# Evaluación de algoritmos de control de cobertura multi-agente dentro del ecosistema Robotat para futuras aplicaciones de medición de contaminación en cuerpos de agua

Christian Gerardo Carazo Viau

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Evaluación de algoritmos de control de cobertura multi-agente dentro del ecosistema Robotat para futuras aplicaciones de medición de contaminación en cuerpos de agua

Trabajo de graduación presentado por Christian Gerardo Carazo Viau
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Evaluación de algoritmos de control de cobertura multi-agente dentro del ecosistema Robotat para futuras aplicaciones de medición de contaminación en cuerpos de agua

Trabajo de graduación presentado por Christian Gerardo Carazo Viau
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

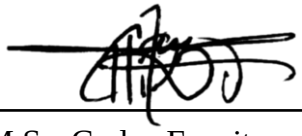
2025

Vo.Bo.:

(f) _____

M. Sc. Carlos Esquit

Tribunal Examinador:

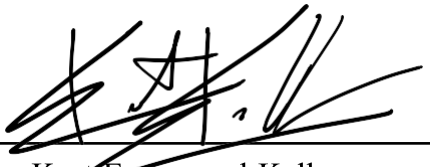


(f) _____

M.Sc. Carlos Esquit

(f) _____

M. Sc. Miguel Enrique Zea Arenales



(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025.

# Prefacio

Desde mi tiempo en el colegio he querido poder hacer una investigación relacionada con la contaminación en el lago de Atitlán. Yo sé que a este trabajo todavía le faltan varias etapas para poder llegar a ser implementado como tal en el lago, pero al menos me satisface saber que pude hacer algo sobre este tema tan interesante.

Espero que los desvelos, las idas a los horarios de atención y la extensa preguntadera a mi asesor, MSc. Miguel Zea, hayan valido la pena y que esta investigación cumpla con las expectativas que tengan. Por esto es que quisiera agradecerle a mi asesor por apoyarme tanto durante todo el proceso y por haberme compartido de su amplio conocimiento para la creación y finalización de este trabajo.

Asimismo, al ser esta la culminación de mi trabajo dentro de la UVG, me gustaría agradecerle a todas las personas que me apoyaron en el transcurso e hicieron que todo este esfuerzo valiera la pena:

Primero, a mi mamá, por siempre estar ahí para mí y apoyarme en todo momento, en las buenas y en las malas, haciendo hasta lo imposible para que saliera adelante. Dándome palabras de aliento cuando más lo necesitaba, preguntándome cómo me había ido aunque estuvieses cansada del trabajo y por darme de tu amor incondicional. Te admiro mucho y te dedico este trabajo.

A mi hermana, por también estar siempre ahí para mí e irme a traer el último semestre de la carrera. Cada vez que necesité ayuda con algo tú siempre estuviste ahí a mi lado. Siempre encontrabas una solución aunque la situación se viese imposible y siempre me diste de tu amor incondicional.

A mi tío Alex Viau, por ser mi inspiración en la vida desde que tengo memoria y por haberme inspirado en estudiar esta carrera que tanto he llegado a amar. Durante toda mi carrera me apoyaste, ya fuese con un proyecto o con algo tan simple como ser alguien con quién hablar sobre lo que había visto en clase. Siempre estaré agradecido contigo por haberme acompañado de universidad en universidad para encontrar la carrera que más me gustara.

A mi tía Ana María Andrade, por siempre apoyarme y darme palabras de aliento cuando más lo necesitaba. Junto con el tío Alex me apoyaron durante toda la carrera, estando pendiente de cómo iba y ayudándome con lo que necesitase sin importar la hora.

A mi tía Maribel Viau, porque sin ti literalmente no hubiese podido estudiar aquí y porque siempre te has preocupado por mí. Desde el momento que dije que quería estudiar en esta universidad me apoyaste y siempre voy a estar agradecido por haberme ayudad a cumplir este sueño.

A Raúl Dacaret, por apoyerme desde el inicio y ayudarme a poder seguir mi sueño de estudiar esta carrera en la UVG. Recuerdo el día que llegué a pedirte ayuda para poder empezar y sin pensarlo me apoyaste con todos los trámites. Sin tu ayuda no me estaría graduando este año ni en esta universidad.

A mis amigos y amigas, por hacer que valiera la pena ir cada día a las clases y hacerlas mil veces mejor con su presencia. Muchas veces no tenía la energía para salir a la universidad, pero pensar en que podría convivir con ustedes me animaba a llegar. Me apoyaron muchísimo académica y emocionalmente, tuvieron un impacto significativo en mi vida y por eso estaré siempre agradecido.

# Índice

# Lista de figuras

# Resumen

El objetivo principal de este trabajo fue implementar, contrastar y validar algoritmos de control de cobertura multi-agente básicos para redes de sensores móviles. Utilizando como inspiración la contaminación en el lago de Atitlán, se planteó una problemática de control de cobertura para censar y monitorear un área de trabajo. Como primer paso para encontrar una solución, se realizó una investigación de distintos algoritmos de control de cobertura multi-agente.

Se seleccionó uno basado en diagramas de Voronoi debido a su robustez para el control de cobertura, adaptibilidad a sistemas dinámicos y capacidad para implementar puntos de concentración. Una vez finalizado esto, se prosiguió con el desarrollo de un entorno de simulación simple pero representativo, el cual utilizó Matlab como su lenguaje de programación. Este entorno permitió ajustar distintos parámetros, logrando simular hasta 1,000 agentes sin problemas, convergiendo a partir de los 8 agentes y modelando diferentes escenarios según las necesidades del área de trabajo.

Luego, se llevó esta solución al ecosistema Robotat, utilizando robots Pololu 3PI+. Se calibró e implementó un controlador PID de acercamiento exponencial que permitió a los agentes alcanzar sus posiciones objetivo con un margen de error de posición del 10 %. Posteriormente, se incorporaron puntos de concentración para imitar los focos de contaminación por cianobacterias. Estos puntos fueron modelados como gradientes de decaimiento exponencial, de manera que las zonas más críticas son priorizadas por los agentes sin que se descuide la cobertura general.

Finalmente, el algoritmo fue simulado sobre una proyección del lago de Atitlán. Se simularon varios puntos de concentración y el movimiento de 20 agentes iniciando desde posiciones aleatorias, observándose cómo se distribuían, asignándose al menos un agente por foco de contaminación y manteniéndose la cobertura en áreas no contaminadas.

Este trabajo crea una base sólida para una futura implementación en cuerpos de agua. No obstante, es relevante que se investiguen distintas plataformas robóticas y protocolos de comunicación que sean capaces de funcionar bajo las condiciones más extremas vistas en cuerpos de agua.

# Abstract

The primary goal of this project was to implement, analyze, and validate fundamental multi-agent coverage control algorithms for mobile sensor networks. Inspired by the issue of pollution in Lake Atitlán, the study addressed a coverage control problem aimed at surveying and monitoring a designated area. The initial step involved researching various multi-agent coverage control algorithms.

A Voronoi-based approach was selected due to its robustness in coverage control, adaptability to dynamic systems, and ability to implement concentration points. Following this, a simple yet representative simulation environment was developed using MATLAB as the programming platform. This simulation environment allowed fine-tuning of various parameters and successfully simulated up to 1,000 agents, with convergence beginning at eight agents. Different scenarios were modeled to meet the specific needs of the target area.

The solution was then implemented in the Robotat ecosystem using Pololu 3PI+ robots. A PID controller with exponential convergence was calibrated and applied, enabling agents to reach their target positions with a position error margin of 10 %. Concentration points were later introduced to simulate cyanobacteria contamination hotspots. These points were modeled as exponential decay gradients, prioritizing critical zones while maintaining overall area coverage.

Finally, the algorithm was simulated on a projection of Lake Atitlán. Several concentration points were modeled alongside the movement of 20 agents starting from random positions. The simulation demonstrated how agents distributed themselves effectively, with at least one agent assigned to each contamination hotspot while ensuring coverage of uncontaminated areas.

This work provides a strong foundation for future implementations in aquatic environments. However, further research is needed into robotic platforms and communication protocols that can operate under the extreme conditions commonly found in bodies of water.

# CAPÍTULO 1

# Introducción

El estudio de algoritmos de control de cobertura multi-agente ha cobrado importancia en el contexto del monitoreo ambiental, especialmente en aplicaciones donde es esencial cubrir grandes áreas de manera eficiente, como se menciona en 2.3. Estos algoritmos permiten a un conjunto de agentes autónomos cubrir áreas de interés, adaptándose a condiciones cambiantes y priorizando zonas críticas.

En este trabajo, se plantea la evaluación de algoritmos de control de cobertura para agentes móviles. El objetivo principal es analizar diversas estrategias de cobertura y seleccionar un algoritmo que logre la distribución satisfactoria de un grupo de agentes en un área de trabajo, tomando en cuenta criterios como la adaptabilidad a cambios en el entorno y el tamaño del área de trabajo. Para ello, se desarrollará un simulador que represente esta problemática de manera simplificada, permitiendo experimentar con diferentes configuraciones y condiciones iniciales.

Además, se propone validar experimentalmente el desempeño del algoritmo seleccionado mediante pruebas realizadas en las instalaciones de la Universidad del Valle de Guatemala. Estas pruebas utilizarán los recursos del laboratorio para emular escenarios simples pero representativos, con el fin de verificar la aplicabilidad del algoritmo en entornos prácticos.

El enfoque integral de este trabajo, que combina simulación y validación experimental, busca contribuir al desarrollo de herramientas en el campo del control multi-agente, con aplicaciones potenciales en diversas áreas como monitoreo de cuerpos de agua.

# CAPÍTULO 2

# Antecedentes

El siguiente trabajo buscará resolver el problema de control de cobertura con una implementación multi-agente. Vale la pena investigar si ha habido similares trabajos en la Universidad del Valle de Guatemala con el fin de poder ser utilizados como una posible base. En el caso de sistemas multi-agente, se encontró uno en simulación; siendo este el trabajo de graduación de la ingeniera Maybell Peña. Este trabajo de graduación tuvo como objetivo el desarrollo e implementación de un algoritmo de control para un sistema de robots multi-agente con orientación a misiones de búsqueda. Para realizar esto se planteó un algoritmo capaz de cumplir con las siguientes métricas: éxito de la formación y cambio de formación ante obstáculos.

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

Estas métricas se consideraron como los subalgoritmos utilizados para el desarrollo del algoritmo final, donde el primero se desarrolló de forma independiente a través de un proceso iterativo. Por otro lado, el segundo se implementó a este como una inclusión, consistiendo en el cambio de formación y el control para evasión de obstáculos al último modelo seleccionado. Una vez ya desarrollado de forma teórica, se implementó el algoritmo en una simulación controlada, pues de esta forma se pudo evaluar la eficiencia al considerar los parámetros físicos de los agentes [1].

Este proyecto tuvo como objetivo que el sistema tuviese la capacidad de movilizar un enjambre de robots de forma eficiente desde una posición aleatoria hasta una meta determinada a través de una variedad de obstáculos. Asimismo, se utilizó un sistema de posicionamiento global, en el cual cada agente posee un identificador único que determina su posición dentro de la formación. Cabe destacar que se utilizaron 10 agentes con la capacidad de tomar 2 posibles formaciones, siendo estos robots modelo E-Puck de GCtronic.

Ahora bien, entre los resultados más relevantes se explicó que la combinación del control de formación y el control contra colisiones en una sola función racional permitió que los agentes alcanzasen las formaciones deseadas con un mínimo error cuadrático medio. No obstante, se mencionó que el utilizar grafos totalmente rígidos disminuye el porcentaje de éxito del sistema, pues se le dificultó pasar a través de los obstáculos. Otro relevante es que utilizar el modelo de control dinámico disminuyó el alejamiento de los agentes entre sí y del radar, mas incrementa el consumo energético.

Tras analizar el proyecto simulado, se trabajó una continuación en el año 2023, investigación donde se mostró que una de las mayores limitantes del algoritmo desarrollado fue el retardado que este mostró al saturar la red Wifi del ecosistema. Ahora bien, en lo que respecta al ecosistema Robotat, la infraestructura para este fue desarrollado por Camilo Perafán en su trabajo de tesis [2]. En este se explicó cómo este ecosistema consiste en una serie de agentes a los cuales se puede conectar a través de Wifi. Esto en conjunto con el sistema de captura OptiTrack dio lugar a la creación del sistema para experimentación robótica conocida como Robotat.

El alcance de la tesis mencionada era desarrollar e implementar una red de comunicación inalámbrica para la experimentación con robots. Asimismo, una diferencia y limitación de esta es que esta buscó lograr un funcionamiento con una gran variedad de robots en vez de solo poder ser usado para un modelo en específico. Ahora bien, se mencionó que la captura de movimientos se implementó solamente para cuerpos rígidos y no para otros usos como lo sería la captura de movimientos de humanos. Entre los resultados más importantes se encuentran que se logró de forma exitosa ensamblar, calibrar y configurar el sistema de captura de movimientos OptiTrack. Otra relevante es que se logró desarrollar e implementar un programa que permitiese el envío de datos del OptiTrack a un dispositivo conectado a través de Wifi.

## 2.2. *Sensor networks: an overview*

En lo que respecta a trabajos desarrollados en la Universidad del Valle de Guatemala sobre el tema de control de cobertura como tal, no se encontró ninguna investigación previa. Es por esto que se acudió a artículos externos como el de esta sección, en el cual se desarrolló el concepto general por el que se entiende una red de sensores y buscó dar una explicación comprensiva y breve de lo que estos son. Es por esto mismo que la forma en la que se consiguió es a través de una explicación detallada de lo que es una red de sensores, los diferentes tipos que hay, las ventajas y desventajas que estos muestran y los retos presentes a la hora de intentar implementar una.

Entre la información más relevante se encuentra que las redes de sensores deben de ser pequeños, de bajo costo de producción, energéticamente eficientes, capaces de comunicarse a diferentes frecuencias y poder mantener conectividad a pesar de ser movidos de su posición inicial. Asimismo, se mencionó que las redes de sensores son de gran utilidad para la medición de datos debido a que, al ser tantos sensores, se puede obtener información bastante completa a su vez de poder grandes áreas de medición [3].

Ahora bien, algunas de las limitantes principales mencionadas fueron: la eficiencia ener-

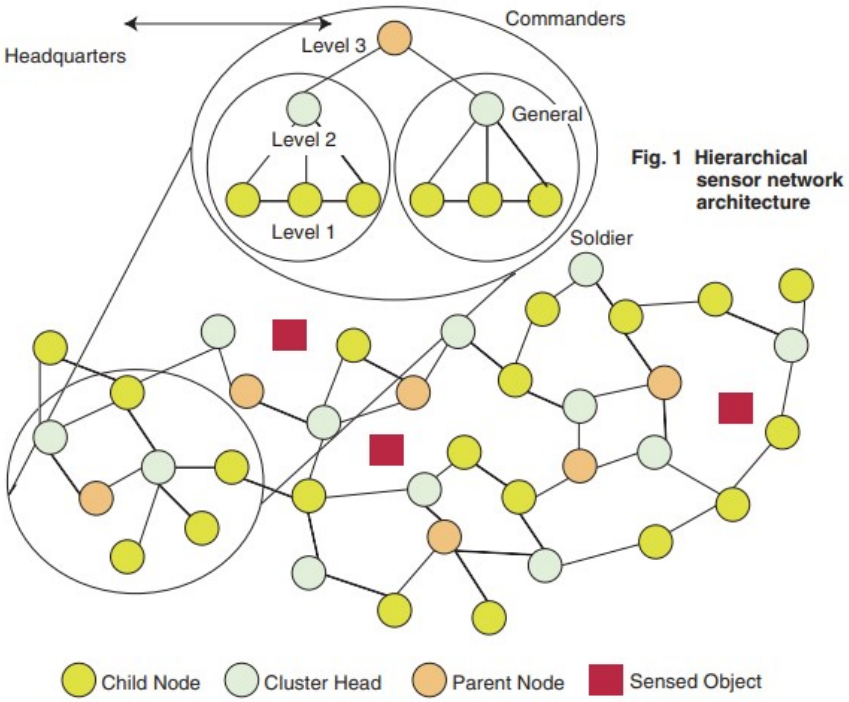



Figura 1: Ejemplo de una red de sensores con diferentes niveles de complejidad.

gética, pues al ser tantos sensores, tener que cambiar o recargar las baterías de estos podrían significar un gasto económico significativo además de ineficiente en tiempo. También se mencionó que deben de poder realizar su trabajo con una cantidad limitada de memoria, pues la recopilación de datos, aseguramiento de la seguridad de estos, entre otros factores deben de poderse lograr en un espacio bastante limitado. Otro factor es la capacidad de la red de poderse autoorganizar debido a que estos se encontrarían en lugares hostiles (como bosques, lagos, áreas de riego).

## 2.3. *Applications of Wireless Sensor Networks in Marine Environment Monitoring: A Survey*

Una vez desarrollado el concepto de las redes de sensores móviles, se prosigue con las aplicaciones que ya han sido implementadas. En este artículo busca realizar un análisis del estado actual de los cuerpos marítimos. Esto se realizó con el fin de analizar la posibilidad de que se pueda implementar un sistema de redes de sensores inalámbricos (WSNs por sus siglas en inglés, siendo *Wireless Sensor Networks*) [4]. A través del artículo mencionado se explicó que primero se realizó un análisis de los diferentes tipos de sensores y redes de sensores existentes, luego se evaluaron las posibles aplicaciones para estos y, por último, se presentaron diferentes opciones utilizadas, además de posibles algoritmos implementables.

Una vez realizada la investigación concluyeron que las redes de sensores remotas son una tecnología prometedora para el monitoreo de ambientes marítimos. Esto se debe al fácil despliegue en el área por analizar, capacidad de poder monitorear en tiempo real,

operación autónoma y bajo costo. También se mencionó que aún existen algunos retos y oportunidades en el desarrollo y despliegue de dichas redes, tales como sensores de protección oceanográfica, diseños avanzados de boyas, diseños de sistemas de recolección de energía y sistemas de estabilidad y fiabilidad. Ahora bien, se mencionó que algunas de las limitantes son la incapacidad de controlar el área donde se encontrará la red, las limitantes topológicas y la cantidad de energía limitada que se le puede proporcionar a la red, tanto para su funcionamiento como para el poder computacional.

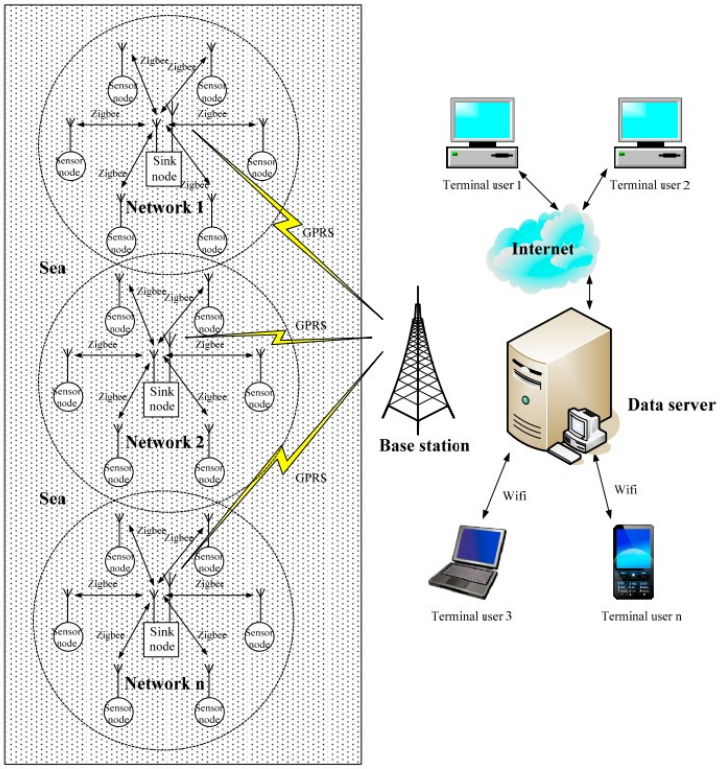



Figura 2: Ejemplo de un sistema de monitoreo marino a través de una red de sensores.

## 2.4. La contaminación del lago de Atitlán, Guatemala, amenaza la subsistencia y la salud de los habitantes

Por último, se debe destacar que el objetivo de lograr implementar el control de cobertura es poder analizar grandes áreas de forma más eficiente, cosa que se podría implementar para el análisis de cuerpos de agua a nivel nacional. Es por esto que se analiza el lago de Atitlán, pues este ha mostrado un gran problema de contaminación que ha sido notado por los habitantes de las comunidades aledañas en los últimos años. La confirmación de cianobacteria se remonta al año 2009, cuando un grupo de científicos realizó un estudio y dio dicha confirmación [5]. También se explicó que la cianobacteria puede llegar a tener un impacto negativo en humanos y animales por igual, pudiendo afectar órganos como hígado, riñones, sistema nervioso central y la dermis. Cabe destacar que en el 2016 confirmó la doctora Margaret Dix de la Universidad del Valle de Guatemala que el lago sigue bajo gran contaminación.

# CAPÍTULO 3

## Justificación

La contaminación de los cuerpos acuáticos en Guatemala es una problemática que impacta la vida de miles de guatemaltecos. Es por esto que el estudio de este fenómeno se ha vuelto de gran relevancia en las últimas décadas, cuyo objetivo es encontrar los puntos principales de contaminación para combatirlos desde su origen. El cuerpo de agua cuyo estudio sería el enfoque principal para la implementación del algoritmo expuesto en este trabajo sería el lago de Atitlán en Sololá, Guatemala. Este ha sido un gran foco de estudio en las últimas décadas, pues se reporta cómo el incremento desmesurado de la cianobacteria en el lago ha tenido un gran impacto socioeconómico en las comunidades aledañas.

El trabajo actual se vio inspirada como una posible solución o apoyo para el problema de contaminación de cianobacteria en dicho lago. Para conseguir esto se espera crear un algoritmo para que en un futuro se pueda implementar en agentes móviles capaces de medir el nivel de cianobacteria en el agua y distribuirse de forma eficiente en el cuerpo de agua que sean liberados. Para esto, se tiene planeado investigar los tipos de algoritmos que puedan ser de utilidad para la distribución de los robots. Asimismo, se realizarán pruebas en el ecosistema Robotat para poder corroborar su correcto funcionamiento.

CAPÍTULO 4

# Objetivos

## 4.1. Objetivo general

Implementar, contrastar y validar algoritmos de control de cobertura multi-agente básicos para redes de sensores móviles.

## 4.2. Objetivos específicos

- Contrastar varias propuestas básicas de algoritmos de control de cobertura multi-agente y encontrar la mejor para la aplicación hipotética para la medición de cianobacteria.
- Desarrollar un entorno de simulación simple, pero representativo, para algoritmos de control de cobertura multi-agente para redes de sensores móviles.
- Validar el algoritmo de control de cobertura seleccionado dentro del ecosistema Robotat empleando los agentes Pololu 3Pi+.

# CAPÍTULO 5

## Alcance

Este proyecto consistió en la investigación de distintos algortimos de control de cobertura multi-agente, culminando en la selección del más adecuado para abordar la problemática de control de cobertura de un área de trabajo rectangular. Posteriormente, se desarrolló y validó un entorno de simulación simple pero representativo, implementado dentro del ecosistema Robotat.

Para la selección del algoritmo, se priorizó uno adaptable a sistemas dinámicos y capaz de solucionar la problemática de control de cobertura sobre la interconectividad de los agentes. Sin embargo, se decidió no considerar factores como la comunicación directa entre los agentes ni los mecanismos específicos para lograr dicha comunicación. En su lugar, se utilizó un agente omnisciente que tenía conocimiento completo de la posición de todos los demás agentes dentro del área de trabajo, en lugar de emplear una red de sensores que transmitan información en cascada.

En la implementación dentro del ecosistema Robotat, se observó que la calibración del controlador PID no era un factor crítico. Los parámetros seleccionados permitieron un desempeño satisfactorio, logrando una distribución de Voronoi adecuada. No obstante, al no realizar una calibración exhaustiva, los agentes no siempre tomaron rutas rectas, generando trayectorias experimentales con curvas más abiertas.

En cuanto a la prevención de colisiones, estas se mitigaron en la etapa de propagación del algoritmo mediante la implementación de una ecuación de peso que ayudó a evitar colisiones durante este proceso. Sin embargo, la prevención de colisiones dentro del ciclo de control presentó desafíos significativos, ya que durante la movilización activa de los agentes no es posible conocer en tiempo real la posición exacta de todos ellos. Como resultado, en algunos casos se observaron colisiones entre agentes mientras estaban en movimiento.

# CAPÍTULO 6

# Marco teórico

Las redes de sensores móviles son capaces de funcionar de forma continua en terrenos cambiantes, donde uno de los mayores retos es el consumo energético y la vida útil de los sensores. Conforme los sensores van desactivándose al descargarse, la red va perdiendo agentes. Para poder contrarrestar esto, la red debe ser capaz de recibir e integrar nuevos agentes a la red existente. Esta opción suele ser la más viable al considerar que suele ser más económico implementar más sensores que recoger los sensores descargados, volverlos a cargar y volverlos a distribuir, como se menciona en [6].

## 6.1. Redes de sensores móviles

El control de cobertura es una herramienta útil cuando se implementa en una red inalámbrica de sensores. Esto se debe a que, con la ayuda de algoritmos para el posicionamiento de los agentes, las redes móviles de sensores muestran una gran cantidad de ventajas para la recopilación de datos. Primero, se logra que el sistema sea más resiliente, pues este puede seguir funcionando a pesar de que uno o varios sensores fallen al tener tantos a su disposición en la red, como se menciona en [3]. También se reduce la interacción humana con el sistema al tener que realizar mantenimientos reducidos, ya que pueden reaccionar de forma dinámica y pueden trabajar en lugares hostiles.

El funcionamiento de una red de sensores inalámbricos móviles consiste de una serie de sensores, los cuales crean una distribución de nodos sin la necesidad de una central de control. Esto lo logran al comunicarse de forma inalámbrica entre sí, pudiendo así saber su posición a la vez de la de los sensores en su proximidad. Con esta información, los sensores se distribuyen y una vez posicionados, empiezan con la medición de datos. Ahora los nodos funcionan como routers, donde estos recopilan información a la vez que la pueden enviarla y recibirla.

## 6.2. Control de cobertura

Control de cobertura se refiere a una red de sensores encargada de la medición de parámetros arbitrarios en diferentes áreas geográficas, como se menciona en [7] y [8]. Esto se logra a través de una serie de nodos de sensores individuales, los cuales están equipados con diferentes módulos tales como: módulo de potencia, CPU, memoria, batería y módulos de medición. Este método de recopilación de datos difiere de otros al utilizar una gran cantidad de agentes dispensables y movibles intercomunicados entre sí (llegando a ser miles de sensores). Al tener tantos agentes se logra cubrir un área mucho mayor a la vez de que se puede asegurar un mejor servicio, pues al ser tantos nodos estos aumentan la exactitud de las mediciones.

Por otro lado, como se menciona en [9], se puede implementar un control de formación para que los agentes del sistema puedan distribuirse de forma eficiente. Esto se puede observar en la naturaleza, donde las aves migratorias crean formaciones en V para migrar (esto reduce la resistencia del aire), por lo que aplicar este concepto de formaciones eficientes ayudaría a que el control de cobertura sea más fiable. A diferencia de los animales que realizan las formaciones debido a una ventaja evolutiva encontrada a través del tiempo, los agentes de una red de sensores móviles podrían encontrar una formación deseada dependiendo de la situación a través de un algoritmo. Este utiliza los datos recopilados por la red para enviar instrucciones a los agentes de qué realizar y cómo realizarlo por sí mismos, sin la necesidad de una central de comando que se los dicte.

### 6.2.1. Problemática de control de cobertura

El concepto planteado como la problemática de control de cobertura, como se menciona en [9], es un área de estudio en robótica aplicada a sistemas multi-agente. Consiste en analizar cómo distribuir y coordinar agentes de manera eficiente para cubrir un área de trabajo específica. Por esto es que es fundamental en aplicaciones donde se necesita monitorear espacios amplios.

Con el fin de lograr solucionar este problema, se pueden utilizar una variedad de algoritmos que, dependiendo del objetivo principal, serán de mayor o menor ayuda. Cabe destacar que esta problemática no solo considera la organización espacial de los agentes, sino que también considera las limitaciones en el rango de detección de los sensores y la conectividad entre los agentes. Esto para asegurar que la red de sensores se mantenga conectada y operativa durante su despliegue.

## 6.3. Diagramas de Voronoi

Se le conoce como diagrama de Voronoi a un diagrama que representa la unión de todas las regiones de Voronoi, como se explica en [10]. Estas regiones se describen como la intersección de todos los planos medios contenidos en un punto $s_n$ rodeado por otros puntos, donde los medios planos se ven descritos por todos los bisectores perpendiculares de un segmento conectado al punto $s_n$ mencionado. Cabe destacar que estas regiones consisten en todos los

puntos más cercanos a $s$ en comparación de otro punto $s'$. Asimismo, la intersección de tres bisectores define a un círculo con la característica de que cada punto $s_n$ de dichos bisectores estará a una distancia $r$ del centro del círculo generado, como se puede observar en la Figura 3. Esto último solo es verdadero si se delimita que no se pueden generar diagramas con cuatro puntos concéntricos.

Al considerar lo anterior, se obtiene una característica relevante de los diagramas de Voronoi, la cual dicta que el círculo generado por tres puntos $s_n$ está vacío. Esto es relevante al considerar que, si se pusiese un cuarto punto dentro del círculo, se puede asegurar con completa certeza que este está más cerca del vértice que los tres puntos $s_n$, ejemplo que se muestra en [10]. Por otro lado, si se conectan los sets de $s_n$ con las esquinas de cada una de sus regiones de Voronoi se obtiene un plano dual del sistema. Este es relevante pues se puede utilizar para la triangulación de un set de puntos, lo que se le conoce como la triangulación de Delauney. Esto se cumple porque dentro de cada región de Voronoi solo se encuentra un vértice, por lo que cada esquina de los triángulos generados comparte sus esquinas con los triángulos adyacentes. Se debe mencionar que existen diferentes métodos y algoritmos para poder realizar los diagramas de Voronoi además de la triangulación de Delaunay.

Se considera a $S$ como una serie de $n$ nodos en un plano bidimensional, donde para dos sitios distintos (descritos por $p,q \in S$), se encuentra que la dominancia de $p$ sobre $q$ está descrita por una subsección del plano original y que está tan cerca de $p$ como de $q$. Esto se explica en [7] a través de la siguiente ecuación:

$$dom(p,q) = x \in R^2 | d(x,p) \leq d(x,q), \tag{1}$$

donde *d(x,p)* es la distancia Euclidiana entre los puntos $x$ y $p$. Luego el segmento de línea generado entre $p$ y $q$ sirve para crear un bisector perpendicular (conocido como separador), que se relaciona con lo explicado anteriormente respecto a la creación de los diagramas de Voronoi como tal. Asimismo, la generación de estas regiones logra realizar una división poligonal de toda la región, donde cada vértice es un punto con equidistancia a otros tres y se considera el final de la región; mientras que cada punto en una esquina es equidistante a exactamente dos sitios.

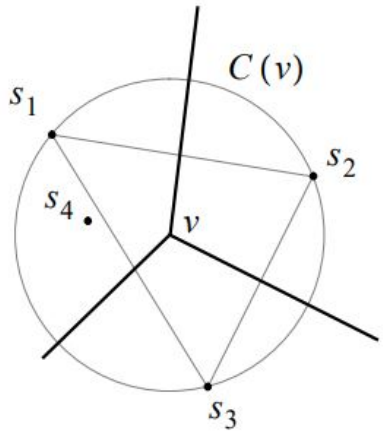



Figura 3: Ejemplo de diagrama de Voronoi generado por tres puntos $s_n$

Los diagramas de Voronoi poseen las siguientes características: si se tiene $n$ sitios/nodos, el diagrama tendrá exactamente $n$ celdas de Voronoi, donde algunas pueden estar sin restricciones (por ejemplo, si está en las afueras del diagrama). No existen vértices de Voronoi solo si todos los nodos de $S$ se encuentran en una única línea recta. No todos los separadores son esquinas del diagrama ni todas las intersecciones son vértices del diagrama.

### 6.3.1. Algoritmo de Fortune

Este algoritmo obtuvo su nombre por su creador, Steven Fortune, quien en [11] planteó un algoritmo con el cual se pueden calcular y plantear diagramas de Voronoi. Este consiste en un método de "deslizamiento lineal", donde se desliza una línea recta horizontal sobre un plano con $x$ puntos. Conforme la línea intercepta los puntos, el algoritmo calcula la transformación geométrica del diagrama de Voronoi para luego reconstruir el diagrama de Voronoi completo en base a la transformada geométrica calculada.

El algoritmo de Fortune se caracteriza por calcular el algoritmo con n cantidad de puntos en $O(n \times log(n))$ tiempo y un uso de espacio de $O(n)$. Es esto lo que le da una ventaja al algoritmo de Fortune sobre otros métodos para el cálculo de los diagramas de Voronoi, pues Fortune explica que previo a su algoritmo existían los algoritmos de incremento (con un posible peor tiempo de $O(n^2)$ y los de *divide-and-conquer*). El de Fortune combina las ventajas de ambos, dando mejores tiempos que el primero y simplificando la implementación en comparación que el segundo.

En términos generales, el algoritmo arrastra la línea horizontal y cada vez que tiene contacto con un punto empieza a generar un área alrededor de este. Esta seguirá siendo generada hasta que ya no pueda crecer más, situación que sucede cuando los puntos alrededor de esta también se expanden de similar forma. Estas van creciendo en forma de parábolas, por lo que solo pueden crecer cierta cantidad hasta cubrir un área por completo. Las regiones resultantes son las regiones de Voronoi mostradas en la Figura 4, que al combinar todas las presentes en el plano se convierte en el diagrama de Voronoi y cumplen con las características mencionadas en la sección anterior.

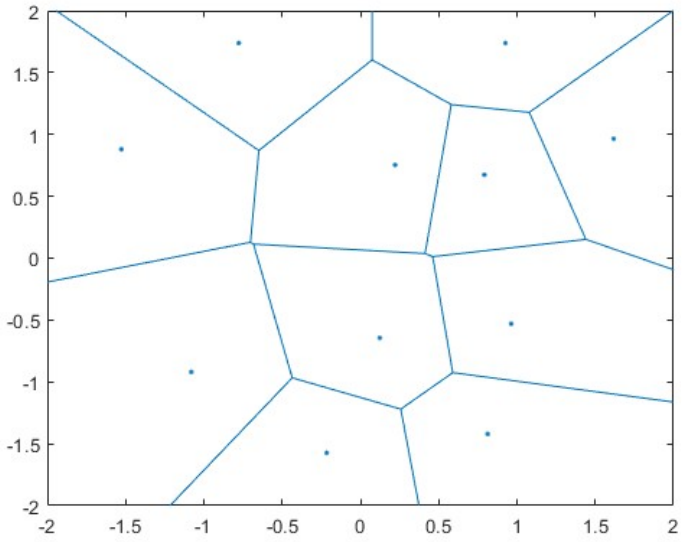



Figura 4: Ejemplo de diagrama de Voronoi completo generado con datos recopilados

## 6.4. Algoritmos de control de cobertura

Las redes de sensores móviles son de gran utilidad para poder aplicar el concepto de control de cobertura multi-agente, mas este último requiere de un algoritmo para poder ser implementado. Existen una gran variedad de algoritmos capaces de completar esta misión, cada uno con sus características, ventajas y limitantes. Por esto se explorarán los grafos de Gabriel y el algoritmo de cobertura basado en diagramas de Voronoi, para así tener un mejor entendimiento de cómo es que estos algoritmos funcionan.

### 6.4.1. Grafos de Gabriel

Este tiene como concepto básico generar una gráfica de proximidad donde se muestre la relación entre los nodos más cercanos entre sí. Sin embargo, también se crean relaciones con nodos relativamente lejanos lo que crea un a gráfica de todos los nodos locales y algunos globales [9]. Un ejemplo de cómo se generan estos es el siguiente: se asume una gráfica con vértices $v_1$, ..., $v_n$ que corresponden a una serie de agentes localizados en $x_1$, ..., $x_n \in R^2$. El grafo de Gabriel generado por lo anterior es dado por $G = (V,E)$, donde $v_i$, $v_j \in E$ siempre y cuando el ángulo entre $(x_i,x_k,x_j)$ sea agudo para todos los otros puntos $x_k$. Cabe destacar que esto toma en consideración que los puntos $(x_i)$ y $(x_j)$ no tienen un vértice en el interior. A través de esto se plantean los siguientes teoremas en [9]:

**Teorema 1.** *Todo grafo de Gabriel es planar.*

**Teorema 2.** *Todo grafo de Gabriel está conectado.*

Se debe de tomar en consideración que los grafos de Gabriel proporcionan una forma de

conectar en un solo plano todos los puntos, mas esta no es una triangulación perfecta. Es por esto que se deben de establecer más vértices si se desea solucionar el problema de cobertura. Para esto, se pueden utilizar estos grafos como una idea inicial para luego implementar una función de potencial para mover los vértices de tal forma que el sistema tenga una triangulación ideal. Con esto como fin se propone la ecuación [9]:

$$U_{ij} = \frac{1}{2}(||x_i - x_j|| - \Delta)^2 \quad \text{para todos los vértices } (v_i, v_j) \in E, \tag{2}$$

donde $\Delta$ representa la distancia entre agentes deseada. Una vez se delimita esta función, se presenta la función de control que, junto a la anterior, se debe de implementar para poder obtener una triangulación óptima.

$$\dot{\mathbf{x}} = -\sum_{j \in N(i)} \nabla_{x_i} U_{ij} = -\sum_{j \in N(i)} \frac{(||x_(t)i - x_j(t)|| - \Delta)}{||x_i(t) - x_j(t)||}(x_i(t) - x_j(t)). \tag{3}$$

### 6.4.2. Algoritmo basado en diagramas de Voronoi

El concepto detrás de este algoritmo se puede explicar como una relación dual entre los diagramas de Voronoi y la Triangulación de Delaunay. Como se explica en la sección 6.3, la idea general detrás de los diagramas de Voronoi es dividir el espacio analizado en regiones de influencia alrededor de un punto, donde la base del mismo son los vértices generados por la Triangulación de Delaunay con el fin de delimitar las celdas entre sí. Ahora bien, a pesar de que esto no logra generar una distribución simétrica y ordenada sino solo logra dividir el área de trabajo, esto sí logra organizar el sistema al dar información sobre los vértices, la posición de los puntos y las distancias entre ellos.

Como se explica en la sección 7.5.2 [9], este algoritmo utiliza el concepto de gradiente descendente, que consiste en minimizar el problema de localización guiando a los agentes hacia el centro de sus respectivas celdas de Voronoi. Esto se calcula con la ecuación:

$$\dot{x}_i(t) = -\frac{\partial H_V(x)}{\partial x_i} = -2\int_{V_i} (x_i(t) - q)\, dq, \tag{4}$$

donde $H_V(x)$ representa la función de costo de localización, $x_i(t)$ es la posición del agente $i$, y $q$ es un punto en el dominio de $V_i$, la celda de Voronoi asociada con el agente $i$.

Esta función se utiliza para medir la efectividad de la cobertura mediante el cálculo de la distancia entre los agentes y los puntos dentro de sus respectivas celdas de Voronoi. A su vez, el algoritmo busca minimizar dicha función, lo que hace que los agentes tiendan a posicionarse cerca de los centroides de sus celdas. Al utilizar esta estrategia, el algoritmo logra que los agentes se distribuyan de manera eficiente en el área de trabajo, asignando a cada uno una región de influencia definida por su celda de Voronoi. Esto permite que los agentes mantengan una cobertura óptima en el área, con una distribución equitativa que evita la acumulación innecesaria de agentes en una misma región. Como resultado, se obtiene

un sistema distribuido en toda el área de trabajo, resolviendo eficazmente el problema de control de cobertura.

Por la naturaleza descrita del algoritmo, este es de gran utilidad tanto en sistemas estáticos como dinámicos. Esto debido a que siempre que se pueda obtener la posición de los agentes, el algortimo puede cambiar su comportamiento dependiendo de cambios en el área de trabajo o número de agentes. Esto demuestra gran robustez ante el cambio. Sin embargo, una limitante que este algoritmo presenta es su dificultad de implementación en áreas de trabajo con formas complejas, pues los diagramas no se logran formar de forma ideal, por lo que se deben de realizar ajustes para que este pueda funcionar de forma deseada.

## 6.5. Ecosistema Robotat

La infraestructura para el ecosistema Robotat fue desarrollada por Camilo Perafán en su trabajo de tesis [2]. En este se explicó que el sistema de captura OptiTrack dio lugar a la creación del sistema para experimentación robótica conocida como Robotat. Para poder conectarse a este, se utiliza la función:

$$robotat = \text{robotat\_connect}(), \tag{5}$$

donde no se tienen argumentos de entrada y la salida es un *struct* que representa una conexión al servidor del ecosistema Robotat.

Primero, se debe conectar la computadora a la red WiFi *Robotat*; luego, se ejecuta una única vez la función (5). De esta forma el usuario ya se encuentra conectado al ecosistema y puede empezar a dar uso del mismo. Dentro de este existen varios marcadores únicos, los cuales al estar dentro del área de trabajo de $5 \times 4$ metros del ecosistema son reconocidos como cuerpos tridimensionales. Esto significa que tanto su posición como su orientación actual pueden ser medidas, utilizando la función:

$$robot = \text{robotat\_get\_pose}(robotat, no\_agente, 'eulxyz'), \tag{6}$$

donde *robotat* es la conexión establecida con la función (5), no_agente representa el número del marcador a leer y $'eulxyz'$ es la configuración en la que se desean los datos, dando esta los valores en $x$, $y$ y $z$ a la vez que los ángulos *yaw*, *pitch* y *roll*.

Esto es de gran utilidad debido a que el usuario puede conectar su computadora al ecosistema y empezar a realizar mediciones instantaneamente. Asimismo, posee gran precisión y mínima latencia, otorgando un entorno de simulación fiel y útil para trabajos de precisión. Lo último siendo útil para trabajos donde la replicabilidad es algo indispensable. Ahora bien, una de las plataformas robóticas más utilizadas en conjunto con el ecosistema Robotat son los robots Pololu 3PI+.

### 6.5.1. Robots Pololu3PI+

El robot Pololu 3PI+ es una plataforma móvil completa y de alto rendimiento basada en el microcontrolador ATmega32U4, por lo que en la universidad se utiliza un microcontrolador ESP32. Este robot incluye encoders de cuadratura duales para control de velocidad o posición en lazo cerrado, sensores de línea, sensores de impacto frontales y una IMU completa. Con el fin de poder utilizar estos agentes en conjunto con el ecosistema Robotat, se crearon una serie de funciones de Matlab capaces de conectarse y controlar la velocidad de los Pololu 3PI+.


Figura 5: Pololu 3PI+.

#### Conexión y uso del Pololu 3Pi+

Al igual que con el ecosistema Robotat, es necesario establecer una conexión entre la computadora y los agentes Pololu 3Pi+. Con el objetivo de proporcionarles instrucciones específicas, como ajustar su velocidad lineal y angular, o detenerlos por completo. Asimismo, es necesario declarar diferentes parámetros iniciales para los agentes, los cuales son sus características físicas y se mantienen constantes entre todos los agentes.

- Radio de las llantas, representado por wheel_radius.
- Distancia entre las llantas, representado por wheel_distance.
- Radio del agente, representado por radio_agente.

donde wheel_radius es 32/1000, wheel_distance es $(96 - 2 * 6.8)/1000$, y radio_agente es 0.12 (todos los datos dados en metros).

Por otro lado, se utiliza una matriz de desfase llamada marker_offsets para representar los desfases que hay entre los marcadores situados sobre los Pololu 3Pi+. Asimismo, cada uno de estos tiene asignado su propio marcador, siendo estos enteros del 2 al 10. Estos presentan desfases diferentes por agente, los cuales se guardan en un vector offset y se extraen del vector marker_offsets.

Estos son necesarios porque la posición de los Pololu 3Pi+ se obtiene a través del Robotat utilizando la función (6), por lo que los marcadores deben ser diferentes entre sí para

que el Robotat pueda distinguirlos como diferentes cuerpos. Ahora bien, los Pololu 3Pi+ tienen distintas funciones necesarias para poder realizar las instrucciones mencionadas con anterioridad, siendo estas las siguientes:

$$\begin{aligned} robots = \text{robotat\_3pi\_connect}(no\_agente), \\ offset = \text{marker\_offsets}(no\_agente), \end{aligned} \tag{7}$$

donde la salida es un *struct* que posee los datos de la conexión con el Pololu 3Pi+ y no_agente representa el número del marcador del agente al que se desea conectar.

$$\text{robotat\_3pi\_set\_wheel\_velocities}(robots, v\_left, v\_right), \tag{8}$$

donde robots es la conexión con el Pololu 3Pi+ y la salida de la función (7), v_left es la velocidad de la llanta izquierda y v_right de la derecha. Esta función establece la velocidad de las llantas del agente a la velocidad indicada.

$$\text{robotat\_3pi\_force\_stop}(robots), \tag{9}$$

donde *robots* es la conexión con el Pololu 3Pi+ y la salida de la función (7). Esta función detiene el agente estableciendo la velocidad de las llantas en cero.

La primera función (7) debe ejecutarse una vez al inicializar el programa. Esto se debe a que, al igual que (5), la conexión se mantiene activa hasta que se realice una desconexión (ya sea manual o accidental) de los agentes. Asimismo, al utilizar los Pololu 3Pi+, los valores del vector no_agente corresponderán a los marcadores de los agentes seleccionados.

La función (8) se utiliza para enviarle la velocidad de las llantas a los agentes. Como se puede enviar la velocidad a cada llanta de forma separada, es necesario especificar lo que valdrá cada una para evitar forzar los motores a moverse de formas erráticas. Asimismo, se recomienda limitar la velocidad máxima que se pueda enviar para evitar que el robot se mueva a altas velocidades en caso de cálculos erróneos. Es por esto último que existe la función (9), pues de esta forma se puede forzar un paro inmediato.

### Diseño del controlador

El controlador seleccionado es un PID de acercamiento exponencial, el cual, como el nombre indica, va reduciendo la velocidad del agente exponencialmente conforme este se acerca a su objetivo. Para lograr esto, se tienen diferentes constantes que influyen directamente en el comportamiento y convergencia del controlador, siendo las siguientes:

- Constante proporcional de orientación, representada por kpO.
- Constante integral de orientación, representada por kiO.

- Constante derivativa de orientación, representada por kdO.
- Velocidad lineal inicial, representada por v0.
- Velocidad angular inicial, representada por $\alpha$.

La implementación de este PID fue vista durante el curso [12], donde se utilizó un agente Pololu 3PI+ y se calibraron las constantes del PID. Estas mostraron el mejor funcionamiento durante el laboratorio, por lo que se optó por utilizarlas por el precedente mencionado, siendo las siguientes:

- kpO $= 0.15$
- kiO $= 0.0001$
- kdO $= 0.01$
- v0 $= 10$
- alpha $= 2 * 0.7$

Una vez declaradas las constantes, se prosigue con los cálculos dentro del controlador. Para que los agentes alcancen las posiciones objetivo, primero se debe calcular el ángulo *theta* que representa la orientación actual del agente, utilizando la ecuación:

$$\theta = \text{atan2}\left(\sin\left(\theta_{\text{actual}} - \text{offset}_{\text{agente}}\right), \cos\left(\theta_{\text{actual}} - \text{offset}_{\text{agente}}\right)\right), \tag{10}$$

donde $\theta_{\text{actual}}$ es el ángulo actual leído del robotat y $\text{offset}_{\text{agente}}$ es el desfase característico del agente, como se menciona en la sección 6.5.1.

Después se calcula el error de posición actual en las coordenadas $(x, y)$, dando así como resultado un vector de dos componentes. El error como tal es la resta entre la posición objetivo y la posición actual. A este error $e$ es que se le calcula la norma, obteniendo así el error $eP$ a la vez de guardarse en el vector eP_tot en la posición del agente actual.

En lo que respecta al error angular, se calcula un ángulo $\theta$ objetivo con la ecuación de tangente de Matlab $atan2$, usando como argumentos de entrada el error en $x$ y en $y$. Asimismo, el error de orientación se calcula realizando la resta entre el ángulo objetivo y el actual, para luego realizar la operación:

$$e_O = \text{atan2}\left(\sin\left(e_O\right), \cos\left(e_O\right)\right), \tag{11}$$

donde $e_O$ como argumento de salida representa el error de orientación resultante mientras que el $e_O$ de entrada es el error de posición calculado haciendo la resta explicada.

Finalizada la sección de medición de posiciones y cálculos de error, se prosigue con la implementación del control de velocidad lineal y angular. Para la lineal se utiliza un controlador de acercamiento exponencial, el cual se utiliza para calcular la constante proporcional de velocidad lineal a través de la ecuación:

$$k_P = \frac{v_0\left(1 - e^{-\alpha e_P^2}\right)}{e_P}, \tag{12}$$

donde $v_0$ es la velocidad lineal incial, *alpha*, la angular inicial y $e_P$ la norma del error de posición del agente actual.

Este valor de $k_P$ se multiplica por la norma del error de posición actual $e_P$, dando como resultado la velocidad lineal deseada. En lo que respecta al controlador de velocidad angular, se deben de calcular dos constantes más:

- Error de orientación integral, representado por $E_O$.
- Error de orientación derivativo, representado por $e_O D$.

El valor de $E_O$ se calcula sumándole el valor de $e_O$ al valor anterior de $E_O$, mientras que el $e_O D$ realizando una resta entre el valor actual de $e_O$ y su valor anterior. Una vez calculados todos los errores de orientación, se prosigue con una ecuación para calcular la velocidad angular, la cual utiliza las constantes enlistadas al inicio de este capítulo 6.5.1 y calculados con la ecuación:

$$w = k_{P_O} e_O + k_{I_O} E_O + k_{D_O} e_{O_D}, \tag{13}$$

Ahora que ya se calcularon las velocidades angular y lineal, se debe calcular la velocidad de cada llanta. Esto se debe a que el Pololu 3Pi+ recibe un valor de velocidad en número de revoluciones, por lo que se utilizan las siguientes ecuaciones para calcular las velocidades de la llanta derecha e izquierda respectivamente:

$$v_{\text{right}} = \frac{v + \left(\frac{\text{wheel\_distance}}{2}\right) w}{\text{wheel\_radius}}, \tag{14}$$

$$v_{\text{left}} = \frac{v - \left(\frac{\text{wheel\_distance}}{2}\right) w}{\text{wheel\_radius}}, \tag{15}$$

donde $v$ es la velocidad lineal calculada, $w$, la angular, *wheel_radius* y *wheel_distance* son constantes declaradas al inicio del capítulo.

Una vez calculadas las funciones, se envían los datos al agente actual a través de la función (8), siendo un ejemplo de esto lo siguiente:

$$\text{robotat\_3pi\_set\_wheel\_velocities}(\text{robots}\{\text{agente}\}, v_{\text{left}}, v_{\text{right}}), \tag{16}$$

donde *robots* es la salida de (7), *agente* es el valor del contador representando el agente actual, $v_{\text{left}}$ y $v_{\text{right}}$ son las velocidades para las llantas derecha e izquierda.

# CAPÍTULO 7

## Entorno de simulación para algoritmos de control de cobertura multi-agente para redes de sensores móviles

Existen diferentes tipos de algoritmos de control de cobertura multi-agente, por ello, crear un entorno de simulación es de vital importancia para evaluar diferentes parámetros. Algunos de estos son la eficiencia, la facilidad y la capacidad de ser implementado en entornos realistas y encontrar posibles restricciones que este pueda llegar a tener. Asimismo, este debe estar en un lenguaje de programación que sea implementable en el ecosistema del Robotat.

### 7.1. Comparación y evaluación de algoritmos de control de cobertura multi-agente

Los algoritmos de control de cobertura multi-agente explicados en la sección 6.4 son herramientas de gran utilidad para el análisis de proximidad y distribución espacial. Ambos se utilizan en sistemas multi-agente y redes de sensores, aunque cada uno aborda la problemática desde una perspectiva diferente.

El algortimo basado en diagramas de Voronoi se centra en la división del espacio en áreas de influencia alrededor de cada agente, logrando así optimizar la cobertura de un área de trabajo mediante una distribución equitativa. Esto lo convierte en un enfoque que aborda la problemática desde una perspectiva espacial, donde se busca maximizar el alcance de los agentes al distribuirlos en un área. Esto último lo hace bastante útil para aplicaciones donde se necesite una cobertura uniforme de un área de trabajo.

Por otro lado, los grafos de Gabriel abordan la problemática de cobertura desde un punto de vista de proximidad. En este enfoque, dos agentes se conectan mediante una arista siempre que no haya ningún otro punto dentro del círculo cuyo diámetro sea la distancia entre ellos.

Esta característica resulta útil cuando se plantean problemáticas donde se desea analizar la conectividad y patrones de agrupamiento en redes de sensores o sistemas distribuidos en vez de abordar la problemática de cobertura espacial directa en un área de trabajo.

El algoritmo a seleccionar debe ser capaz no solo de resolver la problemática de cobertura, sino también presentar la capacidad de implementarle puntos de concentración. Dado que se busca simular la contaminación en cuerpos de agua, la cual en este caso se presenta como áreas verdes de cianobacteria, el algoritmo debe ser capaz de representar estas zonas y considerarlas al momento de realizar el control de cobertura.

Considerando las características más significativas mencionadas anteriormente, el algoritmo basado en grafos de Gabriel no es adecuado para esta implementación. Este enfoque prioriza la conectividad entre agentes. Esto impide que los agentes se concentren en los puntos de contaminación, ignorando así un aspecto fundamental de la problemática planteada. Asimismo, la robustez del algoritmo es limitada en cuanto al objetivo de cobertura del área debido a que siempre priorizará la conectividad entre agentes sobre la cobertura del área de trabajo. Además, como la conectividad entre los agentes no es relevante para esta problemática, este algoritmo soluciona un problema que no existe, por lo que se considera inadecuado.

Por otro lado se encuentra el algoritmo basado en diagramas de Voronoi, cuyo objetivo principal es la cobertura espacial. Esto significa que priorizará siempre cubrir el área sobre la conectividad entre los agentes, lo que se apega a la problemática planteada. Asimismo, al establecer puntos de concentración, los agentes son capaces de modificar su distribución en función a estas áreas de interés. Esto garantiza que permite cubrir la mayor cantidad de área mientras algunos de los agentes se concentran alrededor del punto de concentración. Esto en conjunto con su adaptibilidad a sistemas dinámicos le da mayor robustez al algortimo en cuestión de cobertura de un área de trabajo.

En conclusión, es evidente que el algoritmo preferible para la problemática de control de cobertura multi-agente es el basado en diagramas de Voronoi. La capacidad de adaptarse a sistemas dinámicos, gran robustez y posibilidad de implementación de puntos de concentración demuestra ser superior para la problemática en cuestión. El algoritmo basado en grafos de Gabriel simplemente no se ajusta a la problemática, ya que la conectividad entre agentes no es relevante en este contexto pues la conectividad entre agentes es algo despreciable en este proyecto.

## 7.2. Selección del lenguaje de programación

Dado que se busca implementar el algoritmo dentro del ecosistema Robotat, las opciones de lenguajes de programación que permitan hacerlo de manera sencilla y directa son Python y Matlab. Esto se debe a que la infraestructura para conectarse y extraer datos del Robotat ya existe en estos dos lenguajes, lo que permitiría enfocarse en el desarrollo del algoritmo en sí. De estas opciones, se eligió Matlab debido a que ofreció una gran variedad de herramientas de gran utilidad en relación con operaciones matemáticas y gráficas.

### 7.2.1. Ventajas de Matlab

Para el algoritmo de control de cobertura basado en diagramas de Voronoi, Matlab ofreció funciones de gran utilidad, tales como:

$$[v_x, v_y] = \text{voronoi}(x, y), \tag{17}$$

donde $[v_x, v_y]$ son vectores columna que representan las coordenadas $x$ y $y$ de los puntos, $(x, y)$ es una matriz de vectores a dibujar por la función;

$$[v, c] = \text{voronoin}(P), \tag{18}$$

donde $v$ es una matriz con los vértices del diagrama de Voronoi, $c$ los índices de los vértices de $v$ y $P$ es una matriz de puntos que se desean calcular por la función.

$$in = \text{inpolygon}(xq, yq, xv, yv), \tag{19}$$

donde $xq$ y $yq$ son las coordenadas de un punto, $xv$ y $yv$ son un área e $in$ que devuelve la función confirma si las coordenadas del punto se encuentran dentro del área indicada.

## 7.3. Desarrollo del entorno de simulación

Una vez seleccionado el lenguaje de programación, se comenzó con el desarrollo del entorno de simulación como tal. Esta debió tomar como argumentos de entrada los siguientes datos:

- Número de agentes, para propagar el diagrama de Voronoi una cantidad $n$ de veces.
- Paso del tiempo, el cual dicta la rapidez con la que progresará el algoritmo.
- Posiciones iniciales únicas de los agentes para que el algoritmo tenga un punto de referencia para empezar la propagación.
- Tamaño del área de trabajo, para distribuir y mantener a los agentes dentro de ella.

Una vez definido esto, se prosiguió con un ciclo infinito para que el algoritmo pueda correr de forma indefinida hasta que se llegue a la propagación de Voronoi deseada. Asimismo, se creó un ciclo que corre el algoritmo desde la perspectiva de cada punto inicial. Dentro de estos ciclos se usó la función (18), donde se utilizaron las posiciones iniciales de los agentes como argumento de entrada. Con los índices de los vértices que esta función devuelve, se procedió a realizar una serie de revisiones para asegurarse de que estos índices no causen errores en los cálculos del algoritmo, por lo que se revisa si:

- El índice se encuentra vacío.
- El índice es menor o igual a cero.

- El índice posee un largo menor a 3.

Si alguna de estas condiciones se cumple para alguno de los agentes, se decide no actualizar los datos de las posiciones de dicho agente para evitar errores. Una vez corroborado esto, se prosiguió con la generación de las celdas de Voronoi con base en los vértices devueltos por la función (18) y los índices ya revisados. Cabe destacar que esta función también genera puntos con valores iguales a infinito, por lo que se procedió a eliminar nuevamente cualquier instancia de infinito.

Tras generar los puntos de las celdas de Voronoi, se revisó si estos se encuentran vacíos. Si este es el caso, nuevamente se optó por no actualizar las posiciones nuevas y se conservan las anteriores. Una vez finalizado este proceso, se empleó la función (19) para corroborar que cada agente (representado como un punto dentro de las celdas de Voronoi) se encontrase dentro de los vértices declarados. El resultado de esta función es un vector que indica si cada punto se encuentra dentro del polígono deseado (en este caso la celda de Voronoi). Luego, se revisó que los valores de dicho vector existan para proceder con el cálculo del centroide del polígono analizado con anterioridad.

Una vez calculados los centroides, se utilizó una ecuación de peso para evitar que los puntos pasen sobre otros y colisionen durante la propagación del algoritmo. Esto se hizo con el fin de representar de manera más realista el comportamiento que tendrían los agentes. Para ello, se inició un bucle donde se comparó la posición del punto actual respecto a los demás y luego se calculó la norma, de modo que este valor sea siempre positivo. Una vez realizado esto, se comparó la norma obtenida con dos veces el radio del punto para así verificar si estaban dentro del rango de colisión. En caso de que lo estuviesen, se utiliza la ecuación (20), basada en [9], para calcular la fuerza de repulsión a aplicar, guardándola en una variable.

$$\phi_{j,h} = \sum_{\substack{h=1 \\ h \neq j}}^{n_{robots}} \frac{4r - \|\mathbf{x}_j(t) - \mathbf{x}_h(t)\|}{(2r - \|\mathbf{x}_j(t) - \mathbf{x}_h(t)\|)^2} (\mathbf{x}_j(t) - \mathbf{x}_h(t)), \tag{20}$$

donde $\phi_{j,h}$ es el coeficiente de colisión total de un punto respecto a los otros, $x_j(t)$ es la posición actual del punto $j$, $x_h(t)$ la posición del punto $h$, $r$ el radio del punto a analizar y $n_robots$ el número de puntos analizados.

$$\mathbf{x}_j(t + \Delta t) = \mathbf{x}_j(t) + \Delta t \left(\mathbf{c}_j(t) - \mathbf{x}_j(t) - \phi_{j,h}\right), \tag{21}$$

donde $x_j(t + \Delta t)$ es la posición futura, $\Delta t$ es el paso del tiempo, $x_j(t)$ es la posición actual del punto $j$, $\mathbf{c}_j(t)$ es el centroide en el punto $j$ y $\phi_{j,h}$ es su respectivo coeficiente de colisión.

Una vez calculada la posición futura de cada uno de los puntos, se actualizaron las posiciones actuales por las nuevas posiciones. Esto se hizo con el fin de poder repetir el ciclo y propagar los diagramas de Voronoi hasta llegar a un resultado satisfactorio. Para visualizar esto, se utilizaron las herramientas de Matlab para graficar los puntos y las líneas de los

polígonos generados alrededor de ellos, que corresponden a las celdas de Voronoi. Esto se realizó con la función innata de Matlab plot().

## 7.4. Validación del entorno de simulación

Una vez finalizado el entorno de simulación, se prosiguió con la realización de pruebas para verificar que funcionase de forma deseada. Como referencia para verificar si el entorno presentó un funcionamiento correcto se utilizó la teoría explicada en la sección 6.4.2 y en el capítulo 7, sección 7.5.3 [9].

### 7.4.1. Capacidad máxima de agentes en el algoritmo

Con el fin de realizar la validación del algoritmo, este se sometió a diferentes tipos de pruebas. La primera consistió en variar el número de agentes dentro del área de trabajo para detectar las posibles limitaciones que pueda tener. Para esto, primero se generaron valores iniciales aleatorios para los agentes, que se crearon a través de la función randi(randi_val) de Matlab. Esto se realizó con el fin de poder generar una cantidad $n$ de posiciones aleatorias de forma rápida y sencilla. Luego, el área de trabajo para el simulador se declaró con base en estos valores, con un límite inferior de 0 y un límite superior un 10 % mayor al ingresado en randi(randi_val). Para estas pruebas se utilizó un valor de 50 para randi_val.

Se comenzó probando con 3 agentes ya que la función (18) devuelve error si se ingresa un valor menor a este. No obstante, se observó que el algoritmo no funcionaba al utilizar entre 3 y 6 agentes, pues no existían suficientes agentes para poder crear una distribución satisfactoria lo que impidió que el algoritmo converja (como se demuestra en las figuras 6 y 7). Con 7 agentes se observó un comportamiento similar, aunque este sistema puede converger en ciertos casos con las condiciones iniciales ideales.

Al probar con 8 agentes, se observó un incremento considerable en la probabilidad de convergencia del algoritmo, y las condiciones iniciales de los agentes dejaron de ser un factor tan determinante para la convergencia. En la Figura 8 se observa un sistema convergente. A partir de 9 agentes, el algoritmo comenzó a converger con mayor frecuencia que a diverger. A través de más pruebas donde se incrementó el número de agentes por corrida, se encontró que, conforme se aumenta el número de agentes, las veces que el algoritmo converge también aumenta.

Cabe destacar que al incrementar el número de agentes, el área de trabajo tuvo que incrementarse también con el fin de poder evitar condiciones iniciales repetidas. Esto fue necesario porque la función (18) no puede aceptar como argumento de entrada puntos repetidos. Siempre que se mantenga esta relación, el algoritmo funcionará de manera deseada. En las figuras 9, 10 y 11 se muestran tres casos: el primero es un sistema convergente de 10 agentes, el segundo de 100 y el tercero de 1000, donde el valor de randi_val fue 50, 500 y 5000, respectivamente.

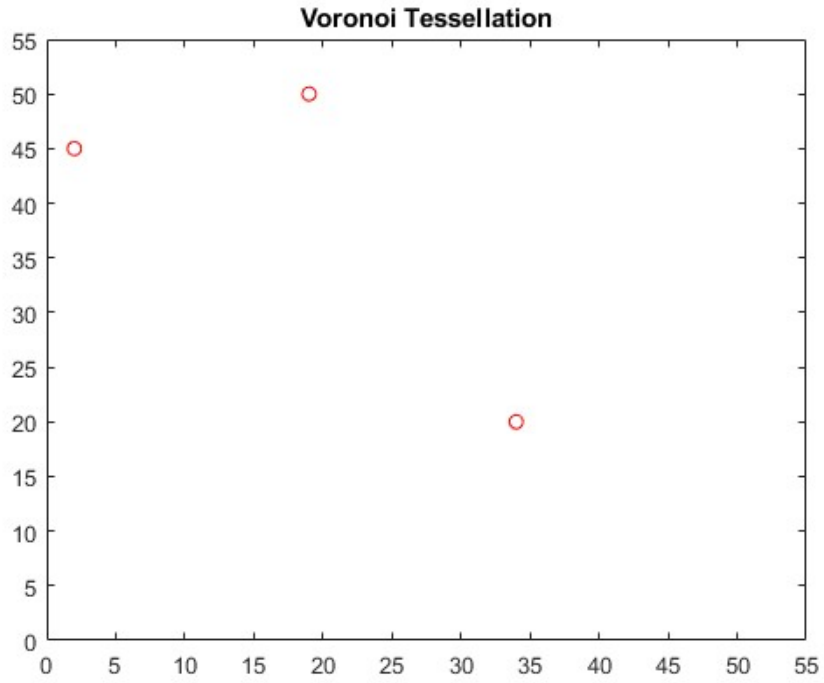



Figura 6: Diagrama insatisfactorio de Voronoi generado por el algoritmo con 3 agentes.

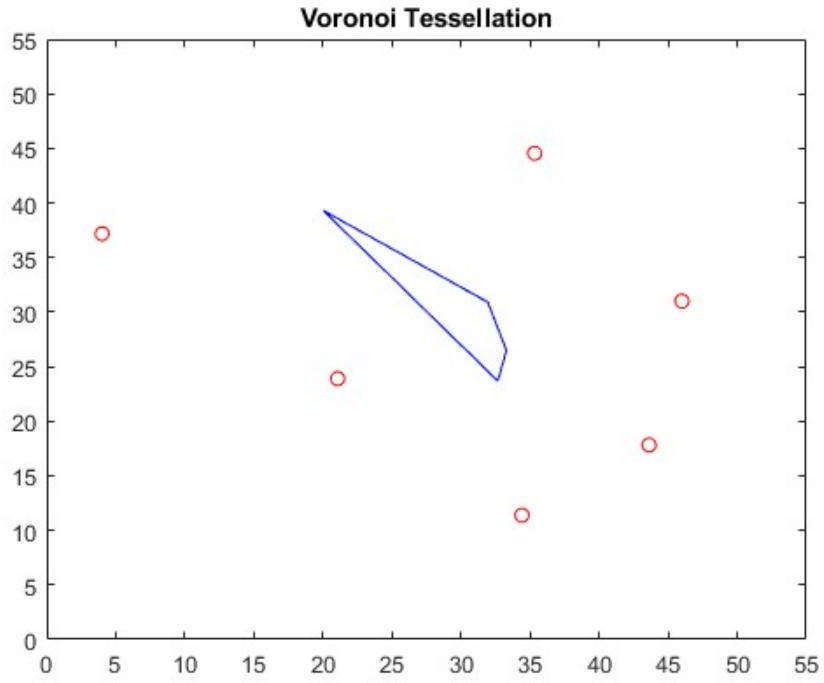



Figura 7: Diagrama insatisfactorio de Voronoi generado por el algoritmo con 6 agentes.

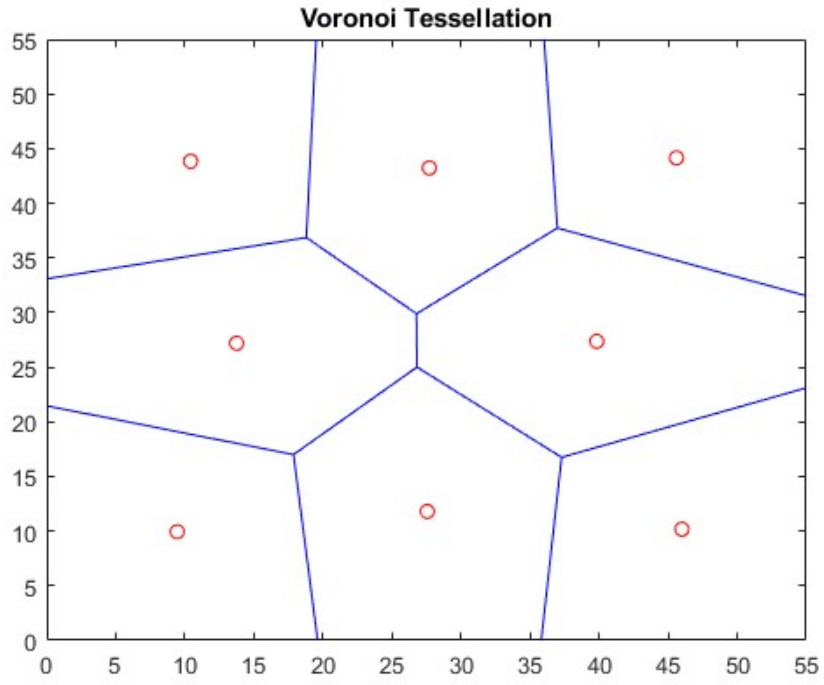



Figura 8: Diagrama satisfactorio de Voronoi generado por el algoritmo con 8 agentes.

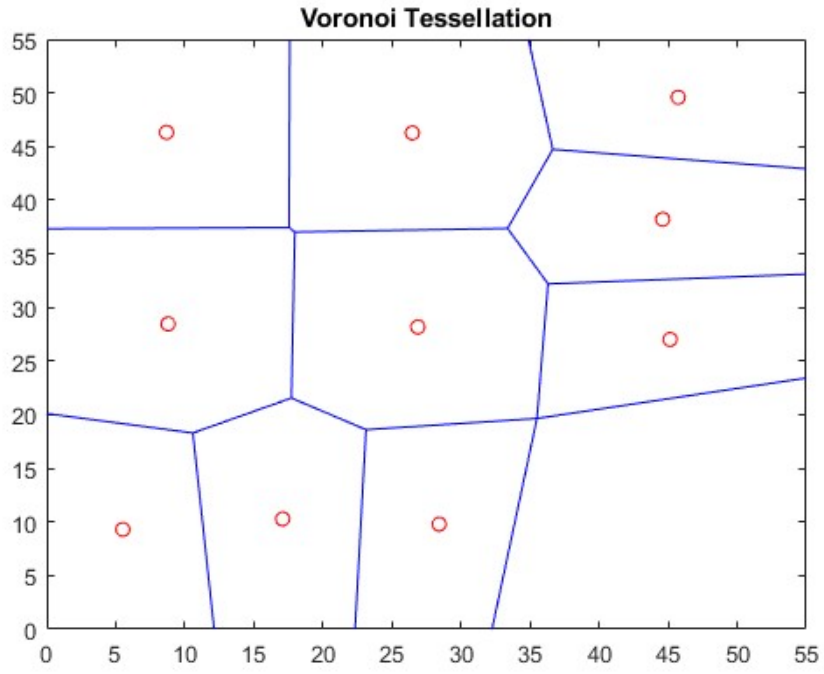



Figura 9: Diagrama satisfactorio de Voronoi generado por el algoritmo con 10 agentes.

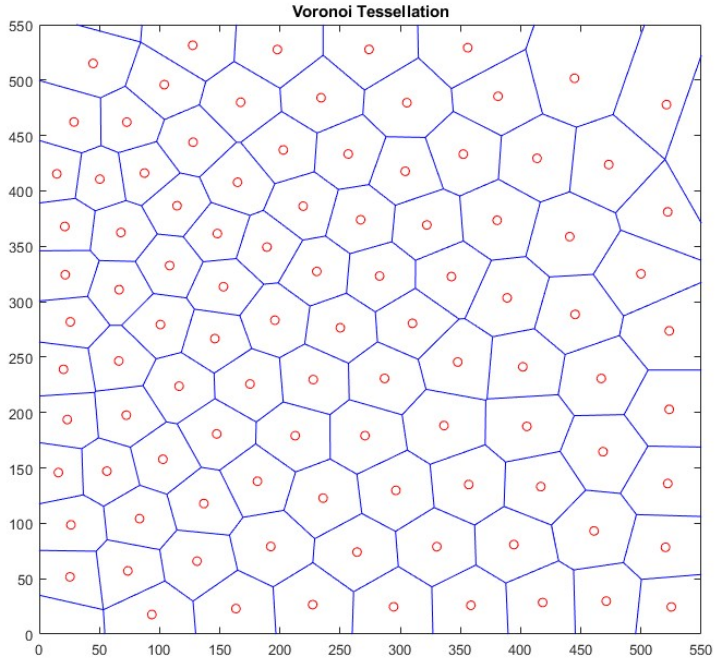



Figura 10: Diagrama satisfactorio de Voronoi generado por el algoritmo con 100 agentes.

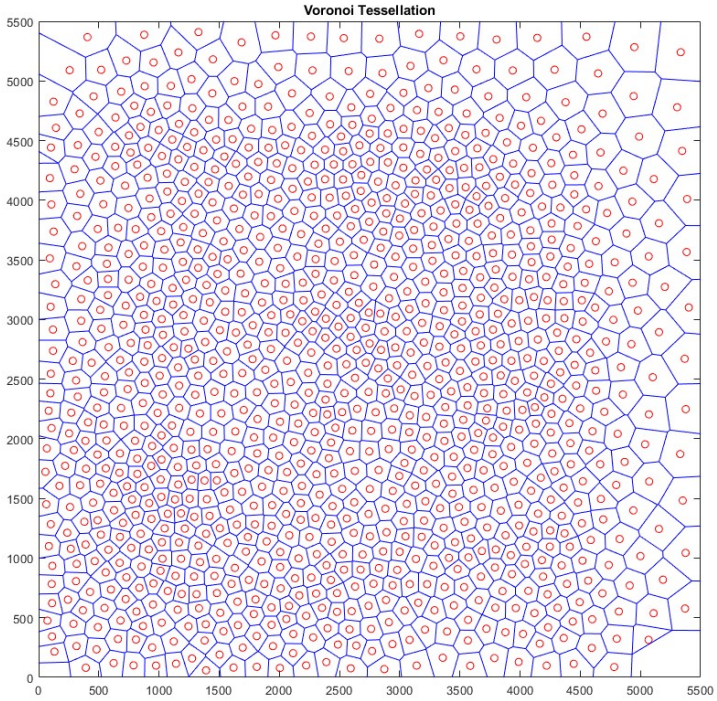



Figura 11: Diagrama satisfactorio de Voronoi generado por el algoritmo con 1000 agentes.

### 7.4.2. Influencia del paso de tiempo en el comportamiento algoritmo

Otro factor que influyó en la convergencia del sistema fue el paso de tiempo, $\Delta t$, utilizado. Esto se debe a que este valor multiplica el cambio calculado por el algoritmo, por lo tanto, cuanto menor sea este valor más lenta será la propagación del algoritmo. A pesar de que un mayor $\Delta t$ puede dar la idea de una convergencia más veloz, se observó que sucede lo contrario, ya que al incrementar demasiado el valor el algoritmo se volvió inestable. Los agentes empezaron a hacer movimientos bruscos, moviéndose largas distancias sin puntos intermedios; además, estos fueron más propensos a 'vibrar' en el mismo lugar. Esto último sucedió porque el paso de tiempo amplifica los movimientos normalmente insignificantes, lo que generó este movimiento de "vibración"mencionado, el cual puede describirse como movimientos bruscos que realiza un agente al moverse de un punto a otro de forma repetida y veloz.

En el otro extremo se encuentra reducir extremadamente el valor del $\Delta t$, pero esto también tuvo efectos adversos. Si el valor se reducía demasiado, la nueva posición era tan similar a la anterior que el algoritmo simplemente no se propaga y se estanca. Asimismo, en caso de que sí se propague, la propagación ocurrió de manera extremadamente lenta, pudiendo duplicar o triplicar el tiempo de convergencia esperado. Es importante destacar que un paso de tiempo mayor o menor puede causar que un sistema convergente diverja o viceversa sin la necesidad de cambiar las condiciones iniciales.

El paso de tiempo más pequeño probado fue 0.0001, el cual no mostró movimiento alguno. El siguiente valor probado fue 0.001, que mostró movimientos extremadamente suaves para los agentes, incrementando el tiempo de convergencia a un nivel inaceptable e inutilizable en escenarios realistas. Al intentar con 0.01, se observó que el sistema se movía más rápido que el anterior y realizaba movimientos suaves, siendo su principal problema que aún era demasiado lento para una implementación realista. Se seleccionó un valor de 0.1, y este mostró una velocidad de convergencia aceptable sin mayor vibración, pero con algunos movimientos bruscos. Por esto se decidió optar por 0.05, pues demostró tener el mejor balance entre velocidad de convergencia, magnitud de la vibración y movimientos suaves. Finalmente, es importante destacar que al utilizar valores mayores a 0.1, el algoritmo logra converger de forma deseada a una gran velocidad, pero las vibraciones en las posiciones finales son tan extremas que suelen hacer oscilar a los agentes demasiado.

### 7.4.3. Implementación con mediciones del ecosistema Robotat

Una vez ya validado el funcionamiento y delimitadas las capacidades del simulador, se prosiguió con su integración con el ecosistema Robotat. Para esto, se establecieron los siguientes casos para las condiciones iniciales, seleccionados dependiendo del valor asignado para la variable condiciones_iniciales:

- **Caso 1:** Un sistema con posiciones iniciales aleatorias y con puntos de atracción.
- **Caso 1.5:** Un sistema con posiciones iniciales aleatorias y sin puntos de atracción.
- **Caso 2:** Cargar posiciones iniciales desde un archivo `.mat` y con puntos de atracción.

- **Caso 2.5:** Cargar posiciones iniciales desde un archivo `.mat` y sin puntos de atracción.
- **Caso 3:** Extraer posiciones iniciales del ecosistema Robotat y sin puntos de atracción.
- **Caso 4:** Extraer posiciones iniciales del ecosistema Robotat y con puntos de atracción.

Luego, se declara un vector $no\_agente$ de longitud $n$ que contiene valores $m$, donde $n$ es la cantidad de marcadores a utilizar y $m$ es el número asignado a cada marcador. Luego, se debe crear un ciclo que se ejecute $i$ veces, donde $i$ es la longitud del vector $no\_agente$. Esto se realiza con el fin de leer la posición de cada uno de los marcadores con la función 6, donde $no\_agente$ se llama de forma secuencial. A continuación se muestra un ejemplo:

$$robot(i,:) = \text{robotat\_get\_pose}(robotat, no\_agente(i), 'eulxyz'), \tag{22}$$

donde $i$ es un valor del ciclo mencionado que incrementa en uno cada ciclo.

Esto último devuelve las posiciones mencionadas en la función (6), las cuales se guardaron en la matriz $robot$ de tamaño $n$x6. No obstante, como los ángulos no son relevantes para el simulador, se procedió a guardar solamente las posiciones en $x$ y $y$ de cada punto en dos vectores, llamados $posx$ y $posy$ respectivamente. Estos valores se utilizaron como los puntos iniciales de los agentes.

Antes de poder empezar con las mediciones, primero se estableció un área de trabajo, para luego posicionar los marcadores seleccionados en la configuración deseada dentro de dicha área. En las pruebas realizadas se utilizó un área de trabajo de $3 \times 3$ metros para poder dejar a todos los agentes en un cuadrado alrededor del centro y asegurar que las lecturas del Robotat fuesen lo más precisas posibles. Asimismo, se posicionaron los marcadores de manera que tengan posiciones únicas y representen una posible formación inicial realista.

Concluido el proceso de preparación del Robotat, el algoritmo tomó las posiciones iniciales de los agentes y las propagó hasta alcanzar una distribución de Voronoi satisfactoria. Se realizaron 100 pruebas y se observó que el algoritmo funcionó de forma idéntica a la simulación con valores aleatorios generados por computadora. Considerando que el Robotat solamente se utilizó al inicio para la preparación de los datos iniciales, los resultados fueron congruentes.

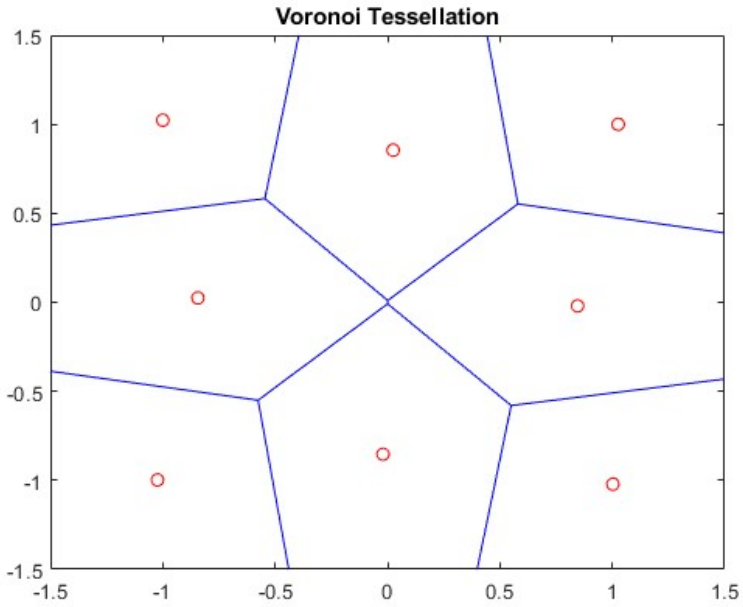



Figura 12: Diagrama satisfactorio de Voronoi con 8 agentes generado a base de condiciones iniciales leídas del ecosistema Robotat

Observando los resultados obtenidos, se confirmó que el algoritmo seleccionado fue el indicado para la problemática de control establecida. Asimismo, el entorno de simulación demostró ser una herramienta útil para la validación del algoritmo, mostrando la evolución del diagrama de Voronoi conforme se fue propagando hasta llegar a una distribución satisfactoria. Lo que prosiguió fue encontrar una forma de implementar el simulador dentro del ecosistema Robotat como tal en vez de solo utilizar una única lectura inicial para las posiciones de los agentes.

# CAPÍTULO 8

## Validación del algoritmo de control de cobertura en el ecosistema Robotat utilizando agentes Pololu 3Pi+

El algoritmo de control de cobertura basado en diagramas de Voronoi mostró un funcionamiento satisfactorio dentro del entorno de simulación. Aunque presentó algunas limitaciones, mencionadas en la sección 7.4 del capítulo 7, estas se mitigaron utilizando un número de agentes igual o mayor a 8, además de mantener un paso de tiempo razonable para evitar que el sistema fuese demasiado errático.

Para realizar la conexión a todos los agentes se utilizó el mismo concepto que con la lectura de los marcadores: s creó un ciclo que se ejecuta $i$ veces, siendo $i$ la longitud del vector no_agente. Esto se realizó con el fin de conectarse con cada uno de los agentes con la función (7), donde no_agente se llamó de forma secuencial.

$$\begin{aligned} robots\{i\} &= \text{robotat\_3pi\_connect}(no\_agente(i)), \\ offset(i) &= \text{marker\_offsets}(no\_agente(i)), \end{aligned} \tag{23}$$

donde $i$ es un valor del ciclo mencionado que incrementa en uno cada ciclo.

Una vez se completó la conexión a los agentes, se procedió a leer sus posiciones iniciales. Para ello, se siguió el mismo procedimiento descrito en la sección 7.4.3, lo que indica que el algoritmo ya tiene un área de trabajo definida y las posiciones iniciales de los agentes establecidas.

## 8.1. Implementación del algoritmo con los agentes Pololu 3Pi+

La estructura del algoritmo se mantuvo igual al simulador para que este pudiese ser válido al momento de comparar el comportamiento entre ambos. No obstante, debido a que los agentes se mantuvieron estáticos a menos que se les enviase alguna velocidad, fue necesario implementar un controlador que calculase y enviase las velocidades a las llantas de los Pololu 3Pi+. Por esta razón, se implementó primero un contador de ciclos, el cual se representó con la constante ciclos y se le asignó un valor inicial de 1. Este se incrementa en uno cada vez que se completa un ciclo de propagación del algoritmo para los $n$ agentes, y cuando este llega a un valor $k$, que representa el límite de ciclos, este detiene la propagación y pasa al ciclo de control.

Antes de explorar el controlador, se deben mencionar algunas de las limitaciones encontradas al variar el límite $k$ mencionado. Cuanto menor fuese este valor, más reactivo se volvió el algoritmo a los cambios, aunque a costa de la velocidad de convergencia. Esto se debió a que el algoritmo podría reaccionar más rápido a los cambios al entrar con mayor frecuencia al ciclo de control y actualizar las posiciones de los agentes. Sin embargo, el tiempo de convergencia incrementó debido a que este tendría menos tiempo para propagar el diagrama de Voronoi. Consecuentemente, los movimientos de los agentes se volvieron más bruscos debido a que recorreieron distancias cortas.

Por otro lado, cuanto mayor fuese el valor de $k$, mayor era la propagación del diagrama, lo que provocaría una disminución en el tiempo de convergencia. No obstante, como la posición de los agentes solamente se actualizó cada vez que se cumplieron los ciclos, el sistema revisaría de manera limitada los cambios imprevistos que las posiciones de los agentes puedan sufrir, haciéndolo menos reactivo a los cambios.

### 8.1.1. Implementación del controlador

Para lograr que los agentes lleguen a su objetivo de forma segura y satisfactoria, se necesitan los siguientes datos del agente para el controlador:

- Posición objetivo del agente, siendo esta las coordenadas $(x, y)$ del punto.
- Posición actual del agente, siendo esta la lectura del Robotat de las coordenadas $(x, y)$ del agente.
- Ángulo actual del agente, siendo este el ángulo $\theta$ del agente respecto al origen.

Completada la declaración de constantes, se prosiguió con el diseño del controlador. El concepto general detrás de este es que se mantendrá activo mientras la norma del error de posición, representado por $eP$, sea menor o igual a un valor arbitrario. Este valor determina el margen de error esperado del algoritmo práctico respecto del teórico. Tras realizar varias pruebas se encontró que un valor aceptable es de 0.1, o un error de posición del 10 %. Al disminuir por debajo de este umbral, el controlador nunca logró salir del ciclo principal pues el error nunca fue tan bajo. Por otro lado, si se incrementaba, el agente no se logró acercar

lo suficiente al punto objetivo y las pruebas prácticas ya no fueron tan representativas del sistema teórico.

Antes de entrar al ciclo principal explicado anteriormente, primero se declaró un vector unitario $n \times 1$, con el nombre $eP\_tot$, con el fin de guardar en este los errores de posición de todos los agentes. Luego, se comenzó el ciclo mencionado, solamente que al tratarse de más de un agente, se tuvo que mantener el ciclo activo siempre que uno de los errores de los agentes se encontrara por encima del límite mencionado. Esto se realizó con la función $\max(eP\_tot)$ de Matlab, la cual solo tomó el valor máximo del vector $eP\_tot$ como referencia en la comparación. Esto se hizo para asegurar que todos los agentes tuviesen un error de posición máximo del 10 %.

Dentro de este ciclo se generó otro, el cual se corrió desde la perspectiva de cada uno de los agentes. Para esto, se utilizó una variable para llevar cuenta de qué agente se estaba controlando, representada por *agente*. Esta variable se mantuvo en un rango entre 1 y la longitud del vector $no\_agente$, se incrementó al final de cada ciclo y se reinició al llegar al valor máximo.

Ya inicializado el ciclo, se utilizó la posición futura guardada del algortimo como la posición objetivo del agente. Luego, se utilizó la función (6) para obtener la posición actual del agente, además que se guardó su orientación para así calcular el ángulo $\theta$ con la función (10). Se prosiguió con la implementación del PID completo como se explicó en la sección 6.5.1, logrando que los agentes se movieran de forma satisfactoria hasta los puntos finales objetivo.

Cabe destacar que una vez se cumplió la condición de error objetivo, el programa abandona el ciclo principal de control y cambia a un estado de finalización. Primero, se detienen a todos los agentes con la función (9). Segundo, se recolectan las nuevas posiciones actuales de los agentes luego del movimiento. Esto con el fin de actualizar las posiciones iniciales para la propagación del algoritmo. Finalmente, se reinicia la constante *ciclos* a un valor de 0. Esto último sirve para que el algoritmo nuevamente se propagase la cantidad de ciclos establecida y todo el proceso se pudiese repetir hasta que se llegase a una distribución de Voronoi satisfactoria.

## 8.2. Validación de la implementación en Robotat

Con el controlador diseñado e implementado en su totalidad, el siguiente paso fue realizar pruebas dentro del ecosistema Robotat junto con los agentes. Para esto, se utilizaron 8 agentes Pololu 3Pi+ pues este era el número mínimo de agentes con el que el algoritmo funcionó de forma deseada a su vez de que fuese capaz de converger. Para poder realizar dichas mediciones se utilizaron dos computadoras: una que corría el algoritmo principal, encargada de propagar el diagrama de Voronoi y enviar los datos a los agentes; y una segunda encargada solamente de realizar mediciones del servidor Robotat.

### 8.2.1. Análisis de movimiento de los agentes

Se realizó una serie de pruebas para evaluar diferentes casos. Estos plantearon distintas formaciones iniciales para poder así corroborar la resiliencia al cambio del algoritmo a la vez de observar el comportamiento que este tuvo al variar las posiciones iniciales. Cabe destacar que, al tener 8 agentes, la formación final siempre será la misma distribución teórica vista en la Figura 12, mostrada en el capítulo 7.

#### Formación inicial central

En las primeras pruebas se optó por organizar a todos los agentes en el centro del área de trabajo. La formación inicial aparenta la de un círculo o cuadrado alrededor del origen del Robotat a la vez que cada agente se encontraba viendo en dirección opuesta al centro. Luego, el algoritmo se dejó correr hasta que el sistema convergiera. En la Figura 13 se puede ver el posicionamiento inicial descrito, mientras que en la Figura 14 se observa cómo los agentes se encontraban distribuidos a través del área de trabajo de $3 \times 3$ metros.

Aparte del comportamiento observado en el Robotat durante la ejecución se realizó una simulación teórica, donde los puntos iniciales de los agentes se ingresaron en el simulador y se calcularon las trayectorias que estos toman. Estos resultados se observan en las figuras 15, 16 y 17. En estas, la posición inicial de los agentes se representa con una $x$, la posición final teórica con un círculo y la final experimental con un triángulo.

En la Figura 15 se observa cómo la trayectoria teórica de los agentes es una línea recta o ligeramente curva, donde estos siguieron un camino relativamente sencillo y llegaron de forma directa al punto objetivo. De igual forma, se puede observar en la Figura 16 cómo los agentes se acercaron al punto, pero no quedaron exactamente en el punto de la distribución de Voronoi. No obstante, esto es más evidente en la Figura 17, donde se observa que los agentes quedan al borde del círculo teórico, mas no lo alcanzan. Esto último se debe al controlador, ya que se mencionó que este funciona con un 10 % de error.

Asimismo, las trayectorias experimentales no siempre siguieron el camino más directo, lo que se debió a que, a diferencia del simulador, el algoritmo se pausaba cada $n$ ciclos para dar lugar al ciclo de control. Asimismo, como el controlador siempre tendrá el 10 % de error de posición y debido a que este se propaga luego de cada ciclo de control, los agentes no pudieron seguir exactamente las rutas teóricas. Esto se evidencia en el agente morado, el cual se desvió de la ruta más adecuada al hacer dos curvas antes de llegar al punto. Sin embargo, este caso resultó ser uno de los mejores, pues los agentes llegaron a los puntos objetivos de forma rápida y directa, sin necesidad de realizar vueltas innecesarias ni perderse durante la propagación.

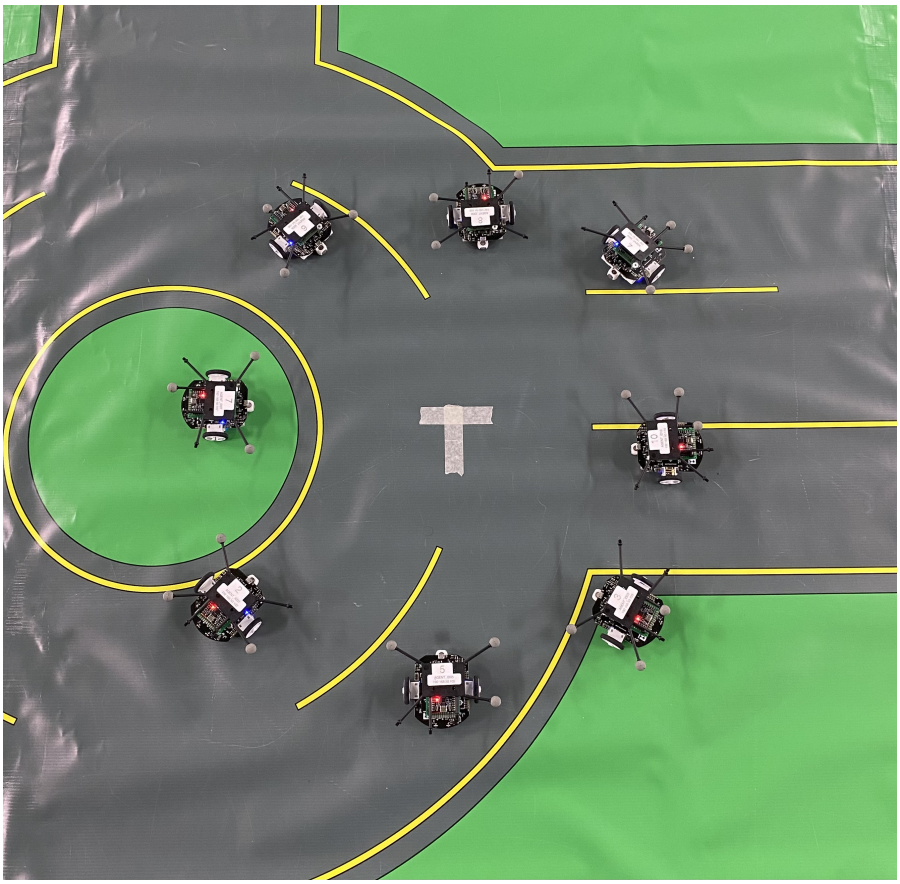


Figura 13: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat.

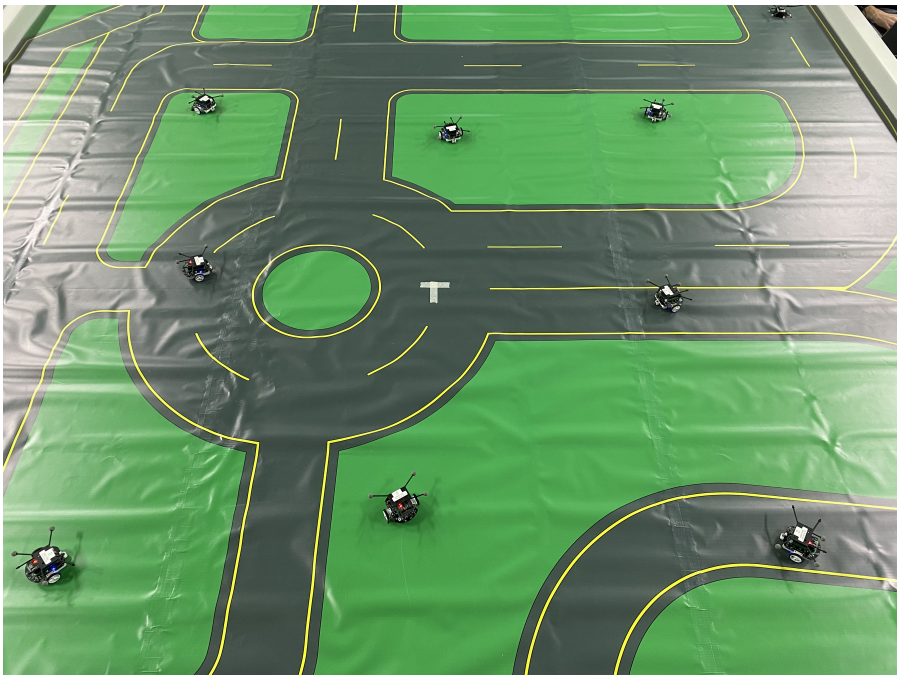


Figura 14: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat.

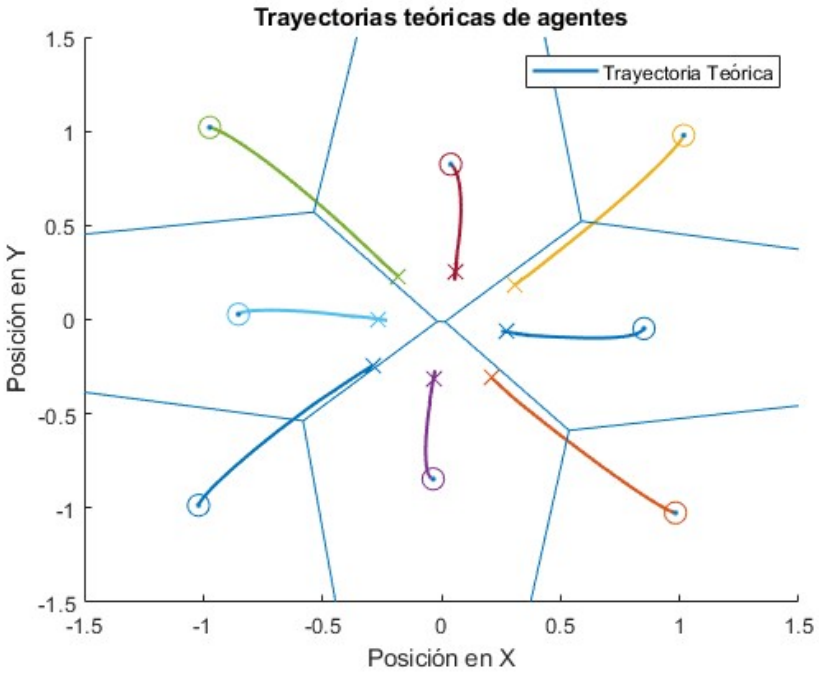



Figura 15: Trayectorias teóricas de agentes para primera formación central.

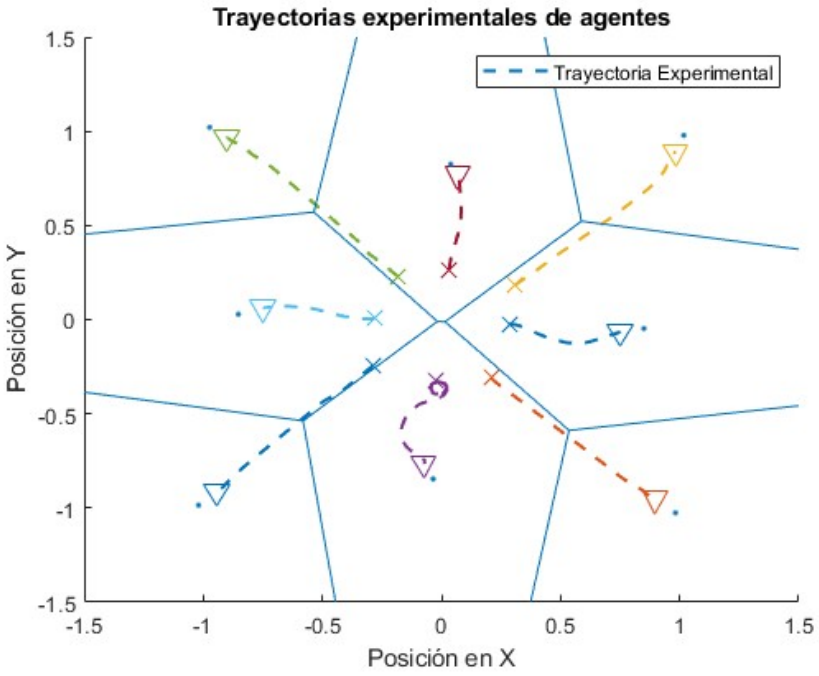



Figura 16: Trayectorias experimental de agentes para primera formación central.

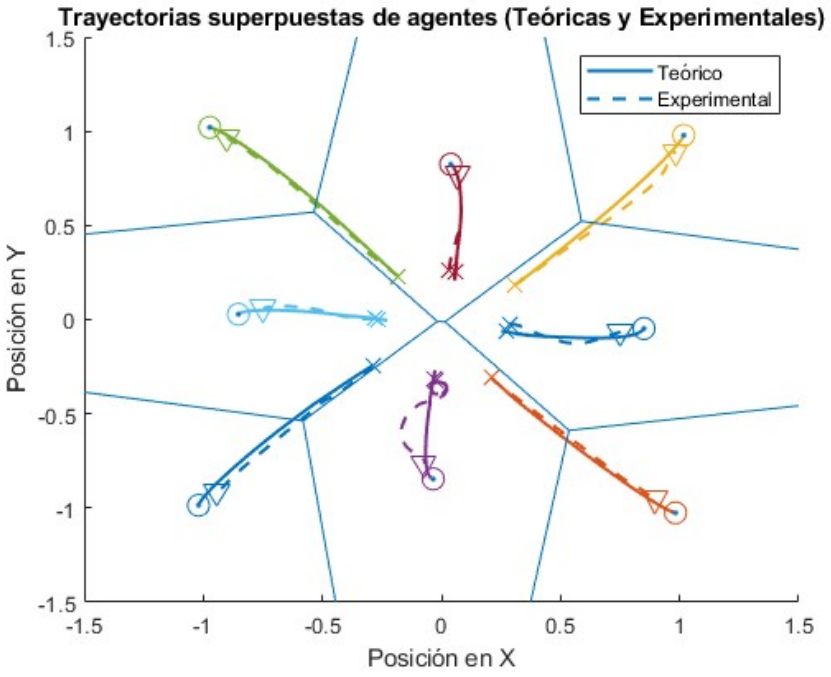



Figura 17: Comparación de trayectorias de agentes para primera formación central.

Se realizó otra prueba con la misma distribución inicial, con la única diferencia de una leve variación en el posicionamiento, como se observa en la Figura 18. Esto tuvo un impacto leve pero notable en las trayectorias tanto teóricas como experimentales de los agentes. Se puede observar en la Figura 20 que las trayectorias teóricas son completamente rectas, demostrando así que los agentes tomaron el camino más directo y eficiente. Un factor que contribuyó a esta mejora en la trayectoria es que el posicionamiento levemente alterado se encontraba mejor alineado con el diagrama de Voronoi final. Como consecuencia, los agentes no tuvieron que realizar ningún desvío sino que pudieron ir directamente al objetivo.

Por otro lado, en la Figura 21 se puede observar que esta vez más agentes tomaron trayectorias no adecuadas. Esto se evidenció en las curvas innecesarias que los agentes azul, morado y amarillo tomaron para llegar al objetivo. No obstante, al analizar la Figura 22 se observa que, una vez más, los agentes llegaron al punto objetivo con un error de posición aproximado del 10 %. Asimismo, en esta figura se observa con mayor claridad que, a pesar de la discrepancia entre las trayectorias teóricas y experimentales, los agentes sí llegaron a los puntos objetivos y generaron un sistema convergente, además de una distribución de Voronoi satisfactoria.

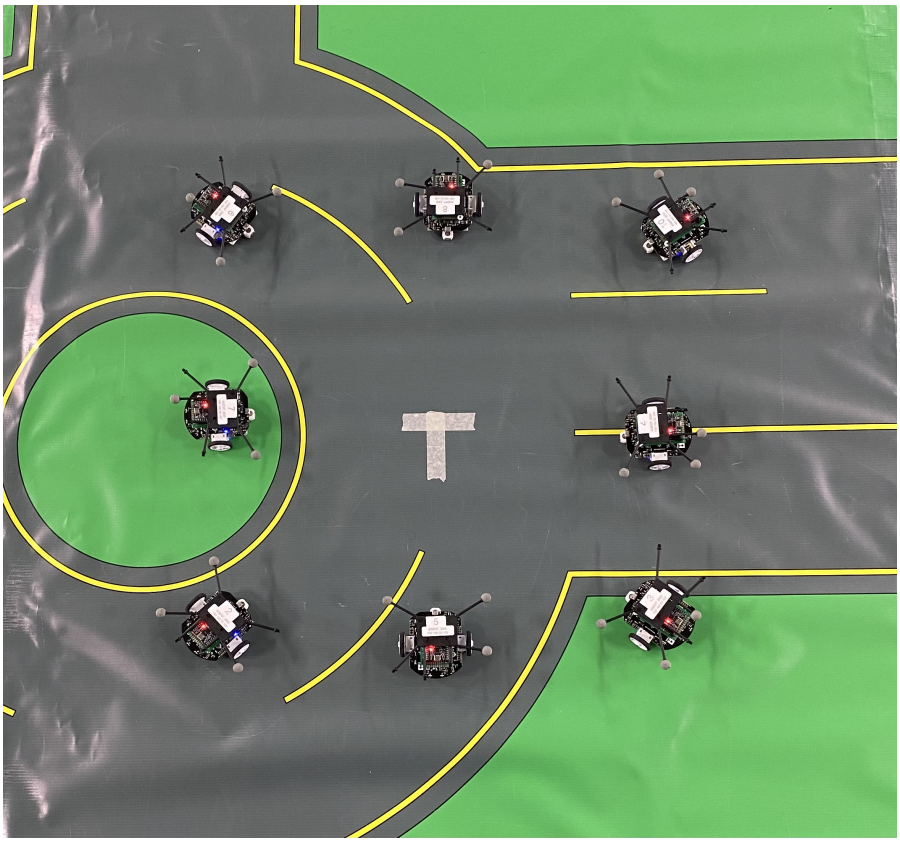


Figura 18: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación central.

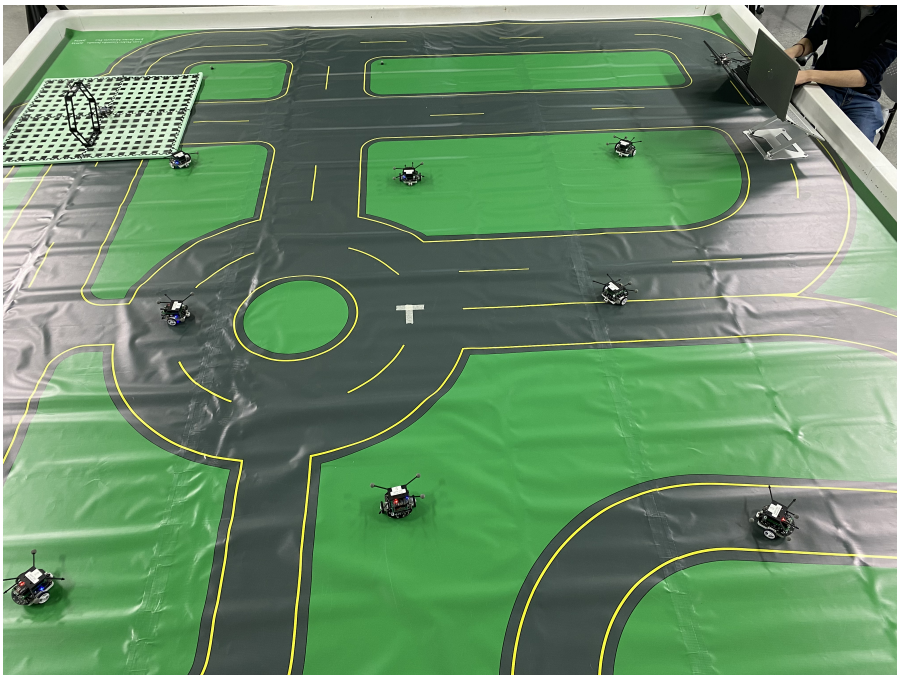


Figura 19: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación central.

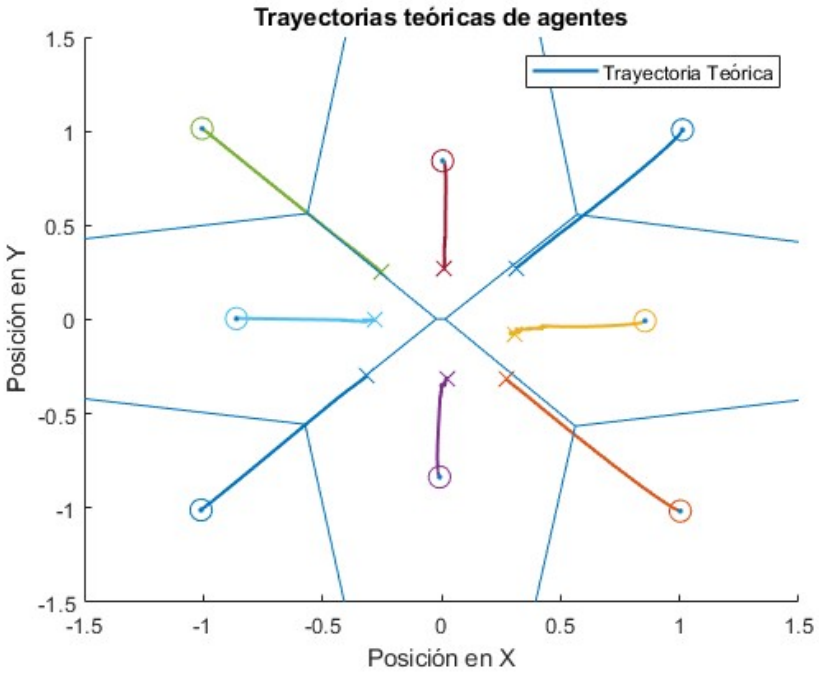



Figura 20: Trayectorias teóricas de agentes para segunda formación central.

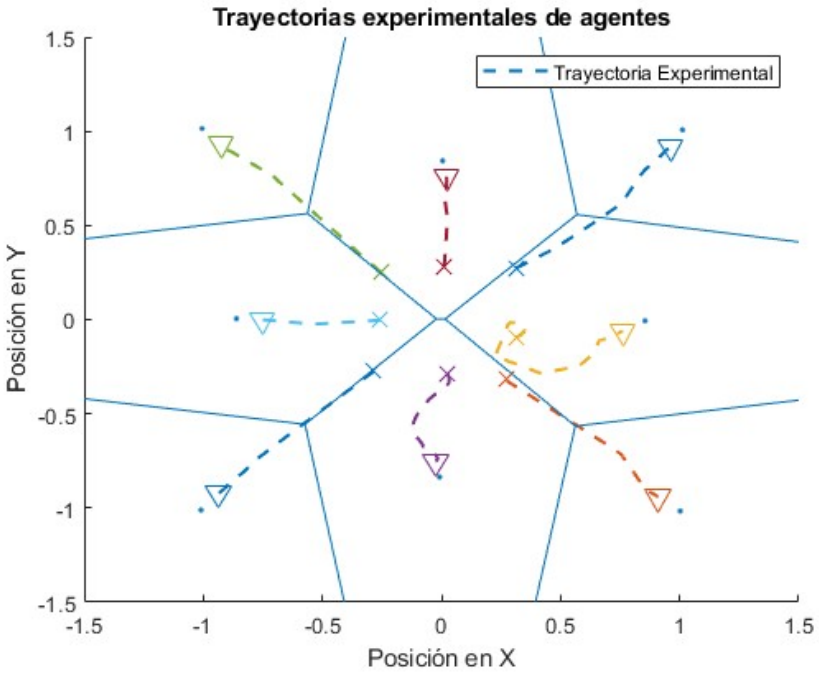



Figura 21: Trayectorias experimental de agentes para segunda formación central.

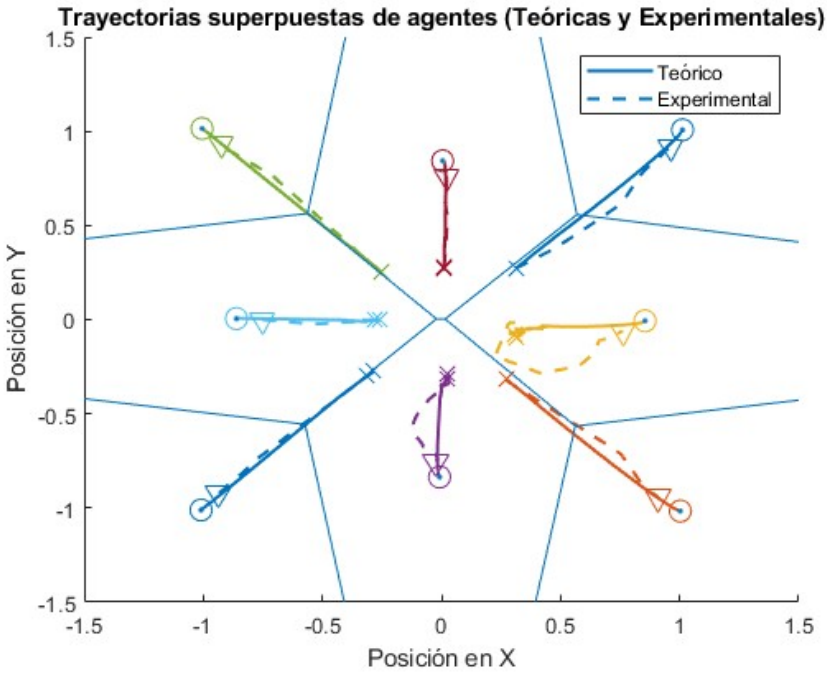



Figura 22: Comparación de trayectorias de agentes para segunda formación central.

### Formación inicial aleatoria

El concepto general de estas pruebas se mantuvo constante respecto a las primeras dos pruenas explicadas en la sección anterior. El cambio principal que este sufrió es el posicionamiento inicial, ya que en este caso se intentó una configuración aleatoria en vez de una formación estructurada. Esto tuvo un impacto notable en el tiempo de resolución del algoritmo, pero el resultado final demostró ser una distribución satisfactoria. En las figuras 23 y 24 se observa el posicionamiento inicial y final del sistema analizado.

Al analizar las trayectorias teóricas en la Figura 25 se puede observar cómo no todos los agentes llegan de forma directa a los puntos objetivos. Además, los agentes corinto, naranja y lima mostraron vibraciones en su trayectoria, efecto causado por el método utilizado para propagar el algoritmo. Cuando se propaga en un ciclo, el algoritmo le dicta al agente moverse una distancia pero en el siguiente ciclo este retrocede un poco. No obstante, el retroceso del agente es menor al avance, dando como resultado un avance neto.

Algo peculiar es el posicionamiento final y las trayectorias de los agentes en la prueba experimental, como se observa en la Figura 26. Esto porque los agentes sí se distribuyeron de forma correcta y dieron como resultado un diagrama de Voronoi satisfactorio, mas todos los agentes con excepción del celeste cambiaron posiciones entre sí, fenómeno que se evidencia en la Figura 27. En este caso se cumplió nuevamente lo explicado en la sección 8.2.1, cuarto párrafo, pues se observa el 10 % de error esperado para algunos agentes. Sin embargo, no se puede omitir que algunos de los agentes muostraron un error bastante mayor al 10 % esperado, lo que se sigue atribuyendo a la diferencia en la propagación del algoritmo entre las trayectorias teóricas y experimentales.

En cuanto a las trayectorias, se observó que algunos de los agentes tomaron rutas más largas de lo esperado. Sin embargo, en lo que respecta al control de cobertura, este sistema funciona de manera adecuada. Esto se debe a que el algoritmo busca distribuir a todos los agentes de tal forma que cubran por completo el área de trabajo. Por lo tanto, la trayectoria que los agentes sigan pasa a ser una prioridad secundaria, siempre que el sistema final sea convergente y muestre un diagrama de Voronoi satisfactorio. Por lo explicado con anterioridad y el hecho de que los agentes cambiaron de posición final entre sí, se consideró este caso como uno de los peores esperados, siendo aceptable pero no deseable.

Figura 23: Posiciones iniciales aleatorias de 8 agentes Pololu 3Pi+ en el Robotat.

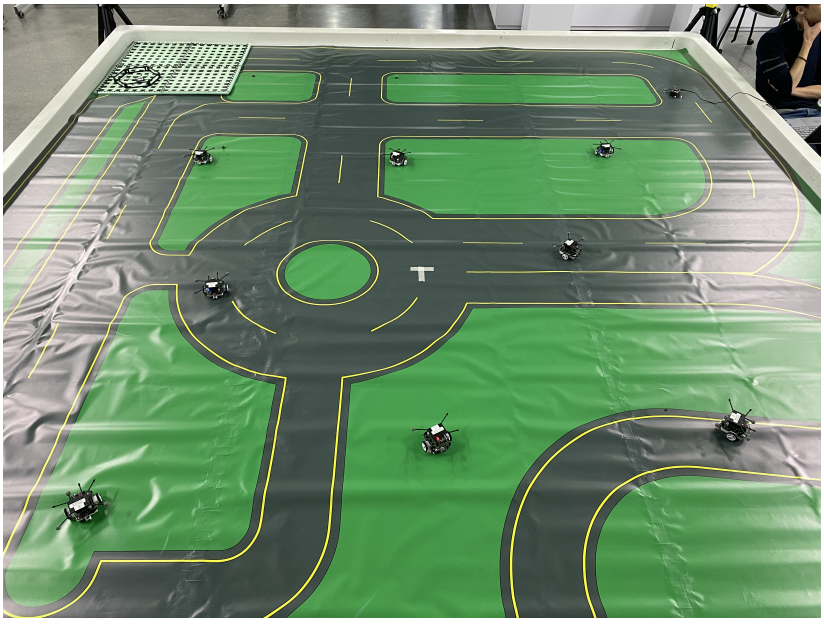


Figura 24: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat.

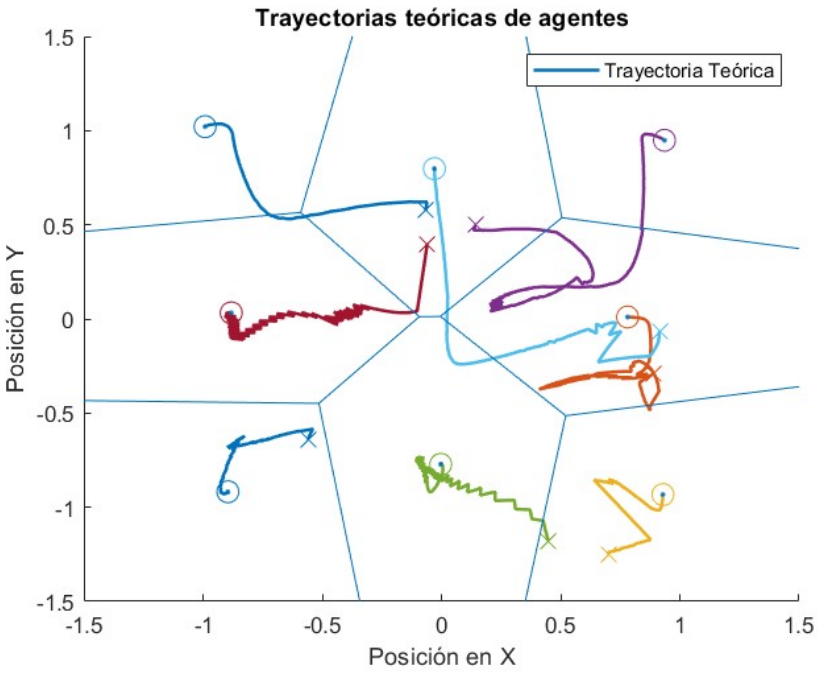



Figura 25: Trayectorias teóricas de agentes para primera formación aleatoria.

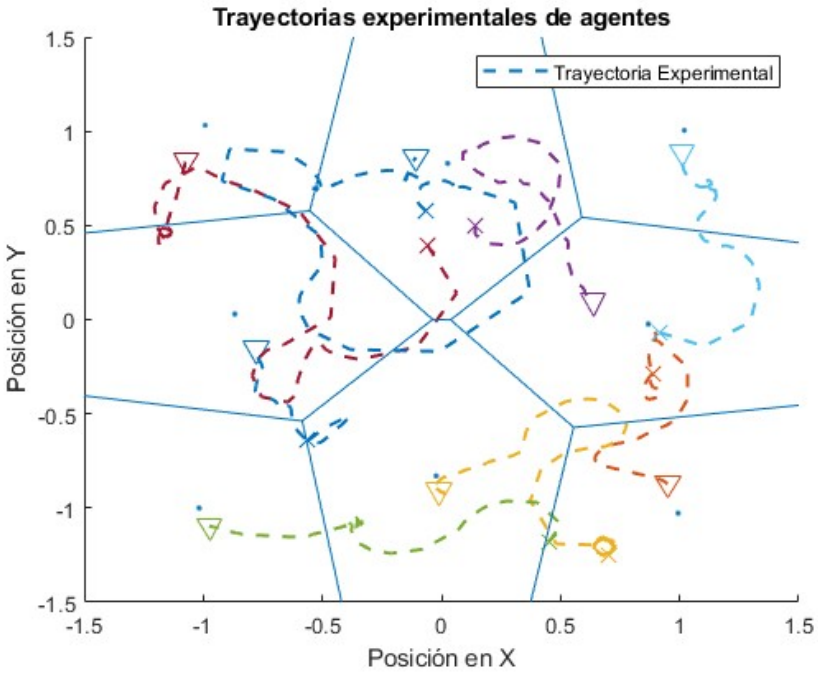



Figura 26: Trayectorias experimental de agentes para primera formación aleatoria.

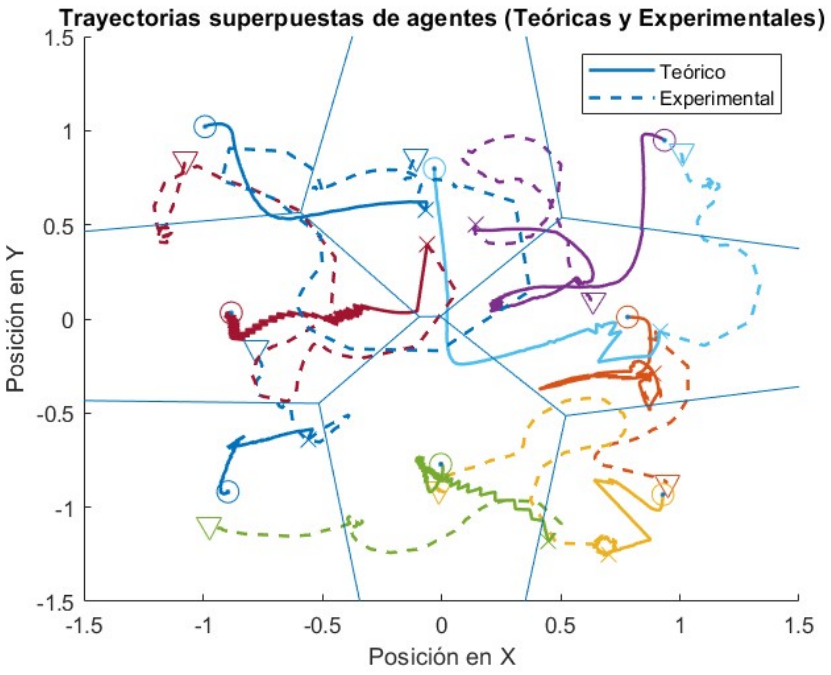



Figura 27: Comparación de trayectorias de agentes para primera formación aleatoria.

En la segunda prueba se observó un comportamiento similar al caso anterior, donde las trayectorias teóricas no siempre fueron las más adecuadas. Sin embargo, el sistema final es convergente y mostró un diagrama de Voronoi satisfactorio. Al analizar la Figura 30 es evidente que los agentes corinto y lima mostraron vibraciones, mientras que los naranja, celeste y azul tomaron el camino más eficiente al punto objetivo.

El comportamiento más peculiar fue el del agente azul, pues en la Figura 31 se observa cómo este se mantuvo moviéndose alrededor del punto objetivo en vez de quedarse quieto. No obstante, parece ser que este comportamiento fue una anomalía pues no tuvo precedente entre las pruebas realizadas.

Finalmente, en este caso se observó que sí se cumplió el 10 % de error de posición esperado por agente, como se evidencia en la Figura 32. También se puede observar que en esta prueba, las trayectorias teóricas fueron casi idénticas a las experimentales, con la diferencia que las exprerimentales fueron más suaves. Esto último se debió a que el simulador realizó movimientos bruscos y cambios de dirección abruptos, mientras que el controlador los suavizó para evitar forzar los motores del Pololu 3Pi+. Es por esto que se pudo considerar esto como un caso ideal para una formación inicial aleatoria.

Figura 28: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación aleatoria.

Figura 29: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación aleatoria.

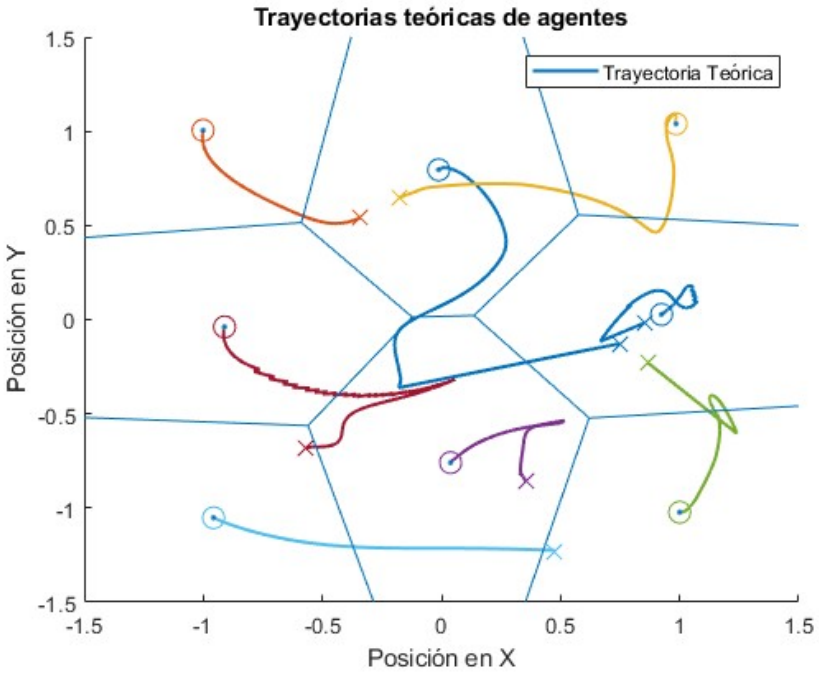



Figura 30: Trayectorias teóricas de agentes para segunda formación aleatoria.

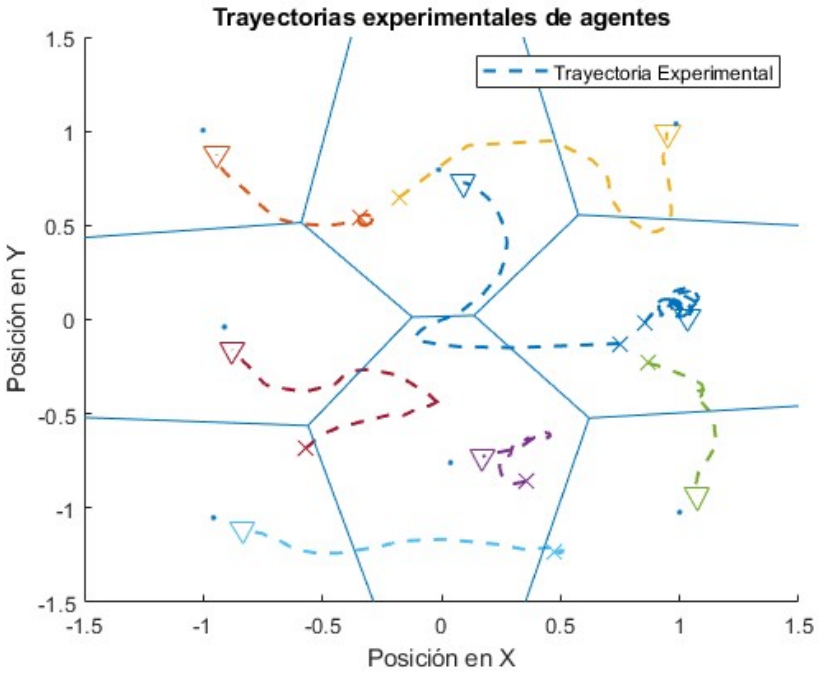



Figura 31: Trayectorias experimental de agentes para segunda formación aleatoria.

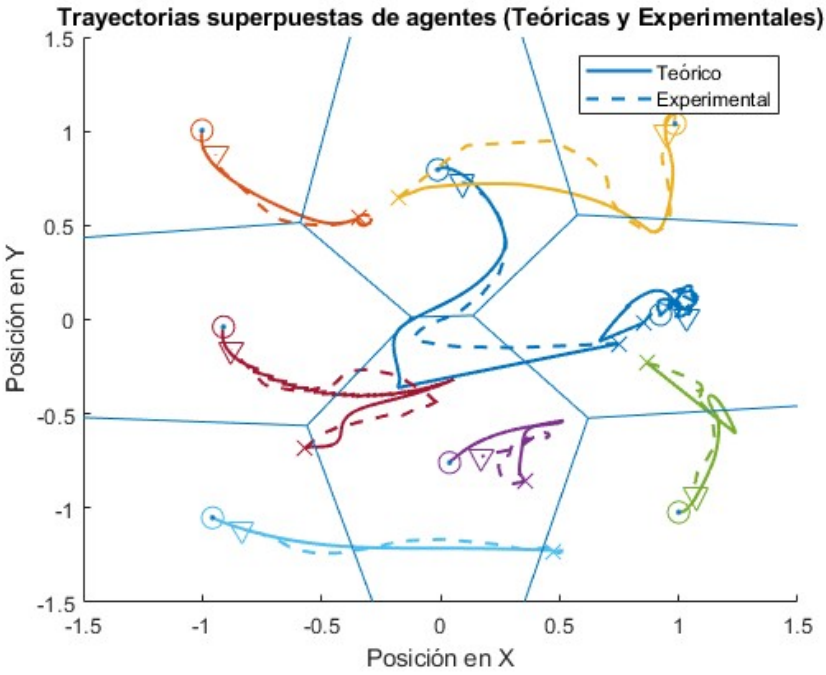



Figura 32: Comparación de trayectorias de agentes para segunda formación aleatoria.

Una vez desarrollado, validado e implementado el algoritmo de control de cobertura seleccionado, se concluyó que este mostró un funcionamiento satisfactorio. Lo que siguió es agregar la funcionalidad de implantar puntos de concentración dentro del área de trabajo. Esto para así poder empezar a simular focos de contaminación y poner a prueba la adaptabilidad del algoritmo a sistemas dinámicos.

# CAPÍTULO 9

## Implementación de puntos de concentración para simular puntos de contaminación en el lago de Atitlán

El algortimo logró satisfacer la problemática de control de cobertura dentro del ecosistema Robotat, logrando cubrir el área de trabajo establecida dentro de este. Lo que prosiguió fue la implementación de puntos de concentración para así simular los focos de contaminación observados en cuerpos de agua, como lo es el caso en el lago de Atitlán. El concepto general de este cambio fue que, en caso de detectar un punto de concentración, el algoritmo debió tenerlo en cuenta y redistribuir a los agentes de tal manera que se cumpliera lo siguiente:

- Mantener una formación que cubra la mayor cantidad posible del área de trabajo.
- Atraer y agrupar a los agentes alrededor del punto de concentración.
- Lograr un sistema convergente que genere un diagrama de Voronoi satisfactorio.

## 9.1. Diseño e implementación del punto de atracción

En el capítulo 7 se explicó en profundidad el diseño completo del algoritmo implementado. Para poder agregar una función de atracción, se tuvieron que agregar los siguientes argumentos de entrada:

- Coordenadas $(x, y)$ del punto de atracción, para determinar dónde se encontrará el centro de este.
- Radio de influencia del punto de atracción.

- Constante de atracción, la cual dicta qué tanto afectará el punto de atracción a los agentes.

Luego de declarar estos valores, se definieron dos funciones que se utilizaron dentro del algoritmo. La primera empleó un concepto de atracción basado en decaimiento exponencial para su funcionamiento. Esto significa que la influencia del punto de atracción disminuye exponencialmente a medida que aumenta la distancia desde su punto central. Con este concepto aclarado se presentaron las funciones de la siguiente forma:

$$I = \exp\left(-\frac{\|\mathbf{p} - \mathbf{c}\|}{r}\right), \tag{24}$$

donde $I$ es un valor entre 0 y 1, $\mathbf{p} = (x, y)$ son las coordenadas del punto donde se desea calcular la influencia, $\mathbf{c} = (\mathbf{cx_k}, \mathbf{cy_k})$ son las coordenadas del centro del área de influencia, y $r$ es el radio que define el área de influencia.

$$Z_{\text{grid}}(x, y) = \sum_{k=1}^{n} \exp\left(-\frac{\|\mathbf{p} - \mathbf{c}_j\|}{r}\right), \tag{25}$$

donde $Z_{\text{grid}}(x, y)$ es el valor acumulado de concentración en el punto $(x, y)$ de la cuadrícula, $\mathbf{p} = (x, y)$ son las coordenadas del punto donde se desea calcular la influencia, $\mathbf{c}_j = (cx_j, cy_j)$ son las coordenadas del centro del área de concentración $j$-ésima, y $r$ es el radio que define el área de influencia del área de concentración $j$-ésima.

En la inicialización, ahora se agregó una nueva sección para la toma de datos del punto de atracción. Si se utiliza el simulador, primero se declaran los valores del radio de influencia, representado por radio_influencia, y la constante de atracción *beta*. Luego, las coordenadas $x$ y $y$ fueron generadas con la función randi(*randi_val*) de Matlab con el mismo valor de 50 para la constante randi_val utilizada para la simulación del sistema original.

Ahora bien, si se decidió utilizar mediciones del Robotat, primero se tuvieron que seguir los pasos de conexión al servidor. Luego, se declaró un marcador que sirvió como el punto de atracción físico dentro del ecosistema Robotat. Después, se utilizó el mismo concepto de medición descrito en la sección 7.4.3 del capítulo 7. Siguiendo esta guía se obtuvieron las coordenadas $(x, y)$ del marcador.

Una vez se obtuvieron las posiciones iniciales y el área de concentración, estos se unificaron en un único vector. Este vector se utilizó como argumento de entrada para la función (25), donde los primeros dos valores del vector representaron las coordenadas de $\mathbf{p}$ mientras que el tercero fue el radio $r$. Finalizada la declaración de constantes y de la posición inicial del punto de concentración, el algoritmo no recibe cambio alguno hasta dentro del ciclo principal. Dentro de este se implementó un cálculo de áreas de concentración luego de calcular el centroide y antes de calcular la fuerza de repulsión explicada en la sección 7.3 del capítulo 7. La ecuación utilizada fue la siguiente:

$$I_{Tot} = \sum_{k=1}^{n} \beta \cdot I_k \cdot \left( \begin{bmatrix} cx_k \\ cy_k \end{bmatrix} - \mathbf{p}_j \right), \tag{26}$$

donde $I_{Tot}$ es el desplazamiento acumulado hacia el área de concentración, $\beta$ es un factor de ajuste que regula la magnitud del desplazamiento, $I_j$ es la influencia del área de concentración $j$-ésima calculada con la función (24), $\begin{bmatrix} cx_j \\ cy_j \end{bmatrix}$ son las coordenadas del centro del área de concentración $j$-ésima, y $\mathbf{x}_j(t)$ representa la posición actual del agente $j$.

Con el valor $I_{Tot}$ calculado, se prosiguió con el cálculo de la posición futura. Esta ecuación se mantuvo muy similar a la vista en la sección 7.3 del capítulo 7, la cual sufrió un ligero cambio. Ahora se tuvo que sumar este valor calculado al final de la ecuación (21) para así obtener la versión final de esta.

$$\mathbf{x}_j(t + \Delta t) = \mathbf{x}_j(t) + \Delta t \left( \mathbf{c}_j(t) - \mathbf{x}_j(t) - \phi_{j,h} \right) + I_{Tot}, \tag{27}$$

donde $x_j(t + \Delta t)$ es la posición futura, $\Delta t$ es el paso del tiempo, $x_j(t)$ es la posición actual del punto $j$, $\mathbf{c}_j(t)$ es el centroide en el punto $j$, $\phi_{j,h}$ es su respectivo coeficiente de colisión y $I_{Tot}$ es el desplazamiento acumulado hacia el área de concentración.

A partir de este punto el algoritmo no presentó más cambios. De igual forma, el ciclo de control se mantuvo exactamente igual al explicado en la sección 8.1.1 del capítulo 8. Una vez finalizado el ciclo de control, se agregó una sección para la actualización de la posición actual del punto de concentración, la cual se ubicó entre la actualización de la posición de los agentes y el reinicio de la constante *ciclos* a un valor de 0. Esto último se realizó con el fin de hacer el algoritmo dinámico y reactivo al cambio. Es decir, durante el ciclo de propagación del algoritmo, el punto de concentración no se actualizaba. Sin embargo, sí lo hacía cada vez que se completaba un ciclo completo de propagación y control.

## 9.2. Validación del algoritmo con puntos de concentración

Al ser una variación del algoritmo original, se sometió a un proceso de validación tanto simulado como implementado con el ecosistema Robotat. Para realizar esto con mayor facilidad, se implementó dentro del programa principal de tal forma que fue omitido dependiendo del valor de la constante condiciones_iniciales. De esta forma se tuvo ambas versiones del algoritmo coexistiendo en el mismo programa.

### 9.2.1. Simulador del algoritmo con puntos de concentración

La implementación de puntos de concentración no tuvo un impacto en el cálculo y propagación de los diagramas de Voronoi en sí. La diferencia que este presentó respecto al original fue la atracción acumulada $I_{Tot}$. Es por esto que las limitantes discutidas en las secciones

7.4.1 y 7.4.2 del capítulo 7 aplicaron para esta versión. No obstante, al incorporar una nueva constante de atracción, el algoritmo tuvo otra variable cuyo incremento o disminución impactaron el funcionamiento de este.

### Impacto del factor de atracción

El factor de atracción $\beta$ influyó en la magnitud de desplazamiento que el punto de concentración tiene sobre los agentes. Esto significó que al incrementarlo, el punto de concentración tuvo una mayor influencia sobre los agentes y estos se acercaron más a este. En cambio, si este se disminuía, el comportamiento contrario sucedió y los agentes se vieron menos atraídos al punto. Para encontrar el valor ideal de $\beta$ se realizaron una serie de pruebas donde se dejaron constante los siguientes parámetros:

- El número de agentes, en 8.
- El paso de tiempo, en 0.1.
- El radio de atracción del punto de concentración, en 0.6.
- Las coordenadas del punto de atracción, siendo 0.1 para $x$ y 0 para $y$.

Como punto de comparación se utilizó una distribución satisfactoria de un sistema convergente con 8 agentes, como se puede observar en la Figura 33. Asimismo, las pruebas comenzaron con los agentes ya distribuidos en el área de trabajo de esta forma. Esto con el fin de poder observar de mejor forma el impacto que tiene el punto de concentración sobre una distribución satisfactoria. En la primera prueba se declaró un valor de $\beta$ de 0.04. Este mostró un impacto insignificante en el sistema, pues se observa en la Figura 34 que los agentes de la derecha se encontraban ligeramente desplazados respecto a 33.

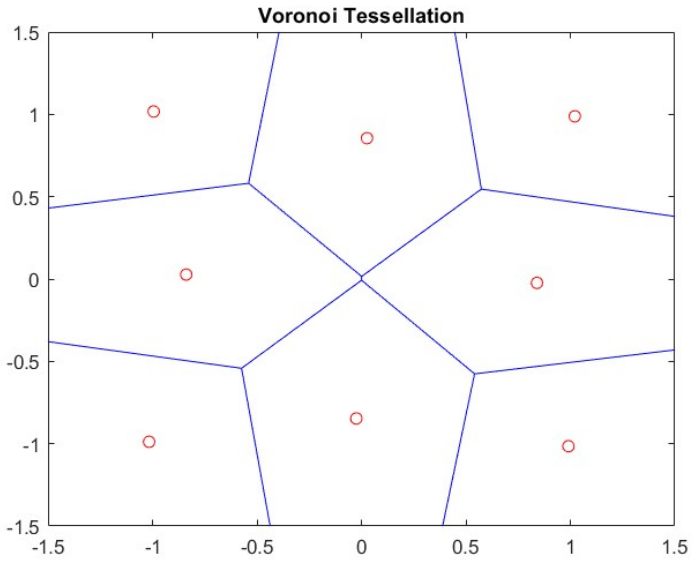



Figura 33: Diagrama de Voronoi base y convergente para 8 agentes.

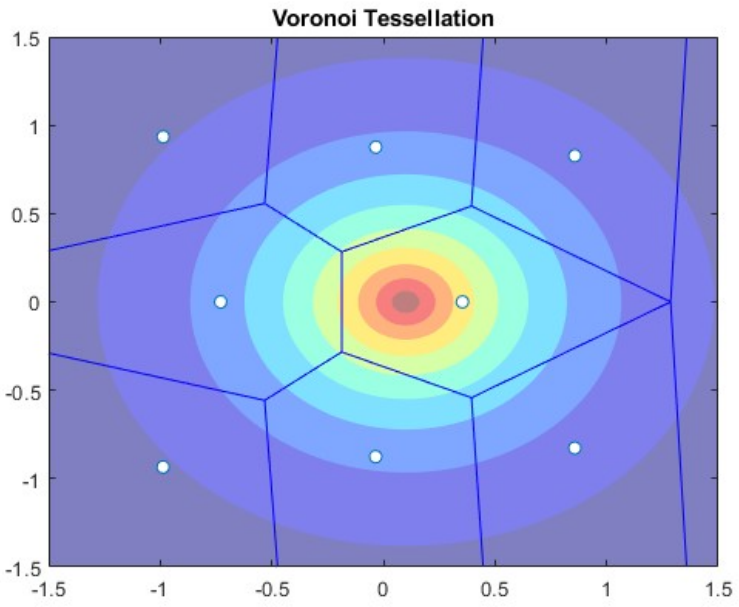



Figura 34: Diagrama de Voronoi para 8 agentes con punto de atracción y factor de atracción de 0.04.

La siguiente pruba fue con un valor $\beta$ de 0.06. En este fue más evidente el impacto del desplazamiento, pues esta vez sí afectó de forma global a todos los agentes. En la Figura 35 se observa que los 8 agentes se encontraban desplazados de su posición final respecto a la vista en 33. No obstante, los agentes aún se encontraban demasiado alejados del punto de concentración, resultando en una formación ineficiente.

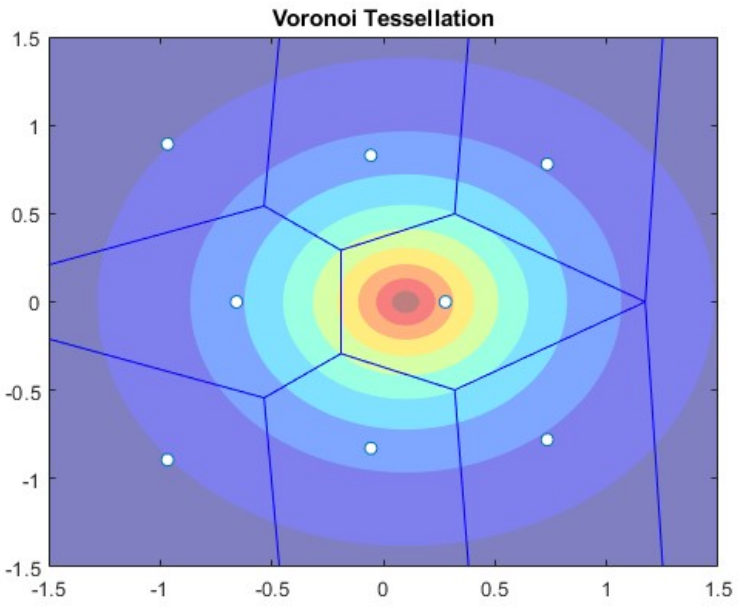



Figura 35: Diagrama de Voronoi para 8 agentes con punto de atracción y factor de atracción de 0.06.

Al utilizar un factor de atracción de 0.08 el impacto sobre el sistema fue más cercano al desado. Esto debido a que, como se muestra en la Figura 36, ahora todos los agentes se trasladaron alrededor del punto de concentración. A pesar de este movimiento notable, la formación observada sigue siendo la misma que la del sistema base solamente que más compacta hacia el centro. Un dato que se debe destacar es que los agentes mostraron cierto nivel de vibración, pues el algoritmo principal luchó contra el punto de atracción. Consecuentemente el agente no se quedó quieto en un solo punto final.

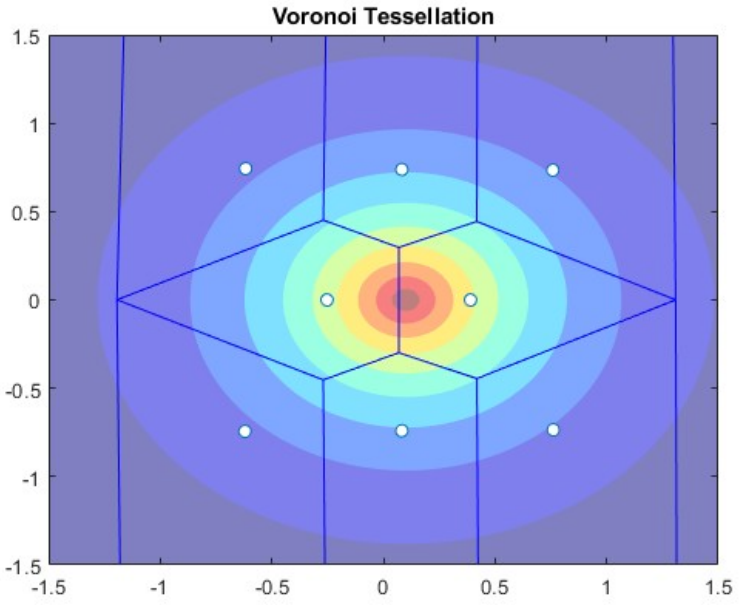



Figura 36: Diagrama de Voronoi para 8 agentes con punto de atracción y factor de atracción de 0.08.

El siguiente valor de $\beta$ utilizado fue de 0.1. Este mostró un desplazamiento considerable de los agentes hacia el punto de concentración. Estos se encontraban mejor ubicados alrededor del punto mientras mantenían la formación deseada y cubrían el área de trabajo de manera satisfactoria, como se evidencia en la Figura 37. Sin embargo, las vibraciones se mantuvieron casi iguales a lo observado con el 0.08, con la ventaja añadida de una mejor formación.

Finalmente, se realizó una prueba con un valor de 0.14. Esta fue la última debido a que, a partir de este valor, los agentes se agrupaban demasiado cerca uno del otro además de incrementar la vibración. Esto causó que el sistema siempre se encontrase oscilando y que no lograse cubrir todo el área de trabajo, como se evidencia en la Figura 38.

Con base en estas pruebas se concluyó que el valor de $\beta$ idóneo para realizar pruebas en el ecosistema Robotat y en el simulador en general fue de 0.1. Esto porque mostró el mejor balance entre vibración del sistema final, organización de formación y distribución en el área de trabajo.

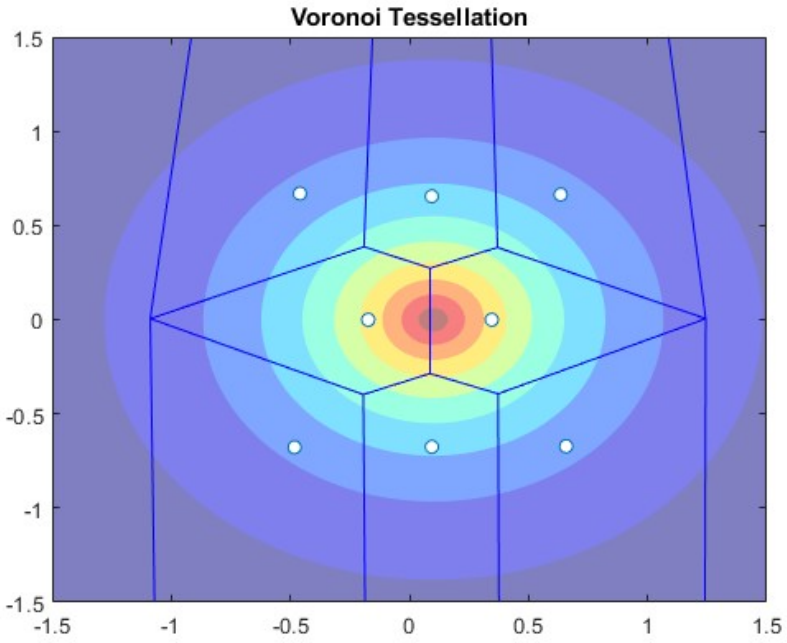



Figura 37: Diagrama de Voronoi para 8 agentes con punto de atracción y factor de atracción de 0.1.

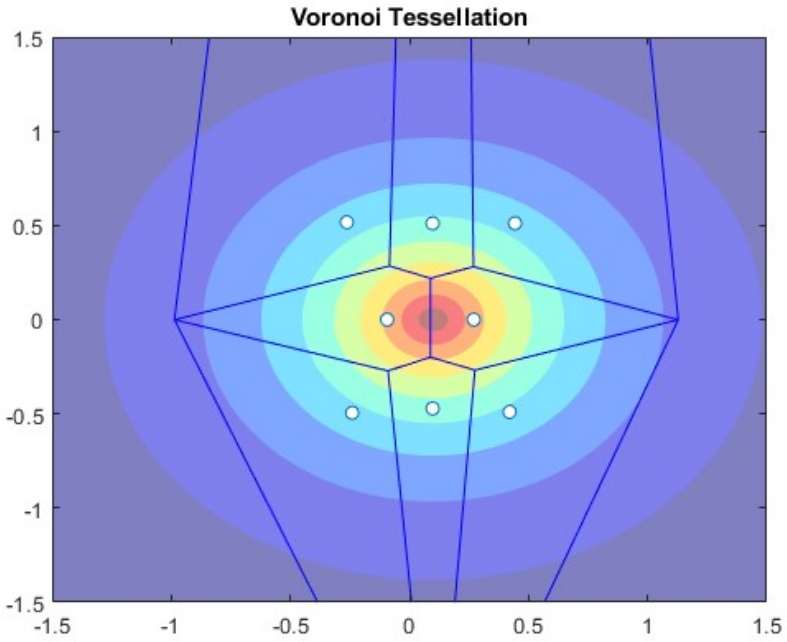



Figura 38: Diagrama de Voronoi para 8 agentes con punto de atracción y factor de atracción de 0.14.

### Impacto del radio de atracción

Una vez determinado el valor adecuado para el factor de atracción $\beta$, se prosiguió a variar el radio de atracción del punto de concentración. Esto para analizar la influencia que esto tenía sobre el algoritmo en general. Se delimitaron las siguientes constantes para los parámetros del sistema:

- El número de agentes, en 8.
- El paso de tiempo, en 0.1.
- El factor de atracción, en 0.1.
- Las coordenadas del punto de atracción, siendo 0.1 para $x$ y 0 para $y$.

El primer caso ya fue analizado en la parte anterior, donde se seleccionó un radio de 0.6 como se puede observar en la Figura 37. Es por esto que la siguiente prueba tuvo un radio de 0.2. En este caso, como el punto de atracción fue tan pequeño, ninguno de los agentes se vio influido por él. Como consecuencia, el sistema fue idéntico al sistema base mostrado en 33.

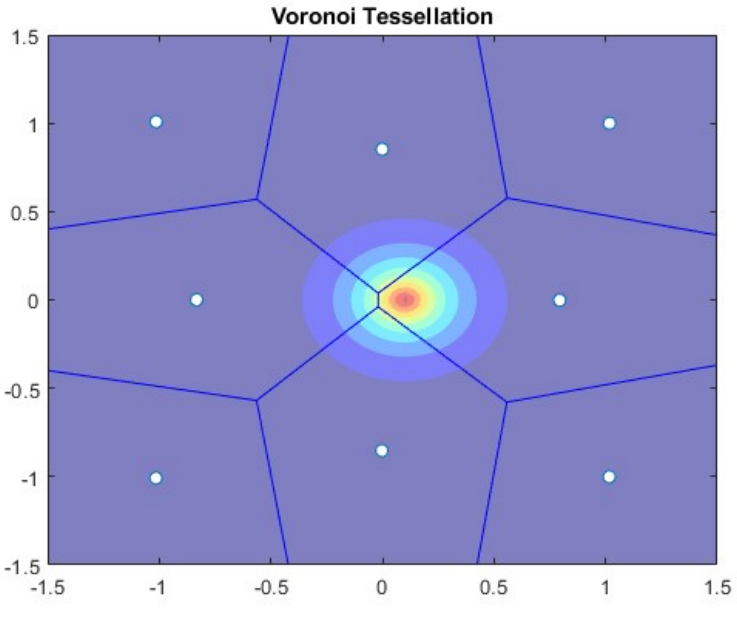



Figura 39: Diagrama de Voronoi para 8 agentes con punto de atracción y radio de atracción de 0.2.

El siguiente caso planteó un punto de concentración con un radio de atracción de 0.4. Esto tuvo como consecuencia un sistema final idéntico al caso con un factor de atracción de 0.06 y un radio de atracción de 0.6. Esto se evidencia al comparar la Figura 40 con la Figura 35, es por esto que se consideró que la distribución final no fue satisfactoria al no agruparse alrededor del punto.

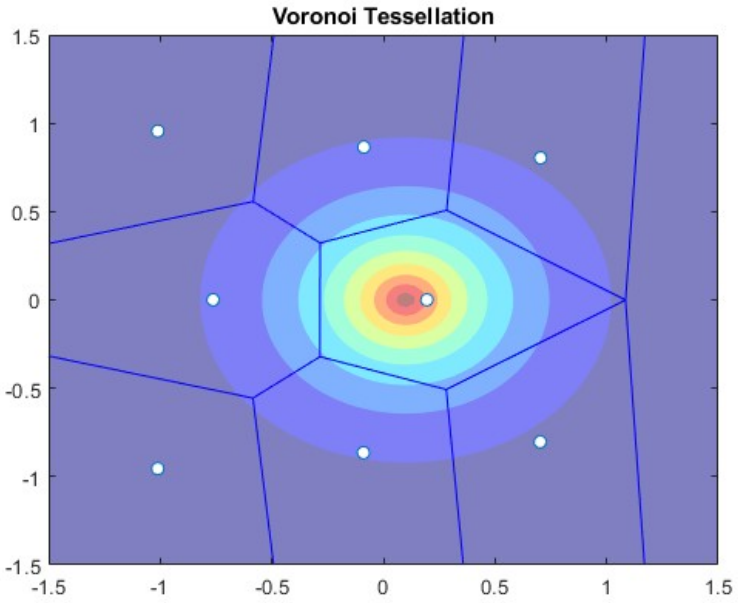



Figura 40: Diagrama de Voronoi para 8 agentes con punto de atracción y radio de atracción de 0.4.

Para el sistema con un valor de 0.8 se observa en la Figura 41 que el sistema fue similar al visto para el radio de 0.6. No obstante, los agentes se encontraban mucho más agrupados hacia el centro que en ese caso, por lo que se consideró la otra configuración como una superior. Asismimo, este sistema mostró vibraciones similares a las observadas con el factor de atracción de 0.14.

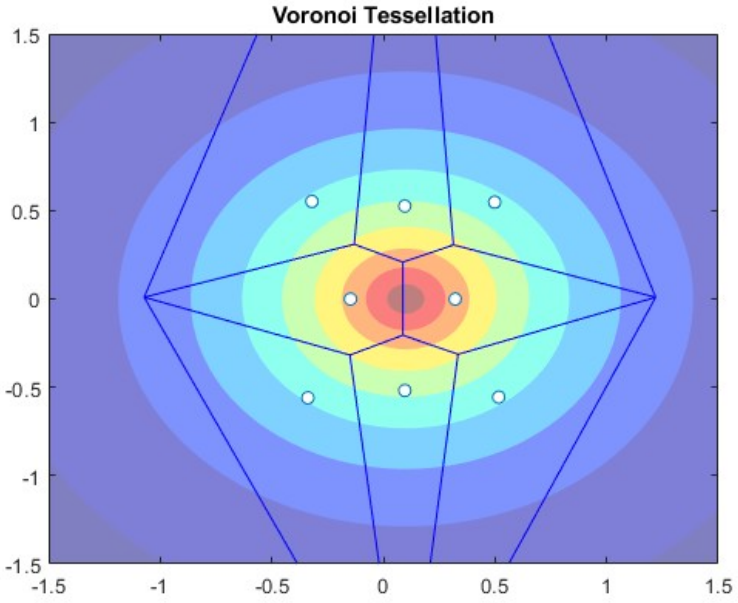



Figura 41: Diagrama de Voronoi para 8 agentes con punto de atracción y radio de atracción de 0.8.

Finalmente, se consideró que la configuración con el radio de atracción de 0.6 y el $\beta$ de 0.1 mostraron el funcionamiento más adecuado. Sin embargo, al realizar más pruebas variando tanto el radio como el factor de atracción, se encontró que hay configuraciones donde el sistema resultante fue casi idéntico al adecuado. Esto implica que existen múltiples configuraciones para plantear el sistema aceptable mostrado en la Figura 37.

### Efecto del posicionamiento del punto de concentración

Una vez delimitado el valor ideal para el radio y factor de atracción, siendo estos 0.6 y 0.1 respectivamente, se prosiguió con las pruebas del algoritmo. Para esto, se varió la posición del punto de concentración, dadas por coordenadas $(x, y)$ para poder así analizar cómo el algoritmo reaccionaba a cada uno de los casos. Es por esto que se mantuvieron constantes los siguientes parámetros:

- El número de agentes, en 8.
- El paso de tiempo, en 0.1.
- El factor de atracción, en 0.1.
- El radio de atracción, en 0.6.

La primera prueba posicionó al punto en las coordenadas $(-0.8, 1)$, dando como resultado lo observado en la Figura 42. En esta se observa que los agentes se agrupaban alrededor del punto de concentración sin romper la formación deseada. Asimismo, estos mantuvieron el área de trabajo cubierta, lo que es evidenciado por los tres agentes que se mantuvieron alejados en la parte inferior. Es posible que al variar alguna de las dos constantes (ya sea el radio o el factor de atracción) se podría obtener una mejor formación, mas la encontrada fue satisfactoria y cumplió con el objetivo de control de cobertura planteado.

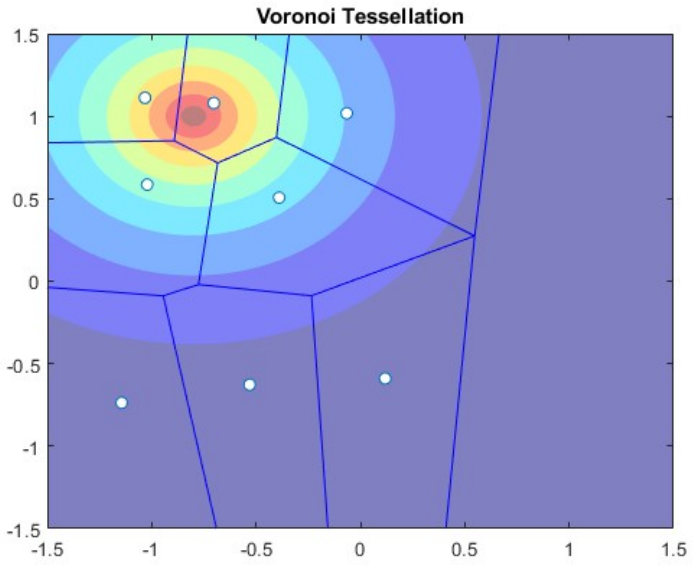



Figura 42: Diagrama de Voronoi para 8 agentes con punto de atracción en coordenadas $(-0.8, 1)$.

Para la siguiente prueba se posicionó en la esquina opuesta, en el punto $(0.8, -1)$. Esto dio como resultado un sistema idéntico al anterior, solamente que agrupado en la esquina opuesta. Como se evidencia en la Figura 43, el algoritmo mantuvo consistencia en los resultados, asegurando así el funcionamiento correcto y la resilencia del mismo. Cabe destacar que el desplazamiento tan marcado observado en las figuras 42 y 43 se debió a que gran parte del punto de concentración se encontraba fuera del área de trabajo.

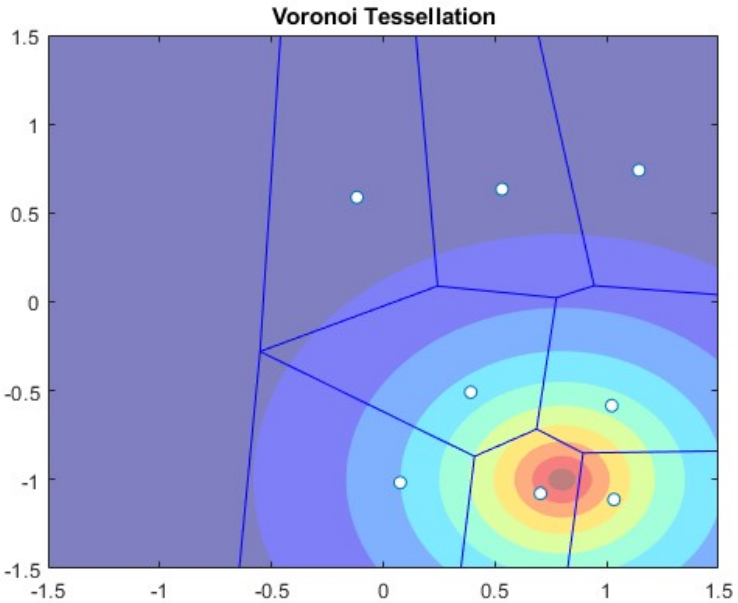



Figura 43: Diagrama de Voronoi para 8 agentes con punto de atracción en coordenadas $(0.8, -1)$.

En esta prueba se utilizó un punto en $(0.4, -0.5)$, como se observa en la Figura 44. A pesar de que todas las constantes sigieron igual, la distribución final fue considerablemente mejor que la de los dos casos anteriores. Para determinar qué tan buena es una distribución se analizó el posicionamiento de los agentes comparado al área de trabajo.

En este caso, tres agentes se mantuvieron en las afueras del círculo en la parte superior mientras cubren parte de la izquiera, el centro y la derecha. Luego, tres agentes se encontraban en la parte inferior cubriendo el centro y la derecha. Por último, los dos agentes del centro se encontraban suficientemente separados para cubrir el punto, el centro y la derecha. Por otro lado, tres agentes se mantuvieron en el área amarilla del punto, lo que es preferible sobre tener solo dos cubriendo el mismo punto, como se observa en las figuras 42 y 43.

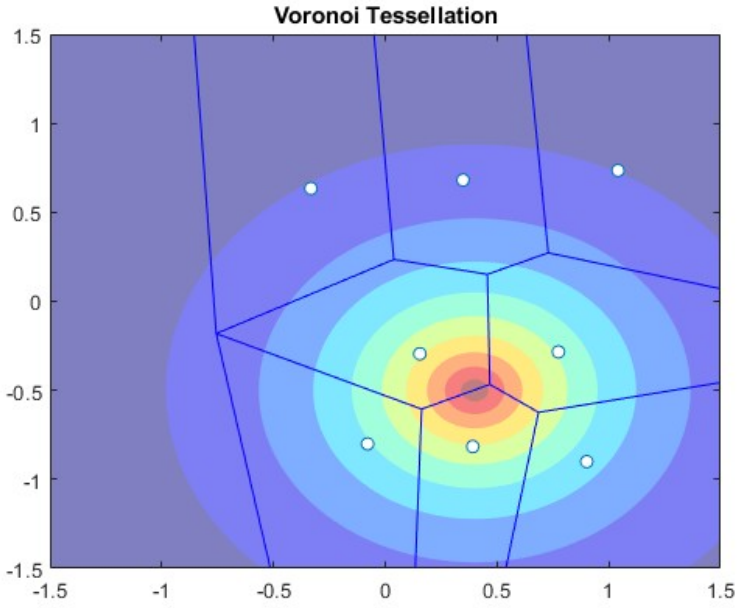



Figura 44: Diagrama de Voronoi para 8 agentes con punto de atracción en coordenadas $(0.4, -0.5)$.

El motivo por el que la formación fue mejor en el último caso es que la gran mayoría del radio de atracción del punto se encontraba dentro del área de trabajo. Esto causó que el algoritmo funcionase de forma más adecuada al no tener un área de atracción desbalanceada por la reducción asimétrica del radio. Asimismo, se consideró a una distribución con punto de concentración como satisfactoria bajo las siguientes condiciones:

- Los agentes mantuvieron la formación base mostrada en la Figura 33.
- Al menos dos agentes se encontraban en el área amarilla.
- Los agentes lograron cubrir un área considerable del área de trabajo.

Cabe destacar que el segundo punto aplica para los sistemas analizados que utilizan 8 agentes. Si se plantease un sistema con más agentes se debería de analizar distintos casos

para encontrar el número mínimo aceptable de agentes que se deberían posicionar en el área amarilla. De igual forma, es factible que dependiendo del posicionamiento del punto de concentranción y el número de agentes a utilizar se deberían de cambiar las constantes del radio y factor de atracción.

### 9.2.2. Análisis del algoritmo con puntos de concentración implementado en Robotat

Con el algoritmo validado se prosiguió a realizar pruebas dentro del ecosistema Robotat. Esto con el fin de comparar el comportamiento de los agentes respecto a la simulación y al algoritmo original sin los puntos. Es por esto que las formaciones iniciales fueron similares a las vistas en el capítulo 8, sección 8.2, así como que se mantuvieron los siguientes valores constantes:

- El número de agentes, en 8.
- El paso de tiempo, en 0.1.
- El factor de atracción, en 0.1.
- El radio de atracción, en 0.6.
- Las coordenadas del punto de concentración $(x, y)$, en -0.8 y 1 respectivamente.

El ciclo de control se mantuvo igual al implementado en el algoritmo original sin puntos de concentración, donde la base para el mismo fue un controlador PID con acercamiento exponencial. El algoritmo calculó los puntos objetivos considerando la influencia que el punto de concentración tendría sobre los agentes y su formación final. En cambio el ciclo de control se encargó de calcular las trayectorias y guiar a los agentes para que alcanzaran las posiciones objetivo ya propagadas e influídas por los puntos de concentración.

En la primera simulación se posicionaron los agentes en una formación alrededor del centro, como se muestra en la Figura 45 y similar a la vista en la Figura 13 de la sección 8.2.1. Con esto, se observa cómo los agentes primero buscaron propagarse para cubrir el área de trabajo como si se tratara de un sistema sin puntos de concentración. No obstante, una vez finalizada dicha propagación, el punto de concentración empezó a tener una mayor influencia sobre las posiciones de los agentes, que ya se encontraban en una formación estable, causando que se movilizaran en dirección a la concentración.

Esto se observa en las figuras 47 y 48, donde las trayectorias de los agentes primero se extendieron hacia afuera para luego dirigirse hacia el punto de concentración. Finalmente, la formación resultante del sistema muostró una gran similitud con la observada en las pruebas teóricas realizadas en la sección anterior. Esto indicó que el sistema estaba funcionando de acuerdo con lo esperado, generando formaciones de Voronoi que cubrían el área deseada mientras se agrupaban alrededor del punto de concentración.

En lo que respecta a las trayectorias realizadas por los agentes, se observa en la Figura 49 que las trayectorias teóricas y experimentales se encontraban bastante cercanas. La

diferencia principal fue que, en las experimentales, se observaron trayectorias más suaves y redondeadas, mientras que el sistema teórico siempre mostró vibraciones. Esto se debió a los mismos factores discutidos en la sección 8.2.1, un fenómeno esperado considerando que se utilizó el mismo controlador y simulador como base.

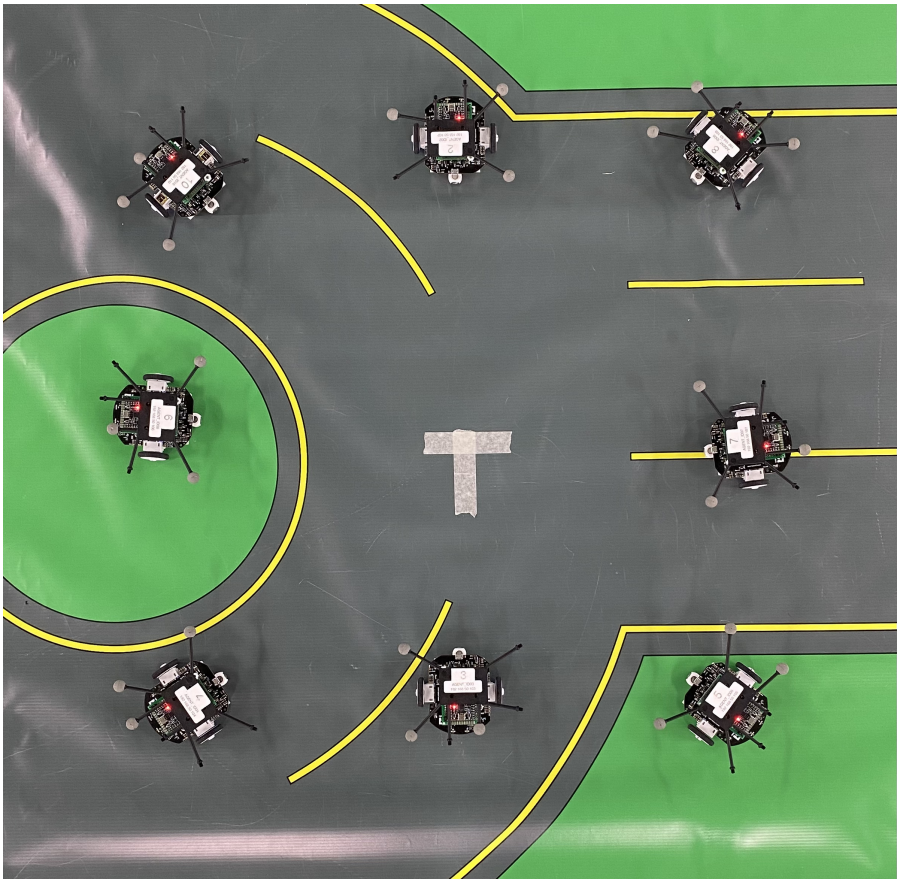


Figura 45: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat para primera formación central con punto de atracción.

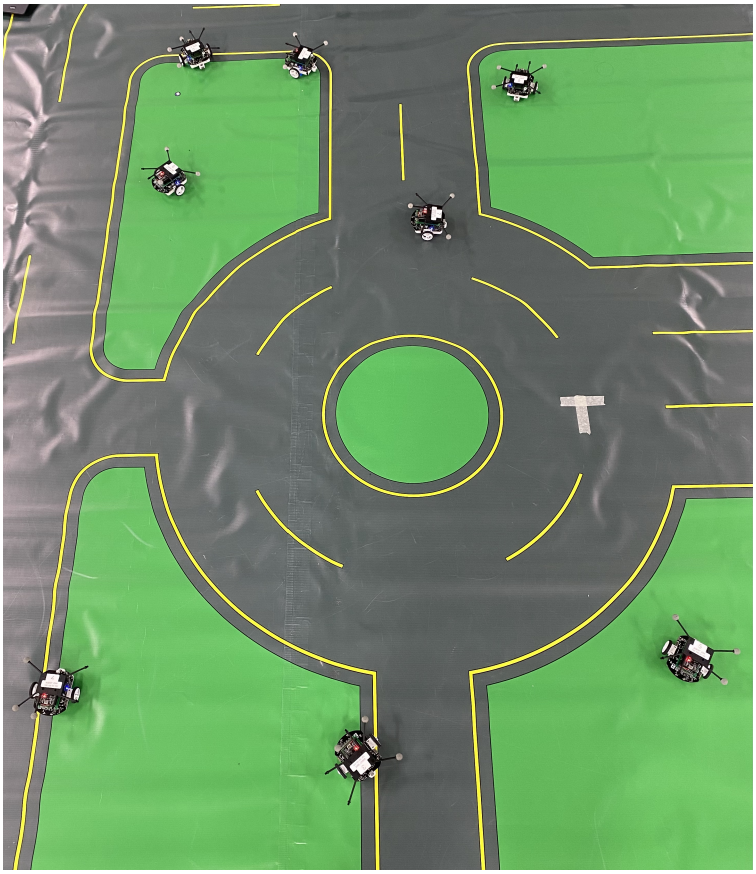


Figura 46: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat para primera formación central con punto de atracción.

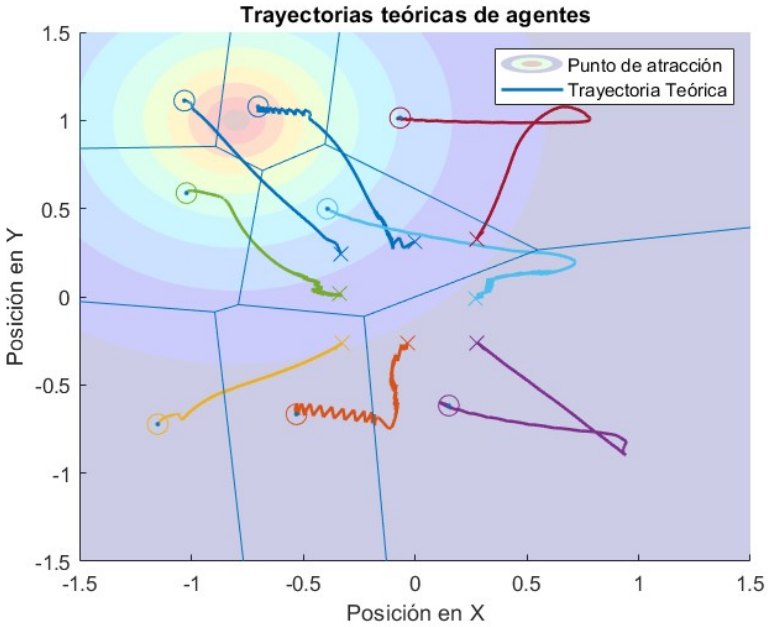



Figura 47: Trayectorias teóricas de agentes para primera formación central con punto de atracción.

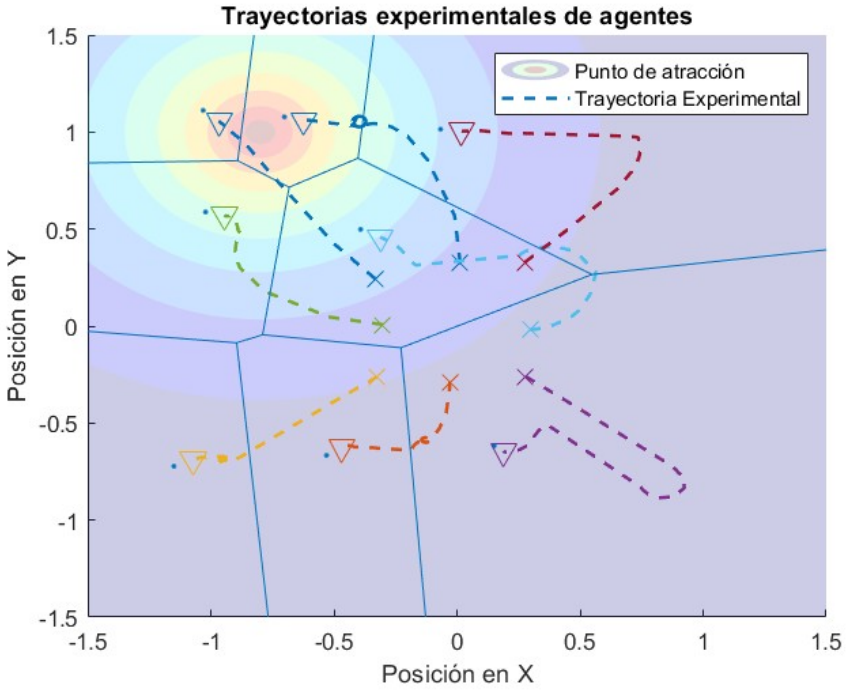



Figura 48: Trayectorias experimental de agentes para primera formación central con punto de atracción.

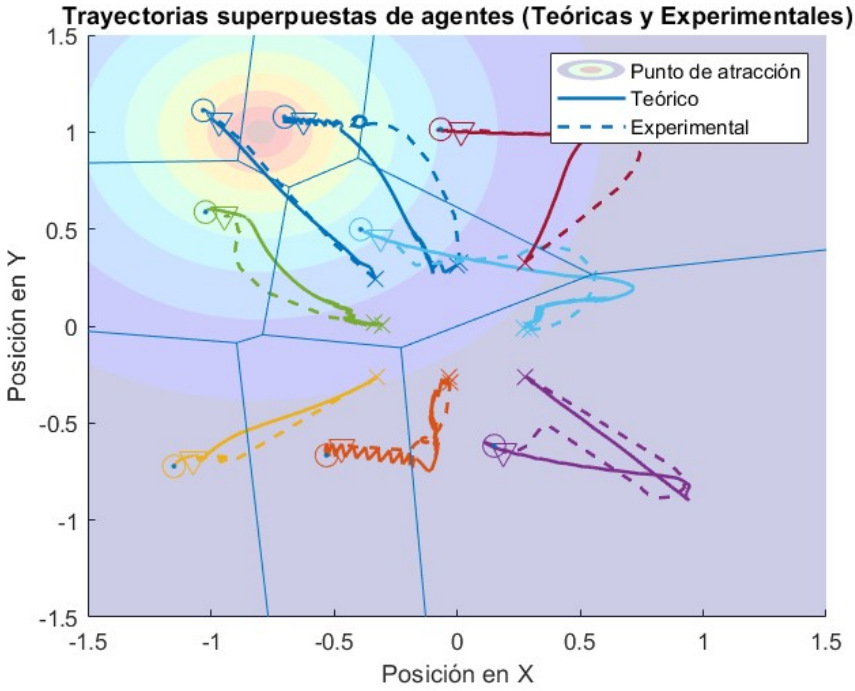



Figura 49: Comparación de trayectorias de agentes para primera formación central con punto de atracción.

Se realizó una segunda ejecución con las mismas posiciones iniciales para corroborar que el algoritmo convergiera a una formación similar a la primera, validando así la reproducibilidad de los resultados. Al observar las trayectorias resultantes, fue evidente que el sistema se comportó casi exactamente igual que en la prueba anterior. Las trayectorias teóricas fueron casi idénticas entre las figuras 47 y 52, mostrando ligeros cambios en las vibraciones de algunos agentes.

Sin embargo, las trayectorias experimentales mostraron un cambio más significativo, aunque siempre alcanzando la formación deseada. Esto se debió al controlador, ya que el error de posición esperado sigue siendo del 10 % al haber utilizado el mismo controlador. No obstante, esto no fue relevante al evaluar las posiciones finales, ya que estas resultaron exactamente iguales entre sí (ver figuras 48 y 53 como referencia), demostrando el correcto funcionamiento del algoritmo.

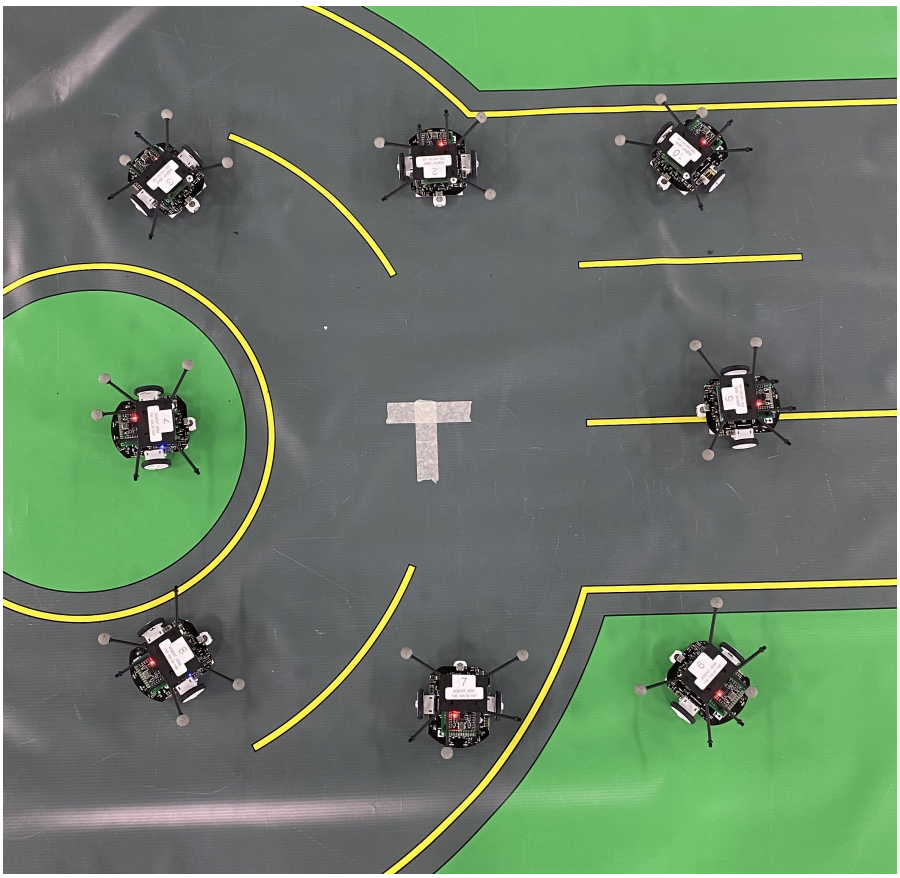


Figura 50: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación central con punto de atracción.

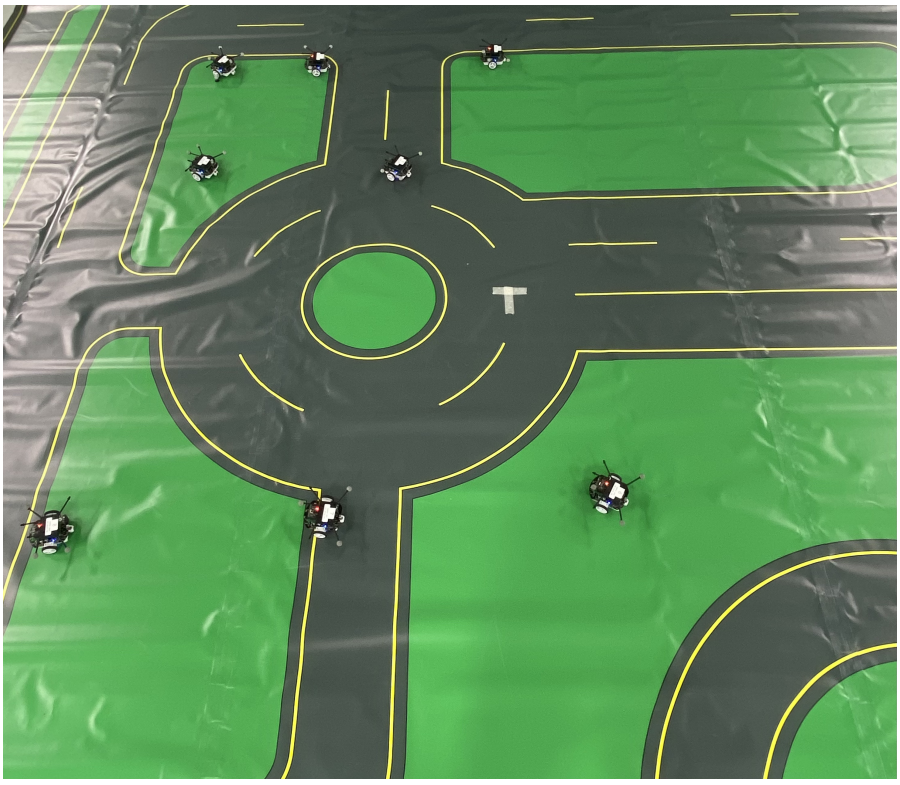


Figura 51: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat para segunda formación central con punto de atracción.

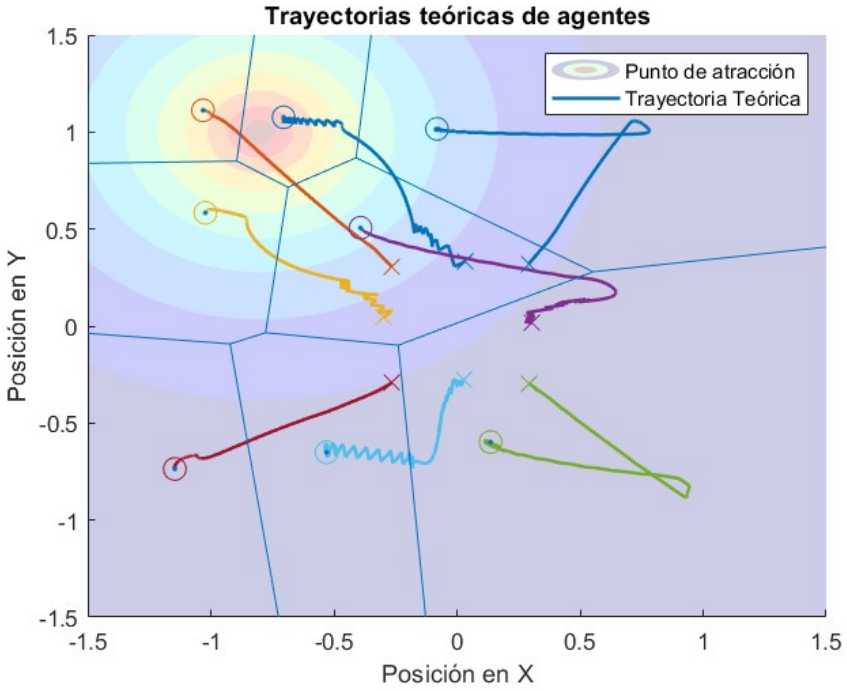



Figura 52: Trayectorias teóricas de agentes para segunda formación central con punto de atracción.

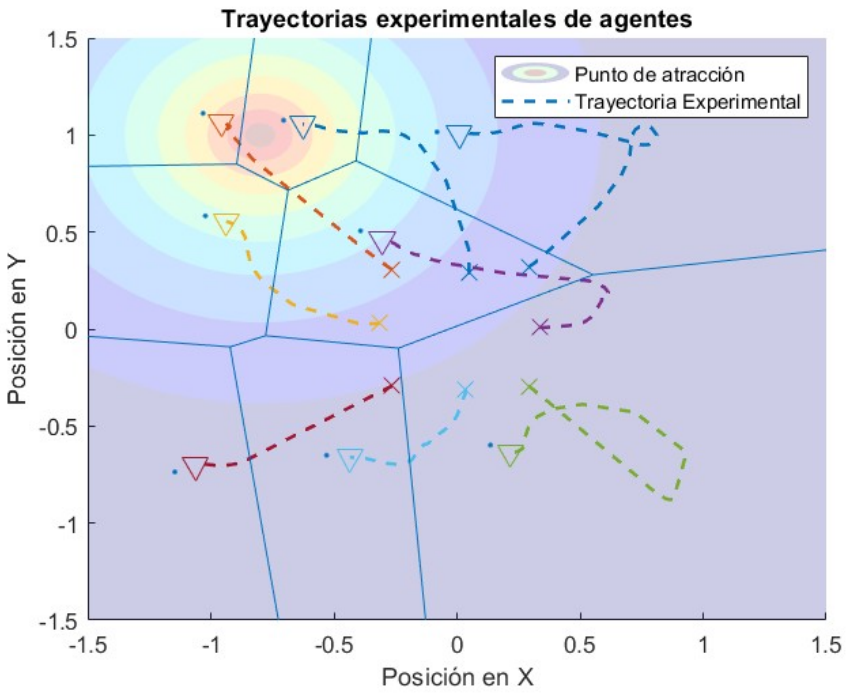



Figura 53: Trayectorias experimental de agentes para segunda formación central con punto de atracción.

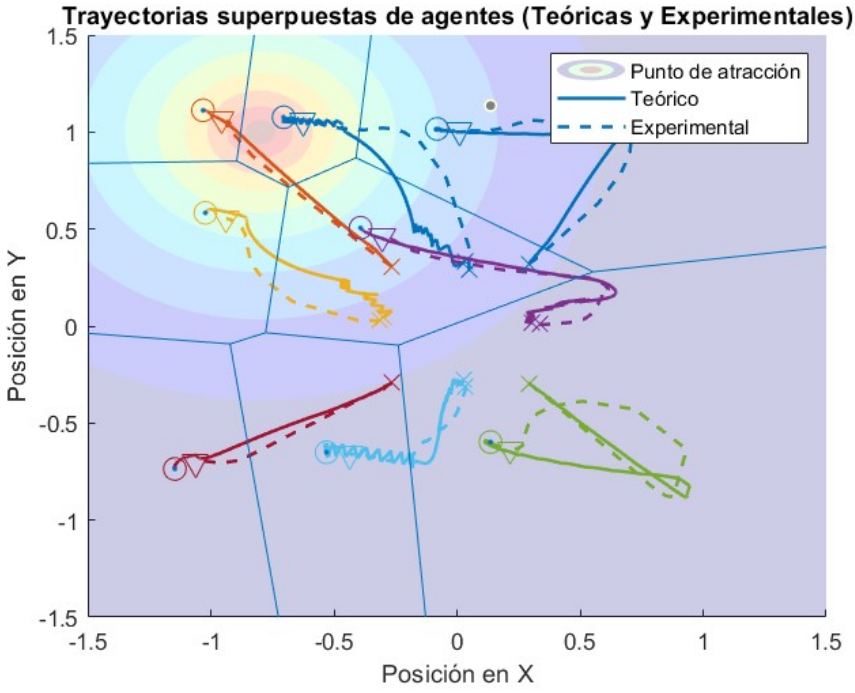



Figura 54: Comparación de trayectorias de agentes para segunda formación central con punto de atracción.

Una vez validado el funcionamiento del algoritmo con las posiciones iniciales en formación central, se procedió a plantear una posición inicial aleatoria con el fin de compararlo con el comportamiento observado sin el punto de atracción. Por esta razón, se optó por posicionar a los agentes lo más parecido posible a la configuración observada en la Figura 23, siendo la formación inicial utilizada la mostrada en la Figura 55.

Como se observa en la Figura 57, las trayectorias teóricas mostraron que el algoritmo convergió en la formación deseada sin mayores complicaciones. Cabe destacar que 2 de los 8 agentes experimentaron una fuerte vibración debido al punto de atracción. Esto se debió a que el algoritmo intentaba distribuir a los agentes para cubrir toda el área de trabajo, mientras el punto los atraía hacia su posición. Sin embargo, el sistema logró estabilizarse una vez que los agentes alcanzaron los objetivos distribuidos alrededor del punto de atracción, manteniendo la mayor cobertura posible del área de trabajo y siempre posicionados en una distribución de Voronoi aceptable, como se muestra en la Figura 57.

Por otro lado, la trayectoria experimental en este caso resultó mucho más errática y menos directa que para la formación central. Si se analizara como un evento aislado, podría parecer un resultado anómalo, sin embargo, al compararlo con la Figura 25 es evidente que este fenómeno es esperable. Al igual que en ese caso, los agentes siguieron trayectorias completamente erráticas, realizando giros que podrían considerarse innecesarios hasta llegar a las posiciones objetivo. No obstante, esto se debió al error de posición del 10 % que presenta el controlador el cual se acumulaba de forma significativa en trayectorias largas, lo que provocó que los agentes tomaran caminos diferentes a los teóricos.

Asimismo, se observa en la Figura 59 que, al igual que en el primer caso de formación aleatoria sin punto de concentración, los agentes rotaron y cambiaron de posición final respecto a la distribución teórica. Esto no significó que el algoritmo no funcionara, ya que, teniendo como objetivo el control de cobertura la trayectoria de los agentes no se consideró relevante siempre y cuando se cumpliera con el objetivo principal de cubrir el área de trabajo de la mejor forma posible.

Por último, con esto se validó que el algoritmo de control de cobertura con puntos de atracción funcionó de manera correcta y aceptable, aunque siempre fue necesario calibrar los parámetros del radio y la constante de atracción para obtener los mejores resultados. Esto implicó que, dependiendo de estos parámetros, los resultados podrían variar, y podría ser posible que existan combinaciones óptimas que mejoren las pruebas realizadas.

Figura 55: Posiciones iniciales de 8 agentes Pololu 3Pi+ en el Robotat para primera formación aleatoria con punto de atracción.

Figura 56: Posiciones finales de 8 agentes Pololu 3Pi+ en el Robotat para primera formación aleatoria con punto de atracción.

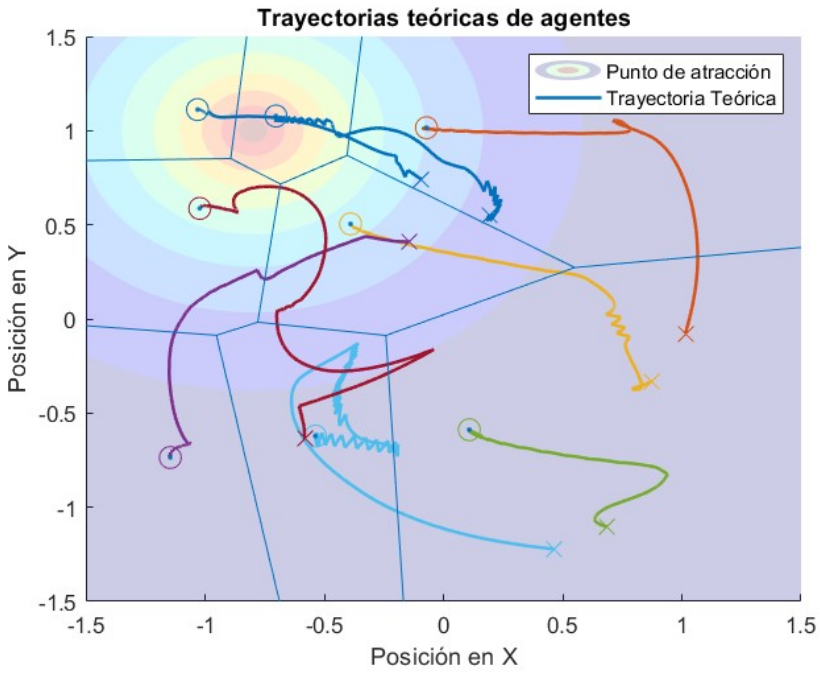



Figura 57: Trayectorias teóricas de agentes para primera formación aleatoria con punto de atracción.

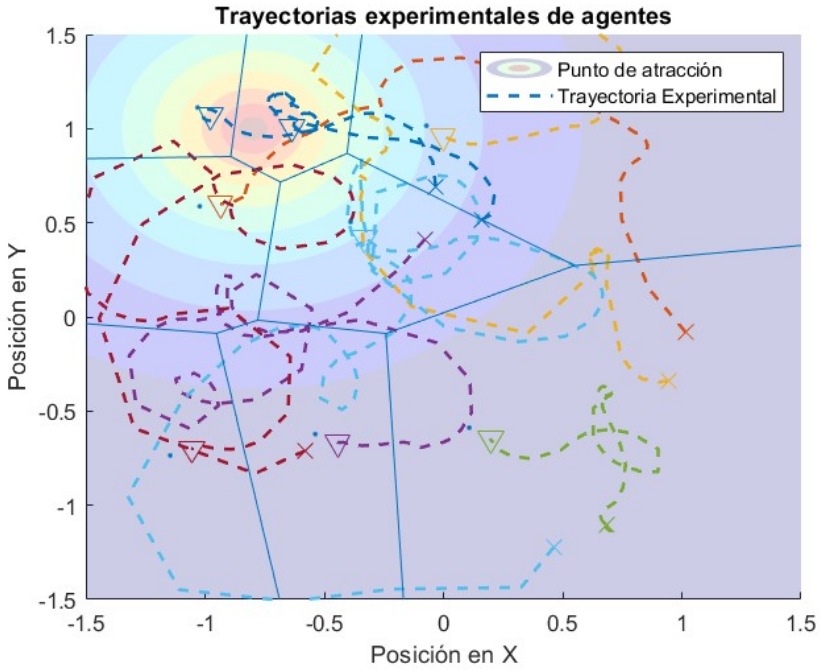



Figura 58: Trayectorias experimental de agentes para primera formación aleatoria con punto de atracción.

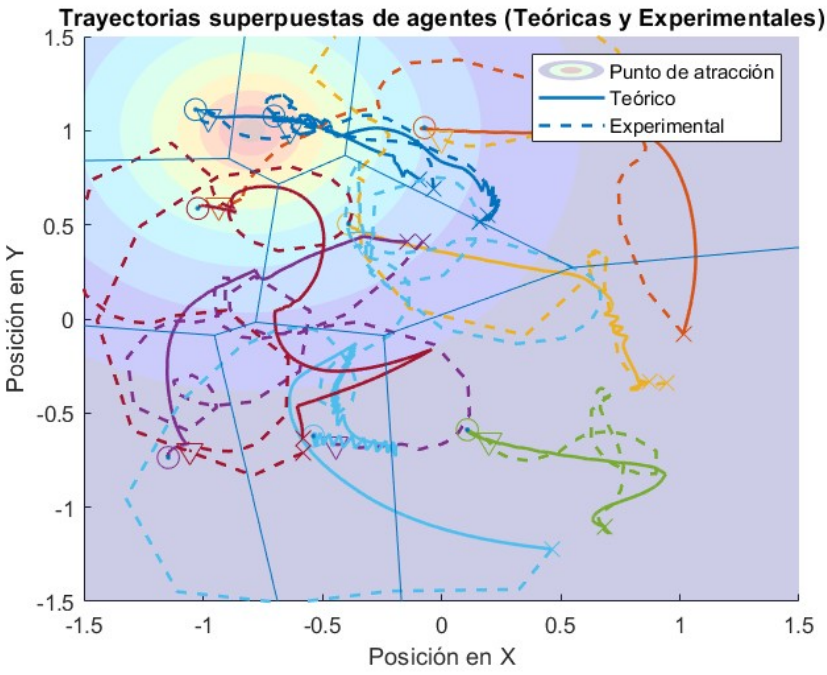



Figura 59: Comparación de trayectorias de agentes para primera formación aleatoria con punto de atracción.

## 9.3. Simulación del algoritmo con puntos de concentración proyectado sobre el lago de Atitlán

Con el algoritmo ya implementado y validado, se decidió diseñar y probar en Matlab una simulación del lago de Atitlán con áreas de contaminación por cianobacteria. Para esto, primero se buscó una imagen que mostrara el lago completo junto con las áreas mencionadas. Se encontró una imagen del estado del lago en 2015 en el artículo [13], mostrada en la Figura 60. En esta se observó cómo existen áreas de color verde musgo que representan las zonas contaminadas del lago.

Con el fin de establecer una escala relativamente realista, se recortó la imagen para que solo apareciera la mayor área posible del lago. Luego, se utilizó un eje $x$ de 12000 y un eje $y$ de 8500, de modo que se representaran las dimensiones del lago en metros, que son de 8 km por 12 km. El sistema ya delimitado dentro de esta área se puede observar en la Figura 61.

Figura 60: Fotografía del lago de Atitlán extraída de [13].

Con este sistema ya a escala, se comenzó con el planteamiento de las áreas de contaminación. Estas se representaron mediante puntos de concentración, cuyas coordenadas se extrajeron directamente de la imagen. Se identificaron las mayores concentraciones de cianobacteria para ubicar en estas posiciones los puntos mencionados, colocando un total de 8 puntos.

Asimismo, se seleccionaron radios de influencia que se ajustaran lo más posible a las áreas mostradas en la imagen, los cuales luego se multiplicaron por un factor de 1000 para mantener la escala con la gráfica utilizada. Las coordenadas y radios utilizados se declararon en los vectores $radio\_influencia$, $X\_concen$, y $Y\_concen$ para el radio y las coordenadas $(x, y)$, respectivamente, siendo los siguientes:

1. Radio de 400, coordenadas (3722, 4257).
2. Radio de 300, coordenadas (4398, 3669).
3. Radio de 400, coordenadas (5750, 3883).
4. Radio de 300, coordenadas (7628, 3722).
5. Radio de 400, coordenadas (9506, 4150).
6. Radio de 400, coordenadas (10483, 4953).
7. Radio de 300, coordenadas (8529, 6505).
8. Radio de 400, coordenadas (7403, 7470).

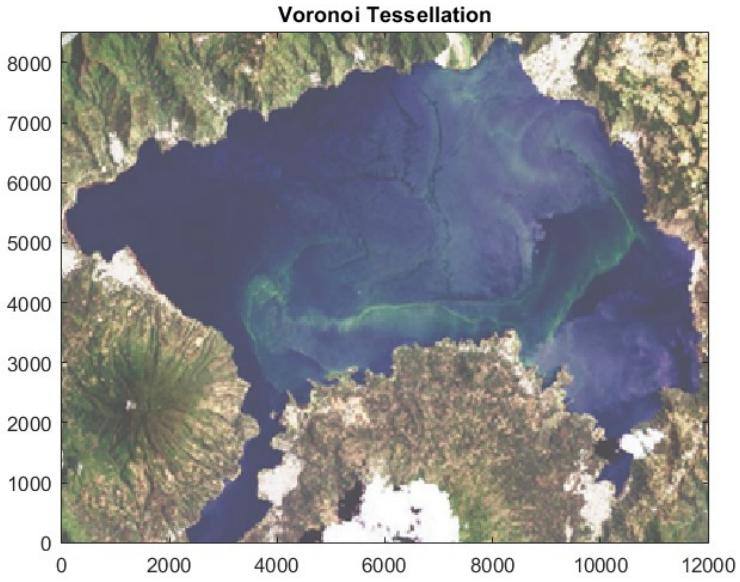



Figura 61: Área de trabajo representada en el lago de Atitlán.

Debido a la naturaleza del simulador desarrollado, el área de trabajo solo se planteó en forma rectangular. Esto podría modificarse en el futuro para adaptarse a una forma poligonal que se asemeje más a la forma del lago. No obstante, para esta prueba no se consideró necesario ya que, como se observa en la tabla anterior, las coordenadas se encontraban dentro de un área rectangular que apenas se superponía con áreas de tierra. Por esta razón, se decidió definir un área de trabajo con los siguientes parámetros:

- Valor mínimo en el eje $x$: 3346.
- Valor máximo en el eje $x$: 10783.
- Valor mínimo en el eje $y$: 2812.
- Valor máximo en el eje $y$: 8112.

Con el área de trabajo, las coordenadas y los radios de los puntos de concentración ya definidos, se procedió a plantear el simulador como tal. Debido a los radios establecidos y al tamaño del área de trabajo, se utilizó un factor de atracción de 0.18, siendo 0.08 mayor al empleado en las pruebas de la sección anterior. Se mantuvo un paso de tiempo $dt$ igual a 0.1, ya que este valor demostró ser el más estable permitiendo que el sistema convergiera relativamente rápido sin provocar vibraciones extremas. Luego, se seleccionó 20 como el número de agentes a simular, buscando un equilibrio en el ecosistema.

Si se colocaran muy pocos agentes, por ejemplo, entre 8 y 9, la cantidad de puntos sería casi igual a la cantidad de agentes lo que haría que cada agente se ubicara en un punto de concentración sin ser suficientes para distribuirse y cubrir todo el área de trabajo. Por otro lado, si se utilizaran 40 o más agentes, estos empezarían a actuar más como una forma

de contaminación y estorbo para el ecosistema, en lugar de ayudar en la medición de la contaminación.

En cuanto al posicionamiento inicial de los agentes, se asumió que estos se liberarían en posiciones semi-aleatorias. Esto significó que serían colocados únicamente en coordenadas dentro del área de trabajo, pudiendo ocupar cualquier punto dentro de esta. Por esta razón, se utilizó la función randi($randi\_val$), donde el valor de $randi\_val$ era igual a 5000, lo que significó que los valores generados estarían entre 0 y 5000. Para ajustar las coordenadas a valores dentro del área de trabajo, se sumó el valor mínimo del respectivo eje a la coordenada generada, siendo 3346 para el eje $x$ y 2812 para el $y$.

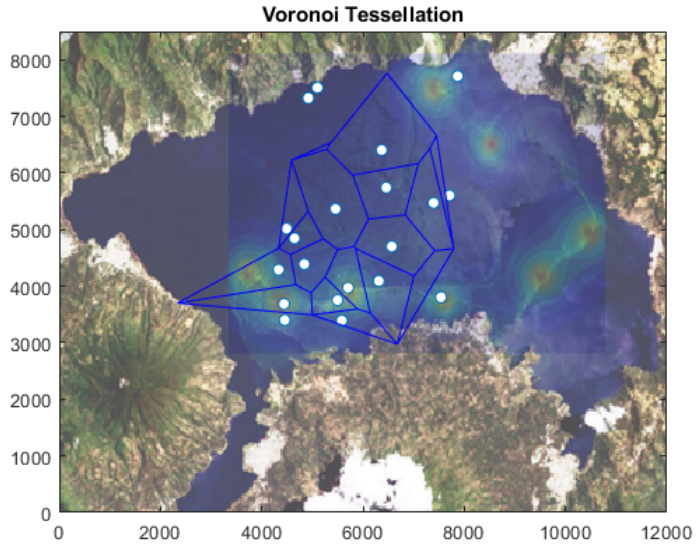



Figura 62: Sistema con posiciones iniciales aleatorias previo a propagación sobre el lago de Atitlán.

Una vez delimitados los parámetros iniciales, generadas las coordenadas iniciales para los 20 agentes y posicionados los puntos de concentración, se procedió a poner el algoritmo a prueba. Al inicio, los agentes aparecieron en diferentes puntos alrededor del lago, pero conforme el algoritmo se propagó, se observó cómo los agentes comenzaron a movilizarse hacia las áreas con cianobacteria. Luego de dejar correr el programa por un tiempo, se obtuvo un resultado satisfactorio: entre 1 y 3 agentes se posicionaron alrededor de cada área de contaminación, mientras que 3 agentes se mantuvieron en el centro como se observa en la Figura 63. Esto con el fin de poder cubrir todo el área de trabajo mientras se enfocaban en las áreas contaminadas.

Esto último fue de vital importancia si se analizara un sistema dinámico, como lo es la contaminación en el lago de Atitlán. Esta suele moverse con el tiempo por una gran variedad de factores (como los vientos, corrientes, cambios de temperatura, entre otros factores). Entonces si esta se moviese hacia un área central, por ejemplo, los agentes del centro lo detectarían y se posicionarían alrededor de esta. Consecuentemente, los agentes cuyas áreas se movieron o desaparecieron pasarían a cubrir el nuevo área libre de contaminación. De esta forma se podría asegurar que todo el área de trabajo se mantendrá observada sin problema mientras que las partes contaminadas siempre tendrán mínimo un agente en su vecinidad.

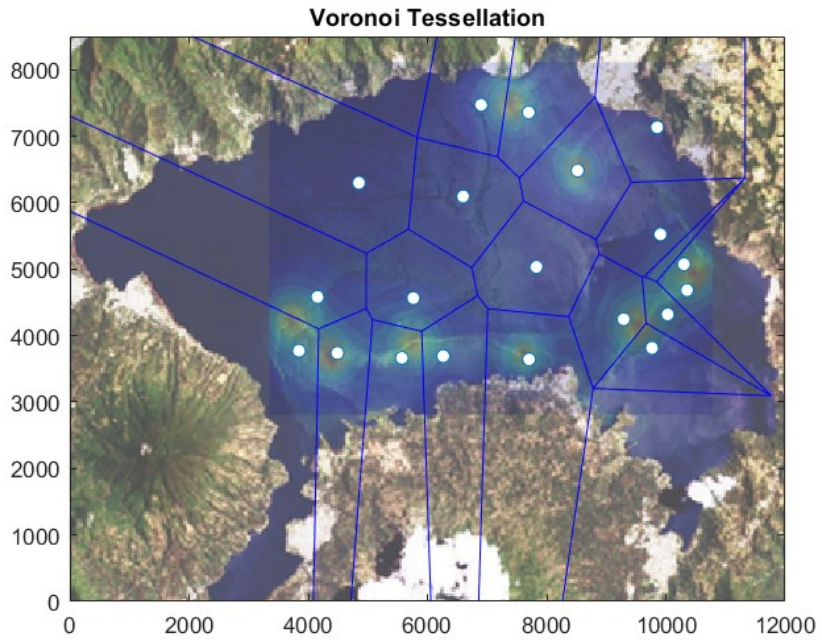



Figura 63: Sistema propagado de forma satisfactoria sobre el lago de Atitlán.

La simulación del algoritmo sobre un modelo del lago de Atitlán permitió validar la funcionalidad del sistema en un entorno representativo, demostrando su capacidad para distribuir agentes de manera eficiente en áreas de alta concentración de puntos de interés, como zonas contaminadas. A pesar de las simplificaciones realizadas, como el uso de un área rectangular en lugar de una forma poligonal, los resultados obtenidos reflejan un comportamiento adaptativo y prometedor. Este enfoque no solo ofrece una solución viable para problemas de cobertura estática, sino que también sugiere su aplicabilidad en sistemas dinámicos, donde la adaptabilidad de los agentes podría ser clave para el monitoreo continuo de fenómenos cambiantes.

# CAPÍTULO 10

# Conclusiones

- El algoritmo de control de cobertura multi-agente basado en diagramas de Voronoi demostró ser el más adecuado para esta aplicación debido a su robustez, alta velocidad de convergencia y amplia disponibilidad de herramientas para su implementación, como lo fueron las funciones 17, 18 y 19.
- El entorno de simulación desarrollado resultó representativo y adecuado para la implementación y evaluación del algoritmo basado en diagramas de Voronoi, demostrado al ser capaz de simular de forma satisfactoria a partir de 8 agentes una gran variedad de casos, donde estos pueden cambiarse al variar el número de agentes o el área de trabajo. Es por esto que se consideraron estas dos juntos con el paso del tiempo como los parámetros más relevantes. Esta última por su influencia directa en la convergencia y estabilidad del sistema.
- La implementación del algoritmo en el ecosistema Robotat fue exitosa, logrando un margen de error de posición final máximo del 10 %. Esto se obtuvo mediante la inclusión de un ciclo de control basado en un controlador PID con acercamiento exponencial, lo cual permitió un direccionamiento preciso y estable de los agentes Pololu 3PI+.
- La implementación de puntos de concentración fue exitosa, permitiendo que tanto el simulador como la implementación en el ecosistema Robotat operaran de manera correcta en sistemas dinámicos, demostrando la capacidad del algoritmo para modelar puntos de contaminación en cuerpos de agua. Además, el simulador del lago de Atitlán resultó adecuado para la implementación de boyas inteligentes que operan bajo el modelo de uniciclo, utilizado también para modelar el comportamiento de los robots móviles Pololu 3PI+.

# CAPÍTULO 11

## Recomendaciones

- Explorar métodos para definir el área de trabajo como un polígono en lugar de un rectángulo, permitiendo una mayor adaptabilidad a sistemas con perímetros más complejos y con formas irregulares.
- Evaluar formas de distribuir el algoritmo entre los agentes para no necesitar un maestro omnisciente con conocimiento completo de la posición de los agentes para la propagación del algortimo. Esto permitiría al algoritmo funcionar de forma autónoma al no necesitar una central, logrando obtener los datos completos con conectarse a algún nodo de la red.
- Realizar pruebas adicionales para identificar una posible relación cuantitativa o modelo de regresión entre el radio de influencia y el factor de atracción. Esto permitiría ajustar estos parámetros de manera precisa sin necesidad de pruebas de calibración extensivas, resultando en la mejora del simulador con puntos de concentración.
- Llevar a cabo pruebas adicionales para optimizar los parámetros del controlador PID utilizado en el ciclo de control, con el objetivo de que los agentes sigan trayectorias más directas y minimicen las curvas amplias.
- Desarrollar e integrar un sistema para evitar colisiones entre los agentes durante el ciclo de control.
- Realizar una evaluación de qué tan factible es la implementación en el lago. Para esto se deben de analizar las plataformas robóticas y protocolos de comunicación que se puedan acoplar a las condiciones del mismo.

# CAPÍTULO 12

# Bibliografía


[1] A. Maybell, "Algoritmo de sincronización y control de sistemas de robots multi-agente para misiones de búsqueda," Tesis de Licenciatura, Universidad del Valle de Guatemala, 2019.

[2] C. Perafán, "Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica," Tesis de Licenciatura, Universidad del Valle de Guatemala, 2022.

[3] S. Tubaishat M. y Madria, "Sensor networks: an overview," *IEEE Potentials*, vol. 22, n.º 2, págs. 20-23, 2003. DOI: 10.1109/MP.2003.1197877.

[4] X. y. o. Xu G.; Wang, "Applications of Wireless Sensor Networks in Marine Environment Monitoring: A Survey," *Sensors*, vol. 14, n.º 9, págs. 16 932-16 954, 2014. DOI: 10.3390/s140916932.

[5] "La contaminación del Lago Atitlán, Guatemala, amenaza la subsistencia y la salud de los habitantes." (2016), dirección: https://globalpressjournal.com/americas/guatemala/contamination-of-guatemalas-lake-atitlan-threatens-livelihoods-health-of-residents/es/.

[6] E. y. o. Akyildiz I.F.; Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, n.º 8, págs. 102-114, 2002. DOI: 10.1109/MCOM.2002.1024422.

[7] B. Wang, *Coverage Control in Sensor Networks*. 2010.

[8] "MULTI-AGENT COVERAGE CONTROL." (), dirección: http://muro.ucsd.edu/CoverageControl/coveragecontrol.html.

[9] M. Mesbahi y M. Egerstedt, "Graph Theoretic Methods in Multiagent Networks," en *Princeton Series in Applied Mathematics*, 2010. dirección: https://api.semanticscholar.org/CorpusID:35274095.

[10] "Introduction to Voronoi Diagrams." (1993).

[11] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, vol. 2, n.º 1, págs. 153-174, 1987. DOI: 10.1007/BF01840357.

[12] M. Zea, "MT3005 Robótica 1," 2024.

[13] “Reaparece la cianobacteria que afecta al Lago de Atitlán.” (2015), dirección: https://www.soy502.com/articulo/nuevo-florecimiento-cianobacterias-afecta-al-lago-atitlan.

# CAPÍTULO 13

# Anexos

## 13.1. Repositorio

Enlace al repositorio con los entornos de simulación desarrollados y la implementación: https://github.com/Christian-20868/Algoritmo-de-control-de-cobertura