Desarrollo de un sistema de localización de inventario en un almacén a escala, utilizando la estrategia de agrupamiento de un enjambre de robots

Ana Valentina Montañez Cuellar

Trabajo de Grado para Optar al Título de Ingeniero Mecánico

Director

Yennifer Yuliana Ríos Díaz

PhD. Ciencias con especialidad en Ingeniería Eléctrica

Codirector

Jeyson Arley Castillo Bohórquez

Ingeniero Electrónico

Universidad Industrial de Santander

Facultad de Ingenierías Físico-mecánicas

Escuela de Ingeniería Mecánica

Ingeniería Mecánica

Bucaramanga

2025

Dedicatoria

A mi mamita, Edilma, por su apoyo y esfuerzo constante. Este logro no habría sido posible sin su fortaleza y dedicación, que siempre me han impulsado a alcanzar mis metas.

A Maito, por darme la oportunidad de ser su hija y ser un pilar para mí. Su amor y enseñanzas han dejado una huella en mi vida. Sé que siempre seré tu orgullo, tu intelectual, y eso me llena de felicidad.

A mi hermana, Melissa, por ser no solo mi familia, sino también mi amiga incondicional. Gracias por estar siempre a mi lado, brindándome tu apoyo en cada paso del camino.

A Juan José, por ser mi apoyo constante y mi luz. Por brindarme su cariño y creer en mí, incluso cuando yo dudaba de mí misma, recordándome que soy capaz.

A cada uno de los mencionados, los amo y les agradezco profundamente por ser parte de mi vida y de este logro.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mis seres queridos: a mi mamá, a Maito, a mi padre, a mi hermana, a Juan José y a mi familia. Ustedes han sido mi fortaleza y mi inspiración. Igualmente, no puedo olvidar a mis fieles compañeros de cuatro patas: Lily, Cece, Winnie Pooh y mi gata Jojo. Gracias por llenar mis días de alegría y cada noche en que estuvieron a mi lado. Su compañía ha sido invaluable.

A mi directora de tesis, la profesora Yennifer Ríos, le agradezco profundamente por su guía y dedicación durante este proceso. Su apoyo fue fundamental para llevar a cabo este proyecto.

Expresar mi agradecimiento a los amigos que hice durante mi carrera, gracias por cada sonrisa, por las experiencias compartidas y por ser parte de este camino, en especial a Majito.

Finalmente, quiero agradecerle a mi tuna, Tunarte UIS, por ser mi segunda familia y regalarme ese granito de felicidad cada semana. Su compañía, sus risas y su música han sido una parte esencial de mi vida universitaria, dejando recuerdos que siempre llevaré en mi corazón.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

pág.

Resumen

Título: Desarrollo de un sistema de localización de inventario en un almacén a escala, utilizando la estrategia de agrupamiento de un enjambre de robots [*]

Autor: Ana Valentina Montañez Cuellar[**]

Palabras Clave: Robótica de enjambre, Beeclust, Algoritmo de agrupamiento, localización inventario perdido, inteligencia de enjambre

Descripción: El presente proyecto desarrolla un sistema para la localización de inventarios en un entorno controlado mediante robótica de enjambre, inspirado en el algoritmo *Beeclust*. Se caracterizó un área de trabajo configurada como un laberinto, con una fuente de luz como estímulo principal y robots Formula AllCode. Este entorno permitió implementar un algoritmo de agrupamiento adaptado para coordinar robots en escenarios experimentales.

El algoritmo fue validado en pruebas controladas que analizaron variables como intensidad lumínica, posición y tamaño de la fuente de luz, así como tamaño del área de trabajo. Los resultados mostraron que las fuentes más grandes generaron tiempos de agrupamiento más rápidos, pero con agrupaciones dispersas, mientras que las fuentes más pequeñas favorecieron agrupaciones más compactas, aunque con tiempos más prolongados. En cuanto al tamaño del área, configuraciones del 50 % presentaron tiempos promedio de 47.87 segundos con menor variabilidad, mientras que áreas muy grandes o pequeñas introdujeron más dispersión. La detección de robots alcanzó una precisión del 96.67 %, demostrando una alta confiabilidad en la colaboración entre ellos, aunque con oportunidades de mejora en la detección de obstáculos.

El prototipo inicial fue validado con dos robots y demostró escalabilidad para enjambres mayores. Este sistema ofrece un enfoque basado en inteligencia de enjambre que representa un punto de partida prometedor para la localización de inventarios en entornos logísticos controlados, con posibilidades de integración de tecnologías complementarias y adaptaciones para aplicaciones más complejas

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería Mecánica. Programa académico. Director: Yennifer Yuliana Ríos Díaz. PhD. Ciencias con especialidad en Ingeniería Eléctrica. Codirector: Jeyson Arley Castillo Bohórquez. Ingeniero Electrónico

Abstract

Title: Development of an Inventory Localization System in a Scaled Warehouse Using a Robot Swarm Clustering Strategy[*]

Author(s): Ana Valentina Montañez Cuellar[**]

Key Words: Swarm robotics, Beeclust, Clustering algorithm, Lost inventory localization, Swarm intelligence

Description: This project develops a system for inventory localization in a controlled environment using swarm robotics inspired by the Beeclust algorithm. A work area configured as a maze was characterized, featuring a light source as the primary stimulus and Formula AllCode robots. This environment allowed the implementation of an adapted clustering algorithm to coordinate robots in experimental scenarios.

The algorithm was validated through controlled tests that analyzed variables such as light intensity, position and size of the light source, and the size of the work area. Results showed that larger light sources generated faster clustering times but resulted in dispersed groupings, whereas smaller light sources favored more compact clusters with longer times. Regarding the work area size, 50% configurations achieved average times of 47.87 seconds with less variability, while very large or small areas introduced greater dispersion. Robot detection reached a precision rate of 96.67%, demonstrating high reliability in collaboration, though with opportunities for improvement in obstacle detection.

The initial prototype was validated with two robots and demonstrated scalability for larger swarms. This system offers a swarm intelligence-based approach that represents a promising starting point for inventory localization in controlled logistics environments, with potential for integration of complementary technologies and adaptations for more complex applications.

[*] Degree Work

[**]Faculty of Physical-Mechanical Engineering. School of Mechanical Engineering. Academic Program. Director: Yennifer Yuliana Ríos Díaz, PhD in Sciences with a specialization in Electrical Engineering. Co-Director: Jeyson Arley Castillo Bohórquez, Electronic Engineer

Introducción

En una organización, el inventario es esencial para un correcto funcionamiento; sin embargo, la gestión adecuada del mismo representa un proceso que requiere una cantidad considerable de tiempo y recursos. Un almacén tradicional de 10.000 m2, requiere aproximadamente 30 operarios, numerosas máquinas y un día entero de actividad (Generix Group, 2022). Por su parte, una gestión no adecuada de inventario genera una serie de complicaciones, que contemplan retrasos en los pedidos, entregas incompletas y un mal funcionamiento de las operaciones logísticas, lo que finalmente se traduce en pérdidas económicas para la empresa. En este contexto, la pérdida de inventario, que puede darse por daño, robos, extravíos, un inapropiado sistema de ordenamiento del producto y una ineficiente regulación del flujo de pedidos, puede significar un problema de la cadena de suministro. Por tal motivo, es fundamental una identificación oportuna y precisa de la ubicación del inventario perdido para garantizar un proceso confiable y rentable.

Actualmente, existen diferentes técnicas para la gestión de inventarios aplicables en el entorno empresarial, como los sistemas de cintas transportadoras o el uso de escáneres de códigos de barras; no obstante, estos mecanismos requieren una reestructuración de la infraestructura de los almacenes o pueden estar sujetos a errores humanos. Es así que, como alternativa, se han desarrollado e implementado nuevas tecnologías como la RFID (Identificación por Radiofrecuencia) que, aunque mejora el sistema de inventario, es costosa, y estudios han demostrado que su precisión disminuye cuando entra el factor de participación humana. Dentro de las soluciones que se han planteado en la actualidad, hay tres destacadas para la automatización de inventarios: estantes inteligentes, antenas aéreas y robots de inventario. Diversos artículos han

dado estudio a este último aspecto, los robots de inventario basados en RFID, los cuales ofrecen soluciones al problema, pero presentan limitaciones de tamaño, falta de autonomía, complejidad y tiempo de trabajo (Casamayor-Pujol et al., 2020).

Frente a esto, en los últimos años se ha llevado a cabo una investigación en el desarrollo de tecnologías innovadoras que permiten a los robots trabajar de manera cooperativa en sistemas de redes de robots, emulando la inteligencia de enjambre que se observa en diferentes especies vivas; como las abejas, hormigas, peces o aves. Estos sistemas ofrecen ventajas frente a un único robot debido a que permiten la subdivisión de tareas, el despliegue de múltiples robots con una estructura menos compleja y cuya eficacia en grupo aumenta respecto a su capacidad individual. Además, en esta aplicación específica, la capacidad de trabajar simultáneamente en diferentes puntos dentro del mismo espacio de trabajo facilita una distribución del área local precisa y reduce significativamente el tiempo de finalización de las tareas (Brambilla et al., 2013).

Los sistemas de robótica de enjambre buscan imitar comportamientos observados en diferentes grupos de animales donde, dependiendo de la tarea o el objetivo, el método de agrupamiento varía. Entre estos comportamientos a los que se ha dado estudio, se encuentra la agregación, método observado en especies como las abejas o las cucarachas, que ocurre cuando un conjunto de robots autónomos se reúne en un mismo lugar. Esta es una técnica aplicable especialmente en la búsqueda de un objetivo común. Para abordar este fenómeno, se han desarrollado diversos algoritmos de agregación, clasificados principalmente en dos categorías: la agregación libre y la agregación mediada por el entorno. En esta última, el comportamiento dependerá de la influencia del entorno (Bayindir, 2016).

Este proyecto se enfoca en el desarrollo de una estrategia de agrupamiento de enjambre robótico para localizar inventario. Se abordan aspectos clave como la caracterización de los robots

y del entorno de trabajo, la implementación de algoritmos para navegación y detección, y el desarrollo de un algoritmo de agrupamiento que permita a los robots encontrar y agruparse en un objetivo. El sistema permite a los robots detectar otros robots en el entorno, empleando sensores que simulen el comportamiento de abejas en busca de un punto caliente. Asimismo, se lleva a cabo una validación experimental mediante pruebas en un entorno controlado para analizar el rendimiento y la precisión del sistema.

El trabajo está estructurado en varias secciones, inicialmente se abarca una explicación teórica de los conceptos fundamentales para el entendimiento del marco trabajado, seguida de una caracterización del robot y del entorno de trabajo (laberinto). Luego se presenta el desarrollo del sistema de control y el algoritmo de navegación para los robots, inicialmente en un solo robot, así como el algoritmo de agregación, en el que la detección de otros robots es de suma importancia. Posteriormente, se procede a una validación mediante pruebas y análisis de resultados y, finalmente, se presentan las conclusiones y recomendaciones del proyecto.

De este modo, se espera que este trabajo contribuya al mejoramiento de sistemas, aprovechando la cooperación entre robots en un entorno controlado al desarrollar un sistema de localización de inventario en un almacén a escala, utilizando la estrategia de agrupamiento de un enjambre de robots.

1. Objetivos

1.1 Objetivo General

Desarrollar un sistema de localización de inventario en un almacén a escala, utilizando la estrategia de agrupamiento de un enjambre de robots.

1.2 Objetivos Específicos

- Caracterizar el área de trabajo (laberinto) para el agrupamiento de robots, teniendo en cuenta el tamaño del robot, zona de agrupación y número de dispositivos estimado que forman el enjambre.
- Diseñar un algoritmo de agrupamiento basado en el algoritmo *BEECLUST* que permita la localización del producto, representado por una fuente de luz y ubicado en el área de trabajo.
- Validar el sistema de localización de inventario, utilizando un protocolo de pruebas de comportamiento del enjambre.

2. Marco Teórico

Este capítulo aborda los conceptos clave para el desarrollo del presente trabajo de investigación. En primer lugar, se presentan los antecedentes y diversos trabajos relacionados que establecen un precedente en la línea de robótica de enjambre. Posteriormente, se introduce una base conceptual mediante nociones y definiciones sobre robótica de enjambre, destacando sus características, métodos de comportamientos colectivos y su aplicación en sistemas robóticos.

Asimismo, se enfatiza en el algoritmo *BEECLUST* y en la caracterización de los robots utilizados en el proyecto, el *Formula AllCode Robot Buggy*, proporcionando una visión general de sus componentes y de la interfaz de trabajo. Finalmente, se conceptualiza el control PID, elemento necesario para el ajuste de las dinámicas del robot y el desempeño en el estado de avance.

2.1. Antecedentes

Para el desarrollo del presente proyecto de investigación, se realizó una búsqueda de trabajos relacionados con la robótica de enjambre y la robótica móvil en las plataformas ResearchGate, IEEE Xplore, ScienceDirect y Scopus. Esta búsqueda se enfocó en estudios realizados entre los años 2009 y 2022, con prioridad hacia aquellos que abordaran algoritmos y comportamientos propios de la robótica de enjambre, con énfasis en la agregación de robots hacia un objetivo común. De esta manera, se seleccionaron diez artículos considerados relevantes por su alineación con los objetivos del proyecto, así como por su aporte teórico y experimental a estrategias de agrupamiento basadas en agregación.

La robótica de enjambre es una rama de la robótica que toma inspiración en los comportamientos colectivos observados en la naturaleza, en especies como hormigas, abejas, peces y aves. Según la literatura revisada, los trabajos de Bayindir (2016); Brambilla et al. (2013)

y Sempere (2013) proporcionan una base sólida para la comprensión de los principios fundamentales de esta área.

Dentro de la inteligencia de enjambre, los algoritmos se inspiran en comportamientos colectivos presentes en la naturaleza, los cuales se traducen en interacciones entre robots. En esta revisión, tres artículos exploran los principales métodos y algoritmos de comportamiento de enjambre desarrollados (Alfredo & Ramos, 2022; Bayindir, 2016; Brambilla et al., 2013). Donde, por su aplicabilidad al proyecto, el algoritmo de agregación resulta ser el enfoque más adecuado. En este comportamiento, destacan dos categorías principales: la agregación libre, en la que los robots se agrupan sin estímulos externos, y la agregación mediada por el entorno.

La agregación mediada por el entorno se presenta cuando los robots se reúnen en puntos específicos influenciados por estímulos externos. Entre los diversos modelos desarrollados en esta área, el algoritmo *Beeclust* destaca por su capacidad para emular el comportamiento de abejas que se agrupan siguiendo gradientes de temperatura o puntos calientes. El trabajo que introdujo este algoritmo por primera vez fue "*Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system*" (Kernbach et al., 2009), en el cual se detalla cómo las abejas suelen guiarse hacia puntos calientes y agruparse en ese espacio. El trabajo propone un modelo robótico de agrupamiento donde una fuente de luz representa dicho punto de reunión, siendo la intensidad lumínica análoga al gradiente de temperatura.

Desde su introducción, el algoritmo *Beeclust* ha sido replicado y validado en entornos robóticos, tanto simulados como experimentales. En estas implementaciones, se ha utilizado una fuente de luz que representa el punto caliente, mientras los robots, equipados con sensores adecuados, toman decisiones basadas en la intensidad lumínica percibida. Este es el caso del trabajo de Hamann et al. (2008), donde se utiliza una configuración con robots que miden la

luminosidad local y, dependiendo de esta medida, se desplazan hacia áreas más iluminadas. Otros estudios han implementado una serie de modificaciones, como el uso de dos fuentes de luz con el propósito de evaluar la respuesta y mejorar la precisión en la agregación (Schmickl et al., 2009), el uso de obstáculos para estudiar la respuesta del comportamiento (Vogrin et al., 2020) y la interacción de dos enjambres en un mismo espacio de trabajo o arena teniendo objetivos opuestos (Bodi et al., 2012).

A su vez, se han explorado diseños con ciertas variaciones que amplían la aplicabilidad del algoritmo, mostrando cómo puede adaptarse a diferentes estímulos sensoriales, como sonido o gradientes complejos. En el trabajo de Arvin et al. (2014) reemplazaron la luz por una fuente de sonido, utilizando robots equipados con micrófonos que les permiten medir la intensidad acústica y ajustar su posición en función de los gradientes sonoros. Además, se han integrado sistemas de decisión basados en lógica difusa con el fin de optimizar el proceso de agrupamiento en escenarios más complejos.

En conclusión, por medio de la revisión de trabajos seleccionados, es notable que, si bien cada investigación presenta diferencias en términos de diseño, sensores o entornos; el algoritmo de agregación mediada por el entorno ha mostrado gran aplicabilidad para tareas de búsqueda y localización en sistemas de enjambre. El algoritmo *Beeclust* se destaca particularmente por su simplicidad y efectividad en la resolución de problemas de agrupamiento, haciéndolo ideal para el objetivo de este proyecto; no obstante, de los resultados obtenidos de los diferentes artículos, es observable que el éxito de los sistemas de agrupamiento depende en gran medida de factores experimentales como el tamaño de enjambre, la disposición de los robots, la precisión de los sensores, la estructura del robot y la naturaleza de los estímulos. Estos elementos serán clave para el desarrollo de la estrategia de agregación propuesta en este proyecto.

2.2. Robótica de Enjambre

La robótica de enjambres es un campo que se inspira en los comportamientos colectivos observados en varias especies, como hormigas, abejas, peces y aves. Estas especies exhiben un fenómeno llamado inteligencia de enjambre, por el cual individuos simples que parecen tener capacidades limitadas se vuelven más efectivos en colaboración.

Este enfoque se basa en la premisa de que hay tareas u objetivos que no pueden ser alcanzados por un solo agente, ya sea porque son demasiado complejos o porque tomaría demasiado tiempo o esfuerzo lograrlo como un agente único. Por ello, esta técnica emplea múltiples robots que tienen un diseño más simple, donde el comportamiento de enjambre permite que los agentes colaboren entre sí y con su entorno (Bayindir, 2016).

En un sistema de robótica de enjambre, los robots presentan las siguientes características:

- Son autónomos, es decir, operan de manera independiente dentro del entorno.
- Poseen capacidades de sensado y comunicación local, lo que les permite interactuar con su entorno y con otros robots cercanos.
- No tienen control centralizado, o tienen un control muy limitado o inexistente, y un acceso muy limitado a información global, lo que favorece la descentralización y hace que el sistema sea escalable y robusto.
- Cooperan entre sí para completar las tareas asignadas, aprovechando la interacción local.

No obstante, el diseño de algoritmos para controlar cada agente es uno de los principales desafíos en la robótica de enjambres. Por lo tanto, estos algoritmos deben ser de baja complejidad y adaptarse tanto a la percepción local del robot como a la proximidad de otros robots (Brambilla et al., 2013).

2.2.1. Métodos de Comportamientos Colectivos

Los sistemas robóticos de enjambre están diseñados para imitar los comportamientos observados en grupos de animales. Estos se caracterizan por una coordinación distribuida para realizar tareas complejas que es observable en diferentes comportamientos colectivos. En la robótica de enjambre, dicho comportamiento se ha llevado a cabo a través de muchas estrategias colectivas, incluyendo agregación, bandada, recogida de recursos o búsqueda de alimento, agrupamiento de objetos, navegación colaborativa y formación de rutas.

La agregación de robots se refiere a la capacidad de los enjambres de robots autónomos para reunirse en una región, como lo hacen las cucarachas o las abejas. Esto se realiza generalmente guiando a un robot con una máquina de estados finita probabilística o un algoritmo basado en evolución artificial para explorar el entorno local y asociarse con un grupo que tiene características definidas para formar un clúster estable.

En contraste, el comportamiento de bandada, que se inspira en el comportamiento de los peces y las bandadas de aves, puede ayudar a mantener formaciones mientras se evitan colisiones. Estos robots deben medir continuamente la distancia y la orientación en relación con sus vecinos más cercanos, lo que lleva a una percepción restringida.

El forrajeo colectivo, inspirado en las hormigas, permite la búsqueda y recolección de recursos para transportarlos a un nido definido. Aplicaciones avanzadas incluyen la clasificación y transporte de objetos por características, siendo útil en tareas como exploración planetaria y limpieza de residuos peligrosos. De manera relacionada, el transporte colectivo involucra la colaboración de múltiples robots para mover objetos pesados o grandes.

Por su parte, el agrupamiento de objetos reúne elementos que se encuentran dispersos en un área específica sin un destino concreto, permitiendo su clasificación en aplicaciones

industriales. Finalmente, la navegación colectiva utiliza la colaboración entre robots para llegar a un objetivo. En este se pueden dar dos casos: que todos los robots lleguen al mismo lugar o que solo uno lo haga, basado en la información proporcionada por el enjambre (Bayindir, 2016; Brambilla et al., 2013; Sempere, 2013).

2.2.2. Algoritmo de Agregación BEECLUST

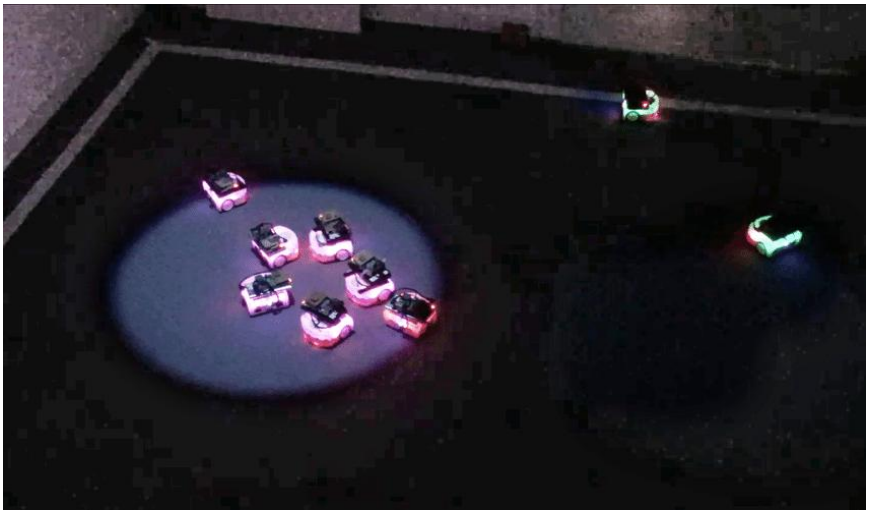
El algoritmo *Beeclust* fue introducido en el artículo *"Re-Embodiment of Honeybee Aggregation Behavior in Artificial Micro-Robotic System"* (Kernbach et al., 2009). Este modelo se inspira en el hecho de que las abejas recién nacidas se sienten instintivamente atraídas por la parte más cálida de la colmena y tienden a agruparse allí, llevando a un comportamiento de agregación basado en estímulos ambientales.

Investigaciones previas han mostrado que las abejas pueden detectar un gradiente de temperatura y utilizarlo para encontrar su camino hacia lugares más cálidos. Pero cuando las fluctuaciones de temperatura son pequeñas, las abejas emplean una estrategia cooperativa de búsqueda distribuida y descentralizada del área objetivo.

En el ámbito robótico, Beeclust fue adaptado para evaluar su eficacia como estrategia de agregación basada en el entorno. Resultados experimentales demostraron su capacidad para ensamblar robots alrededor de una señal, destacándose por sus características de inteligencia de enjambre: trabajo colaborativo, control descentralizado, ausencia de comunicación directa y escalabilidad. Estas propiedades han convertido al algoritmo en un referente estándar para la agregación basada en señales.

Figura 1.

Ejemplo de algoritmo adaptativo BEECLUST



Nota. Un enjambre de nueve robots ejecuta el algoritmo adaptativo BEECLUST. Los robots en color morado se encuentran en estado de espera (agrupamiento), mientras que los robots en color verde están en estado de exploración. Tomado de *Swarm robotics: Robustness, scalability, and self-X features in industrial applications* [Figura científica]. ResearchGate. Recuperado de: https://www.researchgate.net/figure/A-swarm-of-nine-robots-running-an-adaptive-BEECLUST-algorithm-Robots-in-purple-are-in_fig1_336892095 [consultado el 17 de diciembre de 2024].

2.2.2.1. Funcionamiento del Algoritmo

El algoritmo *Beeclust* puede ser representado como una máquina de estados finitos (FSM). Una FSM es un modelo matemático utilizado en la teoría de sistemas y automatización para describir el comportamiento de un sistema que puede encontrarse en un número finito de estados y cambiar entre ellos en respuesta a estímulos externos o condiciones internas. Un estado

representa una acción distinta, y las transiciones de un estado a otro son desencadenadas por eventos o condiciones predefinidas.

En este caso, la FSM organiza el comportamiento del robot en tres estados principales: *FWD* (avance), *RTT o Turn* (Rotación) y *WAIT* (espera) como se detalla en la Figura 2.

En el estado *FWD*, el robot se mueve a velocidad constante mientras verifica el entorno para evitar cualquier obstáculo. Cuando golpea algo, debe decidir si es una pared u otro robot. En caso de que el obstáculo sea una pared, el robot entra en el estado *RTT*, donde selecciona un ángulo aleatorio y realiza un movimiento de rotación. Después de que finaliza la rotación, el robot regresa al estado *FWD (Acevedo, Rios Diaz, Duque, et al., 2022; Acevedo, Rios Diaz, Garcia, et al., 2022)*.

Por otro lado, si el obstáculo es otro robot, el robot pasa al estado *WAIT*. En este estado, el robot espera un momento proporcional a la cantidad de luz observada en su posición de acuerdo con la fórmula:

$$t_{wait} = t_{\max} \cdot \frac{s^2(t)}{s^2(t) + \theta}$$

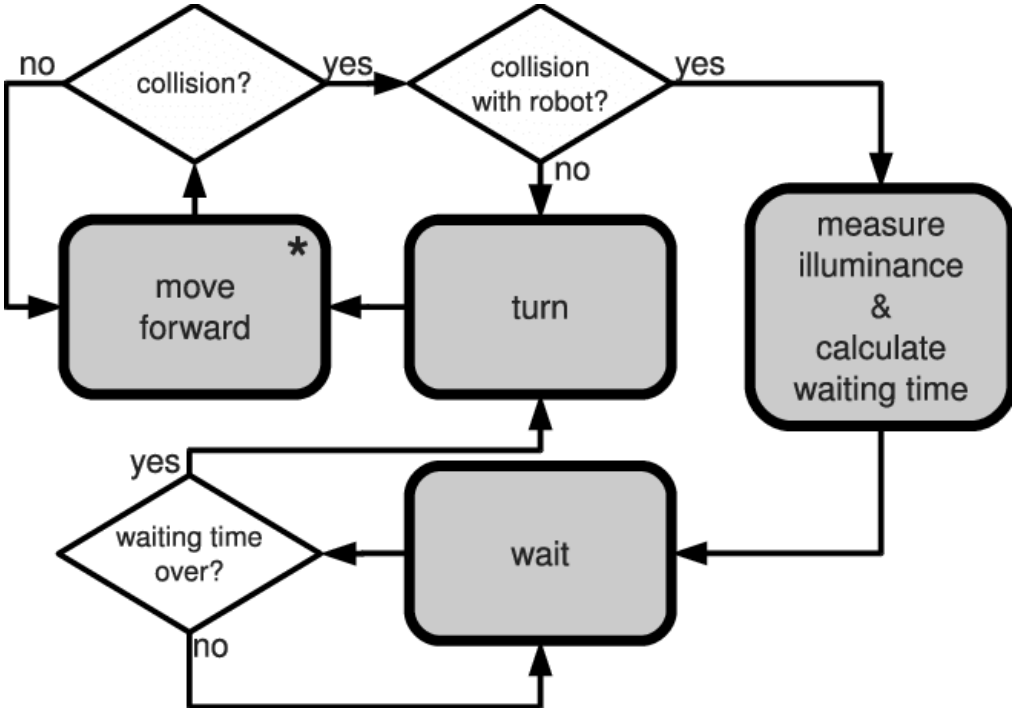
Donde:

- $t_{\max}$ Es el tiempo máximo de espera.
- $s(t)$ Es la intensidad de luz percibida por el sensor.
- θ Es el parámetro que regula la pendiente de la curva.

Después de este tiempo de espera, el robot regresa al estado *RTT*. Es importante señalar que la detección de obstáculos solo se realiza en el estado *FWD* (Arvin et al., 2014; Bodi et al., 2012; Hamann et al., 2008; Kernbach et al., 2009).

Figura 2.

FSM Algoritmo BEECLUST



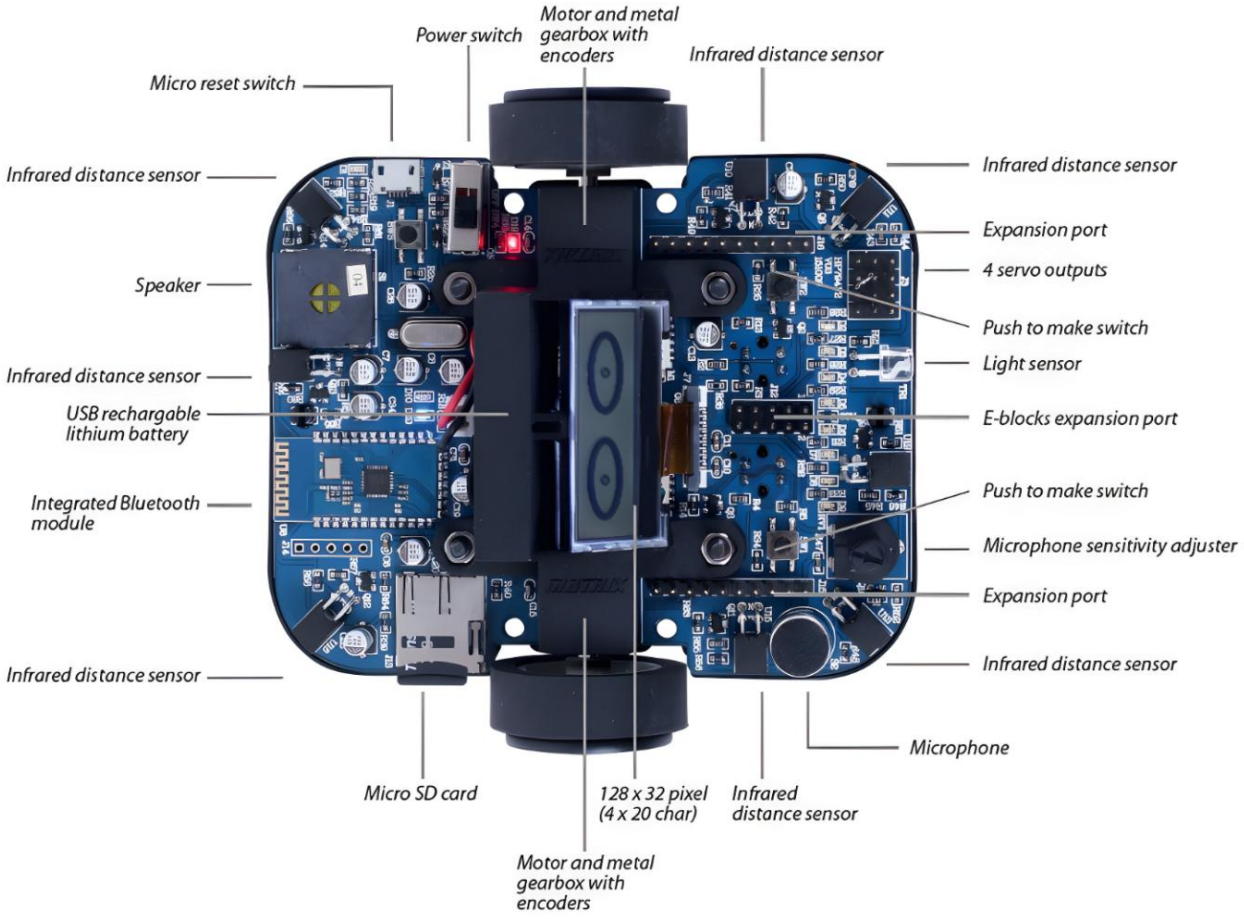
Imagen, fuente: https://www.researchgate.net/figure/Finite-state-machine-of-the-BEECLUST-algorithm-Boxes-represent-the-different-states-of_fig2_221531254 [accessed 21 Jan 2025]

2.3. Robot Formula AllCode

El Formula AllCode es una plataforma robótica diseñada para la experimentación en sistemas autónomos. La Figura 3 ilustra sus componentes principales.

Figura 3.

Partes del Robot Formula AllCode



Imagen, fuente: Currículo Formula AllCode

2.3.1. Controlador

El robot integra un microcontrolador dsPIC, encargado de procesar datos de sensores y controlar los actuadores mediante señales PWM. Su versatilidad permite gestionar señales analógicas y digitales, además de facilitar la comunicación con otros dispositivos.

2.3.2. Sensores

El Formula AllCode cuenta con numerosos sensores que permiten al robot percibir su entorno y tomar decisiones de manera independiente. Aquí hay algunos de estos sensores:

- **Sensores de distancia infrarrojos:** Ocho en total, se emplearon los frontales debido al entorno controlado del laberinto.
- **Sensor de luz:** Mide la intensidad lumínica, clave para el estado *WAIT* del algoritmo *Beeclust.*
- **Micrófono:** Detecta sonidos, útil en tareas de interacción entre robots.

2.3.3. Actuadores

Los actuadores ejecutan las acciones derivadas de los sensores:

- **Motores:** Permiten movimientos independientes como avance, rotación y giros.
- **Bocina (altavoz):** Emite sonidos predefinidos para la detección entre robots.
- **Panel LCD:** Muestra mensajes y gráficos básicos.

2.3.4. Interfaz de Conexión y Programación

El *Formula AllCode* cuenta con una interfaz de programación abierta y flexible, basada en una API (*Application Programming Interfaz* o Interfaz de Programación de Aplicaciones). La API es un conjunto de reglas y comandos que permite la comunicación entre el robot y un programa, facilitando el control sin tener que interactuar directamente con el hardware, actuando como un puente entre el robot y el lenguaje de programación que se decida usar.

La API tiene la ventaja de ser independiente de plataforma y de lenguaje, por lo que es posible la programación en diversos lenguajes como Python, C, C++ u o incluso entornos como Flowcode, LabVIEW, Matlab o APP inventor. Python fue seleccionado por su versatilidad y amplia documentación. La comunicación por Bluetooth facilita cargar programas y monitorear pruebas, lo que resulta idóneo para validar el algoritmo en un entorno controlado.

2.4. Control PID

Un controlador PID es un tipo de controlador que se utiliza ampliamente en aplicaciones industriales y de automatización debido a su capacidad de controlar y mantener una variable de proceso en un valor o rango deseado (*setpoint*). También son altamente flexibles para adaptarse a las diversas dinámicas del sistema (Ogata, 2010).

El término PID proviene de sus tres componentes principales: Proporcional (P), Integral (I) y Derivativo (D). Estos se describen brevemente a continuación:

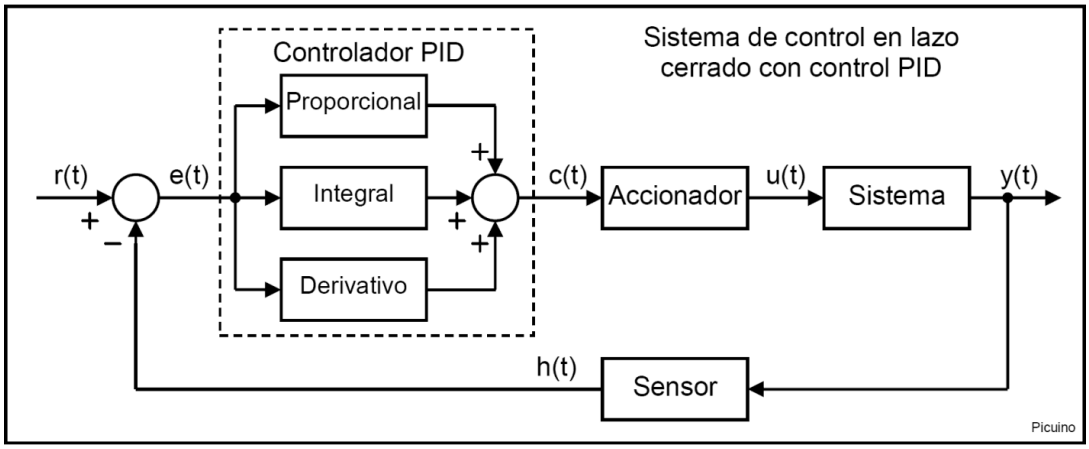
El controlador proporcional reacciona al error actual, que es la diferencia entre el valor deseado (*setpoint*) y el valor medido. Cuanto más significativo es el error, más significativa es la acción correctiva que se aplica de nuevo al sistema, lo cual es fundamental para la reducción exitosa de errores en el sistema.

Por su parte, la acción integral del controlador acumula los errores pasados a lo largo del tiempo. La función de esto es corregir errores constantes o de estado estacionario porque, con perturbaciones o características del sistema, como la fricción o la inercia, el error permanece y no disminuye. La acción integral también puede ser problemática si es excesiva; la acción integral puede llevar a un comportamiento oscilatorio.

Finalmente, la acción derivativa responde a la tendencia futura anticipada del error, calculando qué tan rápido está cambiando ese error a lo largo del tiempo. Al hacerlo, puede suavizar la respuesta del sistema y evitar una oscilación excesiva (Åström & Murray, 2019; Zayas Gato et al., 2020).

Figura 4.

Control PID



Imagen, fuente: https://www.picuino.com/es/control-pid.html

Aunque el controlador PID combina estas tres acciones para proporcionar un control robusto y versátil, existen casos donde la implementación de un controlador simplificado resulta más adecuada. Este es el caso del controlador PD, que utiliza únicamente las acciones proporcional y derivativa; aunque no elimina el error de estado estacionario por sí mismo, el controlador PD es eficaz para mejorar la estabilidad del sistema y la rapidez en la respuesta transitoria, lo que lo hace ideal para sistemas donde no se requiere un ajuste integral. Este tipo de control es común en aplicaciones como el control de velocidad de motores o la regulación de posición en sistemas automatizados.

En el proyecto actual, un controlador PD fue más apropiado que un controlador PID debido a la naturaleza específica del entorno de prueba, priorizando el movimiento continuo y estable del robot.

3. Metodología

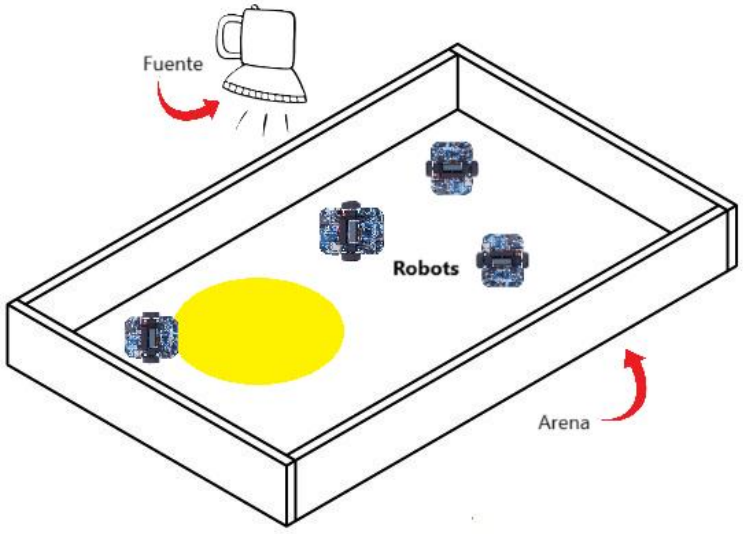
3.1 Caracterización del área de trabajo

El área de trabajo del enjambre está compuesta por tres elementos principales: la arena, la fuente y los robots. Como se muestra en la Figura 5, se dispone de una fuente de luz y robots que interactúan dentro de la arena. En este proyecto, los robots seleccionados son los *Formula AllCode*, y se optó por incluir una fuente de luz para representar el inventario perdido debido a los sensores disponibles y lo documentado en la literatura para la implementación del algoritmo *Beeclust*, ya que esta funciona como estímulo para la agregación.

Por su parte, la arena se define como el espacio físico donde interactúan los robots y la fuente de luz. Se tamaño va a depender de la cantidad de robots que componen el enjambre y el área de detección de cada uno. Por lo tanto, es importante considerar la relación entre el área de la fuente y el espacio ocupado por los robots, así como el radio de sensado que cada dispositivo puede cubrir, ya que estas variables influyen en el comportamiento colectivo observado.

Figura 5.

Esquema del área de trabajo del enjambre



3.1.1 Caracterización de los sensores infrarrojos de los robots

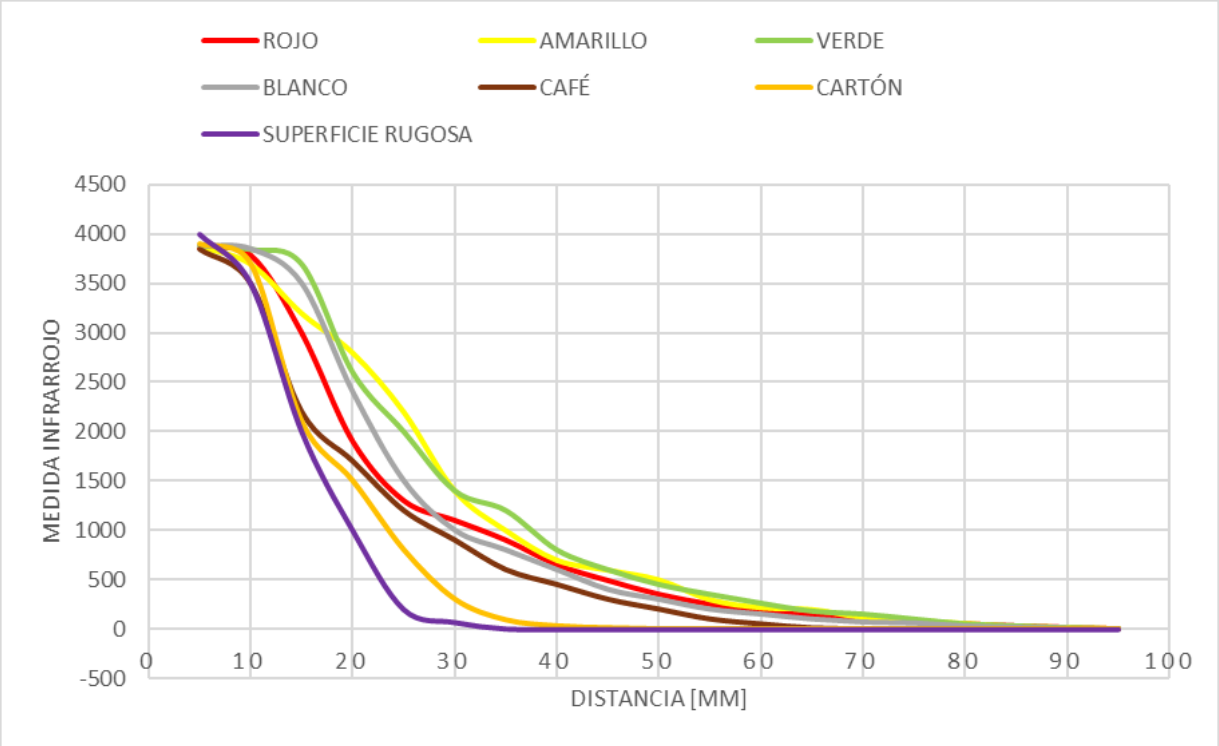
En esta etapa, se definió el rango de operación de los sensores infrarrojos integrados en los robots *Formula AllCode*. Estos sensores, distribuidos alrededor del perímetro del robot, forman un "disco" de sensado que permite una detección de 360 grados. Sin embargo, para este proyecto, se seleccionaron únicamente los sensores frontales debido a que el entorno de pruebas tiene la configuración de un laberinto.

Con el objetivo de determinar el alcance efectivo de los sensores infrarrojos, para lo cual se realizaron mediciones experimentales utilizando diversas superficies. Estos datos se llevaron a cabo registrando la respuesta del sensor al detectar objetos colocados a diferentes distancias, linealizando la señal analógica y convirtiéndola en una medida aproximada en milímetros. Además, se utilizó una muestra variada de superficies de distintos colores y texturas (papel de colores, cartón y superficies rugosas).

Según las especificaciones del sensor, el rango varía entre 0 y 4095 unidades. Pero, según pruebas, la señal puede saturarse en distancias extremadamente cercanas, generando valores con ruido. Entre los resultados se encontró que la lectura máxima registrada fue de aproximadamente 3900 unidades a una distancia de 4 a 5 mm, mostrando que el rango máximo efectivo depende del color y la textura de las superficies. En la Figura 6, se ilustra el comportamiento del sensor ante diferentes superficies y distancias.

Figura 6.

Alcance del Sensor Infrarrojo



Como se observa en la gráfica, la distancia en que las lecturas tienden a cero varía en función del color y la textura de la superficie, lo cual confirma la influencia de estos factores en la precisión del sensor. El rango máximo de detección fue diverso; las superficies de color rojo, amarillo y verde permitieron detecciones hasta aproximadamente 90 mm, mientras que la superficie rugosa y el cartón alcanzaron hasta alrededor de 35 mm.

En distancias que llegaron hasta los 25 mm, el sensor proporcionó lecturas consistentes, manteniéndose en un rango aproximado de 3000 unidades. A medida que aumentó de tal distancia, la señal comenzó a decaer gradualmente, presentando mayor ruido y menor calidad en la medición. Por lo tanto, se estableció un rango de trabajo efectivo entre 20 y 25 mm, seleccionando 20 mm como la distancia de operación para el dimensionamiento en este proyecto.

3.1.2 Determinación del tamaño de enjambre

Para el desarrollo del proyecto, es necesario definir un tamaño inicial para el enjambre de robot que representa el número máximo de agentes utilizados en el sistema. Este número es inicialmente hipotético y sirve como referencia para calcular el tamaño de la arena. En esta etapa, se consideró un enjambre de 10 robots para dimensionar adecuadamente la arena.

No obstante, durante las pruebas experimentales, es posible ajustar el tamaño del enjambre a dos robots, dado que este es el mínimo necesario para observar comportamientos colaborativos según lo señalado en la literatura revisada en el marco teórico (Bayindir, 2016; Brambilla et al., 2013). Igualmente, el tamaño del enjambre puede ampliarse en el futuro.

Este tamaño inicial es adecuado para la etapa experimental del presente proyecto, cuyo objetivo principal es probar y ajustar el algoritmo de agrupación, por lo que las pruebas y validaciones pueden realizarse de manera efectiva con solo dos agentes.

3.1.3 Dimensionamiento del espacio requerido

Para el dimensionamiento del área de trabajo destinada al enjambre de robots, se consideran las siguientes especificaciones:

- Dimensiones del robot: Cada robot tiene un tamaño de 10 x 10 cm.
- Alcance de sensado: Los sensores infrarrojos tienen un alcance efectivo de 2 cm.

Con base en estas especificaciones, se calcula el área de sensado individual de cada robot, considerando que el radio total es la suma del radio del robot (5 cm) y el alcance del sensor (2cm), resultando en un radio total de 7 cm. El área de sensado de cada robot se determina mediante la fórmula:

$$A_{sensado} = \pi r^2 = \pi(7)^2 = 153{,}94\ cm^2$$

Dado que el enjambre está compuesto inicialmente por 10 robots, el área total de sensado se calcula como:

$$A_{sensado_{total}} = A_{sensado} * N_{robots} = 1539{,}4\ cm^2$$

Se determina, como parámetro de diseño, un área de sensado que representa el 25% del área total de la arena necesaria para el enjambre, lo cual asegura suficiente espacio para el movimiento y la interacción entre robots. Por lo tanto, el área total requerida únicamente para el movimiento de los robots es:

$$A_{arena_{robots}} = \frac{A_{sensado_{total}}}{0{,}25} = 6157{,}52\ cm^2$$

- Área adicional para la fuente de luz:

Además del espacio necesario para los robots, se debe incluir un área adicional destinada a la agregación en el punto de mayor luz. Para este propósito, se considera un área equivalente al 20% del área ocupada por los robots, lo que resulta en:

$$A_{Extra_{PuntoCaliente}} = A_{arena_{robots}} * 20\% = 1231{,}50\ cm^2$$

- Área total de la arena

Finalmente, sumando el área para el movimiento de los robots y el área adicional para el punto de mayor luz, se obtiene el área total necesaria:

$$A_{Total_{Arena}} = A_{arena_{robots}} + A_{Extra_{PuntoCaliente}} = 7389\ cm^2$$

Para simplificar el diseño y normalizar las dimensiones, el área total de la arena se redondea a 8000 cm², lo que permite utilizar un espacio rectangular de aproximadamente 80 x 100 cm. Este dimensionamiento garantiza suficiente espacio para el movimiento de los robots y la interacción en torno a la fuente de luz, asegurando condiciones adecuadas para las pruebas experimentales.

3.1.4 Diseño de la arena

En estudios previos que implementan el algoritmo Beeclust, los enjambres de robots suelen operar en entornos abiertos, donde se desplazan aleatoriamente hasta encontrar un estímulo, como la luz. En estos espacios, los robots responden a obstáculos girando en ángulos aleatorios y se detienen temporalmente al detectar otro robot, guiados por la luminosidad captada. Este enfoque, sin barreras físicas, facilita la dispersión eficiente y la agrupación en puntos de interés o "puntos calientes", permitiendo cubrir grandes áreas mientras los robots colaboran de manera efectiva.

En este proyecto, se diseñó un laberinto que simula un entorno delimitado, como un almacén, ajustado a las características técnicas de los robots y los objetivos del estudio. Este diseño se debe a la limitación del alcance de los sensores infrarrojos, lo que dificulta la navegación confiable en espacios abiertos y podría generar colisiones o comportamientos caóticos. Igualmente, al ser un laberinto compuesto por corredores y bifurcaciones, se puede guiar a los robots en sus recorridos y simbolizar un entorno de almacén. El modelo del laberinto fue desarrollado en SolidWorks, permitiendo crear una estructura desmontable y adaptable para modificar tanto el tamaño del espacio como la disposición de los pasillos según los objetivos experimentales. En las Figuras 7 y 8, se muestra el diseño conceptual y el modelo físico del laberinto, proporcionando una vista general del entorno donde se realizaron las pruebas.

Figura 7.

Diseño del Laberinto Modelado

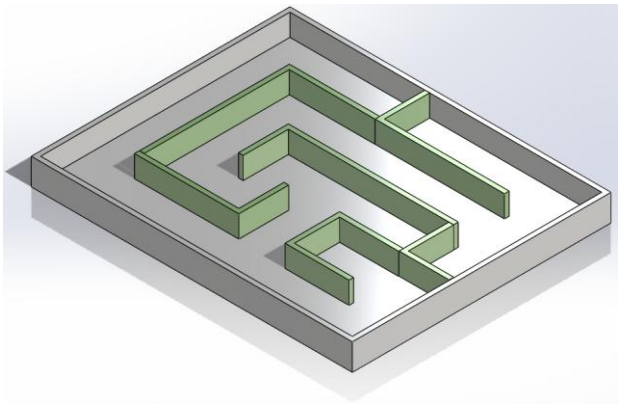


Figura 8.

Modelo Físico del Laberinto

3.2 Diseño del algoritmo de agrupamiento

El diseño del algoritmo de agrupamiento en este proyecto se basa en el algoritmo *Beeclust*, adaptándolo para operar dentro del entorno específico del laberinto y utilizando los robots *Formula AllCode*. Este proceso de diseño se llevó a cabo en tres etapas principales.

En la primera etapa, se diseñó el algoritmo para el funcionamiento de un solo robot, empleando una máquina de estados finitos. Posteriormente, en la segunda etapa, se integró la detección de otros robots dentro del área de trabajo. Esta funcionalidad es esencial para replicar el comportamiento colaborativo observado en el algoritmo *Beeclust*, donde la interacción con otros agentes influye directamente en la agregación hacia el objetivo común.

Finalmente, en la tercera etapa, se diseñó el algoritmo de agrupamiento considerando dos robots como base experimental, asegurando que el sistema sea escalable a un mayor número de robots. Para esto, el diseño se enfoca en mantener la escalabilidad, robustez y ausencia de comunicación centralizada, características fundamentales de la robótica de enjambre y del algoritmo *Beeclust*. En los siguientes apartados, se describen en detalle las etapas y componentes del diseño, así como las modificaciones necesarias para su implementación en este proyecto.

3.2.1 Diseño de algoritmo para el funcionamiento de un robot

El diseño del algoritmo para el funcionamiento de un único robot parte de una máquina de estados finitos (FSM). Este enfoque facilita la navegación y la detección de obstáculos en el laberinto.

En esta primera etapa, el objetivo del diseño es proporcionar al robot la capacidad de avanzar, rotar, retroceder y detenerse al alcanzar la fuente de luz. Para ello, se integra un sistema de control PD en el estado de avance (FWD).

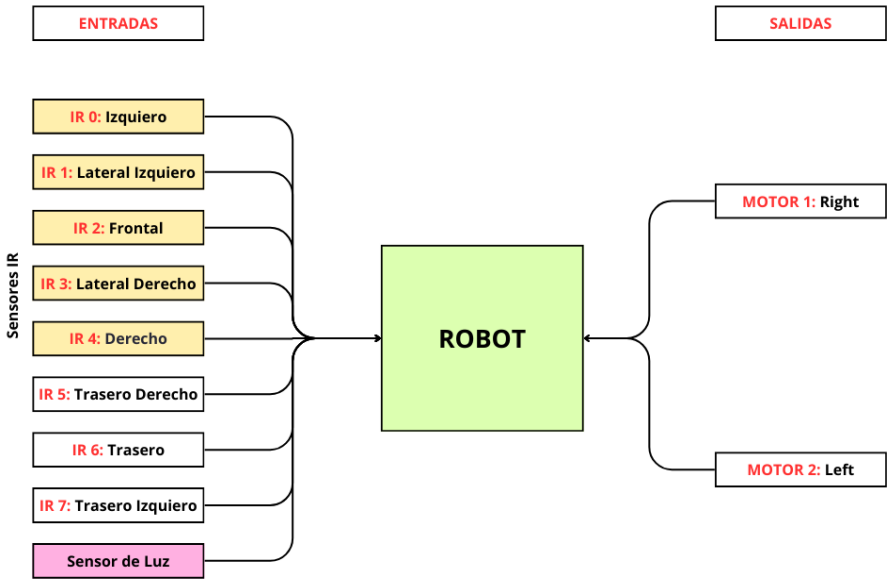
En los siguientes apartados, se detallan las entradas y salidas del sistema. Finalmente, se presenta el algoritmo de la máquina de estados finitos para un robot, que detalla lo desarrollado en esta sección, los estados definidos, eventos por estados y las transiciones entre ellos.

3.2.1.1. Entradas y Salidas del sistema

Las entradas incluyen los sensores infrarrojos, que detectan obstáculos, y el sensor de luz, que mide la intensidad luminosa y permite identificar la fuente. Las salidas corresponden a los motores, que responderán según los eventos definidos en la máquina de estados finitos. La Figura 9 muestra esta relación.

Figura 9.

Entradas y salidas del sistema de navegación de 1 robot



3.2.1.2. Estados del Robot

En un robot, el comportamiento del robot se divide en cuatro estados, definidos en la máquina de estados finitos (FSM):

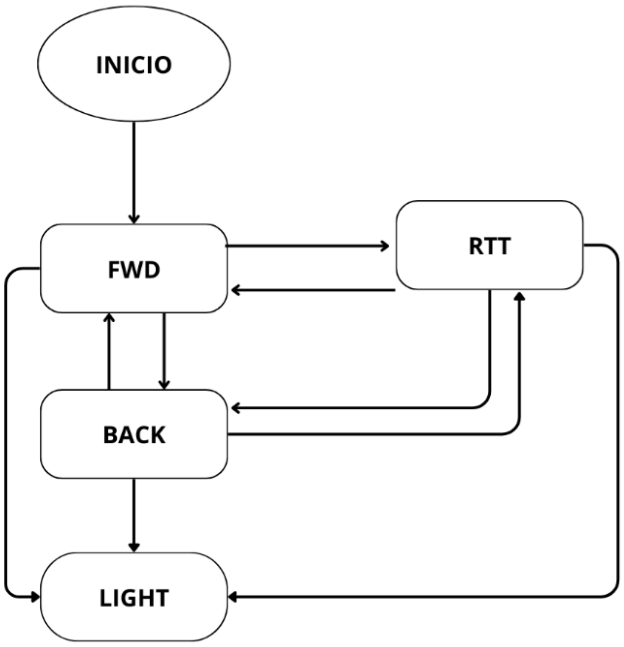
- Estado FWD: En este estado, el robot avanza. El movimiento se ajusta en función de las condiciones del entorno y del control implementado.

- Estado RTT: Cuando el robot detecta un obstáculo, cambia al estado de rotación. El robot realiza un giro que depende del tipo de obstáculo y continua su desplazamiento.
- Estado BACK: Si las ruedas del robot colisionan con las paredes del laberinto, el robot entra en estado de retroceso. Aquí, retrocede y realiza un pequeño giro para corregir su trayectoria y evitar quedar atascado.
- Estado LIGHT: Este es el estado final. El robot se detiene al llegar al objetivo (fuente de luz) y finaliza la navegación.

En la figura 10 se presenta la representación gráfica de los estados y sus transiciones.

Figura 10

Estados de trabajo



3.2.1.3. Implementación del Control PD en el Estado FWD

El controlador Proporcional-Derivativo (PD) implementado en el estado FWD es para mantener al robot centrado entre las paredes del laberinto. Para esto, realizará un ajuste a las velocidades de los motores según las lecturas de los sensores infrarrojos laterales.

A continuación, se describe el modelo matemático del controlador, el proceso de ajuste de las constantes K_p y K_d, y pruebas experimentales.

En primer lugar, se define un error (e) como medida de desviación respecto al centro del pasillo. Este error es la diferencia entre los sensores infrarrojos izquierdo S_i y derecho S_d.

$$e = dif = S_d - S_i$$

Cuando e es positivo, el robot está cerca a la pared izquierda; un valor negativo indica proximidad a la pared derecha.

El modelo matemático continuo del controlador PD se expresa como:

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t)$$

En este sistema digital, la derivada se aproxima utilizando diferencias finitas:

$$\frac{d}{dt} e(t) = \frac{e[k] - e[k-1]}{\Delta t}$$

Sustituyendo esta aproximación en la ecuación continua, obtenemos:

$$u[k] = K_p e[k] + K_d \frac{e[k] - e[k-1]}{\Delta t}$$

Se considera Δt constante, se define la variación del error como Δe[k]= e[k]-e[k-1] y modelo discreto final del controlador queda como:

$$u[k] = K_p e[k] + K_d \Delta e[k]$$

Estas expresiones permiten calcular la acción de control en función del error actual y su variación respecto al instante anterior. Posteriormente, las velocidades de los motores izquierdo V_L y derecho del robot V_R se ajustan utilizando la acción de control:

$$V_L = V_{base} - u[k]$$

$$V_R = V_{base} + u[k]$$

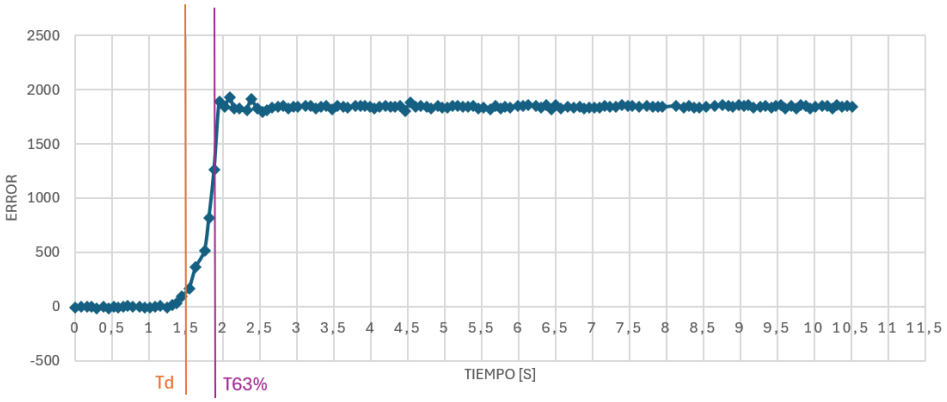
Donde V_{base} es una velocidad base predefinida. Entonces, si $u[k] > 0$, el motor izquierdo disminuye su velocidad y el derecho la incrementa, generando un giro hacia la derecha; mientras que si $u[k] < 0$, ocurre lo contrario.

Para determinar K_p y K_d se utilizó un método basado en la respuesta dinámica del sistema. Inicialmente, el robot se desplazó por un pasillo a una velocidad constante para calcular el error en función del tiempo. Los parámetros obtenidos de la curva de respuesta (Figura 11) son:

- Tiempo muerto (T_d): Instante en que el error comienza a incrementarse significativamente $T_d = 1{,}5\ s$.
- Valor final: El error se estabilizó alrededor de 1850.
- Constante de tiempo (T): Tiempo adicional necesario para que el error alcanzará el 63% de su valor final (1165,5): $T = 1{,}86s - 1{,}5s = 0{,}36s$,

Figura 11.

Respuesta del Sistema en Función del Tiempo



Estos valores se utilizaron para calcular las constantes mediante fórmulas derivadas del método de Ziegler-Nichols:

$$K_p = 1{,}2 \cdot \frac{T}{T_d} = 1{,}2 \cdot \frac{0{,}36}{1{,}5} = 0{,}288$$

$$K_d = 0{,}5 \cdot T = 0{,}5 \cdot 0{,}36 = 0{,}18$$

Por lo tanto, la ecuación del controlador PD implementado es:

$$u[k] = 0{,}288 \cdot e[k] + 0{,}18 \cdot (e[k] - e[k-1])$$

El controlador PD ajustado mostró un comportamiento estable en pruebas iniciales. El robot logró mantener una trayectoria centrada en el pasillo, minimizando colisiones con las paredes y reduciendo oscilaciones. Sin embargo, las constantes K_p y K_d podrán ser refinadas a través de pruebas adicionales para optimizar la fluidez del movimiento.

3.2.1.4. Algoritmo de la Máquina de Estados Finitos de un Robot

A continuación, se presenta el algoritmo implementado en el robot para su navegación en el laberinto. Allí, se indican los 4 estados establecidos junto con los eventos que desencadenan las respectivas acciones. Igualmente, se detalla está información en la Tabla 1.

Tabla 1
Algoritmo de la Máquina de Estados Finitos

ALGORITMO DE LA MÁQUINA DE ESTADOS FINITOS											
ESTADO	**EVENTO**						**ACCIÓN**				
FWD	Sistema de control						fa.SetMotors (speedL, speedR)				
	Caso	**SENSOR IR**					SET MOTOR R	SET MOTOR L	BACKWARDS	RIGHT (GIRO)	LEFT (GIRO)
		0	**1**	**2**	**3**	**4**					
RTT	1. Callejón cerrado	> 1200	NA	> 1200	NA	> 1200	-	-	-	180	-
	2. Obstáculo al frente y a la derecha, izquierda libre	< 10	NA	> 1200	NA	> 10	-	-	-	-	80
	3. Obstáculo al frente y a la izquierda, derecha libre	> 10	NA	> 1200	NA	< 10	-	-	1. 10	2. 80	-
	4. Obstáculo al frente	< 10	NA	> 1200	NA	< 10	-	-	1. 10	2. 90	
BACK	5. Choque rueda derecha	NA	NA	NA	>2500 OR	>2500	-	-	1. 10	-	2. 20
	6. Choque rueda izquierda	>2500 OR	>2500	NA	NA	NA	-	-	1. 10	2. 20	-
LIGHT	Sensor luz >4000						SetMotors (0, 0). Finalizar tarea				

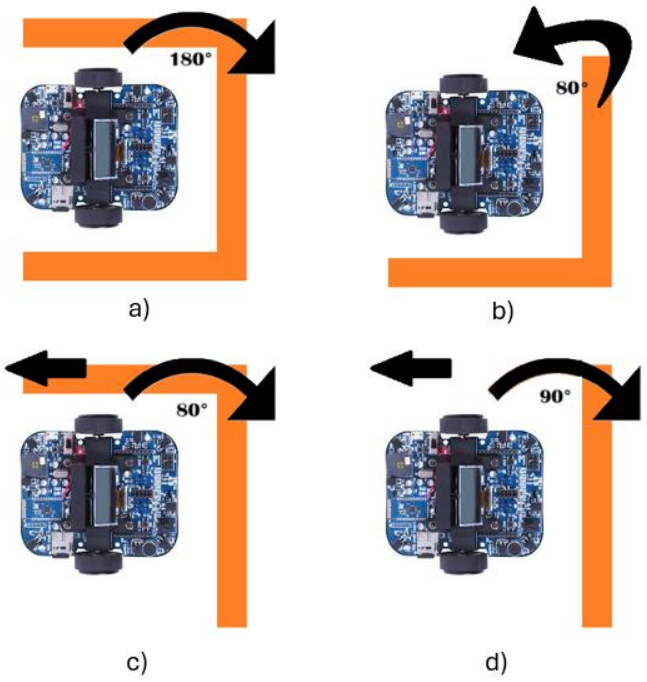
En el estado FWD, el robot avanza utilizando el controlador PD implementado. Este estado es el comportamiento predeterminado siempre que el robot no detecte obstáculos ni colisiones.

El estado RTT se activa cuando el robot detecta un obstáculo. Dependiendo de las condiciones, el robot realiza las siguientes acciones, representadas en la Figura 12.

- Evento 1: Paredes al frente y en ambos costados (callejón cerrado), gira 180° hacia la derecha para regresar por el camino recorrido.
- Evento 2: Obstáculo al frente y en el lado derecho, costado izquierdo libre; giro de 80° hacia la izquierda, ya que este ángulo proporciona un mejor resultado que 90°.
- Evento 3: Obstáculo al frente y en el lado izquierdo, costado derecho libre; pequeño retroceso y luego gira 80° hacia la derecha.
- Evento 4: Obstáculo al frente, realiza retroceso y un giro de 90° hacia la derecha.

Figura 12

Eventos del Estado RTT

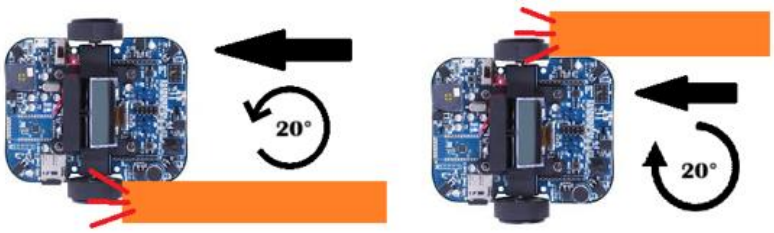


El estado BACK, se activa cuando el robot detecta colisiones a través de sus ruedas laterales, con las siguientes acciones (representadas en la Figura 13):

- Evento 5: Rueda derecha colisiona con una pared o se atasca, el robot retrocede ligeramente y gira 20° hacia la izquierda para corregir su trayectoria.
- Evento 6: Rueda izquierda colisiona con una pared o se atasca, realiza un retroceso y un giro de 20° hacia la derecha.

Figura 13

Eventos del Estado BACK



Finalmente, el estado LIGHT se activa cuando el sensor de luz detecta la fuente de luz objetivo. En este estado, el robot detiene su movimiento, indicando que ha completado la tarea de navegación.

3.2.2 Detección de otros robots

La detección de robots permite diferenciar entre obstáculos estáticos, como las paredes del laberinto, y otros robots en el entorno. Este comportamiento es esencial para replicar la dinámica de los algoritmos de enjambre, como *Beeclust.*

Inicialmente, se consideró el uso de sensores infrarrojos para la detección entre robots debido a su amplia referencia en la literatura. Sin embargo, las limitaciones del hardware y la API del robot *Formula AllCode* restringieron esta opción, ya que los sensores están configurados

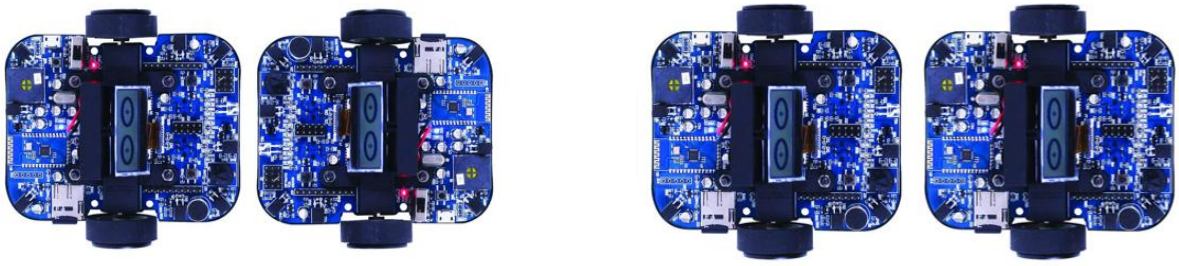
únicamente para reflejar señales y no pueden emitirlas por medio de programación. Ante estas restricciones, se exploraron alternativas con los componentes disponibles en el robot.

Se seleccionaron el micrófono y la bocina integrados como solución para la detección, descartando opciones externas como sensores de color o módulos de radiofrecuencia. El micrófono permite captar niveles de sonido, mientras que la bocina emite tonos específicos. Este enfoque ofrece una solución práctica y escalable, alineada con las capacidades del robot. En este proyecto, se utilizó una frecuencia de 392 Hz (nota Sol, G4) emitida durante 2 segundos. Si el micrófono del robot receptor detecta un nivel de sonido superior a un umbral establecido, se interpreta que otro robot está cercano; de lo contrario, el sistema lo clasifica como un obstáculo estático.

Para determinar el umbral de detección, se realizaron pruebas experimentales bajo distintas configuraciones de posición entre el emisor y el receptor: frente a frente, emisor detrás del receptor y en los costados izquierdo y derecho. Estas mediciones se promediaron para minimizar el impacto del ruido ambiental y garantizar resultados consistentes. Este modelo de detección demostró ser efectivo dentro de las limitaciones del hardware, proporcionando una base funcional para la colaboración entre robots en el contexto del proyecto.

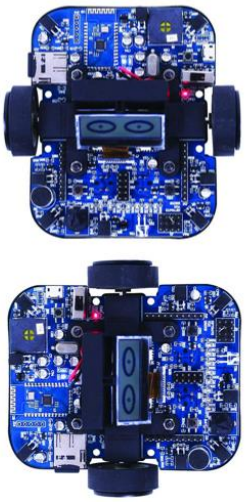
Figura 14

Posiciones de los robots para detección.



a) Frontal con frontal

b) Emisor detrás del detector



c) Emisor al costado izquierdo del detector

d) Emisor al costado derecho del detector

Tabla 2

Mediciones del micrófono.

Posición	**Promedio**				
Frontal con frontal	997	1069	1012	1047	998
Emisor detrás del detector	954	1079	1112	978	1114
Emisor al costado izquierdo	977	1188	994	910	991
Emisor al costado derecho	938	1000	1255	812	906

Los resultados de las mediciones (Tabla 2) evidenciaron variaciones según la posición relativa de los robots. Configuraciones con mayores ángulos o interferencias, como "emisor al costado derecho", mostraron valores más bajos. Basándose en estas mediciones, se estableció un umbral de detección de 918 Hz, promedio de los valores más bajos entre las posiciones evaluadas. Este umbral asegura sensibilidad en la detección de robots cercanos, manteniendo robustez frente a señales de fondo y ruido ambiental.

El sistema, diseñado para emitir y captar tonos específicos, sigue un proceso iterativo que simplifica su implementación y escalabilidad. Aunque su diseño aprovecha exclusivamente los componentes integrados del robot, presenta limitaciones como sensibilidad al ruido ambiental y alcance reducido, adecuado solo para distancias cortas. En iteraciones futuras, incorporar métodos para identificar direcciones relativas del sonido podría potenciar su precisión y permitir aplicaciones en entornos más complejos.

3.2.3 Diseño del algoritmo de agrupamiento para 2 robots

El diseño del algoritmo de agrupamiento en este proyecto amplía la funcionalidad de la máquina de estados previamente desarrollada para la navegación de un solo robot.

En esta etapa, se incorpora la detección de otros robots al algoritmo, introduciendo el nuevo estado *WAIT*, inspirado en la lógica del algoritmo *Beeclust*. Este estado permite que los robots esperen durante un tiempo calculado en el punto caliente (fuente de luz), favoreciendo el agrupamiento y replicando el comportamiento observado en sistemas de enjambre.

La sección se estructura en dos partes principales. Primero, se define el estado *WAIT*, detallando los parámetros y pruebas realizadas. Posteriormente, se describe el algoritmo completo de agrupamiento integrando el nuevo estado.

3.2.3.1. Estado WAIT

El estado *WAIT* permite que los robots permanezcan en el punto caliente (fuente de luz) durante un tiempo determinado, replicando el comportamiento de agrupamiento del algoritmo Beeclust.

El cálculo del tiempo de espera se basa en una versión ajustada de la ecuación *Beeclust*, planteada previamente en el marco teórico. Esto se debe a que la fórmula considera un entorno oscuro con una intensidad mínima de luz ($s(t) = 0$). Sin embargo, en este proyecto, al no contar

con dichas condiciones ni equipos especializado, se consideró la luz ambiental presente en el entorno experimental. De esta forma, la ecuación se modifica para incluir un valor mínimo de intensidad $s_{\min}$ que representa la luz ambiental medida durante las pruebas.

La ecuación ajustada queda expresada como:

$$tiempo\ de\ espera = t_{\max} \cdot \frac{(s(t) - s_{\min})^2}{(s(t) - s_{\min})^2 + \theta}$$

Donde:

- $t_{\max}$ Es el tiempo máximo de espera.
- $s(t)$ Es la intensidad de luz percibida por el sensor.
- $s_{\min}$ Valor mínimo que representa la luz ambiental, definido experimentalmente como 500 unidades.
- θ Es el parámetro que regula la pendiente de la curva, ajustando el tiempo de espera en función de la intensidad lumínica

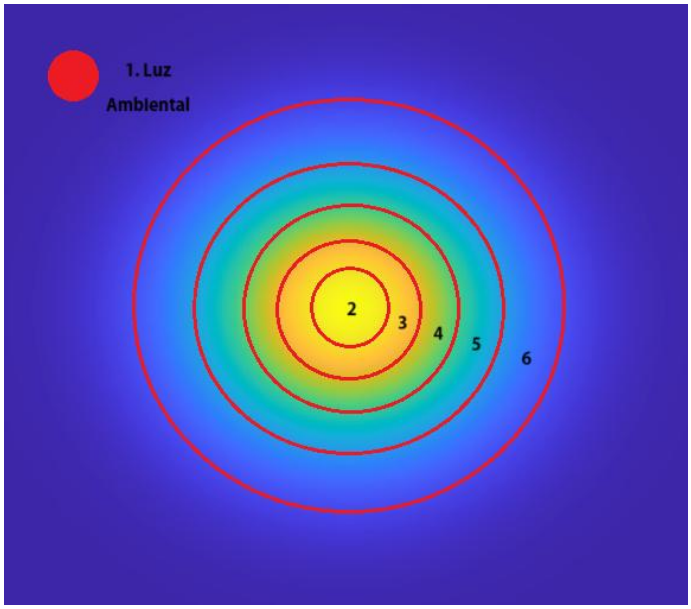
El valor de $s_{\min} = 500$ se determinó durante las pruebas de calibración del sensor, descritas a continuación.

Para establecer el rango efectivo de intensidad lumínica, se realizaron mediciones en seis posiciones distintas alrededor de la fuente de luz (Figura 15):

- Posición 1: Luz ambiental (sin fuente de luz).
- Posición 2: Centro de la fuente de luz, zona de máxima intensidad.
- Posición 3: A un cuarto de distancia desde el centro.
- Posición 4: A la mitad del radio de la circunferencia de la fuente.
- Posición 5: A tres cuartos de distancia desde el centro.
- Posición 6: En el borde de la circunferencia de la fuente.

Figura 15

Posiciones de las mediciones de la fuente de luz.



En cada posición, se tomaron 10 mediciones con el sensor de luz y se calculó el promedio para mitigar el impacto del ruido ambiental. Los resultados obtenidos se presentan en la Tabla 3.

Tabla 3

Mediciones del sensor de luz

	Posición					
	1	2	3	4	5	6
Promedio	488	3968	3311	2853	2327	1056

Con base en estos datos, se definió un rango efectivo de intensidad lumínica entre 500 y 4000 unidades. Este rango asegura que el robot distinga entre la luz ambiental ($s_{\min} < 500$) y la fuente de luz. Cuando la intensidad lumínica es inferior a 500, el robot continúa su navegación sin detenerse.

La fórmula ajustada para calcular el tiempo de espera es:

$$tiempo\ de\ espera = t_{\max} \cdot \frac{(s(t) - 500)^2}{(s(t) - 500)^2 + \theta}$$

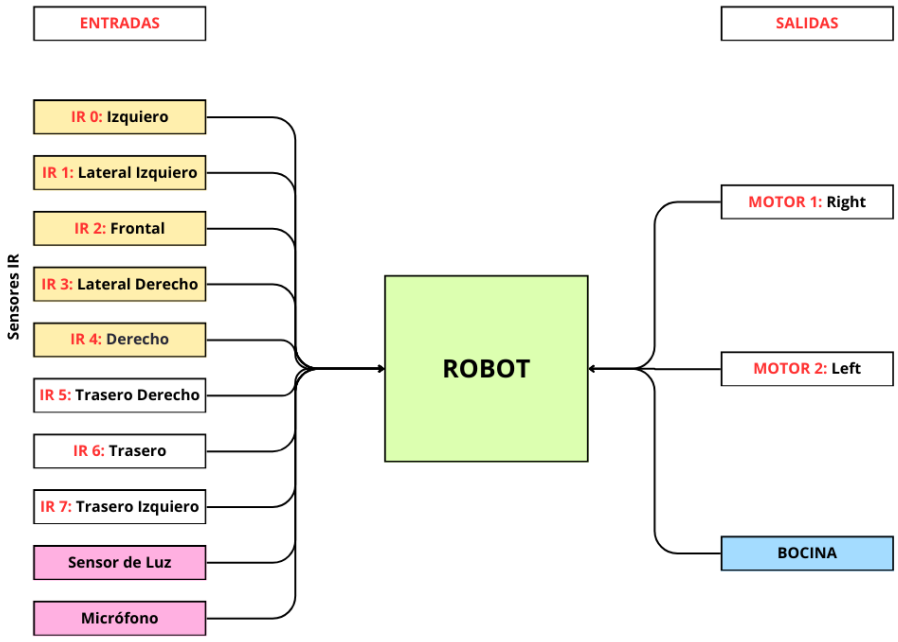
Esta modificación permite adaptar la ecuación original del algoritmo *Beeclust* a las condiciones experimentales de este proyecto, manteniendo la esencia del comportamiento esperado mientras se consideran las limitaciones prácticas del entorno y del sensor.

3.2.3.2. Entradas y Salidas

El sistema de agrupamiento de robots con las entradas y salidas modificadas se presenta en la Figura 16, añadiendo el micrófono y la bocina.

Figura 16

Entradas y salidas del sistema de agrupamiento de robots



3.2.3.3. Estados del Robot para el Algoritmo de Agrupamiento

El comportamiento del robot se organiza en una máquina de estados finitos compuesta por cuatro estados principales*: FWD, RTT, BACK* y *WAIT*. Estos estados definen las acciones del robot

según las condiciones detectadas en el entorno, siguiendo una lógica de navegación y agrupamiento inspirada en *Beeclust*.

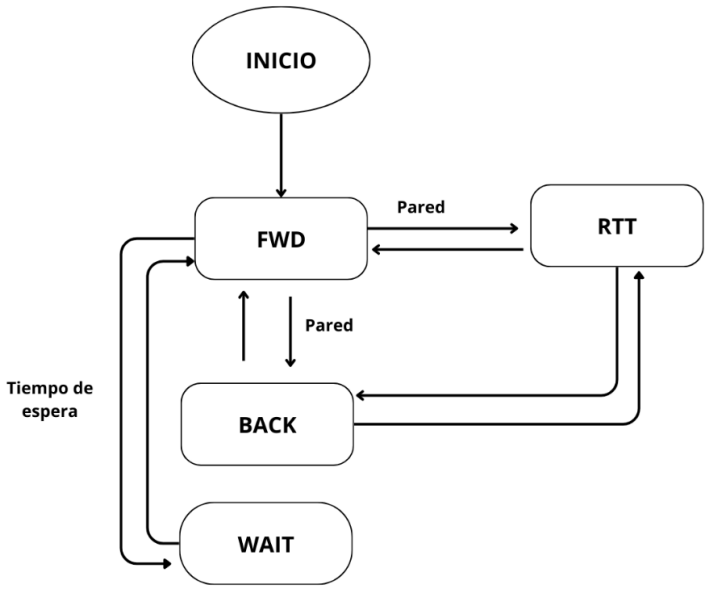
En el estado FWD, el robot avanza mientras utiliza el control PD para mantenerse centrado en el pasillo. Si detecta una pared, pasa al estado RTT, donde gira según la configuración del obstáculo para evitar colisiones. En caso de choque, entra en el estado BACK, retrocede ligeramente y corrige su trayectoria antes de regresar al estado FWD.

El estado WAIT se activa al identificar otro robot como obstáculo o al detectar la luz. En este estado, el robot se detiene en la fuente de luz y calcula un tiempo de espera basado en la intensidad lumínica percibida. Este tiempo, ajustado dinámicamente, favorece el agrupamiento colaborativo antes de volver al estado FWD.

En la Figura 17, se ilustra el diagrama de estados y sus transiciones.

Figura 17

Estados de trabajo



3.2.3.4. Algoritmo Máquina de Estados del Agrupamiento de Robots

El algoritmo de navegación y agrupamiento divide el comportamiento del robot en los estados *FWD, RTT, BACK* y *WAIT*. Los estados *FWD, RTT* y *BACK* mantienen las acciones y transiciones descritas previamente, mientras que el estado *LIGHT* ha sido reemplazado por *WAIT* para implementar la lógica de agrupamiento inspirada en *Beeclust*.

En la Tabla 4, se describen las condiciones, eventos y acciones asociadas a cada estado, incluyendo las transiciones actualizadas para el estado *WAIT*. Esta tabla resume las configuraciones de los sensores y las acciones de los motores, lo que facilita la comprensión de la lógica del sistema.

Tabla 4

Algoritmo de la Máquina de Estados Finitos Agrupamiento

ALGORITMO DE LA MÁQUINA DE ESTADOS FINITOS												
ESTADO	**¿Detecta Obstáculo?**	**EVENTO**						**ACCIÓN**				
FWD	No	Sistema de control						fa.SetMotors(speedL, speedR)				
		Caso	**SENSOR IR**					SET MOTOR R	SET MOTOR L	BACKWARDS	RIGHT (GIRO)	LEFT (GIRO)
			0	**1**	**2**	**3**	**4**					
RTT	Si. Es Pared	1. Callejón cerrado	> 1200	NA	> 1200	NA	> 1200	-	-	-	180	-
		2. Obstáculo al frente y a la derecha, izquierda libre	< 10	NA	> 1200	NA	> 10	-	-	-	-	80
		3. Obstáculo al frente y a la izquierda, derecha libre	> 10	NA	> 1200	NA	< 10	-	-	1. 10	2. 80	-
		4. Obstáculo al frente	< 10	NA	> 1200	NA	< 10	-	-	1. 10	2. 90	
BACK	Si. Es Pared	5. Choque rueda derecha	NA	NA	NA	>2500 OR	>2500	-	-	1. 10	-	2. 20
		6. Choque rueda izquierda	>2500 OR	>2500	NA	NA	NA	-	-	1. 10	2. 20	-
WAIT	Si. Es robot	Leer Sensor de Luz entre [500, 4000]						SetMotors(0, 0).		$tiempo\ de\ espera = t_{max} \cdot \frac{(s(t) - s_{min})^2}{(s(t) - s_{min})^2 + \theta}$		

3.3 Validación del sistema

En esta sección se valida el sistema diseñado para garantizar el desempeño del algoritmo de agrupamiento. Por este motivo, se describen las pruebas experimentales realizadas para evaluar el sistema bajo diferentes condiciones controladas. Estas pruebas abordan tres aspectos fundamentales:

1. **Tiempo de espera basado en la luminosidad**: Determinar la precisión del cálculo del tiempo de espera en el estado *WAIT*, derivado de la ecuación ajustada *Beeclust*, considerando las condiciones lumínicas detectadas.
2. **Precisión en la detección de robots**: Validar la capacidad del sistema para diferenciar entre robots cercanos y obstáculos estáticos.
3. **Respuesta a variables externas**: Evaluar el impacto de factores como el tamaño del área de trabajo, la posición de la fuente de luz y el tamaño de la fuente de luz en el desempeño del sistema de agrupamiento.

Para todas las pruebas, se utilizaron robots Formula AllCode en un entorno controlado con parámetros específicos para la iluminación, el tamaño del área de trabajo y la posición de la fuente de luz.

3.3.1 Tiempo de espera basado en la luminosidad.

En esta sección, se valida el cálculo del tiempo de espera implementado en el estado *WAIT*. Para llevar a cabo esta validación, se han diseñado dos pruebas experimentales. La primera prueba verifica el robot espera el tiempo calculado por la fórmula. La segunda prueba evalúa el impacto de dos parámetros de la ecuación: θ y el tiempo máximo de espera $t_{\max}$, sobre el comportamiento del sistema.

3.3.1.1. Verificación Experimental del Tiempo de Espera.

En esta prueba se verifica que el robot espera el tiempo calculado por la fórmula en condiciones controladas de iluminación. Se mide el tiempo real transcurrido desde que el robot entra en el estado de espera hasta que lo abandona, comparándolo con el tiempo calculado por la ecuación.

Los parámetros establecidos para estas pruebas fueron:

- $t_{\max} = 30\ segs$
- $s_{min} = 500$
- $\theta = 10000$

3.3.1.2. Impacto del Parámetro θ en el Comportamiento del Sistema.

Esta prueba está enfocada en el impacto del parámetro θ, por el cual se relaciona la intensidad lumínica con el tiempo de espera. Para evaluar su influencia, se realizó una serie de pruebas donde θ se ajustó en un rango amplio, entre 5.000 y 100.000. Se analizaron los tiempos de espera en diferentes niveles de intensidad lumínica: baja ($s < 800$), media ($800 \leq s < 2.500$) y alta ($s > 2.500$), manteniendo las condiciones previamente descritas.

3.3.2 Precisión en la Detección de Robots.

La detección de otros robots influye directamente en la capacidad de colaboración y agrupamiento del sistema. Esta evaluación consideró tanto la detección de robots como de obstáculos, para cuantificar la precisión y robustez del sistema. Igualmente, se diseñó la prueba experimental con diferentes escenarios, distancias y orientaciones:

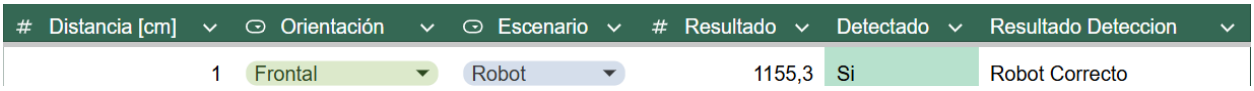
- Distancias: 5 niveles en centímetros.
- Orientaciones: Frontal-Frontal, lateral izquierdo y lateral derecho.
- Escenarios: Detección de un robot o detección de un obstáculo.

Durante la prueba, se utilizó un umbral previamente definido para clasificar cada detección como positiva (“Si”) o negativa (“No”). La clasificación correcta requería que el sistema identificara como "Sí" cuando había un robot presente o como "No" cuando el objeto era un obstáculo. Por el contrario, las detecciones incorrectas se registraron si el sistema confundía un robot con un obstáculo o viceversa.

La Figura 18 ilustra un ejemplo representativo del diseño de la prueba, mostrando las diferentes configuraciones. Las detecciones se registraron en una tabla de resultados, que incluyó una columna de clasificación visual con colores para facilitar la interpretación, siendo verde una detección correcta y rojo una incorrecta.

Figura 18

Muestra de la prueba de detección



# Distancia [cm]	Orientación	Escenario	# Resultado	Detectado	Resultado Deteccion
1	Frontal	Robot	1155,3	Si	Robot Correcto

3.3.3 Respuesta a Variables Externas

En esta sección se valida el funcionamiento del algoritmo frente a variables externas a través de tres pruebas. La primera prueba consiste en variar el tamaño del área de trabajo para medir el tiempo que tardan los dos robots en agruparse bajo la fuente de luz. La segunda prueba evalúa el efecto de cambiar la ubicación de la fuente de luz en la arena. La tercera revisará el impacto del tamaño de la fuente de luz en el tiempo, cercanía y dispersión del agrupamiento de robots.

3.3.3.1. Tamaño del Área de Trabajo

Para esta prueba se consideraron cuatro tamaños proporcionales al tamaño completo de la arena: 100%, 75%, 50% y 25%. Estos tamaños se seleccionaron para analizar cómo diferentes escalas del área afectan la capacidad de los robots para localizar y agruparse en la fuente de luz.

Se establecieron las siguientes condiciones iniciales constantes:

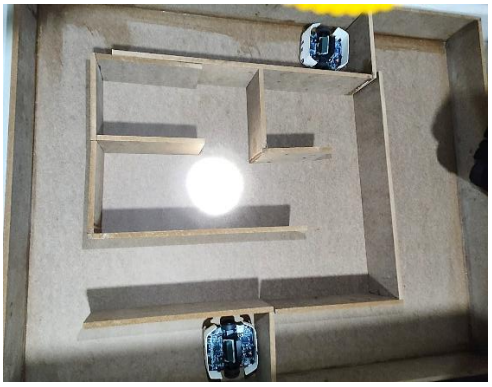
- Los robots se ubicaron siempre en posiciones aleatorias al inicio de cada prueba, manteniendo la misma configuración para todas las repeticiones.
- La fuente de luz permaneció fija en el centro del área de trabajo en todas las muestras.
- La distribución de paredes se mantuvo constante en cada uno de los tamaños.

La Figura 19 muestra la distribución inicial para cada tamaño, incluyendo las posiciones de los robots y la ubicación de la fuente de luz.

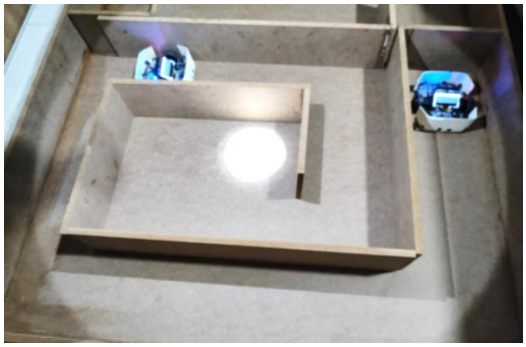
Figura 19

Distribución Inicial de la Prueba de Tamaño de la Arena.

a) 100%



b) 75%



c) 50%

d) 25%

Además de registrar el tiempo que tardaron los robots en agruparse bajo la fuente de luz en cada repetición, se calcularon métricas como el tiempo promedio, la desviación estándar y el coeficiente de variación.:

- Tiempo promedio: Representa el tiempo típico de agrupamiento para cada tamaño del área.

$$T_{promedio} = \frac{\sum Tiempos\ individuales}{n}$$

- Desviación estándar (σ): Evalúa la dispersión de los tiempos respecto al promedio, indicando la consistencia del sistema.

$$\sigma = \sqrt{\frac{\sum\left(Tiempo\ individual - T_{promedio}\right)^2}{n}}$$

- Coeficiente de variación (CV): Relaciona la desviación estándar con el promedio, expresando la variabilidad en términos relativos.

$$CV = \frac{\sigma}{T_{promedio}} * 100$$

3.3.3.2. Posición de la Fuente de Luz dentro de la Arena

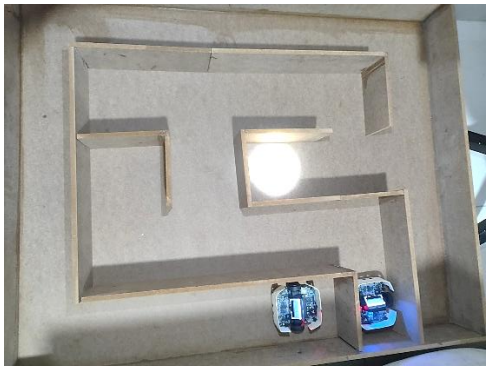
Para evaluar cómo la posición de la fuente de luz afecta el tiempo de agrupamiento de los robots, se definieron cuatro ubicaciones dentro del área de trabajo: el centro de la arena, dos esquinas y un punto aleatorio dentro del espacio.

En cada prueba, los robots comenzaron desde posiciones iniciales fijas para garantizar la consistencia en las condiciones de partida. Estas posiciones fueron determinadas aleatoriamente dentro de la arena, pero se mantuvieron iguales para las repeticiones realizadas en cada ubicación de la fuente de luz. El tamaño de la arena se mantuvo al 100% durante toda la prueba, y las condiciones de iluminación controladas. Se utilizó el algoritmo de agrupamiento Beeclust con los parámetros establecidos en pruebas previas.

A continuación, se presenta una referencia visual que muestra las posiciones evaluadas dentro de la arena, para facilitar la interpretación de los resultados (Figura 20).

Figura 20

Distribución Inicial de la Prueba de Posición de la Fuente de Luz en la Arena.



a) Centro

b) Esquina 1

c) Esquina 2

d) Punto Aleatorio

3.3.3.3 Tamaño de la Fuente de Luz

Por último, se evalúa cómo el tamaño de la fuente de luz puede influir en el tiempo de agrupamiento de los robots y en su cohesión final. Para este fin, se llevaron a cabo pruebas utilizando cuatros tamaños diferentes de fuente en relación con el tamaño total de la arena: 1/8, 1/4, 3/8, y 1/2.

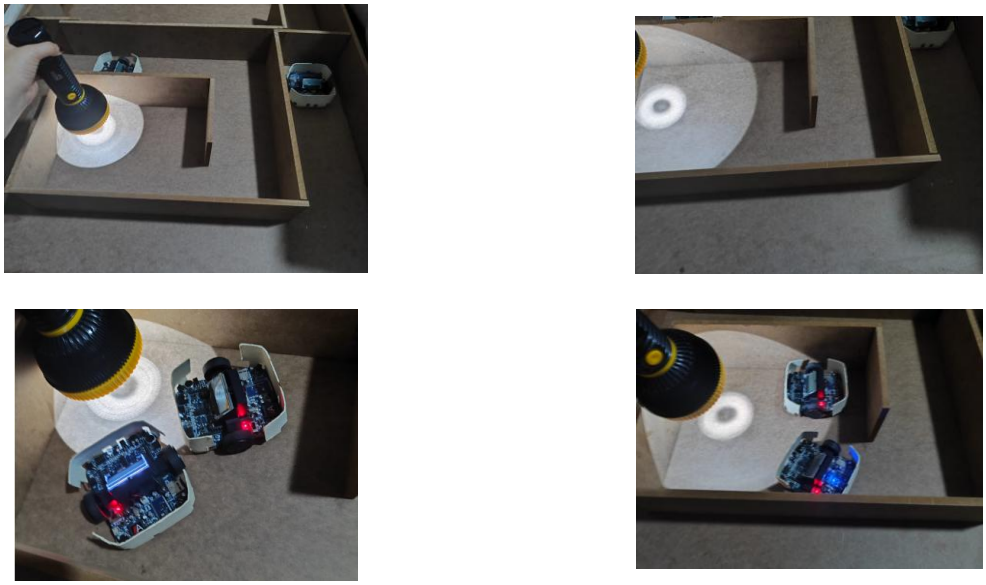
Durante las pruebas, los robots iniciaron desde posiciones fijas para garantizar la consistencia entre repeticiones. Estas posiciones iniciales, así como la ubicación de la fuente de luz (siempre en el centro de la arena), se mantuvieron constantes para todos los tamaños evaluados. Para cada configuración, se registraron dos métricas principales:

- Tiempo de agrupamiento: Tiempo en segundos que tardaron ambos robots en llegar y permanecer bajo la fuente de luz.
- Distancia entre robots: Distancia final en milímetros entre ambos robots una vez agrupados.

La Figura 21 ilustra tanto las posiciones iniciales de los robots y la fuente de luz, así como ejemplos de las posiciones finales alcanzadas para cada tamaño.

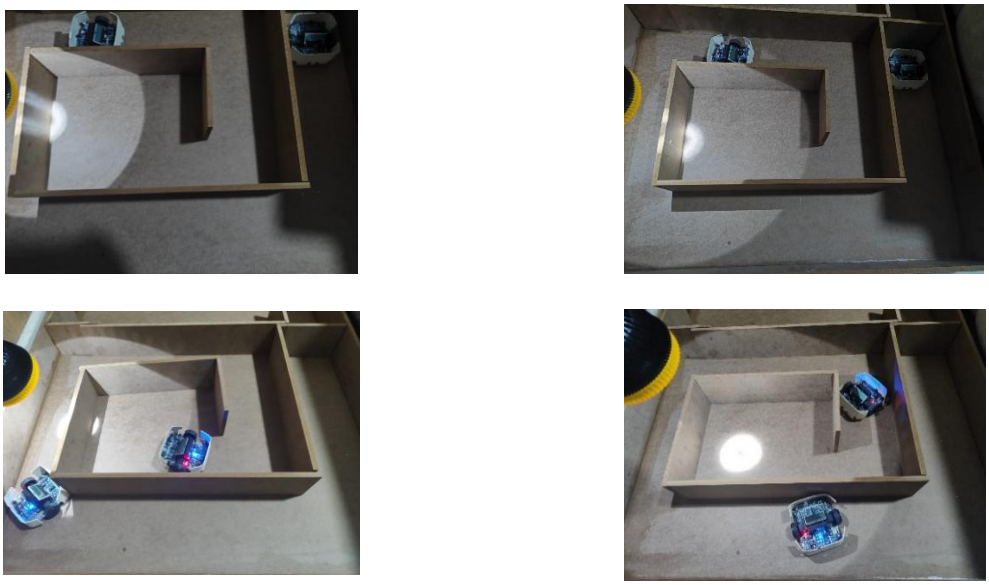
Figura 21

Distribución Inicial y Final de la Prueba del Tamaño de la Fuente de Luz



a) Fuente de tamaño 1/8

b) Fuente de tamaño 1/4



c) Fuente de tamaño 3/8

d) Fuente de tamaño 1/2

Esta figura permite observar cómo las variaciones en el tamaño de la fuente afectan la posición final de los robots, reflejando tanto la rapidez del agrupamiento como su dispersión.

4. Resultados

La presente sección describe los resultados obtenidos durante las pruebas experimentales realizadas para validar el comportamiento del sistema de agrupamiento en el enjambre de robots.

4.1 Tiempo de espera basado en la luminosidad.

4.1.1. Verificación Experimental del Tiempo de Espera

En primer lugar, se presentan los resultados relacionados con la validación del tiempo de espera calculado por la ecuación ajustada del algoritmo. a Tabla 5 presenta los valores medidos, incluyendo la intensidad de luz, los tiempos calculados y reales, y los errores porcentuales asociados.

Tabla 5
Verificación Experimental del Tiempo de Espera

Intensidad de Luz	Tiempo Calculado (s)	Tiempo Real (s)	Diferencia (s)	% Error
1489	29,6964	29,6969	0,0005	0,0017%
1475	29,6877	29,6894	0,0017	0,0057%
4001	29,9755	29,9765	0,0010	0,0033%
3977	29,9752	29,9768	0,0016	0,0053%
524	1,6339	1,6354	0,0015	0,0918%
1847	29,8356	29,8365	0,0009	0,0030%
3951	29,9748	29,9761	0,0013	0,0043%
1215	29,4244	29,4256	0,0012	0,0041%
1752	29,8098	29,8108	0,0010	0,0034%
829	27,4628	27,4635	0,0007	0,0025%
734	25,3672	25,3684	0,0012	0,0047%
905	28,2761	28,2766	0,0005	0,0018%
847	27,6995	27,7004	0,0009	0,0032%
3176	29,9582	29,9595	0,0013	0,0043%
937	28,5072	28,5087	0,0015	0,0053%

Las diferencias promedio entre los tiempos calculados y reales son mínimas, con errores porcentuales inferiores al 0,01%. Esto indica que el algoritmo cumple con los valores teóricos en condiciones controladas. Los casos de mayor error relativo, como el registrado en s=524, reflejan que hay mayor sensibilidad en valores cercanos al umbral mínimo.

4.1.2. Impacto del Parámetro θ en el Comportamiento del Sistema.

Los resultados de las pruebas se presentan en la Tabla 6. Adicionalmente, las Figuras 22 y 23 permiten visualizar el impacto de θ con mayor detalle, evidenciando cómo el parámetro θ afecta la estabilidad y sensibilidad del sistema.

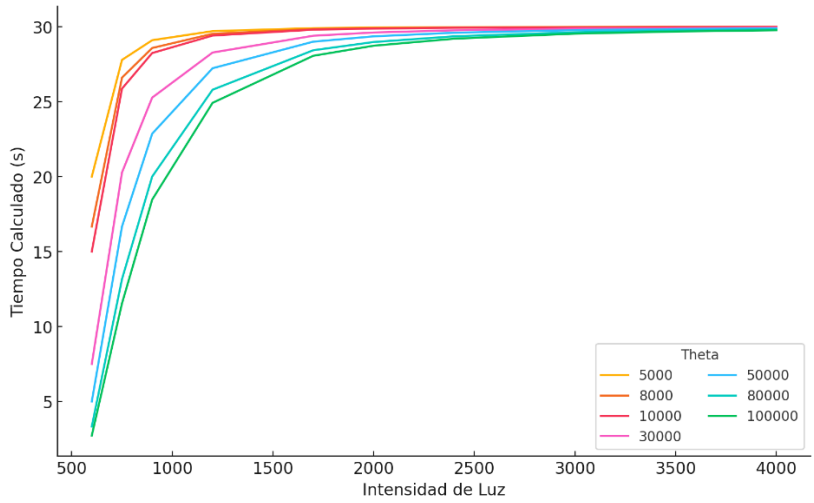
Tabla 6

Impacto del Parámetro θ en el Comportamiento del Sistema

Intensidad de Luz	Tiempo Calculado							
	θ	5000	8000	10000	30000	50000	80000	100000
600		20	16,66666667	15	7,5	5	3,33333333	2,72727273
750		27,77777778	26,59574468	25,862069	20,2702703	16,6666667	13,1578947	11,5384615
900		29,09090909	28,57142857	28,2352941	25,2631579	22,8571429	20	18,4615385
1200		29,6969697	29,51807229	29,4	28,2692308	27,2222222	25,7894737	24,9152542
1700		29,89619377	29,83425414	29,7931034	29,3877551	28,9932886	28,4210526	28,0519481
2000		29,93348115	29,89371125	29,8672566	29,6052632	29,3478261	28,9699571	28,7234043
2400		29,95850622	29,93366501	29,9171271	29,7527473	29,5901639	29,3495935	29,1913747
3000		29,97601918	29,96164909	29,9520767	29,8566879	29,7619048	29,6208531	29,5275591
3500		29,98334259	29,97335702	29,9667037	29,9003322	29,8342541	29,7356828	29,6703297
4000		29,9877601	29,98042095	29,9755302	29,9267101	29,8780488	29,8053528	29,757085

Figura 22

Relación entre Intensidad de Luz y Tiempo Calculado para diferentes valores de θ

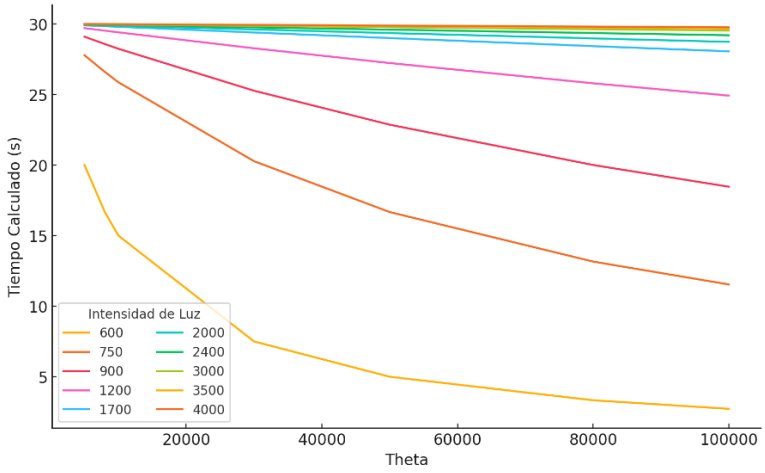


La Figura 22 muestra que al tener valores bajos de θ, las curvas presentan una pendiente más pronunciada, por lo cual el tiempo calculado tiende a acercarse al máximo permitido, en este caso 30 segundos. Esto genera baja variabilidad en los tiempos de espera, limitando la sensibilidad del sistema.

Por otro lado, al aumentar los valores de θ, las curvas se suavizan, implicando mayor variabilidad en los tiempos calculados, lo que permite respuestas más variadas a intensidades de luz intermedias y bajas.

Figura 23

Sensibilidad del sistema frente a θ



Por su parte, la Figura 23, confirma el comportamiento descrito anteriormente. Se resalta que, en intensidades lumínicas bajas, las líneas fluctúan más y que para θ bajos, los tiempos calculados tienden al máximo permitido, generando una respuesta homogénea del sistema. Así como, para intensidades lumínicas altas, las líneas son prácticamente horizontales, es decir, tiempos más estables.

4.2 Precisión en la Detección de Robots.

La evaluación de la detección incluyó robots y obstáculos en diferentes escenarios, orientaciones y distancias. Los resultados se presentan en la Tabla 7, que resume la precisión por tipo de objeto y el desempeño global del sistema.

Tabla 7

Resultados de la Prueba de Detección

Detección	Correcta	%Detecciones Correctas	Incorrecta	%Detecciones Incorrectas	Total de Pruebas
Robots	58	96,67%	2	3,33%	60
Obstáculo	39	65,00%	21	35,00%	60
Total de Detecciones	97	80,83%	23	19,17%	120

Igualmente muestra la precisión de detección por escenario. Para la detección de robots, se obtuvo un 96.67% de detección correcta y un 3.33% de error. Este resultado confirma la capacidad del sistema para reconocer robots, algo esencial para la coordinación de un enjambre.

Sin embargo, en la detección de obstáculos, la precisión disminuyó a un 65,00% de aciertos, con un 35,00% de errores. Esto sugiere limitaciones en la capacidad del sistema para distinguir entre robots y objetos estáticos. En la mayoría de los casos, los robots son identificados correctamente, independientemente de la distancia o la orientación. Los errores en la detección de obstáculos se concentran principalmente a distancias más cortas, indicativo de interferencias de los sensores o limitaciones en el alcance efectivo del sensor.

Las gráficas asociadas (Figura 24 y 25) reflejan las detecciones correctas e incorrectas en número y porcentaje, así como la precisión global del sistema.

Figura 24

Pruebas de Detección: Robots y Obstáculos Correctos e Incorrectos, en Número y Porcentaje.

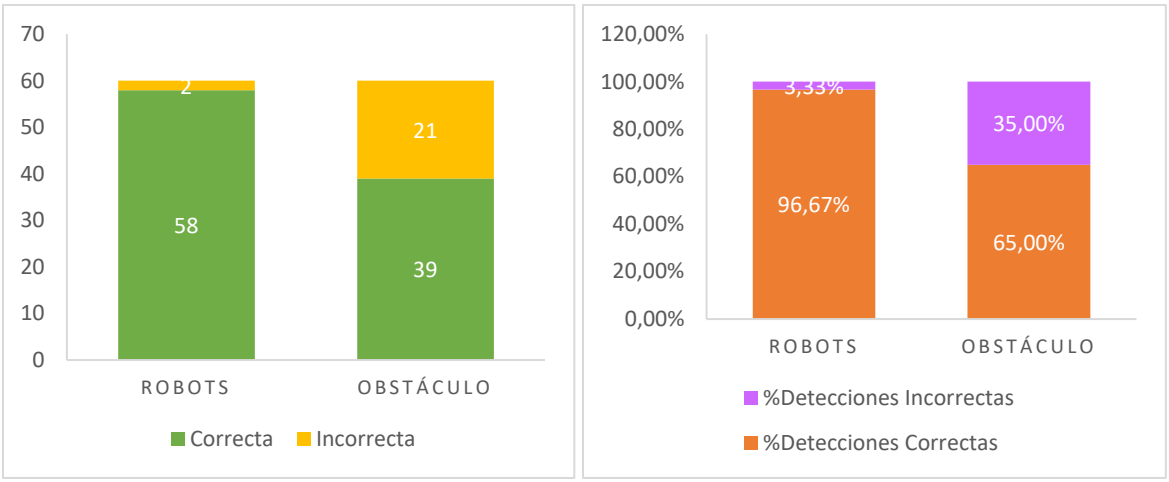
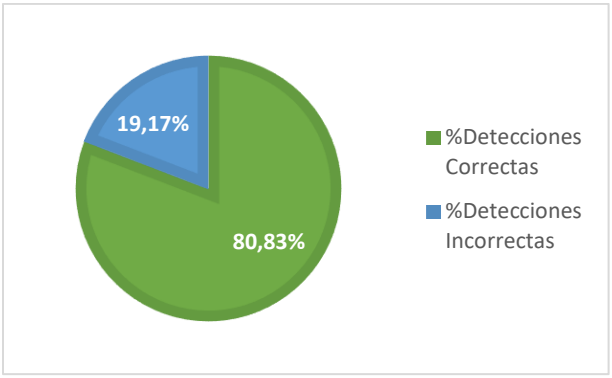


Figura 25

Porcentaje de la Precisión Global del Sistema de Detección



Se observa que el sistema obtuvo una precisión global del 80,83%, que es adecuado para los objetivos del proyecto. Esta precisión en la detección de robots podría deberse a la capacidad del sistema para procesar correctamente las señales emitidas por los sensores, debido a las características del diseño del enjambre. No obstante, los resultados muestran dificultades para identificar obstáculos, posiblemente causadas por interferencias ambientales, ruido de los sensores o la configuración del entorno experimental.

4.3 Respuesta a Variables Externas

4.3.1 Tamaño del Área de Trabajo

La Tabla 8 muestra los tiempos promedio, desviaciones estándar y coeficientes de variación (CV) obtenidos para cada tamaño de área evaluado. Las gráficas asociadas ilustran el comportamiento del sistema.

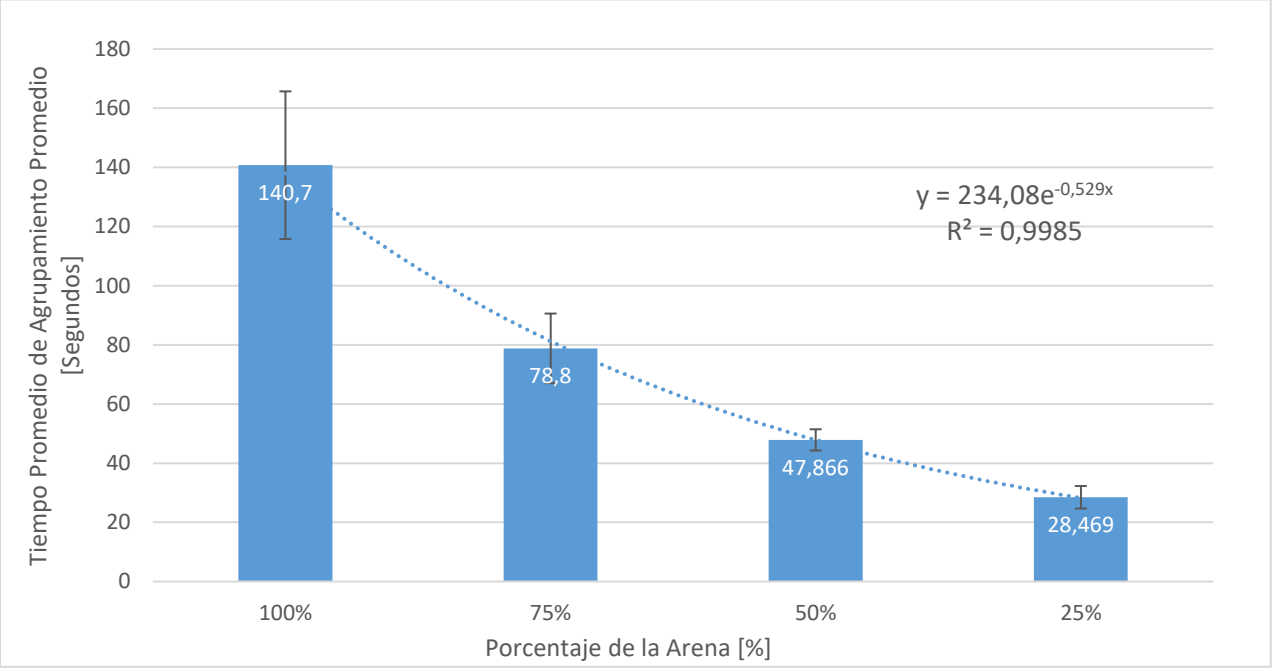
Tabla 8

Resultados del Análisis de Tamaño del Área

Tamaño del Área (%)	Tiempo Promedio [Segundos]	Desviación Estándar [Segundos]	Coeficiente de Variación CV (%)
100%	140,7	24,96017	17,7%
75%	78,8	11,76265	14,9%
50%	47,866	3,590700	7,5%
25%	28,469	3,800841	13,4%

Figura 26

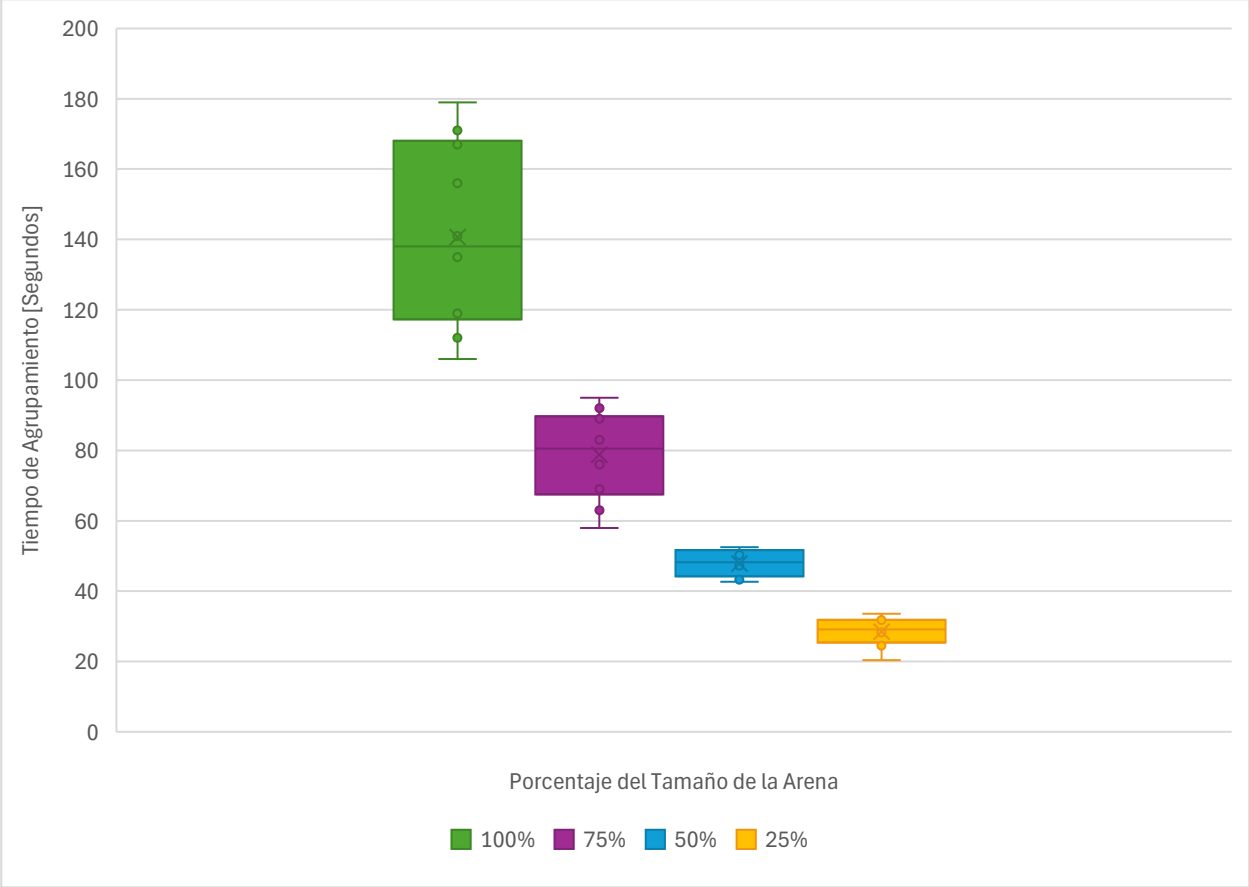
Relación entre el Tamaño del Área y el Tiempo Promedio



La Figura 26 muestra cómo el tiempo promedio disminuye de forma exponencial, como indica la ecuación, a medida que se reduce el tamaño del área, reflejando que los robots recorren distancias más cortas en áreas pequeñas, lo que hace que sea más rápido el agrupamiento. Sin embargo, esto no necesariamente garantiza una mayor consistencia en todos los casos.

Figura 27

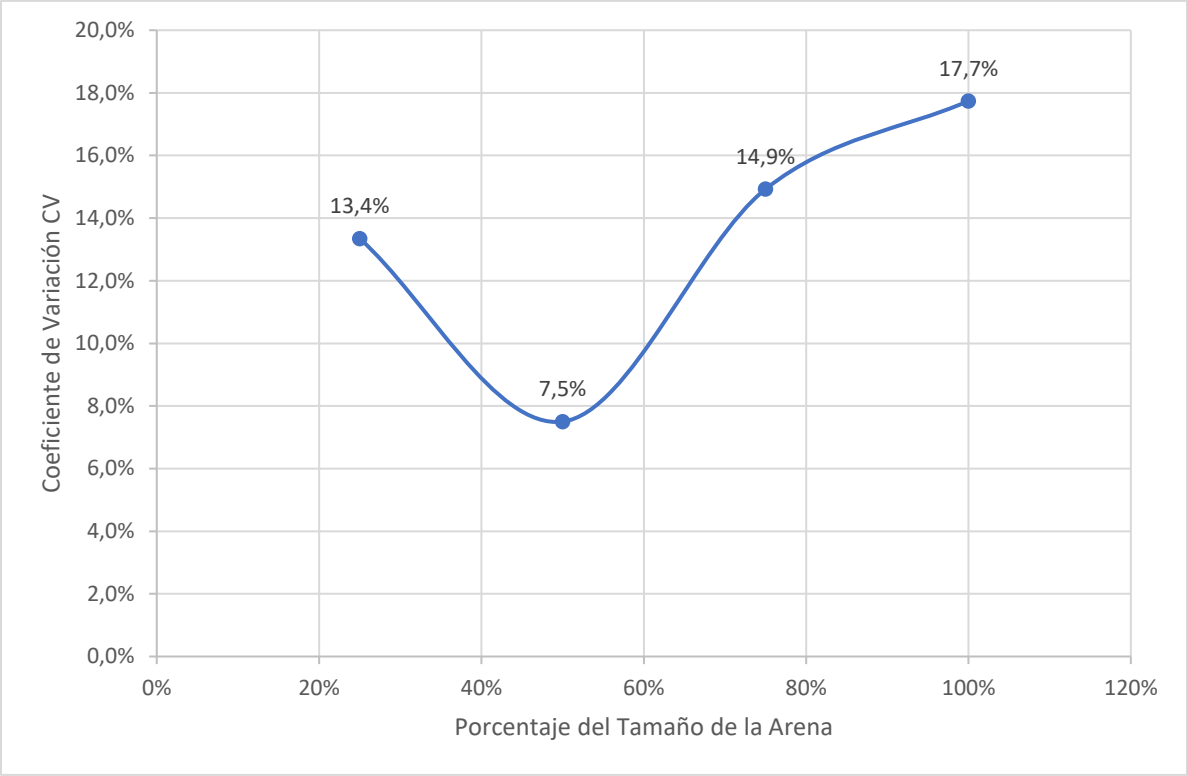
Distribución de los Tiempos por Tamaño del Área



En la Figura 27, se revela que en áreas grandes (100 % y 75 %), los tiempos tienen una mayor dispersión, con rangos intercuartiles amplios y valores atípicos que reflejan mayor variabilidad en el desempeño. En áreas más pequeñas, como 50 % y 25 %, la distribución es más compacta, indicando una mayor estabilidad en los tiempos registrados.

Figura 28

Relación entre el Tamaño del Área y el Coeficiente de Variación (CV)



Por otro lado, el análisis del coeficiente de variación (CV), mostrado en la Figura 28, sugiere que el tamaño del área influye en la variabilidad relativa. El área del 50 % es donde el sistema muestra el mejor desempeño en términos de consistencia, con el CV más bajo (7,5 %). Por el contrario, áreas grandes (100 %) y pequeñas (25 %) presentan mayor variabilidad relativa, lo que podría indicar que, aunque los tiempos sean más bajos en áreas pequeñas, dinámicas internas pueden influir en la estabilidad del sistema.

4.3.2 Posición de la Fuente de Luz dentro de la Arena

La Tabla 9 presenta los valores obtenidos para cada posición evaluada en las métricas establecidas.

Tabla 9

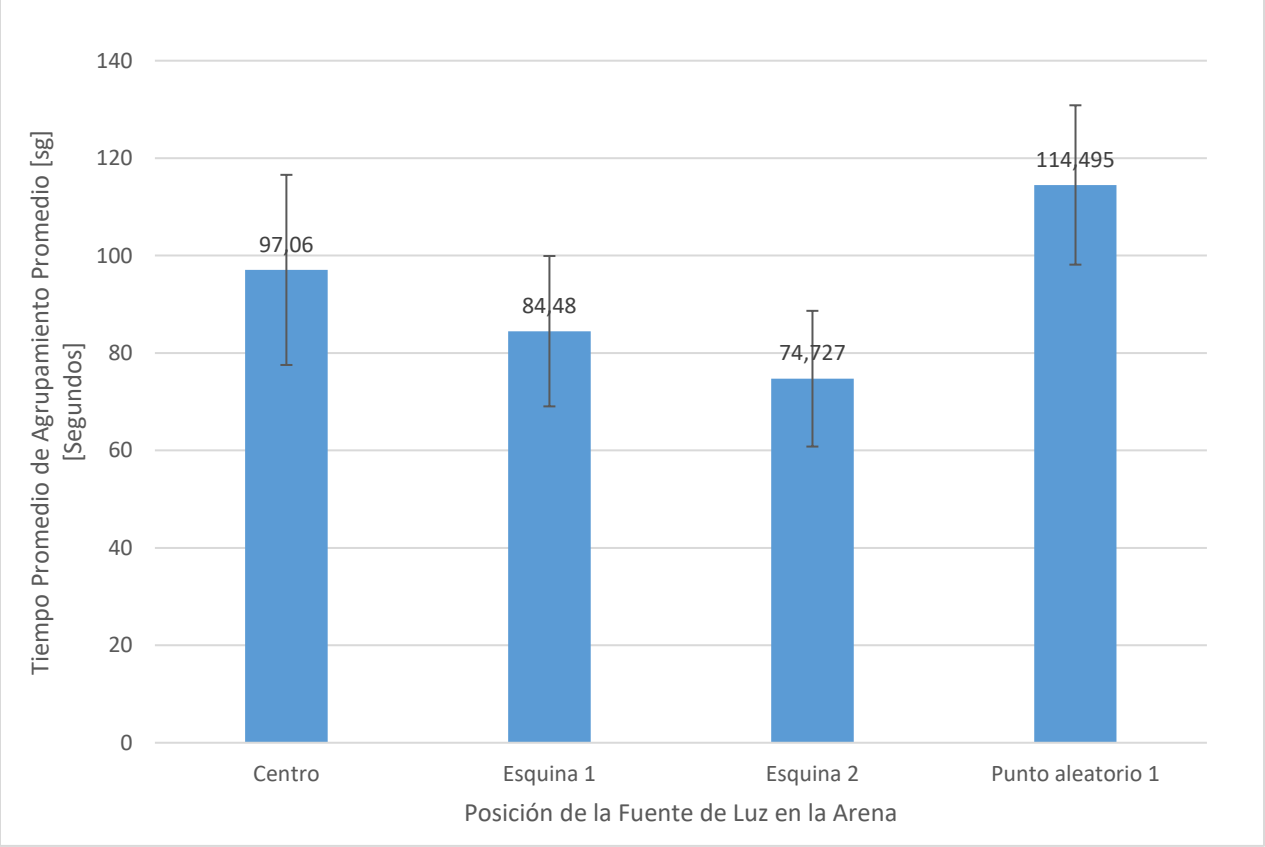
Resultados de la Posición de la Fuente de Luz en la Arena

Posición	Tiempo Promedio [Segundos]	Desviación Estándar [Segundos]	Coeficiente de Variación CV
Centro	97,06	19,51228	20,1%
Esquina 1	84,48	15,43427	18,3%
Esquina 2	74,727	13,929307	18,6%
Punto aleatorio 1	114,495	16,355666	14,3%

Las gráficas destacan los tiempos promedio y la dispersión de los resultados:

Figura 29

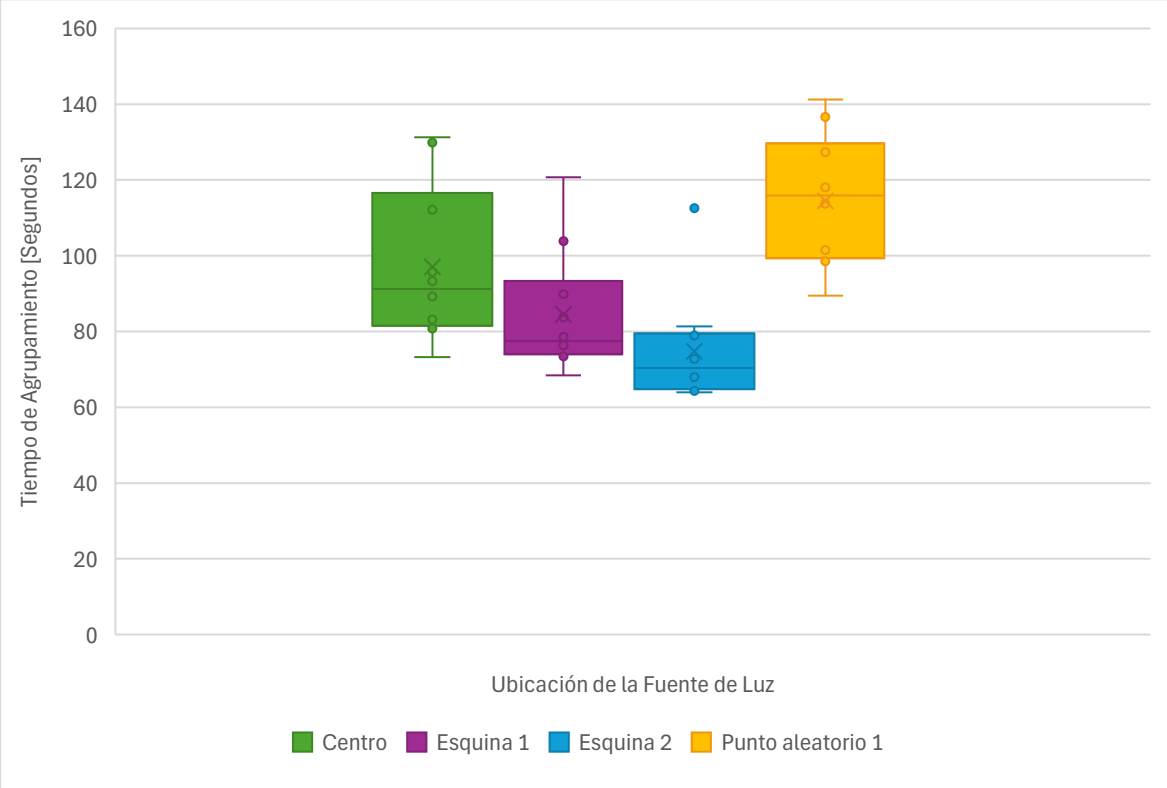
Tiempo Promedio de Agrupamiento por Posición de la Fuente de Luz



La Figura 29 muestra una representación comparativa del tiempo promedio para cada posición, acompañada de barras de error que reflejan la desviación estándar. En esta visualización, se observa que las posiciones en las esquinas (Esquina 1 y Esquina 2) presentan tiempos promedios más bajos y menor dispersión en comparación con el centro o el punto aleatorio.

Figura 30

Distribución de los Tiempos de Agrupamiento por Posición de la Fuente de Luz



La Figura 30 refuerza el análisis mostrando que las esquinas no solo facilitan tiempos más bajos, sino también una mayor consistencia en los resultados. Este comportamiento puede atribuirse a la literatura, que sugiere que las esquinas actúan como áreas naturales de confinamiento, limitando el rango de movimiento de los robots y promoviendo una convergencia más rápida hacia la fuente de luz.

Por el otro lado, el punto aleatorio presenta el mayor tiempo promedio de agrupamiento (114,50 s), aunque su coeficiente de variación (14,3 %) es el más bajo, indicando que, aunque los robots logran agruparse, el proceso es más lento debido a la dispersión inicial respecto a la fuente de luz. Por otro lado, el centro de la arena muestra un comportamiento intermedio, con un tiempo promedio de 97,06 s y la mayor desviación estándar (19,51 s). Esto sugiere que, aunque los robots logran agruparse, las trayectorias pueden ser menos predecibles debido a la simetría del espacio abierto.

4.3.3 Tamaño de la Fuente de Luz

La Tabla 10 muestra los valores obtenidos para el tiempo de agrupamiento y la distancia entre robots, junto con las métricas de desviación estándar y CV.

Tabla 10

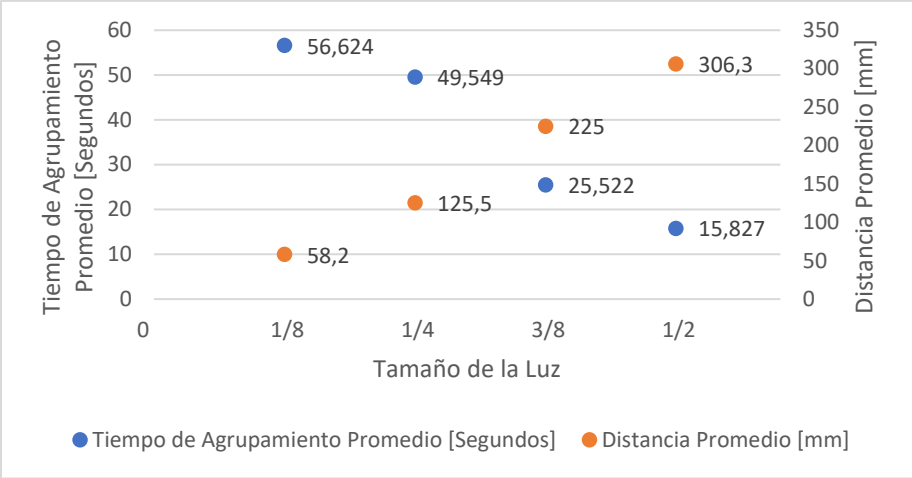
Resultados del Análisis del Tamaño de la Fuente de Luz

Tamaño de la Luz	Tiempo [Segundos]			Distancia [mm]		
	Promedio	Desviación Estándar	Coeficiente de Variación CV	Promedio	Desviación Estándar	Coeficiente de Variación CV
1/2	15,83	1,498132504	9,5%	306,30	52,87920196	17,3%
3/8	25,52	5,680404563	22,3%	225,00	37,74917218	16,8%
1/4	49,55	6,397534603	12,9%	125,50	22,29910312	17,8%
1/8	56,62	9,175083869	16,2%	58,20	6,493073232	11,2%

Los valores obtenidos destacan un comportamiento inversamente proporcional entre el tamaño de la fuente de luz y la cohesión del agrupamiento. Mientras que fuentes más grandes permiten tiempos de agrupamiento más cortos, la distancia promedio entre robots aumenta, lo que indica una agrupación más dispersa. Esto puede observarse claramente en las siguientes gráficas.

Figura 31

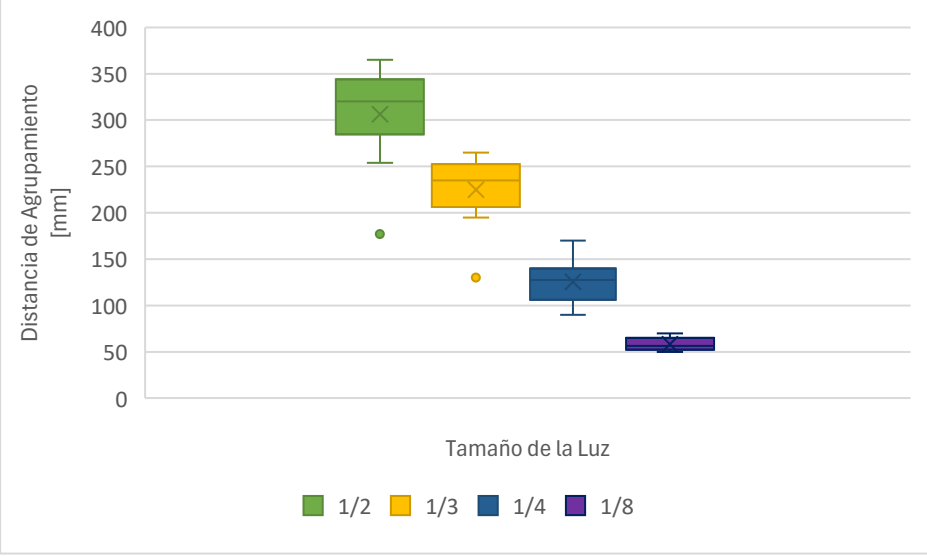
Relación entre el tamaño de la luz y el tiempo/distancia promedio



En la Figura 31, se evidencia que fuentes más grandes (1/2) permiten tiempos de agrupamiento más rápidos, pero generan distancias promedio mayores entre robots, indicando una agrupación más dispersa. Por el contrario, fuentes más pequeñas (1/8) resultan en tiempos más prolongados, pero con agrupaciones más compactas.

Figura 32

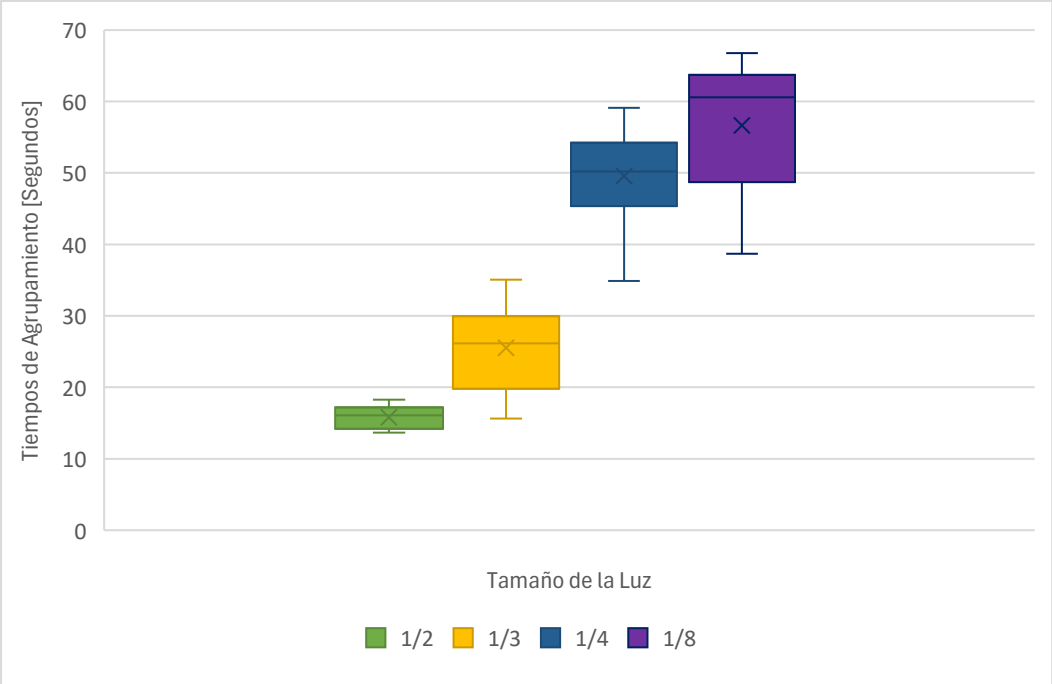
Distribución de las distancias entre robots



La Figura 32 refuerza el impacto del tamaño de la luz en la cohesión del agrupamiento. Las fuentes pequeñas generan agrupaciones más consistentes, reflejadas en una menor dispersión de las distancias. En contraste, las fuentes grandes presentan mayor variabilidad, lo que compromete la cohesión del sistema.

Figura 33

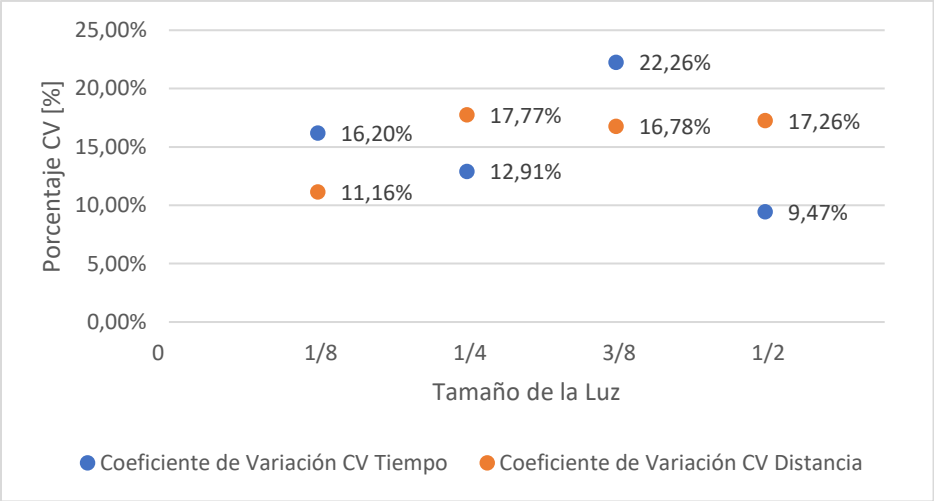
Distribución de los tiempos de agrupamiento



La Figura 33 muestra que los tiempos de agrupamiento son más rápidos y consistentes con fuentes grandes (1/2), pero presentan mayor dispersión con fuentes pequeñas (1/8). Esto refleja un equilibrio entre rapidez y precisión en la interacción del sistema.

Figura 34

Coeficiente de variación (CV) para Tamaño de Fuente de Luz Variable



En términos de variabilidad, la Figura 34 muestra el coeficiente de variación (CV) para el tiempo y la distancia promedio. El CV para las distancias es más bajo en fuentes pequeñas, lo que confirma su consistencia. Sin embargo, para tiempos, el CV no es consistente y presenta fluctuaciones, aunque su mayor punto se encuentra en 3/8.

5. Discusión de Resultados

La relación entre la intensidad lumínica y los tiempos de espera permitió observar que, cerca del umbral mínimo, los robots se desplazaron rápidamente hacia las áreas iluminadas. Esto demuestra la capacidad del sistema para adaptarse a estímulos débiles y actuar con rapidez. Por otro lado, intensidades mayores prolongaron los tiempos de espera, favoreciendo la cohesión en la agrupación en torno a la fuente de luz. Este comportamiento refuerza el potencial del modelo para gestionar escenarios donde la precisión en la localización es prioritaria sobre la velocidad.

En cuanto a la detección de robots y obstáculos, el sistema logró una precisión del 96.67 % al identificar otros robots, lo que evidencia una colaboración confiable entre dispositivos. Sin embargo, al enfrentar obstáculos estáticos, la precisión fue limitada (65 %). Esto sugiere que la integración de sensores adicionales o algoritmos complementarios podría ser una vía efectiva para mejorar el desempeño en entornos logísticos complejos, donde los elementos estáticos y móviles suelen interactuar.

El análisis del tamaño del área de trabajo reveló que reducirla al 50 % optimizó la estabilidad del sistema, logrando tiempos promedio de agrupamiento de 47.87 segundos con baja variabilidad (CV de 7.5 %). Por el contrario, áreas muy grandes o pequeñas introdujeron dispersión en los resultados, lo que indica que el diseño del entorno debe ajustarse cuidadosamente según las necesidades específicas de la aplicación.

En las pruebas relacionadas con el tamaño de la fuente de luz, se observó que fuentes grandes (1/2 de la arena) redujeron significativamente los tiempos de agrupamiento, con un promedio de 15.83 segundos. Sin embargo, estas configuraciones generaron agrupaciones dispersas (306.30 mm de distancia promedio entre robots). Por el contrario, fuentes más pequeñas (1/8 de la arena) promovieron una mayor cohesión en las agrupaciones (58.20 mm de distancia promedio), aunque los tiempos aumentaron a 56.62 segundos. Esto resalta la importancia de equilibrar rapidez y cohesión en función de los objetivos del sistema.

Finalmente, la posición de la fuente de luz también tuvo un impacto significativo en el desempeño del sistema. Las ubicaciones en esquinas facilitaron tiempos de agrupamiento más bajos y consistentes (74.73 a 84.48 segundos, con CV del 18.3 % y 18.6 %, respectivamente). En contraste, posiciones en el centro o en puntos aleatorios incrementaron los tiempos promedio y la dispersión de trayectorias, con 97.06 y 114.49 segundos, respectivamente. Este comportamiento

puede atribuirse al confinamiento natural de las esquinas en el laberinto, lo que facilita la detección y el agrupamiento de los robots.

Este análisis general confirma que las condiciones del entorno, como la intensidad lumínica, el tamaño y la posición de la fuente de luz, y las dimensiones del área de trabajo, influyen significativamente en el desempeño del sistema, subrayando la importancia de ajustar estos parámetros según los requerimientos específicos de la aplicación logística.

6. Conclusiones

- La caracterización del área de trabajo, compuesta por la arena, la fuente de luz y los robots *Formula AllCode*, permitió diseñar un entorno adecuado para implementar y validar el algoritmo Beeclust. El análisis del rango operativo de los sensores infrarrojos estableció un rango efectivo de 20 mm, que fue usado para dimensionar el espacio de trabajo y garantizar lecturas consistentes y precisas, fundamentales para el correcto desempeño del sistema.
- El diseño del laberinto, inspirado en entornos logísticos, proporcionó un contexto experimental controlado que facilitó la navegación y la detección. Este enfoque replicó condiciones similares a las de un almacén, superando las limitaciones asociadas a entornos abiertos y mejorando la evaluación del sistema en condiciones prácticas.
- El diseño del algoritmo de agrupamiento inspirado en Beeclust sentó las bases para un sistema colaborativo capaz de localizar inventarios perdidos en entornos controlados. A través de la incorporación del estado WAIT, los robots lograron agruparse en puntos clave, como la fuente de luz, replicando comportamientos de enjambre observados en la naturaleza y demostrando el potencial del sistema para organizar la búsqueda de inventarios.
- La implementación del algoritmo fue adaptada a las capacidades del hardware disponible, utilizando componentes integrados como el micrófono y la bocina para la detección entre robots. Esto no solo permitió implementar una solución funcional dentro de las limitaciones de esta etapa inicial, sino también explorar la viabilidad

de aplicar estrategias de enjambre a problemas más avanzados en logística en el contexto industrial.

- Aunque el prototipo fue probado con dos robots, su diseño es escalable, lo que permite extender su funcionalidad a enjambres más grandes. Esto refuerza el potencial de los sistemas de enjambre como una herramienta versátil y escalable para sistemas más complejos y a mayor escala.
- Por medio de la validación experimental del sistema diseñado, fue posible analizar el desempeño en condiciones controladas y cómo el algoritmo responde a distintos escenarios. Los resultados evidenciaron la capacidad del sistema de replicar comportamientos de enjambre al variar la intensidad lumínica, la disposición espacial y el tamaño de la fuente de luz. Esto posiciona al algoritmo de agrupamiento como una herramienta prometedora para la localización de inventarios en entornos delimitados.
- Finalmente, el sistema diseñado mostró un desempeño sólido en la validación experimental y presenta una base funcional para la localización de inventarios en entornos controlados. Aunque este prototipo inicial se limitó a dos robots y un entorno controlado, los resultados respaldan su escalabilidad y adaptabilidad para futuros trabajos. Mejoras en sensores y algoritmos podrían ampliar su aplicabilidad, contribuyendo significativamente a la gestión moderna de inventarios al reducir tiempos y costos operativos

7. Recomendaciones

- El sistema de detección podría mejorarse con sensores adicionales, como cámaras o ultrasonidos, para diferenciar con mayor precisión entre obstáculos estáticos y robots. Además, técnicas de fusión de sensores podrían incrementar la confiabilidad en entornos complejos. Pruebas en condiciones variables permitirían ajustar el sistema a escenarios más exigentes.
- Se recomienda explorar sistemas de control más avanzados para mejorar la estabilidad y precisión del desplazamiento. También cabe la posibilidad de explorar la viabilidad y capacidad de implementar interrupciones a nivel de hardware, en lugar de depender exclusivamente de bibliotecas de software como *asyncio* y *threading*. Aunque esto implique mayor esfuerzo técnico, podría optimizar las respuestas del sistema.
- La escalabilidad del sistema debe validarse probando con enjambres más grandes. Esto permitirá identificar desafíos como congestión, interferencias y consumo energético, además de ajustar los parámetros del algoritmo para mantener la consistencia del comportamiento en escenarios más complejos y cercanos a aplicaciones reales.
- Para aplicaciones en almacenes logísticos, sería útil incorporar tecnologías como etiquetas RFID o códigos QR para identificar inventarios con mayor precisión. Probar el sistema en entornos reales, con configuraciones logísticas diversas, fortalecería su aplicabilidad y utilidad en operaciones cotidianas.

Referencias Bibliográficas

Acevedo, O., Rios Diaz, Y., Duque, J., Gomez, E., & Garcia, L. (2022). *A Software for Simulating Robot Swarm Aggregation* (pp. 386–399). https://doi.org/10.1007/978-3-031-20611-5_32

Acevedo, O., Rios Diaz, Y., Garcia, L., & Narvaez, D. (2022). *A study of the Beeclust algorithm for robot swarm aggregation*. 1–6. https://doi.org/10.1109/ICMLANT56191.2022.9996514

Alfredo, D., & Ramos, G. (2022). *Empirical comparison of methods for the design of robot swarms: a simulation study of swarms that coordinate other swarms*.

Arvin, F., Turgut, A. E., Bazyari, F., Arikan, K. B., Bellotto, N., & Yue, S. (2014). Cue-based aggregation with a mobile robot swarm: A novel fuzzy-based method. *Adaptive Behavior*, *22*(3), 189–206. https://doi.org/10.1177/1059712314528009

Åström, K. J., & Murray, R. M. (2019). Chapter Ten. PID Control. En *Feedback Systems* (pp. 293–314). Princeton University Press. https://doi.org/10.1515/9781400828739-011

Bayindir, L. (2016). A review of swarm robotics tasks. *Neurocomputing*, *172*, 292–321. https://doi.org/10.1016/j.neucom.2015.05.116

Bodi, M., Thenius, R., Szopek, M., Schmickl, T., & Crailsheim, K. (2012). Interaction of robot swarms using the honeybee-inspired control algorithm BEECLUST. *Mathematical and Computer Modelling of Dynamical Systems*, *18*(1), 87–100. https://doi.org/10.1080/13873954.2011.601420

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41. https://doi.org/10.1007/s11721-012-0075-2

Casamayor-Pujol, V., Morenza-Cinos, M., Gastón, B., & Pous, R. (2020). Autonomous stock counting based on a stigmergic algorithm for multi-robot systems. *Computers in Industry*, *122*. https://doi.org/10.1016/j.compind.2020.103259

Hamann, H., Wörn, H., Crailsheim, K., & Schmickl, T. (2008). Spatial macroscopic models of a bio-inspired robotic swarm algorithm. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. https://doi.org/10.1109/IROS.2008.4651038

Kernbach, S., Thenius, R., Kernbach, O., & Schmickl, T. (2009). Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system. *Adaptive Behavior*, *17*(3), 237–259. https://doi.org/10.1177/1059712309104966

Ogata, K. (2010). Ingenieria-de-Control-Moderna-Ogata-5ed. En *Pearson* (Vol. 5, Número 12).

Schmickl, T., Thenius, R., Moeslinger, C., Radspieler, G., Kernbach, S., Szymanski, M., & Crailsheim, K. (2009). Get in touch: Cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, *18*(1). https://doi.org/10.1007/s10458-008-9058-5

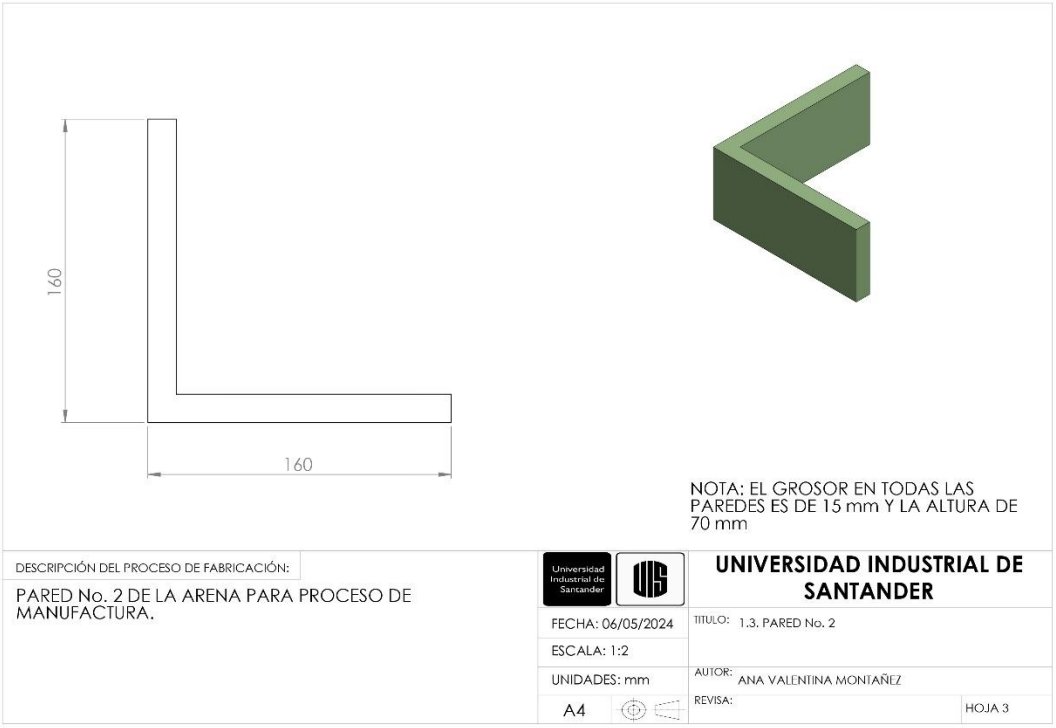
Sempere, M. (2013). *Agentes y enjambres artificiales: modelado y comportamientos para sistemas de enjambre robóticos*. http://hdl.handle.net/10045/36619

Vogrin, M., Stefanec, M., & Schmickl, T. (2020). Social Distancing in Robot Swarms: Modulating Exploitation and Exploration without Signal Exchange. *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*. https://doi.org/10.1109/SSCI47803.2020.9308502

Zayas Gato, F., Quintián Pardo, H., Jove Pérez, E., Casteleiro Roca, J. L., & Calvo Rolle, J. L. (2020). Diseño de controladores PID. En *Diseño de controladores PID*. https://doi.org/10.17979/spudc.9788497497855

Apéndices

Apéndice A. Planos Laberinto



Apéndice B. Resultados Prueba de Detección

Distancia [cm]	Orientación	Escenario	Resultado	Detectado	Resultado Deteccion
1	Frontal	Robot	1155,3	Si	Robot Correcto
1	Frontal	Robot	927	Si	Robot Correcto
1	Frontal	Obstáculo	872,3	No	Obstáculo Correcto
1	Frontal	Obstáculo	748,7	No	Obstáculo Correcto
1	Lateral Izquierdo	Robot	1155,1	Si	Robot Correcto
1	Lateral Izquierdo	Robot	1195,2	Si	Robot Correcto
1	Lateral Izquierdo	Obstáculo	900	No	Obstáculo Correcto
1	Lateral Izquierdo	Obstáculo	1103,6	Si	Obstáculo Incorrecto
1	Lateral Derecho	Robot	1316,8	Si	Robot Correcto
1	Lateral Derecho	Robot	1216,6	Si	Robot Correcto

1	Lateral Derecho	Obstáculo	707,2	No	Obstáculo Correcto
1	Lateral Derecho	Obstáculo	993,5	Si	Obstáculo Incorrecto
2	Frontal	Robot	1475,4	Si	Robot Correcto
2	Frontal	Robot	1184,9	Si	Robot Correcto
2	Frontal	Obstáculo	890	No	Obstáculo Correcto
2	Frontal	Obstáculo	954,7	Si	Obstáculo Incorrecto
2	Lateral Izquierdo	Robot	1007,3	Si	Robot Correcto
2	Lateral Izquierdo	Robot	1165,8	Si	Robot Correcto
2	Lateral Izquierdo	Obstáculo	997,4	Si	Obstáculo Incorrecto
2	Lateral Izquierdo	Obstáculo	831,5	No	Obstáculo Correcto
2	Lateral Derecho	Robot	1421,8	Si	Robot Correcto
2	Lateral Derecho	Robot	1240,5	Si	Robot Correcto
2	Lateral Derecho	Obstáculo	1107,7	Si	Obstáculo Incorrecto
2	Lateral Derecho	Obstáculo	831,5	No	Obstáculo Correcto
3	Frontal	Robot	942,9	Si	Robot Correcto
3	Frontal	Robot	1031,8	Si	Robot Correcto
3	Frontal	Obstáculo	1042,8	Si	Obstáculo Incorrecto
3	Frontal	Obstáculo	1036,8	Si	Obstáculo Incorrecto
3	Lateral Izquierdo	Robot	964,5	Si	Robot Correcto
3	Lateral Izquierdo	Robot	848,9	No	Robot Incorrecto
3	Lateral Izquierdo	Obstáculo	915,8	No	Obstáculo Correcto
3	Lateral Izquierdo	Obstáculo	811,3	No	Obstáculo Correcto
3	Lateral Derecho	Robot	1273,1	Si	Robot Correcto
3	Lateral Derecho	Robot	1462,5	Si	Robot Correcto
3	Lateral Derecho	Obstáculo	767,8	No	Obstáculo Correcto
3	Lateral Derecho	Obstáculo	750,1	No	Obstáculo Correcto
4	Frontal	Robot	1113,6	Si	Robot Correcto
4	Frontal	Robot	1213,6	Si	Robot Correcto
4	Frontal	Obstáculo	874,2	No	Obstáculo Correcto
4	Frontal	Obstáculo	968,1	Si	Obstáculo Incorrecto
4	Lateral Izquierdo	Robot	1308,9	Si	Robot Correcto
4	Lateral Izquierdo	Robot	1092	Si	Robot Correcto

4	Lateral Izquierdo	Obstáculo	1053,1	Si	Obstáculo Incorrecto
4	Lateral Izquierdo	Obstáculo	921,8	Si	Obstáculo Incorrecto
4	Lateral Derecho	Robot	1005,3	Si	Robot Correcto
4	Lateral Derecho	Robot	995,5	Si	Robot Correcto
4	Lateral Derecho	Obstáculo	850,6	No	Obstáculo Correcto
4	Lateral Derecho	Obstáculo	756,2	No	Obstáculo Correcto
5	Robot	Robot	1049,8	Si	Robot Correcto
5	Robot	Robot	917	No	Robot Incorrecto
5	Obstáculo	Obstáculo	1128,9	Si	Obstáculo Incorrecto
5	Obstáculo	Obstáculo	740,1	No	Obstáculo Correcto
5	Lateral Izquierdo	Robot	949,8	Si	Robot Correcto
5	Lateral Izquierdo	Robot	949,4	Si	Robot Correcto
5	Lateral Izquierdo	Obstáculo	695,3	No	Obstáculo Correcto
5	Lateral Izquierdo	Obstáculo	910	No	Obstáculo Correcto
5	Lateral Derecho	Robot	1048,9	Si	Robot Correcto
5	Lateral Derecho	Robot	1261,5	Si	Robot Correcto
5	Lateral Derecho	Obstáculo	1081,6	Si	Obstáculo Incorrecto
5	Lateral Derecho	Obstáculo	988	Si	Obstáculo Incorrecto
1	Frontal	Robot	1768	Si	Robot Correcto
1	Frontal	Robot	1528	Si	Robot Correcto
1	Frontal	Obstáculo	1007,1	Si	Obstáculo Incorrecto
1	Frontal	Obstáculo	891	No	Obstáculo Correcto
1	Lateral Izquierdo	Robot	952	Si	Robot Correcto
1	Lateral Izquierdo	Robot	1000	Si	Robot Correcto
1	Lateral Izquierdo	Obstáculo	889	No	Obstáculo Correcto
1	Lateral Izquierdo	Obstáculo	869	No	Obstáculo Correcto
1	Lateral Derecho	Robot	1350	Si	Robot Correcto
1	Lateral Derecho	Robot	967	Si	Robot Correcto
1	Lateral Derecho	Obstáculo	1207	Si	Obstáculo Incorrecto
1	Lateral Derecho	Obstáculo	755	No	Obstáculo Correcto
2	Frontal	Robot	966	Si	Robot Correcto
2	Frontal	Robot	941	Si	Robot Correcto

2	Frontal	Obstáculo	784	No	Obstáculo Correcto
2	Frontal	Obstáculo	568	No	Obstáculo Correcto
2	Lateral Izquierdo	Robot	995	Si	Robot Correcto
2	Lateral Izquierdo	Robot	943	Si	Robot Correcto
2	Lateral Izquierdo	Obstáculo	877	No	Obstáculo Correcto
2	Lateral Izquierdo	Obstáculo	814	No	Obstáculo Correcto
2	Lateral Derecho	Robot	955	Si	Robot Correcto
2	Lateral Derecho	Robot	1015	Si	Robot Correcto
2	Lateral Derecho	Obstáculo	927	Si	Obstáculo Incorrecto
2	Lateral Derecho	Obstáculo	911	No	Obstáculo Correcto
3	Frontal	Robot	1652	Si	Robot Correcto
3	Frontal	Robot	1040	Si	Robot Correcto
3	Frontal	Obstáculo	985	Si	Obstáculo Incorrecto
3	Frontal	Obstáculo	854	No	Obstáculo Correcto
3	Lateral Izquierdo	Robot	1664	Si	Robot Correcto
3	Lateral Izquierdo	Robot	1275	Si	Robot Correcto
3	Lateral Izquierdo	Obstáculo	915	No	Obstáculo Correcto
3	Lateral Izquierdo	Obstáculo	930	Si	Obstáculo Incorrecto
3	Lateral Derecho	Robot	1110	Si	Robot Correcto
3	Lateral Derecho	Robot	995	Si	Robot Correcto
3	Lateral Derecho	Obstáculo	925	Si	Obstáculo Incorrecto
3	Lateral Derecho	Obstáculo	890	No	Obstáculo Correcto
4	Frontal	Robot	1023	Si	Robot Correcto
4	Frontal	Robot	1035	Si	Robot Correcto
4	Frontal	Obstáculo	855	No	Obstáculo Correcto
4	Frontal	Obstáculo	864	No	Obstáculo Correcto
4	Lateral Izquierdo	Robot	966	Si	Robot Correcto
4	Lateral Izquierdo	Robot	973	Si	Robot Correcto
4	Lateral Izquierdo	Obstáculo	881	No	Obstáculo Correcto
4	Lateral Izquierdo	Obstáculo	871	No	Obstáculo Correcto
4	Lateral Derecho	Robot	1163	Si	Robot Correcto
4	Lateral Derecho	Robot	1419	Si	Robot Correcto

4	Lateral Derecho	Obstáculo	750	No	Obstáculo Correcto
4	Lateral Derecho	Obstáculo	906	No	Obstáculo Correcto
5	Robot	Robot	976	Si	Robot Correcto
5	Robot	Robot	1356	Si	Robot Correcto
5	Obstáculo	Obstáculo	630	No	Obstáculo Correcto
5	Obstáculo	Obstáculo	496	No	Obstáculo Correcto
5	Lateral Izquierdo	Robot	921	Si	Robot Correcto
5	Lateral Izquierdo	Robot	1030	Si	Robot Correcto
5	Lateral Izquierdo	Obstáculo	901	No	Obstáculo Correcto
5	Lateral Izquierdo	Obstáculo	1036	Si	Obstáculo Incorrecto
5	Lateral Derecho	Robot	1165	Si	Robot Correcto
5	Lateral Derecho	Robot	1005	Si	Robot Correcto
5	Lateral Derecho	Obstáculo	536	No	Obstáculo Correcto
5	Lateral Derecho	Obstáculo	1083	Si	Obstáculo Incorrecto

Apéndice C. Datos de la Prueba de Tamaño de la Arena

Tamaño	**Tiempo [segundos]**									
	1	2	3	4	5	6	7	8	9	10
100%	179	106	141	121	112	156	135	167	145	128
75%	69	89	76	83	58	95	85	63	78	92
50%	52,27	42,64	44,5	45,25	52,5	43,23	51,5	50,27	47,3	49,2
25%	20,34	31,78	25,63	30,23	28,24	24,5	29,52	32,15	28,76	33,54

Apéndice D. Datos de la Prueba de Posición de la Fuente de Luz en la Arena

Posición	**Tiempo [segundos]**									
Centro	73,27	80,84	93,32	83,17	95,71	131,24	112,2	89,21	129,89	81,78
Esquina 1	120,74	74,14	78,62	76,35	89,81	68,46	83,75	75,6	103,84	73,49
Esquina 2	67,93	74,75	64,37	81,34	63,93	72,74	112,5	65,87	78,91	64,89
Punto aleatorio 1	89,45	101,48	118,85	141,23	113,74	98,53	127,4	118,05	136,64	99,63

Apéndice E. Datos de la Prueba del Tamaño de la Fuente de Luz

1/2		1/3		1/4		1/8	
Tiempo [Segundos]	Distancia entre robots [mm]	Tiempo [Segundos]	Distancia entre robots [mm]	Tiempo [Segundos]	Distancia entre robots [mm]	Tiempo [Segundos]	Distancia entre robots [mm]
14,27	254	28,53	130	55,47	110	66,75	70
16,22	177	24,68	240	59,1	130	42,75	55
17,13	300	35,04	260	49,6	95	50,7	61
16,53	350	20,3	245	50,48	120	62,62	65
13,95	342	15,62	265	45,68	140	38,68	58
15,94	295	25,47	230	53,83	125	63,21	50
18,26	310	29,75	250	49,87	90	55,15	53
14,84	365	18,36	210	34,89	140	59,35	65
17,45	330	30,63	195	44,37	170	61,81	50
13,68	340	26,84	225	52,2	135	65,22	55

Apéndice F. Código de Funciones del Robot Formula AllCode

```
"""Formula AllCode Robot Buggy functions"""


import time
import serial as _serial
import sys as _sys
from sys import platform as _platform


if _platform == "win32":
    if (_sys.version_info.major == 2):
        import _winreg as _winreg
    else:
        import winreg as _winreg


class Create:
    __ser = _serial.Serial()
    __verbose = 0

    def __init__(self):
        self.__ser.close()
        return;

    def ComOpen(self, port):
        """Open a communication link to the robot

        Args:
            port: The COM port to open
        """

        if _platform == "linux" or _platform == "linux2":
            # linux
            s = '/dev/rfcomm{0}'.format(port)
        elif _platform == "darwin":
```

```
            # MAC OS X
            print("Error - MAC OS TODO")
        elif _platform == "win32":
            # Windows
            s = '\\\\.\\COM{0}'.format(port)
        else:
            print("Error - unsupported platform")

        self.__ser = _serial.Serial(port=s,\
                            baudrate=115200,\
                            parity=_serial.PARITY_NONE,\
                            stopbits=_serial.STOPBITS_ONE,\
                            bytesize=_serial.EIGHTBITS,\
                            timeout=1)
        #TODO: add checking to ensure port is open
        return;

    def ComClose(self):
        """Close the communication link to the robot
        """
        self.__ser.close()
        return;

##    def _comlist(self):
##        #TODO: this is Windows-only at the moment
##        if _platform == "win32":
##            key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE,
'HARDWARE\\DEVICEMAP\\SERIALCOMM')
##            i = 0
##            while True:
##                try:
##                    name = _winreg.EnumValue(key, i)[1][3:]
##                except OSError:
##                    #no more COM ports
##                    break
##                yield name #, '\\\\.\\{0}'.format(name)
##                i += 1
##
##            _winreg.CloseKey(key)

##    def ComQuery(self, port):
##        #TODO
##        comlist = self._comlist()
##        print (comlist)
##        return;
##
##    def ComFindFirst(self):
##        #TODO
##        return;

##    def Test(self):
```

```
##        #TODO: this is Windows-only at the moment
##        key = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE,
'HARDWARE\\DEVICEMAP\\SERIALCOMM')
##        i = 0
##        while True:
##            try:
##                name = _winreg.EnumValue(key, i)[1][3:]
##            except OSError:
##                #no more COM ports
##                break
##            yield name #, '\\\\.\\{0}'.format(name)
##            i += 1
##
##        _winreg.CloseKey(key)


    def _readval(self, cmd, loop_max):
        r = -1
        loop = 0
        while (loop < loop_max):
            try:
                r = int(self.__ser.readline().rstrip())
                if (self.__verbose != 0):
                    msg = '{0}: {1}'.format(cmd, r)
                    print(msg)
                loop = loop_max + 1   #break out of loop
            except ValueError:
                if (self.__verbose != 0):
                    msg = '{0}: No return({1})'.format(cmd,loop)
                    print(msg)
            loop = loop + 1
        return(r);

    def _flush(self):
        count = self.__ser.in_waiting
        while (count > 0):
            self.__ser.readline().rstrip()
            count = self.__ser.in_waiting
        return;

    def _set_verbose(self, value):
        self.__verbose = value
        return;



    def GetAPIVersion(self):
        """Retrieves the API version from the robot

        Returns:
            int: The API version in the robot
```

```
    """
    self._flush()
    s = 'GetAPIVersion\n'
    self.__ser.write(s.encode())
    r = self._readval("GetAPIVersion", 1)
    return(r);



def ReadSwitch(self, index):
    """Read the switch value

    Args:
        index: 0 (left) or 1 (right)

    Returns:
        int: 0 (false) or 1 (true)
    """
    self._flush()
    s = 'ReadSwitch {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    r = self._readval("ReadSwitch", 1)
    return(r);

def ReadIR(self, index):
    """Read the IR value

    Args:
        index: IR sensor to query (0 to 7)

    Returns:
        int: Value of IR sensor (0 to 4095)
    """
    self._flush()
    s = 'ReadIR {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    r = self._readval("ReadIR", 1)
    return(r);

def ReadLine(self, index):
    """Read the line sensor value

    Args:
        index: Line sensor to query (0 to 1)

    Returns:
        int: Value of Line sensor (0 to 4095)
    """
    self._flush()
    s = 'ReadLine {0}\n'.format(int(index))
```

```
        self.__ser.write(s.encode())
        r = self._readval("ReadLine", 1)
        return(r);

    def ReadLight(self):
        """Read the light sensor value

        Returns:
            int: Value of light sensor (0 to 4095)
        """
        self._flush()
        s = 'ReadLight\n'
        self.__ser.write(s.encode())
        r = self._readval("ReadLight", 1)
        return(r);

    def ReadMic(self):
        """Read the microphone sensor value

        Returns:
            int: Value of microphone sensor (0 to 4095)
        """
        self._flush()
        s = 'ReadMic\n'
        self.__ser.write(s.encode())
        r = self._readval("ReadMic", 1)
        return(r);

    def ReadAxis(self, index):
        """Read the axis value of the accelerometer

        Args:
            index: Axis to query (0 to 3)

        Returns:
            int: Value of accelerometer axis (-32768 to 32767)
        """
        self._flush()
        s = 'ReadAxis {0}\n'.format(int(index))
        self.__ser.write(s.encode())
        r = self._readval("ReadAxis", 1)
        return(r);



    def SetMotors(self, left, right):
        """Set the motors speed

        Args:
            left: value of left motor (0 to 100)
            right: value of right motor (0 to 100)
```

```
        """
        s = 'SetMotors {0} {1}\n'.format(int(left),int(right))
        self.__ser.write(s.encode())
        return;

    def Forwards(self, distance):
        """Set the robot moving forward

        Args:
            distance: distance to move (0 to 1000) in mm
        """
        self._flush()
        s = 'Forwards {0}\n'.format(int(distance))
        self.__ser.write(s.encode())
        timeout = abs(int(distance / 50))
        if (timeout <= 0):
            timeout = 1
        r = self._readval("Forwards", timeout)
        return(r);

    def Backwards(self, distance):
        """Set the robot moving backwards

        Args:
            distance: distance to move (0 to 1000) in mm
        """
        self._flush()
        s = 'Backwards {0}\n'.format(int(distance))
        self.__ser.write(s.encode())
        timeout = abs(int(distance / 50))
        if (timeout <= 0):
            timeout = 1
        r = self._readval("Backwards", timeout)
        return(r);

    def Left(self, angle):
        """Set the robot to turn left

        Args:
            angle: angle to rotate (0 to 360) in degrees
        """
        self._flush()
        s = 'Left {0}\n'.format(int(angle))
        self.__ser.write(s.encode())
        timeout = abs(int(angle / 45))
        if (timeout <= 0):
            timeout = 1
        r = self._readval("Left", timeout)
        return(r);

    def Right(self, angle):
```

```
    """Set the robot to turn right

    Args:
        angle: angle to rotate (0 to 360) in degrees
    """
    self._flush()
    s = 'Right {0}\n'.format(int(angle))
    self.__ser.write(s.encode())
    timeout = abs(int(angle / 45))
    if (timeout <= 0):
        timeout = 1
    r = self._readval("Right", timeout)
    return(r);



def LEDWrite(self, value):
    """Set the value of the LEDs

    Args:
        value: Value to set the LEDS (0 to 255)
    """
    s = 'LEDWrite {0}\n'.format(int(value))
    self.__ser.write(s.encode())
    return;

def LEDOn(self, index):
    """Turn an LED on

    Args:
        index: The LED to turn on (0 to 7)
    """
    s = 'LEDOn {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    return;

def LEDOff(self, index):
    """Turn an LED off

    Args:
        index: The LED to turn off (0 to 7)
    """
    s = 'LEDOff {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    return;

def PlayNote(self, note, length):
    """Play a note on the speaker

    Args:
        note: The frequency of the note (1 to 10000) in Hz
```

```
            length: The duration of the note (1 to 10000) in ms
        """
        s = 'PlayNote {0} {1}\n'.format(int(note),int(length))
        self.__ser.write(s.encode())
        time.sleep(length/1000)
        return;

    def GetBattery():
        s = 'GetBattery\n'
        self.__ser.write(s.encode())
        r = self._readval("Battery", timeout)
        return(r);

    def ExpDDR(self, bitnumber,direction):
        s = 'ExpDDR {0} {1}\n'.format(bitnumber,direction)
        self.__ser.write(s.encode())
        return;

    def ExpRead(self, bitnumber):
        s = 'ExpRead {0}\n'.format(bitnumber)
        self.__ser.write(s.encode())
        r = self._readval("Bit", timeout)
        return(r);

    def ExpWrite(self, bitnumber,value):
        s = 'ExpWrite {0} {1}\n'.format(bitnumber,value)
        self.__ser.write(s.encode())
        return;

    def ExpAn(self, annumber):
        s = 'ExpAn {0}\n'.format(annumber)
        self.__ser.write(s.encode())
        r = self._readval("An", timeout)
        return(r);

##    #blocking...
##    def PlayNote(self, note, time):
##        self._flush()
##        s = 'PlayNote {0} {1}\n'.format(int(note),int(time))
##        self.__ser.write(s.encode())
##        timeout = abs(int(time / 1000))
##        if (timeout <= 0):
##            timeout = 1
##        r = self._readval("PlayNote", timeout)
##        return;


    def ServoEnable(self, index):
        """Enable a servo motor

        Args:
```

```
        index: The servo to control (0 to 3)
    """
    s = 'ServoEnable {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    return;

def ServoDisable(self, index):
    """Disable a servo motor

    Args:
        index: The servo to control (0 to 3)
    """
    s = 'ServoDisable {0}\n'.format(int(index))
    self.__ser.write(s.encode())
    return;

def ServoSetPos(self, index, position):
    """Move a servo immediately to a position

    Args:
        index: The servo to control (0 to 3)
        position: The position of the servo (0 to 255)
    """
    s = 'ServoSetPos {0} {1}\n'.format(int(index),int(position))
    self.__ser.write(s.encode())
    return;

def ServoAutoMove(self, index, position):
    """Auto-move a servo to a position

    Args:
        index: The servo to control (0 to 3)
        position: The position of the servo (0 to 255)
    """
    s = 'ServoAutoMove {0} {1}\n'.format(int(index),int(position))
    self.__ser.write(s.encode())
    return;

def ServoMoveSpeed(self, speed):
    """Set the auto-move speed

    Args:
        speed: The servo speed (1 to 50)
    """
    s = 'ServoMoveSpeed {0}\n'.format(int(speed))
    self.__ser.write(s.encode())
    return;
```

```
def LCDClear(self):
    """Clear the LCD
    """
    s = 'LCDClear\n'
    self.__ser.write(s.encode())
    return;

def LCDPrint(self, x, y, text):
    """Display text on the LCD

    Args:
        x: The x-coordinate (0 to 127)
        y: The y-coordinate (0 to 31)
        text: The text to display
    """
    s = 'LCDPrint {0} {1} {2}\n'.format(int(x),int(y),text)
    self.__ser.write(s.encode())
    return;

def LCDNumber(self, x, y, value):
    """Display a number on the LCD

    Args:
        x: The x-coordinate (0 to 127)
        y: The y-coordinate (0 to 31)
        value: The number to display (-32768 to 32767)
    """
    s = 'LCDNumber {0} {1} {2}\n'.format(int(x),int(y),int(value))
    self.__ser.write(s.encode())
    return;

def LCDPixel(self, x, y, state):
    """Display a pixel on the LCD

    Args:
        x: The x-coordinate (0 to 127)
        y: The y-coordinate (0 to 31)
        state: 0 (off) or 1 (on)
    """
    s = 'LCDPixel {0} {1} {2}\n'.format(int(x),int(y),int(state))
    self.__ser.write(s.encode())
    return;

def LCDLine(self, x1, y1, x2, y2):
    """Display a line on the LCD between points A and B

    Args:
        x1: The x-coordinate of point A (0 to 127)
        y1: The y-coordinate of point A (0 to 31)
        x2: The x-coordinate of point B (0 to 127)
```

```
            y2: The y-coordinate of point B (0 to 31)
        """
        s = 'LCDLine {0} {1} {2}
{3}\n'.format(int(x1),int(y1),int(x2),int(y2))
        self.__ser.write(s.encode())
        return;

    def LCDRect(self, x1, y1, x2, y2):
        """Display a rectangle on the LCD between points A and B

        Args:
            x1: The x-coordinate of point A (0 to 127)
            y1: The y-coordinate of point A (0 to 31)
            x2: The x-coordinate of point B (0 to 127)
            y2: The y-coordinate of point B (0 to 31)
        """
        s = 'LCDRect {0} {1} {2}
{3}\n'.format(int(x1),int(y1),int(x2),int(y2))
        self.__ser.write(s.encode())
        return;

    def LCDBacklight(self, value):
        """Control the backlight of the display

        Args:
            value: The brightness of the backlight (0 to 100)
        """
        s = 'LCDBacklight {0}\n'.format(int(value))
        self.__ser.write(s.encode())
        return;

    def LCDOptions(self, foreground, background, transparent):
        """Set the options for the display

        Args:
            foreground: The foreground colour, 0 (white) or 1 (black)
            background: The background colour, 0 (white) or 1 (black)
            transparent: The transparency, 0 (false) or 1 (true)
        """
        s = 'LCDOptions {0} {1} {2}\n'.format(int(foreground),
int(background), int(transparent))
        self.__ser.write(s.encode())
        return;

    def _LCDVerbose(self, value):
        s = 'LCDVerbose {0}\n'.format(int(value))
        self.__ser.write(s.encode())
        return;
```

```
    def CardInit(self):
        self._flush()
        s = 'CardInit\n'
        self.__ser.write(s.encode())
        r = self._readval("CardInit", 2)
        return(r);

    def CardCreate(self, filename):
        self._flush()
        s = 'CardCreate {0}\n'.format(filename)
        self.__ser.write(s.encode())
        r = self._readval("CardCreate", 2)
        return(r);

    def CardOpen(self, filename):
        self._flush()
        s = 'CardOpen {0}\n'.format(filename)
        self.__ser.write(s.encode())
        r = self._readval("CardOpen", 2)
        return(r);

    def CardDelete(self, filename):
        self._flush()
        s = 'CardDelete {0}\n'.format(filename)
        self.__ser.write(s.encode())
        r = self._readval("CardDelete", 2)
        return(r);

    def CardWriteByte(self, data):
        #self._flush()
        s = 'CardWriteByte {0}\n'.format(int(data))
        self.__ser.write(s.encode())
        #r = self._readval("CardWriteByte", 2)
        return();

    def CardReadByte(self):
        self._flush()
        s = 'CardReadByte\n'
        self.__ser.write(s.encode())
        r = self._readval("CardReadByte", 2)
        return(r);

    def CardRecordMic(self, bitdepth, samplerate, time, filename):
        self._flush()
        s = 'CardRecordMic {0} {1} {2}
{3}\n'.format(int(bitdepth),int(samplerate),int(time),filename)
        self.__ser.write(s.encode())
        r = self._readval("CardRecordMic", time+1)
        return(r);

    def CardPlayback(self, filename):
```

```
        self._flush()
        s = 'CardPlayback {0}\n'.format(filename)
        self.__ser.write(s.encode())
        r = self._readval("CardPlayback", 50)
        return(r);

    def CardBitmap(self, x, y, filename):
        self._flush()
        s = 'CardBitmap {0} {1} {2}\n'.format(int(x),int(y),filename)
        self.__ser.write(s.encode())
        r = self._readval("CardBitmap", 5)
        return(r);
```

Apéndice G. Código Máquina de Estados de un Robot

```
import time
import FA

# Configuración inicial del robot Formula AllCode
fa = FA.Create()
comport = 5

# Definición de estados
ESTADO_NORMAL = "NORMAL"
ESTADO_CALLEJON = "CALLEJON"
ESTADO_GIRO_IZQUIERDA = "GIRO_IZQUIERDA"
ESTADO_GIRO_DERECHA = "GIRO_DERECHA"
ESTADO_OBSTACULO_FRONTAL = "OBSTACULO_FRONTAL"
ESTADO_CHOQUE_IZQUIERDO = "CHOQUE_IZQUIERDO"
ESTADO_CHOQUE_DERECHO = "CHOQUE_DERECHO"
ESTADO_LUZ_DETECTADA = "LUZ_DETECTADA"
ESTADO_HUECO_DERECHO = "HUECO_DERECHO"

# Variables globales para control PD
error = 0
dif = 0
difAnt = 0
Kp = 0.288
Kd = 0.18

estado_actual = ESTADO_NORMAL  # Estado inicial

def setup():
    fa.ComOpen(comport)

def leer_sensores():
    """Función para leer los sensores del robot."""
    sensor_izquierdo = fa.ReadIR(0)
    sensor_frontal_izquierdo = fa.ReadIR(1)
    sensor_central = fa.ReadIR(2)
```

```
    sensor_frontal_derecho = fa.ReadIR(3)
    sensor_derecho = fa.ReadIR(4)
    sensor_luz = fa.ReadLight()

    return sensor_izquierdo, sensor_frontal_izquierdo, sensor_central,
sensor_frontal_derecho, sensor_derecho, sensor_luz

# Funciones que definen cada estado
def estado_normal():
    global estado_actual, error, dif, difAnt

    sensor_izquierdo, _, _, _, sensor_derecho, _ = leer_sensores()

    # Control PD
    dif = sensor_derecho - sensor_izquierdo
    error = round(Kp * dif + Kd * (dif - difAnt))
    difAnt = dif

    speedL = max(0, min(30, 30 - error))
    speedR = max(0, min(30, 30 + error))
    fa.SetMotors(speedL, speedR)

def estado_callejon():
    global estado_actual
    fa.SetMotors(0, 0)
    time.sleep(0.01)
    fa.Right(180)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_giro_izquierda():
    global estado_actual
    fa.SetMotors(0, 0)
    time.sleep(0.01)
    fa.Left(80)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_giro_derecha():
    global estado_actual
    fa.SetMotors(0, 0)
    time.sleep(0.01)
    fa.Right(80)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_obstaculo_frontal():
    global estado_actual
    fa.SetMotors(0, 0)
    fa.Backwards(10)
    time.sleep(0.01)
```

```
    fa.Right(80)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_choque_izquierdo():
    global estado_actual
    fa.SetMotors(0, 0)
    fa.Backwards(10)
    time.sleep(0.01)
    fa.Right(20)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_choque_derecho():
    global estado_actual
    fa.SetMotors(0, 0)
    fa.Backwards(10)
    time.sleep(0.01)
    fa.Left(20)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

def estado_luz_detectada():
    fa.SetMotors(0, 0)
    print("Luz detectada, deteniendo el robot.")

def estado_hueco_derecho():
    global estado_actual
    fa.Left(15)
    time.sleep(0.01)
    estado_actual = ESTADO_NORMAL

# Diccionario que mapea estados a funciones
maquina_estados = {
    ESTADO_NORMAL: estado_normal,
    ESTADO_CALLEJON: estado_callejon,
    ESTADO_GIRO_IZQUIERDA: estado_giro_izquierda,
    ESTADO_GIRO_DERECHA: estado_giro_derecha,
    ESTADO_OBSTACULO_FRONTAL: estado_obstaculo_frontal,
    ESTADO_CHOQUE_IZQUIERDO: estado_choque_izquierdo,
    ESTADO_CHOQUE_DERECHO: estado_choque_derecho,
    ESTADO_LUZ_DETECTADA: estado_luz_detectada,
    ESTADO_HUECO_DERECHO: estado_hueco_derecho
}

def cambiar_estado():
    """Función que decide el cambio de estado basado en sensores."""
    global estado_actual

    sensor_izquierdo, sensor_frontal_izquierdo, sensor_central,
sensor_frontal_derecho, sensor_derecho, sensor_luz = leer_sensores()
```

```
    # Decidir estado en función de los sensores
    if sensor_luz > 4000:
        estado_actual = ESTADO_LUZ_DETECTADA
    elif sensor_frontal_derecho < 10 and sensor_derecho > 2500:
        estado_actual = ESTADO_HUECO_DERECHO
    elif sensor_central > 1100 and sensor_derecho > 10 and sensor_izquierdo >
10:
        estado_actual = ESTADO_CALLEJON
    elif sensor_central > 1100 and sensor_derecho > 10:
        estado_actual = ESTADO_GIRO_IZQUIERDA
    elif sensor_central > 1100 and sensor_izquierdo > 10:
        estado_actual = ESTADO_GIRO_DERECHA
    elif sensor_central > 1100:
        estado_actual = ESTADO_OBSTACULO_FRONTAL
    elif sensor_derecho > 3500 or sensor_frontal_derecho > 3500:
        estado_actual = ESTADO_CHOQUE_DERECHO
    elif sensor_izquierdo > 3500 or sensor_frontal_izquierdo > 3500:
        estado_actual = ESTADO_CHOQUE_IZQUIERDO
    else:
        estado_actual = ESTADO_NORMAL

def ejecutar_estado():
    """Función que ejecuta la acción basada en el estado actual."""
    maquina_estados[estado_actual]()

# Loop principal
def loop():
    while True:
        cambiar_estado()
        ejecutar_estado()
        time.sleep(0.01)
        time.sleep(0.01)

if __name__ == '__main__':
    try:
        setup()
        while True:
            loop()
    except KeyboardInterrupt:
        fa.SetMotors(0, 0)
        fa.ComClose()
```

Apéndice H. Código Pruebas de Detección

```
import time
import csv
```

```
import FA

# Configuración de los robots
robots = {
    "FA104318": 4,  # Puerto COM para el primer robot
    "FA104350": 5   # Puerto COM para el segundo robot
}

nota_frecuencia = 392  # Frecuencia de la nota G4
duracion_nota = 5000   # Duración de la emisión en ms (5 segundos)
umbral_microfono = 918  # Umbral para considerar un sonido detectado

distancias = [1, 2, 3, 4, 5]  # Distancias en cm
orientaciones = ["frontal", "lateral izquierda", "lateral derecha"]  #
Posiciones relativas


def emitir_sonido(robot_id, puerto):
    """Emite la nota G4 desde el robot."""
    fa = FA.Create()
    fa.ComOpen(puerto)
    try:
        print(f"Robot {robot_id} emitiendo sonido a {nota_frecuencia} Hz por
{duracion_nota / 1000} segundos...")
        fa.PlayNote(nota_frecuencia, duracion_nota)
        time.sleep(duracion_nota / 1000)  # Esperar la duración
        print(f"Robot {robot_id} finalizó la emisión.")
    finally:
        fa.ComClose()


def detectar_sonido_promedio(robot_id, puerto, muestras=10, umbral=918):
    """Detecta sonido, aplica filtro y devuelve si supera el umbral."""
    fa = FA.Create()
    fa.ComOpen(puerto)
    try:
        valores = []
        for _ in range(muestras):
            nivel = fa.ReadMic()
            valores.append(nivel)
            time.sleep(0.1)  # Pausa entre lecturas

        # Aplicar mediana como filtro
        promedio = sorted(valores)[len(valores) // 2]
        return promedio > umbral, promedio
    finally:
        fa.ComClose()


def pruebas_deteccion():
```

```
    """Ejecuta pruebas de emisión y detección y guarda resultados en un
archivo CSV."""
    archivo_salida = "resultados_deteccion.csv"
    with open(archivo_salida, mode="w", newline="") as archivo:
        writer = csv.writer(archivo)
        writer.writerow(["Distancia (cm)", "Orientación", "Emisor",
"Receptor", "Promedio Micro", "Detectado", "Tipo Escenario"])

        for distancia in distancias:
            for orientacion in orientaciones:
                for tipo_escenario in ["robot", "obstaculo"]:  # Escenarios:
sonido presente (robot) y ausente (obstáculo)
                    for emisor_id, receptor_id in [(list(robots.keys())[0],
list(robots.keys())[1]),
                                                   (list(robots.keys())[1],
list(robots.keys())[0])]:
                        input(f"Coloca los robots a una distancia de
{distancia} cm y orientación {orientacion}. Escenario: {tipo_escenario}.
Presiona Enter para continuar...")

                        print(f"Prueba: Distancia {distancia} cm, Orientación
{orientacion}, Escenario {tipo_escenario}, Emisor {emisor_id}, Receptor
{receptor_id}")

                        if tipo_escenario == "robot":
                            # Emitir sonido
                            emitir_sonido(emisor_id, robots[emisor_id])
                        else:
                            print(f"Simulando obstáculo: {emisor_id} no emite
sonido.")
                            time.sleep(2)  # Pausa equivalente a la duración
de emisión

                        # Detectar sonido
                        detectado, promedio =
detectar_sonido_promedio(receptor_id, robots[receptor_id])

                        # Guardar resultados
                        writer.writerow([distancia, orientacion, emisor_id,
receptor_id, promedio, "Sí" if detectado else "No", tipo_escenario])
                        print(f"Resultados: Promedio={promedio:.2f},
Detectado={'Sí' if detectado else 'No'}")
                        time.sleep(1)  # Pausa entre pruebas

    print(f"Pruebas completadas. Resultados guardados en {archivo_salida}")


if __name__ == "__main__":
    pruebas_deteccion()
```

Apéndice I. Código de Algoritmo de Navegación threading

```
import time
import threading
import FA
import deteccion

# Configuración inicial de los robots Formula AllCode
fa1 = FA.Create()
fa2 = FA.Create()
comport1 = 4
comport2 = 5

# Definición de estados
ESTADO_FWD = "FWD"
ESTADO_RTT = "RTT"
ESTADO_BACK = "BACK"
ESTADO_WAIT = "WAIT"

# Variables globales para control PD
error = 0
dif = 0
difAnt = 0
Kp = 0.288
Kd = 0.18

# Estado actual de los robots
estado_actual1 = ESTADO_FWD  # Estado inicial del robot 1
estado_actual2 = ESTADO_FWD  # Estado inicial del robot 2

# Umbrales de los sensores
umbrales = {
    "sensor_izquierdo": 3500,
    "sensor_frontal_izquierdo": 1800,
    "sensor_frontal_derecho": 1800,
    "sensor_derecho": 2500,
    "sensor_central": 1200,
    "sensor_luz": 1500  # Umbral de luz para activar el estado WAIT (ajustado
a 1500)
}

# Tiempo máximo de espera (ajustado para que sea más largo y observable)
t_max = 60  # Aumentado el valor para un tiempo de espera más largo

# Parámetro θ (puede ser ajustado según el comportamiento deseado)
theta = 0.5

def setup():
    # Abrir la comunicación con los robots
    fa1.ComOpen(comport1)
    fa2.ComOpen(comport2)
```

```
def leer_sensores(robot):
    """Función para leer los sensores del robot."""
    sensor_izquierdo = robot.ReadIR(0)
    sensor_frontal_izquierdo = robot.ReadIR(1)
    sensor_central = robot.ReadIR(2)
    sensor_frontal_derecho = robot.ReadIR(3)
    sensor_derecho = robot.ReadIR(4)
    sensor_luz = robot.ReadLight()  # Leer sensor de luz

    return sensor_izquierdo, sensor_frontal_izquierdo, sensor_central,
sensor_frontal_derecho, sensor_derecho, sensor_luz

def calcular_tiempo_espera(sensor_luz):
    """Calcula el tiempo de espera basado en la intensidad de luz."""
    s_min = 0  # El valor mínimo de intensidad de luz
    t_espera = t_max * ((sensor_luz - s_min) ** 2) / ((sensor_luz - s_min) **
2 + theta)  # Ajustada la fórmula
    return t_espera

# Funciones que definen cada estado
def estado_fwd(robot):
    global error, dif, difAnt

    sensor_izquierdo, _, sensor_central, _, sensor_derecho, _ =
leer_sensores(robot)

    # Control PD
    dif = sensor_derecho - sensor_izquierdo
    error = round(Kp * dif + Kd * (dif - difAnt))
    difAnt = dif

    speedL = max(0, min(30, 30 - error))
    speedR = max(0, min(30, 30 + error))
    robot.SetMotors(speedL, speedR)

    # Verificar si el sensor frontal detecta una colisión
    if sensor_central > umbrales["sensor_central"]:
        # Realizar la detección para saber si es una pared o un robot
        if deteccion.certificar_colision(robot):  # Llamar a la función de
detección
            print("Robot detectado, cambiando a estado WAIT")
            return ESTADO_WAIT  # Es otro robot, se detiene y espera
        else:
            print("Pared detectada, cambiando a estado RTT")
            return ESTADO_RTT  # Es una pared, realiza un giro para evitarla

    return ESTADO_FWD

def estado_rtt(robot):
```

```
    sensor_izquierdo, _, sensor_central, sensor_frontal_derecho,
sensor_derecho, _ = leer_sensores(robot)

    # Evento 1: Callejón cerrado
    if sensor_central > umbrales["sensor_central"] and sensor_derecho > 10
and sensor_izquierdo > 10:
        print("Evento 1: Callejón cerrado. Girando 180° a la derecha.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Right(180)
        time.sleep(0.5)
        return ESTADO_FWD

    # Evento 2: Obstáculo al frente y a la derecha, costado izquierdo libre
    elif sensor_central > umbrales["sensor_central"] and sensor_derecho > 10:
        print("Evento 2: Obstáculo al frente y a la derecha. Girando 80° a la
izquierda.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Left(80)
        time.sleep(0.5)
        return ESTADO_FWD

    # Evento 3: Obstáculo al frente y a la izquierda, costado derecho libre
    elif sensor_central > umbrales["sensor_central"] and sensor_izquierdo >
10:
        print("Evento 3: Obstáculo al frente y a la izquierda. Retrocediendo
y girando 80° a la derecha.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Backwards(10)
        time.sleep(0.5)
        robot.Right(80)
        time.sleep(0.5)
        return ESTADO_FWD

    # Evento 4: Solo obstáculo al frente
    elif sensor_central > umbrales["sensor_central"]:
        print("Evento 4: Obstáculo al frente. Retrocediendo y girando 90° a
la derecha.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Backwards(10)
        time.sleep(0.5)
        robot.Right(90)
        time.sleep(0.5)
        return ESTADO_FWD

    return ESTADO_RTT

def estado_back(robot):
```

```
    sensor_izquierdo, sensor_frontal_izquierdo, _, _, sensor_derecho, _ =
leer_sensores(robot)

    # Evento 5: Choque en la rueda derecha
    if sensor_derecho > umbrales["sensor_derecho"]:
        print("Evento 5: Choque en la rueda derecha. Retrocediendo y girando
20° a la izquierda.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Backwards(10)
        time.sleep(0.5)
        robot.Left(20)
        time.sleep(0.5)
        return ESTADO_FWD

    # Evento 6: Choque en la rueda izquierda
    elif sensor_izquierdo > umbrales["sensor_izquierdo"]:
        print("Evento 6: Choque en la rueda izquierda. Retrocediendo y
girando 20° a la derecha.")
        robot.SetMotors(0, 0)
        time.sleep(0.5)
        robot.Backwards(10)
        time.sleep(0.5)
        robot.Right(20)
        time.sleep(0.5)
        return ESTADO_FWD

    return ESTADO_BACK

def estado_wait(robot):
    sensor_izquierdo, _, _, _, _, sensor_luz = leer_sensores(robot)

    # Detener los motores cuando se entra en el estado WAIT
    print("Entrando en estado WAIT, deteniendo los motores.")
    robot.SetMotors(0, 0)

    # Calcular el tiempo de espera basado en la intensidad de la luz
    t_espera = calcular_tiempo_espera(sensor_luz)

    # Imprimir el tiempo de espera calculado
    print(f"Luz detectada: {sensor_luz}. Tiempo de espera calculado:
{t_espera:.2f} segundos.")

    time.sleep(t_espera)  # Esperar el tiempo calculado
    return ESTADO_FWD

# Diccionario que mapea estados a funciones
maquina_estados = {
    ESTADO_FWD: estado_fwd,
    ESTADO_RTT: estado_rtt,
    ESTADO_BACK: estado_back,
```

```
    ESTADO_WAIT: estado_wait
}

def cambiar_estado(robot, estado_actual):
    """Función que decide el cambio de estado basado en sensores."""
    sensor_izquierdo, sensor_frontal_izquierdo, sensor_central,
sensor_frontal_derecho, sensor_derecho, sensor_luz = leer_sensores(robot)

    # Decidir estado en función de los sensores
    if sensor_luz > umbrales["sensor_luz"]:
        estado_actual = ESTADO_WAIT  # Estado WAIT si se detecta luz intensa
    elif sensor_central > umbrales["sensor_central"] and sensor_derecho > 10
and sensor_izquierdo > 10:
        estado_actual = ESTADO_RTT  # Callejones o callejones cerrados
    elif sensor_central > umbrales["sensor_central"]:
        estado_actual = ESTADO_BACK  # Si detecta un obstáculo en el frente
    elif sensor_derecho > umbrales["sensor_derecho"] or
sensor_frontal_derecho > umbrales["sensor_frontal_derecho"]:
        estado_actual = ESTADO_BACK  # Si detecta un choque a la derecha
    elif sensor_izquierdo > umbrales["sensor_izquierdo"] or
sensor_frontal_izquierdo > umbrales["sensor_frontal_izquierdo"]:
        estado_actual = ESTADO_BACK  # Si detecta un choque a la izquierda
    else:
        estado_actual = ESTADO_FWD  # Continúa avanzando

    return estado_actual

def ejecutar_estado(robot, maquina_estados, estado_actual):
    """Función que ejecuta la acción basada en el estado actual."""
    estado_actual = maquina_estados[estado_actual](robot)
    print(f"Estado actual del robot: {estado_actual}")  # Imprimir el estado
en la consola
    return estado_actual

# Función para manejar un robot
def manejar_robot(robot, maquina_estados, estado_actual):
    while True:
        estado_actual = cambiar_estado(robot, estado_actual)
        estado_actual = ejecutar_estado(robot, maquina_estados,
estado_actual)

# Loop principal
if __name__ == '__main__':
    try:
        setup()
        # Crear hilos para cada robot
        hilo_robot1 = threading.Thread(target=manejar_robot, args=(fa1,
maquina_estados, ESTADO_FWD))
        hilo_robot2 = threading.Thread(target=manejar_robot, args=(fa2,
maquina_estados, ESTADO_FWD))
```

```
        hilo_robot1.start()
        hilo_robot2.start()

        hilo_robot1.join()
        hilo_robot2.join()
    except KeyboardInterrupt:
        fa1.SetMotors(0, 0)
        fa2.SetMotors(0, 0)
        fa1.ComClose()
        fa2.ComClose()
```

Apéndice J. Código de Detección

```
import FA
import time

# Umbral configurable para detección de sonido
UMBRAL_SONIDO = 918  # Nivel mínimo para considerar otro robot
FRECUENCIA_EMISION_1 = 392  # Primer tono de emisión (G4)
FRECUENCIA_EMISION_2 = 440  # Segundo tono de emisión (A4)
DURACION_EMISION = 1000  # Duración de cada tono en ms
RETARDO_RESPUESTA = 500  # Retardo entre emisión y respuesta en ms


def emitir_sonido(fa, patron=True):
    """
    Hace que el robot emita un sonido en un patrón específico.
    """
    if patron:
        # Patrón de dos tonos
        fa.PlayNote(FRECUENCIA_EMISION_1, DURACION_EMISION)
        time.sleep(DURACION_EMISION / 1000)
        fa.PlayNote(FRECUENCIA_EMISION_2, DURACION_EMISION)
    else:
        # Tono continuo
        fa.PlayNote(FRECUENCIA_EMISION_1, DURACION_EMISION * 2)

    print("Robot emitiendo sonido.")


def escuchar_sonido(fa):
    """
    Escucha sonidos y detecta si el nivel supera el umbral definido.

    Args:
        fa: Objeto de conexión al robot.

    Returns:
        bool: True si se detecta otro robot, False si no.
    """
```

```
    nivel_sonido = fa.ReadMic()
    print(f"Nivel de sonido detectado: {nivel_sonido}")
    return nivel_sonido >= UMBRAL_SONIDO


def certificar_colision(fa):
    """
    Certifica la colisión con otro robot basado en la detección mutua.

    Args:
        fa: Objeto de conexión al robot.

    Returns:
        bool: True si se certifica la colisión, False si no.
    """
    # Emitir sonido inicial
    emitir_sonido(fa)

    # Esperar respuesta
    time.sleep(RETARDO_RESPUESTA / 1000)

    # Escuchar si hay sonido de retorno
    if escuchar_sonido(fa):
        print("Colisión certificada: otro robot detectado.")
        return True
    else:
        print("No se certificó colisión: obstáculo estático o ruido.")
        return False
```

Apéndice K. Código del Algoritmo de Agrupamiento asyncio

```
import asyncio
import FA

# Configuración inicial
comport1 = 4
comport2 = 5
fa1 = FA.Create()
fa2 = FA.Create()

# Definición de estados
ESTADO_FWD = "FWD"
ESTADO_RTT = "RTT"
ESTADO_BACK = "BACK"
ESTADO_WAIT = "WAIT"

# Configuración del control PD
Kp = 0.288
Kd = 0.18
t_max = 30
```

```
theta = 100000
s_min = 500

# Umbrales
umbrales = {
    "sensor_izquierdo": 3500,
    "sensor_frontal_izquierdo": 1800,
    "sensor_frontal_derecho": 1800,
    "sensor_derecho": 2500,
    "sensor_central": 1200,
    "sensor_luz": 1000
}

async def leer_sensores(robot):
    """Lee los valores de los sensores del robot."""
    return {
        "sensor_izquierdo": robot.ReadIR(0),
        "sensor_frontal_izquierdo": robot.ReadIR(1),
        "sensor_central": robot.ReadIR(2),
        "sensor_frontal_derecho": robot.ReadIR(3),
        "sensor_derecho": robot.ReadIR(4),
        "sensor_luz": robot.ReadLight()
    }

def calcular_tiempo_espera(sensor_luz):
    """Calcula el tiempo de espera basado en la intensidad de luz."""
    return t_max * ((sensor_luz - s_min) ** 2) / ((sensor_luz - s_min) ** 2 +
theta)

async def detectar_robot(robot):
    """Detecta si el obstáculo es otro robot o una pared."""
    robot.PlayNote(392, 2000)  # Emite un tono de 392 Hz
    await asyncio.sleep(2)
    valores = [robot.ReadMic() for _ in range(10)]
    promedio = sum(valores) / len(valores)
    return promedio > 2000

async def estado_fwd(robot, sensores, error_prev):
    """Estado de avance (FWD)."""
    dif = sensores["sensor_derecho"] - sensores["sensor_izquierdo"]
    error = round(Kp * dif + Kd * (dif - error_prev))
    speedL = max(0, min(30, 30 - error))
    speedR = max(0, min(30, 30 + error))
    robot.SetMotors(speedL, speedR)

    # Condiciones para transicionar a otros estados
    if sensores["sensor_luz"] > umbrales["sensor_luz"]:
        return ESTADO_WAIT, error
    elif sensores["sensor_central"] > umbrales["sensor_central"]:
        if await detectar_robot(robot):
            return ESTADO_WAIT, error
```

```
            return ESTADO_RTT, error

        return ESTADO_FWD, error

async def estado_rtt(robot, sensores):
    """Estado de giro (RTT)."""
    if sensores["sensor_central"] > umbrales["sensor_central"] and
sensores["sensor_derecho"] > 10 and sensores["sensor_izquierdo"] > 10:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Right(180)
            return ESTADO_FWD
    elif sensores["sensor_central"] > umbrales["sensor_central"] and
sensores["sensor_derecho"] > 10:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Left(80)
            return ESTADO_FWD
    elif sensores["sensor_central"] > umbrales["sensor_central"] and
sensores["sensor_izquierdo"] > 10:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Backwards(10)
            await asyncio.sleep(0.5)
            robot.Right(80)
            return ESTADO_FWD
    elif sensores["sensor_central"] > umbrales["sensor_central"]:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Backwards(10)
            await asyncio.sleep(0.5)
            robot.Right(90)
            return ESTADO_FWD

    return ESTADO_RTT

async def estado_back(robot, sensores):
    """Estado de retroceso (BACK)."""
    if sensores["sensor_derecho"] > umbrales["sensor_derecho"]:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Backwards(10)
            await asyncio.sleep(0.5)
            robot.Left(20)
            return ESTADO_FWD
    elif sensores["sensor_izquierdo"] > umbrales["sensor_izquierdo"]:
            robot.SetMotors(0, 0)
            await asyncio.sleep(0.5)
            robot.Backwards(10)
            await asyncio.sleep(0.5)
            robot.Right(20)
```

```
            return ESTADO_FWD

        return ESTADO_BACK

async def estado_wait(robot, sensores):
    """Estado de espera (WAIT)."""
    tiempo_espera = calcular_tiempo_espera(sensores["sensor_luz"])
    robot.SetMotors(0, 0)
    print(f"Tiempo de espera calculado: {tiempo_espera:.2f} segundos.")
    await asyncio.sleep(tiempo_espera)
    return ESTADO_FWD

async def maquina_estados(robot, estado_inicial):
    """Ejecuta la máquina de estados."""
    estado_actual = estado_inicial
    error_prev = 0

    while True:
        sensores = await leer_sensores(robot)
        if estado_actual == ESTADO_FWD:
            estado_actual, error_prev = await estado_fwd(robot, sensores,
error_prev)
        elif estado_actual == ESTADO_RTT:
            estado_actual = await estado_rtt(robot, sensores)
        elif estado_actual == ESTADO_BACK:
            estado_actual = await estado_back(robot, sensores)
        elif estado_actual == ESTADO_WAIT:
            estado_actual = await estado_wait(robot, sensores)

        print(f"Estado actual: {estado_actual}")
        await asyncio.sleep(0.1)

async def main():
    """Función principal para controlar ambos robots."""
    fa1.ComOpen(comport1)
    fa2.ComOpen(comport2)

    try:
        await asyncio.gather(
            maquina_estados(fa1, ESTADO_FWD),
            maquina_estados(fa2, ESTADO_FWD)
        )
    except KeyboardInterrupt:
        fa1.SetMotors(0, 0)
        fa2.SetMotors(0, 0)
        fa1.ComClose()
        fa2.ComClose()

if __name__ == "__main__":
    asyncio.run(main())
```