
Optimización de un algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente para escenarios con obstáculos móviles

Gerardo Paz Fuentes



UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería



**Optimización de un algoritmo de inteligencia de enjambre
enfocado en sincronización y control de formaciones de
sistemas robóticos multi-agente para escenarios con obstáculos
móviles**

Trabajo de graduación presentado por Gerardo Paz Fuentes para optar
al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____
Dr. Luis Rivera

Tribunal Examinador:

(f) _____
Dr. Luis Rivera

(f) _____
Examinador 1

(f) _____
Examinador 2

Fecha de aprobación: Guatemala, 5 de diciembre de 2024.

Prefacio

Índice

Prefacio	III
Lista de figuras	VIII
Lista de cuadros	IX
Resumen	X
Abstract	XI
1. Introducción	1
2. Antecedentes	2
2.1. Robótica de enjambre	2
2.2. Robótica de enjambre inspirada en peces	2
2.3. Robotarium de Georgia Tech	3
2.4. Robotat de la Universidad del Valle de Guatemala	3
2.5. Validación de algoritmos de algoritmos de <i>Particle Swarm Optimization</i> (PSO) y <i>Ant Colony Optimization</i> (ACO)	4
2.6. Algoritmo de sincronización y control de sistemas de robots multiagente para misiones de búsqueda	5
2.7. Validación de un algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente en un entorno físico	5
3. Justificación	6
4. Objetivos	7
4.1. Objetivo general	7
4.2. Objetivos específicos	7
5. Alcance	8

6. Marco teórico	10
6.1. Conceptos fundamentales en robótica de enjambre	10
6.1.1. Enjambre	10
6.1.2. Agente	11
6.1.3. Formaciones	11
6.1.4. Control centralizado y descentralizado	11
6.2. Conceptos básicos en Teoría de grafos	12
6.2.1. Teoría de grafos	12
6.2.2. Vértices y aristas	13
6.2.3. Tipos de grafos	13
6.2.4. Matrices asociadas a un grafo	14
6.3. Control de la formación	15
6.3.1. Grafo de formación	15
6.3.2. Construcción de un grafo mínimamente rígido	16
6.3.3. Grafo para la red de comunicación	16
6.4. Teoría de control	17
6.4.1. Control de formación	17
6.5. Cinemática de robots diferenciales con ruedas	17
6.6. Otras ecuaciones matemáticas relevantes	18
6.6.1. Ecuación de consenso	18
6.6.2. Funciones racionales para hallar la tensión	19
6.7. Herramientas de software	20
6.7.1. Matlab	20
6.7.2. Entorno de simulación Webots	20
6.7.3. Paralelismo computacional	20
6.8. Ecosistema Robotat	21
6.8.1. Mesa de pruebas	21
6.8.2. Sistema de captura de movimiento OptiTrack	22
6.8.3. Comunicación del Robotat	22
6.9. Hardware	23
6.9.1. Plataforma móvil Pololu 3pi+ modificado	23
7. Algoritmo de sincronización y control de formaciones	24
7.0.1. Controlador del supervisor	24
7.0.2. Controlador de los agentes	25
7.0.3. Funcionamiento del algoritmo	25
8. Restauración del algoritmo desarrollado en fases previas	28
8.1. Replicar simulaciones de Webots	28
8.1.1. Comunicación entre supervisor y agentes	29
8.1.2. Prueba del algoritmo con simulaciones basadas en escenarios previos .	29
8.2. Replicar el funcionamiento del algoritmo en el Robotat	33
8.2.1. Pruebas de conexión con el Pololu 3Pi+ y el Robotat	33
8.2.2. Calibración de marcadores	34
8.2.3. Selección de marcadores a utilizar	38
8.2.4. Ajuste de parámetros en el algoritmo	39
8.2.5. Configuraciones para el escenario	42
8.2.6. Prueba del algoritmo en físico con escenarios previos	42

8.3.	Prueba de funcionamiento del algoritmo con obstáculos móviles	52
8.3.1.	Simulaciones utilizando obstáculos móviles	53
8.3.2.	Escenarios físicos utilizando obstáculos móviles	55
9.	Optimización del algoritmo y su implementación	60
9.1.	Lenguaje de programación y limpieza de código	60
9.2.	Posición de cada agente según el grafo de formación	61
9.3.	Mejorar la eficiencia computacional con NumPy	63
9.3.1.	Aplicar desfases de los marcadores	63
9.3.2.	Cálculo de la distancia entre agentes	65
9.3.3.	Cálculo del error de formación	67
9.4.	Ajuste de parámetros del algoritmo	68
10.	Validación del algoritmo optimizado en escenarios físicos con obstáculos móviles	69
11.	Conclusiones	70
12.	Recomendaciones	71
13.	Bibliografía	72

Lista de figuras

1.	Prototipo físico de Bluebot[3].	3
2.	Laboratorio Robotarium de Georgia Tech [4].	3
3.	Plataforma Robotat de la Universidad del Valle de Guatemala [6].	4
4.	Simulación en Webots implementando el algoritmo para evadir obstáculos [6].	5
5.	Ejemplo de enjambre de robots [10].	10
6.	Ejemplo de un control centralizado y descentralizado [12].	12
7.	Ejemplo de un grafo [8].	12
8.	Ejemplo de algunos tipos de grafos. a: completo, b: simple, c: árbol, d: dígrafo, e: estrella, f: no conexo, g: de celosía, h: de mundo pequeño [14].	14
9.	Grafos de formación en triángulo y hexágono [8].	16
10.	Sistema de control en lazo cerrado [19].	17
11.	Modelo de uniciclo [20].	18
12.	Entorno de simulación en Webots [22].	20
13.	Ejemplo de dependencia de tareas [23].	21
14.	Cámara de captura de movimiento Prime ^x 41 de OptiTrack [24].	22
15.	Sensores y componentes del Pololu 3pi+ 32U4 OLED Robot [25].	23
16.	Diagrama de flujo para el supervisor.	26
17.	Diagrama de flujo para el programa de los agentes.	27
18.	Ejecución de la primera simulación en Webots 2023b.	29
19.	Ejecución del algoritmo en la primera simulación	31
20.	Ejecución del algoritmo en la segunda simulación	32
21.	Ejecución del algoritmo en la tercera simulación	33
22.	Error de conexión con Pololu 3Pi+.	34
23.	Marcadores del OptiTrack disponibles para su uso.	35
24.	Marcadores alineados sobre el eje <i>y</i> de la mesa de pruebas del Robotat.	35
25.	Problema de funcionamiento en físico, agentes permanecen inmóviles.	38
26.	Selección de marcadores para cada agente antes y después de las modificaciones.	39
27.	Problema de funcionamiento en físico, agentes divergen hacia posiciones iniciales.	40

28.	Problema de funcionamiento en físico, agentes divergen luego de colocarse en sus posiciones iniciales.	40
29.	Primera prueba exitosa de funcionamiento en físico.	41
30.	Mesa de pruebas con 2 agentes en el escenario AAA, corrida 1, en físico.	43
31.	Trayectoria de los 2 agentes en el escenario AAA, corrida 1, en físico.	44
32.	Mesa de pruebas con 6 agentes en el escenario AAA, corrida 1, en físico.	45
33.	Trayectoria de los 6 agentes en el escenario AAA, corrida 1, en físico.	46
34.	Mesa de pruebas con 6 agentes en el escenario ACA, corrida 1, en físico.	47
35.	Trayectoria de los 6 agentes en el escenario ACA, corrida 1, en físico.	48
36.	Mesa de pruebas con 3 agentes en el escenario ACC, corrida 1, en físico.	49
37.	Trayectoria de los 3 agentes en el escenario ACC, corrida 1, en físico.	50
38.	Mesa de pruebas con 3 agentes en el escenario ACA, corrida 1, en físico.	51
39.	Trayectoria de los 3 agentes en el escenario ACA, corrida 1, en físico.	52
40.	Trayectoria de los 6 agentes en el escenario ADA, corrida 1, en simulación. . .	53
41.	Trayectoria de los 6 agentes en el escenario ADA, corrida 2, en simulación. . .	54
42.	Trayectoria de los 6 agentes en el escenario ADA, corrida 3, en simulación. . .	55
43.	Trayectoria de los 3 agentes en el escenario ADB, corrida 1, en físico.	56
44.	Trayectoria de los 3 agentes en el escenario ADC, corrida 1, en físico.	57
45.	Trayectoria de los 3 agentes en el escenario ADC, corrida 3, en físico.	58
46.	Trayectoria de los 3 agentes en el escenario ADC, corrida 2, en físico.	59
47.	Posiciones de agentes según el grafo de formación con la implementación original del algoritmo.	62
48.	Posiciones de agentes según el grafo de formación con la implementación original del algoritmo.	62
49.	Gráfica con medias de tiempo para aplicar desfases de marcadores con ciclos <i>for</i> y operaciones con NumPy.	65
50.	Gráfica con medias de tiempo para calcular la distancia entre agentes con ciclos <i>for</i> y operaciones con NumPy.	66
51.	Gráfica con medias de tiempo para calcular el error de formación con ciclos <i>for</i> y operaciones con NumPy.	68

Lista de cuadros

1.	Desfases de marcadores disponibles alineados con el eye <i>y</i> de la mesa de pruebas del Robotat	36
2.	Desfases de marcadores obtenidos en la fase previa por José Alejandro Rodríguez [9].	37
3.	Desfases finales para calibración de marcadores del 1 al 22.	37
4.	Configuraciones para el escenario.	42
5.	Ventajas y desventajas de lenguajes de programación.	61
6.	Media de tiempo y desviación estándar para aplicar desfases de marcadores con ciclos <i>for</i>	64
7.	Media de tiempo y desviación estándar para aplicar desfases de marcadores con NumPy.	64
8.	Media de tiempo y desviación estándar para calcular la distancia entre agentes con ciclos <i>for</i>	66
9.	Media de tiempo y desviación estándar para calcular la distancia entre agentes con NumPy.	66
10.	Media de tiempo y desviación estándar para calcular el error de formación con ciclos <i>for</i>	67
11.	Media de tiempo y desviación estándar para calcular el error de formación con NumPy.	67

Resumen

Abstract

CAPÍTULO 1

Introducción

INTRODUCCIÓN

CAPÍTULO 2

Antecedentes

2.1. Robótica de enjambre

La robótica de enjambre consiste en el uso de robots relativamente sencillos que, al organizarse y funcionar en conjunto, pueden llevar a cabo tareas complejas que un solo robot no podría realizar. Esto proporciona soluciones flexibles, optimizadas y de un menor costo. Además, no se requiere un número específico de agentes robóticos ya que pueden ir desde las dos unidades hasta miles de ellas [1].

Algunas de las aplicaciones de la robótica de enjambre son el control de tráfico, realizar formaciones en movimiento, misiones de búsqueda y rescate, mapeo de entornos, simulación de comportamientos biológicos, exploración de zonas y comunicación de rutas.

Estas últimas dos han sido estudiadas a profundidad para su implementación en la cosecha de la fresa donde el enjambre realiza la exploración de una zona de cultivo, analiza el estado de los frutos por medio de imágenes computacionales para determinar el momento óptimo para su cosecha y comunica los datos procesados para realizar una cosecha automatizada [2].

2.2. Robótica de enjambre inspirada en peces

En 2021 un equipo de investigadores del instituto Wyss de Harvard y la Escuela de Ingeniería y Ciencias Aplicadas John A. Paulson (SEAS) desarrollaron un robot llamado Bluebot inspirado en un pez. Este cuenta con dos cámaras y tres luces LED para su sistema de visión guiado.

Los investigadores模拟aron una misión de búsqueda con una luz roja intermitente e implementaron un algoritmo de dispersión. Para iniciar, los Bluebots se dispersaron en el tanque y una vez que un robot detectó la luz roja, sus leds comenzaron a parpadear, lo que

activó el algoritmo de agregación al resto de robots que lograron sincronizar sus movimientos y agruparse al rededor del robot que detectó la luz, imitando a un banco de peces real [3].

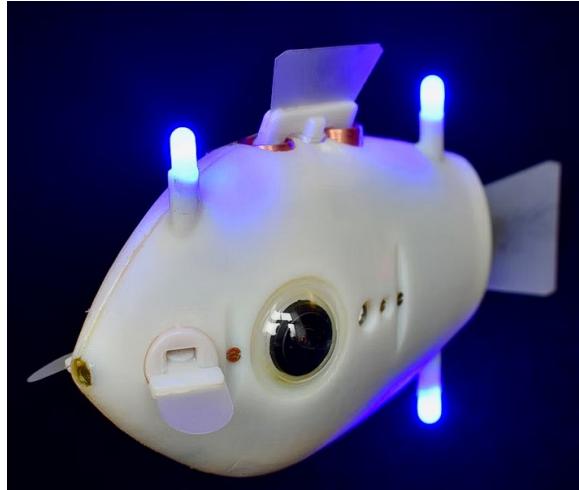


Figura 1: Prototipo físico de Bluebot[3].

2.3. Robotarium de Georgia Tech

En el Instituto Tecnológico de Georgia se ha desarrollado el proyecto Robotarium [4]. Se realizó para proveer acceso gratuito a una plataforma de robótica de enjambre a la que pueda acceder cualquier persona del mundo. Lo único que se necesita para experimentar con la plataforma es descargar un simulador en Matlab o Python, registrarse en la página de Robotarium y esperar la aprobación por parte de un administrador para realizar el experimento.



Figura 2: Laboratorio Robotarium de Georgia Tech [4].

2.4. Robotat de la Universidad del Valle de Guatemala

En el Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala, se encuentra una plataforma de robótica para experimentación llamada Robotat la cual está inspirada en el Robotarium del Instituto Tecnológico de Georgia. Esta se conforma de una plataforma de acero y plycem con un espacio útil de $5 \times 5 \times 3$ m, capaz de soportar

cargas puntuales de hasta dos toneladas. También cuenta con un sistema de captura de movimiento de OptiTrack, compuesto de seis cámaras Prime^x 41 de alta precisión y baja latencia para realizar experimentos en tiempo real. Además, el sistema funciona con una red local inalámbrica WiFi a través de la cual se realiza la comunicación entre robots [5].

Para el funcionamiento del sistema OptiTrack, se utilizan “marcadores” que son figuras plásticas con reflectivos. Estos permiten reflejar la luz infrarroja que emiten las cámaras para obtener, de manera precisa, la posición y el movimiento de objetos en un espacio tridimensional.



Figura 3: Plataforma Robotat de la Universidad del Valle de Guatemala [6].

2.5. Validación de algoritmos de algoritmos de *Particle Swarm Optimization (PSO)* y *Ant Colony Optimization (ACO)*

En la Universidad del Valle de Guatemala, la tesis de Jonathan Menéndez [7] se enfocó en realizar pruebas físicas para validar los algoritmos de robótica de enjambre PSO y ACO utilizando los robots móviles Pololu 3pi+ en la plataforma de Robotat.

Luego de realizar múltiples pruebas físicas, se encontró que el algoritmo de ACO es capaz de generar trayectorias de manera satisfactoria en el ecosistema del Robotat, respetando las limitaciones físicas de espacio en la mesa para no colisionar y evitar que las cámaras de OptiTrack pierdan la detección de los agentes. Además, este algoritmo demostró tener una mayor eficiencia al corregir las diferencias entre el nodo de inicio y la posición inicial del robot al orientarlo hacia el punto de inicio.

Dichos estudios se limitaron al espacio disponible en la mesa de la plataforma del Robotat, a un espacio libre de obstáculos y a la implementación de solo diez Pololu 3pi+ debido a la disponibilidad de equipo.

2.6. Algoritmo de sincronización y control de sistemas de robots multiagente para misiones de búsqueda

El trabajo de investigación de Andrea Maybell Peña [8] se basó en utilizar un sistema de robots multi-agente para realizar misiones de búsqueda. El algoritmo se basa en la teoría de grafos y el control moderno para tener formaciones específicas de los agentes que permiten su movilización a través de obstáculos que se limitaron a una geometría toroidal y la cantidad de agentes robóticos se limitó a diez unidades del modelo E-Puck de GCTronic.

Se realizó la implementación del algoritmo en el simulador Webots de Cyberbotics. Como resultado se obtuvo una tasa de éxito del 80 % utilizando el algoritmo completo para llegar a la meta.



Figura 4: Simulación en Webots implementando el algoritmo para evadir obstáculos [6].

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

En el trabajo de investigación de Alejandro Rodríguez [9] se realizó una validación del algoritmo de inteligencia de enjambre enfocado a sincronización vertical y control de formaciones de sistemas robóticos multi-agente. La validación se realizó utilizando el ecosistema del Robotat de la Universidad del Valle de Guatemala y los robots Pololu 3pi+ modificados.

Para la validación física, se realizaron diversos experimentos donde se evaluó el desempeño de cada agente, la generación de trayectorias, el posicionamiento de los agentes, las distintas configuraciones de formación y los escenarios utilizando obstáculos.

Los experimentos realizados se limitaron a utilizar un máximo de nueve agentes robóticos debido a su disponibilidad en la universidad, el tiempo máximo por semana para realizar pruebas en el Robotat fue de seis horas y estas se realizaron utilizando obstáculos estáticos.

Los resultados de la experimentación demostraron que, el algoritmo evade los obstáculos satisfactoriamente y mantiene una distancia adecuada entre agentes. Sin embargo, el algoritmo en simulación es aproximadamente un 70 % más rápido que en físico. Esto se debe a que cada prueba física tomaba alrededor de 8 minutos debido a la latencia del servidor que aumentaba con el número de agentes conectados.

CAPÍTULO 3

Justificación

La inteligencia de enjambre es un campo reciente en el área de inteligencia artificial. Sus aplicaciones abarcan desde realizar tareas cotidianas hasta resolver problemas complejos. Uno de sus mayores beneficios es utilizar agentes robóticos de bajo costo para ejecutar acciones en conjunto en lugar de utilizar un solo robot complejo. Además, esto crea un sistema robótico robusto lo que resulta útil para diversas aplicaciones.

Se decidió estudiar la robótica de enjambre ya que en la Universidad del Valle de Guatemala, en trabajos anteriores, se han realizado pruebas implementando algoritmos de sincronización y control utilizando simuladores como Webots, así como agentes físicos Pololu 3Pi+. Estas pruebas tuvieron éxito, sin embargo, estuvieron limitadas a generar trayectorias en un ambiente con obstáculos fijos y los tiempos de ejecución eran muy largos. Por lo tanto, en este trabajo de graduación se desea optimizar el algoritmo desarrollado previamente y realizar nuevas pruebas para validar las trayectorias generadas en un ambiente controlado con obstáculos móviles.

Esto permitirá conocer el alcance de los algoritmos de sincronización y control, además que abrirá las puertas a su implementación en aplicaciones de la vida real como misiones de búsqueda y rescate o análisis de zonas de cultivos, donde en los escenarios se encontrarán obstáculos móviles como personas, animales o incluso cambios en el propio escenario debido a factores externos.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Optimizar la implementación del algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente desarrollado anteriormente, y validarlo en escenarios con obstáculos móviles, en el ecosistema Robotat.

4.2. Objetivos específicos

- Evaluar la implementación del algoritmo de sincronización y control desarrollado anteriormente e identificar deficiencias y puntos de mejora en código, lenguaje de programación y métodos de comunicación.
- Optimizar el algoritmo tomando en cuenta los puntos de mejora identificados y evaluar su rendimiento en escenarios similares a los utilizados anteriormente.
- Adaptar el algoritmo para generar trayectorias en escenarios con obstáculos móviles.
- Validar el rendimiento del algoritmo optimizado con agentes robóticos como el Pololu 3Pi+ en el ecosistema Robotat.

CAPÍTULO 5

Alcance

Con la creación del algoritmo de sincronización y control de formaciones se abrió paso a experimentar con él en diferentes escenarios. Además, la previa adaptación del algoritmo para utilizarlo tanto en simulaciones como en escenarios reales en el Robotat, permitió tener la infraestructura necesaria para seguir evaluando el alcance de este utilizando las plataformas robóticas como el Pololu 3Pi+.

El algoritmo funciona en un ambiente cambiante gracias a la forma en que se planteó originalmente la ecuación de consenso y el cálculo del peso. Así como que en un ambiente real, se evalúa constantemente la posición actual de cada agente y se compara con la posición actual de los obstáculos para ajustar la ecuación de consenso a manera de evitar posibles colisiones.

Actualmente, se puede utilizar formaciones con un máximo de 10 agentes. Esto es debido a que el grafo de formación que utiliza el algoritmo, se planteó a manera de que fuera mínimamente rígido para únicamente 10 agentes. Sin embargo, en futuras implementaciones este se podría modificar para crear formaciones con más agentes.

El algoritmo está completamente desarrollado en Python que es un lenguaje de código abierto (*open source*). Esto permite la facilidad de desarrollo para futuros investigadores, además que cuenta con librerías para mejorar la eficiencia computacional.

En este proyecto, se optimizó el algoritmo aplicando operaciones matriciales en lugar de bucles con Python para reducir el tiempo de procesamiento computacional. Esto permitió disminuir el tiempo que toma cada ciclo de ejecución del algoritmo, que es más notorio conforme se aumenta el número de agentes en la formación. Además, se modificó el código de procesamiento de datos que genera las gráficas y trayectorias para mostrar el comportamiento de los obstáculos móviles.

Una de las limitantes al trabajar en la optimización del algoritmo y realizar las pruebas en físico, fue que solo se cuenta con 8 agentes robóticos disponibles. Esto se debe a la disponibilidad de robots en la UVG y a que otros estudiantes los utilizan para sus proyectos.

Además, el tiempo máximo para realizar pruebas en el Robotat fue limitado ya que fue necesario compartir la mesa de pruebas y los agentes robóticos con otros estudiantes que también los utilizaban, requiriendo todo el espacio disponible.

CAPÍTULO 6

Marco teórico

6.1. Conceptos fundamentales en robótica de enjambre

6.1.1. Enjambre

En la robótica, un enjambre es el conjunto de individuos que colaboran entre sí para lograr un objetivo. En la Figura 5 se muestra un enjambre de robots.

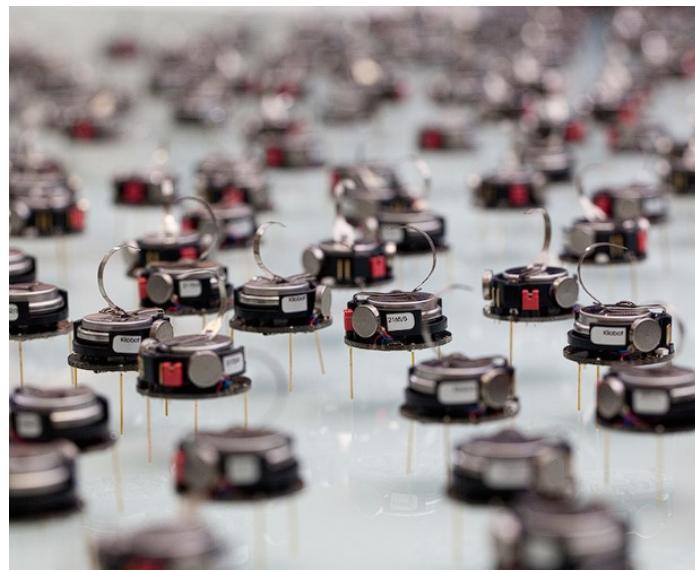


Figura 5: Ejemplo de enjambre de robots [10].

6.1.2. Agente

Se le conoce como agente a un individuo que forma parte del enjambre. Este es capaz de realizar acciones simples para lograr un objetivo de forma colaborativa con los demás individuos. En la robótica de enjambre, los agentes son robots [11].

6.1.3. Formaciones

Es común que en la naturaleza se observen enjambres que forman patrones. Esto es gracias a que los individuos realizan movimientos coordinados para crear formaciones. En la robótica de enjambre sucede lo mismo, se tiene un conjunto de agentes que se organizan y coordinan sus movimientos para lograr objetivos. Estos pueden ser evadir obstáculos, mapear un entorno, llegar a un objetivo siguiendo una trayectoria, entre otros. Para lograr la coordinación de formaciones, se requiere aplicar un control que puede ser centralizado o descentralizado [11].

6.1.4. Control centralizado y descentralizado

En la robótica de enjambre, el control centralizado consiste en una red de comunicación donde una unidad central de procesamiento (CPU) tiene comunicación con cada agente para transmitir y recibir información. Asimismo, cada agente se comunica únicamente con el CPU.

El control descentralizado, también llamado control distribuido, es donde cada agente tiene comunicación con los demás para transmitir información. Este tipo de control requiere un protocolo de comunicación más robusto y complejo, sin necesidad de un CPU [12].

En la Figura 6 se observa la diferencia entre ambos tipos de control.

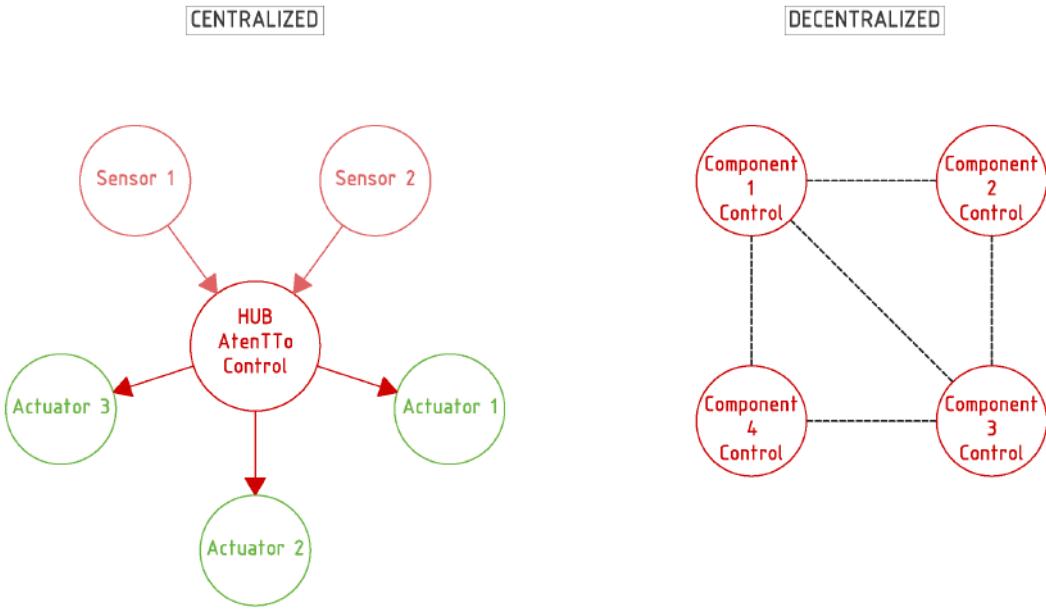


Figura 6: Ejemplo de un control centralizado y descentralizado [12].

6.2. Conceptos básicos en Teoría de grafos

6.2.1. Teoría de grafos

La teoría de grafos es una rama de la matemática y ciencias de la computación que se dedica a estudiar los grafos [13]. El grafo es un conjunto de vértices conectados por aristas tal como se observa en la Figura 7. Esta teoría se utiliza en diversas aplicaciones como mapeo de entornos, generación de rutas, análisis de datos, telecomunicaciones, entre otras. En este trabajo de graduación, se utilizó la teoría de grafos para definir formaciones y la red de comunicación entre agentes.

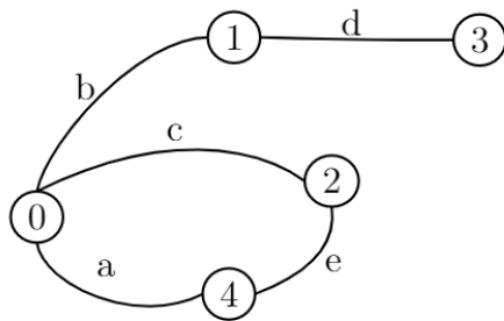


Figura 7: Ejemplo de un grafo [8].

6.2.2. Vértices y aristas

Los vértices, también llamados nodos, son los puntos donde se conectan las aristas de un grafo. El grado de un vértice será el número de aristas asociados a él [13].

Las aristas, también llamadas arcos o enlaces, son las líneas que conectan un par de vértices y se clasifican en [13]:

- Lazo: Es una arista que tiene en sus extremos el mismo vértice.
- Paralelas o múltiples: Es cuando dos o más aristas tienen en los extremos el mismo par de vértices.

6.2.3. Tipos de grafos

En un grafo, los vértices se puede conectar de distintas maneras según la aplicación que se requiera tal como se observa en la Figura 8. Algunas de sus clasificaciones son [14]:

- Simple o anillo: Cada par de nodos se conecta por una sola arista.
- Multigrafo: En cada par de nodos puede haber más de un enlace.
- Árbol: Es un grafo que no tiene ciclos.
- Dirigidos o dígrafos: Las aristas entre los nodos tienen dirección.
- Estrella: Tiene un nodo central que conecta con los demás nodos.
- No conexos: Hay uno o más nodos que no conectan con el resto.
- Completo: Cada nodo está conectado con los demás.
- Regular: Todos los nodos tienen el mismo número de aristas.
- De celosía y mundo pequeño: Estos grafos se utilizan para entender el comportamiento y estructura de ciertos fenómenos en la naturaleza.

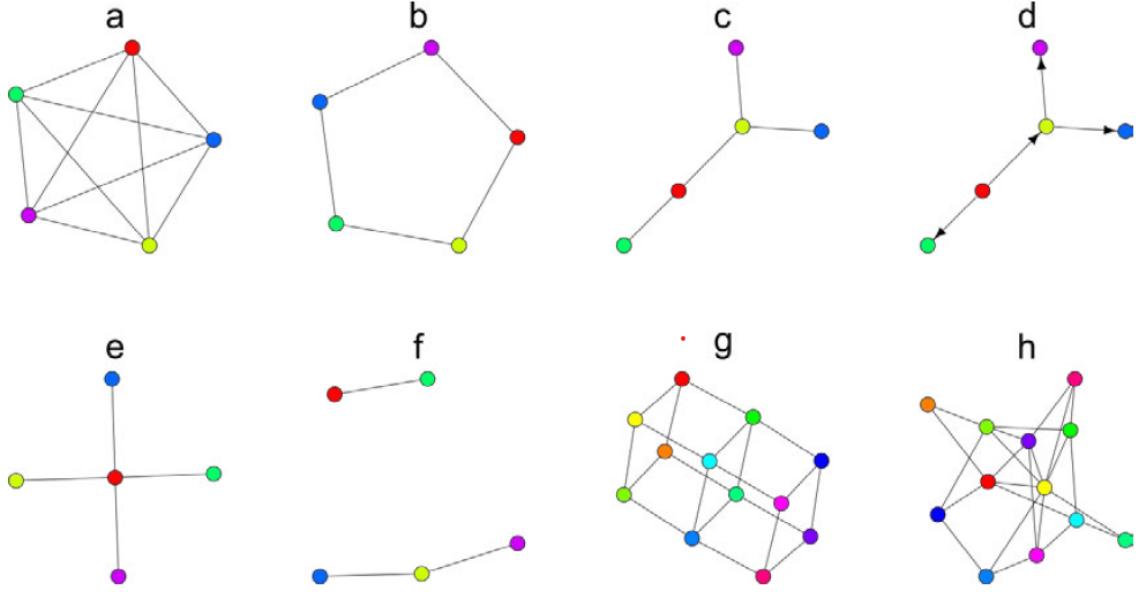


Figura 8: Ejemplo de algunos tipos de grafos. a: completo, b: simple, c: árbol, d: dígrafo, e: estrella, f: no conexo, g: de celosía, h: de mundo pequeño [14].

6.2.4. Matrices asociadas a un grafo

La representación gráfica de un grafo es poco práctica para su análisis, por esto es mejor utilizar su representación matricial [15]. A continuación, se mencionan algunas matrices útiles y su implementación para el grafo de la Figura 7.

- Matriz de incidencia (I): Se tiene una matriz de v vértices por e aristas. El elemento $a_{ve} = 1$ si la arista conecta con el vértice, de lo contrario $a_{ve} = 0$

$$I = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matriz de adyacencia (A): El grafo se representa con una matriz cuadrada de tamaño $v \times v$. Si hay una arista entre el vértice v_1 y v_2 , el elemento $a_{v_1 v_2} = 1$.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Matriz de grados (D): Se tiene una matriz diagonal de tamaño $v \times v$ que contiene los grados de cada vértice.

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

- Matriz laplaciana (L): Esta matriz es resultado de operar otras matrices asociadas al grafo. Para grafos no dirigidos, esta se calcula como $L = D - A$. Para grafos dirigidos se calcula como $L = II^T$.

$$l = \begin{bmatrix} 3 & -1 & -1 & 0 & -1 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 2 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & 2 \end{bmatrix}$$

- Matriz de rigidez (K): Esta matriz surge de qué tanto es posible deformar el grafo sin doblar ni modificar la longitud de las aristas [16]. A continuación se muestra la matriz de adyacencia totalmente rígida.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Esta corresponde a la matriz de adyacencia del grafo completo.

6.3. Control de la formación

Para resolver el problema de control de formaciones se utilizan dos grafos asociados. El primero es el grafo de formación. Este es un grafo no dirigido que contiene la configuración deseada. Además, se utiliza un grafo ponderado donde las longitudes de las aristas representan la distancia entre los agentes. El segundo grafo es el que representa la red de comunicación entre agentes. Este es un grafo dirigido donde las aristas están definidas por la dinámica de lazo cerrado del sistema multi-agente [8].

6.3.1. Grafo de formación

Para definir un grafo de formación se debe entender el concepto de rigidez. Una estructura es rígida cuando todas las distancias entre los vértices están definidas y se mantienen constantes. Para evaluar la rigidez de un grafo se debe cumplir $e = 2v - 3$, donde v es el número de vértices y e el número de aristas. Si un grafo tiene menos aristas que vértices, este ya no se considera rígido [17].

Con esto, se introduce el término de un grafo mínimamente rígido. Para que un grafo se considere mínimamente rígido, debe tener un estado donde si se elimina cualquier arista, el grafo deja de ser rígido.

La ventaja de trabajar con grafos mínimamente rígidos es que no tienen restricciones innecesarias a la hora de mantener la formación.

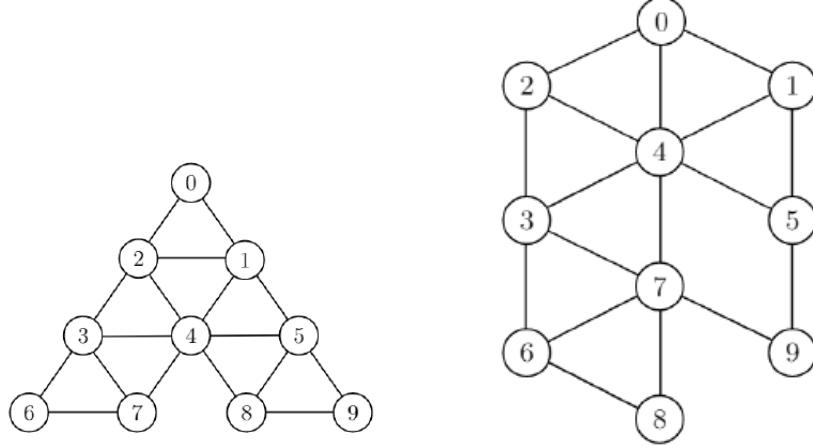


Figura 9: Grafos de formación en triángulo y hexágono [8].

6.3.2. Construcción de un grafo mínimamente rígido

El método de Henneberg es utilizado para construir grafos mínimamente rígidos donde cada vértice mantiene dos grados de libertad [17]. Los pasos para realizar el método son:

1. Numerar todos los vértices.
2. Agregar una arista entre el vértice 1 y el vértice 2.
3. Agregar los demás vértices a la estructura utilizando dos aristas.
4. Verificar que se cumple la condición de $e = \frac{v^2-v}{2}$

6.3.3. Grafo para la red de comunicación

La comunicación entre agentes se representa por un grafo de comunicación. Este es un dígrafo que indica cuáles agentes tienen comunicación entre ellos y hacia qué dirección se comunican [18]. A continuación se muestran tres tipos de redes.

- Red estática: Las aristas se mantienen invariantes en el tiempo.
- Red dinámica o dependiente del estado: Las aristas son variantes en el tiempo y pueden desaparecer o aparecer según el estado de la red de agentes.

- Red aleatoria: La existencia de una arista se da mediante una distribución de probabilidad.

6.4. Teoría de control

En la robótica, se utilizan sistemas de control que integran varios subsistemas y procesos para estabilizar una respuesta inestable. Esto se logra mediante la implementación de un controlador que manipula las entradas del sistema para obtener la salida deseada [19]. En la Figura 10 se muestra el sistema de control más utilizado.

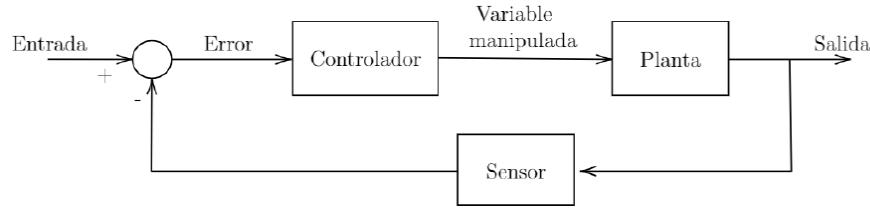


Figura 10: Sistema de control en lazo cerrado [19].

6.4.1. Control de formación

Para realizar el control de formaciones se requieren dos niveles de control, uno superior y otro inferior. El control de capa superior maneja el comportamiento de los agentes y sus posiciones. El control de capa inferior controla la velocidad de las ruedas en cada agente [8].

6.5. Cinemática de robots diferenciales con ruedas

Para la implementación física en la robótica de enjambre se requiere un modelo de movimiento que contemple las dimensiones y características físicas del robot como se observa en la Figura 11. Para esto se tienen las ecuaciones (1) y (2). Donde ℓ es la distancia del motor hasta su centro, φ es el ángulo de orientación del uniciclo dentro del plano XY , v es la velocidad lineal del robot, w es la velocidad angular de las ruedas y r es el radio de las ruedas [20].

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

$$w = \frac{r(\dot{\varphi}_R - \dot{\varphi}_L)}{2\ell} \quad (2)$$

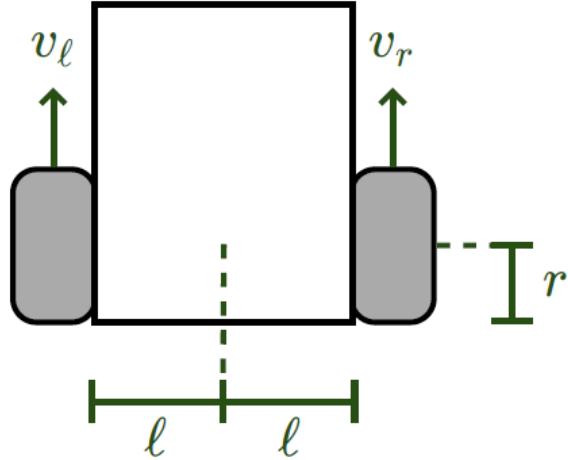


Figura 11: Modelo de uniciclo [20].

Adicional a esto, se puede calcular la velocidad controlada de la rueda derecha y la rueda izquierda con las ecuaciones (3) y (4), respectivamente.

$$\dot{\varphi}_{R,crtl} = \frac{v_{ctrl} + \ell w_{ctrl}}{r} \quad (3)$$

$$\dot{\varphi}_{L,crtl} = \frac{v_{ctrl} - \ell w_{ctrl}}{r} \quad (4)$$

6.6. Otras ecuaciones matemáticas relevantes

Para mantener una correcta formación y distribución del enjambre, se necesitan herramientas como la ecuación de consenso y la función de tensión.

6.6.1. Ecuación de consenso

La ecuación 5 se utiliza para mantener la formación de los agentes en la posición asignada. Esta toma en cuenta el centro de masa de la formación y calcula la velocidad de cada agente para mantener la forma del grafo [8].

$$v_i(t) = \sum_{j \in N(i)} (x_j(t) - x_i(t)), i = 1, 2, \dots, n \quad (5)$$

Donde $N(i)$ es el conjunto de unidades adyacentes de la unidad i en la red multi-agente.

Luego de añadir los pesos, se obtiene la ecuación 6:

$$\frac{\partial \varepsilon_{i,j}}{\partial x_i} = w_{i,j}(\|x_i - x_j\|)(x_i - x_j) \quad (6)$$

Donde de puede despejar el peso $w_{i,j}$.

6.6.2. Funciones racionales para hallar la tensión

Para modificar la ecuación de consenso es necesario hallar la tensión utilizando funciones racionales que toman en cuenta las distancias deseadas y las distancias restringidas [8].

- Modelo 1: Combinación aditiva del control de formación y evasión de obstáculos.

$$\epsilon_{ij} = \frac{1}{2}(\|x_i - x_j\| - d_{ij})^2 + \frac{\|x_i - x_j\|^2}{\|x_i - x_j\| - r} \quad (7)$$

Donde, d_{ij} es la distancia entre los agentes i y j , y r es el radio de los agentes.

- Modelo 2: Combinación de control de formación, evasión de obstáculos y mantenimiento de la conectividad.

$$\epsilon_{ij} = \frac{(\|x_i - x_j\| - d_{ij})^2}{(\|x_i - x_j\| - r)(\|x_i - x_j\| - R)} \quad (8)$$

Donde, R es la distancia máxima que se pueden alejar los agentes sin salirse del rango del radar de los otros.

- Modelo 3: Combinación de control de formación y evasión de obstáculos.

$$\epsilon_{ij} = \frac{2(\|x_i - x_j\| - d_{ij})^2}{\|x_i - x_j\| - r} \quad (9)$$

- Modelo 4: Modelo dinámico con control de formación y evasión de obstáculos.

Se utiliza un modelo dinámico que al inicio, cuando los agentes comienzan en posiciones aleatorias, se utiliza únicamente el control para evitar colisiones. Luego, cuando los agentes están lo suficientemente cerca sin chocarse, se cambia al modelo de la ecuación 9 que toma en cuenta el control de la formación

- Modelo 5: Modelo dinámico con control de formación usando coseno hiperbólico y evasión de obstáculos.

$$\epsilon_{ij} = 0.01 \cosh(1.8\|x_i - x_j\| - 8.4) \quad (10)$$

Se utilizó el coseno hiperbólico ya que es una función “plana” que permite que el control de formación sea el que decida dónde deben posicionarse los agentes en caso de tener un mínimo erróneo en una distancia de 0.

- Modelo 6: Modelo dinámico con control de formación usando coseno hiperbólico y evasión de obstáculos incluyendo límites de velocidad.

Esta modificación no afecta directamente a la ecuación. Se realiza después de esta y consiste en incluir un límite de velocidad al modelo de la ecuación 10. Por lo tanto, si la velocidad obtenida con el modelo es mayor al límite establecido, solo se tomará en cuenta la dirección.

6.7. Herramientas de software

Para realizar los cálculos, el control y las pruebas con los agentes robóticos, se necesita utilizar software especializado como los siguientes:

6.7.1. Matlab

Matlab es una plataforma de programación desarrollada por MathWorks con su propio lenguaje especializado para aplicaciones de ingeniería, análisis de datos, generación de algoritmos, entre otras [21].

Para el presente trabajo, se utiliza Matlab como la fuente de procesamiento de datos y generación de trayectorias para el algoritmo de sincronización y control de formaciones.

6.7.2. Entorno de simulación Webots

Webots es un software de código abierto creado por Cyberbotics [22]. Esta plataforma se centra en la simulación de escenarios que permiten replicar situaciones reales con distintos tipos de robots y objetos tal como se muestra en la Figura 12. Varios de los modelos disponibles en Webots están calibrados para comportarse como el modelo real y el usuario puede modificar los parámetros para garantizar una simulación realista.

Webots también cuenta con la implementación de controladores en lenguajes como C, C++, Matlab, Python, Java, ROS, o con API.

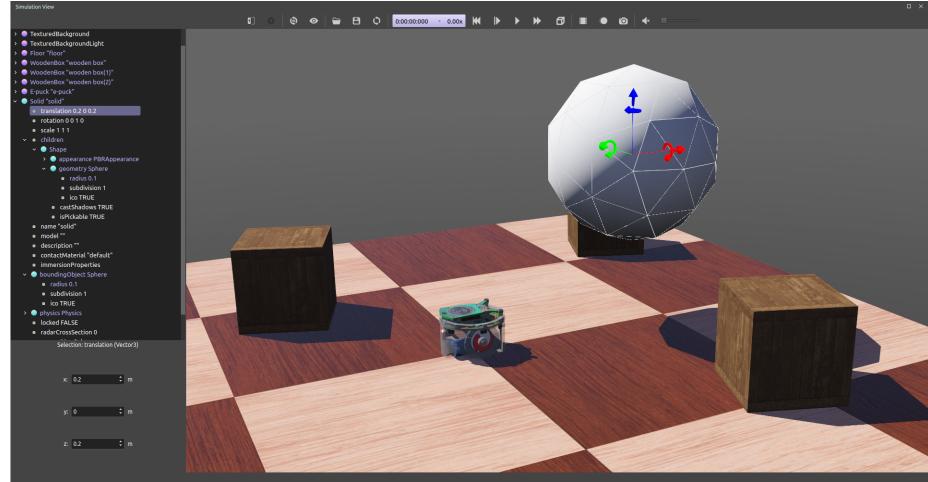


Figura 12: Entorno de simulación en Webots [22].

6.7.3. Paralelismo computacional

Al momento de trabajar con algoritmos y problemas complejos, se requiere utilizar métodos de optimización de software para agilizar el procesamiento de cálculos y la ejecución de

tareas. Para esto, se utilizan herramientas como el paralelismo computacional. Consiste en dividir las tareas y asignarlas a cada núcleo del procesador para ejecutarlas simultáneamente [23]. Esto reduce considerablemente el tiempo de procesamiento. Además es importante considerar las dependencias entre las tareas como:

- Dependencia de control de secuencia: Es el orden secuencial clásico de los algoritmos secuenciales.
- Dependencia de comunicación: Es cuando una tarea depende de la información que envíe otra tarea.

En la Figura 13 se observa un ejemplo donde puede existir tiempos muertos ya que la tarea 3 depende de la información enviada por la tarea 1 y la tarea 4 depende de la información enviada por la tarea 3.

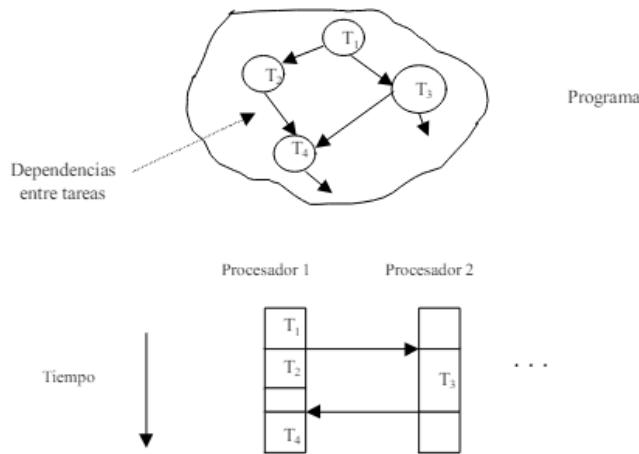


Figura 13: Ejemplo de dependencia de tareas [23].

6.8. Ecosistema Robotat

El Robotat es una plataforma que cuenta con herramientas y sistemas de captura de movimiento útiles para experimentar con la robótica de enjambre. A continuación se describen las herramientas más importantes.

6.8.1. Mesa de pruebas

Es una plataforma de acero y plycem con un espacio útil de $5 \times 5 \times 3$ m, capaz de soportar cargas puntuales de hasta dos toneladas.

6.8.2. Sistema de captura de movimiento OptiTrack

Es un sistema de captura de movimiento que consta de 6 cámaras OptiTrack Prime^x 41 al rededor de la mesa de pruebas. Estas son cámaras de alta precisión y baja latencia que permiten realizar experimentos en tiempo real. En la Figura 14 se muestra el modelo disponible en la Universidad del Valle de Guatemala que tiene las siguientes características [24]:

- Resolución de 4.1 mega píxeles (MP).
- Precisión de ± 0.10 mm.
- Errores rotacionales menores a 0.5 grados.
- Lentes de 12 mm.
- Tasa de refresco nativa de 180 FPS con un máximo de hasta 250 FPS.
- Distancia de captura de hasta 100 pies desde la cámara al marcador.
- Rango de captura de hasta 290,000 pies cúbicos por cámara para marcadores pasivos
- Rango de captura de hasta 1,000,000 pies cúbicos por cámara para marcadores activos.



Figura 14: Cámara de captura de movimiento Prime^x 41 de OptiTrack [24].

6.8.3. Comunicación del Robotat

La plataforma del Robotat utiliza un protocolo de comunicación TCP para transmitir la información de las cámaras OptiTrack al servidor principal del laboratorio. Este servidor de Python envía los datos a través de Wi-Fi en una red local. A esta red local, se puede acceder con una computadora para extraer la información que se necesite, además las plataformas robóticas también se pueden conectar a la red local para recibir instrucciones.

6.9. Hardware

Para poner a prueba el algoritmo de sincronización y control de formaciones en un ambiente físico, se necesita de una plataforma robótica móvil que en este proyecto será el Pololu 3pi+.

6.9.1. Plataforma móvil Pololu 3pi+ modificado

La plataforma móvil a utilizar es una modificación basada en el Pololu 3pi+ 32U4 OLED Robot [25]. Tiene un diámetro de 97 mm y altura de 36 mm, además, cuenta con lo siguiente:

- Procesador ATmega32U4 MCU @ 16 MHz.
- Sensores de línea y de choque frontales.
- Encoders de doble cuadratura para control de posición y velocidad en lazo cerrado.
- IMU (acelerómetro de 3 ejes, magnetómetro y giroscopio).
- Pantalla OLED integrada.

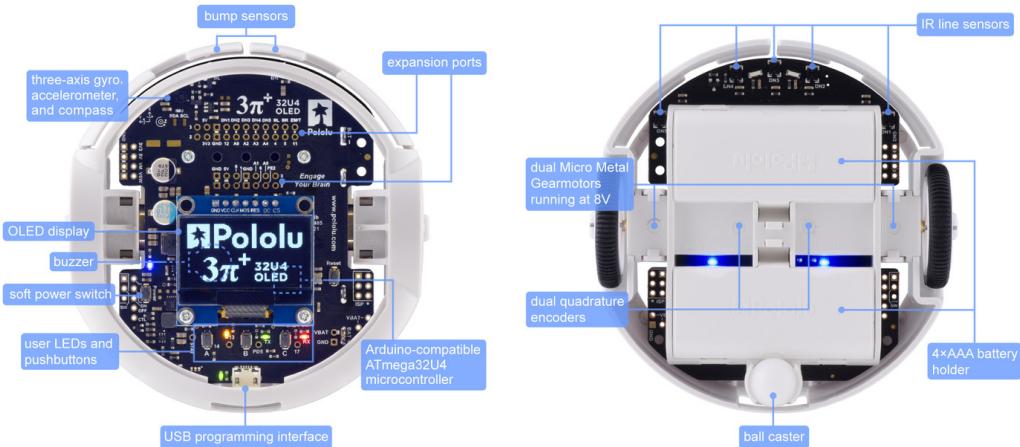


Figura 15: Sensores y componentes del Pololu 3pi+ 32U4 OLED Robot [25].

A esta plataforma móvil se le adaptó un ESP32 ya que el microcontrolador original tiene poca capacidad de procesamiento. Esto permitió tener control de capa superior e inferior, además de incorporar una red de comunicación WiFi con el Robotat para controlar los agentes.

CAPÍTULO 7

Algoritmo de sincronización y control de formaciones

El algoritmo de sincronización y control de formaciones funciona con dos controladores principales. El primero es el algoritmo de sincronización y control centralizado llamado “supervisor” y el segundo es el algoritmo de control de uniciclo para los agentes. En este capítulo se explicará el funcionamiento de cada controlador y la lógica detrás de cada uno para cumplir con las tareas de formación, evasión de colisiones y movilización hacia un objetivo.

7.0.1. Controlador del supervisor

El programa del supervisor tiene como tarea coordinar a los agentes, realizar procesamiento de datos y generar trayectorias para movilizar a los agentes hacia un objetivo evadiendo colisiones entre ellos y con los obstáculos. Para esto, el algoritmo cuenta con diferentes segmentos.

Adquisición de posiciones

La primera parte del algoritmo consiste en la toma de posiciones X y Y actuales de los agentes, obstáculos y el objetivo.

Cálculo de velocidades de agentes

Luego, se realiza el cálculo de las velocidades de los agentes para moverlos hacia su posición en la formación y hacia el objetivo de interés. Este cálculo de velocidades se realiza

por medio de la ecuación de consenso con un factor de peso ω que depende de las ecuaciones de tensión y de evasión de obstáculos.

Evasión de obstáculos y colisiones

Seguido a esto, se tiene la evasión de obstáculos y colisiones que se realiza comparando la posición actual de cada agente con la posición actual de los agentes vecinos y los obstáculos.

Control proporcional

El siguiente segmento implementa el control proporcional de la Ecuación (11) para movilizar a los agentes hacia un punto de interés.

$$v_{n+1} = v_n + k(x_{objetivo} - x_{agente}) \quad (11)$$

Donde v es la velocidad del agente y k es la ganancia o constante de proporcionalidad que multiplica a la distancia entre el punto de interés y la posición del agente a evaluar. Una vez se tiene la velocidad en X y Y de los agentes, se calcula la norma de velocidad y esta se utiliza para la toma de decisiones dentro del algoritmo.

7.0.2. Controlador de los agentes

El programa del los agentes recibe las velocidades calculadas por el supervisor. Luego, se encarga de convertirlas en velocidad lineal y angular para calcular la velocidad de cada rueda de los Pololu 3Pi+ según el modelo del uniciclo. Este programa es individual para cada agente y solo procesa los datos para si mismo.

7.0.3. Funcionamiento del algoritmo

El algoritmo consta de las siguientes etapas para su ejecución:

- Etapa 0: los agentes se movilizan hacia sus posiciones iniciales para iniciar el experimento.
- Etapa 1: comienza el acercamiento de agentes hasta que la norma de velocidad esté por debajo de un valor seleccionado.
- Etapa 2: los agentes se colocan en sus posiciones de la formación hasta que el error cuadrático medio entre la formación actual y la formación deseada esté por debajo de un valor seleccionado.
- Etapa 3: el líder se mueve hacia el objetivo y los agentes de la formación lo siguen.

En las Figuras 16 y 17 se muestra el diagrama de flujo para los programas del supervisor y el control de los agentes.

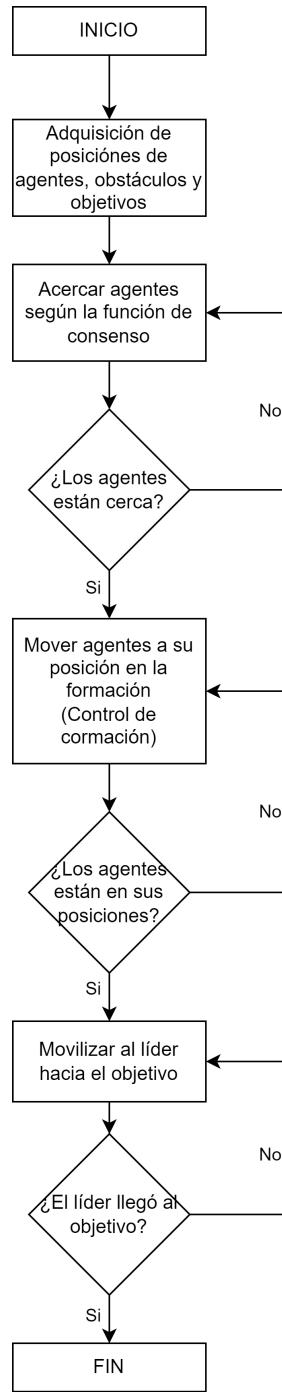


Figura 16: Diagrama de flujo para el supervisor.

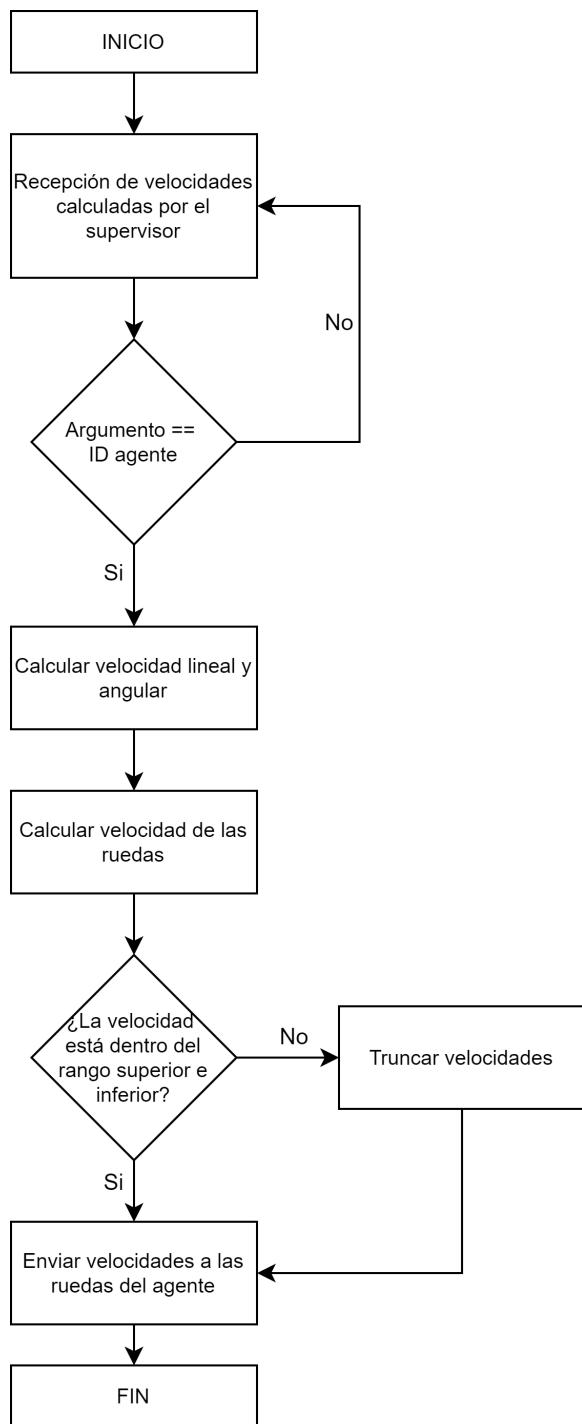


Figura 17: Diagrama de flujo para el programa de los agentes.

CAPÍTULO 8

Restauración del algoritmo desarrollado en fases previas

La implementación en físico del algoritmo de sincronización y control de formaciones llevada a cabo por José Alejandro Rodríguez [9] se realizó en Webots con la versión 2023b. Esta es la misma versión que la utilizada en la fase actual. También, se optó por utilizar Python 3.10 como lenguaje de programación para el controlador del supervisor y de los agentes ya que es una de las versiones más recientes de software, además que es la misma versión utilizada en la fase anterior.

Por otro lado, en el Robotat se han realizado cambios en las conexiones con el servidor y los Pololu 3Pi+ por lo que se sería necesario un reajuste de parámetros.

En este capítulo se detallarán las modificaciones necesarias para la restauración del algoritmo y su correcta ejecución en simulaciones con Webots y en un entorno físico con el Robotat. Además, se realizará la validación de la naturaleza del algoritmo en escenarios con obstáculos móviles.

8.1. Replicar simulaciones de Webots

El primer paso de la restauración, fue recrear algunas simulaciones en Webots. Para esto, fue necesario instalar las siguientes librerías que no se encuentran dentro de la biblioteca estándar de Python:

- Keyboard - versión 0.13.5
- NumPy - versión 1.23.2

Luego, se realizó la prueba del código para el controlador del supervisor y los agentes

incluyendo sus respectivas funciones:

- Supervisor: Supervisor_simulacion_y_fisico_v4.py
- Agentes: pruebaMatrizDifeomorfismo.py
- Funciones algoritmo: funciones.py, funVel.py

8.1.1. Comunicación entre supervisor y agentes

Al ejecutar la primera simulación, se encontró que dos agentes permanecían inmóviles durante la primera etapa del algoritmo que consiste en mover a los agentes a sus posiciones iniciales, tal como se muestra en la Figura 18.

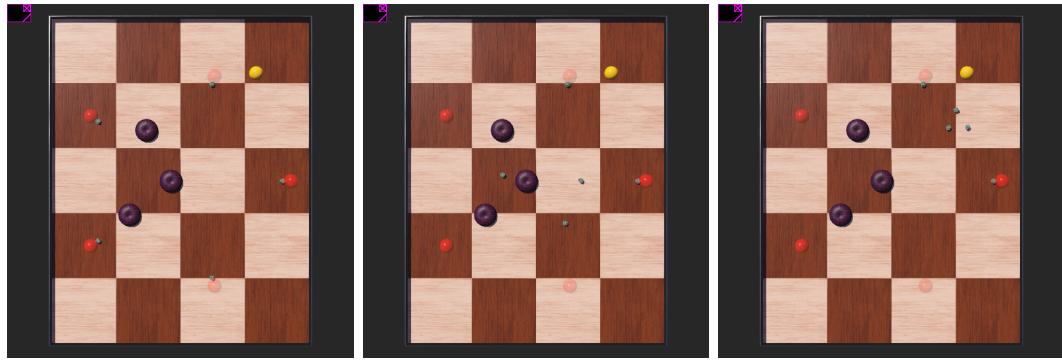


Figura 18: Ejecución de la primera simulación en Webots 2023b.

Esto se debe a que actualmente se está utilizando una computadora con mayor capacidad de procesamiento que la utilizada en la fase anterior. Esta capacidad de procesamiento impacta en la velocidad de ejecución de los programas.

Dado que la comunicación entre el supervisor y los agentes se realiza por medio de una memoria compartida y la velocidad de procesamiento es mayor, el controlador de los dos primeros agentes se ejecuta antes de que el espacio de memoria compartida se haya inicializado correctamente. Para solucionar el problema, en el controlador de los agentes se agregó un tiempo de espera de 3 segundos antes de inicializar y acceder a la memoria compartida para asegurar que esta se cree correctamente en el controlador del supervisor.

8.1.2. Prueba del algoritmo con simulaciones basadas en escenarios previos

Una vez solucionado el problema de comunicación, se realizó las primeras simulaciones para replicar algunos de los escenarios realizados por José Alejandro Rodríguez [9] y comprobar el funcionamiento correcto del algoritmo.

El código del supervisor ya cuenta con un modo de simulación en el que se ejecuta el algoritmo basado en condiciones iniciales tomadas de un escenario físico. Para esto, se debe

cargar un archivo .npz que cuenta con la información necesaria para configurar la simulación según el escenario real en que se ejecutó el algoritmo.

Primer escenario

Para la primera simulación, se utilizó el archivo “finaltrial_6A_AAA_f_1.npz” que consiste en la siguiente configuración:

- Cantidad de agentes: 6
- Posición inicial de agentes: línea
- Obstáculos: ninguno
- Objetivo: ubicado en la esquina

En la Figura 19, las imágenes en el orden de izquierda a derecha y luego de arriba hacia abajo muestran la secuencia de ejecución del algoritmo.

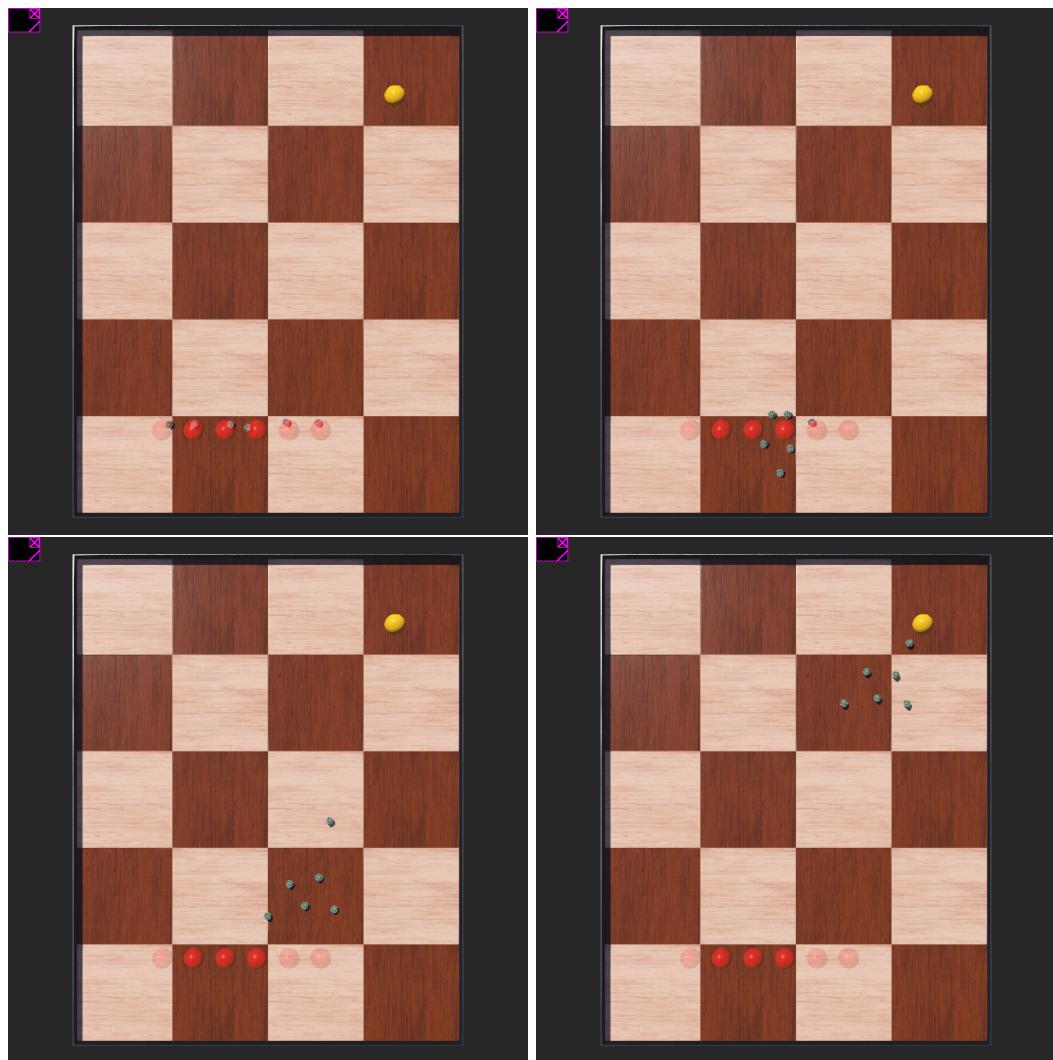


Figura 19: Ejecución del algoritmo en la primera simulación

Segundo escenario

Para la segunda simulación, se utilizó el archivo “finaltrial_6A_AB1B_f_2.npz” que consiste en la siguiente configuración:

- Cantidad de agentes: 6
- Posición inicial de agentes: línea
- Obstáculos: ubicados en el centro
- Objetivo: ubicado en el centro

En la Figura 20, las imágenes en el orden de izquierda a derecha y luego de arriba hacia abajo muestran la secuencia de ejecución del algoritmo.

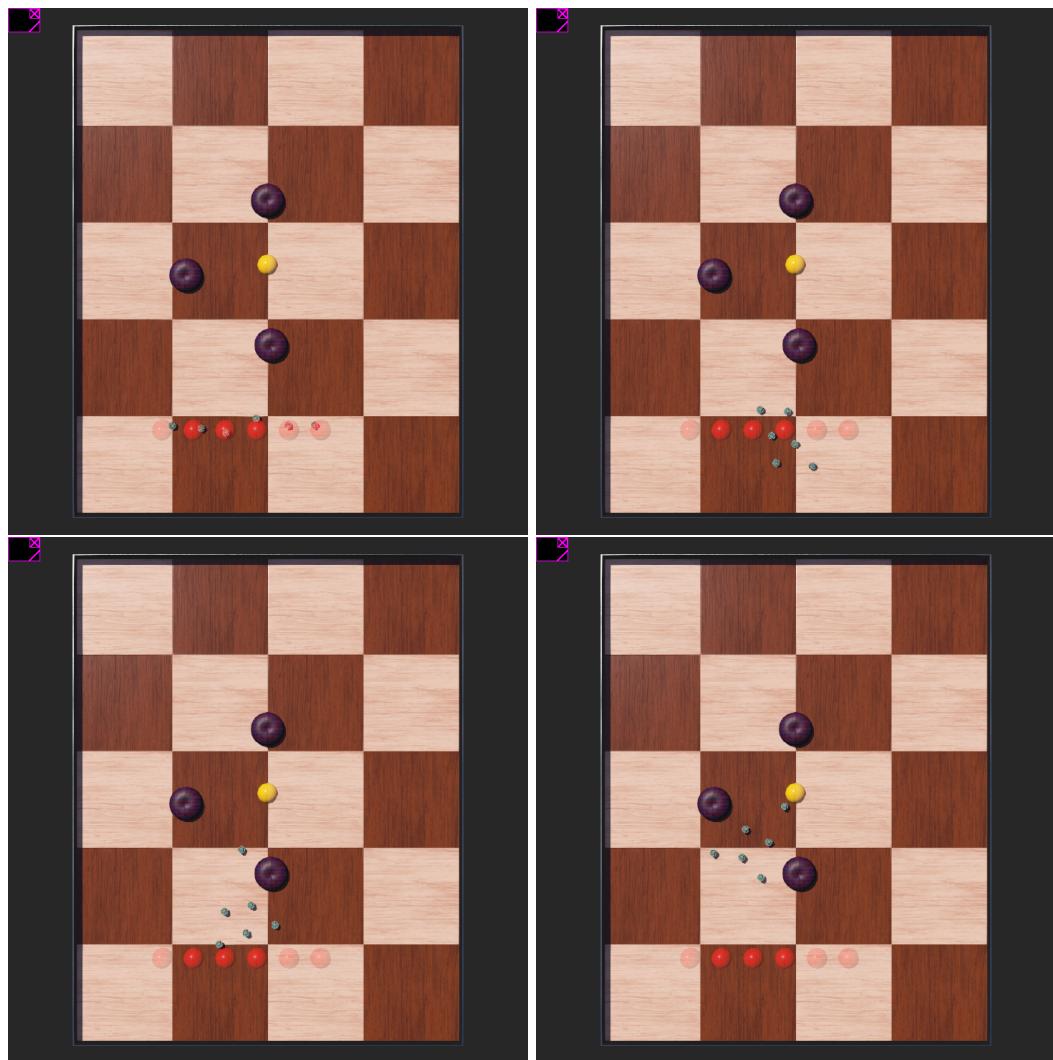


Figura 20: Ejecución del algoritmo en la segunda simulación

Tercer escenario

Para la tercera simulación, se utilizó el archivo “finaltrial_6A_BCA_f_1.npz” que consiste en la siguiente configuración:

- Cantidad de agentes: 6
- Posición inicial de agentes: círculo
- Obstáculos: ubicados en posiciones aleatorias
- Objetivo: ubicado en la esquina

En la Figura 21, las imágenes en el orden de izquierda a derecha y luego de arriba hacia abajo muestran la secuencia de ejecución del algoritmo.

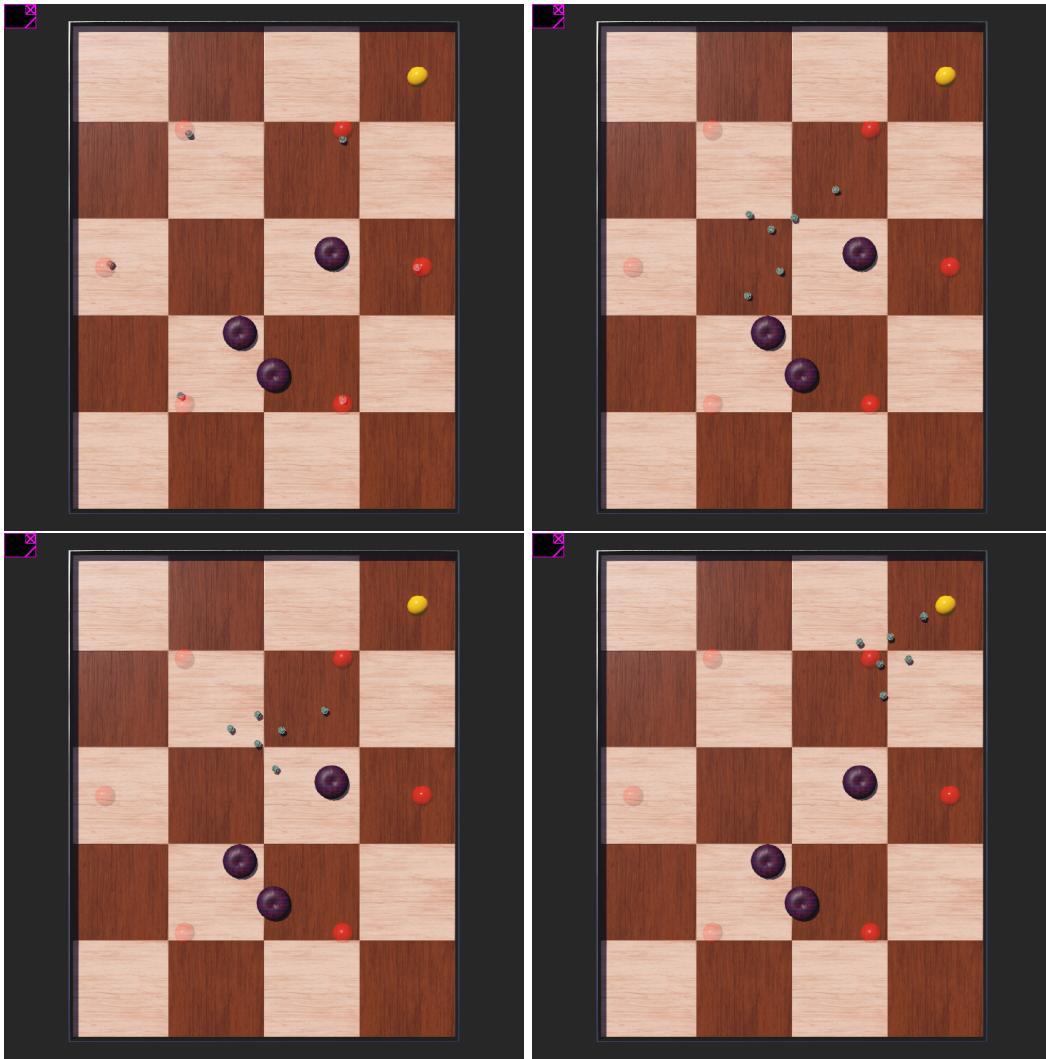


Figura 21: Ejecución del algoritmo en la tercera simulación

8.2. Replicar el funcionamiento del algoritmo en el Robotat

Una vez teniendo el algoritmo funcionando en Webots, el siguiente paso fue verificar que el algoritmo se ejecute correctamente en el Robotat.

8.2.1. Pruebas de conexión con el Pololu 3Pi+ y el Robotat

Primero, se realizaron pruebas de conexión y obtención de datos con el Robotat y los Pololu 3Pi+ utilizando las funciones creadas en Python por José Alejandro Rodríguez [9]:

- `robotat_connect`
- `robotat_disconnect`

- robotat_get_pose
- robotat_3pi_connect
- robotat_3pi_disconnect
- robotat_3pi_set_wheel_velocities
- robotat_3pi_force_stop

Al realizar las pruebas de conexión con los agentes Pololu 3Pi+ se tenía el error mostrado en la Figura 22. Este error se debe a que hubo un cambio en los puertos de conexión con el ESP32 de los agentes. Anteriormente la conexión se realizaba en el puerto 8888 y actualmente se realiza en el puerto 9090. Al actualizar este dato dentro de la función “robotat_3pi_connect” en del archivo “funciones_conjunto_3pi.py” se logró una conexión exitosa con el agente para el envío de comandos de velocidades.

```
ERROR: Could not connect to the robot.
[WinError 10061] No se puede establecer una conexión ya que el equipo de destino denegó expresamente dicha conexión
An error occurred while sending the wheel velocities command: 'tcpsock'
An error occurred while sending the force stop command: 'tcpsock'
```

Figura 22: Error de conexión con Pololu 3Pi+.

8.2.2. Calibración de marcadores

Una vez lograda la conexión con los Pololu 3Pi+, se tuvo que realizar una nueva calibración de los marcadores del OptiTrack para obtener el desfase del ángulo orientación (*bearing*) de cada marcador. Estos desfases son diferentes en cada marcador y se producen por la forma en que el OptiTrack identifica cada uno según la posición de sus esferas reflectivas.

En la fase anterior, se realizó una calibración con los marcadores del 1 al 15, sin embargo, se agregaron nuevos marcadores por lo que ahora se cuenta con 22 de ellos para utilizar, de los cuales están inhabilitados el 1 y 9 ya que serán utilizados en otros proyectos de graduación. Por tanto, ahora se tienen disponibles los marcadores de la Figura 23.

Para corregir el desfase de cada marcador, primero se obtiene la orientación con ángulos de Euler en la secuencia *zyx*, luego, al primer ángulo que representa la rotación respecto al eje *z* se le resta el desfase obtenido θ_z . Para realizar la calibración, se colocaron todos los marcadores disponibles de la Figura 23 con la misma orientación sobre el eje *y* de la mesa de pruebas del Robotat tal como se observa en la Figura 24.

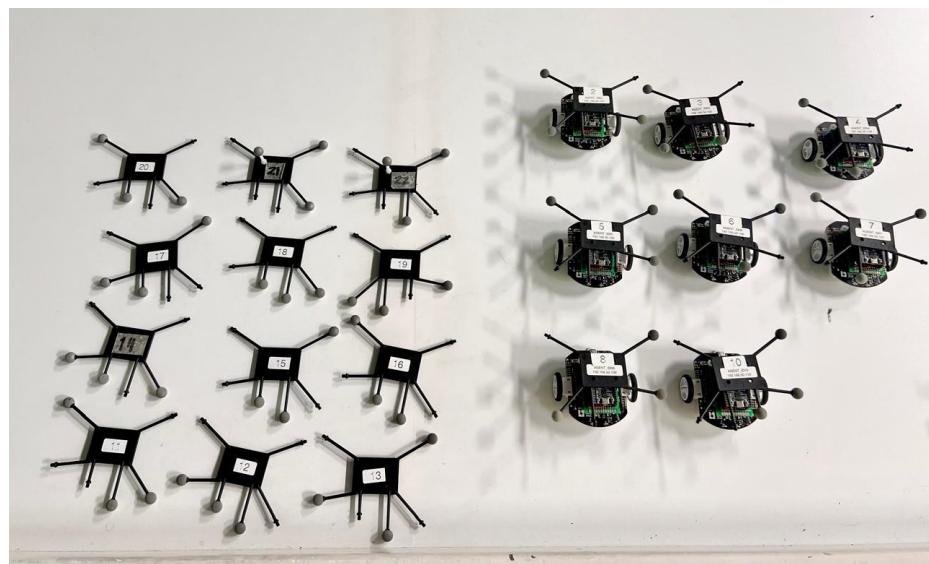


Figura 23: Marcadores del OptiTrack disponibles para su uso.

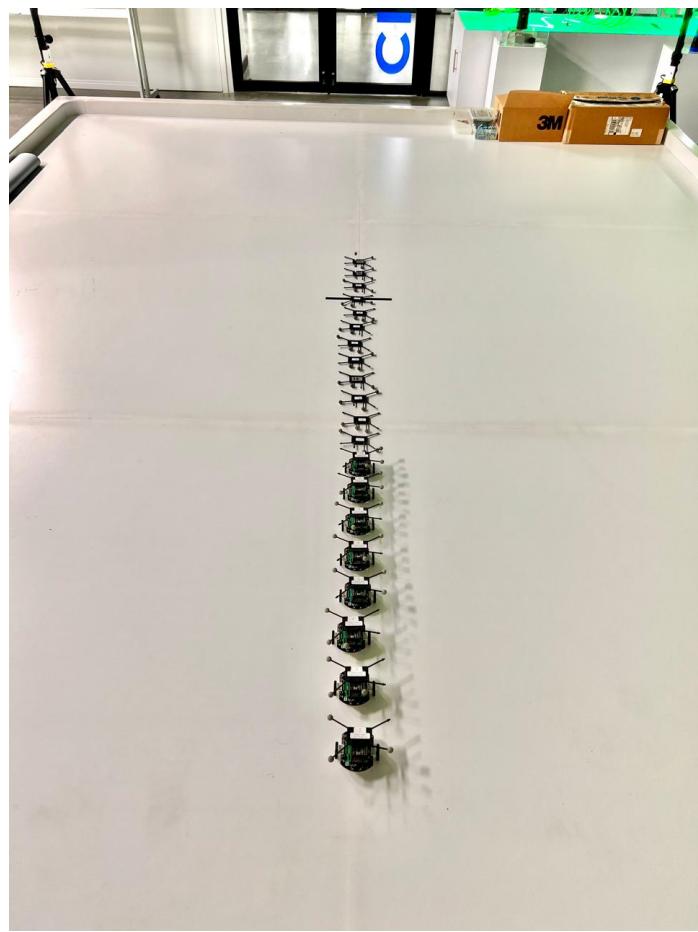


Figura 24: Marcadores alineados sobre el eje y de la mesa de pruebas del Robotat.

Al obtener la pose de cada marcador, se realizó la conversión a ángulos de Euler en la secuencia zyx y se obtuvo los ángulos de desfase del Cuadro 1. Al comparar los desfases actuales con los obtenidos en la fase previa mostrados en el Cuadro 2, se observó que los valores son similares por lo que se optó tomar los desfases anteriores para los marcadores 1 y 9 faltantes en la calibración actual. Por último, en el Cuadro 3 se observan los desfases finales para cada uno de los marcadores del 1 al 22. Estos valores se guardaron en un archivo .npy llamado “nueva_calibracion_markers_1_al_22.npy” para aplicarlos luego de obtener la pose de cada marcador en el algoritmo y así tener siempre un ángulo respecto del eje z igual a cero ($\theta_z = 0$).

Marcador	Desfase θ_z en grados
2	-47.2746822
3	-90.32278167
4	-135.7890952
5	179.3715195
6	-141.0274548
7	-175.0541793
8	-78.11543097
10	143.2079276
11	111.0832797
12	166.1718189
13	-127.3111755
14	-109.4993486
15	-40.73944282
16	-104.1691129
17	-121.1927571
18	-92.48122033
19	4.298050244
20	-133.2161012
21	-112.0477043
22	-15.28469666

Cuadro 1: Desfases de marcadores disponibles alineados con el eje y de la mesa de pruebas del Robotat.

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 2: Desfases de marcadores obtenidos en la fase previa por José Alejandro Rodríguez [9].

Marcador	Desfase θ_z en grados
1	91.99470275
2	-47.2746822
3	-90.32278167
4	-135.7890952
5	179.3715195
6	-141.0274548
7	-175.0541793
8	-78.11543097
9	-9.876210458
10	143.2079276
11	111.0832797
12	166.1718189
13	-127.3111755
14	-109.4993486
15	-40.73944282
16	-104.1691129
17	-121.1927571
18	-92.48122033
19	4.298050244
20	-133.2161012
21	-112.0477043
22	-15.28469666

Cuadro 3: Desfases finales para calibración de marcadores del 1 al 22.

8.2.3. Selección de marcadores a utilizar

Una vez guardada la calibración de marcadores, se realizó pruebas en el Robotat para ejecutar el algoritmo de sincronización y control de formaciones. Sin embargo, el primer problema que se identificó fue que al menos un agente siempre permanecía inmóvil. En la Figura 25 se observa la ejecución del algoritmo donde únicamente se mueve un agente de los dos utilizados.

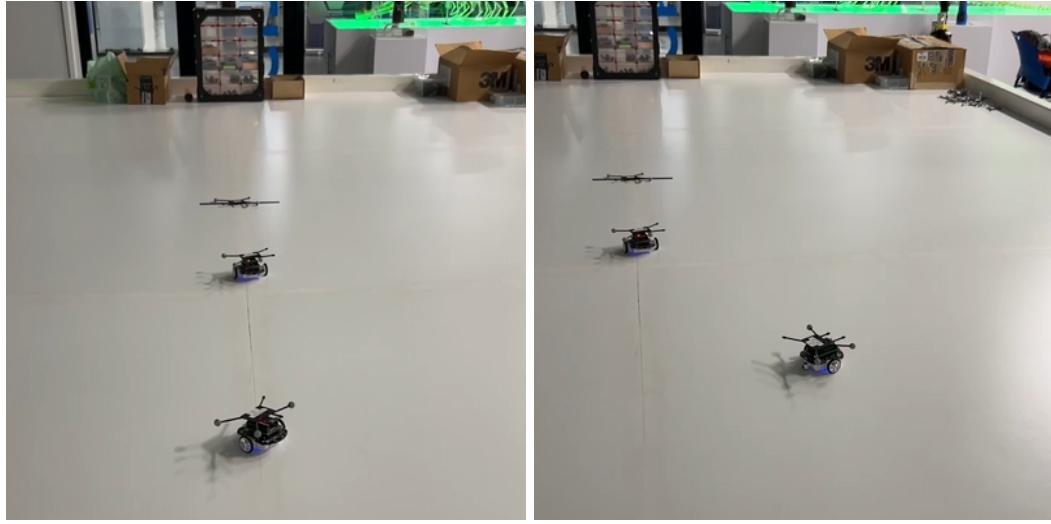


Figura 25: Problema de funcionamiento en físico, agentes permanecen inmóviles.

Anteriormente, para obtener las poses de los marcadores se utilizaba una lista predeterminada con los valores enteros del 1 al 12 en orden ascendente que serían los marcadores a utilizar. Sin embargo esto ocasionaba diferentes problemas.

Para obtener las poses de los marcadores se utiliza la función “robotat_get_pose”, esta recibe como argumentos el objeto TCP y los números de marcadores. Al tener una lista predeterminada de valores, siempre se está solicitando la pose de los 12 marcadores aunque no todos estén en uso, resultando en una solicitud de datos innecesaria para el servidor. Además, dado que actualmente no se tienen disponibles los marcadores 1 y 9, se excede el tiempo de espera (*Time Out*) para obtención de la pose de estos, lo que resulta en tiempos muertos durante la ejecución del código.

Para solucionar el problema, se optó por solicitar únicamente las poses de los marcadores a utilizar con una lista que contiene los números de todos los marcadores con el orden: agentes, objetivo y obstáculos. A continuación se muestra un ejemplo de esto.

Marcadores de agentes = [1, 2, 3, 4]

Marcador del objetivo = [22]

Marcadores de obstáculos = [14, 15, 16]

Marcadores a solicitar = [1, 2, 3, 4, 22, 14, 15, 16]

Por otro lado, para asignar los marcadores de cada agente, se debía elegir un intervalo de valores consecutivos de manera ascendente dentro de la lista predeterminada. Esto era problemático ya que limita a utilizar únicamente los agentes dentro del intervalo seleccionado, por lo que si un agente se descargaba, era obligatorio reemplazar las baterías para volver a utilizarlo. Por esto, se optó por cambiar la asignación de los marcadores para cada agente permitiendo seleccionar cualquier marcador disponible en el Robotat sin algún orden específico y ahora, para obtener las poses de los marcadores, únicamente se solicitan los datos al servidor de los marcadores en uso. Esto además, permite agilizar las pruebas a realizar ya que en múltiples ocasiones es necesario compartir los agentes Pololu 3Pi+ con otros compañeros.

En la Figura 26, el número dentro del círculo, representa el número de marcador asignado al agente. A la izquierda se puede observar la asignación de los marcadores a cada agente con la implementación del algoritmo original, mientras del lado derecho se observa una asignación arbitraria luego del cambio mencionado.

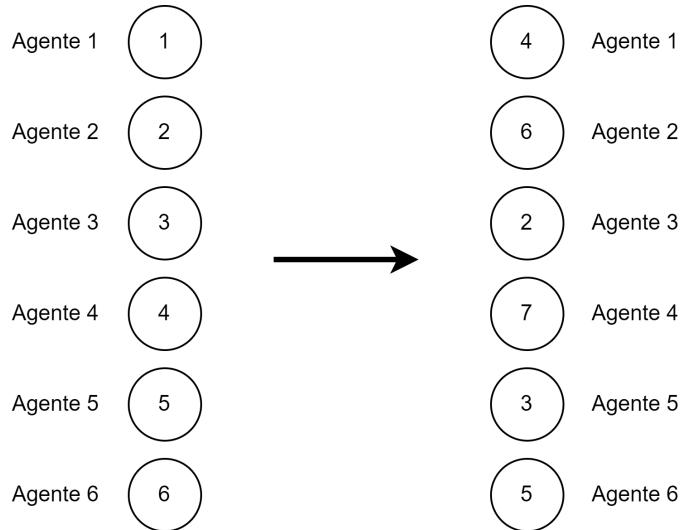


Figura 26: Selección de marcadores para cada agente antes y después de las modificaciones.

8.2.4. Ajuste de parámetros en el algoritmo

Una vez realizadas las modificaciones anteriores, al ejecutar el algoritmo en físico con tres agentes se encontró que su comportamiento es divergente en la etapa donde se movilizan hacia sus posiciones iniciales. En la Figura 27 se observa la trayectoria que toman los agentes.

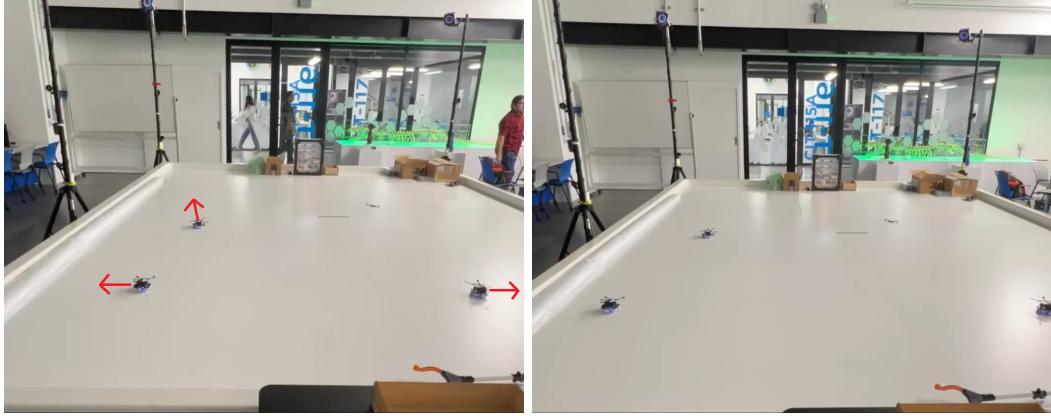


Figura 27: Problema de funcionamiento en físico, agentes divergen hacia posiciones iniciales.

Para intentar solucionar esto, se invirtió signo de la constante de proporcionalidad en el control de velocidad aplicado en la etapa 0 del algoritmo. Anteriormente se utilizaba $k = 5$, ahora se utiliza $k = -5$. Al aplicar el cambio, se encontró que ahora los agentes si se colocan en sus posiciones iniciales, sin embargo, ahora el comportamiento de divergencia se sigue presentando en las demás etapas del algoritmo tal como se observa en la Figura 28.

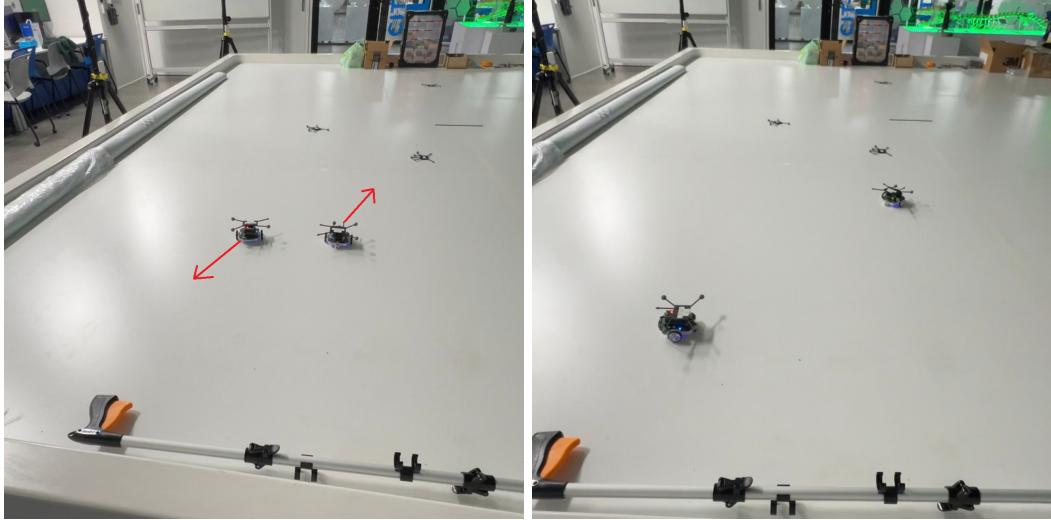


Figura 28: Problema de funcionamiento en físico, agentes divergen luego de colocarse en sus posiciones iniciales.

Al observar esto, se encontró que el patrón de divergencia se presentaba en todas las secciones donde se aplica el control proporcional para la velocidad de los agentes. Sabiendo esto, se optó por invertir el signo de la velocidad v_n y la constante k de la Ecuación (11) tal como se observa a continuación:

$$v_{n+1} = -v_n - k(x_{\text{objetivo}} - x_{\text{agente}}) \quad (12)$$

Con esto, se logró solucionar la divergencia de los agentes y se realizó la primera ejecución

del algoritmo exitosa utilizando dos agentes, sin obstáculos tal como se observa en la Figura 29. Las imágenes en el orden de izquierda a derecha y luego de arriba hacia abajo muestran la secuencia de ejecución del algoritmo.



Figura 29: Primera prueba exitosa de funcionamiento en físico.

Una vez realizada la primera prueba, se simuló un escenario arbitrario en Webots, sin embargo se encontró que ahora la simulación presentaba los mismos patrones de divergencia en todas las etapas del algoritmo. Por esto, se optó por implementar el control proporcional de la Ecuación (11) para las simulaciones y otro control proporcional con la Ecuación (12) para ejecutar el algoritmo en físico. Con esta modificación, se logró replicar el funcionamiento completo del algoritmo tanto para simulaciones en Webots como en un ambiente físico en el Robotat.

8.2.5. Configuraciones para el escenario

Para clasificar las ejecuciones del algoritmo según las condiciones del escenario, se decidió utilizar la misma convención implementada por José Alejandro Rodríguez [9]. Esta se enfoca en diferenciar los escenarios según las posiciones de las marcas iniciales para los agentes, los obstáculos y el objetivo.

En el Cuadro 4 se muestra la codificación a utilizar para los experimentos. El orden para identificarlos es el siguiente: (Posición inicial de los agentes)-(Obstáculos)-(Objetivo). A continuación, se presenta el ejemplo de clasificación A-D-B, que significa:

- Posicionamiento inicial de los agentes en línea.
- Obstáculos móviles.
- Objetivo en el centro de la mesa de pruebas.

Letra de designación	Posición inicial de los agentes	Obstáculos	Objetivo
A	Línea	Ninguno	Esquina
B	Círculo	1 = Centro 2 = Cerca del objetivo	Centro
C	Aleatorio	Aleatorio	Aleatorio
D	N/A	Móviles	Móvil

Cuadro 4: Configuraciones para el escenario.

8.2.6. Prueba del algoritmo en físico con escenarios previos

Una vez solucionados los problemas anteriores, se realizó las primeras pruebas para replicar algunos de los escenarios físicos realizados por José Alejandro Rodríguez [9] y comprobar el funcionamiento correcto del algoritmo.

El código del supervisor ya cuenta con un sistema de guardado de información. Este almacena en un archivo .npz todos los datos relevantes al final del experimento. Además, con el archivo “data_graphing.py” se generan las gráficas y trayectorias luego de que los agentes se hayan colocado en sus posiciones iniciales.

A continuación, se muestran algunos de los datos almacenados más relevantes:

- Historial durante todo el experimento de:
 - Posiciones y orientaciones de los agentes
 - Posiciones de los obstáculos
 - Posición del objetivo
 - Velocidades de los agentes
- Datos y configuraciones de:

- Ciclos del experimento
- Ciclo en que se logra el objetivo
- Ciclo en que inicia la etapa 1 del algoritmo
- Cantidad de agentes y sus números de marcadores asignados
- Cantidad de obstáculos y sus números de marcadores asignados
- Marcas de posiciones iniciales de los agentes
- Paso de tiempo del programa

Primer escenario

Para el primer escenario, se utilizó la configuración AAA con 2 agentes. En las Figuras 30 y 31 se muestra el escenario en la mesa de pruebas y las trayectorias de los agentes durante el experimento.

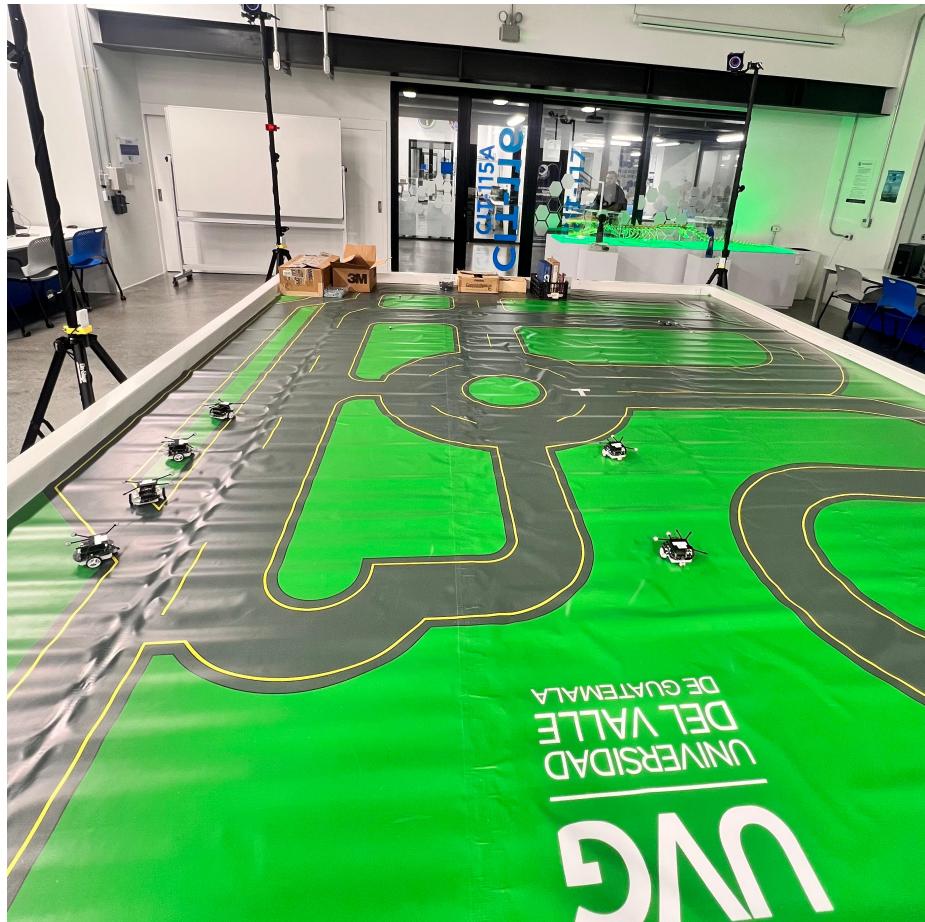


Figura 30: Mesa de pruebas con 2 agentes en el escenario AAA, corrida 1, en físico.

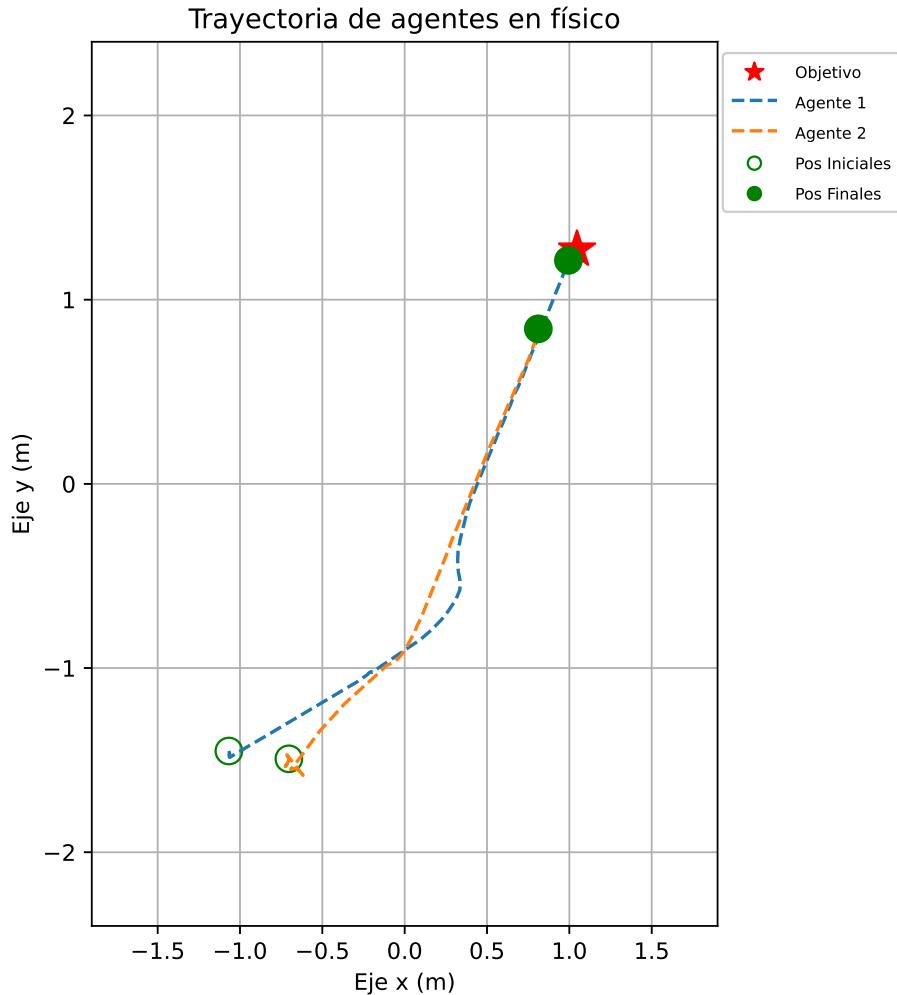


Figura 31: Trayectoria de los 2 agentes en el escenario AAA, corrida 1, en físico.

Segundo escenario

Para el segundo escenario, se utilizó la configuración AAA con 6 agentes. En las Figuras 32 y 33 se muestra el escenario en la mesa de pruebas y las trayectorias de los agentes durante el experimento.



Figura 32: Mesa de pruebas con 6 agentes en el escenario AAA, corrida 1, en físico.

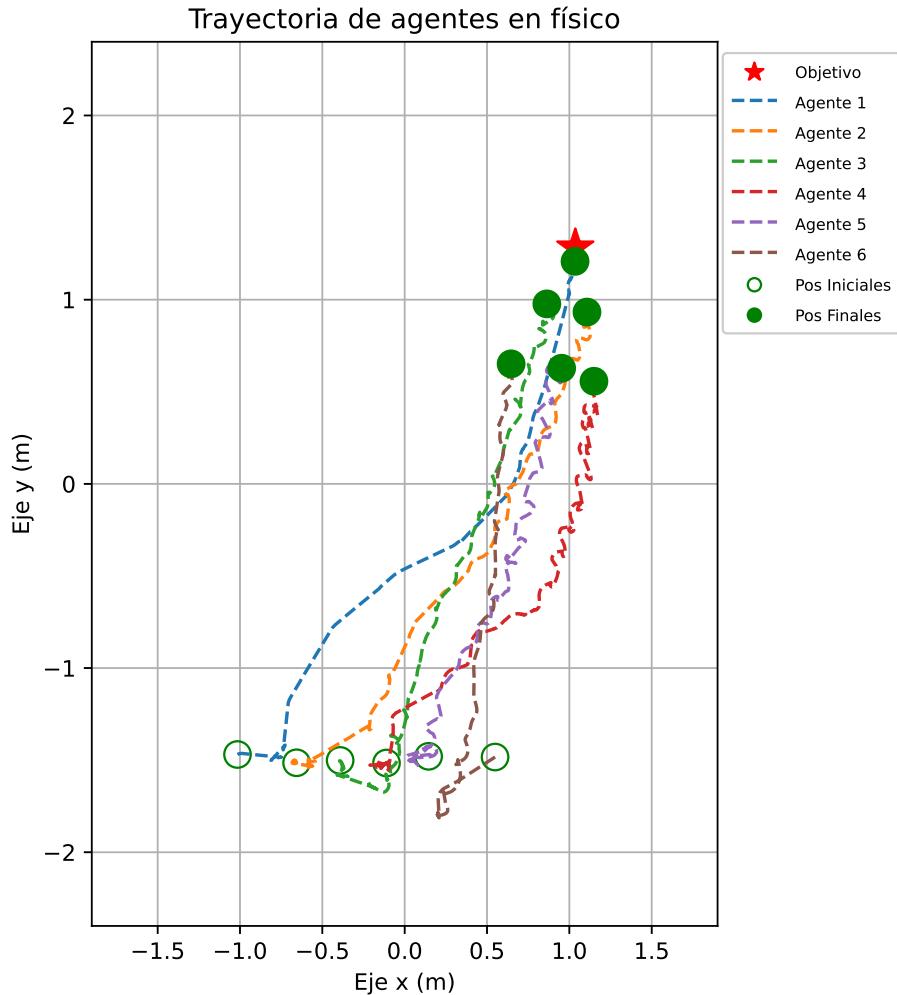


Figura 33: Trayectoria de los 6 agentes en el escenario AAA, corrida 1, en físico.

Tercer escenario

Para el tercer escenario, se utilizó la configuración ACA con 6 agentes. En las Figuras 34 y 35 se muestra el escenario en la mesa de pruebas y las trayectorias de los agentes durante el experimento.

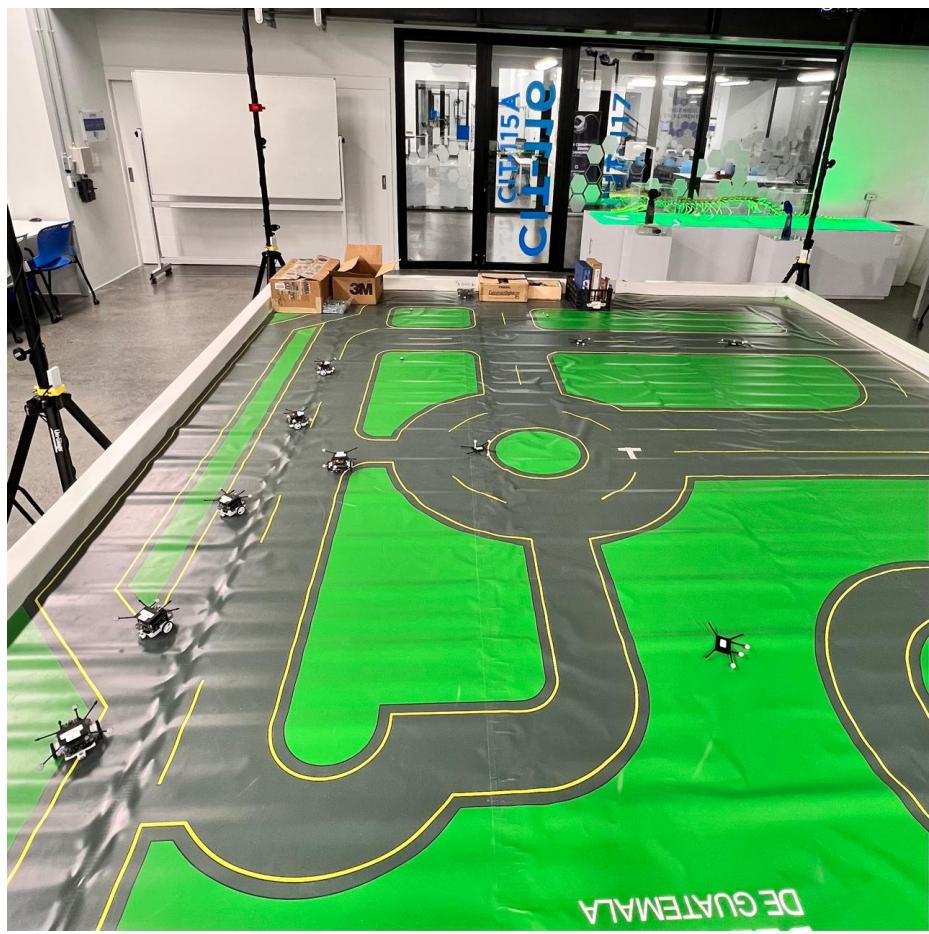


Figura 34: Mesa de pruebas con 6 agentes en el escenario ACA, corrida 1, en físico.

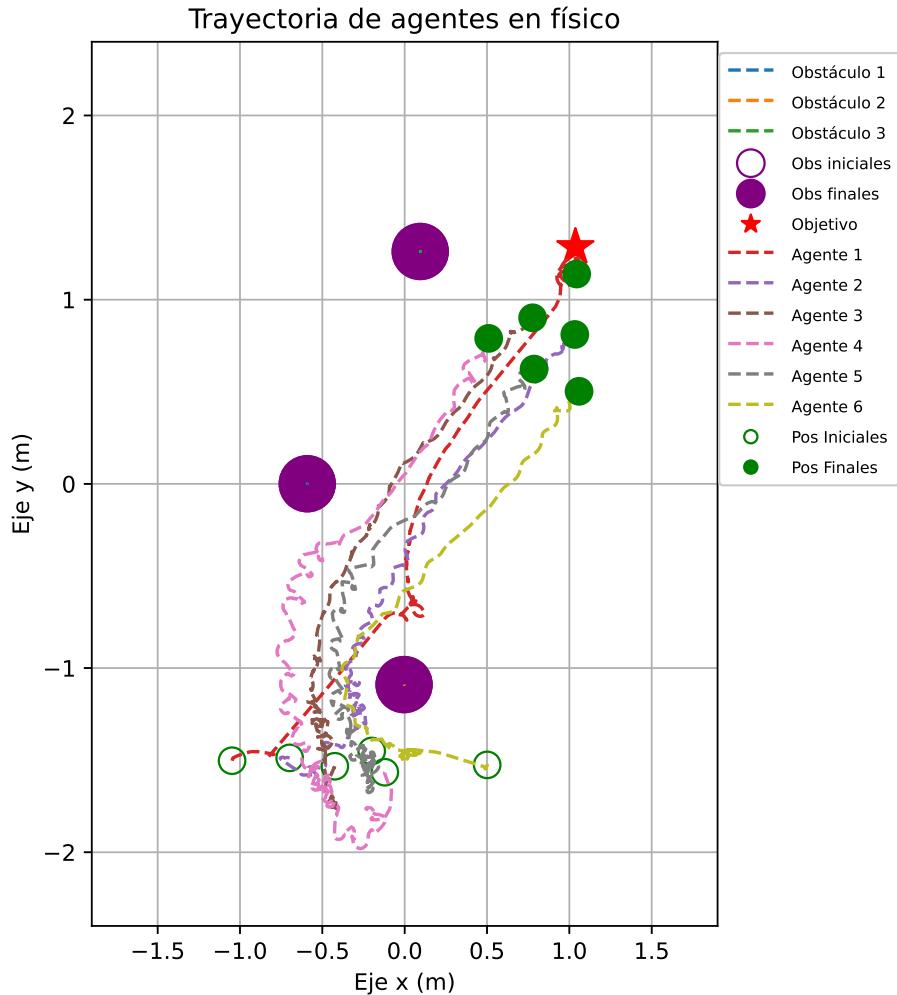


Figura 35: Trayectoria de los 6 agentes en el escenario ACA, corrida 1, en físico.

Cuarto escenario

Para el tercer escenario, se utilizó la configuración ACC con 3 agentes. En las Figuras 36 y 37 se muestra el escenario en la mesa de pruebas y las trayectorias de los agentes durante el experimento.

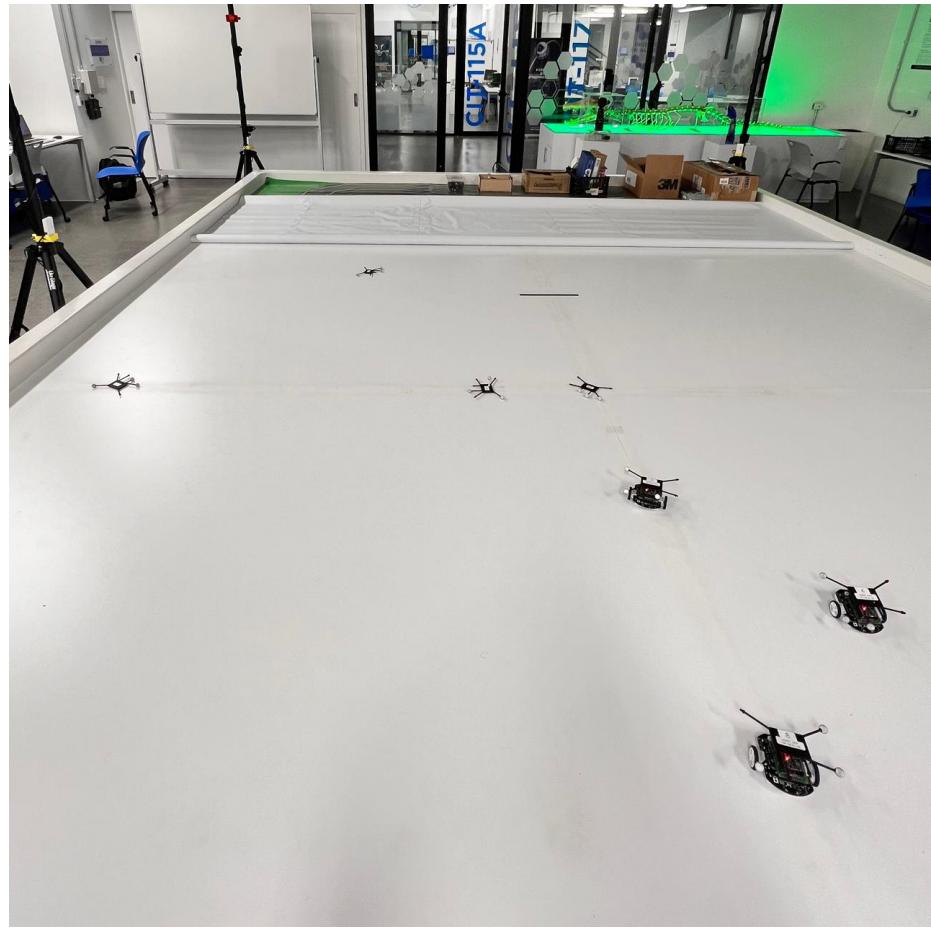


Figura 36: Mesa de pruebas con 3 agentes en el escenario ACC, corrida 1, en físico.

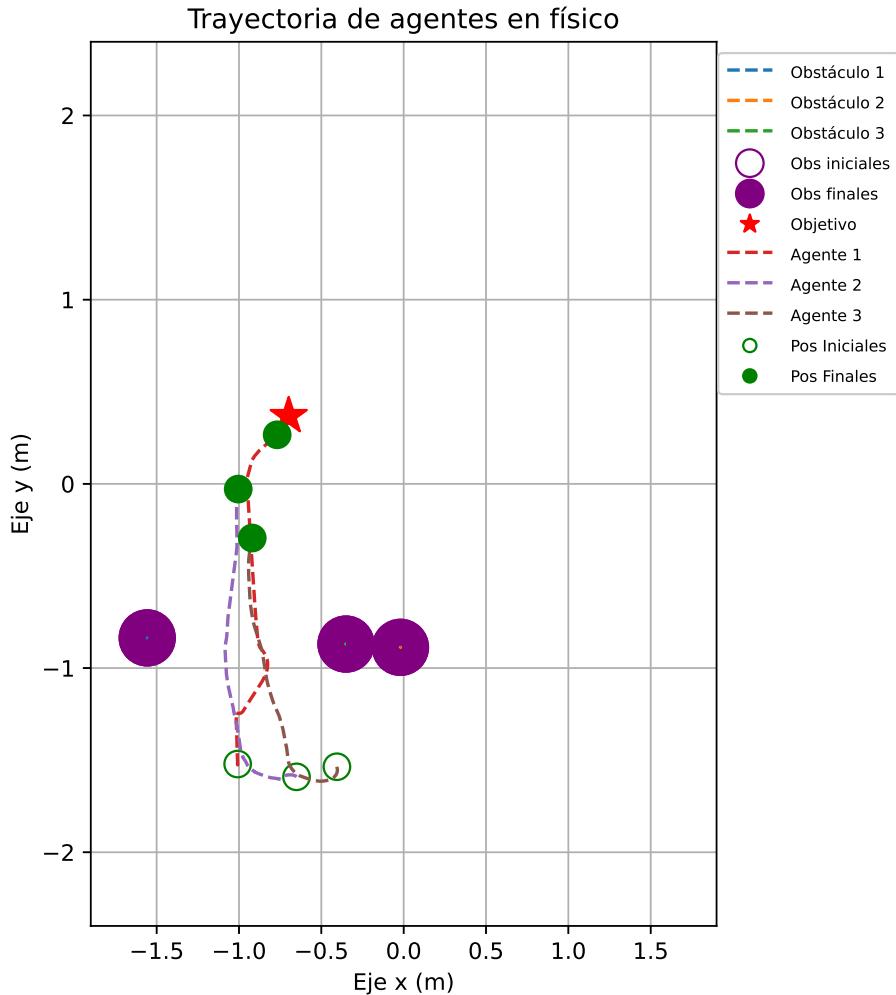


Figura 37: Trayectoria de los 3 agentes en el escenario ACC, corrida 1, en físico.

Quinto escenario

Para el tercer escenario, se utilizó la configuración ACA con 3 agentes. En las Figuras 38 y 39 se muestra el escenario en la mesa de pruebas y las trayectorias de los agentes durante el experimento.



Figura 38: Mesa de pruebas con 3 agentes en el escenario ACA, corrida 1, en físico.

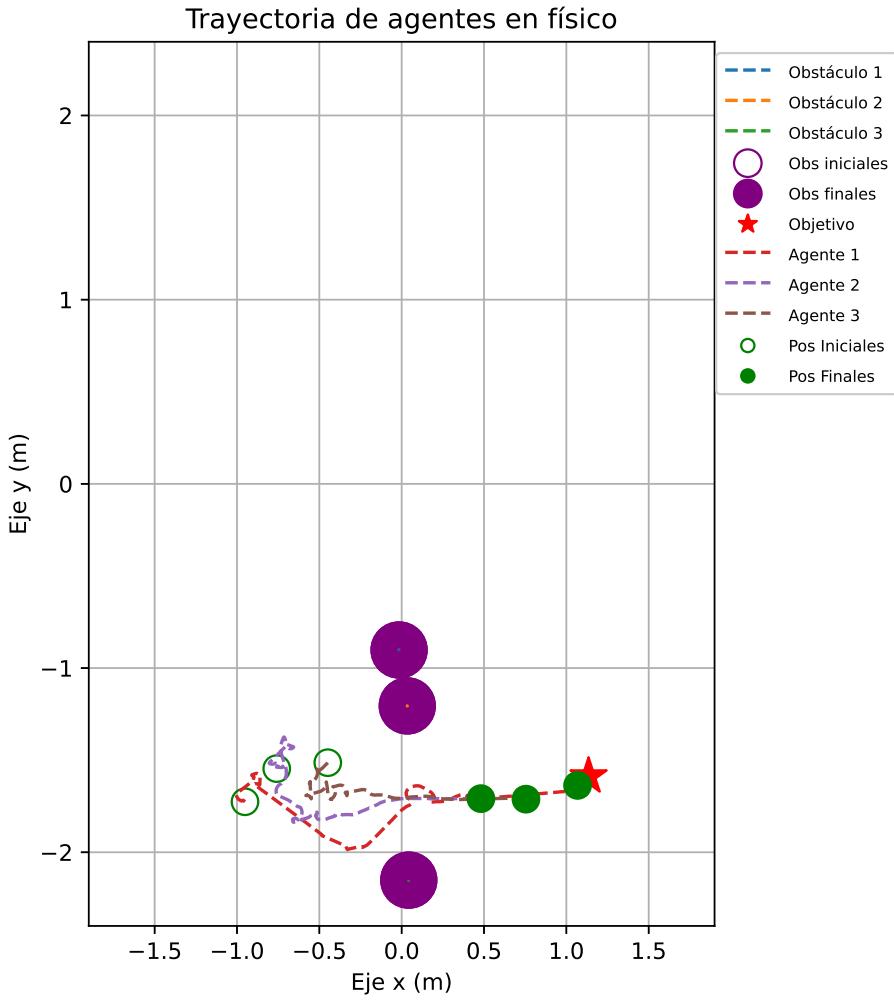


Figura 39: Trayectoria de los 3 agentes en el escenario ACA, corrida 1, en físico.

8.3. Prueba de funcionamiento del algoritmo con obstáculos móviles

Una vez que se validó el funcionamiento del algoritmo utilizando escenarios previos, se puso a prueba la naturaleza dinámica del algoritmo de sincronización y control de formaciones. Como se mencionó anteriormente, se cuenta con el archivo “data_graphing.py” para generar las gráficas y trayectorias del experimento. Sin embargo, este no muestra el recorrido de los obstáculos en movimiento, por lo que se agregó esta función a manera de visualizar la posición inicial, y final, así como las trayectorias de los obstáculos.

A continuación, se presentan los escenarios en los que se puso a prueba el algoritmo con obstáculos móviles en simulaciones y en físico. En todos los casos, se observa que los agentes logran repositionarse y cambiar su trayectoria al moverse los obstáculos, manteniendo su formación mientras siguen el objetivo. Esto es posible gracias a la naturaleza dinámica del algoritmo, que se debe a la manera en que se planteó originalmente la ecuación de

consenso y el cálculo del peso para determinar las velocidades. Además, el algoritmo está constantemente evaluando la posición actual de cada agente y la compara con la posición actual de los obstáculos. Con esto, se ajusta de forma dinámica el peso w y se re calcula la ecuación de consenso para evitar posibles colisiones.

8.3.1. Simulaciones utilizando obstáculos móviles

Para las simulaciones mostradas en las Figuras 40, 41 y 42 se optó por utilizar una configuración ADA con 6 agentes y se movilizó los obstáculos a manera de que estos interfirieran con la trayectoria de la formación durante el seguimiento al objetivo. Cabe mencionar que la movilización de los obstáculos se hizo de manera aleatoria y sin ningún patrón en específico.

Primer escenario

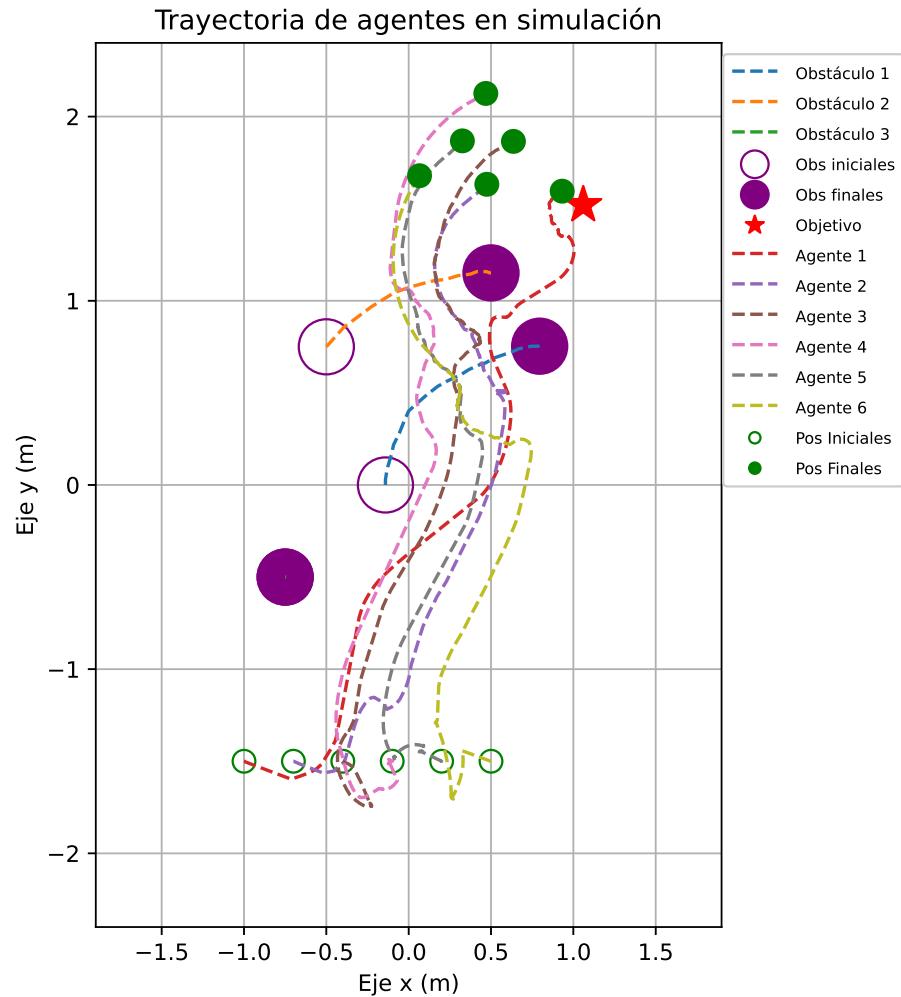


Figura 40: Trayectoria de los 6 agentes en el escenario ADA, corrida 1, en simulación.

Segundo escenario

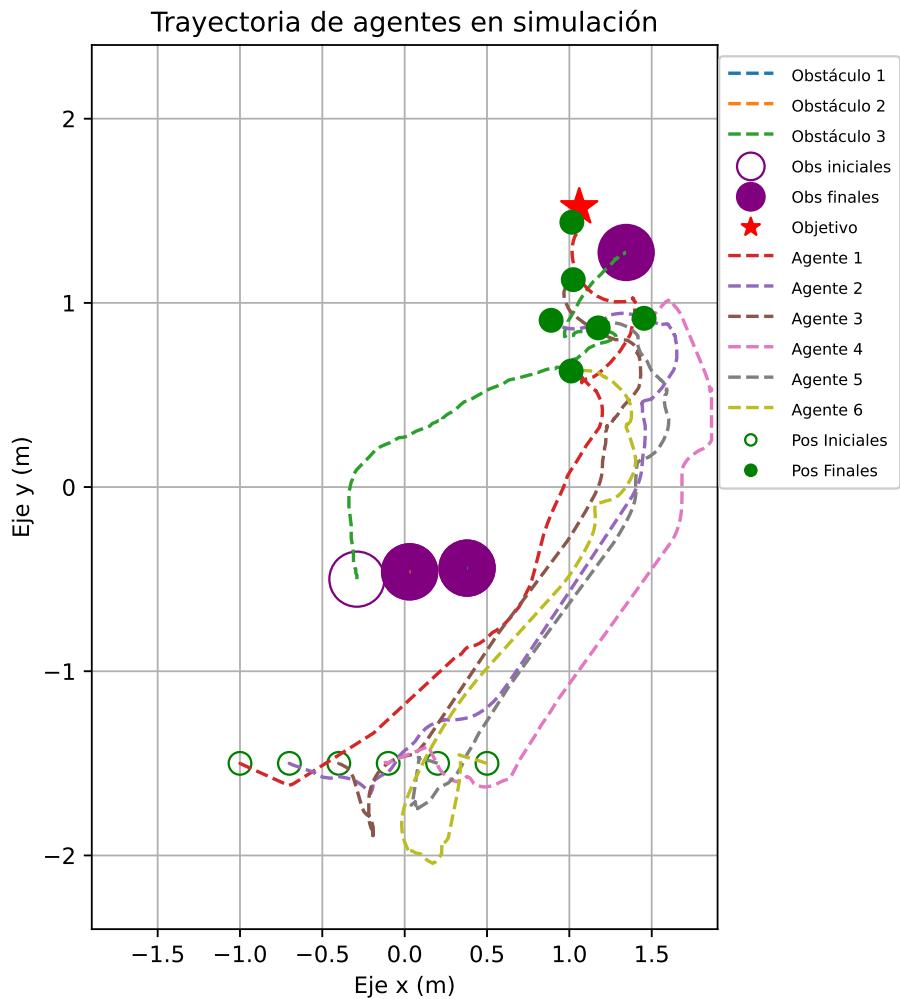


Figura 41: Trayectoria de los 6 agentes en el escenario ADA, corrida 2, en simulación.

Tercer escenario

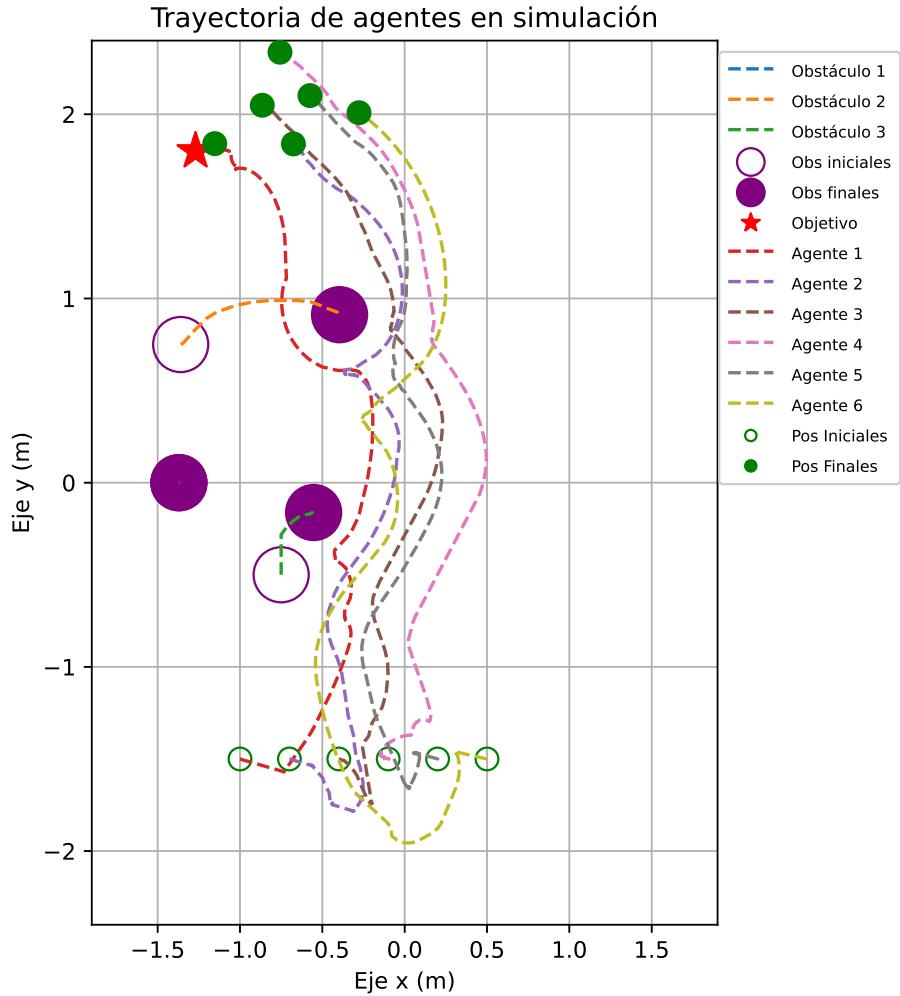


Figura 42: Trayectoria de los 6 agentes en el escenario ADA, corrida 3, en simulación.

8.3.2. Escenarios físicos utilizando obstáculos móviles

A diferencia que con los escenarios en simulaciones, para las pruebas en físico se movilizó los obstáculos a manera de que estos interfirieran o liberaran una trayectoria de la formación durante el seguimiento al objetivo.

Primer escenario

Para este escenario, se movilizaron los obstáculos para que estos dejaran libre la trayectoria de los agentes hacia el objetivo en lugar de interferir en esta. Como se observa en la Figura 43, el agente 1 con la trayectoria roja, pasa sobre la posición donde se encontraba el obstáculo inicialmente. Además, evita colisionar con los obstáculos colocados en sus posiciones finales.

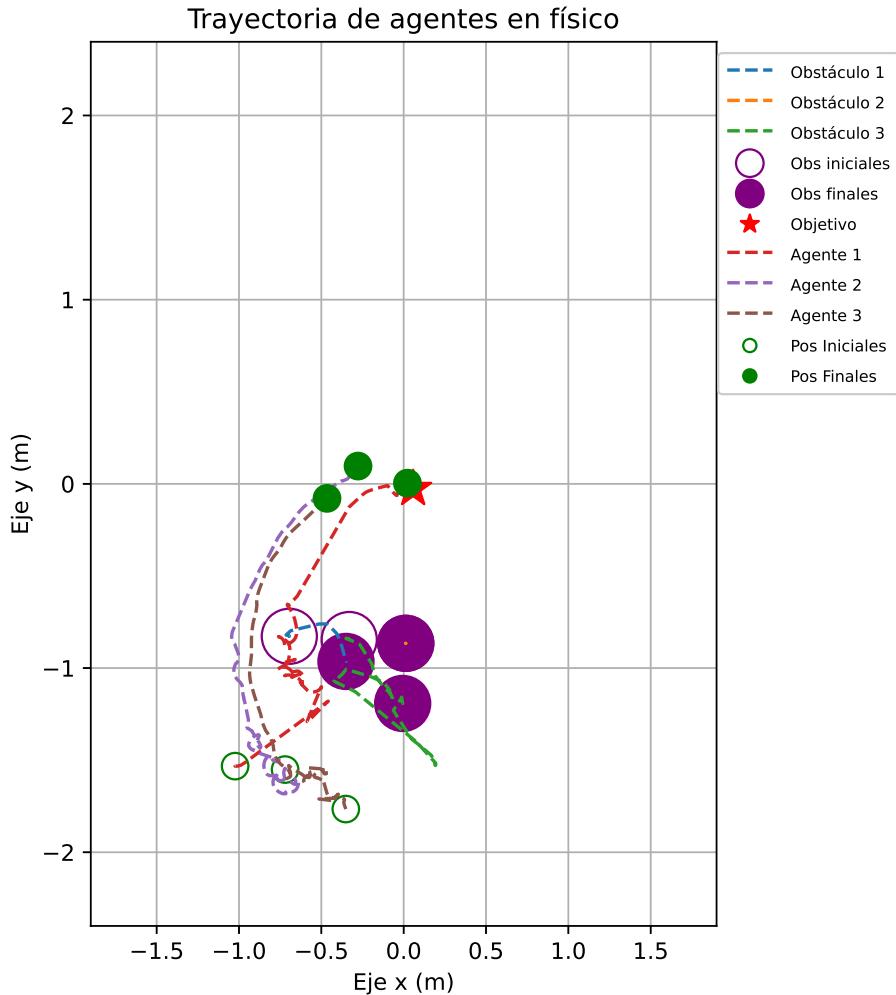


Figura 43: Trayectoria de los 3 agentes en el escenario ADB, corrida 1, en físico.

Segundo escenario

Para este escenario, se planteó el recorrido a manera que los agentes tuvieran que pasar en medio de los obstáculos. Como se observa en la Figura 44, con la trayectoria en verde, el obstáculo se movilizó a lo largo del espacio libre para que este bloqueara el camino hacia el objetivo. Luego se volvió a colocar cercano a su posición inicial. Además, se observa cómo los agentes lograron evitar la colisión y seguir su trayecto una vez que el camino se liberara.

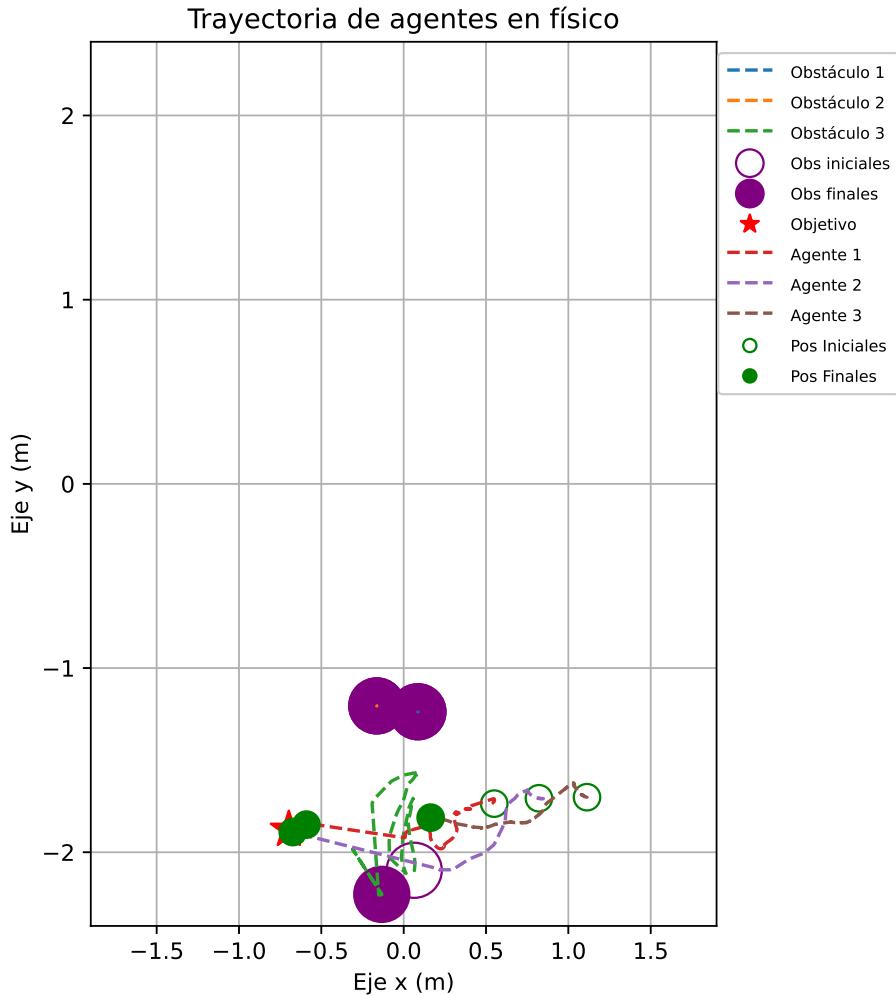


Figura 44: Trayectoria de los 3 agentes en el escenario ADC, corrida 1, en físico.

Tercer escenario

Para este caso, se realizó lo mismo que en el segundo escenario. Como se observa en la Figura 45, con la trayectoria en verde, el obstáculo se movilizó a lo largo del espacio libre a manera que este bloqueara el camino hacia el objetivo. Luego se volvió a colocar cercano a su posición inicial, donde se observa cómo los agentes lograron evitar la colisión y seguir su trayecto una vez que el camino se liberara.

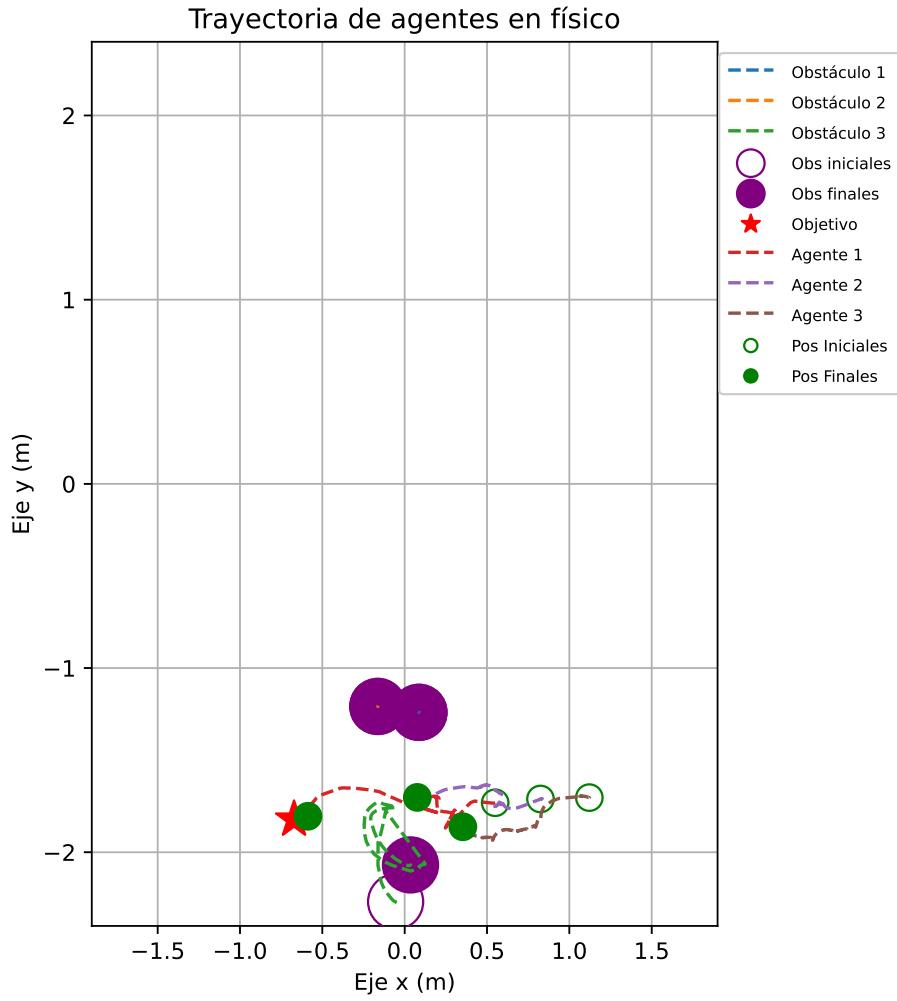


Figura 45: Trayectoria de los 3 agentes en el escenario ADC, corrida 3, en físico.

Cuarto escenario

Para este escenario, se movilizó el obstáculo una sola vez a manera que cambiara de posición e interfiriera con una trayectoria en línea recta de la formación hacia el objetivo. En la Figura 46, se observa que los agentes modificaron su trayectoria evadiendo el obstáculo por un costado y siguieron exitosamente su camino hacia el objetivo.

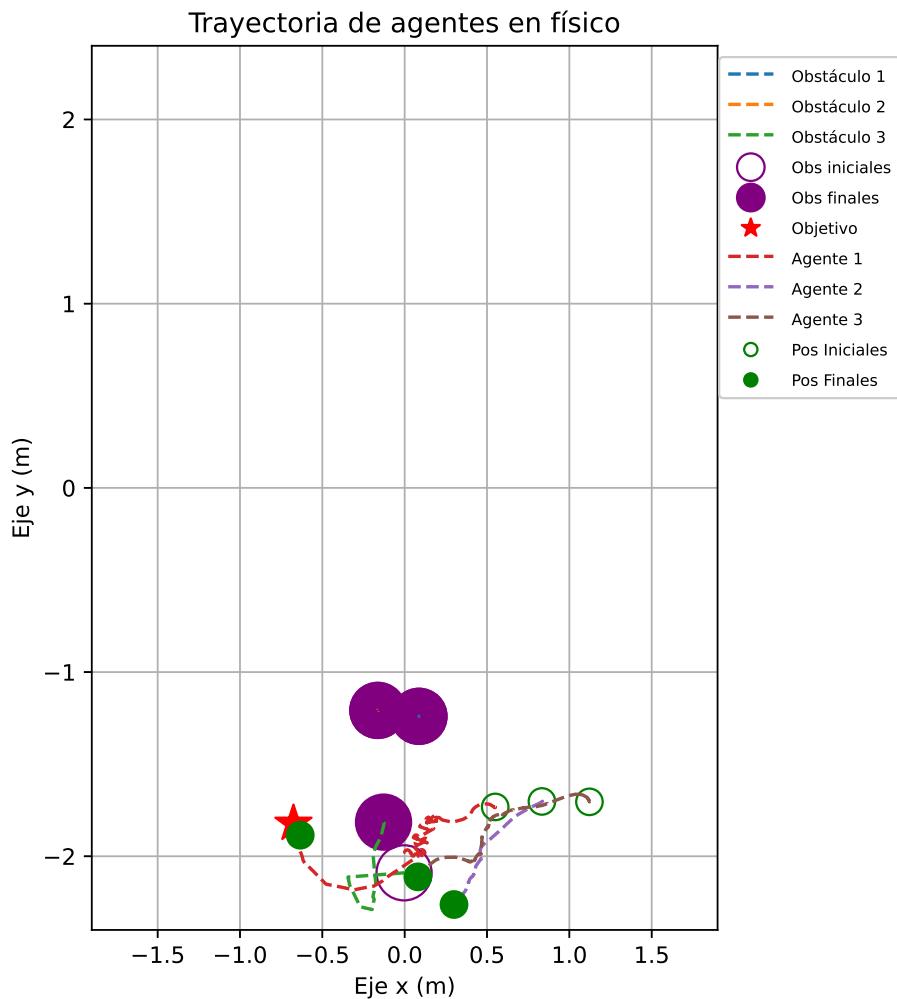


Figura 46: Trayectoria de los 3 agentes en el escenario ADC, corrida 2, en físico.

CAPÍTULO 9

Optimización del algoritmo y su implementación

En este capítulo se describen los puntos de mejora identificados en la implementación del algoritmo de sincronización y control de formaciones. Luego, se detallará cómo se abordó cada uno de ellos empleando técnicas de optimización como la reducción de complejidad computacional y ajuste de parámetros de control.

9.1. Lenguaje de programación y limpieza de código

El primer paso para la optimización fue analizar el lenguaje de programación empleado para los controladores. Se tomó en cuenta tres lenguajes para compararlos en base a las ventajas y desventajas que proponen para su implementación con el estado actual del algoritmo: MATLAB, Python y C. En el Cuadro 5 se describen las ventajas y desventajas que tiene un lenguaje sobre otro.

Lenguaje	Ventajas	Desventajas
MATLAB	<ul style="list-style-type: none"> - Es fácil de usar para simulaciones y prototipos. - Tiene una amplia variedad de herramientas matemáticas. - Es excelente para algoritmos que involucran operaciones matriciales. 	<ul style="list-style-type: none"> - Es un lenguaje propio de MATLAB por lo que requiere comprar licencias de software. - Su eficiencia computacional es menor en cuanto a rendimiento comparado con Python o C.
Python	<ul style="list-style-type: none"> - Es un lenguaje de código abierto. - Tiene una amplia variedad de librerías para optimizar el cálculo numérico como NumPy, SciPy, multithreading o multiprocessing. - Tiene un buen equilibrio entre rendimiento computacional y facilidad de desarrollo. 	<ul style="list-style-type: none"> - Es un lenguaje más eficiente que MATLAB pero menos eficiente que C debido a que es un lenguaje interpretado. - Requiere dependencias externas para lograr un alto rendimiento (como integraciones con C).
C	<ul style="list-style-type: none"> - Tiene un rendimiento superior ya que es un lenguaje compilado de bajo nivel. - Ofrece un control sobre la memoria y el hardware. 	<ul style="list-style-type: none"> - Su complejidad es mucho mayor en cuanto a implementación y mantenimiento. - Tiene una alta probabilidad de fugas de memoria. - Requiere mucho más tiempo de desarrollo y es menos intuitivo. - No es tan flexible para realizar cambios rápidos en algoritmos.

Cuadro 5: Ventajas y desventajas de lenguajes de programación.

Una vez comparados los lenguajes de programación se optó por seguir utilizando Python para el desarrollo de los controladores. Las razones principales fueron que es un lenguaje de código abierto y ofrece un buen equilibrio entre rendimiento computacional y facilidad de desarrollo. Además, ya se cuenta con todo el algoritmo previo desarrollado en Python funcionando en un entorno físico como el Robotat. Esto incluye el desarrollo de los controladores, funciones para cálculos propios del algoritmo de sincronización y control de formaciones y funciones de conexión con el servidor del Robotat y los Pololu 3Pi+. Por lo que, dado el poco tiempo disponible para utilizar el Robotat no es conveniente migrar a un nuevo lenguaje de programación para esta fase de la implementación del algoritmo. Además, python cuenta con librerías como NumPy que está basada en C y otras librerías de comunicación y procesamiento que permiten optimizar el rendimiento computacional.

Finalmente, se realizó un proceso de limpieza y documentación del código para facilitar las iteraciones a realizar. Además, esto permitió eliminar segmentos de código obsoletos, así como identificar otras deficiencias que se mencionarán a continuación.

9.2. Posición de cada agente según el grafo de formación

Durante la restauración del algoritmo desarrollado en fases previas, al implementar la selección manual de marcadores para los agentes, se encontró que anteriormente el número de marcador asignado a cada agente estaba directamente relacionado con la posición que este debe tomar según el grafo de formación. En la Figura 47 se muestra la asignación de posiciones según el grafo de formación para una selección arbitraria de agentes con la implementación original del algoritmo.

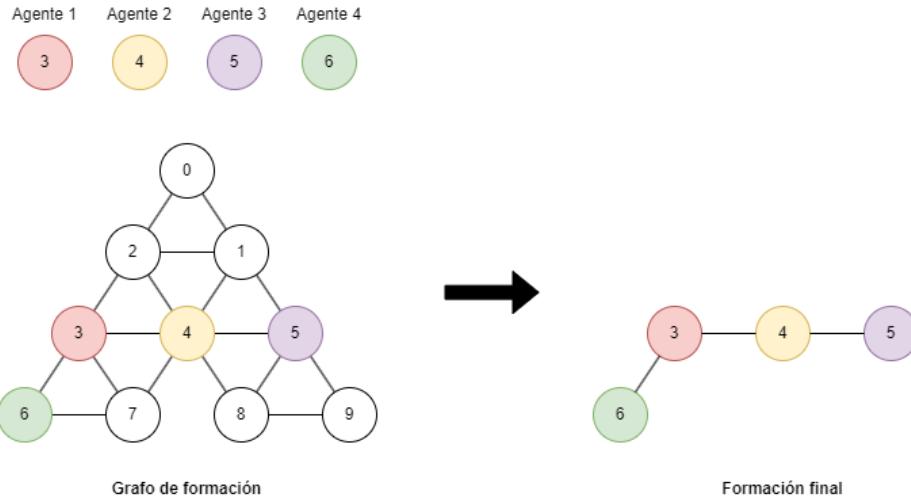


Figura 47: Posiciones de agentes según el grafo de formación con la implementación original del algoritmo.

Al tener una formación como la mostrada anteriormente se vuelve poco intuitiva su visualización a la hora de ejecutar el algoritmo. Por esto, se decidió cambiar la asignación de posición de cada agente a manera que siempre se mantenga la misma estructura del grafo de formación y los agentes se posicionen de manera ascendente tal como se observa en la figura 48.

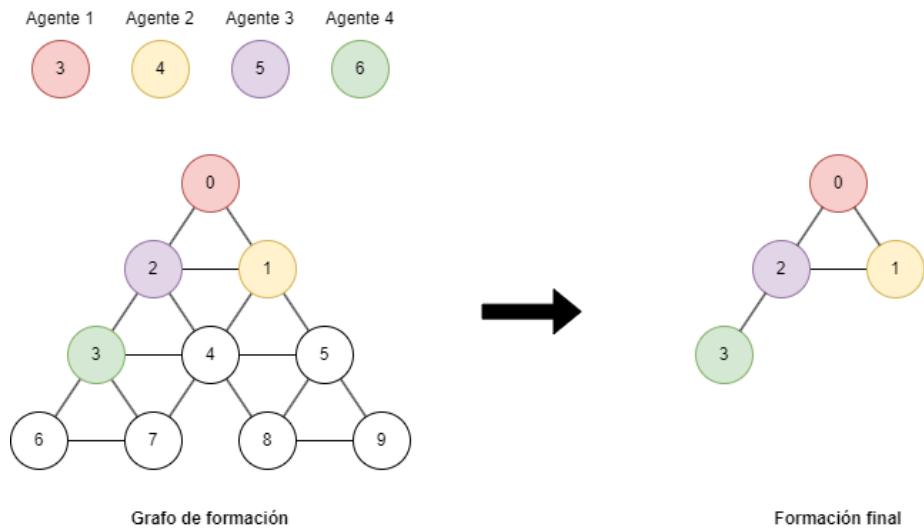


Figura 48: Posiciones de agentes según el grafo de formación con la implementación original del algoritmo.

9.3. Mejorar la eficiencia computacional con NumPy

NumPy es una librería de Python especializada para el cálculo numérico y científico. Destaca por su soporte para trabajar con operaciones vectorizadas o vectores multidimensionales sin realizar bucles. También cuenta con una amplia variedad de funciones matemáticas y estadísticas, así como integración con otras bibliotecas como SciPy. Además, NumPy está implementado en C por lo que permite mejorar la eficiencia computacional para grandes cantidades de datos.

Al revisar los programas de los controladores, se encontró varios segmentos de código que se realizan con ciclos *for* simples y otros anidados. Esto presenta una deficiencia al realizar los cálculos y solicitud de datos al servidor del Robotat por lo que se optó por optimizar dichos procesos utilizando funciones matemáticas y operaciones matriciales con NumPy. A continuación, se detallará los segmentos del algoritmo que se optimizaron implementando operaciones con NumPy y se mostrará la comparativa en cuanto a tiempos de ejecución utilizando la librería “time” de Python y análisis estadísticos.

9.3.1. Aplicar desfases de los marcadores

A continuación se explica los pasos que se realizaban anteriormente para aplicar los desfases de la calibración de marcadores, donde n es la cantidad de marcadores a utilizar.

- Configuración
 1. Cargar el archivo .npy con los desfases de los marcadores.
 2. Solicitar las poses de n marcadores al sevidor del Robotat.
 3. Aplicar el desfase de cada marcador con un ciclo *for* de n iteraciones.
- Ciclo principal
 1. Solicitar las poses de n marcadores al servidor del Robotat.
 2. Aplicar el desfase de cada marcador con un ciclo *for* de n iteraciones.

Sin embargo, el implementar ciclos *for* dentro del ciclo principal significa un aumento de tiempo computacional que es más notorio al aumentar la cantidad de marcadores a utilizar. Para optimizar el proceso se optó por aplicar los desfases a cada marcador utilizando operaciones matriciales e implementando únicamente una resta de matrices dentro del ciclo principal, por lo que ahora el proceso es el siguiente:

- Configuración
 1. Cargar el archivo .npy con los desfases de los marcadores.
 2. Almacenar los desfases en la cuarta columna de una matriz de ceros de tamaño $n \times 6$.
 3. Solicitar las poses de los marcadores al sevidor del Robotat.

4. Aplicar el desfase de los marcadores con una resta de la matriz con los desfases a la matriz con las poses de los marcadores.
- Ciclo principal
 1. Solicitar las poses de los marcadores al servidor del Robotat
 2. Aplicar el desfase de los marcadores con una resta de la matriz con los desfases a la matriz con las poses de los marcadores.

Una vez implementada la optimización, se realizó un análisis estadístico con mil muestras donde se tomó el tiempo que toma aplicar los desfases a los marcadores con un ciclo *for* y con una resta de matrices con NumPy.

Para la toma de muestras, se evaluó diferentes cantidades de marcadores a los que se les aplicaría el desfase, siendo estos 5, 10, 15 y 20. En los Cuadros 6 y 7 se muestra la media y desviación estándar del tiempo que toma aplicar los desfases.

En la Figura 49 se observa que la optimización con NumPy presenta una reducción notable en el tiempo de procesamiento, la cual es aún más significativa a medida que aumenta la cantidad de marcadores a utilizar. Por otro lado, los tiempos al aplicar ciclos *for* incrementan de manera proporcional y más pronunciada con el aumento de los marcadores, mientras que con NumPy el crecimiento es más moderado ya que la pendiente de la curva es dos órdenes de magnitud menor.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
5	1000	0.002802	0.000308
10	1000	0.005353	0.000469
15	1000	0.008187	0.001654
20	1000	0.010485	0.000843

Cuadro 6: Media de tiempo y desviación estándar para aplicar desfases de marcadores con ciclos for.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
5	1000	0.000734	0.000119
10	1000	0.000756	0.000113
15	1000	0.000794	0.000225
20	1000	0.000837	0.000147

Cuadro 7: Media de tiempo y desviación estándar para aplicar desfases de marcadores con NumPy.

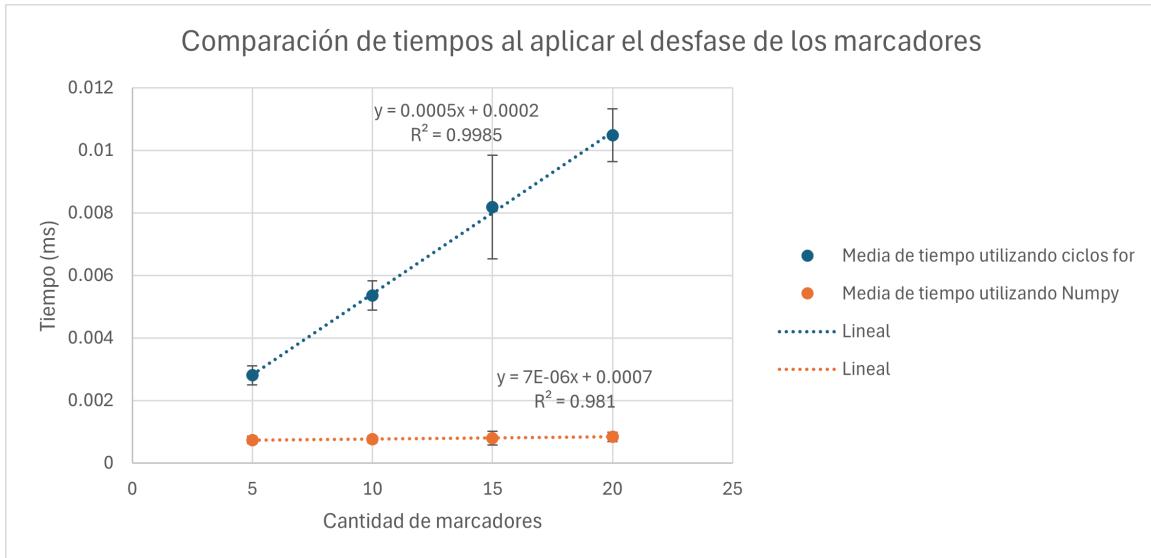


Figura 49: Gráfica con medias de tiempo para aplicar desfases de marcadores con ciclos *for* y operaciones con NumPy.

9.3.2. Cálculo de la distancia entre agentes

Para el cálculo de distancia entre agentes se tiene una función llamada “DistBetweenAgents” dentro del archivo “funciones.py”. Esta utiliza dos ciclos *for* anidados para calcular la norma de distancia basada en la distancia en *x* y *y* de cada agente hacia sus vecinos más próximos. El cálculo de la distancia se realiza entre los agentes *i* y *j* y luego para *j* e *i*, esto produce una matriz simétrica por lo que se puede realizar únicamente el cálculo de una mitad y luego duplicarla. Para esto, se creó una versión optimizada de la función llamada “DistBetweenAgentsOptimized”. En esta, únicamente se calcula la diferencia de posiciones aprovechando el *broadcasting* en NumPy y luego se obtiene la norma de la matriz de distancias utilizando la función “linealg.norm”.

En los Cuadros 8 y 9 se muestra la media y desviación estándar del tiempo que toma en realizar los cálculos con la función original y la optimizada. Para esto, se tomó mil muestras aumentando la cantidad de agentes desde 1 hasta 10.

En la Figura 50 se observa un incremento exponencial en el tiempo de procesamiento al calcular las distancias entre los agentes conforme se aumenta el número de estos en la formación. Además, la optimización con NumPy presenta un incremento significativamente menor. Esto se confirma al observar que el exponente de la ecuación que describe el comportamiento utilizando ciclos *for* es 1.6982, mientras que el exponente al utilizar NumPy es 0.1775.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
1	1000	0.00213	0.000343
2	1000	0.005569	0.006185
3	1000	0.009883	0.000914
4	1000	0.016406	0.001267
5	1000	0.026257	0.005967
6	1000	0.036697	0.00706
7	1000	0.051342	0.003965
8	1000	0.062752	0.012055
9	1000	0.079696	0.009144
10	1000	0.098319	0.009456

Cuadro 8: Media de tiempo y desviación estándar para calcular la distancia entre agentes con ciclos *for*.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
1	1000	0.0061	0.001477
2	1000	0.008134	0.001457
3	1000	0.008187	0.007005
4	1000	0.008354	0.006599
5	1000	0.008918	0.00158
6	1000	0.008835	0.001695
7	1000	0.009344	0.001726
8	1000	0.009559	0.004158
9	1000	0.009581	0.002898
10	1000	0.009614	0.002514

Cuadro 9: Media de tiempo y desviación estándar para calcular la distancia entre agentes con NumPy.

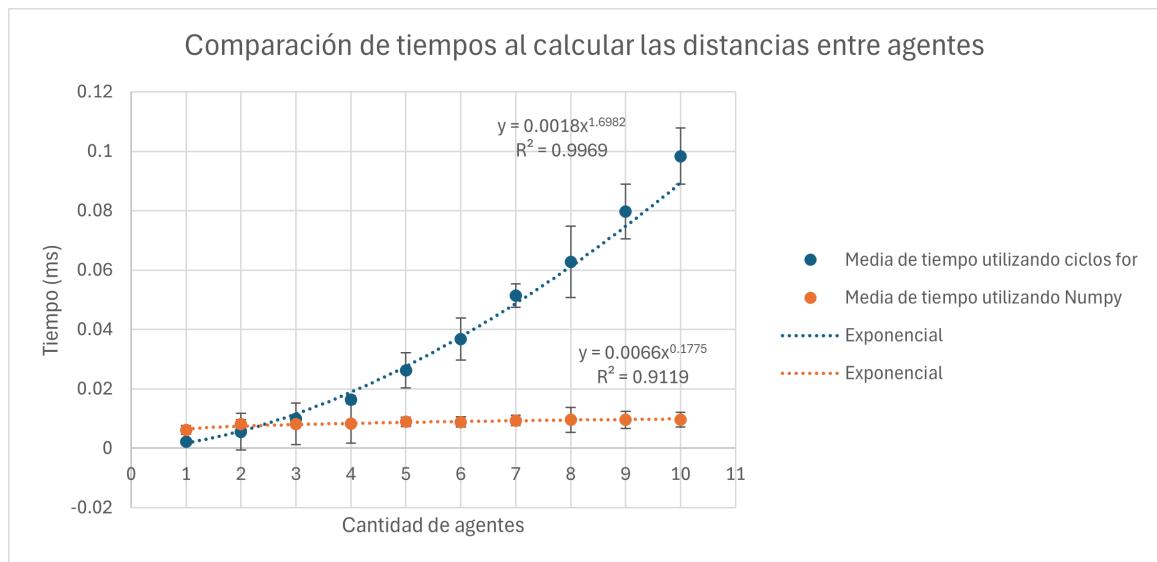


Figura 50: Gráfica con medias de tiempo para calcular la distancia entre agentes con ciclos *for* y operaciones con NumPy.

9.3.3. Cálculo del error de formación

Para calcular el error de formación se utiliza la función “FormationError” dentro del archivo “funciones.py”. Esta función utiliza dos ciclos *for* anidados para calcular el error cuadrático medio entre la formación actual y la deseada utilizando como parámetros las matrices de adyacencia de ambas formaciones. Dado que ambas matrices son cuadradas y, al considerar únicamente los agentes de interés, se convierten en matrices del mismo tamaño, se desarrolló una versión optimizada de la función llamada “FormationErrorOptimized”. Esta versión emplea la función “square” de NumPy que eleva al cuadrado cada elemento de la matriz y luego calcula la media utilizando la función “mean”.

En los Cuadros 10 y 11 se muestra la media y desviación estándar del tiempo que toma en realizar el cálculo del error cuadrático medio con la función original y la optimizada. Para esto, se tomó mil muestras aumentando la cantidad de agentes desde 1 hasta 10.

En la Figura 51 se muestra una gráfica en la que se evidencia un crecimiento exponencial en el tiempo de procesamiento al calcular las distancias entre los agentes a medida que se aumenta el número de agentes en la formación. Por otro lado, la optimización con Numpy, presenta un aumento significativamente menor. Además, esto se demuestra ya que el exponente de la ecuación que describe el comportamiento utilizando ciclos *for* es 1.4753, mientras que el exponente para la optimización con NumPy es 0.112.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
1	1000	0.005665	0.000809
2	1000	0.01101	0.000895
3	1000	0.018721	0.008085
4	1000	0.029093	0.00714
5	1000	0.043071	0.003761
6	1000	0.058559	0.008292
7	1000	0.075147	0.010952
8	1000	0.103554	0.008648
9	1000	0.128032	0.011905
10	1000	0.149833	0.013256

Cuadro 10: Media de tiempo y desviación estándar para calcular el error de formación con ciclos *for*.

Cantidad de marcadores	Número de muestras	Media de tiempo (ms)	Desviación estándar (ms)
1	1000	0.016538	0.007916
2	1000	0.020229	0.002389
3	1000	0.020081	0.003186
4	1000	0.020134	0.003423
5	1000	0.020714	0.003501
6	1000	0.021065	0.00453
7	1000	0.021143	0.005589
8	1000	0.022082	0.003146
9	1000	0.022348	0.00473
10	1000	0.022455	0.024888

Cuadro 11: Media de tiempo y desviación estándar para calcular el error de formación con NumPy.

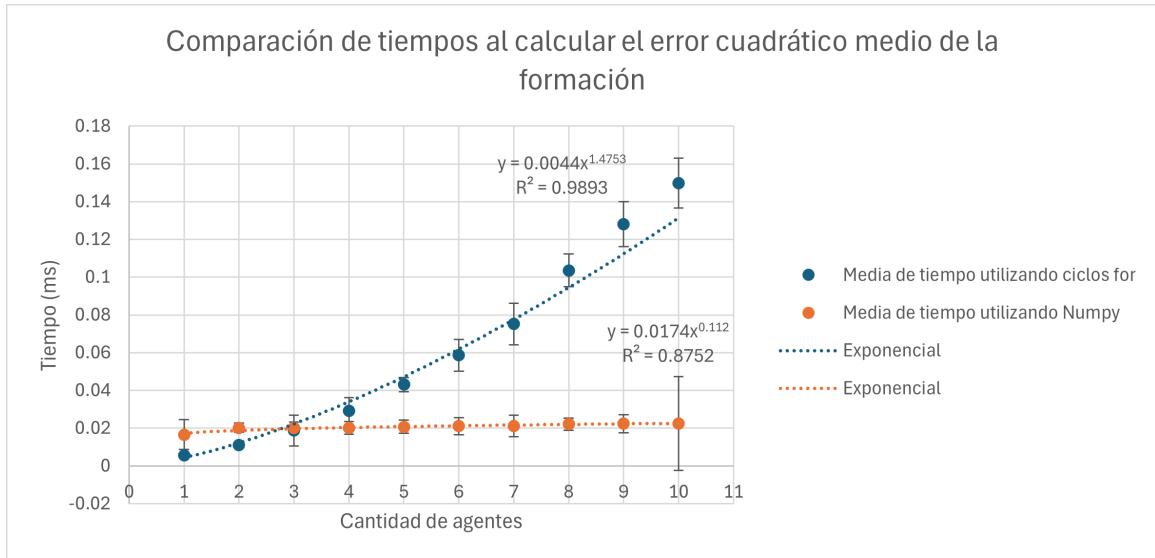


Figura 51: Gráfica con medias de tiempo para calcular el error de formación con ciclos *for* y operaciones con NumPy.

9.4. Ajuste de parámetros del algoritmo

CAPÍTULO 10

Validación del algoritmo optimizado en escenarios físicos con obstáculos móviles

CAPÍTULO 11

Conclusiones

CAPÍTULO 12

Recomendaciones

CAPÍTULO 13

Bibliografía

- [1] M. Ekelhof y G. Persi, *Robótica de enjambre*, 2023. dirección: https://unidir.org/wp-content/uploads/2023/05/UNIDIR_Swarms_SinglePages_web_SP.pdf.
- [2] R. Solís, *Enjambres de robots y sus aplicaciones en la exploración y comunicación*, 2019. dirección: <http://hdl.handle.net/2238/13120>.
- [3] L. Burrows, *Robotic swarm swims like a school of fish*, 2021. dirección: <https://wyss.harvard.edu/news/robotic-swarm-swims-like-a-school-of-fish/>.
- [4] J. Maderer, *Robotarium: A Robotics Lab Accessible to All*, 2017. dirección: <https://news.gatech.edu/archive/features/robotarium-robotics-lab-accessible-all.shtml>.
- [5] P. Barrera, *16 datos sobre el Robotat*, 2023. dirección: <https://noticias.uvg.edu.gt/datos-robotat-habitat-robotica-cit-116/>.
- [6] P. Barrera, *Cuando la física granular y la robótica se apoyan una a otra*, 2022. dirección: <https://noticias.uvg.edu.gt/robotat-robotario-fisica-granular-robotica/>.
- [7] J. Menéndez, “Validación de los algoritmos de robótica de enjambre Particle Swarm Optimization y Ant Colony Optimization con sistemas robóticos físicos en el ecosistema Robotat,” Tesis de licenciatura, Universidad Del Valle de Guatemala, 2023.
- [8] A. 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.
- [9] J. Rodríguez, “Validación de un algoritmo de inteligencia de enjambre enfocado en sincronización y control de formaciones de sistemas robóticos multi-agente en un entorno físico,” Tesis de licenciatura, Universidad Del Valle de Guatemala, 2023.
- [10] S. Chatterjee, *Part 1; Swarm Robots: Definition, System and Cycles*, 2017. dirección: <https://csoham.com/2017/08/09/part-1-swarm-robots-definition-system-and-cycles/>.

- [11] Leotronics, *Interacción de los enjambres y robótica de grupo en la resolución de diversas tareas*. dirección: <https://leotronics.eu/es/nuestro-blog/interaccion-de-los-enjambres-y-robotica-de-grupo-en-la-resolucion-de-diversas-tareas> (visitado 2024).
- [12] B. Moreno y J. Hernández, “Control centralizado y descentralizado de edificaciones mediante acristalamientos activos,” *Revista de Investigación “Pensamiento Matemático”*, vol. 7, n.º 1, págs. 19-38, 2017.
- [13] V. Noble, *Teoría de grafos*, 2022. dirección: https://rpubs.com/Yelky99/Tgrafos_pmai.
- [14] A. Ruiz-Ruano y J. López, “Modelos Gráficos y Redes en Psicología,” *Revista de Historia de la Psicología*, vol. 41, n.º 4, págs. 24-33, 2020.
- [15] F. Franco, “Aspectos algebraicos en Teoría de Grafos,” Tesis de licenciatura, Universidad de Sevilla, España, 2016.
- [16] L. Crespo, *Matroides y rigidez de grafos*, 2020. dirección: <http://hdl.handle.net/10902/20492>.
- [17] L. Krick, “Application of Graph Rigidity in Formation Control of Multi-Robot Networks,” Tesis de maestría, Universidad de Toronto, 2007.
- [18] M. Mesbahi y M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010, ISBN: 9780691140612. dirección: https://www.ebook.de/de/product/10047947/mehran_mesbahi_magnus_egerstedt_graph_theoretic_methods_in_multiagent_networks.html.
- [19] N. Nise, *Control system engineering*. Benjamin/Cummings, 1995.
- [20] M. Zea, *Lectura 12: Control de robots móviles con ruedas*. (visitado 2024).
- [21] MathWorks, *MATLAB - El lenguaje del cálculo técnico*. dirección: <https://la.mathworks.com/products/matlab.html> (visitado 2024).
- [22] Cyberbotics, *Webots*. dirección: <https://cyberbotics.com/> (visitado 2024).
- [23] J. Aguilar y E. Leiss, *Introducción a la Computación Paralela*. Universidad de los andes Mérida-Venezuela, 2004, ISBN: 980-12-0752-3. dirección: https://gc.scalahed.com/recursos/files/r161r/w25041w/introduccionalacomputacionparalela_S5.pdf.
- [24] OptiTrack, *Primex 41*. dirección: <https://optitrack.com/cameras/primex-41/> (visitado 2024).
- [25] Pololu, *3pi+ 32U4 OLED Robot - Standard Edition (30:1 MP Motors), Assembled*. dirección: <https://www.pololu.com/product/4975> (visitado 2024).