
Validación de un algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente en un entorno físico

José Alejandro Rodríguez Porras



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



Validación de un algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente en un entorno físico

Trabajo de graduación presentado por José Alejandro Rodríguez Porras para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

Vo.Bo.:

(f) _____
Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) _____
Dr. Luis Alberto Rivera Estrada

(f) _____
—

(f) _____
—

Fecha de aprobación: Guatemala, -de -de 2023.

Prefacio

Llevar a cabo una de las primeras validaciones físicas de robótica de enjambre en Guatemala ha sido una de las mejores experiencias en el campo de investigación que he experimentado, y considero que es un paso en la dirección correcta para el crecimiento del área de la robótica en el país. Espero que en el futuro se continúe con el estudio del algoritmo validado en este trabajo, ya que tiene un gran valor de aplicación en un mundo constantemente cambiante a nivel tecnológico.

Que el lector encuentre el valor aportado en este proyecto y lo use para seguir desarrollando nuevas aplicaciones de robótica de enjambre.

A lo largo de este año he tenido muchas lecciones importantes en mi vida, y reconozco que gracias a la oportunidad de trabajar en este proyecto he desarrollado más mi potencial en el área de investigación y búsqueda del conocimiento, así como adaptabilidad.

Quisiera agradecer a las siguientes personas que me ayudaron a lograr este trabajo:

■ A mi familia:

A mis hermanas Dani y Marisa por siempre estar allí para mí, con apoyo y diversión. A mi hermano y perro, Timoteo, quien me acompaña desde que era un niño y compartimos un amor incondicional. A mis padres, por siempre apoyarme tanto en mi vida como mis logros, en las buenas y en las malas, que sé que siempre lo han hecho con amor.

■ A mis amigos de la carrera, especialmente a Jorge Lanza, Fernando Caceres, Fredy Godoy y Juan Diego Villafuerte, realmente estoy agradecido por haberlos conocido y espero sigamos creando buenos proyectos.

■ A mi pareja, Dina, por estar a mi lado con bondad.

■ A las personas que me han aportado de su conocimiento de buena forma

■ A mi asesor, el Dr. Luis Rivera, por la disponibilidad de apoyo, resolución de dudas, aporte de consejos en mi trabajo y compartir su conocimiento conmigo.

- A Andrea Maybell Peña y su buen trabajo en el desarrollo del algoritmo en la fase de simulación previa a mi trabajo.

Prefacio	IV
Lista de figuras	X
Lista de cuadros	XI
Resumen	XII
Abstract	XIII
1. Introducción	1
2. Antecedentes	3
2.1. Robótica de enjambre	3
2.2. Kilobots	4
2.3. WsBot	4
2.4. Robotat	6
2.5. Algoritmos de Robótica de Enjambre implementados en UVG	6
2.5.1. Particle Swarm Optimization (PSO)	6
2.5.2. Ant Colony Optimization	8
2.5.3. Algoritmo de sincronización y control de sistemas de robots multi-agente para misiones de búsqueda	9
2.6. Plataformas Robóticas Móviles en UVG	11
3. Justificación	13
4. Objetivos	15
5. Alcance	16
6. Marco teórico	18
6.1. Definiciones importantes de robótica de enjambre	18
6.1.1. Agente	18
6.1.2. Formaciones	18

6.1.3. Control centralizado	19
6.1.4. Control descentralizado	19
6.2. Teoría de grafos	19
6.2.1. Conceptos básicos en teoría de grafos	19
6.2.2. Matrices de interés	21
6.3. Teoría de control	22
6.3.1. Control de formaciones	22
6.3.2. Modelo del robot diferencial	23
6.4. Herramientas matemáticas relevantes	24
6.4.1. Ecuación de consenso	24
6.4.2. Función de tensión	24
6.5. Infraestructura en Robotat	25
6.5.1. Mesa de pruebas	25
6.5.2. OptiTrack	25
6.5.3. Comunicación del Robotat	26
6.6. Software	27
6.6.1. Matlab	27
6.6.2. Webots	27
6.7. Hardware	28
6.7.1. Plataforma Móvil: Robot Diferencial Pololu 3Pi+ modificado	28
6.7.2. Microcontrolador del Agente: ESP32	30
7. Algoritmo de sincronización y control de formaciones	32
7.1. Funcionamiento	32
7.1.1. Algoritmo de sincronización y control centralizado (Supervisor)	32
7.1.2. Algoritmo de control de uniciclo para cada agente	33
7.2. Lógica	33
7.3. Restauración y actualización del algoritmo	36
7.3.1. Recreación del mundo de Webots	37
7.3.2. Actualización del código de 2019 para su funcionamiento en 2023	38
7.4. Variantes	45
7.4.1. Creación de subgrupos de formación	45
7.4.2. Alejamiento de agentes en formación	46
8. Adaptación del algoritmo para su ejecución en un entorno físico	47
8.1. Migración de funciones	47
8.1.1. Comunicación con servidor Robotat	48
8.1.2. Control de Pololu 3Pi+	48
8.2. Modificaciones al algoritmo en entorno Webots con pruebas preliminares	51
8.2.1. Calibración de marcadores para su uso en el algoritmo	53
8.2.2. Modelo 1: Aplicación de información de marcadores	54
8.2.3. Modelo 2: Pruebas preliminares de algoritmo dinámico en físico	56
8.2.4. Modelo 3: Integración para funcionamiento híbrido de algoritmo en físico y almacenamiento de datos	67
9. Diseño Experimental	69
9.1. Opciones de configuración	69
9.2. Configuración de escenario	71

10. Ejecución de algoritmo de sincronización y control de formaciones en los robots diferenciales en ambiente controlado	72
10.1. Resultados	72
10.1.1. Experimento 1: Ejecución del algoritmo completo en físico con 4 agentes	72
10.1.2. Experimento 2: Reproducibilidad del algoritmo en físico con 4 agentes	75
10.1.3. Experimento 3: Visualización de gráficas generadas por la ejecución del algoritmo en físico con 4 agentes con distintas condiciones iniciales de agente pero mismas marcas de inicio.	77
10.1.4. Experimento 4: Ejecución del algoritmo completo en físico con 5 agentes y generación de gráficas de velocidad en x y y, así como el análisis de la trayectoria de cada agente involucrado.	81
10.2. Comparación de Resultados	85
11. Conclusiones	86
12. Recomendaciones	87
13. Bibliografía	88
14. Anexos	91

Lista de figuras

1.	Taxonomía de la robótica de enjambre [3].	4
2.	Kilobots [5].	5
3.	WsBot [6].	5
4.	WsBot en ambiente de trabajo [6].	6
5.	Trayectorias generadas de PSO para un enjambre de 10 agentes [8].	7
6.	PSO en la naturaleza [9].	7
7.	Diagrama de Feromona y camino encontrado [12].	8
8.	Trayectoria del camino encontrado con feromona [13].	8
9.	Movimiento en formación mínimamente rígida a través de obstáculos [16].	9
10.	Simulación en Webots de la formación moviéndose a través de obstáculos [16].	9
11.	Control de formaciones usando un grafo rígido en Matlab [16].	10
12.	Formación inicial experimentando deformación para estar en las posiciones adecuadas [16].	11
13.	Simulación en Webots de formación y dirección hacia meta [16].	11
14.	Ejemplo de un grafo.	20
15.	Ejemplo de dígrafo.	21
16.	Lazo cerrado de control.	22
17.	Modelo de un uniciclo en 2D.	23
18.	Ejemplo de sistema de captura de movimiento OptiTrack [30].	26
19.	Cámara de Captura de Movimiento Prime ^x 41 [30].	26
20.	Entorno de desarrollo en Webots.	28
21.	Pololu 3Pi+ [33].	29
22.	Especificaciones de sensores del Pololu 3Pi+ [33].	29
23.	Especificaciones del Pololu 3Pi+ [33].	30
24.	ESP32 Pinout [35].	31
25.	Diagrama de flujo para el Supervisor/Algoritmo de Sincronización.	34
26.	Diagrama de flujo para procesamiento de velocidades en agente individual.	35
27.	Estado del mundo de Webots creado en 2019 al abrirlo con Webots 2023b.	37
28.	Recreación del mundo de Webots en versión 2023b.	38
29.	Prueba de funcionamiento de formación.	42
30.	Prueba de funcionamiento de movimiento en formación hacia la meta.	43

31.	Resultado de optimización de parámetros para formación.	44
32.	Variante de creación de subgrupos de formación.	45
33.	Variante de alejamiento de agentes en formación.	46
34.	Obtención de poses de múltiples marcadores usando las funciones migradas a Python.	48
35.	Conexión con Pololu 3Pi+ con ID 7, usando las funciones migradas a Python.	49
36.	Envío de instrucciones para poner a girar a agentes Pololu, usando las funciones migradas a Python.	50
37.	Envío de instrucciones para mover en línea recta a agentes Pololu, usando las funciones migradas a Python.	51
38.	Mundo de Webots basado en Robotat y optimizado para pruebas.	52
39.	Calibración de marcadores 1 al 15 para su uso en Pololus 3Pi+.	53
40.	Proceso de calibración de ángulo de desfase de <i>bearing</i>	54
41.	Escenario de primera prueba en el Robotat, visto desde distintos ángulos.	55
42.	Resultado al cargar la información guardada del mundo real en Webots.	55
43.	Movimiento en círculos alrededor del objetivo, del modelo 2.0.	56
44.	Secuencia del movimiento de alineación perpendicular del Modelo 2.0.	57
45.	Descripción del movimiento de alineación perpendicular del Modelo 2.0.	57
46.	Orientación del E-Puck vs. Pololu con respecto de sus ejes.	58
47.	Secuencia de seguimiento dinámico de objetivo, prueba 1.	58
48.	Descripción del seguimiento dinámico de objetivo, prueba 1.	59
49.	Secuencia de seguimiento dinámico de objetivo, prueba 2.	59
50.	Descripción del seguimiento dinámico de objetivo, prueba 2.	60
51.	Descripción del seguimiento dinámico de objetivo, prueba 3.	60
52.	Agente detenido al llegar al marcador objetivo.	61
53.	Descripción de movimiento de 1 agente hacia objetivo virtual, prueba 1.	61
54.	Secuencia de movimiento de 2 agentes hacia objetivos virtuales, prueba 2.	62
55.	Descripción de movimiento de 2 agentes hacia objetivos virtuales, prueba 2.	62
56.	Secuencia de movimiento de 9 agentes, para colocarse en posiciones iniciales en línea, prueba 1.	63
57.	Descripción de movimiento de 9 agentes, para colocarse en posiciones iniciales en línea, prueba 1.	64
58.	Escenario de posiciones iniciales para prueba 2 con obstáculos virtuales.	64
59.	Secuencia (izquierda a derecha) de movimiento de 9 agentes físicos y 1 agente inmóvil (marcador) hacia objetivos virtuales en forma de L invertida, prueba 2.	65
60.	Secuencia (izquierda a derecha) de movimiento de 9 agentes físicos y 1 agente inmóvil (marcador) para acercar a los agentes entre sí, prueba 1.	66
61.	Resultado de la formación con 9 agentes físicos y 1 agente inmóvil (marcador), prueba 2.	66
62.	Secuencia (izquierda a derecha) del resultado del movimiento artificial del líder (marcador agente 1) y el seguimiento de la formación (agentes 2 al 10).	67
63.	Diagrama de diseño experimental 1.	71
64.	Secuencia Experimento 1: primera fase del algoritmo, posicionamiento de agentes en marcas iniciales.	73

65.	Secuencia Experimento 1: segunda fase del algoritmo, acercamiento de agentes y construcción de formación.	73
66.	Secuencia Experimento 1: tercera fase del algoritmo, movimiento de líder hacia objetivo manteniendo formación.	74
67.	Secuencia Experimento 1.1: Verificación del comportamiento dinámico de los agentes en formación luego de llegar al objetivo.	75
68.	Secuencia Experimento 2: Reproducibilidad de algoritmo en físico con 4 agentes. .	76
69.	Trayectoria desde marcas iniciales de los 4 agentes, excluyendo el camino hacia las marcas iniciales.	77
70.	Histórico de velocidades, incluyendo el camino hacia las marcas iniciales. . . .	78
71.	Trayectoria desde marcas iniciales de los 4 agentes, excluyendo el camino hacia las marcas iniciales.	79
72.	Histórico de velocidades, excluyendo el camino hacia las marcas iniciales. . . .	80
73.	Secuencia Experimento 4: Ejecución del algoritmo completo en físico con 5 agentes.	81
74.	Trayectoria completa de los 5 agentes, incluyendo el camino hacia las marcas iniciales.	82
75.	Histórico de velocidades, incluyendo el camino hacia las marcas iniciales. . . .	83
76.	Trayectoria de los 5 agentes a partir de las marcas iniciales.	84
77.	Histórico de velocidades, a partir de las marcas iniciales.	85
78.	Perspectiva adicional 1 de la adquisición de datos para la calibración de los marcadores.	91
79.	Perspectiva adicional 2 de la adquisición de datos para la calibración de los marcadores.	92
80.	Trayectoria desde marcas iniciales de los 4 agentes del experimento 1, incluyendo el camino hacia las marcas iniciales.	92

Lista de cuadros

1.	Resultados de éxito de simulación en la fase previa [16].	10
2.	Comparación de disposición de ejes de mundo de Webots, según el año.	40
3.	Comparación de indexación de posiciones de mundo de Webots, según el año.	41
4.	Desfases angulares al alinear los marcadores con el eje y.	54
5.	Tabla de variables almacenadas al final de la corrida del algoritmo.	70

Resumen

En el presente trabajo se llevó a cabo una validación de un algoritmo previamente desarrollado en 2019, específicamente el algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente. La validación se llevó a cabo en un entorno físico, específicamente en el ecosistema de robots existente en la universidad, llamado Robotat, usando un robot diferencial Pololu 3Pi+ modificado para servir como agente del enjambre.

Como primer paso se realizó una migración del software previamente desarrollado en Matlab hacia Python con la adición de nuevas funciones y características, para facilitar su compatibilidad con el ambiente de Webots, haciendo los ajustes y variantes del algoritmo necesarias para optimizar el proceso de validación.

La validación física consistió en realizar una serie de diversos experimentos para poner a prueba el algoritmo en el ambiente controlado. Entre los factores de evaluación más importantes se hizo énfasis en el estudio del desempeño de cada agente, la generación de las trayectorias, el posicionamiento inicial, intermedio y final de los agentes, las configuraciones de formación, los diversos escenarios de interés con obstáculos, etc. En síntesis, se examinó la ejecución del algoritmo para observar su desempeño, encontrando puntos de mejora y posibles aplicaciones según los escenarios propuestos.

Abstract

In the present work, the validation of an algorithm previously developed in 2019 was carried to completion, specifically focused in the synchronization and formation control algorithm for multi-agent robotic systems. The validation was carried out in a physical environment, to be specific the robot ecosystem at the university, called Robotat, using a modified Pololu 3Pi+ differential robot to serve as a swarm agent.

As first step, a migration from a previously developed software in Matlab was done, towards Python, with the addition of new functions and characteristics, to facilitate the compatibility of the Webots environment, making the necessary variants and adjustments to optimize the validation process.

The validation consisted in making a series of diverse experiments to test the algorithm in the controlled environment. Among the most important evaluation factors the emphasis was in the study of the performance of each agent, the trajectory generation, initial, intermediate and final positioning of the agents, the formation configurations, the diverse settings of interest with obstacles, etc. In summary, the execution of the algorithm was examined to observe its performance, find its improvement areas and possible applications according to the proposed scenarios.

CAPÍTULO 1

Introducción

El presente trabajo consistió en la continuación de la línea de investigación de desarrollo de robótica de enjambre, específicamente con enfoque en el desarrollo y aplicación del algoritmo de sincronización y control de formaciones de sistemas robóticos multi-agente. Se trató de una continuación debido a que en la fase previa se llegó hasta pruebas del algoritmo desarrollado en simulaciones, y en el presente trabajo ya se contaba con la infraestructura necesaria para las pruebas en un ambiente controlado con agentes físicos.

El objetivo principal giró en torno a la validación del algoritmo previamente mencionado en sistemas físicos, con su respectiva evaluación de funcionamiento en el ecosistema Robotat. Para cumplir con el mismo, se utilizaron las herramientas disponibles en la universidad como apoyo, entre las cuales se puede mencionar: el sistema de captura de movimiento OptiTrack, la plataforma de pruebas, los robots diferenciales Pololu 3Pi+ modificados y la red de comunicación TCP/IP existente en el Robotat.

Como tareas principales que se llevaron a cabo de forma específica para lograr llegar al objetivo principal, se asignó la tarea de migrar el software previamente desarrollado para su uso en 2023 en el Robotat. Una vez logrado, el siguiente paso propuesto fue la evaluación de trayectorias usando los marcadores del OptiTrack, siguiendo con la observación de la ejecución del algoritmo en la mesa de pruebas con los Pololu 3Pi+ y finalmente una comparación de resultados físicos con simulados.

El contenido relacionado a las tareas llevadas a cabo se muestra en los Capítulos 7 al 10. En los dos primeros Capítulos se detalla sobre el proceso para integración de funcionamiento. En el Capítulo 7 se aborda la explicación y restauración del algoritmo, luego, en el Capítulo 8 se elabora sobre la adaptación del algoritmo para su funcionamiento en un entorno físico real. En los últimos Capítulos se muestran los diseños de experimentos (Capítulo 9) y sus resultados (Capítulo 10).

Con la validación de este algoritmo se creó un buen cimiento para la continuación con la creciente tecnología por la robótica de enjambre. Observar la ejecución de las trayectorias generadas por el algoritmo en un entorno controlado representa un paso significativo en el

campo, pues abre las puertas a nuevas aplicaciones y un mayor potencial de este tema en el mundo moderno.

CAPÍTULO 2

Antecedentes

2.1. Robótica de enjambre

Conforme ha avanzado la tecnología en el área de la ingeniería, el área de robótica ha ido creciendo tanto en capacidad como complejidad, lo que se puede observar en proyectos como el Atlas de Boston Dynamics [1], el cual presenta grandes capacidades de movilidad, pero que también puede llegar a ser muy costoso por las mismas razones. Habiendo mencionado sus fortalezas, cabe resaltar que este trabaja de forma individual, lo que resulta siendo una limitación para distintas aplicaciones que involucran tareas que necesitan más de un individuo, como es el caso de búsquedas de rescate, explorar un área, o incluso realizar varias tareas de forma paralela. Es en estos casos donde entra el campo de investigación de la robótica de enjambre, el cual, contrario a Atlas se basa en robótica “simple”.

El área de robótica de enjambre consiste en realizar alguna tarea compleja a gran escala usando múltiples robots de construcción y capacidad simple. Estos robots suelen ser de bajo costo y consumo, de dimensiones relativamente pequeñas y su funcionamiento a nivel individual no es demasiado exigente en términos de procesamiento, pero al momento de trabajar como conjunto estos pueden desempeñar tareas a niveles de alta complejidad [2], teniendo así una operación multiagente. Esta área de la robótica está inspirada en sistemas en la naturaleza compuestos por múltiples individuos, como es el caso de insectos sociales como las colmenas de abejas y colonias de hormigas, bancos de peces, etc. Entre las principales ventajas de la robótica de enjambre se destacan la robustez, flexibilidad y escalabilidad. A grandes rasgos esta disciplina puede tomar modelos de la naturaleza y adaptarlos al ámbito de control para los robots, o bien, se pueden crear nuevos algoritmos [3].

En el diagrama de la Figura 1 se muestran los principales ejes de estudio de la robótica de enjambre, los cuales tienen ramificaciones según su aplicación.

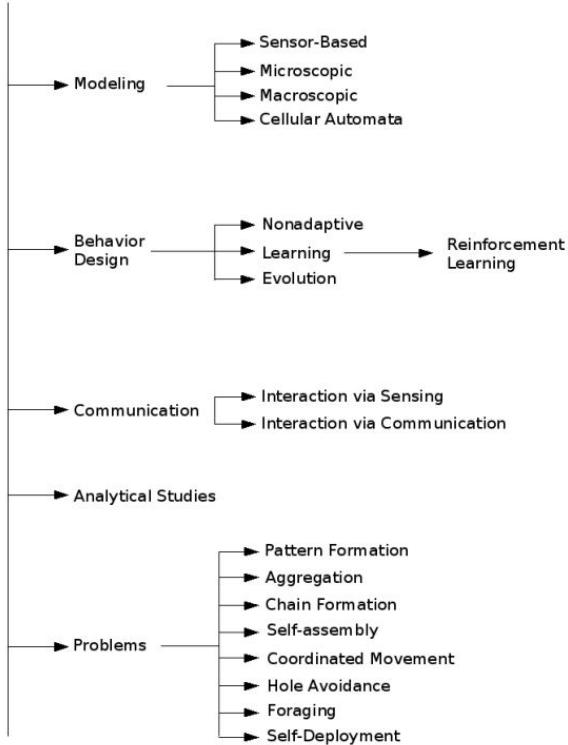


Figura 1: Taxonomía de la robótica de enjambre [3].

2.2. Kilobots

En 2014 la Universidad de Harvard (Instituto de Wyss) desarrolló estudios con enjambres de hasta 1024 agentes, los cuales se denominan Kilobots (Figura 2), cuyas dimensiones son muy cercanas a las de un centavo estadounidense y un costo menor a los 20 dólares. El enjambre que formó parte del estudio fue de 1024 robots con lo que se logró realizar distintas formaciones bidimensionales tanto artificiales como imitaciones de la naturaleza en entre 2 y 12 horas según la complejidad de la figura. Estos robots tienen la capacidad de comunicarse entre sí de forma local a través de señales de luz infrarroja y descentralizada, con la característica de tener una alta robustez.

El principio de movimiento de Kilobot se basa en vibraciones, estos poseen tres patas y según a donde se necesite mover estos vibran de una manera específica para moverse en esa dirección. La meta principal detrás del desarrollo de estos robots es desarrollar agentes cada vez más pequeños para que estos se puedan ensamblar en objetos como herramientas o bien mover objetos en conjunto [4].

2.3. WsBot

En 2019 se desarrolló el WsBot, con el objetivo principal de servir como un robot de bajo costo para aplicaciones de enjambre inteligente en la industria, principalmente pero no



Figura 2: Kilobots [5].

de forma exclusiva para fábricas inteligentes. El WsBot posee atributos similares a los de máquinas industriales, como máquinas elevadoras (forklifts), para lograr comportamientos inteligentes ya sea con tareas individuales pequeñas para lograr una mayor o trabajar en conjunto con los demás agentes para completar un trabajo global. La estructura de este agente es la de un robot diferencial basado en un ROS (Sistema Robótico Operativo), capaz de comunicarse con los demás agentes a través de Wi-Fi [6].

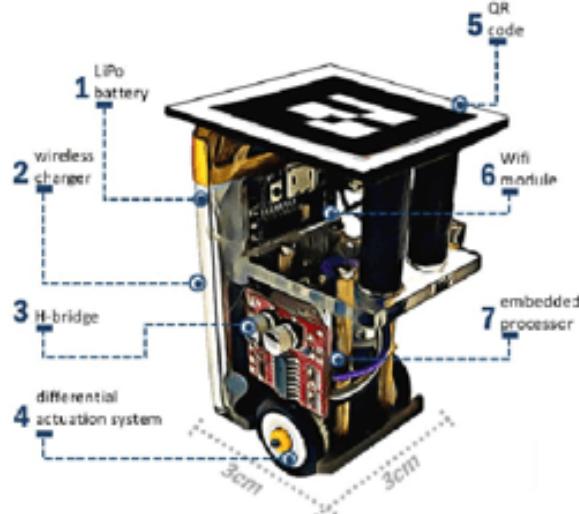


Figura 3: WsBot [6].

En la Figura 3 se puede apreciar con más detalle el Wsbot y en la Figura 4 se muestra este en un ambiente de aplicación. Con la aparición de esta plataforma robot se busca posicionar esta tecnología en un área de mayor acceso a la población y aplicaciones de uso más cotidiano.



Figura 4: WsBot en ambiente de trabajo [6].

2.4. Robotat

En la Universidad del Valle de Guatemala se realizó el megaproyecto de múltiples fases, Robotat, en las cuales tanto catedráticos como alumnos colaboraron en su desarrollo. El Robotat es un ecosistema de experimentación robótico, diseñado para que este sirva como un campo de pruebas para que los estudiantes puedan estudiar a los robots con herramientas avanzadas de las cuales destacan el sistema de captura de movimiento OptiTrack y la red de comunicación Wi-Fi.

En la última tesis, de Camilo Perafán Montoya en 2021 [7], se completó la red de comunicación para múltiples agentes, expansión que permite la implementación de nuevas tecnologías que en las iteraciones previas no se había logrado, entre las cuales está principalmente la robótica de enjambre. Además se desarrolló una librería en el lenguaje de programación C para que un microcontrolador ESP32 se pudiera conectar a la red implementada, y recibir datos del OptiTrack por medio del protocolo MQTT, así como enviar datos por el mismo hacia un servidor. Es importante recalcar que una de las conclusiones principales de este proyecto fue que el número máximo de agentes que se pueden conectar en simultáneo a la red para que su decodificación de datos se mantenga eficiente es de 11 agentes. Para que agentes no robóticos puedan interactuar con el sistema se diseñó una antena inteligente.

2.5. Algoritmos de Robótica de Enjambre implementados en UVG

2.5.1. Particle Swarm Optimization (PSO)

Este algoritmo consiste en que un conjunto de partículas comiencen desde una posición aleatoria y arbitraria para luego reunirse en conjunto en un punto específico. En la UVG, desde 2019 se ha trabajado con este algoritmo desde una perspectiva modificada, es decir un Modified PSO (MPSO), para dejar de estudiar a los agentes como partículas solamente, sino tomar en cuenta sus características al tratarse de robots diferenciales.

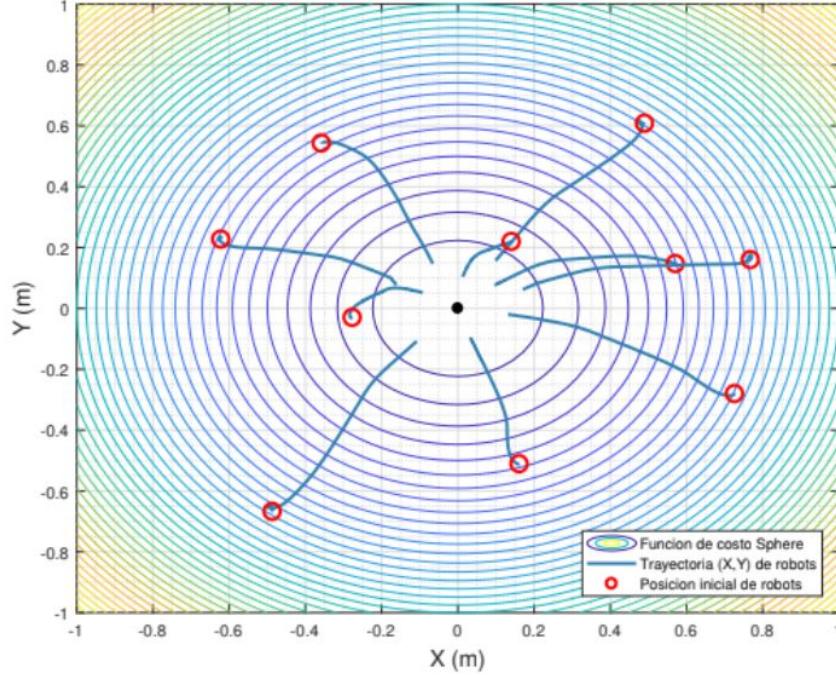


Figura 5: Trayectorias generadas de PSO para un enjambre de 10 agentes [8].

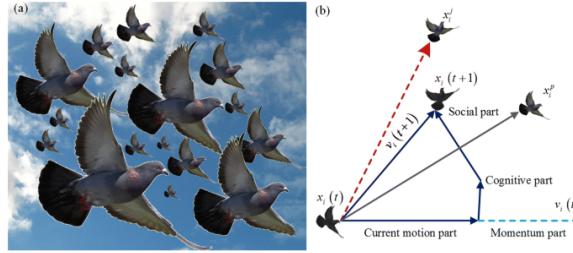


Figura 6: PSO en la naturaleza [9].

Primero se estudió en la tesis de Aldo Aguilar [8], donde se avanzó hasta simulaciones de Matlab y Webots. Luego, en años siguientes se continuó con el desarrollo de estos algoritmos para refinarlos hasta llegar a intentos de implementación en físico, como es el caso del trabajo de Alex Maas [10] usando una plataforma Raspberry Pi para obtener las poses en tiempo real y la comunicación entre robots por medio de UDP, aunque cabe resaltar que existieron limitaciones relacionadas a la falta de una plataforma móvil por lo que la validación se tuvo que realizar de forma que se movieran los marcadores manualmente, haciendo varias capturas para emular el movimiento que hubiera presentado un robot móvil, la verificación de las poses/posiciones se hizo mediante un algoritmo de visión de computadora.

Los últimos avances de la implementación de este algoritmo se dieron en 2022, con la tesis de Rubén Lima [11], al comenzar con la validación del algoritmo en físico usando el equipo de captura de movimiento del Robotat, continuando con el diseño de una plataforma móvil llamada ByteBot3D compuesta por una Raspberry Pi. Se incursionó con la comunicación TCP/IP. Debido a que no se usaron encoders el rendimiento del comportamiento mecánico no fue el esperado por lo que se optó por mover la plataforma manualmente.

2.5.2. Ant Colony Optimization

El algoritmo se basa en el comportamiento que tienen las colonias de hormigas para encontrar rutas hacia comida mediante feromonas, cada hormiga (agente) explora el terreno y si una encuentra comida (meta) va marcando el camino con lo que lo encontró con feromonas (probabilidad) y las demás hormigas siguen este camino eventualmente. Si por el contrario una hormiga no encuentra alimento en su exploración, su rastro se desvanece y termina siguiendo el rastro de feromonas de otra hormiga.

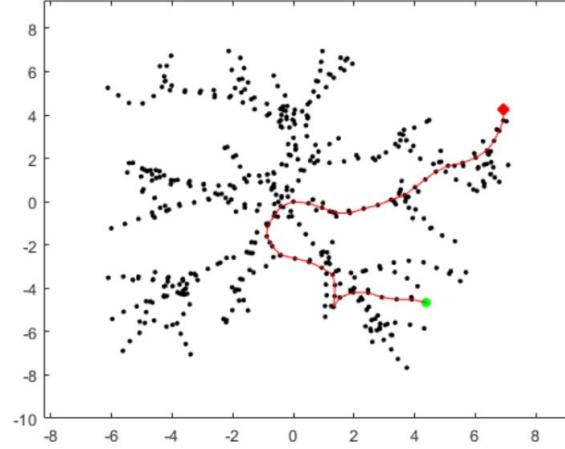


Figura 7: Diagrama de Feromona y camino encontrado [12].

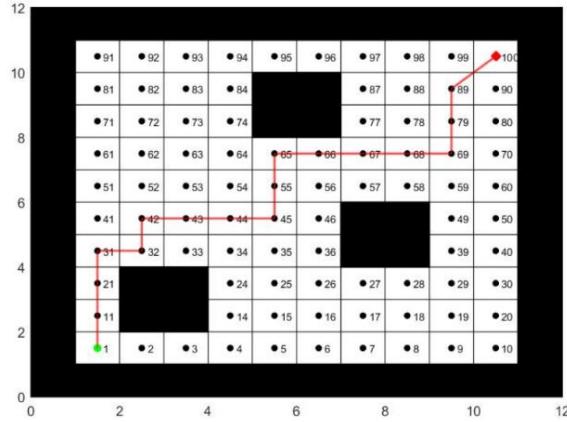


Figura 8: Trayectoria del camino encontrado con feromona [13].

En la tesis de Gabriela Iriarte [12] se desarrolló un algoritmo de *Ant Colony* en UVG por primera vez y en otra iteración Daniela Baldizón [13] realizó una versión modificada, *Modified Ant Colony Optimization (ACO)*, para adaptarlo a exploración de terrenos, planificación de trayectorias, reconocer y evadir obstáculos. Se implementó en Matlab a nivel de simulación con validación usando 3 mapas distintos para probar el algoritmo.

El trabajo realizado por Walter Sierra [14] continuó con el algoritmo de Ant System (AS), implementado previamente en la universidad solamente a nivel de simulación. En este caso, se expandió la implementación a una plataforma Raspberry Pi para probar su

funcionamiento en un ambiente físico. Debido a que no se contaba con una plataforma móvil se realizaron las pruebas de forma manual analizando mediante marcadores en una mesa de pruebas y visión de computadora.

2.5.3. Algoritmo de sincronización y control de sistemas de robots multi-agente para misiones de búsqueda

Este algoritmo fue desarrollado e implementado por Andrea Maybell Peña [15], centrado en manejar el sistema de robots multi-agente para misiones de búsqueda, apoyándose de teoría de grafos y control moderno para mantener a los agentes en formaciones específicas, generando una especie de cuerpo con los robots diferenciales como sus partes, con la capacidad de “fluir” a través de obstáculos. Se crearon subalgoritmos para mantener la formación y un control para la evasión de obstáculos.

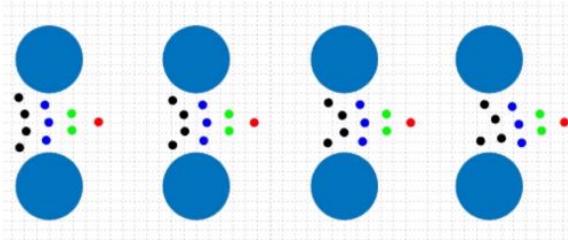


Figura 9: Movimiento en formación mínimamente rígida a través de obstáculos [16].

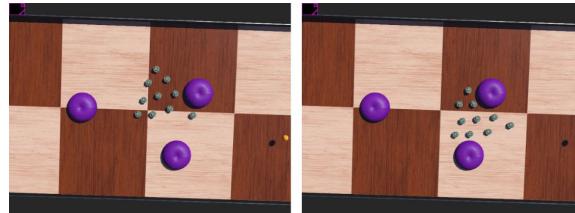


Figura 10: Simulación en Webots de la formación moviéndose a través de obstáculos [16].

Se avanzó hasta simulaciones en Matlab modelando al agente como partícula, para luego realizar simulaciones en Webots, adaptando al agente a un modelo de un robot diferencial. La plataforma robótica elegida en ese momento para la tarea fue un BitBot, mas nunca se llegó a probar en dicho robot.

En 2022, se volvió a estudiar este algoritmo de formaciones con un enfoque 3D, en la tesis de Kenneth Aldana [17]. Se implementó el algoritmo en simulaciones de Matlab y Webots solamente, no se llegó a implementar en físico tampoco.

Desarrollo previo en simulaciones para el algoritmo de interés

El trabajo actual consiste en continuar con la siguiente fase de implementación del algoritmo de sincronización y control de formaciones, por lo que es útil revisar los resultados principales de la fase anterior. La fase anterior fue realizada por Maybell Peña, y su trabajo consistió en el desarrollo del algoritmo, con sus respectivas simulación en Matlab y Webots.

A continuación se muestran los resultados de las simulaciones, con distintas configuraciones de obstáculos:

Configuración de Obstáculos	Rigidez	Goal	No Goal
1	Total	58	42
1	Minimal	86	0
2	Total	50	40
2	Minimal	78	10

Cuadro 1: Resultados de éxito de simulación en la fase previa [16].

Se puede observar en el Cuadro 1, la cantidad de simulaciones con éxito y fallidas para llegar a la meta en distintas configuraciones de obstáculos. Enfocándose en la comparación de usar un grafo rígido contra uno mínimamente rígido, el mínimamente rígido tuvo una visible mayor eficiencia para llegar a la meta.

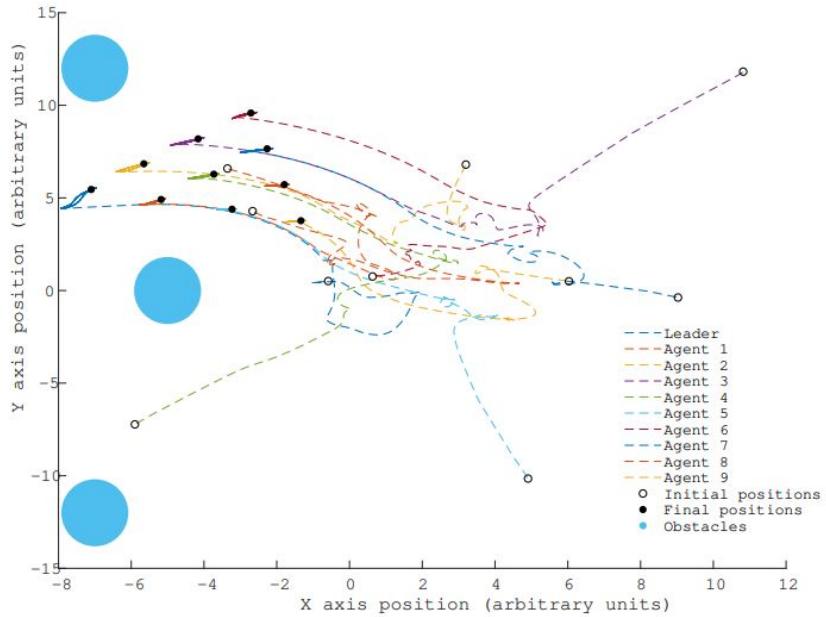


Figura 11: Control de formaciones usando un grafo rígido en Matlab [16].

En la Figura 11 se puede observar las trayectorias ejecutadas en la simulación con 10 agentes, uno de ellos siendo el líder, usando un grafo rígido.

En la Figura 12 se puede observar tres reconfiguraciones de formación para poder pasar a través de los obstáculos y mantener la formación, debido a que esta es un grafo mínimamente rígido.

En la Figura 13 se puede observar la simulación completa (el orden es de izquierda a derecha y de arriba hacia abajo) de Webots con los robots diferenciales ejecutando la formación, luego las trayectorias evadiendo obstáculos hasta llegar a la meta. Se puede observar con claridad la efectividad que presentan los grafos mínimamente rígidos debido a su capacidad de deformarse para atravesar los obstáculos y llegar a la meta.

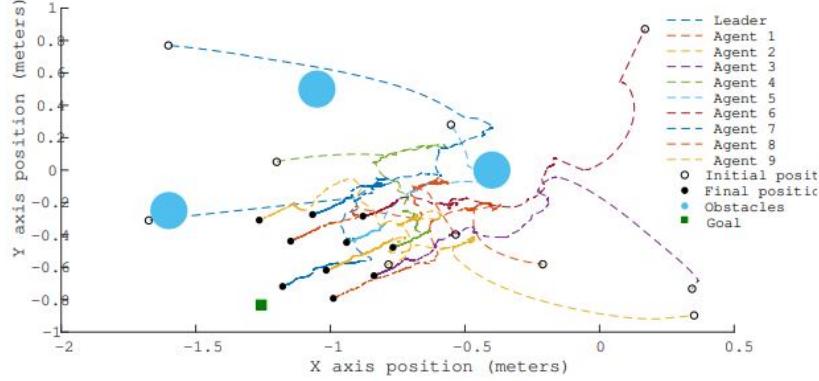


Figura 12: Formación inicial experimentando deformación para estar en las posiciones adecuadas [16].

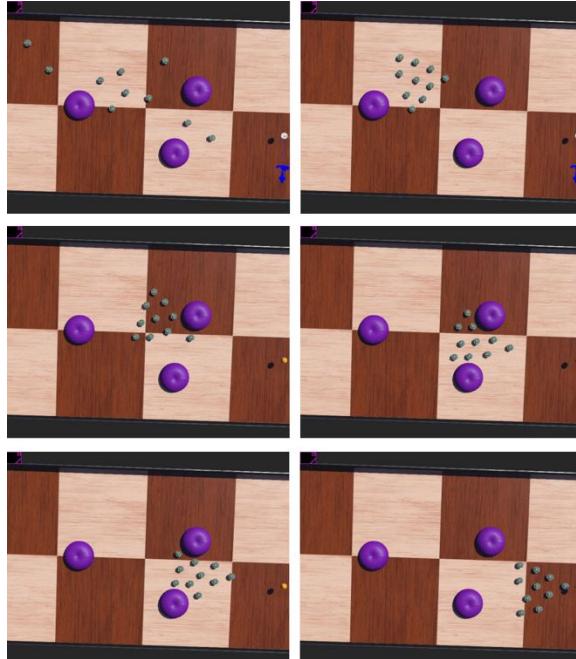


Figura 13: Simulación en Webots de formación y dirección hacia meta [16].

2.6. Plataformas Robóticas Móviles en UVG

A lo largo de los últimos años se ha buscado desarrollar plataformas móviles adecuadas para la robótica de enjambre en el ambiente Robotat de la UVG. Entre los primeros prospectos se mencionaron los Bitbots [15], y también se mostró interés en los Epucks. En los años más recientes se han desarrollado varios robots diferenciales como lo son:

- ByteBot: Implementado por Julio Rodríguez [18], consiste de una Raspberry Pi Zero con Servomotor, sensores de proximidad y cámara; cuenta con comunicación inalámbrica TCP IP y UDP.
- Alphabot: Es un Alphabot con modificaciones que usa una Raspberry Pi, se imple-

mentó en el Robotat con sistema OptiTrack. Rediseño por Luis Nij [19].

- ByteBot3D: Es una continuación del trabajo realizado en el ByteBot, modificado por Rubén Lima [11], utiliza el mismo controlador y comunicación que la versión anterior, usa el sistema de captura de movimiento del Robotat.

CAPÍTULO 3

Justificación

La inteligencia de enjambre es un campo relativamente nuevo en el área de inteligencia artificial, con potencial en diversas aplicaciones que van desde el contexto de la vida cotidiana hasta un ambiente industrial. La mayor ventaja que presenta es que se compone de múltiples agentes para llevar a cabo tareas complicadas y se forma de unidades simples de bajo costo. Con este atributo, los sistemas robóticos basados en la inteligencia de enjambre no son vulnerables en caso de que uno de los agentes falle o se pierda, lo que les da robustez. Esta misma ventaja resulta útil en misiones de búsqueda y rescate, así como la movilización de distintos tipos de agentes (entre ellos terrestres, acuáticos y aéreos) según la aplicación lo requiera.

Se ha elegido estudiar la robótica de enjambre debido a que esta resulta ideal para poner en práctica los conocimientos adquiridos a lo largo de la carrera de Ingeniería Mecatrónica, principalmente los conceptos avanzados sobre robótica y control adquiridos en los últimos años de estudio.

Desde 2019 en la Universidad del Valle de Guatemala se comenzó con el estudio y mejora de algoritmos de inteligencia de enjambre, de entre los cuales se destaca el algoritmo de sincronización y control de formaciones en sistemas robóticos multiagente, cuya alta coordinación y flexibilidad representa una alta eficiencia al momento recorrer áreas estratégicamente. Esto se puede aplicar a la búsqueda de sobrevivientes para catástrofes naturales, dada la versatilidad del algoritmo de control de los robots.

Habiendo mencionado los avances previos en la UVG, cabe resaltar que todos los avances que se han hecho han sido a nivel de simulación en software como Matlab y Webots, por lo que realmente no se ha tenido una validación formal física de la mayoría de los algoritmos en robots móviles funcionales, entre ellos el algoritmo de sincronización y control de formaciones desarrollado por Maybell Peña [15]. Esta falta de validación se ha debido a que no había infraestructura adecuada, pero ahora sí existe y el algoritmo está listo para realizar pruebas en el Robotat, con el sistema OptiTrack y los robots diferenciales con ESP32. Con la oportunidad que estos recursos presentan, se puede validar el algoritmo en un ambiente físico, lo que sería un gran paso para la investigación de la inteligencia de enjambre. Final-

mente, esto dejaría un precedente con buena base para futuro estudio del tema, haciendo cada vez más factible su uso en el mundo moderno.

CAPÍTULO 4

Objetivos

Objetivo General

Validar el algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente previamente desarrollado a nivel de simulación, en sistemas físicos, y evaluar su funcionamiento en el ecosistema Robotat.

Objetivos Específicos

- Adaptar el software desarrollado anteriormente para su aplicación en la mesa de pruebas del Robotat y los robots diferenciales Pololu 3Pi+.
- Evaluar la generación de trayectorias usando marcadores en el Robotat y el sistema de captura de movimiento OptiTrack.
- Examinar el comportamiento de los robots diferenciales al ejecutar las trayectorias generadas por el algoritmo de sincronización y control, en distintos escenarios.
- Comparar los resultados obtenidos en simulaciones con los obtenidos con los agentes físicos en ambientes controlados.

CAPÍTULO 5

Alcance

Con esta validación, se produjo un algoritmo funcional para la ejecución de trayectorias para agentes, ya sea en un contexto de simulación o en el contexto de un ambiente real y controlado como lo es el Robotat. Se optimizó el algoritmo previamente desarrollado en 2019, otorgando una mayor flexibilidad de número de agentes y obstáculos, con una comunicación mejorada y una sincronización aumentada para evitar la pérdida de datos y coordinación.

Cabe resaltar que el algoritmo generado como producto de la migración es dinámico respecto a los agentes, pues constantemente se está actualizando su posición con los marcadores del OptiTrack, proporcionando una mejor adaptación a cambios en el entorno.

Se añadió un sistema de configuración de escenarios dentro de la inicialización del algoritmo para facilitar el estudio y generación de datos del mismo, así como un aumento de su reproducibilidad al permitir guardar las condiciones iniciales de cada prueba en físico. Con este sistema, la comparación entre las pruebas físicas y simuladas se vuelve más precisa, ya que existen menos diferencias innecesarias entre las pruebas.

También se generaron funciones de comunicación del Robotat y el Pololu 3Pi+ en Python, que antes solo existían en Matlab, lo que representa un progreso en términos de eficiencia computacional, así como el añadido de estar hecho en un lenguaje *open-source*. El hecho de ser *open-source* representa una mayor accesibilidad para futuros investigadores del tema.

Otro de los productos resultantes del proyecto fue un código de procesamiento de datos para la generación de las gráficas de las trayectorias, facilitando su visualización, manipulación de los datos y análisis de los mismos.

El proyecto llegó hasta pruebas en físico, con la limitación de número de agentes robóticos siendo 5, debido a la disponibilidad de robots en la UVG. La escasez de robots se debió tanto a que el número máximo de Pololus 3Pi+ habilitados por la universidad fue de 10, de los cuales algunos se encontraban fuera de servicio, mientras que otros se tenían que compartir con los demás estudiantes para sus proyectos respectivos. Otra de las limitaciones fue la

disponibilidad del ecosistema Robotat, pues se tenían un tiempo máximo de pruebas por semana de alrededor de 6 horas, de las cuales, también se tenía que compartir la mesa con otros estudiantes. Esto resultaba en un tiempo disminuido para hacer pruebas exclusivas del algoritmo, por lo que no se pudo hacer demasiadas pruebas de repetición en cuanto a escenarios.

En síntesis, se llegó a adaptar el algoritmo para su ejecución para pruebas físicas en el Robotat, generando trayectorias de forma dinámica a partir de las mediciones obtenidas del OptiTrack, y comparar los resultados de escenarios de interés.

CAPÍTULO 6

Marco teórico

Es importante entender el contexto del algoritmo de sincronización y control de formaciones multi-agente, para lograr aplicarlo correctamente a un entorno real. Es por esto que en las siguientes secciones se desarrollará un poco más sobre los conceptos claves detrás del algoritmo que se compone de teoría de grafos principalmente, funciones y herramientas matemáticas de interés, sistemas de control, funcionamiento, resultados previos de simulación, el software y hardware relevante, la infraestructura disponible, el sistema de captura de movimiento, etc.

6.1. Definiciones importantes de robótica de enjambre

6.1.1. Agente

Un agente es un individuo simple que forma parte del enjambre, en este caso un robot de dimensiones relativamente pequeñas que posee la capacidad de realizar acciones simples para cumplir un objetivo más complejo de forma cooperativa; recibir y transmitir información, así como instrucciones [3].

6.1.2. Formaciones

En los enjambres de la naturaleza comúnmente se puede notar la emergencia de patrones entre los agentes, creando formaciones y comportamientos aparentemente coordinados, ya sea a propósito o no. Esto no es diferente en la robótica de enjambre, pues también se pueden realizar formaciones simples o complejas según la aplicación lo requiera. Se pueden crear movimiento coordinados mediante la generación de trayectorias con evasión de obstáculos y control de velocidad, con restricciones de distancia entre agentes, se elaborará sobre esto más adelante. Existen dos tipos principales para la coordinación de las formaciones, los cuales son control centralizado y descentralizado [3].

6.1.3. Control centralizado

Este enfoque no está construido alrededor de comunicación entre agentes por medio de sensores, sino que alrededor de un sistema de transmisión de datos a un mismo receptor (un CPU que se encarga de coordinación). Esto se debe a que agentes robóticos por lo general solo reportan su posición a un sistema de cómputo principal y este le devuelve instrucciones al robot, sin necesidad de que este se tenga que comunicar con los demás agentes de forma directa. La formación de patrones descentralizada por lo general es más costosa debido a la necesidad de un CPU de control y procesamiento, también es menos escalable y con menor robustez a fallas [20]. Cabe resaltar que la adaptación a nivel grupal solo se puede obtener mediante un control centralizado o permitiendo que los agentes comuniquen sus información del exterior entre sí [3].

6.1.4. Control descentralizado

Este tipo de control consiste en que los agentes realicen mediciones individuales de su entorno y se comuniquen entre sí, sin necesidad de un CPU externo que les otorgue instrucciones. Esto implica que el programa y procesamiento debe ocurrir completamente en los agentes, según la información que obtengan de sus agentes cercanos. Este enfoque, al no poseer una unidad central de procesamiento, resulta ser más barato y menos propenso a fallas o pérdidas, pues cada uno de los robots tiene el mismo peso de información para la formación [3].

6.2. Teoría de grafos

Es una rama de la matemática que se centra en el estudio de los grafos, los cuales son una especie de mapas de ruta, que se dibujan con puntos y líneas. Por lo general los grafos son herramientas útiles en el modelado de problemas, para representar las relaciones de los componentes importantes y sirven para resolver problemas de búsqueda de caminos eficientes, entre otros. Esta teoría emplea estructuras matriciales para representar a los grafos y realizar operaciones con los mismos [21].

6.2.1. Conceptos básicos en teoría de grafos

Vértices: También llamados nodos, son los puntos que conforman al grafo. Cada vértice tiene una valencia asociada según la cantidad de aristas que confluye en él.

Aristas: También llamados arcos, son las líneas que conectan a los vértices, y tienen una longitud de la conexión asociada. Cuando dos aristas se cruzan se le llama cruce [21]. Se pueden clasificar en:

- **Adyacentes:** Estas convergen en un mismo vértice.
- **Paralelas:** Se caracterizan por compartir tanto el vértice inicial como el final.

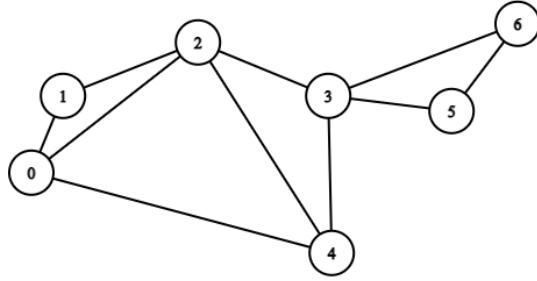


Figura 14: Ejemplo de un grafo.

- **Cíclicas:** Su vértice final es el mismo que el inicial.

Camino: Conjunto de vértices interconectados por aristas.

Tipos de grafo:

Existen diferentes tipos de grafos según sus características, configuración, utilidad y complejidad. A continuación se muestran las distintas clases de grafos principales clasificados según las cualidades de las aristas, vértices, su peso y formas de conexión [21] [22]:

- **Simple:** Definición estándar de un grafo, que indica que este solo acepta una arista para unir dos vértices.
- **Multigrafo:** Acepta más de una arista.
- **Dígrafo:** Poseen una orientación en las aristas, representada por una flecha (Ver Figura 15).
- **Etiquetado:** Los vértices tienen etiqueta y las aristas un peso.
- **Aleatorio:** Sus aristas están asociadas a una probabilidad.
- **Hipergrafo:** Las aristas son incidentes a 3 o más vértices.
- **Infinito:** El cardinal de los vértices y aristas es infinito.
- **Plano:** Este se puede representar sin ninguna intersección entre vértices y aristas.
- **Regular:** Todos sus vértices tienen el mismo nivel de valencia.

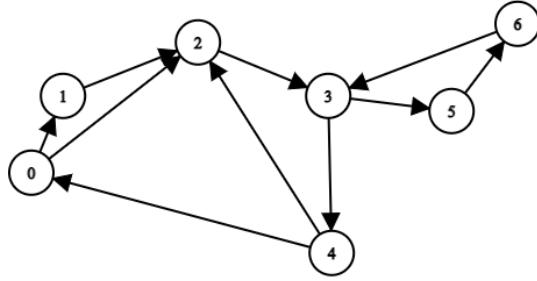


Figura 15: Ejemplo de dígrafo.

6.2.2. Matrices de interés

Entre las matrices que serán de principal utilidad para el algoritmo de sincronización y control de formaciones se puede mencionar las siguientes [21] [22]:

- **Matriz de Adyacencia:**

El grafo se representa con una matriz cuadrada A de tamaño n^2 , siendo n el número de vértices. Si existe una arista entre el vértice x y el y , el elemento $a_{x,y}$ es 1, si no, es 0.

- **Matriz de Incidencia:**

El grafo se representa por una matriz de $A \times V$, es decir aristas por vértices. La matriz según v,m brinda información sobre la arista, con 1 siendo conectado y 0 no conectado.

- **Matriz de grados:**

Matriz diagonal D que contiene información del grado de cada vértice, lo que indica cuantas aristas están conectadas al vértice. Combinando esta y la matriz de adyacencia se consigue una matriz laplaciana [23][24].

- **Matriz Laplaciana:**

Esta matriz L se obtiene al restar la matriz de adyacencia A a la matriz de grados D de un grafo. Es decir $L = D - A$ [25].

- **Matriz de Rígidez:**

Esta matriz sirve para representar grafos rígidos, y cuyo enfoque es relacionar los desplazamientos de un conjunto de vértices de una estructura, con las fuerzas exteriores que se necesitan para poder lograr un desplazamiento. Ya que sus distancias entre

vértices son constantes, el grafo se mueve en su totalidad como una estructura rígida [22].

Existe otra variación de este grafo, que forma parte de la familia de gráficos de Lamam, llamado grafo mínimamente rígido, o gráfico de Laman. Se describe como un grafo $G = (V, E)$ con n vértices $V = \{1, 2, \dots, n\}$, $m = |E|$ aristas que cumple con $m = 2n - 3$ y cada subconjunto con $k \geq 2$ vértices abarca hasta $2k - 3$ aristas [26].

Para construir un grafo mínimamente rígido es necesario usar la variación de la matriz de rigidez y ejecutar el algoritmo de inserción de Henneburg, que sirve para la generación de este tipo de grafo “flexible”, ya que cada vértice se queda con dos grados de libertad. El algoritmo consiste en los siguientes pasos [27]:

- Numerar todos los vértices.
- Agregar una arista entre el vértice 1 y el vértice 2.
- Los vértices restantes se van agregando en orden al componente conectado del grafo, conectando cada uno a la estructura de grafo con 2 aristas.
- Mantener el número de conexiones necesarias para n vértices menor a $(n^2 - n)/2$.

6.3. Teoría de control

Para lograr implementar un manejo de las formaciones de agentes robusto, y mantenerlos en las posiciones adecuadas de un grafo mínimamente rígido es necesario tomar en cuenta el elemento de control. Luego de haber construido un grafo mínimamente rígido, se debe tomar en cuenta la información devuelta por los agentes. Esto desemboca en la necesidad de un grafo etiquetado, es decir sus aristas tienen una ponderación o peso, según la dinámica de un lazo cerrado del sistema de control multi-agente. El hecho de que sea dinámico implica que la longitud de las aristas varía en el tiempo [15] [27].

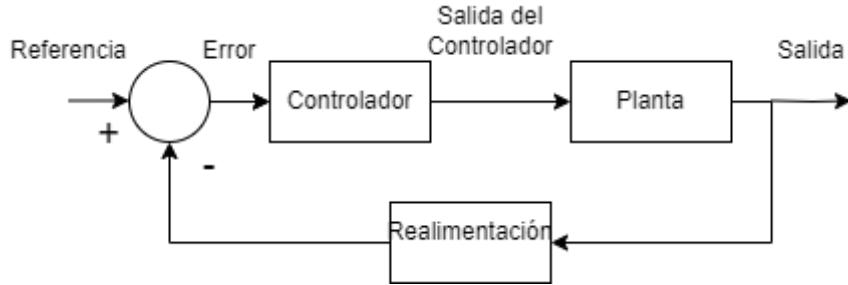


Figura 16: Lazo cerrado de control.

6.3.1. Control de formaciones

El control de formaciones se basa en dos niveles de control, uno superior y uno inferior. El superior se encarga del comportamiento de los robots como agentes, para mantener sus posiciones relativas entre así, así como la posición global de los mismos [15]. El control de capa inferior se encarga de controlar la velocidad de las ruedas para que los motores de las ruedas izquierda y derecha coincidan con la magnitud esperada.

6.3.2. Modelo del robot diferencial

Es importante adaptar el movimiento de formaciones de enjambre al contexto donde se implementará el algoritmo, pues en una simulación de partículas el movimiento es mucho más simple, pues carecen de dimensiones, volumen y masa. La adaptación corresponde en este caso al modelo de un robot diferencial, pues estos son los agentes con la tarea de ejecutar las trayectorias encontradas[15] [16].

Este modelo contempla las dimensiones físicas del robot, y las distancias l del motor hasta su centro, ϕ es el ángulo de orientación del uniciclo en el plano XY, v es la velocidad lineal, ω es la velocidad angular del robot y r es el radio de las ruedas del robot. El subíndice $ctrl$ indica control.

Se tienen las siguientes ecuaciones principales para analizar la cinemática del robot [28]:

$$v = \frac{r(\dot{\Phi}_R + \dot{\Phi}_L)}{2} \quad (1)$$

$$\omega = \frac{r(\dot{\Phi}_R - \dot{\Phi}_L)}{2l} \quad (2)$$

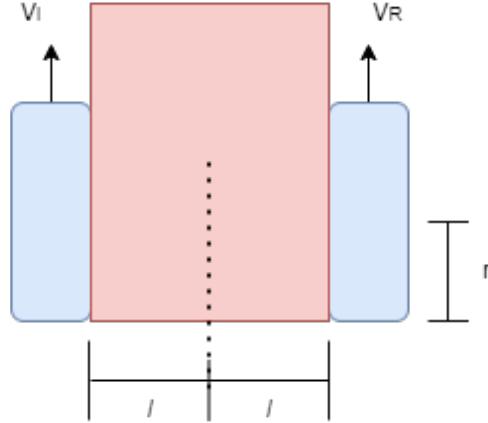


Figura 17: Modelo de un uniciclo en 2D.

De donde se puede llegar a las siguientes expresiones de la velocidad angular controlada de tanto la rueda izquierda como la derecha, como se muestra a continuación:

$$\dot{\phi}_{L,ctrl} = \frac{v_{ctrl} + l\omega_{ctrl}}{r} \quad (3)$$

$$\dot{\phi}_{R,ctrl} = \frac{v_{ctrl} - l\omega_{ctrl}}{r} \quad (4)$$

De forma resumida, para pasar de un modelo de uniciclo no holónomico, al robot móvil, se deben realizar tres pasos:

- Se prueba que el modelo del agente robótico pueda mapearse al uniciclo.
- Se aplica control al uniciclo para calcular las velocidades de control v_{ctrl} y ω_{ctrl} .
- Se realiza un mapeo de las velocidades de control hacia el robot real.

6.4. Herramientas matemáticas relevantes

Para lograr efectuar un control de múltiples capas con un funcionamiento correcto y adecuado para mantener la formación del enjambre de forma óptima se recurre a funciones, ecuaciones y otras herramientas matemáticas.

6.4.1. Ecuación de consenso

Para mantener la formación de agentes en los lugares asignados es necesario aplicar el concepto de la ecuación de consenso. Esta herramienta toma en cuenta el centro de masa de la formación y se obtiene la velocidad para cada agente individual, lo que induce a mantener la forma del grafo. Se describe como:

$$v_i = \sum_{j \in N(i)} (x_i - x_j) \quad (5)$$

Donde N es el número de vecinos j , es decir agentes en los vértices adyacentes/conectados a la unidad de interés i .

Al añadir pesos se obtiene la ecuación derivada de velocidad para la formación con tensiones de aristas entre agentes (vértices del grafo):

$$\frac{\partial e_{ij}}{\partial x_i} = \omega_{ij}(\|x_i - x_j\|)(x_i - x_j) \quad (6)$$

Con esta ecuación ya es posible despejar el peso para construir el control de formación tomando en cuenta tensiones, así como el mantenimiento de la conectividad [16] [29].

6.4.2. Función de tensión

Las funciones de tensión definen como se comportarán los agentes al intentar mantenerse en sus posiciones asignadas. Según la función de tensión así son las características de reunión de los agentes y cuanta holgura pueden tener estos entre su posición actual y la objetivo. Las principales funciones de tensión usadas para el control de formaciones en la ecuación de consenso son [15]:

- Evasión de colisiones.

$$\epsilon(x) = \frac{x^2}{x - r} \quad (7)$$

- Control de formación: Donde d es la distancia entre agentes.

$$\epsilon(x) = \frac{(x - d)^2}{2} \quad (8)$$

- Combinación de las previamente mencionadas con mantenimiento de conectividad.

$$\epsilon(x) = \frac{(x - d)^2}{(x - r)(x - R)} \quad (9)$$

- Otras combinaciones.
- Coseno Hiperbólico.

$$\epsilon(x) = \frac{\cosh(1.8x - 8.4)}{10} \quad (10)$$

6.5. Infraestructura en Robotat

El Robotat (Robot Habitat), como se explicó previamente, es un ecosistema de desarrollo de robots, consistente de varias herramientas útiles para experimentar con la robótica de enjambre, y a continuación se describirán las herramientas generales con las que cuenta el Robotat.

6.5.1. Mesa de pruebas

Es una plataforma plana de 3.8×4.8 m, con bordes alrededor de ella para servir como barreras delimitadoras para que los robots se mantengan dentro del ecosistema. Además cuenta con 6 cámaras de captura de movimiento alrededor de la plataforma que forman parte del sistema OptiTrack.

6.5.2. OptiTrack

Es un sistema de captura de movimiento con cámaras ultra precisas fabricado por la empresa OptiTrack, cuyas principales aplicaciones en el mundo son: producción virtual para películas, ciencias del movimiento, realidad virtual, robótica, animaciones, etc. A continuación se muestran los distintos dispositivos que lo componen y un ejemplo de montaje [30].

El modelo disponible en la universidad es el Prime^x 41 (Figura 19), cuyas características más importantes son:

- Rango de captura: 100 pies, 290 pies³/por cámara para marcadores pasivos y 1000000 pies³/por cámara para marcadores activos.

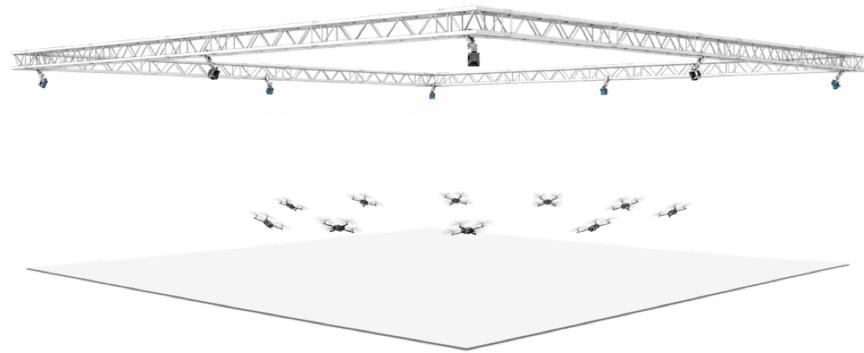


Figura 18: Ejemplo de sistema de captura de movimiento OptiTrack [30].



Figura 19: Cámara de Captura de Movimiento Prime^x 41 [30].

- Infrarrojos discretos.
- Captura de imágenes.
- Incertidumbre de \pm de 0.1 mm y errores rotacionales menores a 0.5 grados.
- Lentes de 12 mm.
- Retrocompatibilidad.
- Captura de Datos 2D/3D y cuerpos rígidos.
- Calibración y sincronización de cámaras.
- Precio: \$6499

6.5.3. Comunicación del Robotat

El Robotat tiene la capacidad de realizar mediciones por medio de las cámaras OptiTrack, transmitirlas usando un switch de por medio, a un servidor principal en una computadora

de laboratorio por medio del protocolo UDP. Este servidor de Python luego transmite los datos por medio de Wi-Fi usando un Router. Las computadoras que se encuentre en el rango del inalámbrico del Router pueden conectarse al servidor y extraer datos específico según se necesiten, como coordenadas, ángulos de Euler, pose, etc.

Por último, el Robotat cuenta con plataformas robóticas diferenciales, los Pololus 3Pi+, y se les puede enviar comandos a estos por medio de Wi-Fi también.

6.6. Software

Para lograr manejar y controlar los agentes de robótica de enjambre es necesario involucrar varios tipos de software, entre ellos lenguajes de programación especializados y simuladores de precisión con enfoque en entornos físicos. A continuación se menciona el software principal.

6.6.1. Matlab

Matlab es una plataforma de programación y cómputo numérico especializada en temas de ingeniería y ciencias, para analizar datos, desarrollar algoritmos y crear modelos; desarrollada por la compañía MathWorks [31].

Para este trabajo de validación del algoritmo de sincronización y control de formaciones, se utiliza como el centro de procesamiento de datos y generación de trayectorias, lo que le da un enfoque centralizado a este caso de robótica de enjambre.

6.6.2. Webots

Es una plataforma de código abierto creada por la compañía Cyberbotics[32] centrada en la simulación de robots, provee una ambiente completo de desarrollo para programar, simular y modelar distintos tipos de robots. Cuenta con un GUI moderno, un *physics engine*, un *rendering engine* y un editor de texto. La plataforma permite crear y añadir distintos tipos de objetos dentro de un mundo .wbt, modificar los parámetros de varios componentes, usar modelos ya preexistentes, sensores, implementar controladores (en lenguajes C, C++, Python, Java, Matlab, ROS, o un API).

En esta plataforma se realizan las simulaciones con el modelo adaptado a un uniciclo para los robots diferenciales, en lugar de solo partículas. Los códigos desarrollados en la fase previa trabajada por Maybell Peña [15] están programados en Python para los controladores de Webots.

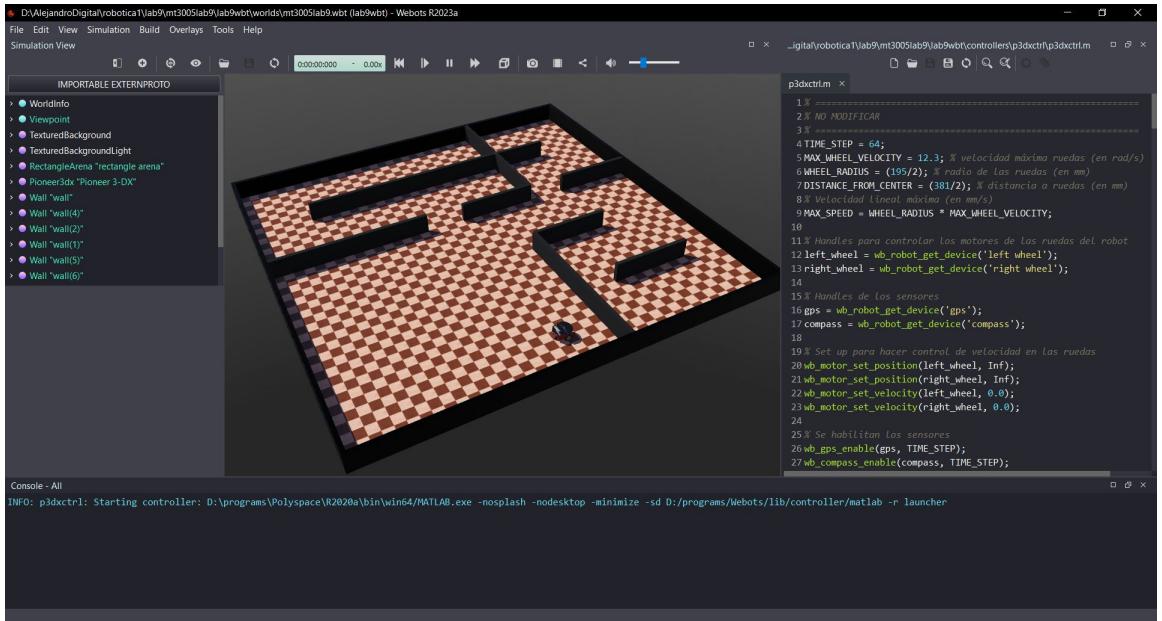


Figura 20: Entorno de desarrollo en Webots.

6.7. Hardware

Como último tema a profundizar, se aborda el hardware que se utilizará con el agente para ejecutar las trayectorias generadas por el algoritmo de sincronización y control de formaciones. El robot a utilizar como agente es un robot diferencial Pololu 3Pi+.

6.7.1. Plataforma Móvil: Robot Diferencial Pololu 3Pi+ modificado

La plataforma móvil elegida para realizar la validación del algoritmo es un Pololu 3Pi+ modificado, pues el original tiene una capacidad de procesamiento menor, ya que utiliza como cerebro del robot a un Arduino (ATmega32U4 MCU). Ya que el robot necesita una mayor capacidad de procesamiento para ejecutar las trayectorias, se decidió incluir un microcontrolador ESP32, para realizar el control de capa superior para la ejecución de trayectorias, mientras que el Arduino se encarga de realizar el control de capa baja, es decir se enfoca en controlar la velocidad de los motores.

Las características principales del Pololu 3Pi+ son [33]:

- Modelo: modelo 3pi+ 32U4 OLED Robot
- Encoders de cuadratura dual para control de lazo cerrado de posición o velocidad
- Sensores de línea
- Sensores de choque frontales
- IMU: Acelerómetro de 3 ejes, magnetómetro, giroscopio



Figura 21: Pololu 3Pi+ [33].

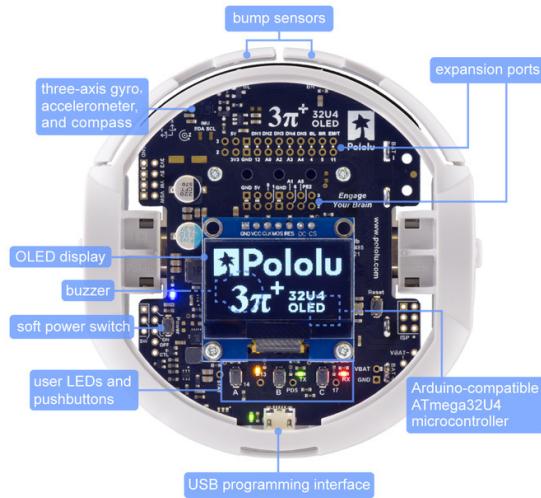


Figura 22: Especificaciones de sensores del Pololu 3Pi+ [33].

- Motores reductores micrometálicos de 30:1 MP de 6V
- Precio: \$159.95
- Alimentación: 4 baterías AAA
- cable USB A a Micro-B cable para programarlos y debuguearlos
- Velocidad máxima: 1.5 m/s
- Peso: 100 gramos (sin ESP32)
- Dimensiones: 97×96×36 mm
- Diámetro de las ruedas: 32 mm

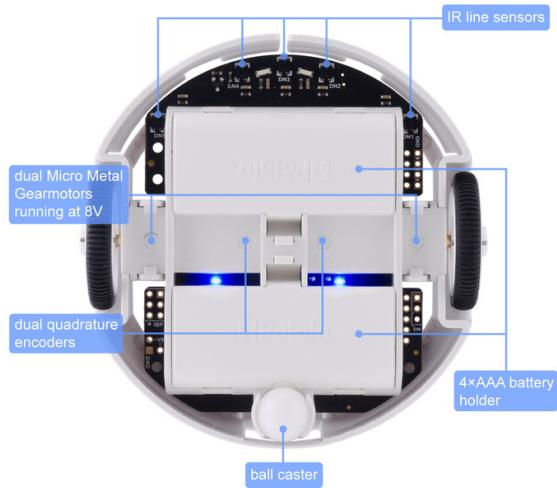
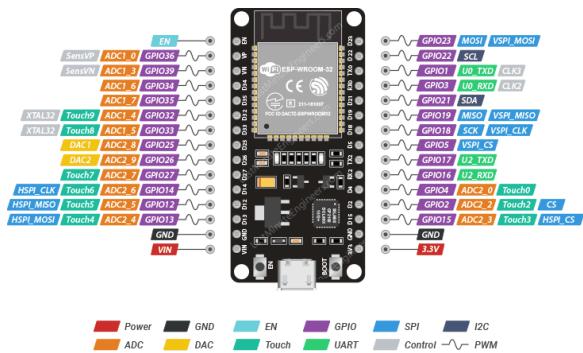


Figura 23: Especificaciones del Pololu 3Pi+ [33].

6.7.2. Microcontrolador del Agente: ESP32

El microcontrolador ESP32 fue el elegido para adaptarse al Pololu 3Pi+ debido a sus características ventajosas de procesamiento y conectividad. El ESP32 posee un módulo Bluetooth y Wi-Fi integrado, por lo que esto facilita la comunicación a distancia con el control centralizado de los agentes, para recibir la trayectoria a ejecutar. Además cuenta con las siguientes características principales [34]:

- Procesador: microprocesador de 32-bit Xtensa LX6 de doble núcleo (o de un solo núcleo), operando a 160 o 240 MHz y rindiendo hasta 600 DMIPS. Co-procesador de ultra baja energía (ULP).
- Memoria: 540 KiB SRAM
- Conectividad inalámbrica por Wi-Fi y Bluetooth
- Periféricos:
 - 12 bit ADC de hasta 18 canales
 - 2 DACs de 8 bits
 - 4 SPI
 - 2 I2C
 - 2 I2S
 - 3 UART
 - PWM
 - LED PWM
 - Cifrado flash



ESP32 Dev. Board / Pinout

Last Minute
ENGINEERS.com

Figura 24: ESP32 Pinout [35].

CAPÍTULO 7

Algoritmo de sincronización y control de formaciones

El algoritmo de sincronización y control de formaciones está conformado por varios subalgoritmos, que realizan tareas ya sea de forma individual o en conjunto para lograr su objetivo. Debido a esto, en el presente capítulo se explica la anatomía de sus partes, funcionamiento, parámetros y lógica, para poder entrar en contexto y familiarizarse con el mismo. Asimismo, se elabora en el proceso de restauración por el que tuvo que pasar el algoritmo y modificaciones que se tuvieron que hacer en el mismo para su funcionamiento en 2023.

7.1. Funcionamiento

El algoritmo como tal se divide en dos programas distintos, uno que se coordina el control del enjambre y el segundo que es el programa de recepción de datos de cada agente para su control individual. A continuación se explica como funciona cada uno y su relación.

7.1.1. Algoritmo de sincronización y control centralizado (Supervisor)

Por facilidad se le llamará también Supervisor, es la parte del algoritmo que realiza el trabajo de coordinación, procesamiento de datos y generación de valores para crear trayectorias. Se puede subdividir en cuatro principales segmentos para explicar su funcionamiento. El primer segmento tiene como trabajo la adquisición de datos del entorno, esto incluye las coordenadas en X y Y de los obstáculos, los agentes y las posiciones de interés, como el objetivo. El segundo segmento es la forma en que se calculan las velocidades de los agentes hacia los demás agentes y/o posiciones de interés, según la etapa de condiciones en las que se encuentre. Este cálculo se realiza mediante la ecuación de consenso, con un factor de peso

aplicado a ω , cuyo valor es afectado por las funciones de tensión y evasión de obstáculos.

El tercer segmento consiste en la evasión de obstáculos y colisiones, que toma en cuenta la posición actual de cada uno de los agentes y la compara con las posiciones de los agentes vecinos y obstáculos. Con esta información, calcula el peso ω a aplicar a la ecuación de consenso según que tan cerca está, por lo que la velocidad disminuye al acercarse a un objeto con posible colisión. El cuarto segmento se trata del movimiento por medio de un control proporcional, para movilizar a los agentes a un punto de interés. En la mayoría de escenarios, se utiliza para el movimiento del líder hacia un objetivo específico. El control proporcional en este caso es:

$$v_{n+1} = v + u \quad (11)$$

Donde u es:

$$u = K \cdot (x_{\text{objetivo}} - x_{\text{agente}}) \quad (12)$$

Donde K es la constante proporcional de ganancia que se multiplica con la diferencia de distancia entre el objetivo y el agente seleccionado.

El parámetro principal que funciona como variable a controlar es la velocidad en x y y , con la señal de control siendo la respectiva posición actual del agente, con la cual se calcula la norma de velocidades de todos los agentes. En base a esto se toman las decisiones relevantes dentro del código para ejecutar acciones siguiendo los enfoques relevantes mencionados previamente.

7.1.2. Algoritmo de control de uniciclo para cada agente

Este programa se encarga de procesar las instrucciones de velocidad calculadas en el algoritmo de sincronización, para realizar el cálculo de las velocidades lineales y angulares que se necesitan para encontrar la velocidad que cada rueda tiene que aplicar para poder seguir la trayectoria, según el modelo del uniciclo (Figura 17). Se encarga del control individual que tiene cada agente para seguir las instrucciones del programa de sincronización.

7.2. Lógica

Como se mencionó previamente, el algoritmo tiene una parte de Supervisor y una parte de control individual del agente. El Supervisor calcula las velocidades en x y y necesarias para orientar al agente a una posición deseada, mientras que el control de agente individual es el mismo programa para cada robot diferencial, con el detalle de que cada agente solo “procesa” a su instrucción correspondiente según un argumento de identificación.

A continuación se muestran los diagramas de flujo del funcionamiento base del algoritmo para cada programa. En secciones más adelante se introducirán variantes del mismo:

Con respecto a la Figura 25, se puede destacar que NormV es la norma de velocidades entre todos los agentes involucrados, y resulta siendo una buena variable de control para



Figura 25: Diagrama de flujo para el Supervisor/Algoritmo de Sincronización.

realizar los cambios de formación y movimiento, debido a que entre más cerca estén esta norma disminuye. En este caso se muestra 0.3 como valor de ejemplo, para representar una norma de velocidad baja. También cabe resaltar que la distancia entre líder a objetivo de 0.05 m es arbitraria y se puede modificar según las especificaciones de la tarea a realizar.

Por último, se explica que al llegar al objetivo, se puede comenzar con otra tarea. Ejemplos de procesos a realizar podrían ser:

- Detener a todos los robots (enviar comando de velocidades = 0).
- Crear una nueva formación.
- Comenzar movimiento de líder hacia nuevo objetivo.
- Segmentar la formación en subgrupos.
- Otras aplicaciones.

En el diagrama de flujo no se despliega la evasión de obstáculos ya que permanece activa durante todo el algoritmo.

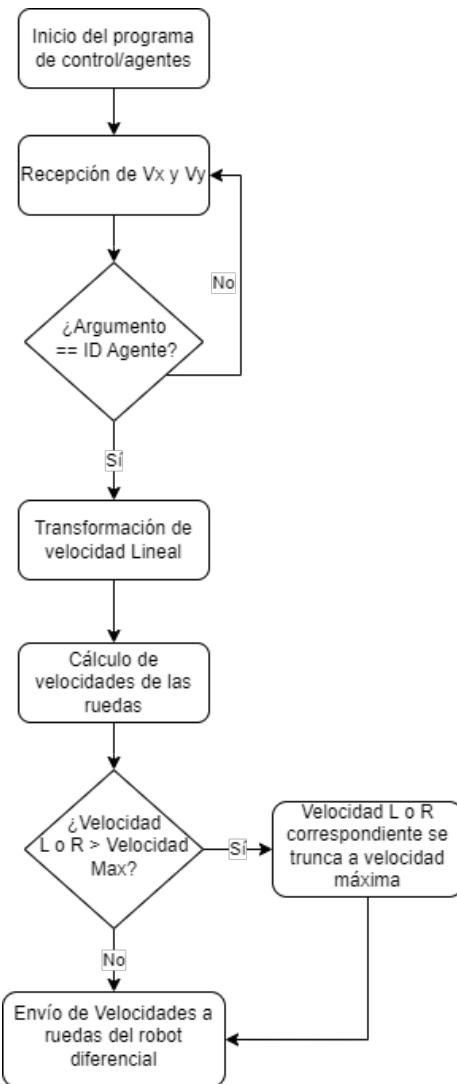


Figura 26: Diagrama de flujo para procesamiento de velocidades en agente individual.

En cuanto a la Figura 26, se muestra el programa de recepción de velocidades y procesamiento de las mismas para el agente individual. En Webots cada agente tiene su propio número de ID, con el cual se puede hacer una comparación de la instrucción de velocidad y el agente correspondiente. Se trabajó con un mismo programa generalizado que todos los agentes robóticos pueden usar y según el número de ID de cada uno, procesar únicamente los datos de interés. Además, se tiene una etapa de truncamiento de velocidades, para no superar las velocidades máximas que puede soportar el robot, ya sea para la rueda izquierda o derecha.

7.3. Restauración y actualización del algoritmo

La fase previa del algoritmo desarrollada por Andrea Maybell Peña [15], que llegó hasta pruebas en simulación en la versión de Webots 2019, trabajada cuatro años antes del desarrollo de este trabajo. Por lo que tanto el código como el mundo y simulación de Webots, quedaron obsoletos para versiones más recientes de Webots, como es el caso de la versión de Webots más actual en 2023, la versión 2023b. Esto significó que para llevar a cabo la siguiente fase de pruebas físicas, sería necesario una modernización de la fase previa, una recreación del mundo de Webots para la versión 2023b, limpieza de funciones abandonadas, así como un reajuste de parámetros.

Para la fase actual se optó por usar las versiones más recientes de software. Se eligió Python 3.10 para usar como lenguaje de programación destinado a los controladores de agentes robots y Supervisor de Webots, mientras que para Webots se seleccionó la versión 2023b.

7.3.1. Recreación del mundo de Webots

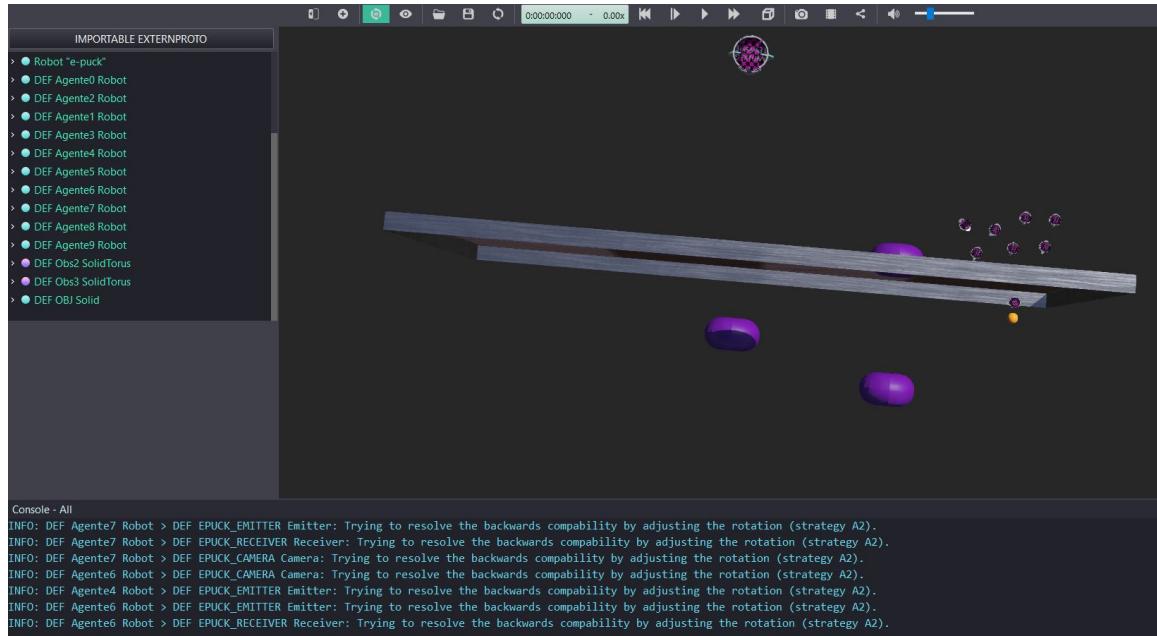


Figura 27: Estado del mundo de Webots creado en 2019 al abrirlo con Webots 2023b.

Como primer paso de la restauración, se buscó recrear el mundo previo del último modelo elaborado en 2019, correspondiente a PruebasObstaculos.wbt. Este mundo, debido a las diversas actualizaciones a lo largo de los años, presentaba los objetos en desorden, desactualizados, con varias dependencias perdidas y con los ejes rotados, perdiendo su utilidad en 2023. En la Figura 27 se puede apreciar el estado en el que se encontraba dicho mundo al abrirlo con Webots 2023b.

Como primera instancia, se intentó hacer la migración automática al enlazar los objetos con un *script* proporcionado en el repositorio de GitHub de Webots, mas la antigüedad del mundo no permitió que varios de los objetos se actualizaran correctamente. Por esto, se decidió empezar un nuevo mundo en 2023b desde cero, buscando replicar los obstáculos, objetivo, arena, agentes, supervisor y características del mundo previo. En la Figura 28 se muestra la recreación del mundo en Webots 2023b.

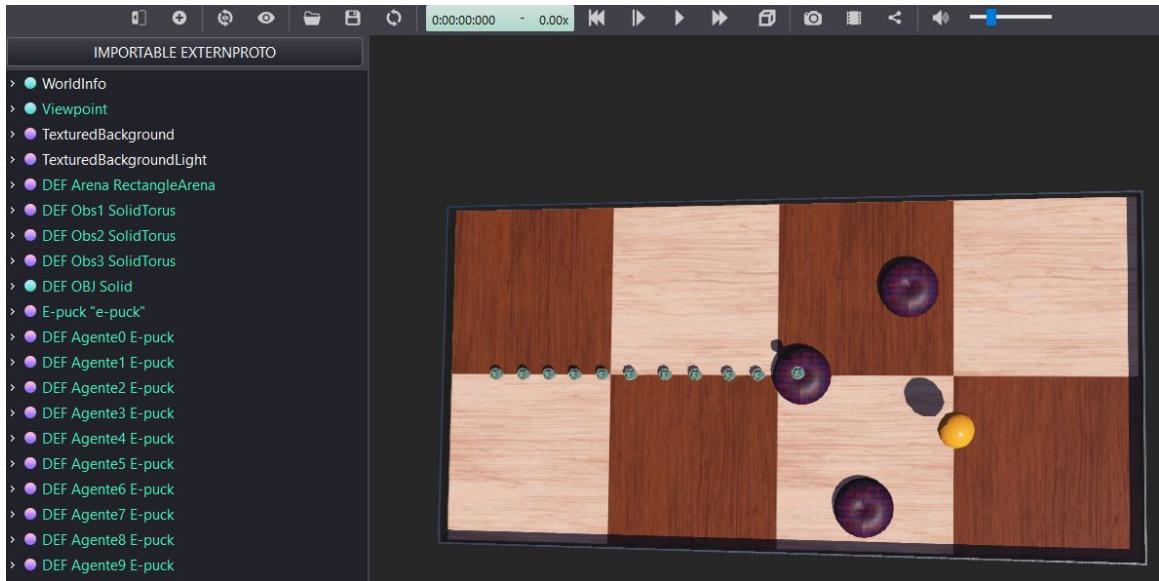


Figura 28: Recreación del mundo de Webots en versión 2023b.

Los objetos presentes en la Figura 28 son:

- Obstáculos: Toroides morados.
- E-pucks (Robots diferenciales): Pequeños círculos de color aqua.
- Objetivo final: Esfera amarilla.

Cabe resaltar que uno de los e-pucks se encuentra sobre uno de los obstáculos (obstáculo central). Este e-puck no es un agente, sino que representa al Supervisor, ya que Webots requiere de tener un robot que ejecute dicho programa.

7.3.2. Actualización del código de 2019 para su funcionamiento en 2023

Una vez recreado el mundo se prosiguió a probar los códigos de Supervisor creados en 2019, entre ellos Supervisor3.py, SupervisorObstaculos3.py, ErroresSupervisorObstaculos3.py, Errores2SupervisorObstaculos3.py, con sus respectivos archivos de funciones: funVel.py y funciones.py. También se realizaron pruebas con el archivo de controlador de agente individual: pruebaMatrizDifeomorfismo.py.

Comunicación entre programas

Al correr los programas, los robots permanecían inmóviles. Entre las primeras modificaciones que se tuvo que hacer, fue actualizar las rutas donde se escribían los archivos .pickle con las instrucciones binarias desde el Supervisor, para que el controlador del agente lograra

leer los datos, pues las rutas estaban dispuestas de forma no relativa, y eran inexistentes en computadoras distintas a la usada en el trabajo de 2019 [15].

Una vez restablecida la comunicación al actualizar las rutas, se encontró una forma más eficiente de comunicar las velocidades entre programas. Esta diferencia resulta por el uso de archivos .pickle, que implica una escritura de parte del Supervisor y una lectura por parte del controlador del agente. Estas operaciones no están sincronizadas y provocan pérdidas de datos por corrupción de archivos, desembocando en una comunicación poco robusta. La corrupción de archivos por lo general ocurre cuando un programa intenta leer un archivo que está en medio de una operación de escritura. En la tesis de Peña se menciona entre las recomendaciones que se buscara una mejor comunicación entre programas, pues esta presentaba errores de sincronización. Teniendo esta información, se procedió a buscar un método alternativo para la comunicación de datos entre archivos, seleccionando un método más formal de comunicación interprocesos (*IPC*).

Al momento de elegir el método *IPC* más adecuado para la comunicación, la forma más adecuada según los requerimientos de la tarea resultó ser el método de memoria compartida.

El método de memoria compartida consiste en crear un bloque de memoria con un tamaño específico (en bytes), para usarlo como medio de comunicación, donde el Supervisor escribe y el controlador del agente hace la lectura. Python cuenta con una librería especializada para estos procesos llamada *multiprocessing* con módulo *shared_memory*. La memoria compartida resulta conveniente para este proceso, debido a:

- No es necesario especificar ninguna ruta, lo que lo hace portátil de una computadora a otra con mayor facilidad.
- El espacio de memoria se puede acceder desde cualquier parte de la computadora, ya que se inicializa con un nombre único con el que se invoca a dicha área y sirve identificador universal desde cualquier programa de Python en la computadora.
- La rapidez de la comunicación es mayor al escribir a un buffer de espacio de memoria compartida, contrario a escribir y leer directamente al disco.
- Se evita el riesgo de corrupción de archivos.

Los pasos de implementación fueron:

- Supervisor:
 1. Creación del espacio de memoria, con n bytes necesario según la aplicación.
 2. Serialización en formato binario de los datos usando la librería *pickle*.
 3. Escribir los datos serializados al espacio de memoria compartido.
- Controlador de Agente:
 1. Acceso al espacio de memoria creado por el Supervisor.
 2. Lectura de los datos serializados.
 3. Deserialización de los datos usando la librería *pickle*.

Cabe resaltar que la creación del espacio de memoria debe ser creado por el Supervisor antes de que el controlador de agente intente acceder al mismo, ya que de lo contrario ocurre un error al intentar acceder a un espacio inexistente. Además, luego de terminar de usar el espacio de memoria compartida, este se debe cerrar apropiadamente con el método del objeto de memoria compartida “`.close()`”.

Sincronización entre programas

Una vez creada la comunicación usando los bloques de memoria compartida, se resolvió el problema de la corrupción de datos, pero el tema de sincronización de datos aún se tenía que solventar. Esto debido a que ambos lados de la comunicación tenían la posibilidad de acceder a los datos al mismo tiempo, existía la probabilidad de que el programa de control de agente hiciera la lectura mientras el Supervisor escribía. Esto daría como resultado un desfase de los datos y una inconsistencia en cuanto a los datos procesados para la formación, pues se podrían combinar las velocidades obtenidas en un ciclo presente con las del ciclo pasado.

La solución para sincronizar los datos y evitar inconsistencias en los datos fue el uso de primitivos de sincronización, específicamente los *locks*. Estos funcionan de manera que aseguran que solo un programa acceda al bloque de memoria a la vez, por lo que si el Supervisor está escribiendo, el controlador del agente no puede leerlo y viceversa.

La lógica de funcionamiento es:

1. Antes de escribir/leer los datos, se bloquea el espacio de memoria con `Lock().acquire()`.
2. Se escriben/leen los datos.
3. Se libera el espacio de memoria compartido con `Lock().release()`.

Cambios principales entre versiones de Webots

Para lograr una migración de versiones más eficaz, se compararon las entidades del mundo de Webots con las reconstruidas en la nueva versión, y se encontró que la diferencia principal radicaba en la disposición de los ejes. En 2019, XYZ correspondían a NUE (*North, Up, East*), mientras que en 2023, por defecto el mundo está configurado para la convención ENU (*East, North, Up*), como se puede observar en el Cuadro 2. Además se verificó en los *patch notes* de Webots el cambio en cuestión [32].

Año	+x	+y	+z
2019	Norte (N)	Arriba (U)	Este (E)
2023	Este (E)	Norte (N)	Arriba (U)

Cuadro 2: Comparación de disposición de ejes de mundo de Webots, según el año.

Con esta información, se actualizó el algoritmo en cuanto a la obtención de posiciones con las funciones de Webots. Claramente se puede observar que el desfase es de 1 posición

reducida, para todos los valores que involucraran las posiciones de Webots. Como las posiciones se leían por medio de vectores unidimensionales, el cambio fue universal, corriendo todos los índices de posición 1 unidad de aumento, haciendo un *wraparound* al llegar al valor 2, volviéndose este 0. A continuación se muestra una tabla con la lógica de reemplazo de valores:

Año	Posición		
2019	0	1	2
2023	1	2	0

Cuadro 3: Comparación de indexación de posiciones de mundo de Webots, según el año.

Para ilustrar de mejor manera la transición de los índices en el código a continuación se muestra un ejemplo.

Ejemplo de código en 2019:

```
posActuales[0][c] = posC.getSFVec3f()[0]
```

```
posActuales[1][c] = posC.getSFVec3f()[2]
```

Ejemplo de código en 2023:

```
posActuales[0][c] = posC.getSFVec3f()[1]
```

```
posActuales[1][c] = posC.getSFVec3f()[0]
```

Con este cambio, las velocidades calculadas ya tenían una mejor orientación con respecto a lo que buscaba producir el algoritmo original, aunque los robots todavía no tenían la suavidad de trayectoria esperada, teniendo problemas incluso con la formación inicial, ya que algunos subgrupos de agentes divergían.

Actualización de parámetros

Para refinar la migración, se recurrió a hacer un reajuste de parámetros tanto para los pesos de la ecuación de consenso, las funciones de tensión, así como la constante K del control proporcional para la meta. Para encontrar los parámetros óptimos, se optó por reconstruir el algoritmo por partes, activando segmentos y procesos paso a paso. A continuación se muestran las iteraciones para comprobación de funcionamiento de cada paso.

En la Figura 29 se puede observar el funcionamiento obtenido con los parámetros no optimizados. También se observa la trayectoria del primer agente (líder) hacia el objetivo generada con control proporcional en la Figura 30. Se destaca que la forma triangular de la formación no está muy bien definida, siendo un poco curvada, mientras que los robots oscilaban mucho en sus posiciones, lo que podría causar problemas para los agentes en las pruebas físicas, por lo que se analizaron los parámetros. Se encontró los cambios necesarios a los parámetros que se tenían que realizar, con los que se obtuvo el resultado en la Figura 31.

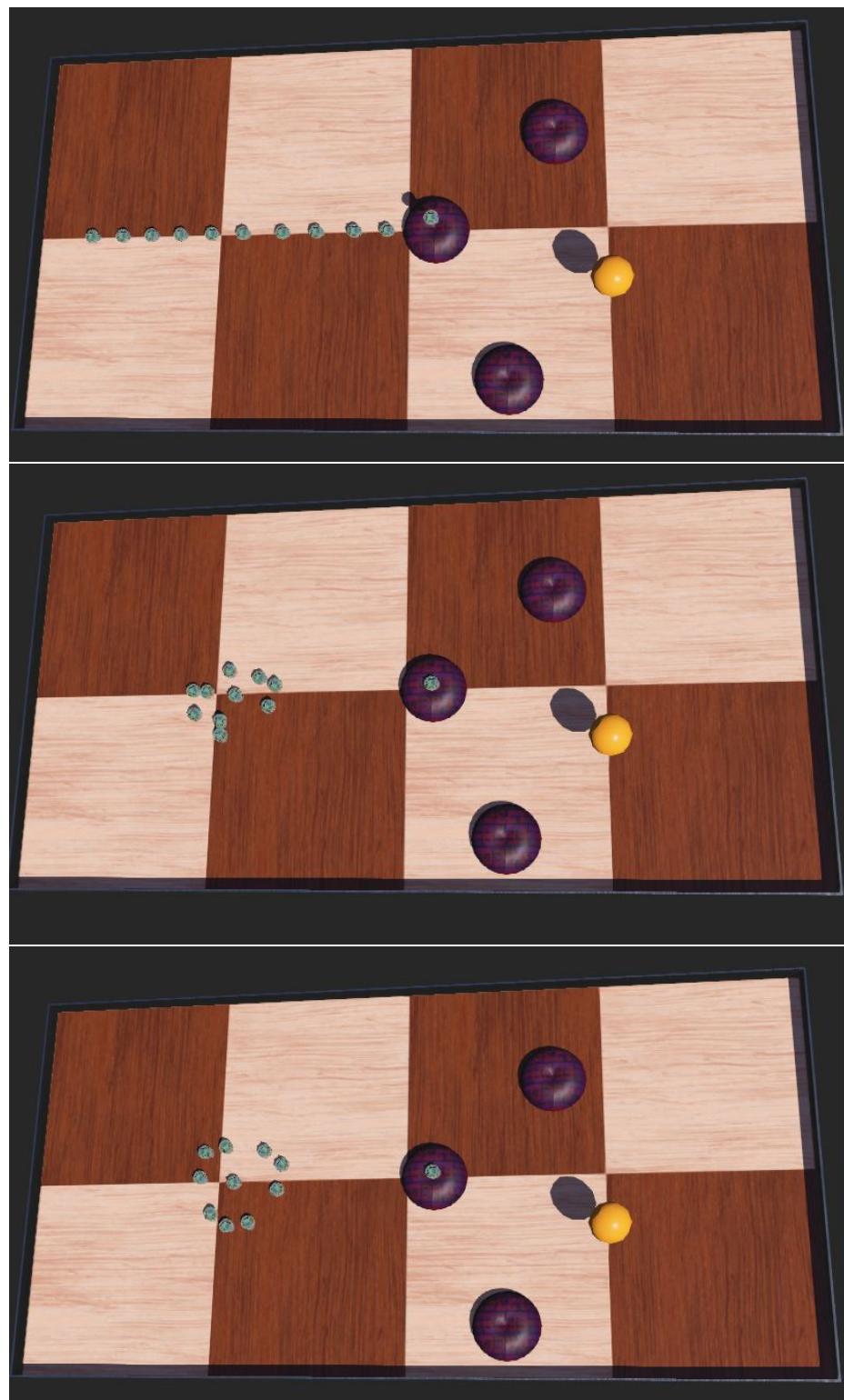


Figura 29: Prueba de funcionamiento de formación.

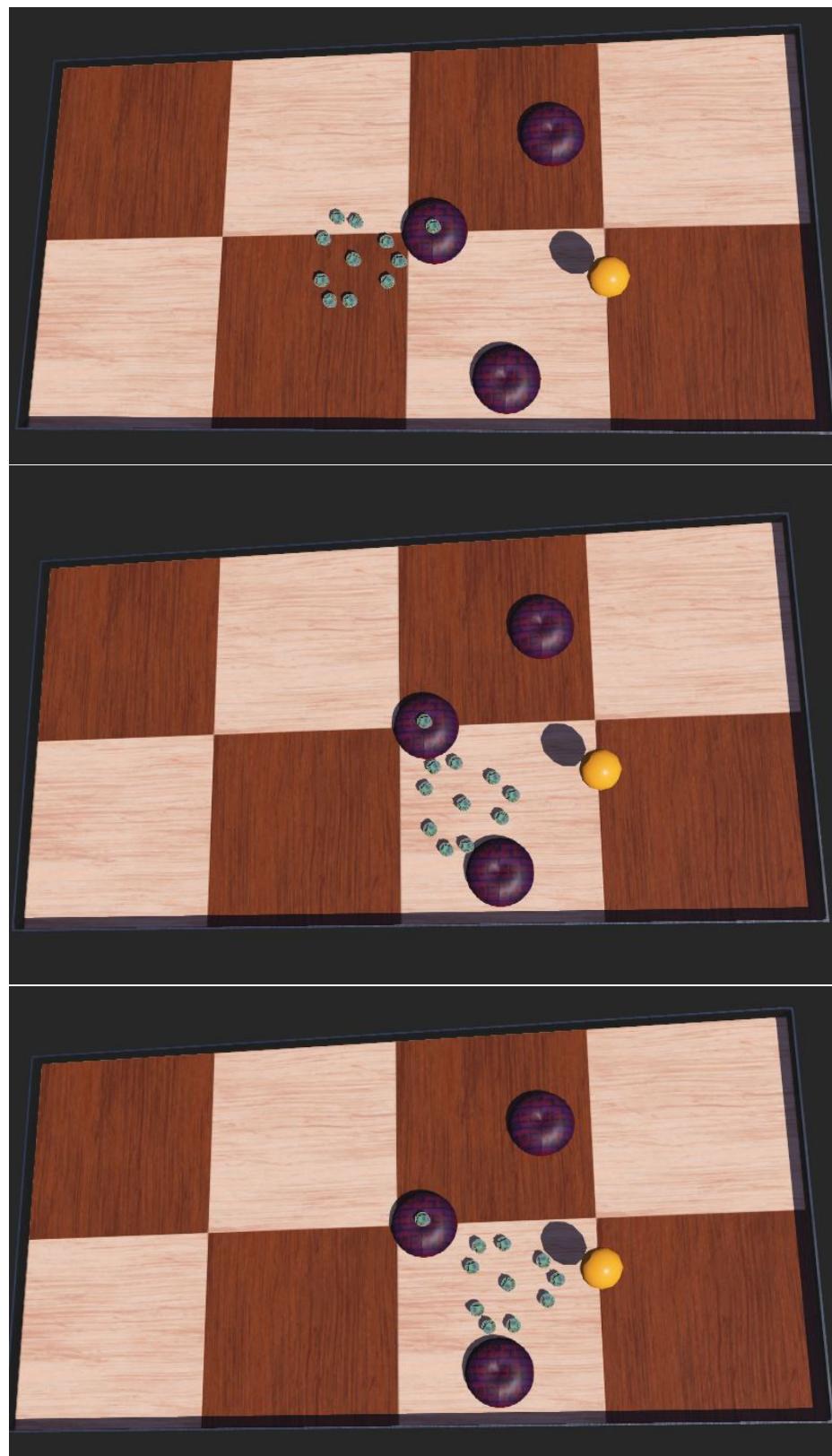


Figura 30: Prueba de funcionamiento de movimiento en formación hacia la meta.

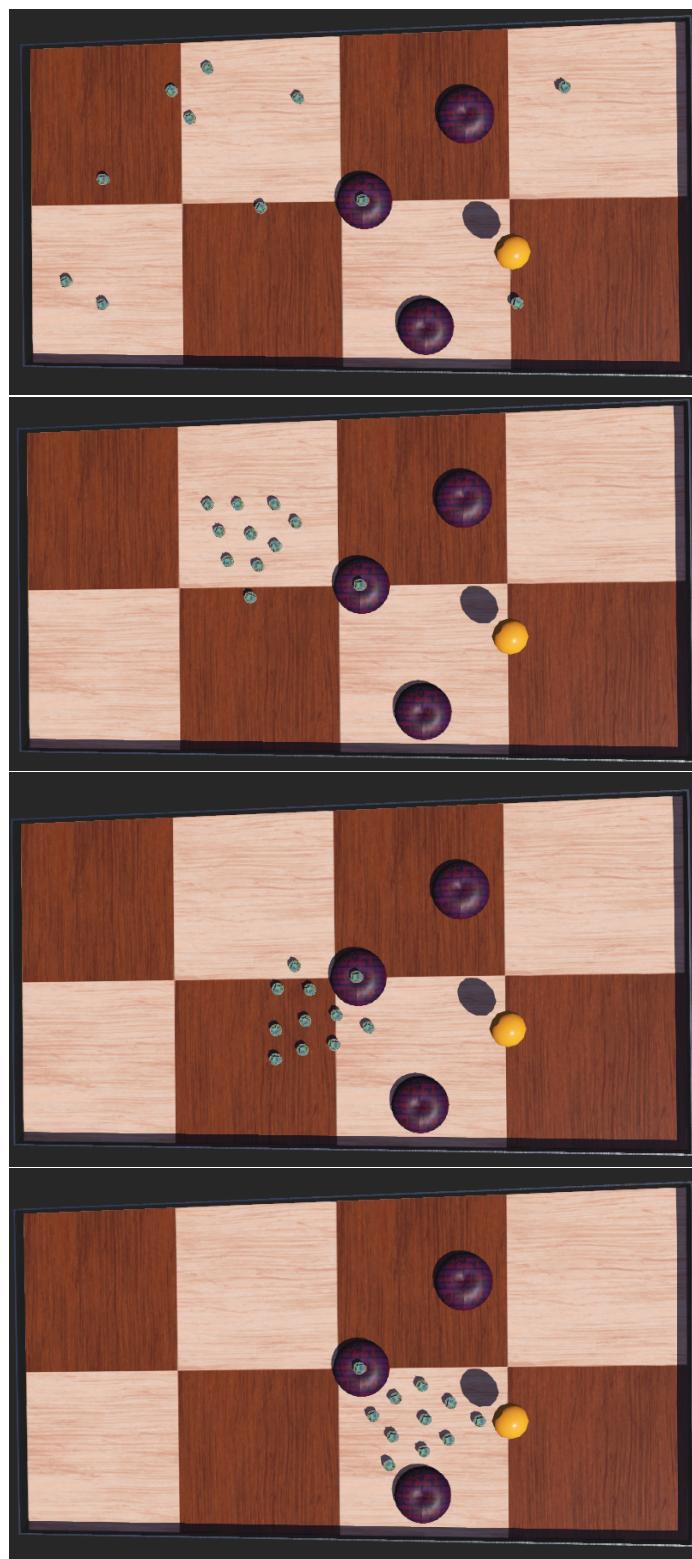


Figura 31: Resultado de optimización de parámetros para formación.

En la Figura 31 se puede observar las mejoras tanto en forma como en el movimiento

hacia el objetivo, siendo más representativas a lo que busca el algoritmo original. Además, se restauró el posicionamiento aleatorio inicial de agentes dentro de la arena.

Los parámetros que se modificaron para las mejoras fueron:

- Radio para evitar colisiones: cambio de 0.49 a 0.1.
- Se duplicó el peso para la tensión de aristas entre agentes.
- Se recortó a la mitad el peso para el cálculo de la velocidad final de cada agente de la formación, así como se invirtió el signo del cálculo de la velocidad, pues de lo contrario el resultado era que los agentes se alejaran en formación, contrario al acercamiento requerido.
- Se activó la evasión de colisiones con obstáculos, con un peso de

7.4. Variantes

Durante el desarrollo de la restauración del algoritmo surgieron diversas variantes del algoritmo, los cuales resultan ser potenciales objetos de estudio, por lo que su mención es valiosa para futuras investigaciones. Ambas variantes usan la función de tensión de la Ecuación 10.

Algunas variantes relevantes se mencionan a continuación.

7.4.1. Creación de subgrupos de formación

Esta variante surgió de invertir el signo del peso del cálculo de velocidades final, y aumentar el radar de colisiones ($r = 0.49$). El resultado en cuestión era la reducción del rango de detección de agentes lejanos pero la optimización de búsqueda de subgrupos de formación. Los agentes solo se podían juntar completamente si sus posiciones iniciales estaban dentro del radar de detección. Ver Figura 32.

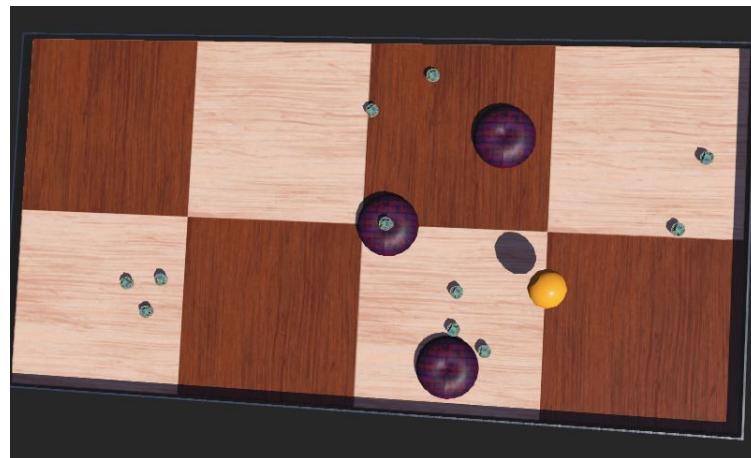


Figura 32: Variante de creación de subgrupos de formación.

7.4.2. Alejamiento de agentes en formación

Este resultado surge de invertir el signo del peso del cálculo de velocidades final, sin aumentar el radar de colisiones ($r = 0.1$). El comportamiento producido es el de una formación, pero en lugar de ser convergente es divergente, por lo que los agentes se “evitan” en formación. Este resultado podría ser útil para aplicaciones de reconocimiento de campo, optimizando la acción de cubrir el mayor terreno posible al estar separados de forma coordinada, además de posicionamiento estratégico para recepción, búsqueda de señales. Este comportamiento puede tener un mayor potencial para un enfoque de robótica de enjambre descentralizado, pero requiere de más investigación. Ver Figura 32.



Figura 33: Variante de alejamiento de agentes en formación.

CAPÍTULO 8

Adaptación del algoritmo para su ejecución en un entorno físico

Una vez restaurado el algoritmo en simulación, se prosiguió con la fase de adaptación del algoritmo y su respectiva optimización. En este capítulo se profundiza en la migración de funciones necesarias para comunicarse con el ecosistema Robotat, los robots diferenciales Pololu 3Pi+, así como los modelos generados a través de las distintas iteraciones para incluir los segmentos físicos, integrarlos y probarlos con sus opciones de configuración.

8.1. Migración de funciones

Desde la integración del Robotat y los Pololus 3Pi+ al laboratorio de la universidad, estudiantes y catedráticos [7] colaboraron para levantar un servidor llamado *mocap_server* para la obtención de información del sistema de captura de movimiento OptiTrack. También desarrollaron las funciones de adquisición de información por medio de Matlab, así como el envío de instrucciones a los Pololu 3Pi+. Para optimizar el algoritmo del presente trabajo, se decidió migrar las funciones ya existentes desde Matlab hacia Python. Las razones principales fueron:

- Estandarizar todo el código para trabajar con un solo lenguaje de programación.
- Python resulta más rápido y eficiente que Matlab en su funcionamiento de Webots y conexión/procesamiento de datos con el servidor (el servidor también está construido en Python).
- Resulta más simple migrar solamente las funciones de comunicación hacia Python, que todo el algoritmo optimizado hacia Matlab.

8.1.1. Comunicación con servidor Robotat

Para comunicarse con el servidor TCP del Robotat, existían 3 funciones fundamentales, en Matlab, las cuales eran:

- robotat_connect.m:

Sirve para establecer la conexión con el servidor.

- #### ■ robotat disconnect.m:

Realiza una desconexión del servidor.

- robotat_get_pose.m:

Obtiene las poses de los agentes requeridos. Recibe como argumento el objeto TCP, los marcadores de interés y el tipo de representación (secuencia de ángulos de Euler o cuaterniones). Devuelve la posición en x,y,z y la secuencia de ángulos de Euler intrínsecos (3 valores) o cuaterniones de forma (4 valores con magnitud primero: w,x,y,z).

Se replicaron las funciones previamente mencionadas en un archivo de funciones de Python, llamado funciones_conjunto. Este archivo tiene la diferencia de que la función de robotat_get_pose devuelve solo cuaterniones y la función quat2eul realiza la conversión de cuaterniones a ángulos de Euler intrínsecos. La función de quat2eul se construyó con la librería de Python *scipy*.

A continuación se muestra una prueba de obtención de poses con las funciones de Python.

Figura 34: Obtención de poses de múltiples marcadores usando las funciones migradas a Python.

8.1.2. Control de Pololu 3Pi+

Como en el caso de las funciones para la obtención de poses de marcadores, también existían las funciones para controlar el Pololu 3Pi+, las cuales se detallan a continuación:

- #### ■ robotat 3pi connect.m:

Realiza la conexión con el agente deseado, recibe como argumento el número de Pololu a conectar.

- robotat 3pi disconnect.m:

Desconecta al robot seleccionado. Toma como argumento el objeto TCP del agente, creado por la conexión.

- robotat_3pi_force_stop.m:

Envía un comando al robot seleccionado para detenerse poniendo las velocidades en cero. Toma como argumento el objeto TCP del agente, creado por la conexión.

- robotat_3pi_set_wheel_velocities.m:

Envía un comando al robot seleccionado para actualizar las velocidades de las 2 ruedas, las velocidades mínima y máxima son -850 y 850 rpm respectivamente, si se ingresan valores mayores se truncan a los mencionados previamente. Toma como argumento el objeto TCP del agente, creado por la conexión, y las velocidades de la rueda izquierda y derecha.

Las funciones previamente se replicaron en Python en un solo archivo llamado funciones_conjunto_3pi.py. A continuación se muestran los resultados de pruebas de funcionamiento:

```
PS D:\AlejandroDigital\tesisAlejandro\codigo\comunicacion_pololu>
Connected to robot ID: 7, IP: 192.168.50.107, Port: 8888
Disconnected from robot.
```

Figura 35: Conexión con Pololu 3Pi+ con ID 7, usando las funciones migradas a Python.

En la Figura 36 se muestra una prueba en la que se pone a 2 Pololus a girar sobre su propio eje en direcciones contrarias, por 6 segundos antes de detenerse.

En la Figura 37 se puede observar la secuencia de una prueba de envío de instrucciones para que los Pololus se movieran en línea recta, a su vez adquiriendo la pose de los marcadores montados sobre los Pololus. Con este demo se probó usar las funciones de obtención de pose del servidor combinadas con las funciones de envío de instrucciones hacia Pololus, en simultáneo, la cual fue exitosa.

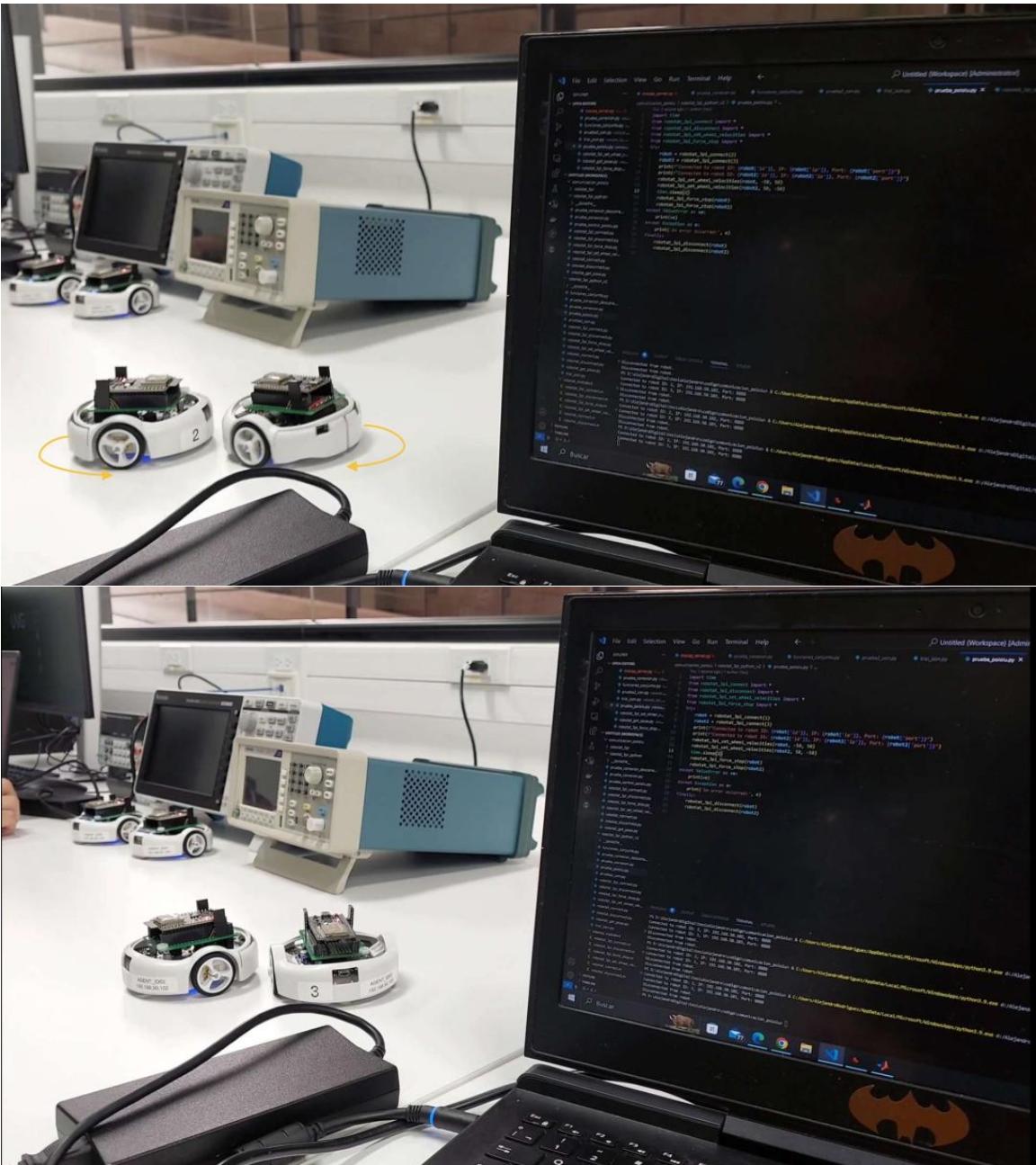


Figura 36: Envío de instrucciones para poner a girar a agentes Pololu, usando las funciones migradas a Python.



Figura 37: Envío de instrucciones para mover en línea recta a agentes Pololu, usando las funciones migradas a Python.

8.2. Modificaciones al algoritmo en entorno Webots con pruebas preliminares

Una vez verificado el funcionamiento de las funciones para interactuar con el sistema físico en el Robotat, se procedió a integrar gradualmente estas funciones al programa de Supervisor de Webots, y subsecuentemente al programa de control individual de agente.

También se realizaron cambios al archivo de mundo de Webots, para tener una representación más fiel de las pruebas en el Robotat. Las modificaciones y adiciones principales fueron:

- Las dimensiones de la arena del mundo de Webots adoptaron las medidas de la mesa de pruebas del Robotat, es decir: $3.8 \text{ m} \times 4.8 \text{ m}$ ($x \times y$), con el origen en el centro de la arena.
- Implementación de marcas de posición inicial de agentes para escenario, para facilitar la planeación, orden, inicialización y ejecución de pruebas.
- El número de agente ahora no comienza desde 0, sino desde 1, para que tenga la misma convención de números que los agentes físicos.

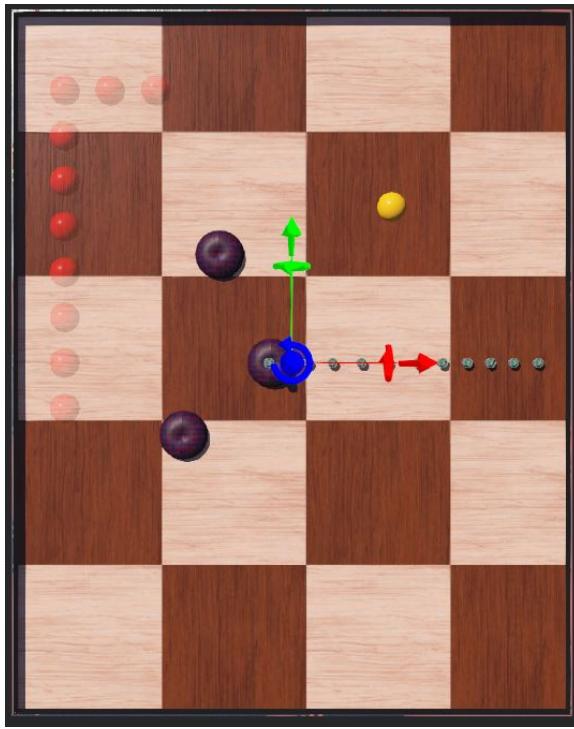


Figura 38: Mundo de Webots basado en Robotat y optimizado para pruebas.

En la Figura 38 se puede observar el mundo modificado en base a las especificaciones del Robotat con la adición de marcas. Los cuadros miden 1 m^2 , el eje y se muestra en color verde, el eje x se muestra en rojo y el eje z en azul.

Los objetos presentes son:

- Obstáculos: Toroides morados.
- E-pucks (Robots diferenciales): Pequeños círculos de color aqua.
- Marcas de posición inicial de agentes: Esferas rojas semitransparentes.
- Objetivo final: Esfera amarilla.

8.2.1. Calibración de marcadores para su uso en el algoritmo

Antes de comenzar a realizar las pruebas con los agentes físicos, se llevó a cabo una calibración de los marcadores, alineando a todos con el eje y, desde la parte negativa en dirección al origen. Al alinear los marcadores, con la misma orientación con la que irían montados sobre los Pololus, se logró encontrar el desfase del ángulo de *bearing*, el cual es diferente para cada uno de los marcadores. Este desfase es causado por la naturaleza de la forma en que el OptiTrack identifica los marcadores, pues el ID de cada marcador está representado por su configuración de esferas reflectivas, produciendo diferentes combinaciones, lo que produce que cada marcador se encuentre por defecto en una orientación distinta.

La forma en que se corrige el desfase del ángulo es simple, pues al momento de obtener los ángulos de Euler en secuencia *zyx*, el primer valor representa el desfase del *bearing* respecto del eje *z*. Al valor extraído luego se le resta al ángulo obtenido cada vez que se adquieran datos, lo que compensa el desfase.

Para hacer el proceso de calibración más eficiente, se obtuvieron los datos de los primeros 15 marcadores inmediatamente después de hacer la calibración del OptiTrack para minimizar el error, y se guardaron en un archivo .npy, optimizado para arreglos de numpy. Antes de cada corrida en físico, se cargan estos valores al programa al comienzo del mismo y se aplican en cada actualización de nuevos datos para realizar la compensación. En la Figura 39 se puede observar la calibración inicial.

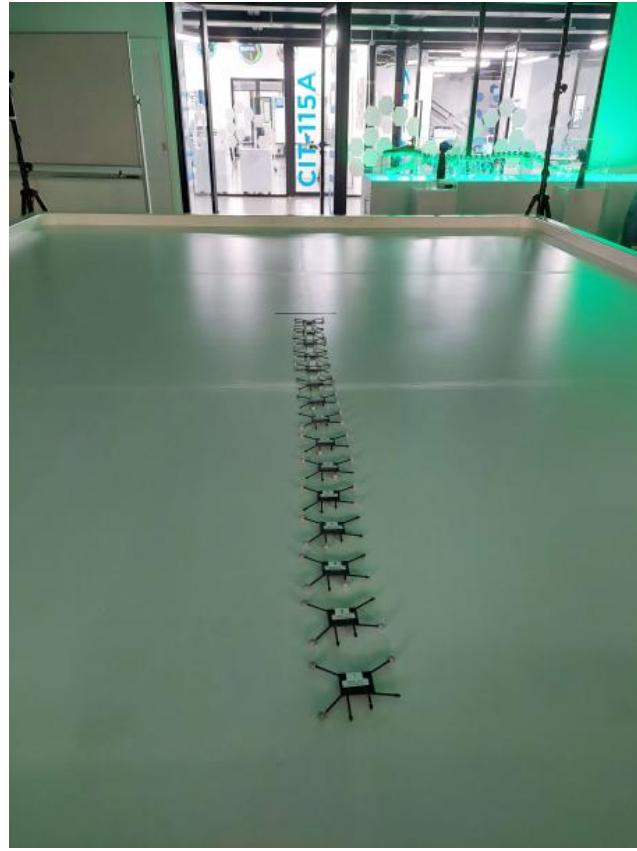


Figura 39: Calibración de marcadores 1 al 15 para su uso en Pololus 3Pi+.

Marcador	Desfase θ_z en grados
1	91.99470274710572
2	-46.814569482191594
3	-92.39049071644509
4	-138.20668559103328
5	176.37515477240987
6	-144.1821533175259
7	-176.31925348204803
8	-79.95245389000435
9	-9.87621045801094
10	139.3578557303511
11	111.93284607034238
12	167.57610128913143
13	-128.0708601137765
14	-111.1403638963379
15	-43.41121657780576

Cuadro 4: Desfases angulares al alinear los marcadores con el eje y.

En el Cuadro 4 se puede apreciar los desfases resultantes del procedimiento mostrado en la Figura 39, al alinear los marcadores con el eje y . Se busca tener un $\theta_z = 0$ para todos los marcadores, por lo que se aplica el proceso mostrado en la Figura 40, en cada ciclo del algoritmo.

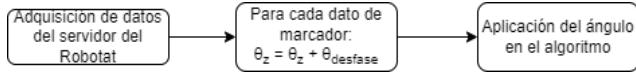


Figura 40: Proceso de calibración de ángulo de desfase de *bearing*.

Una vez calibrados los marcadores que sirven para localizar a los agentes, se prosiguió a ejecutar pruebas con los Pololus 3Pi+ en el Robotat. A continuación se enumeran los modelos desarrollados con sus respectivas pruebas para la adecuación del algoritmo a su aplicación física, así como los resultados obtenidos para cada modelo.

8.2.2. Modelo 1: Aplicación de información de marcadores

Esta iteración tuvo como objetivo obtener una configuración de marcadores del entorno físico y transferirla al entorno de simulación de Webots.

Modelo 1.0: Simulación con información guardada de marcadores

La primera integración de información del entorno real se realizó al guardar las posiciones de una configuración específica de marcadores en un archivo .pickle, para luego abrirlo desde el Supervisor en Webots, y configurar el posicionamiento inicial de las entidades virtuales.

En la Figura 41 se logra observar los marcadores iniciales, dispuestos en la mesa, y al

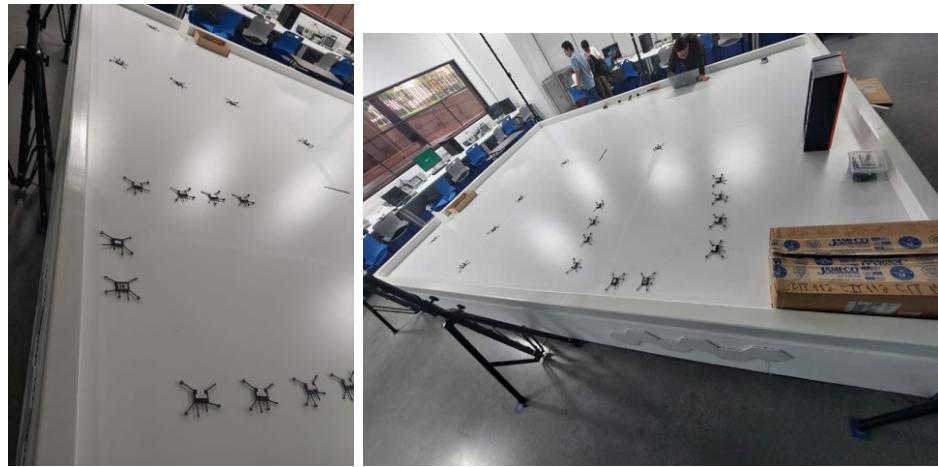


Figura 41: Escenario de primera prueba en el Robotat, visto desde distintos ángulos.

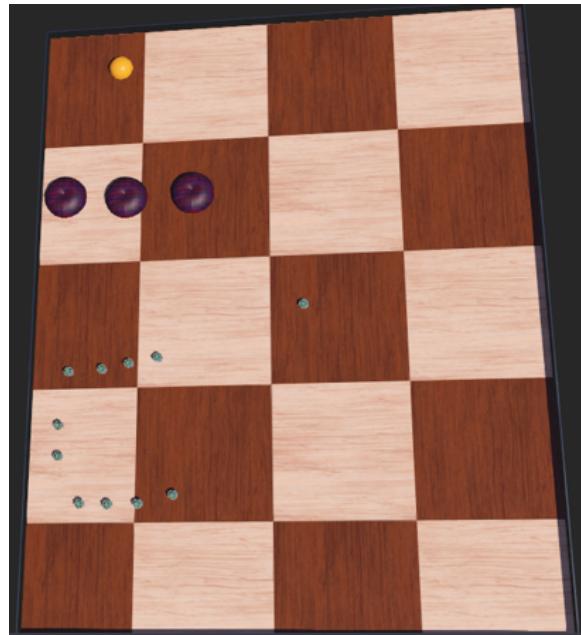


Figura 42: Resultado al cargar la información guardada del mundo real en Webots.

comparar la imagen de la izquierda con la Figura 42 es notable que la configuración de marcadores se trasladó con éxito del entorno físico al entorno virtual. Con esta prueba se confirmó el funcionamiento del posicionamiento y localización de marcadores.

Modelo 1.1: Simulación con información obtenida en tiempo real de marcadores

La segunda iteración de este modelo consistió en adquirir las posiciones de los marcadores con el algoritmo corriendo en tiempo real, dentro de un controlador de Webots. Con esto se verificó que era posible usar las funciones del Robotat desde el Supervisor de Webots y actualizar las entidades virtuales según la configuración actual física de la mesa de pruebas. Además, en el Modelo 1.1 se buscó una adquisición más práctica de la información de los

marcadores sin la necesidad de un paso intermedio de guardado, optimizando así el proceso.

El resultado fue el mismo que en la Figura 42, con la diferencia de que ahora las posiciones se actualizaban constantemente en tiempo real, no solo al comienzo.

8.2.3. Modelo 2: Pruebas preliminares de algoritmo dinámico en físico

Este modelo consistió en probar a los agentes Pololu 3Pi+ en el algoritmo de Webots, tanto del programa del Supervisor como del agente individual, para verificar su funcionamiento, desempeño y parámetros óptimos. Todas las iteraciones de este modelo tienen la característica de que funcionan exclusivamente en físico o en simulación, mas no ambas en simultáneo.

Modelo 2.0: Seguimiento de objetivo real con 1 agente

El primer prototipo de este modelo se concentró en aplicar la parte de seguimiento de un objetivo usando un marcador real, para realizar la calibración de velocidad y ajustes de ángulo necesarios en el algoritmo. El seguimiento es realizado por un Pololu 3Pi+ con un marcador montado encima de él.

Al comienzo de esta iteración se obtuvo como resultado una variante, en la que el Pololu sí buscaba ir hacia el objetivo, pero de una forma perpendicular, lo que significaba que buscaba alinear el eje de sus llantas para que siempre tuviera al objetivo en paralelo a las mismas. Esto ocasionaba que el robot se mantuviera dando vueltas alrededor del objetivo, con una posibilidad muy baja de realmente llegar al punto, debido a la lentitud de acercamiento. Su comportamiento se puede observar en las Figuras 43, 44 y 45.

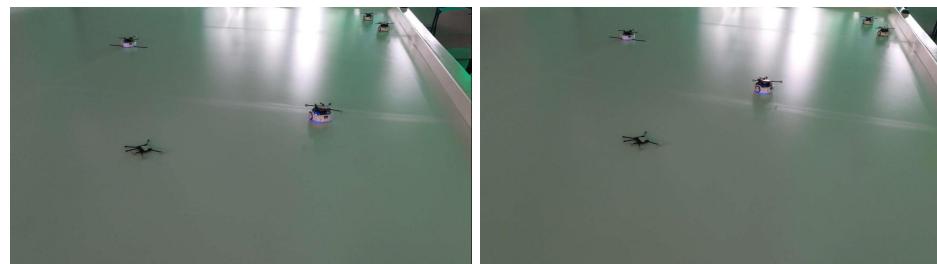


Figura 43: Movimiento en círculos alrededor del objetivo, del modelo 2.0.

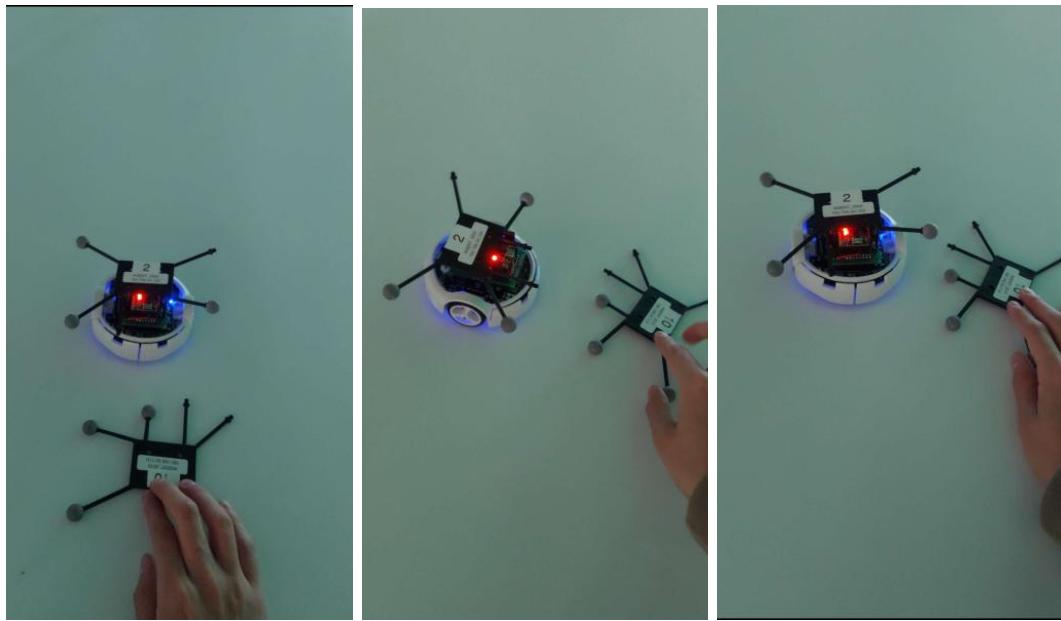


Figura 44: Secuencia del movimiento de alineación perpendicular del Modelo 2.0.

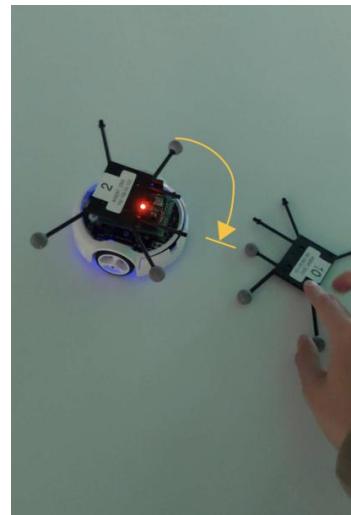


Figura 45: Descripción del movimiento de alineación perpendicular del Modelo 2.0.

Luego de un análisis de su comportamiento, se encontró la causa del mismo. La alineación perpendicular era resultado de una discrepancia entre la forma en que se calibraron los desfases, pues los marcadores (que a su vez están alineados con los Pololus), se alinearon hacia el eje y , mientras que el algoritmo en Webots se había programado en base a los E-pucks, cuyas entidades están alineadas al eje x , como se muestra en la Figura 46. El error fue simple de corregir, pues simplemente se añadió un desfase de compensación extra de $+90^\circ$ al ángulo θ_z , en adición a los desfases mencionados en el Cuadro 4, volviendo así al algoritmo compatible con la orientación calibrada.

Una vez corregido, el agente fue capaz de seguir al objetivo con éxito. Se requirió de un

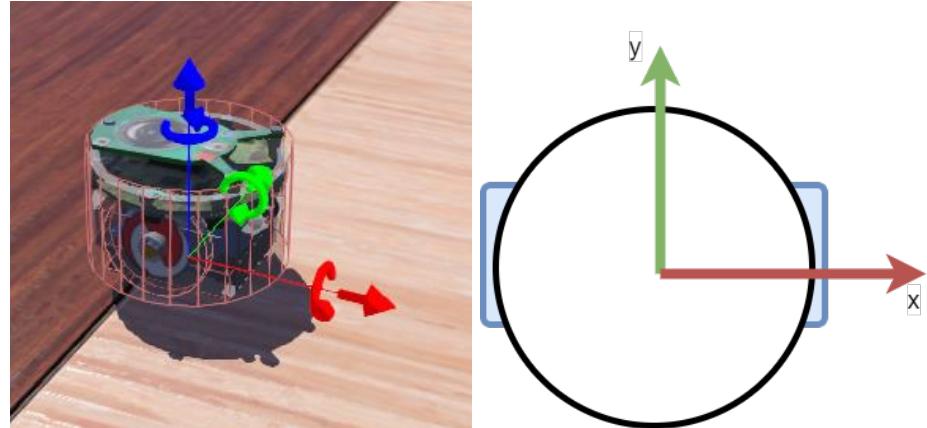


Figura 46: Orientación del E-Puck vs. Pololu con respecto de sus ejes.

ajuste heurístico de la velocidad con la que el agente se acercaba hacia el objetivo, pues en un principio la velocidad elevada de giro producía cambios muy agresivos de orientación en el agente, por lo que se optó por reducir la velocidad angular de las llantas a 50 rpm. Otro cambio que se implementó fue el de modificar la constante del control proporcional para que el agente no disminuyera drásticamente su velocidad conforme se acercara al objetivo, pues con la constante unitaria luego de recorrer la mitad del trayecto hacia el objetivo, se tardaba demasiado en llegar al mismo. La constante que presentó un desempeño más equilibrado fue $K = 5$. El comportamiento exitoso se muestra a continuación, en las Figuras 47, 48, 49, 50 y 51.

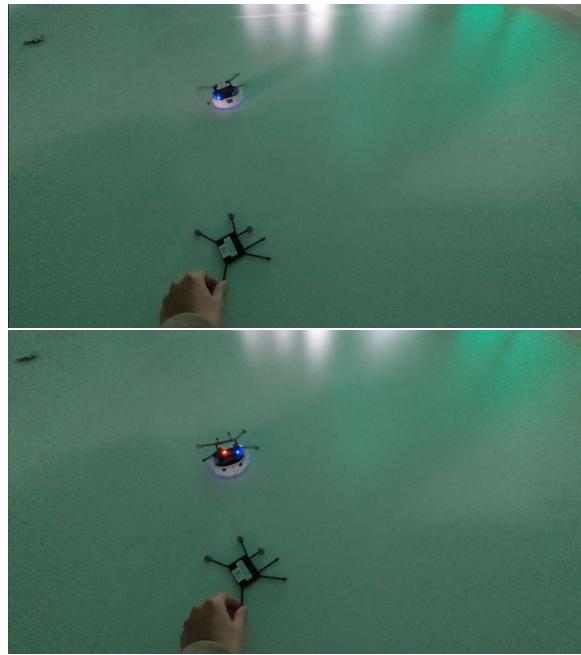


Figura 47: Secuencia de seguimiento dinámico de objetivo, prueba 1.



Figura 48: Descripción del seguimiento dinámico de objetivo, prueba 1.

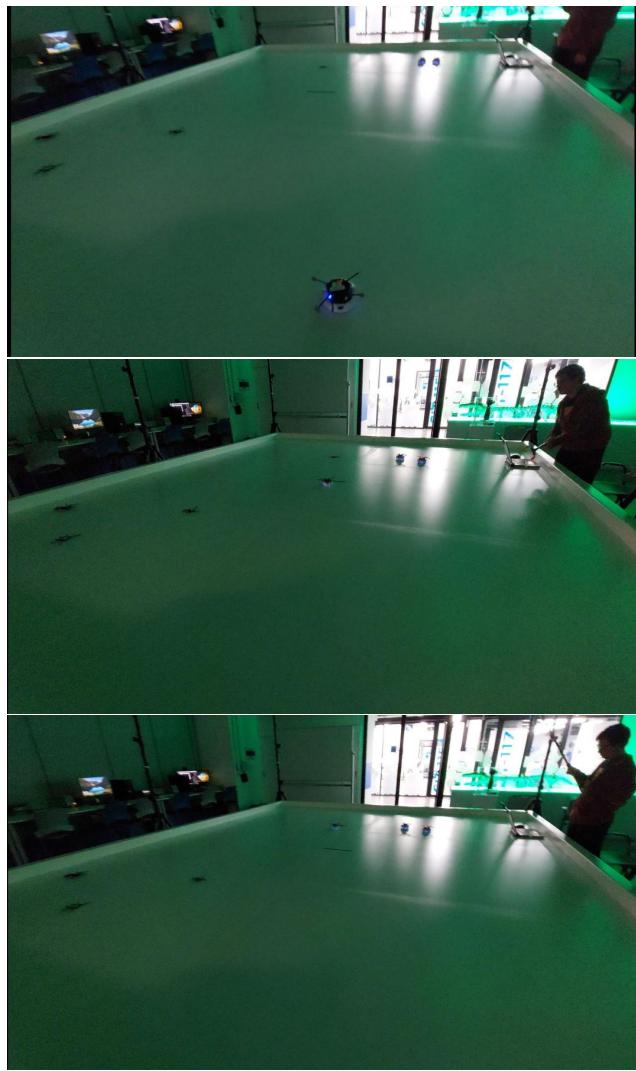


Figura 49: Secuencia de seguimiento dinámico de objetivo, prueba 2.

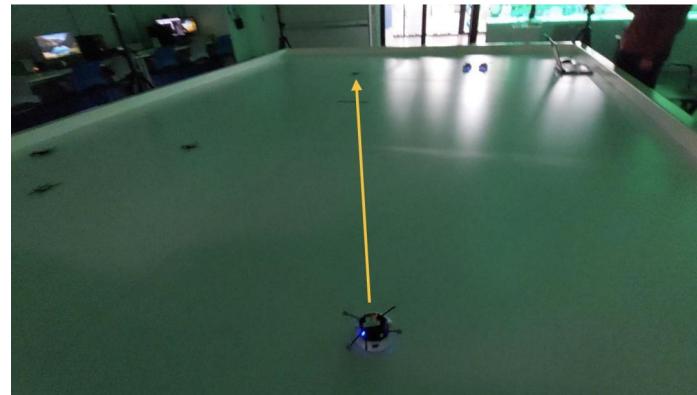


Figura 50: Descripción del seguimiento dinámico de objetivo, prueba 2.

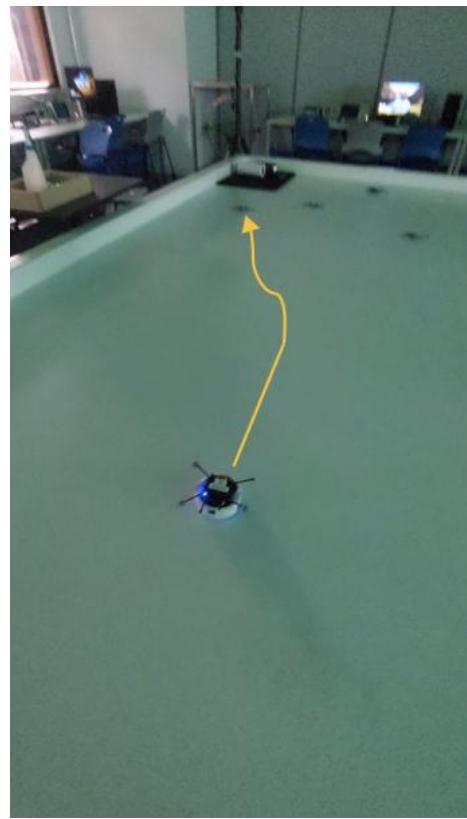


Figura 51: Descripción del seguimiento dinámico de objetivo, prueba 3.

Cabe resaltar que para todos los casos el Pololu se reduce su velocidad hasta prácticamente detenerse una vez su posición en x y y coincida con la del marcador objetivo, como se ve en la Figura 52.



Figura 52: Agente detenido al llegar al marcador objetivo.

Modelo 2.1: Seguimiento de objetivo virtual con agentes físicos

La segunda versión del modelo consistió en realizar experimentos híbridos, en los que los Pololus se guiaban hacia un objetivo virtual, fijado como una entidad de Webots. A continuación, en las Figuras 53, 54 y 55 se muestran los resultados del comportamiento en varias pruebas.

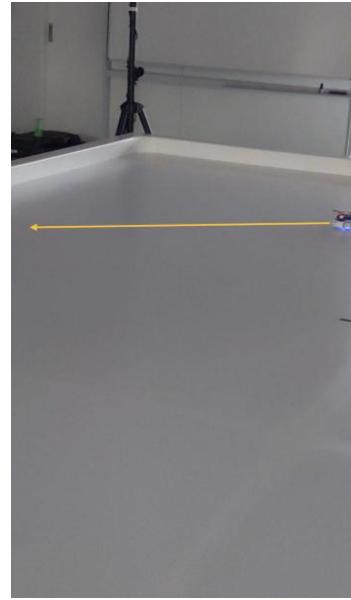


Figura 53: Descripción de movimiento de 1 agente hacia objetivo virtual, prueba 1.

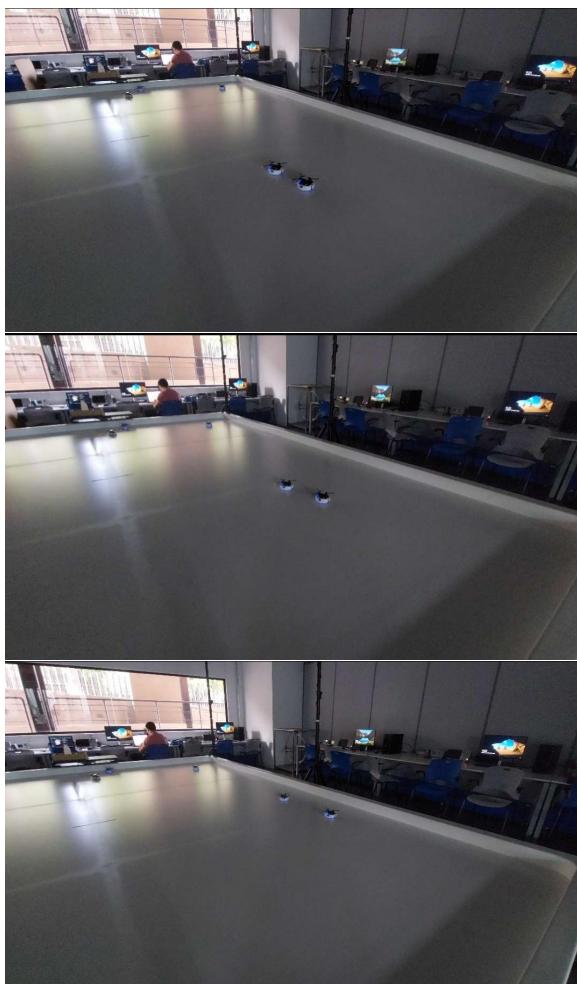


Figura 54: Secuencia de movimiento de 2 agentes hacia objetivos virtuales, prueba 2.

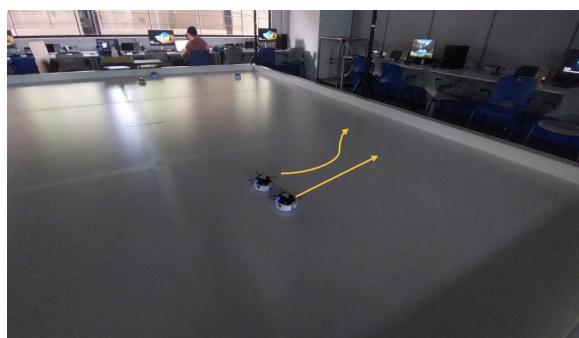


Figura 55: Descripción de movimiento de 2 agentes hacia objetivos virtuales, prueba 2.

Con la realización de estas pruebas se verificó el correcto funcionamiento de los Pololus para seguir un objetivo virtual, lo que permite la realización de pruebas híbridas. Las condiciones híbridas resultan ventajosas en caso no se tengan suficientes marcadores, ya que virtualmente no se tiene realmente un límite de entidades máximas disponibles.

Modelo 2.2: Seguimiento de objetivo virtual con 1 agente y obstáculos virtuales

En esta iteración se agregaron obstáculos virtuales para evaluar el desempeño del Pololu y el reconocimiento de los mismos.

Modelo 2.3: Posicionamiento de configuración inicial con múltiples agentes

El propósito de esta versión fue evaluar el posicionamiento inicial de múltiples agentes mediante la aplicación de múltiples objetivos iniciales, para facilitar la configuración de escenario, previo a la ejecución del algoritmo. En este caso los objetivos de posiciones de inicio se seleccionan virtualmente mediante Webots. En las Figuras 56, 57, 57 y 59 se muestran los resultados de las pruebas de esta iteración.

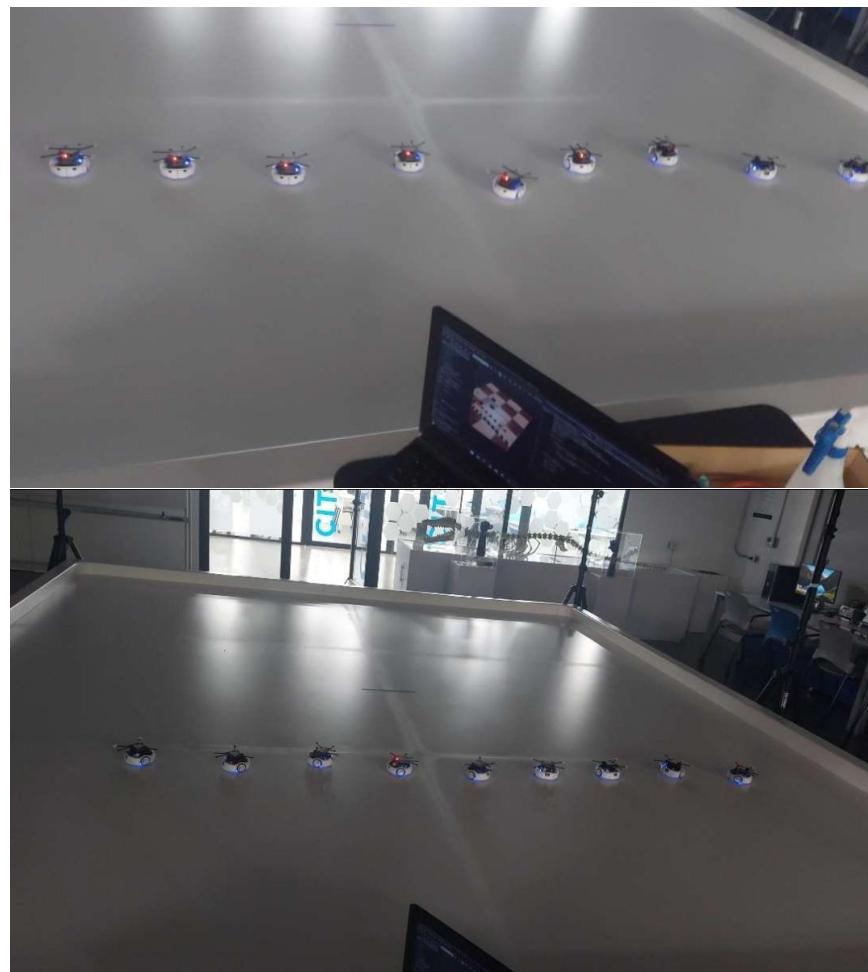


Figura 56: Secuencia de movimiento de 9 agentes, para colocarse en posiciones iniciales en línea, prueba 1.



Figura 57: Descripción de movimiento de 9 agentes, para colocarse en posiciones iniciales en línea, prueba 1.

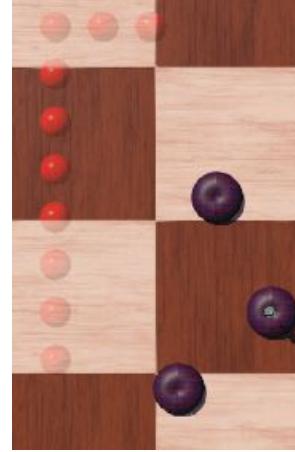


Figura 58: Escenario de posiciones iniciales para prueba 2 con obstáculos virtuales.

En la Figura 59 se puede observar la ejecución de posicionamiento inicial en forma de L invertida, basado en marcas y obstáculos virtuales mostrados en la Figura 58. Se puede observar que 8 de los 9 agentes llegaron al objetivo, 1 de los agentes no logró llegar debido a que se acercó demasiado a la orilla del Robotat. Esto sucede debido a que algunas de estas zonas lejanas al centro no son correctamente detectables por el sistema OptiTrack, por lo que en experimentos posteriores se mantuvo las posiciones iniciales dentro de un margen alejado aproximadamente 30 cm de las orillas.



Figura 59: Secuencia (izquierda a derecha) de movimiento de 9 agentes físicos y 1 agente inmóvil (marcador) hacia objetivos virtuales en forma de L invertida, prueba 2.

Modelo 2.4: Prueba de Acercamiento de agentes y formación

La versión final de este modelo consistió en la evaluación de la ejecución del segmento de acercamiento de agentes físicos y posteriormente mantener una formación. Esto sirvió para calibrar la evasión de colisiones y sus pesos para su ejecución en un entorno físico real.

En la primera prueba, mostrada en la Figura 60 se realizó únicamente un acercamiento de los agentes por medio de la ecuación de consenso, acercando todos los agentes desde sus posiciones iniciales en línea, hasta formar un cúmulo de agentes reunidos en un área del mapa.

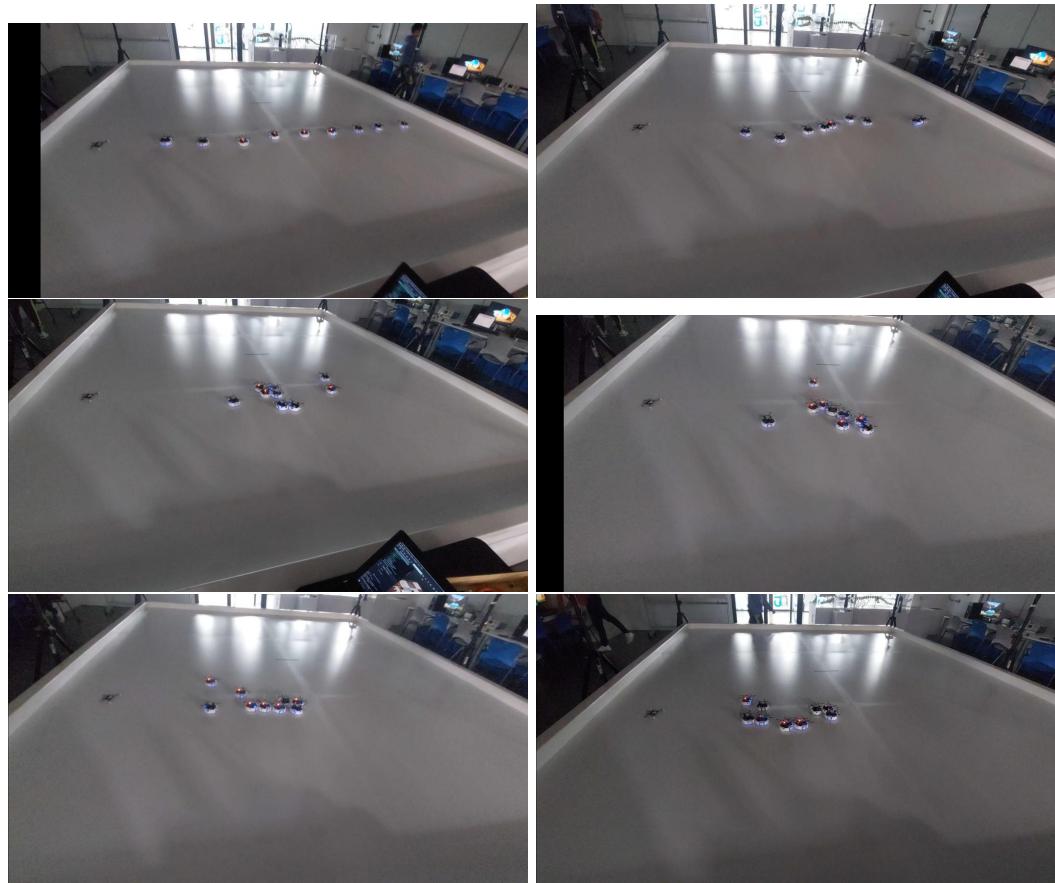


Figura 60: Secuencia (izquierda a derecha) de movimiento de 9 agentes físicos y 1 agente inmóvil (marcador) para acercar a los agentes entre sí, prueba 1.

En la segunda prueba, se activó la función de tensión para la formación, con lo que se obtuvo el resultado de formación mostrado en la Figura 61.



Figura 61: Resultado de la formación con 9 agentes físicos y 1 agente inmóvil (marcador), prueba 2.

Por último, se verificó el seguimiento dinámico del líder por parte del conjunto de agentes en formación, con lo que se obtuvo la secuencia mostrada en la Figura 62.

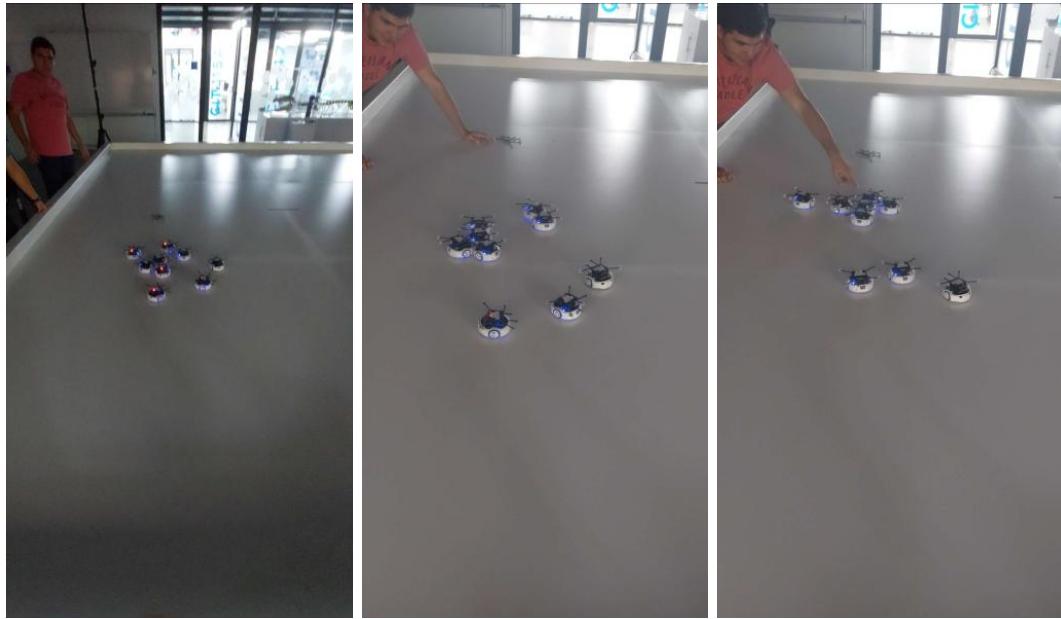


Figura 62: Secuencia (izquierda a derecha) del resultado del movimiento artificial del líder (marcador 1) y el seguimiento de la formación (agentes 2 al 10).

Luego de la ejecución de las pruebas del algoritmo en el Modelo 2.4, se observó que los agentes se acercaban demasiado entre sí en la formación, por lo que se requirió un reajuste de parámetros, específicamente el factor de escalamiento para el peso ω tanto para la evasión de obstáculos como para la tensión de aristas de la formación.

Las nuevas constantes de escalamiento (factor que multiplica a ω para cada caso fueron:

- Tensión de la formación: 2
- Evasión de obstáculos: 0.5

8.2.4. **Modelo 3: Integración para funcionamiento híbrido de algoritmo en físico y almacenamiento de datos**

Este fue el último modelo, con el conjunto de partes del algoritmo habilitadas para su ejecución, es decir:

- Movimiento de los agentes Pololu 3Pi+ hacia las posiciones iniciales del escenario.
- Evasión de colisión con obstáculos y agentes.
- Acercamiento de agentes entre sí.
- Identificación de los agentes cuando están suficientemente cerca de sí mismos.

- Activación de función de tensión para armar y mantener la formación.
- Movimiento del líder hacia el objetivo.

Optimización de versatilidad y aplicación del código

Otro de los cambios más importantes para este último modelo fue la modificación de la inicialización y actualización de varias entidades en Webots. Previamente tanto los agentes, como los obstáculos se inicializaban uno por uno, alargando el código de forma innecesaria. Para una mayor eficiencia de ejecución y facilitar la comprensión del código se implementaron las inicializaciones con ciclos *for* para iterar sobre todas las entidades presentes en Webots. Este cambio proveyó una mayor flexibilidad al momento de cambiar las condiciones iniciales, de disposición de agentes, posiciones iniciales, posicionamiento de obstáculos y objetivo, así como la actualización en tiempo real de los mismos.

Esta optimización permitió tener un mayor control sobre el número de agentes a tomar en cuenta para la ejecución del algoritmo, en caso de que no todos estuvieran disponibles, o se quisiera obtener resultados con distinto número de agentes. Con este modelo se puede variar el número de agentes a manipular, por medio de intervalos y se puede usar desde 1 hasta 10 agentes para la ejecución del algoritmo, ya sea en físico o en simulado.

Se realizó una segmentación del modo físico y el modo de simulación, para poder elegir si el algoritmo se ejecuta en los agentes del mundo real o en los agentes de la simulación. En los siguientes capítulos, se profundiza más sobre las opciones de configuración, la recolección y almacenamiento de los datos, y se muestran los resultados obtenidos con este último modelo.

CAPÍTULO 9

Diseño Experimental

En este capítulo se incluyen diagramas de los escenarios de interés en los que el algoritmo se pone a prueba y se explican los experimentos relevantes a nivel situacional. Además se explicará sobre las adiciones relevantes al algoritmo para optimizar la recolección, procesamiento de datos y generación de gráficas.

9.1. Opciones de configuración

Para fines de estudio del algoritmo, se incluyeron 4 formas de ajuste de escenario inicial:

1. Posicionamiento de la simulación por defecto (estado actual del mundo de Webots).
2. Aparición aleatoria de agentes virtuales en la arena de simulación.
3. Posicionamiento instantáneo de agentes virtuales basado en una configuración cargada desde un archivo.
4. Posicionamiento de marcas iniciales virtuales según la elección del usuario, puede ser aleatorio o en un patrón específico deseado.

Para las primeras tres formas de ajuste, su uso está enfocado en el ambiente de simulación, mientras que en la cuarta se tienen variantes dependiendo de si se está usando el modo físico o el modo simulación.

La diferencia principal entre el modo físico y el modo de simulación es de donde provienen los datos de las posiciones de los agentes, ya que si se encuentra en modo simulación, estas provienen de los e-pucks de la simulación, mientras que si se encuentra en modo físico las

posiciones provienen de los marcadores sobre los Pololu 3Pi+ a través de la comunicación con el servidor.

Cabe resaltar que para lograr entablar un medio de comparación entre las corridas en simulación y las corridas en físico, se implementó un sistema de almacenamiento de datos en archivos *.npz*, una variante del archivo *.npy* para guardar múltiples arreglos en lugar de uno solo. Se eligió guardar en este tipo de archivo debido a que es más eficiente para la carga y lectura de datos conformados por arreglos.

Independientemente del modo, siempre se guardan los datos relevantes para generar la gráfica de las trayectorias, al final de cada corrida en un archivo *.npz*, mostrados en el Cuadro 5. En caso el modo físico esté activado, también se guardan las condiciones iniciales en otro archivo *.npz* de todas las entidades relevantes del mundo de Webots, así como las posiciones de los Pololus en la mesa de pruebas al momento de iniciar la ejecución el algoritmo.

El archivo generado con las condiciones iniciales tiene como propósito poder replicar en Webots a los experimentos realizados en el Robotat para tener una comparación más confiable y representativa de la ejecución del algoritmo. Con esta opción activada, las condiciones iniciales guardadas durante la corrida en físico se aplican al entorno de simulación y se ejecuta el algoritmo.

Variable	Tipo de dato	Información
trajectory_data	arreglo $[2 \times n \text{ agentes}] \times n \text{ ciclos}$	Cada elemento del arreglo principal contiene un arreglo con las posiciones en x y y para cada uno de los agentes involucrados. De ese arreglo interno, la primera fila representa posiciones en x y la segunda posiciones en y .
velocity_data	arreglo $[2 \times n \text{ agentes}] \times n \text{ ciclos}$	Cada elemento del arreglo principal contiene un arreglo con las velocidades en x y y para cada uno de los agentes involucrados. De ese arreglo interno, la primera fila representa velocidades en x y la segunda velocidades en y .
ciclo	entero	Representa el número de ciclos que duró la corrida del algoritmo.
posObsAct	arreglo $2 \times n \text{ obstáculos}$	Contiene las posiciones en x y y para cada obstáculo.
sizeO	entero	Su valor indica el tamaño del obstáculo a considerar.
NStart	entero	Indica el agente con ID más bajo desde el cual se incluye en la formación. También representa al líder de la formación.
pObjVec	arreglo 1×3	Un vector en x , y y z que indica las coordenadas del objetivo.

Cuadro 5: Tabla de variables almacenadas al final de la corrida del algoritmo.

9.2. Configuración de escenario

Para validar el funcionamiento del algoritmo se tomó como base el escenario con 3 obstáculos y 1 objetivo del otro lado de los obstáculos, estudiado previamente por [15], con la modificación de que en esta instancia se añadieron marcas de posiciones iniciales para ejecutar el algoritmo consistentemente entre las corridas en simulación y físico. Se optó por generar un patrón de línea para asignar las posiciones de las marcas iniciales.

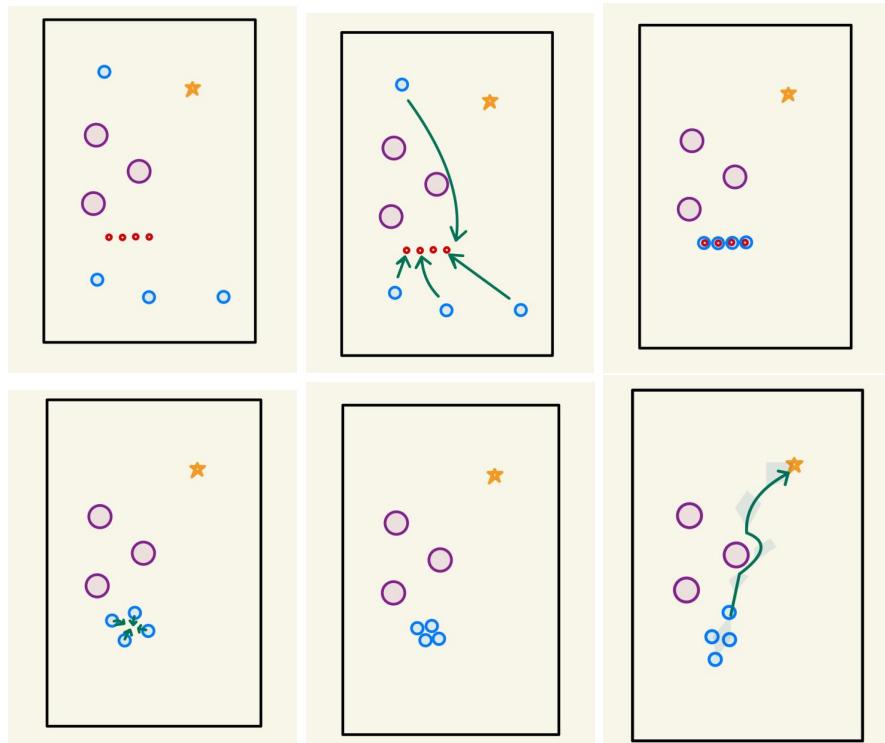


Figura 63: Diagrama de diseño experimental 1.

CAPÍTULO 10

Ejecución de algoritmo de sincronización y control de formaciones en los robots diferenciales en ambiente controlado

En este capítulo se presentan los resultados de los experimentos y configuraciones de escenario planteadas en el capítulo anterior. Además se muestran las trayectorias en el escenario, así como la comparación entre la ejecución en simulación y físico. Todos las secuencias de fotogramas desplegadas se interpretan de izquierda a derecha y luego de arriba hacia abajo.

10.1. Resultados

A continuación se muestran ejecuciones del algoritmo, variando el número de agentes utilizados para observar su comportamiento y trayectorias.

10.1.1. Experimento 1: Ejecución del algoritmo completo en físico con 4 agentes

Este experimento consistió en observar la ejecución completa del algoritmo, con su evaluación de colisión con obstáculos y agentes, posicionamiento inicial de agentes para la prueba. En las figuras 64, 65 y 66 se muestran las secuencias de movimiento de la ejecución del algoritmo con el respectivo comportamiento de cada etapa.

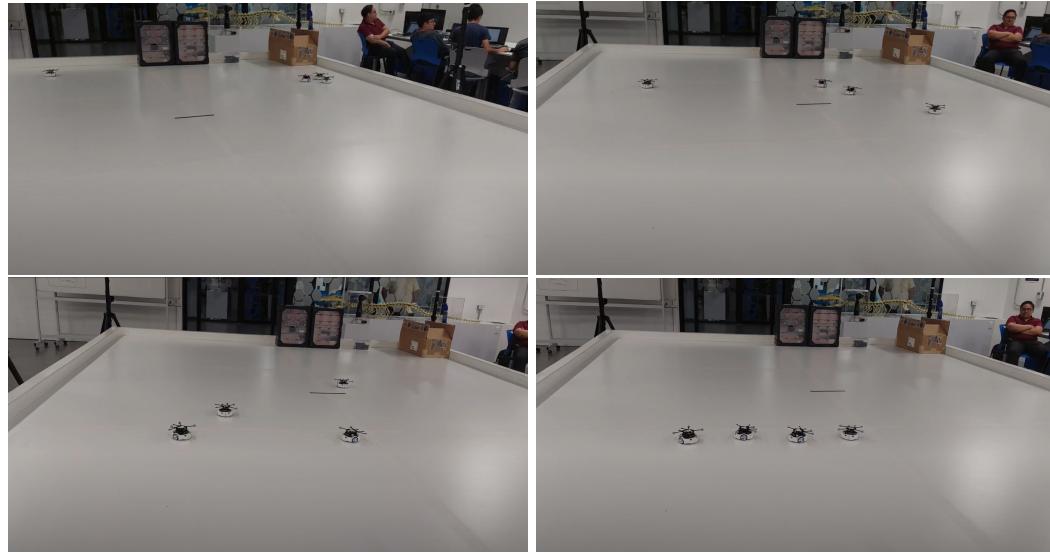


Figura 64: Secuencia Experimento 1: primera fase del algoritmo, posicionamiento de agentes en marcas iniciales.

En la Figura 64 se muestra la primera etapa de ejecución del algoritmo, en el que los agentes se encuentran en posiciones arbitrarias en la mesa. Estos proceden a dirigirse hacia el posicionamiento en forma de línea recta, para sus marcas iniciales de la prueba. Se puede observar que los agentes evaden correctamente el obstáculo virtual localizado en el centro (línea negra en el centro de la mesa), para llegar a su marca inicial. Se uso un nivel de rigidez para mantener la formación con valor 1, es decir, el nivel más bajo de rigidez posible.

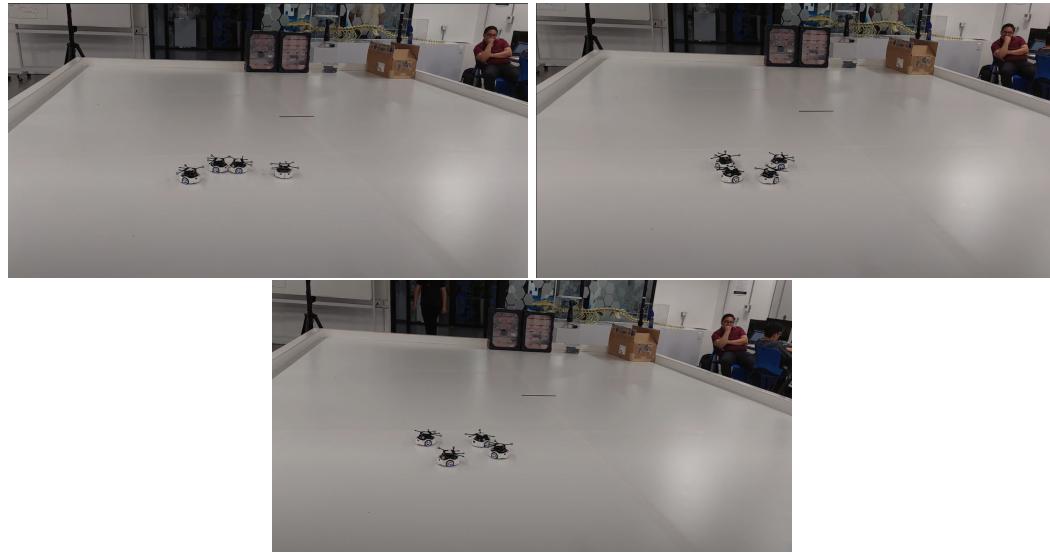


Figura 65: Secuencia Experimento 1: segunda fase del algoritmo, acercamiento de agentes y construcción de formación.

En la Figura 65 se observa la etapa de acercamiento de agentes y construcción de la formación. Primero se juntan entre sí, procurando mantener una distancia entre ellos mismos,

mientras que su velocidad disminuye. Cuando la norma de velocidad cae por debajo del valor de 1, se activa la función de tensión entre agentes, entablando las relaciones de formación. En este caso debido a la cantidad de agentes, siendo 4, se forma una especie de rombo.



Figura 66: Secuencia Experimento 1: tercera fase del algoritmo, movimiento de líder hacia objetivo manteniendo formación.

En la Figura 66, se demuestra la trayectoria seguida por los 4 agentes luego del comienzo de la etapa de movimiento de líder hacia el objetivo. El movimiento del líder hacia el objetivo comienza una vez la norma de velocidades del conjunto de valores cae por debajo de 0.3, lo que indica que los agentes ya se encontraban mayoritariamente en formación. En la secuencia de imágenes se ve claramente como el líder “arrastra” a la formación debido a la tensión con el grupo, mientras se evade el obstáculo virtual del centro. Se puede observar un relativo alejamiento del líder con la formación en comparación a los demás agentes. Este efecto se debe al mismo efecto de “arrastre”, ya que para lograr llevar a los demás agentes al objetivo, es necesario que el factor aplicado sea mayor que el de la formación, provocando el alejamiento ligero. En el último fotograma se observan los agentes ya en el objetivo final con su formación.

Experimento 1.1: Comprobación de la naturaleza dinámica del algoritmo

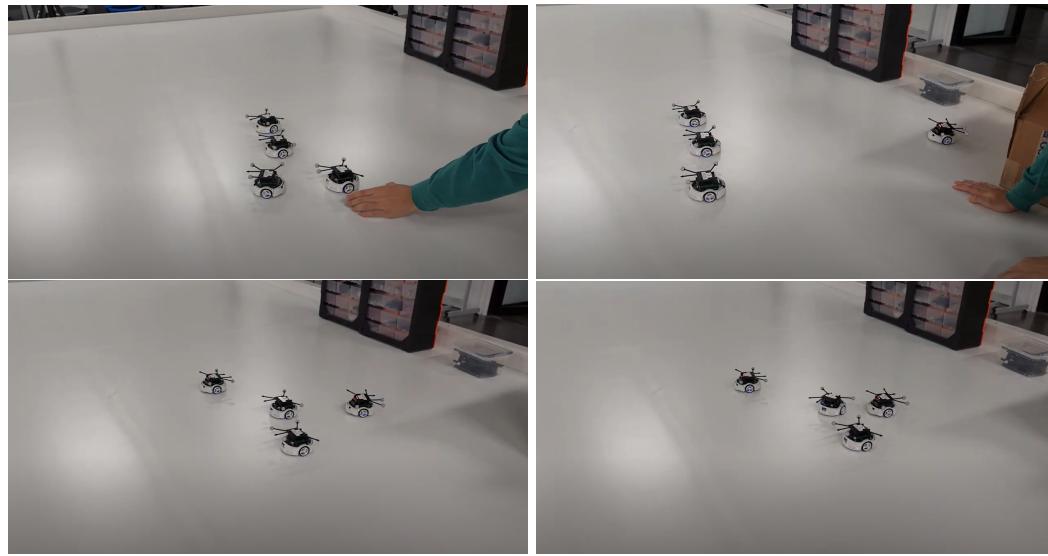


Figura 67: Secuencia Experimento 1.1: Verificación del comportamiento dinámico de los agentes en formación luego de llegar al objetivo.

En la Figura 67 se demuestra el funcionamiento dinámico del algoritmo. Para evidenciarlo, una vez llegó la formación al objetivo final, se movió uno de los agentes lejos de la formación. El resultado fue que el agente regresó a su posición con la formación.

10.1.2. Experimento 2: Reproducibilidad del algoritmo en físico con 4 agentes

Esta prueba consistió en verificar que el algoritmo es replicable y consistente con el mismo número de agentes.



Figura 68: Secuencia Experimento 2: Reproducibilidad de algoritmo en físico con 4 agentes.

En la Figura 68, se evidencia que en efecto el algoritmo es consistente al momento de su ejecución. En esta secuencia se muestran todas las etapas del algoritmo. En los primeros 3 fotogramas se ve el posicionamiento de agentes en marcas iniciales en línea, luego se juntan en el siguiente fotograma y se colocan en formación. En los últimos 5 fotogramas se muestra la trayectoria completa de los agentes hacia el objetivo.

10.1.3. Experimento 3: Visualización de gráficas generadas por la ejecución del algoritmo en físico con 4 agentes con distintas condiciones iniciales de agente pero mismas marcas de inicio.

El Experimento 3 consistió en ejecutar el algoritmo con distintas condiciones iniciales de agente, sin variar las marcas de inicio, para verificar su robustez. A continuación se analizan las gráficas generadas con los datos extraídos de la corrida en el Robotat.

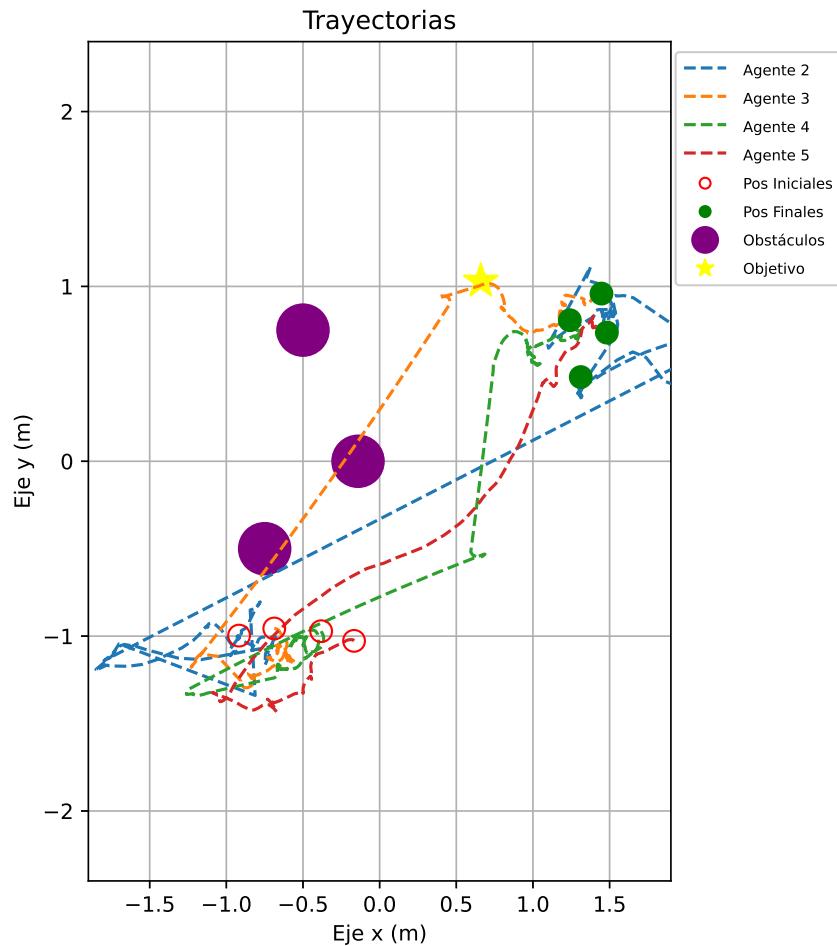


Figura 69: Trayectoria desde marcas iniciales de los 4 agentes, excluyendo el camino hacia las marcas iniciales.

La Figura 69 es la gráfica resultante de las trayectorias a lo largo de la corrida de la prueba, a partir del ciclo 400, donde los agentes ya se encontraban en sus marcas iniciales. Se puede observar que el agente con menor ID es el 2 (trayectoria azul), por lo que le corresponde tomar el liderazgo de la formación. Este presenta el movimiento más directo hacia el objetivo, pues es el menos restringido en cuanto a tensión resultante de la formación.

Además se observa como el agente físico con ID 3 bordea los obstáculos virtuales, con mínima intersección, para seguir en la formación.

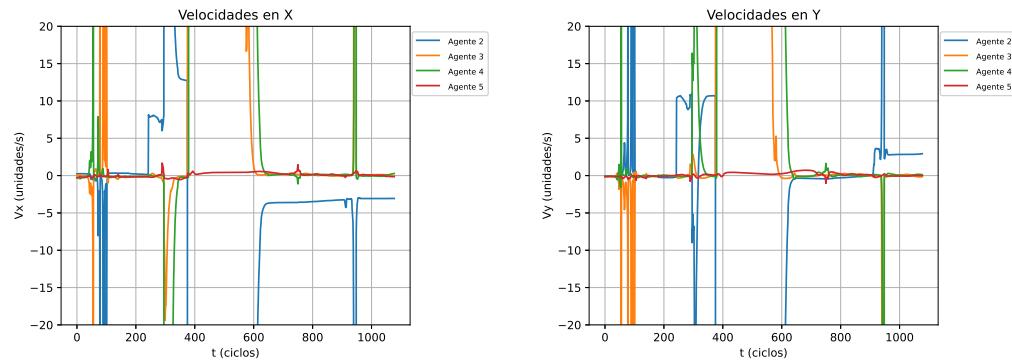


Figura 70: Histórico de velocidades, incluyendo el camino hacia las marcas iniciales.

En la Figura 70 se observa el histórico de velocidades de los robots en X y Y calculadas por el algoritmo. Se puede ver que el valor de velocidad del agente 2 (azul), es el que más cambios presenta, debido a que es el líder.

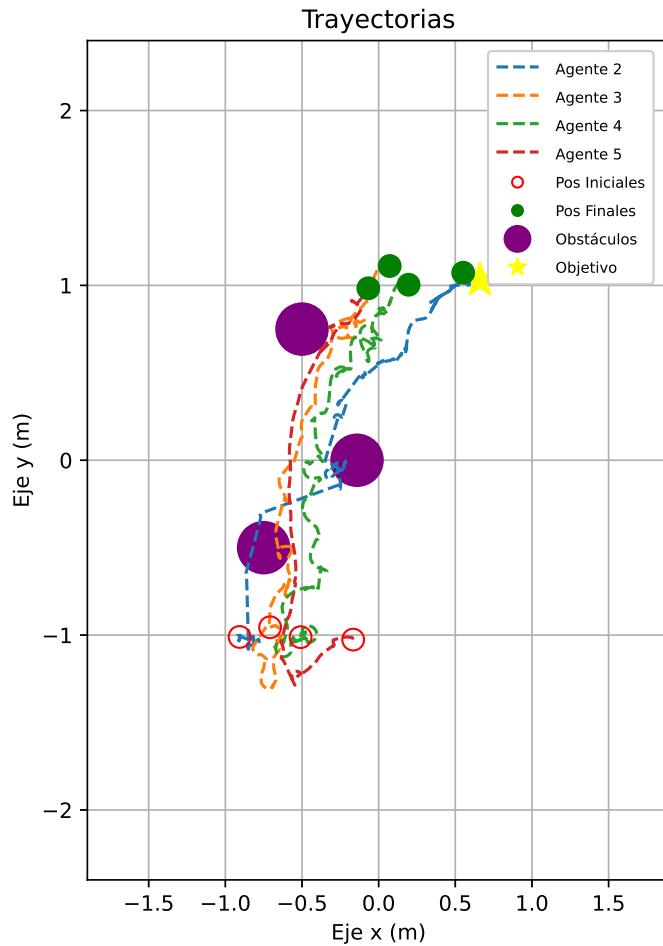


Figura 71: Trayectoria desde marcas iniciales de los 4 agentes, excluyendo el camino hacia las marcas iniciales.

En la Figura 71, se muestra otra corrida del algoritmo, para comparar con la anterior, en la que las condiciones iniciales variaron, pero las marcas permanecieron prácticamente iguales. Aunque en este caso el líder tomó un camino distinto, la formación se mantuvo a lo largo del trayecto, evadiendo correctamente los obstáculos y llegando al objetivo. Con esto se evidencia que el algoritmo es consistente respecto a su ejecución y reproducible en cuanto al éxito de llegada al objetivo manteniendo su funcionamiento.

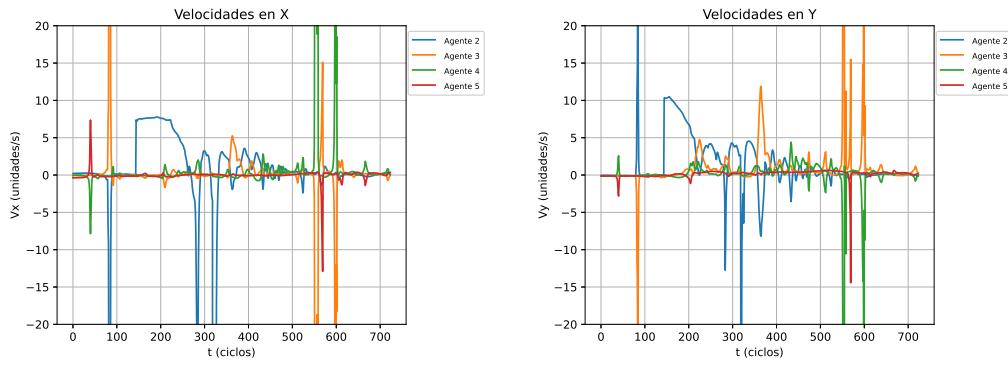


Figura 72: Histórico de velocidades, excluyendo el camino hacia las marcas iniciales.

En la Figura 72 se evidencian los cambios de velocidad más relevantes, de los cuales destaca el del líder. El líder comienza a moverse con mayor magnitud de velocidad una vez la formación se construye.

10.1.4. Experimento 4: Ejecución del algoritmo completo en físico con 5 agentes y generación de gráficas de velocidad en x y y, así como el análisis de la trayectoria de cada agente involucrado.

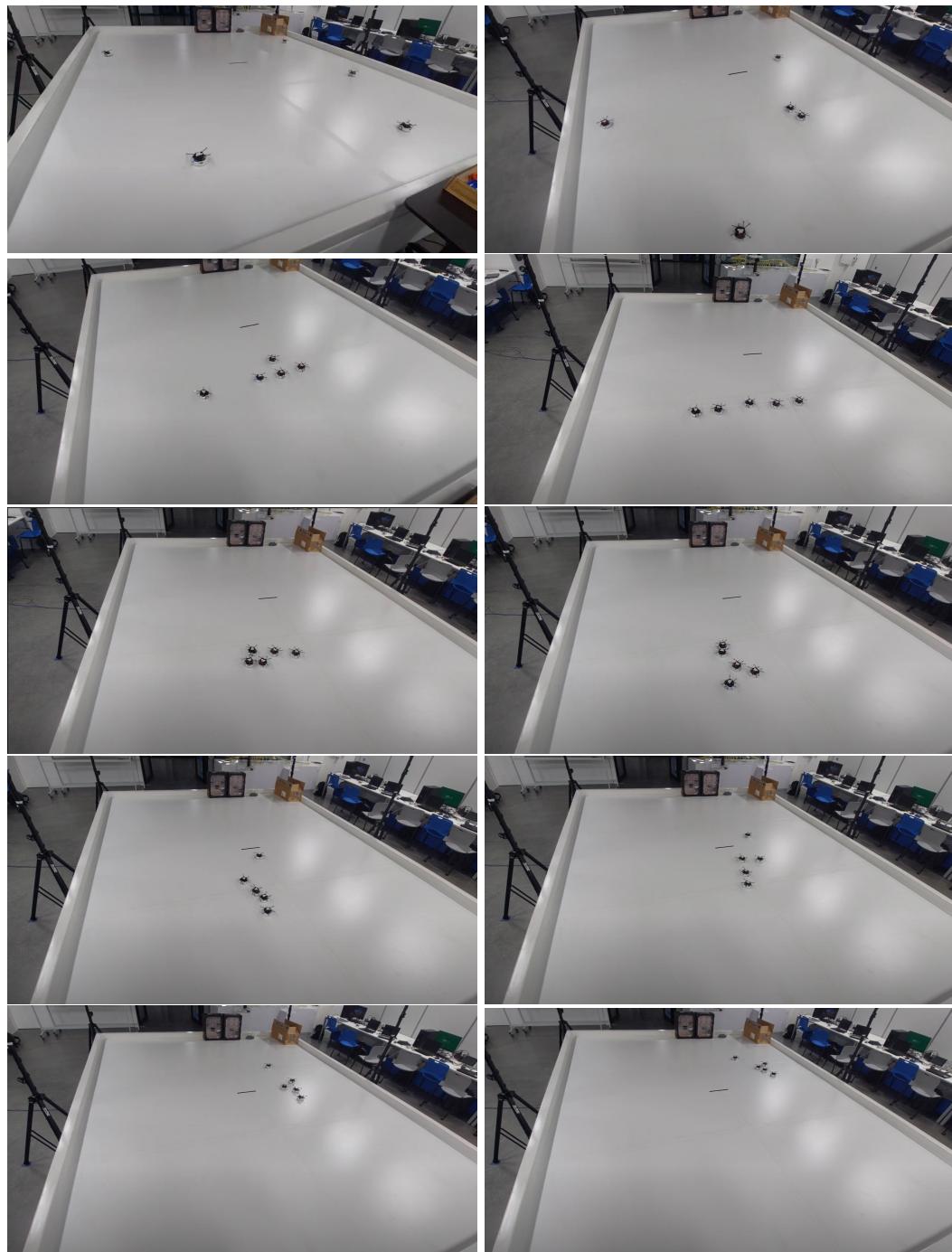


Figura 73: Secuencia Experimento 4: Ejecución del algoritmo completo en físico con 5 agentes.

En la Figura 73 se evidencia el funcionamiento del algoritmo completo (etapa de marcas iniciales, acercamiento y formación; movimiento hacia objetivo). En esta ocasión se probó

con un número distinto de agentes, siendo 5.

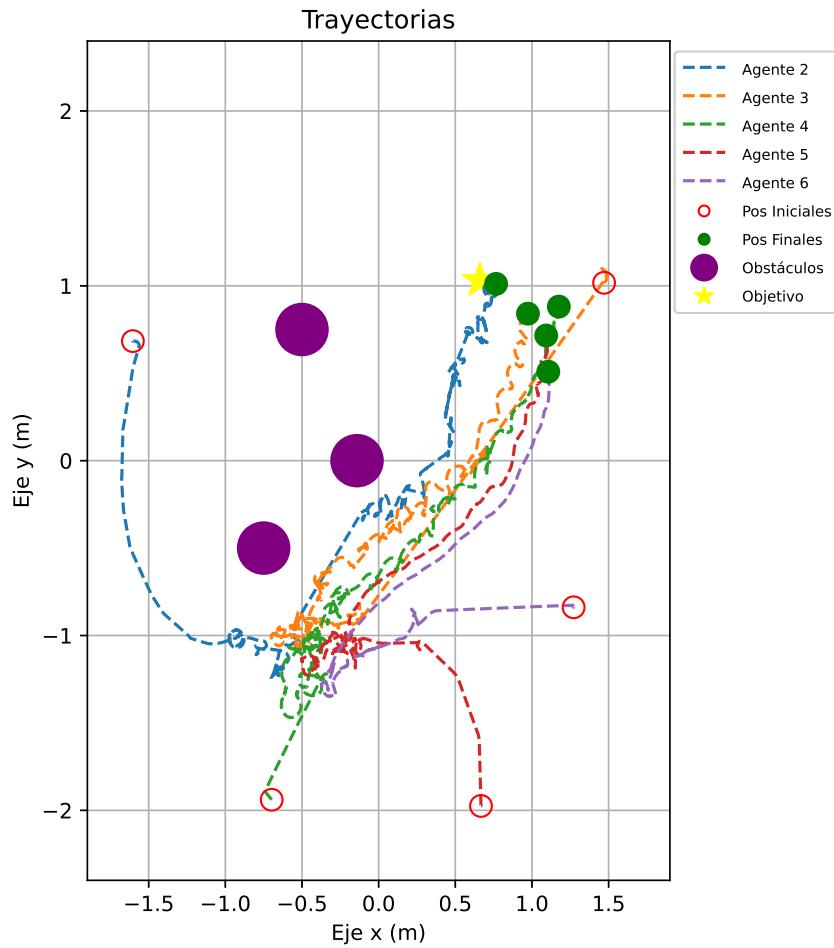


Figura 74: Trayectoria completa de los 5 agentes, incluyendo el camino hacia las marcas iniciales.

En la Figura 74 se logra apreciar la trayectoria completa del algoritmo, en esta ocasión desde sus posiciones arbitrarias. Es claro observar donde se encuentran sus marcas iniciales, pues se nota que existen 5 puntos en los que se tienen varias oscilaciones en círculos. Esto se debe a que una vez llegan a sus marcas iniciales, los agentes esperan a llegar al ciclo 400 para iniciar con la fase de formación. Luego de esas oscilaciones, se puede ver las trayectorias exitosas en formación hasta el objetivo.

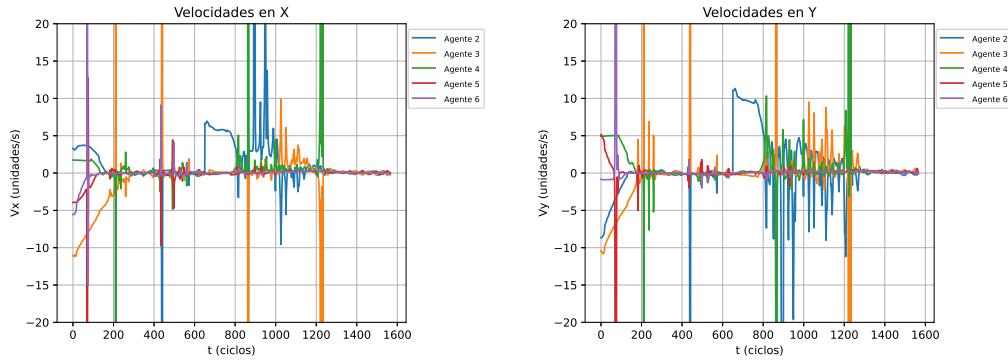


Figura 75: Histórico de velocidades, incluyendo el camino hacia las marcas iniciales.

En la Figura 75 se muestra el histórico de velocidades de toda la trayectoria de formación. Aquí se puede apreciar el comportamiento del camino hacia las marcas iniciales por parte de los agentes desde sus posiciones arbitrarias. Como se mencionó con anterioridad, esto sucede previo al ciclo 400. Se destaca el comportamiento de velocidades que empiezan relativamente altas y convergen a un valor muy pequeño, pues la velocidad está relacionada directamente con la diferencia de distancia.

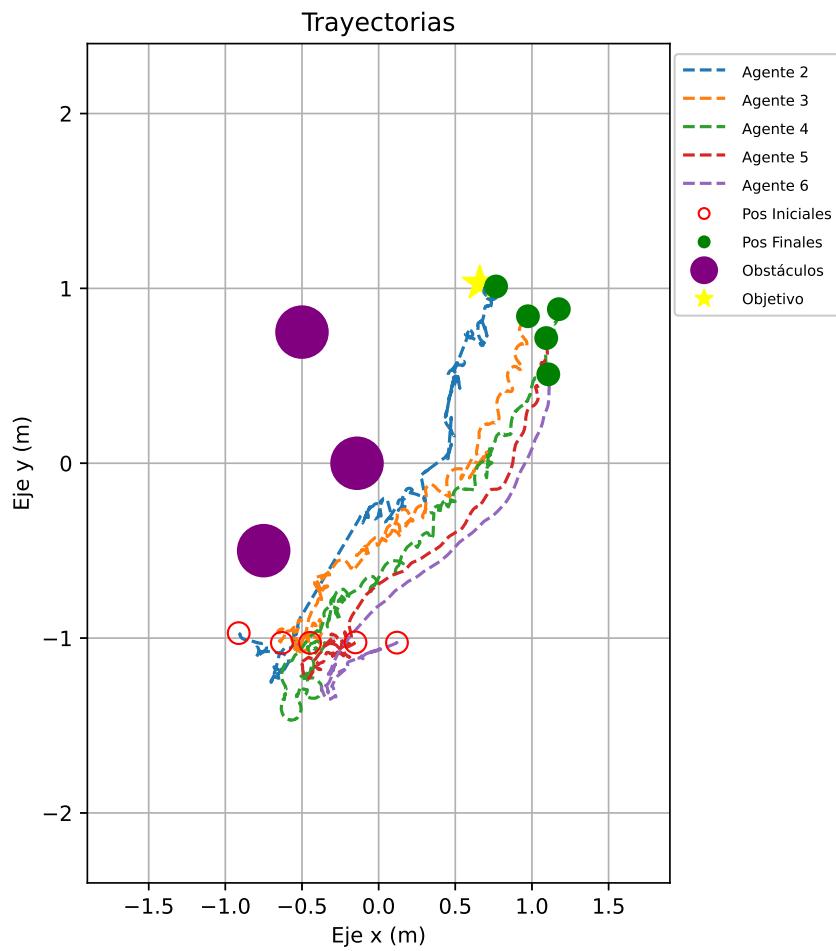


Figura 76: Trayectoria de los 5 agentes a partir de las marcas iniciales.

Para fines de comparación, en la Figura 76 se muestra la misma corrida con 5 agentes, solo que en esta ocasión se omite el recorrido de los agentes de sus posiciones arbitrarias hasta las marcas iniciales.

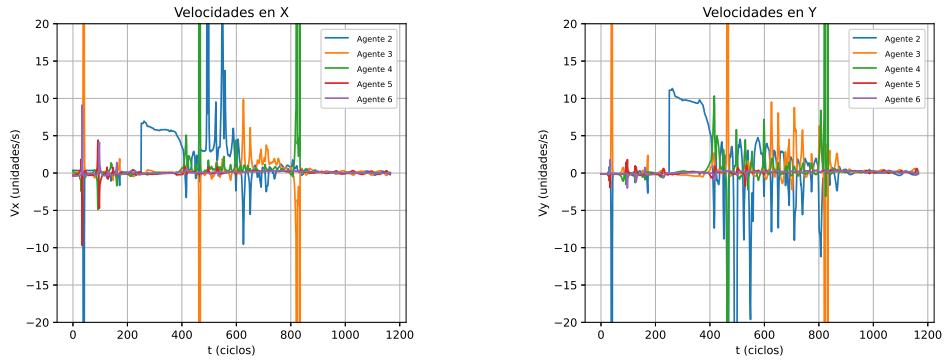


Figura 77: Histórico de velocidades, a partir de las marcas iniciales.

Por último, en la Figura 77 se logra ver el comportamiento de las velocidades de los 5 agentes a lo largo del trayecto mencionando con en la Figura 76.

10.2. Comparación de Resultados

Como producto de los experimentos se logró observar varios hallazgos y comprobaciones relevantes respecto al funcionamiento del algoritmo. El primero de esto fue que el algoritmo es funcional, ejecutando cada etapa con éxito, como se demostró en el experimento 1. Además se estudió su reproducibilidad tanto al variar número de agentes y posiciones iniciales con mismas marcas. El resultado de este estudio fue que en efecto el algoritmo es versátil en cuanto a número de agentes independientemente de sus condiciones iniciales, lo cual está asegurado por el condicionamiento de marcas iniciales para que exista certeza respecto al punto de inicio de interés.

CAPÍTULO 11

Conclusiones

1. Se encontraron mejoras de implementación al momento de restaurar el algoritmo de 2019, con lo que se halló que el algoritmo es adaptable en cuanto al número de agentes a usar, ya que aunque la formación esté diseñada para 10 agentes, esta se mantiene, aunque el número de agentes sea menor a 10. Como se pudo observar en las pruebas realizadas en físico, sólo se utilizaron 4 y 5 agentes por la escasez de robots en la universidad, mas estos se siguieron formando en subsecciones de la formación.
2. Se verificó que la generación de trayectorias es dinámica en el ecosistema real, lo que permite cambios en el entorno y provee mayor adaptabilidad a imprevistos.
3. La ejecución exitosa del algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi- agente en un entorno físico fue comprobada mediante varios experimentos, y se estudió su reproducibilidad, así como robustez cuando está sujeta a condiciones iniciales distintas.
4. El algoritmo evade satisfactoriamente las colisiones con obstáculos y es tiene una calibración adecuada para mantener la distancia entre agentes y alejarse si están demasiado cerca.
5. Se implementó correctamente un sistema para almacenar eficientemente los datos generados para replicar experimentos y optimizar la comparación entre físico y simulado.

CAPÍTULO 12

Recomendaciones

- Se recomienda automatizar la generación de entidades en el mundo de Webots, no solamente su configuración, ya que esto favorecería la escalabilidad de las formaciones, así como los objetos de interés en el mundo.
- Existe la posibilidad de implementar algoritmo para su funcionamiento en simultáneo tanto en físico como en simulado, aunque esto podría ralentizar el cálculo de las trayectorias, por lo que se debe evaluar formas de implementarlo que no sean computacionalmente caras.
- Es una buena idea implementar una interfaz, ya sea gráfica o en una terminal para hacer la selección de configuración de escenario más amigable al usuario, sin tener que modificar el código directamente.
- Estudiar el caso híbrido en el que agentes virtuales simulados mantengan una formación con agentes físicos.
- Usar una computadora adecuada para procesos computacionalmente caros, por la cantidad de programas que se corren en simultáneo. Como mínimo es recomendable tener 16 GB de RAM y un disco SSD de 512 GB, en la UVG se cuenta con equipo que cumple con los requisitos mínimos.

CAPÍTULO 13

Bibliografía

- [1] BostonDynamics, *BostonDynamics*, <https://www.bostondynamics.com/atlas>, Accessed: 2023-04-23.
- [2] P. Muniganti y A. Pujol, “A Survey on Mathematical models of Swarm Robotics,” ene. de 2010.
- [3] L. Bayindir y E. Şahin, “A review of studies in swarm robotics,” *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, n.º 2, págs. 115-147, 2007.
- [4] M. Rubenstein, A. Cornejo y R. Nagpal, “Programmable self-assembly in a thousand-robot swarm,” *Science*, vol. 345, n.º 6198, págs. 795-799, 2014. DOI: 10.1126/science.1254295. eprint: <https://www.science.org/doi/pdf/10.1126/science.1254295>. dirección: <https://www.science.org/doi/abs/10.1126/science.1254295>.
- [5] W. Institute, *Kilobots: A Thousand-Robot Swarm*, <https://wyss.harvard.edu/media-post/kilobots-a-thousand-robot-swarm>, Accessed: 2023-04-23.
- [6] M. Limeira, L. Piardi, V. C. Kalempa, A. S. de Oliveira y P. Leitão, “WsBot: A Tiny, Low-Cost Swarm Robot for Experimentation on Industry 4.0,” *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, págs. 293-298, 2019.
- [7] C. Perafán, “Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica,” Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [8] A. Aguilar, “Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO),” Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [9] A. H. Elsheikh y M. Abd Elaziz, “Review on applications of particle swarm optimization in solar energy systems,” *International Journal of Environmental Science and Technology*, vol. 16, n.º 2, págs. 1159-1170, feb. de 2019, ISSN: 1735-2630. DOI: 10.1007/s13762-018-1970-x. dirección: <https://doi.org/10.1007/s13762-018-1970-x>.
- [10] A. Maas, “Implementación y Validación del Algoritmo de Robótica de Enjambre Particle Swarm Optimization en Sistemas Físicos,” Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.

- [11] R. Lima, "Implementación y validación de algoritmos de robótica de enjambre en plataformas móviles en la nueva mesa de pruebas del laboratorio de robótica de la UVG," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [12] G. Iriarte, "Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2021.
- [13] D. Baldizón, "Aplicaciones Prácticas para Algoritmos de Inteligencia y Robótica de Enjambre," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [14] W. Sierra, "Aplicaciones Prácticas para Algoritmos de Inteligencia y Robótica de Enjambre," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [15] A. M. Peña, "Algoritmo de sincronización y control de sistemas de robots multi-agente para misiones de búsqueda," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.
- [16] A. M. Peña, M. Zea y L. A. Rivera, "Flat Tension Functions and Minimally Rigid Graphs for Tasks of Synchronization and Control of Multi-Agent Robotic Systems," en *2022 IEEE 40th Central America and Panama Convention (CONCAPAN)*, 2022, págs. 1-6. DOI: [10.1109/CONCAPAN48024.2022.9997593](https://doi.org/10.1109/CONCAPAN48024.2022.9997593).
- [17] K. Aldana, "Desarrollo e implementación de algoritmos de control para un enjambre de drones Crazyflie 2.0 mediante un sistema de visión de cámaras OptiTrack," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2019.
- [18] J. Rodríguez, "Diseño e Implementación de una Plataforma Móvil Para Aplicaciones de Robótica de Enjambre - Fase III," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [19] L. Nij, "Evaluación y validación de plataformas móviles para aplicaciones prácticas de robótica," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.
- [20] O. Soysal, E. Bahçeci y E. Şahin, "Aggregation in swarm robotic systems: Evolution and probabilistic control," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, n.º 2, 2007, ISSN: 1300-0632. dirección: <http://search/yayin/detay/67384>.
- [21] A. M. T. Lucca, "Teoría de Grafos (Primera Parte)," *Revista de Educación Matemática*, vol. 15, n.º 1, ago. de 2021. dirección: <https://revistas.unc.edu.ar/index.php/REM/article/view/10926>.
- [22] Wikipedia, *Teoría de grafos*, https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos, Accessed: 2023-05-20.
- [23] F. Chung, L. Lu y V. Vu, "Spectra of random graphs with given expected degrees," en, *Proc Natl Acad Sci U S A*, vol. 100, n.º 11, págs. 6313-6318, mayo de 2003.
- [24] B. Mohar, "On the Laplacian coefficients of acyclic graphs," *Linear Algebra and its Applications*, vol. 422, n.º 2, págs. 736-741, 2007, ISSN: 0024-3795. DOI: <https://doi.org/10.1016/j.laa.2006.12.005>. dirección: <https://www.sciencedirect.com/science/article/pii/S0024379506005325>.
- [25] Hmong, *Matriz laplaciana*, https://hmong.es/wiki/Laplacian_matrix, Accessed: 2023-05-20.

- [26] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu y W. Whiteley, “Planar minimally rigid graphs and pseudo-triangulations,” *Computational Geometry*, vol. 31, n.^o 1, págs. 31-61, 2005, Special Issue on the 19th Annual Symposium on Computational Geometry - SoCG 2003, ISSN: 0925-7721. DOI: <https://doi.org/10.1016/j.comgeo.2004.07.003>. dirección: <https://www.sciencedirect.com/science/article/pii/S0925772104001063>.
- [27] L. Krick, “Application of graph rigidity in formation control of multi-robot networks,” Tesis doctoral, Universidad de Toronto, 2007.
- [28] M. Zea, *MT3005 - Lecture 13 slides*, Notas de clase, Accessed: 2023-05-21.
- [29] M. Rebollo, “Generalización de procesos de consenso en redes complejas,” Tesis doctoral, Universidad Politécnica de Madrid, 2019.
- [30] OptiTrack, *OptiTrack*, <https://optitrack.com/>, Accessed: 2023-05-21.
- [31] MathWorks, *Matlab*, <https://la.mathworks.com/products/matlab.html>, Accessed: 2023-05-21.
- [32] Cyberbotics, *Cyberbotics: Robotics simulation with Webots*, <https://cyberbotics.com/>, Accessed: 2023-05-21.
- [33] Pololu, *3pi+ 32U4 OLED Robot*, <https://www.pololu.com/category/280/3pi-plus-32u4-oled-robot>, Accessed: 2023-05-21.
- [34] *ESP32 Series*, ESP32, v4.2, Espressif Systems, mayo de 2023.
- [35] L. M. Engineers, *ESP32 Pinout Reference*, <https://lastminuteengineers.com/esp32-pinout-reference/>, Accessed: 2023-05-21.

CAPÍTULO 14

Anexos

Enlace a repositorio en GitHub: <https://github.com/rod19131/tesisAlejandro>



Figura 78: Perspectiva adicional 1 de la adquisición de datos para la calibración de los marcadores.



Figura 79: Perspectiva adicional 2 de la adquisición de datos para la calibración de los marcadores.

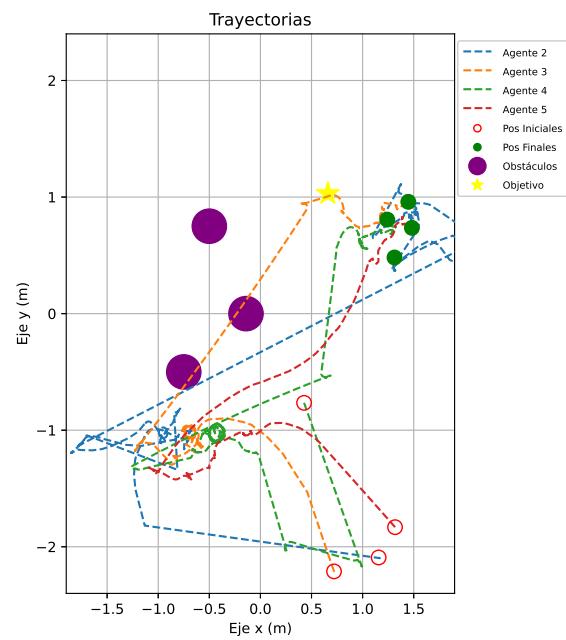


Figura 80: Trayectoria desde marcas iniciales de los 4 agentes del experimento 1, incluyendo el camino hacia las marcas iniciales.

