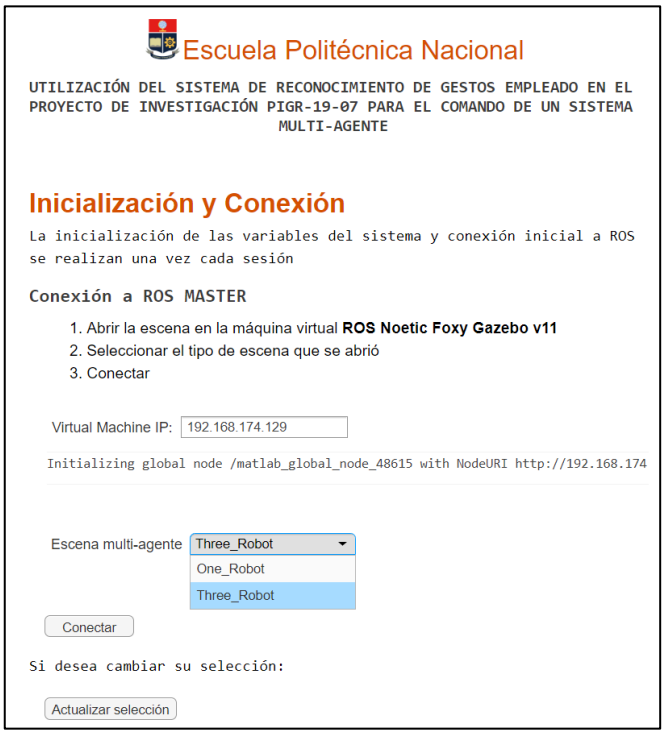


Figura IV.3 Interfaz de Inicio en MATLAB

a. En caso de elegir la escena One_Robot, la cual se puede utilizar para realizar pruebas con un agente, se dirige hacia **prueba1.mlx**, en donde se posee la interfaz que se muestra en las imágenes de las Figuras IV.4 y IV.5.

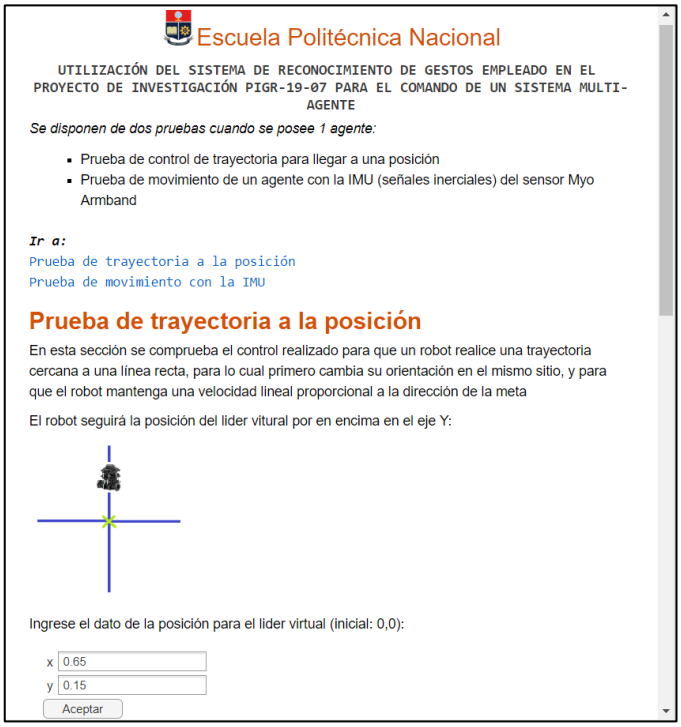


Figura IV.4 Interfaz para realizar pruebas con 1 agente (parte 1)

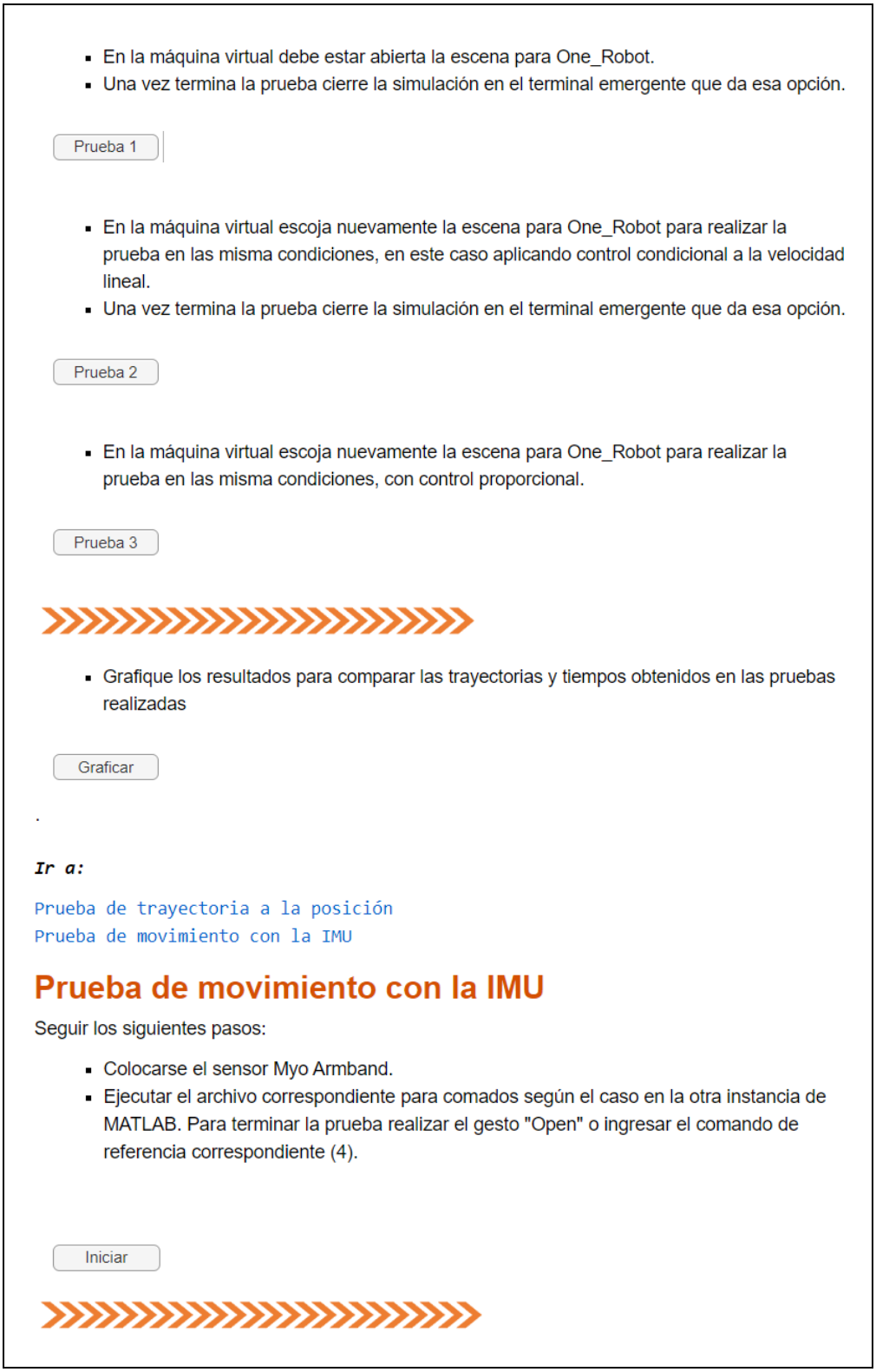


Figura IV.5 Interfaz para realizar pruebas con 1 agente (parte 2)

b. Si por el contrario, la escena multi-agente elegida es Three_Robot, entonces se abre el menú que permite comandar al sistema multi-agente que consta de 3 TurtleBot3 Burger, **menu.mlx**. A partir de esta sección ya es necesario los comandos para la selección de acción.

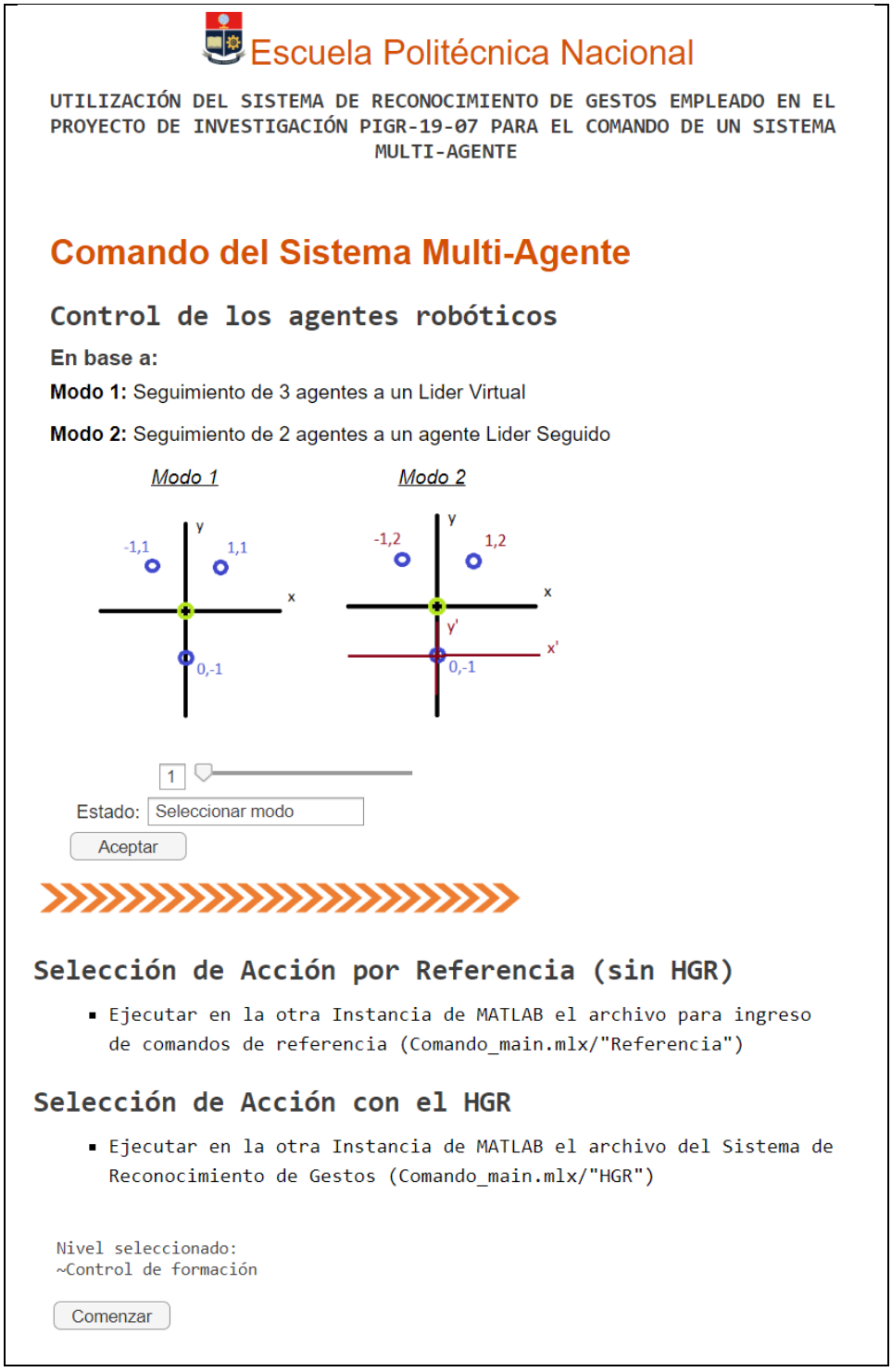


Figura IV.6 Interfaz del Menú Inicial para el Comando del Sistema Multi-Agente

3) Antes de continuar, es necesario abrir otra ventana de MATLAB, y en este caso se abre el archivo **Comando_main.mlx**, el cual permite abrir el programa de reconocimiento de gestos o enviar comandos de referencia.

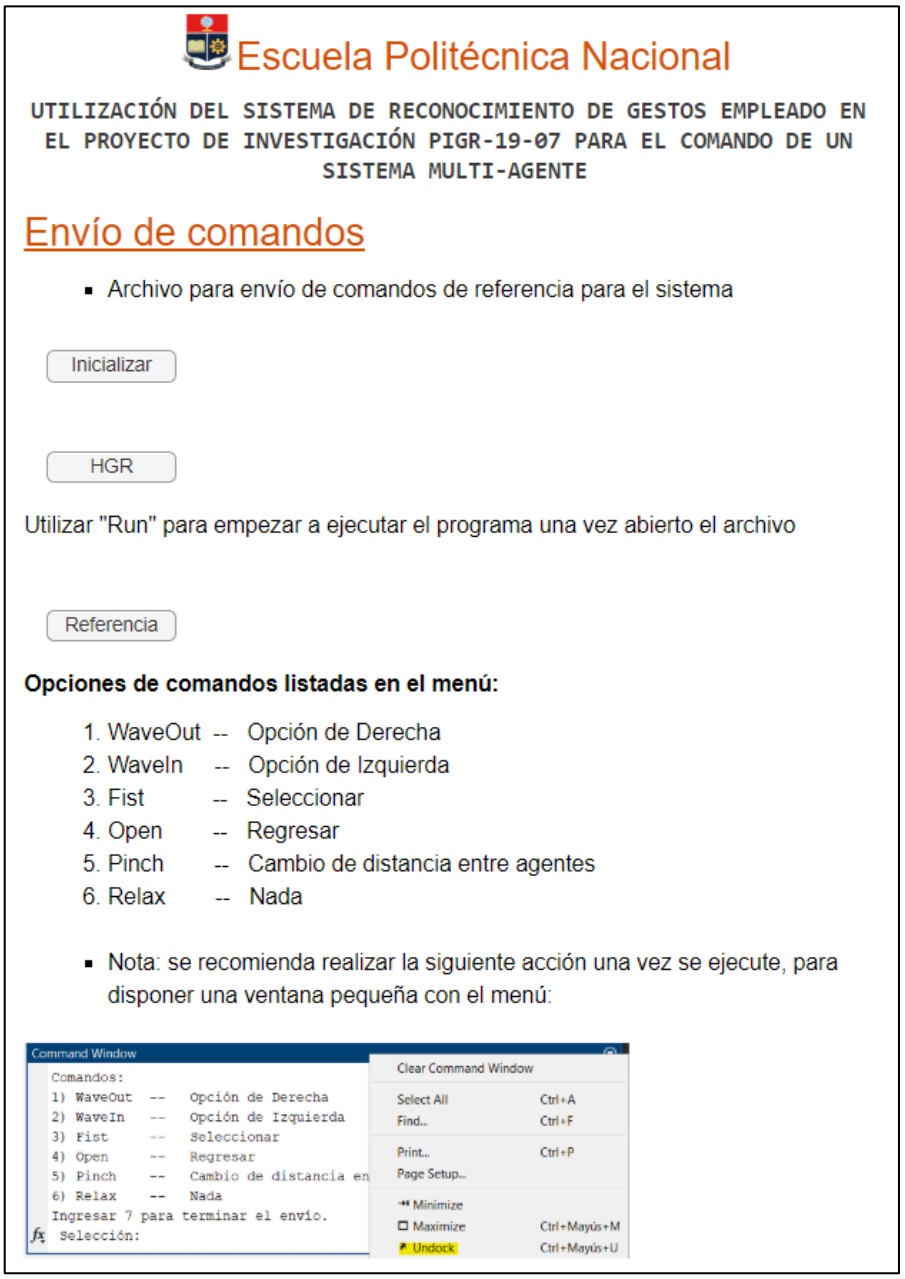


Figura IV.7 Interfaz inical para selección de envío de comandos

4) Una vez se posea ambas instancias de MATLAB abiertas se puede navegar entre Control de Formación y Movimiento, submenús de los Comandos del Sistema Multi-Agente y realizar las acciones que se requieran.

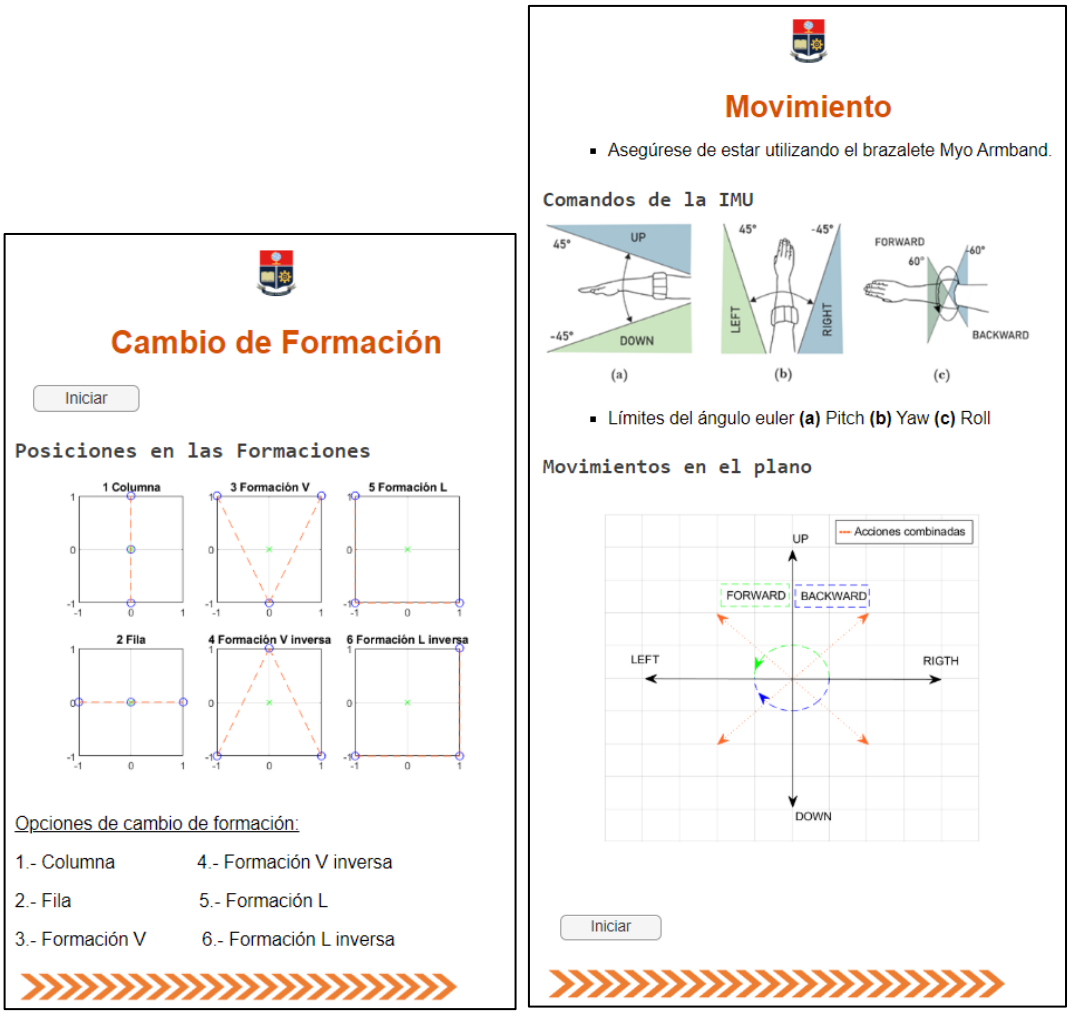


Figura IV.8 Interfaz para el Control de Formación y el Movimiento de trayectoria grupal

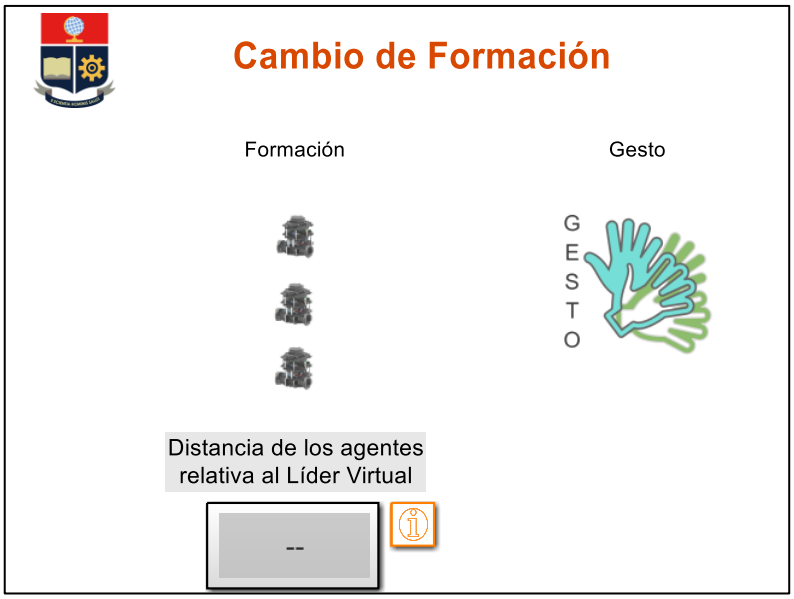
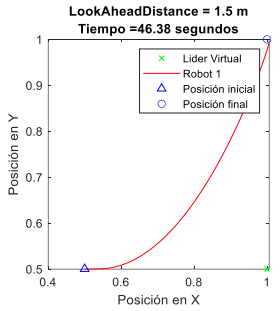


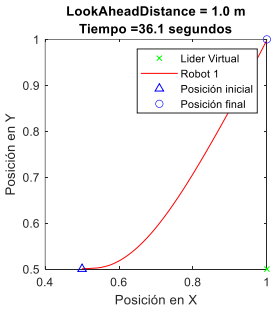
Figura IV.9 Parte de la Interfaz para el Control de Formación (Simulink)

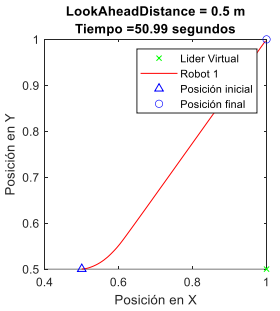
ANEXO V

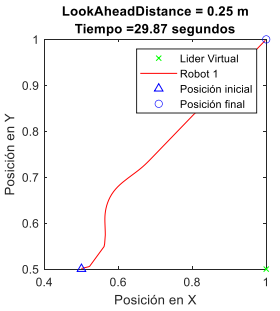
Pruebas con 1 agente

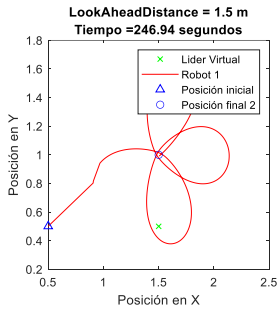
Pruebas de sintonización del parámetro LookAheadDistance

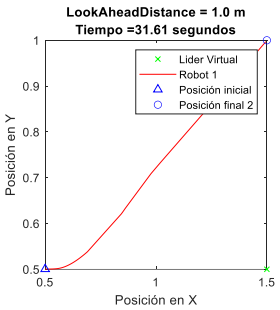


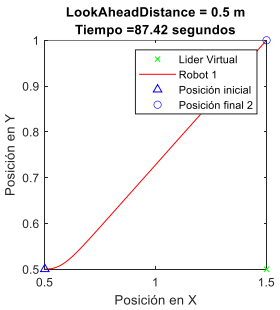


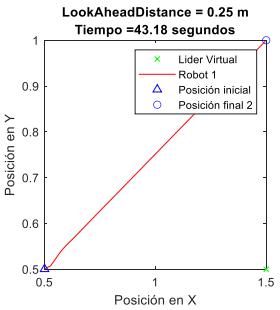


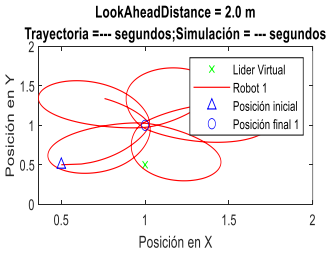


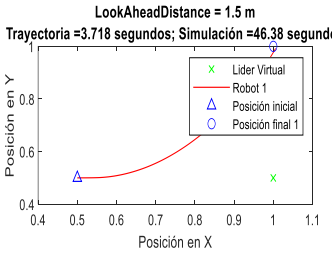


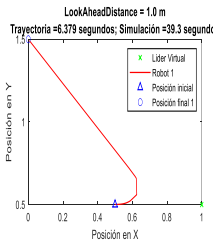


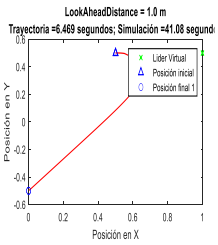


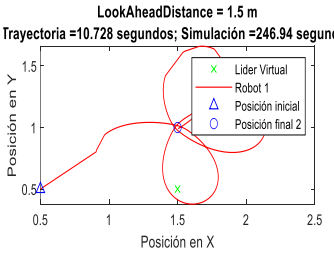


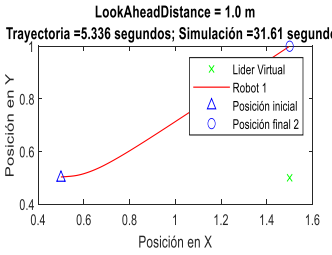


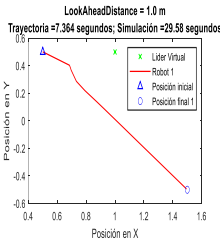


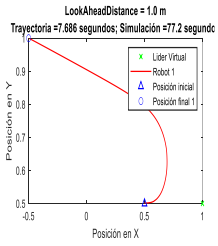


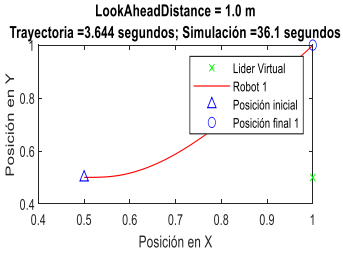


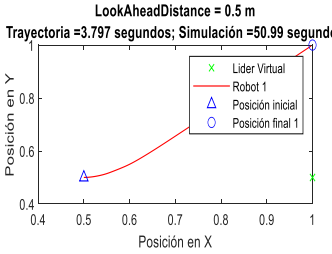












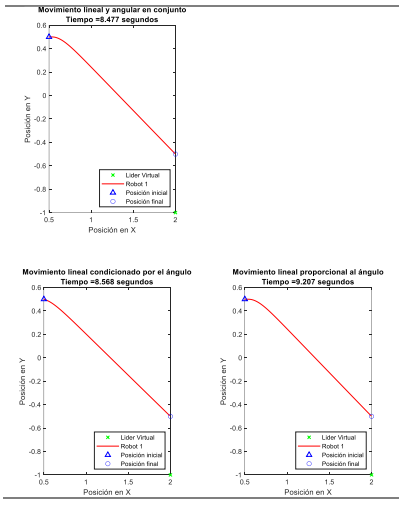
Pruebas de condiconalidad de la velocidad lineal respecto al ángulo de la meta

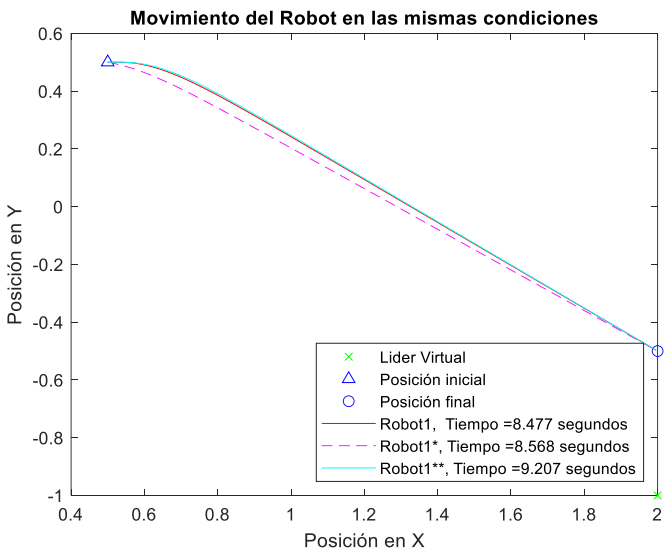
Se realizaron diez pruebas pero que se ha colocado por espacio solamente los resultados de 4, siendo estas las que se escogió como más representativas del objetivo de la prueba. A continuación, se muestra la tabla que resume los resultados y también se presenta en la Sección 3.1.1.2 – Condicionalidad de la velocidad lineal al ángulo.

Tabla V.1 Resultados de prueba de control de seguimiento con condicionamiento en la velocidad lineal a distintos puntos

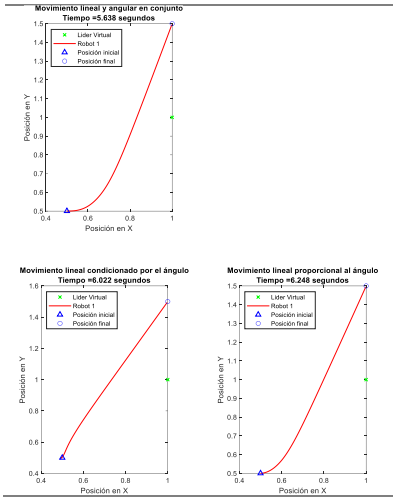
Punto de destino [x , y]	**Caso 1** Tiempo [seg]	Trayectoria	**Caso 2** Tiempo [seg]	Variación %	Tiempo	Trayectoria	**Caso 3** Tiempo [seg]	Variación %	Tiempo	Trayectoria
Prueba 1 [2.0 , -1.0]	8,477	0	8,568	-1,07	0	1	9,207	-8,61	-1	0
Prueba 2 [2.0 , 1.0]	8,554	0	8,597	-0,50	0	1	9,416	-10,08	-1	1
Prueba 3 [1.5 , -1.0]	6,802	0	7,004	-2,97	-1	1	7,249	-6,57	-1	0
Prueba 4 [2.5 , 1.0]	10,489	0	10,452	0,35	0	0	11,5	-9,64	-1	0
Prueba 5 [1.0 , 1.0]	5,638	0	6,022	-6,81	-1	0	6,248	-10,82	-1	0
Prueba 6 [0.0 , -1.0]	6,278	1	6,389	-1,77	-1	2	7,055	-12,38	-1	0
Prueba 7 [-0.5 , 1.0]	8,039	0	8,066	-0,34	0	2	8,232	-2,40	-1	1
Prueba 8 [1.5 , 0.0]	4,707	0	4,725	-0,38	0	0	5,142	-9,24	-1	0
Prueba 9 [0.5 , 1.0]	5,642	0	5,581	1,08	1	2	5,784	-2,52	-1	1
Prueba 10 [-0.5 , 0.0]	11,314	0	8,675	23,33	1	2	9,312	17,69	1	1
Puntuación		1		1,09	-1	11		-5,46	-8	4

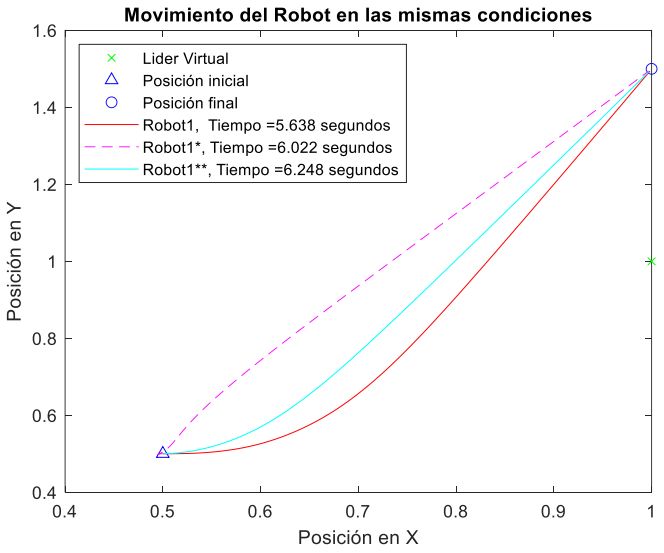
Prueba 1 [2.0 , -1.0]



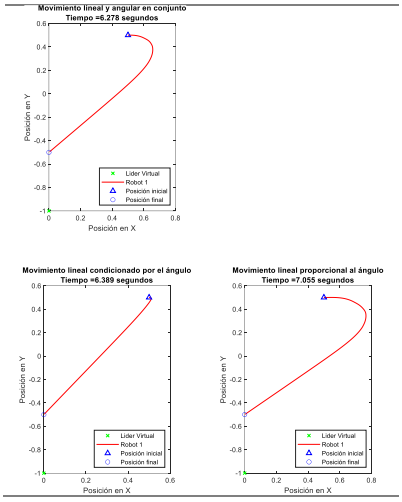


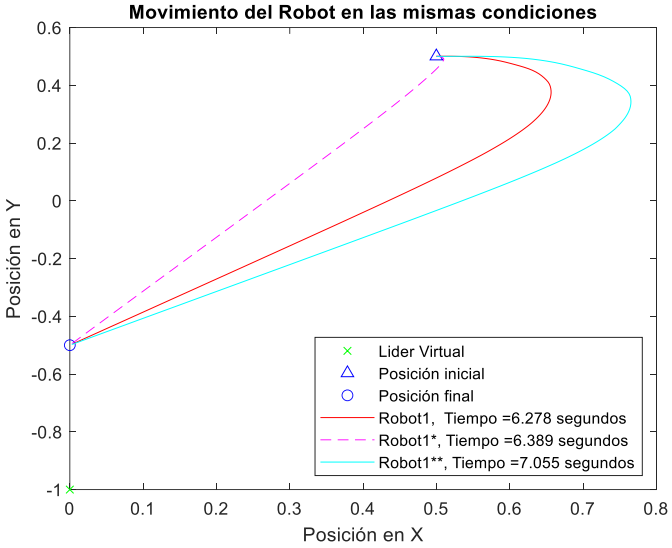
Prueba 5 [1.0 , 1.0]



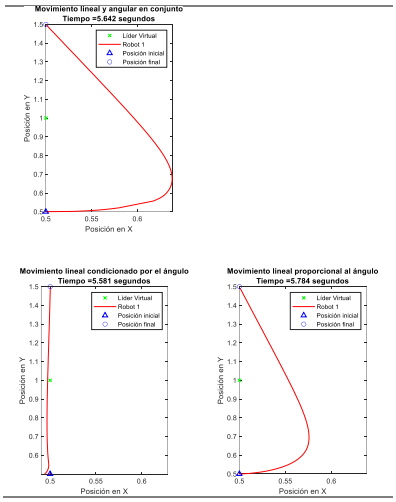


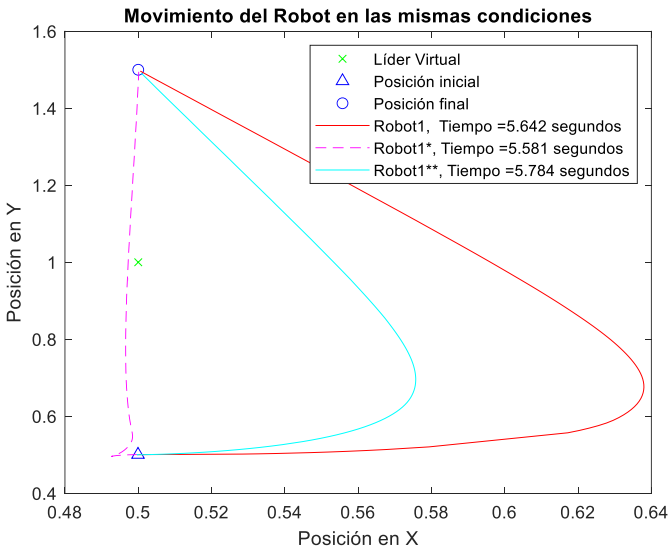
Prueba 6 [0.0 , -1.0]





Prueba 9 [0.5 , 1.0]





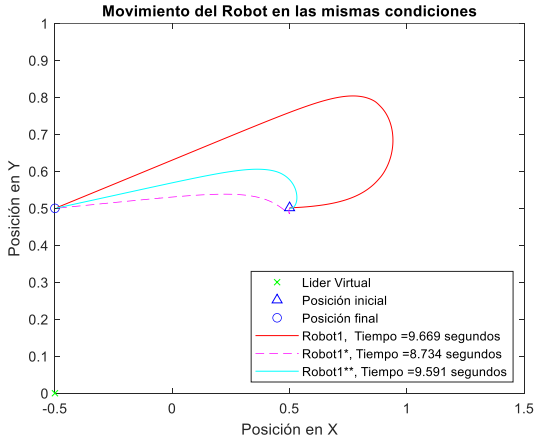
Pruebas hacia el punto [-0.5 , 0.0] (Giro de 180°)

A continuación, se muestra la tabla que resume los resultados y también se presenta en la Sección 3.1.1.2 – Condicionalidad de la velocidad lineal al ángulo, donde se menciona esta respectiva prueba. Por espacio, se muestran las figuras de las pruebas 1 a la 4.

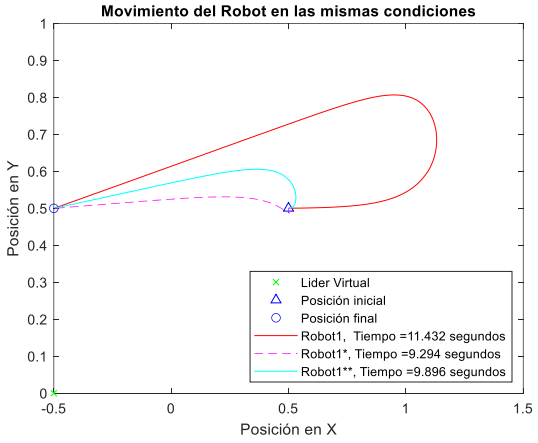
Tabla V.2 Resultados de prueba de control de seguimiento con condicionamiento en la velocidad lineal a un mismo punto

Punto de destino [-0.5 , 0.0]	**Caso 1** Tiempo [seg]	**Caso 2** Tiempo [seg]	% Variación	Tiempo	**Caso 3** Tiempo [seg]	% Variación	Tiempo
Prueba 1	9,669	8,734	9,67	1	9,591	0,81	0
Prueba 2	11,432	9,294	18,70	0	9,896	13,44	1
Prueba 3	11,238	9,432	16,07	1	9,989	11,11	1
Prueba 4	9,513	9,169	3,62	1	9,593	-0,84	0
Prueba 5	11,431	9,053	20,80	1	8,466	25,94	1
Puntuación			13,77	4		10,09	3

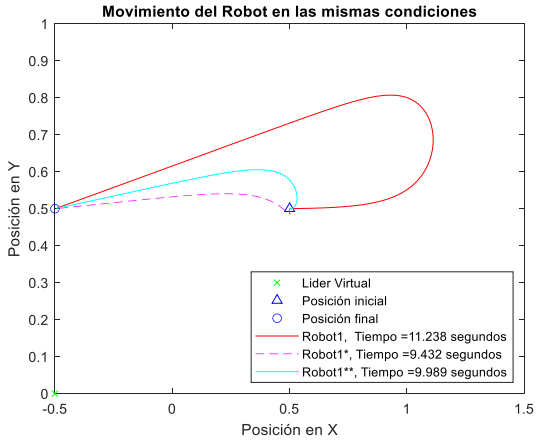
Prueba 1



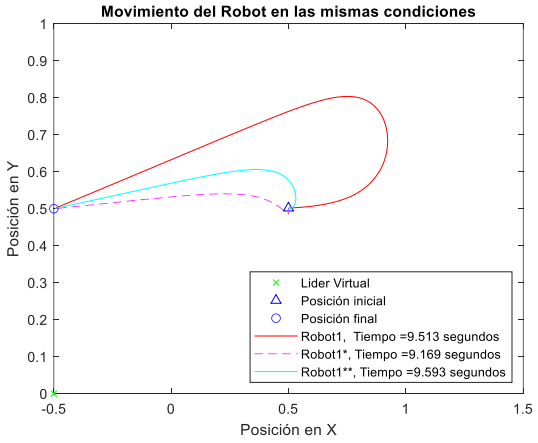
Prueba 2



Prueba 3



Prueba 4



ANEXO VI

Movimientos para la obtención de comandos de la IMU

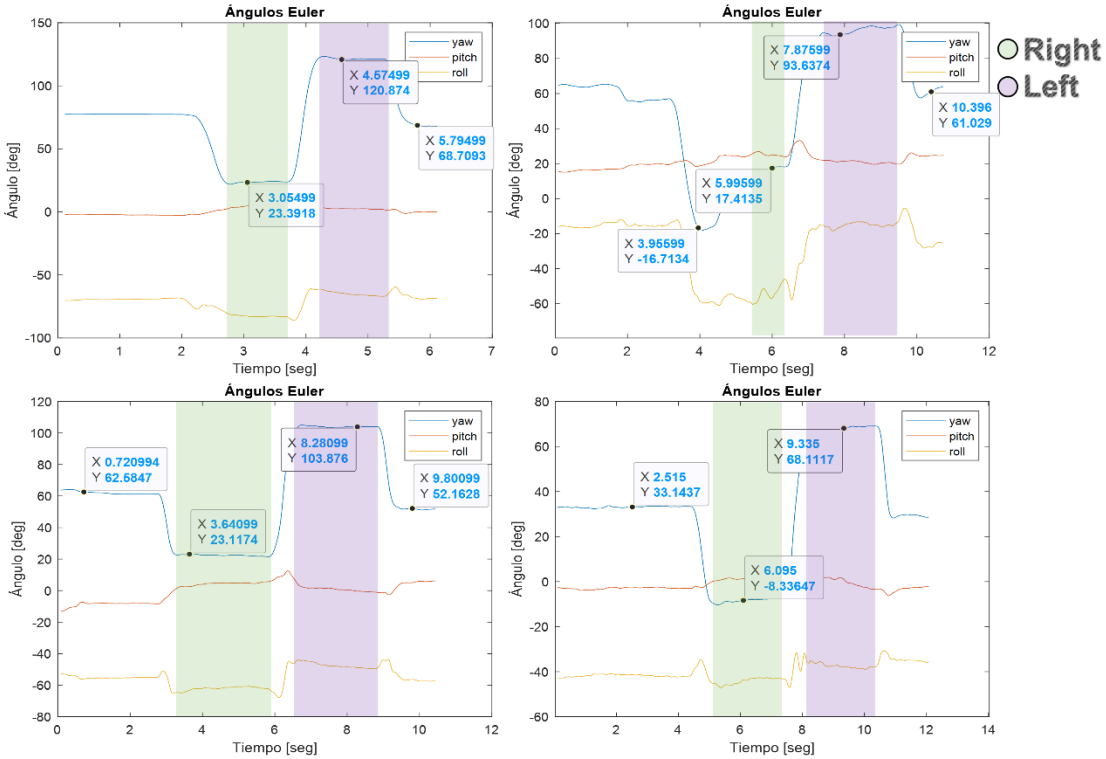


Figura VI.1 Movimientos para obtener los límites del ángulo Yaw

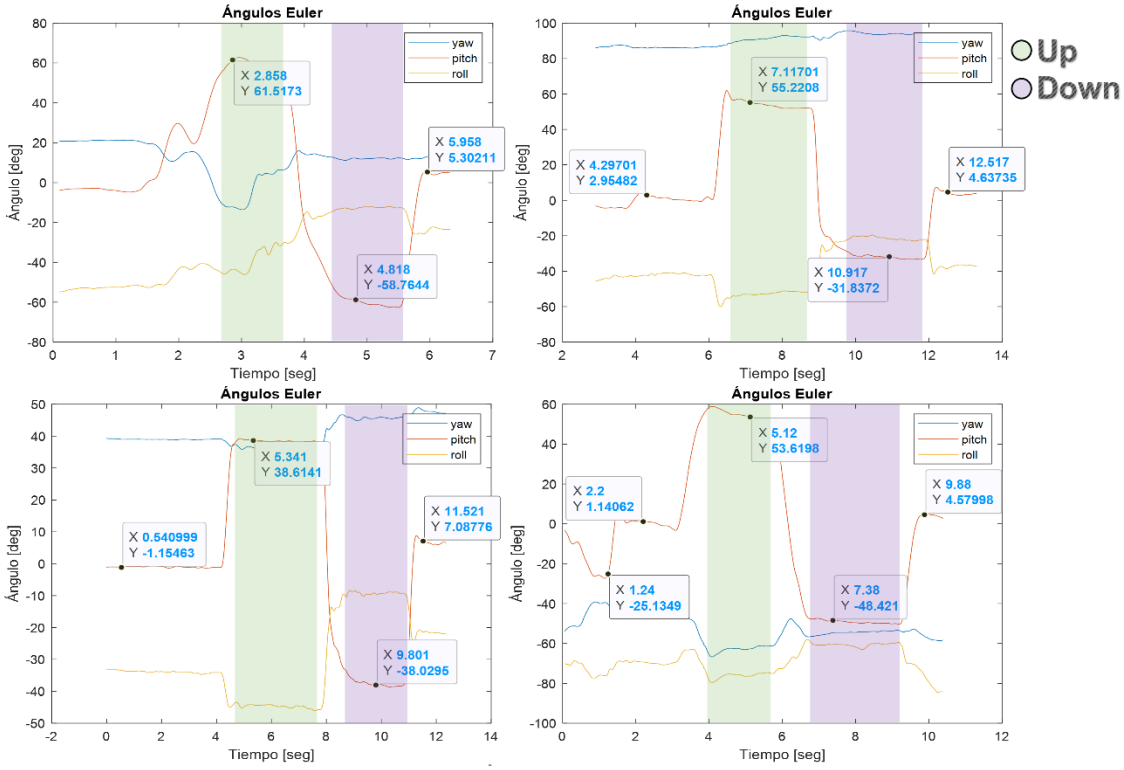


Figura VI.2 Movimientos para obtener los límites del ángulo Pitch

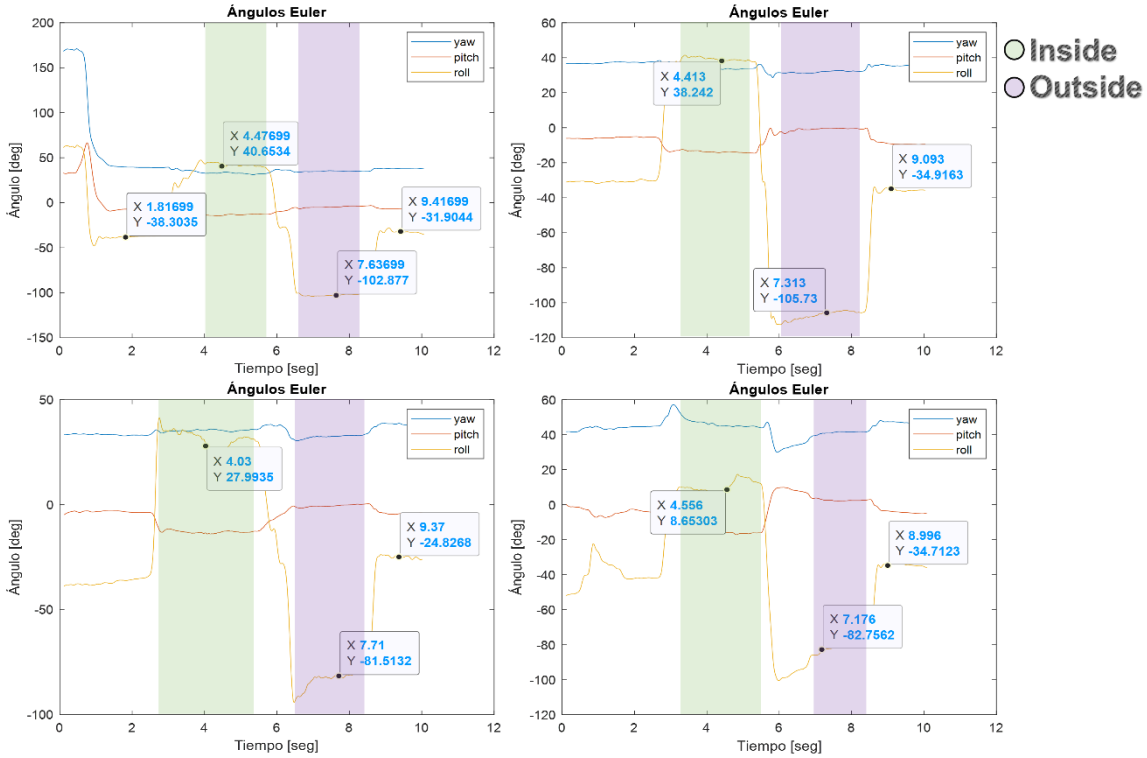


Figura VI.3 Movimientos para obtener los límites del ángulo Roll

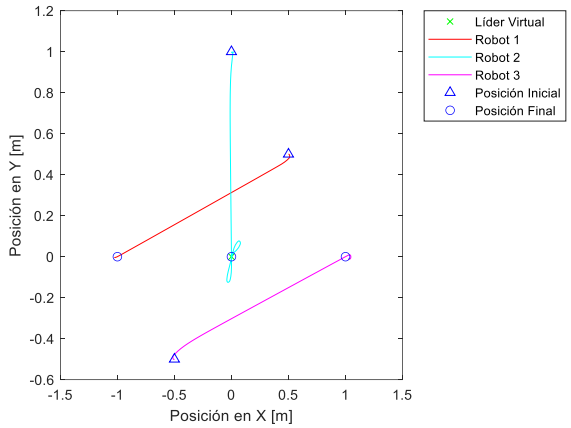
ANEXO VII

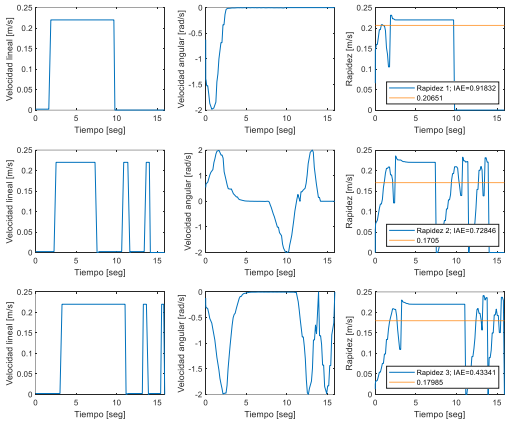
Pruebas de cambio de formación con comando de referencia

Para cada prueba se obtuvo la figura de la trayectoria realizada por los agentes y de las velocidades que emplearon. Se posee figuras de pruebas hacia las formaciones con una distancia relativa *d* de 1.0 m, distancia aplicada en las pruebas en las que se basan los datos expuestos en el escrito, siendo posible cambiar su distancia a 0.5 m, y ubicar a los agentes más juntos entre sí. Las figuras que se presentan a continuación son en las cuales se diferencia de mejor manera las trayectorias de los Modos de seguimiento.

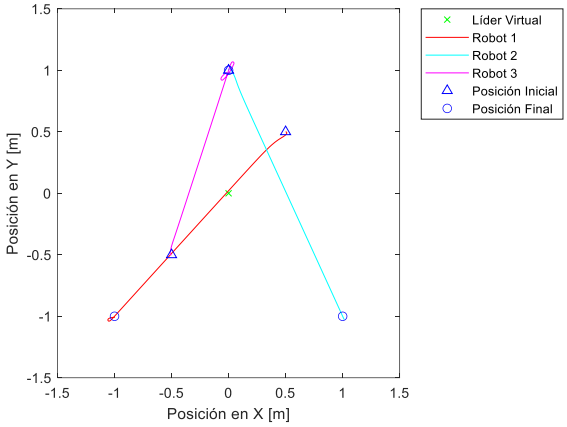
A partir de la posición inicial de la escena con el Modo 1

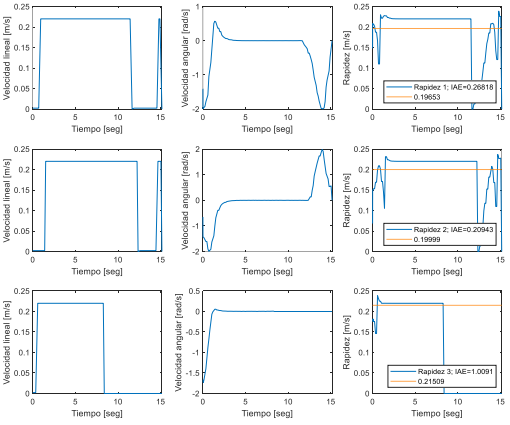
Formación 2



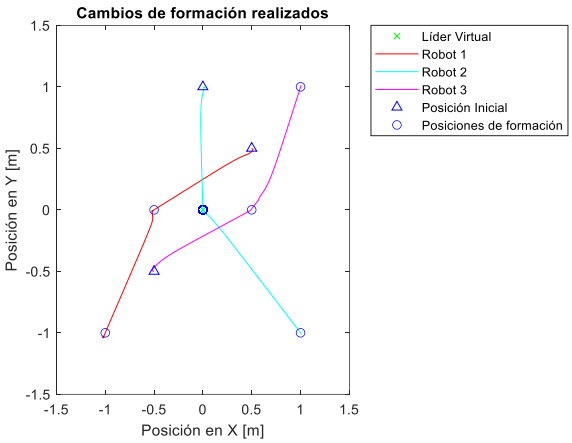


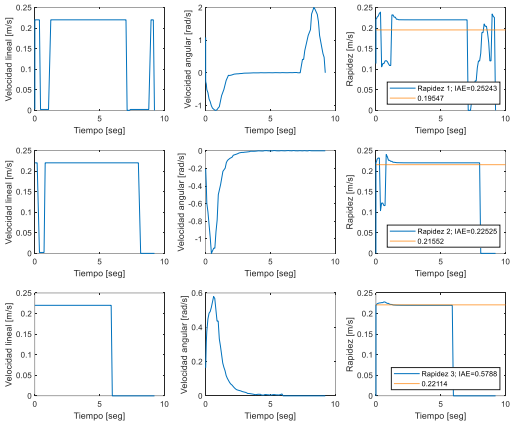
Formación 4





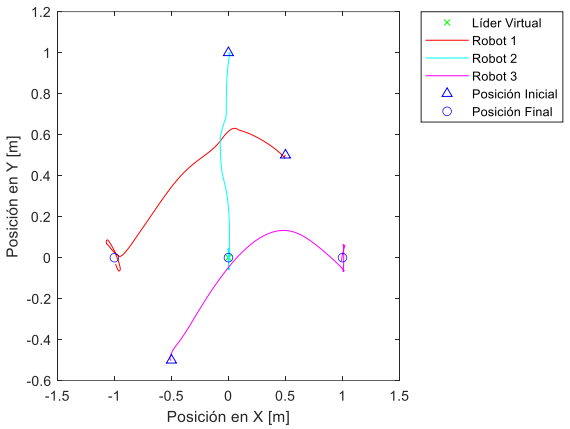
Formación 6

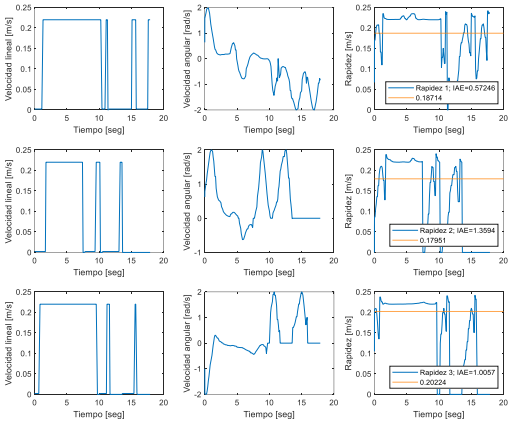




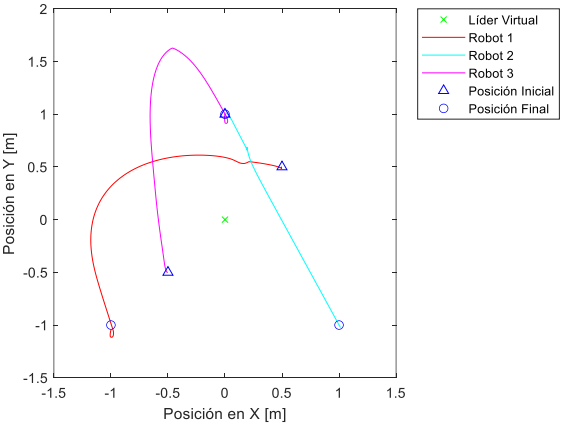
A partir de la posición inicial de la escena con el Modo 2

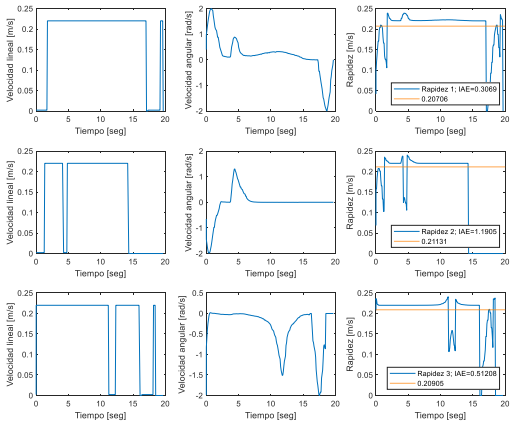
Formación 2



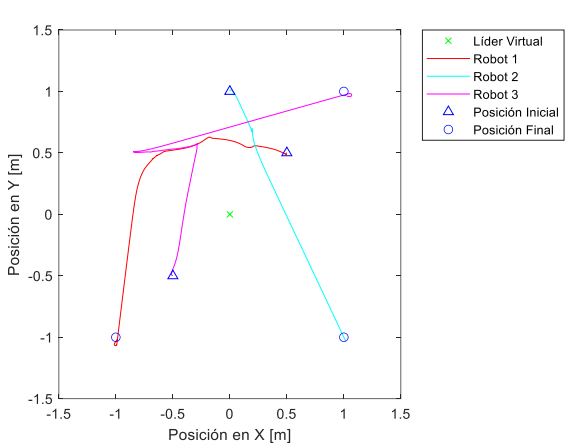


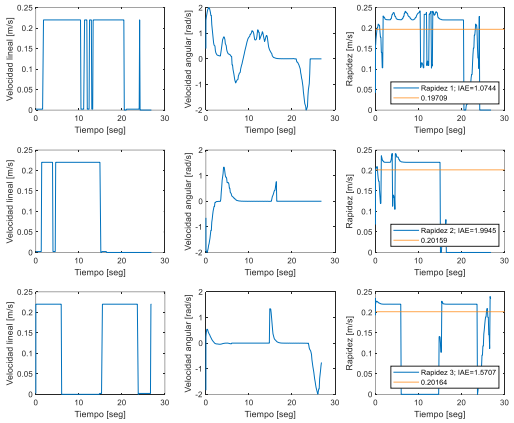
Formación 4



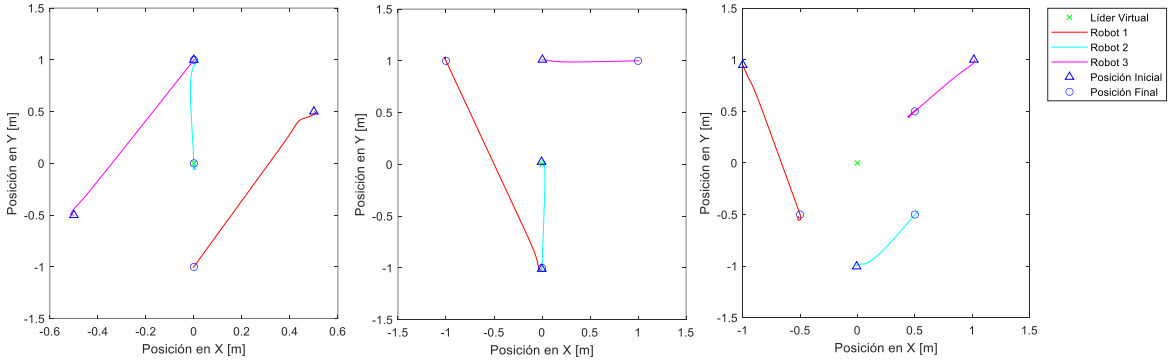


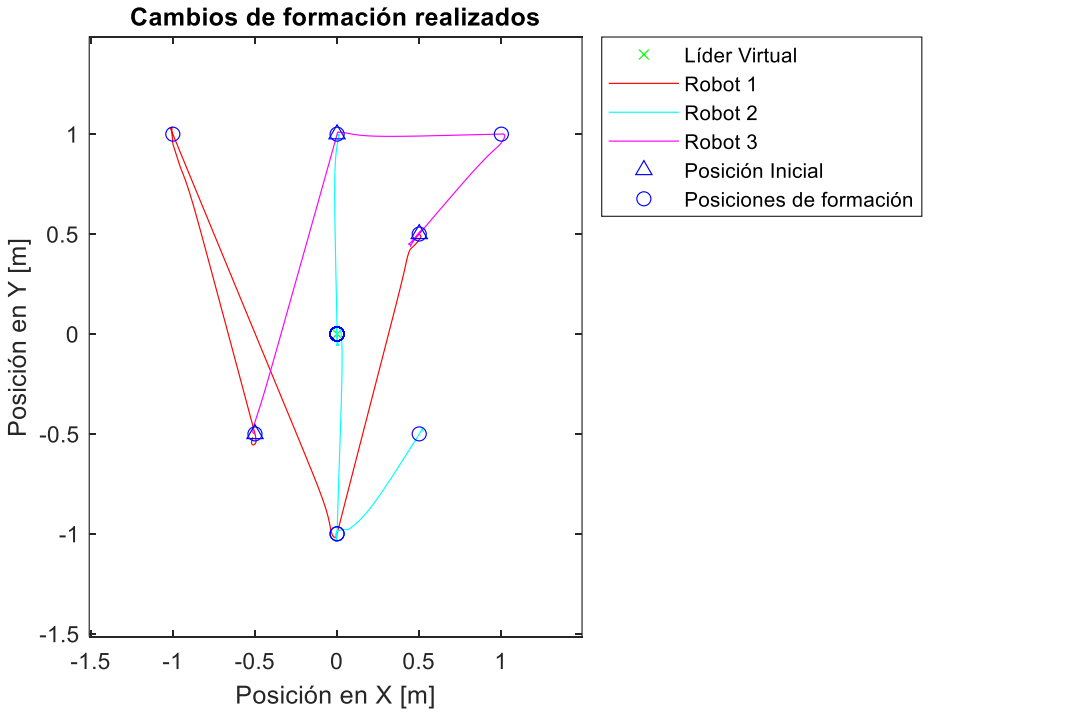
Formación 6





Emplenado una Secuencia de cambios de formación





ANEXO VIII

Videos de Funcionamiento

A continuación, se presenta el enlace que dirige hacia videos donde se puede observar el funcionamiento del sistema implementado.

Lista de reproducción en la plataforma de Youtube:

https://youtube.com/playlist?list=PLDPood9E6gIZDzyVxCg9xBSyCjIqkmu2J