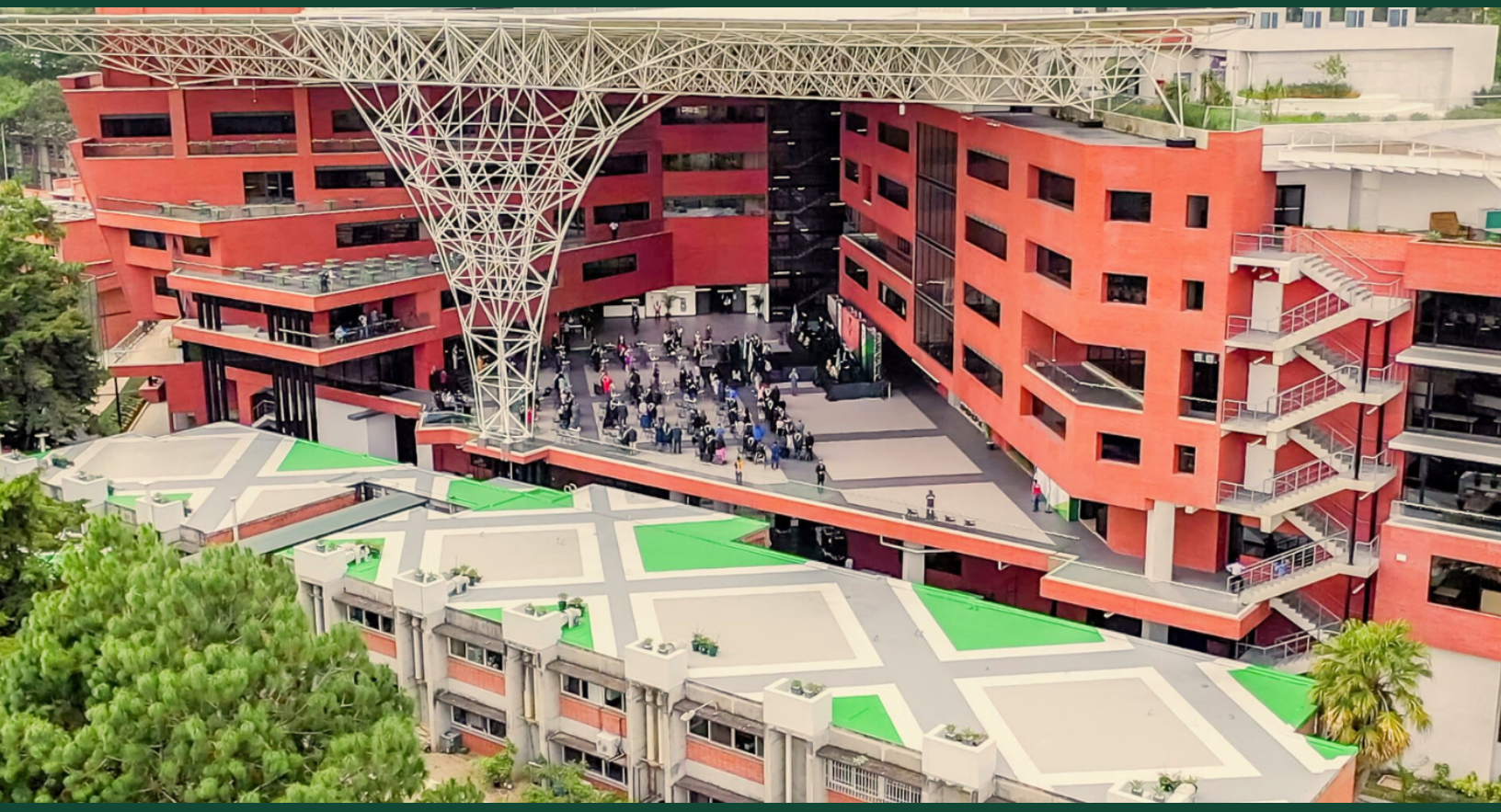
Implementación y validación de distintos algoritmos de inteligencia de enjambre para aplicaciones de ingeniería mecatrónica

Pablo Javier Peña Echeverría



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

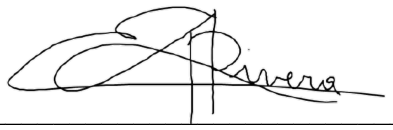
Implementación y validación de distintos algoritmos de inteligencia de enjambre para aplicaciones de ingeniería mecatrónica

Trabajo de graduación presentado por Pablo Javier Peña Echeverría
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

M.Sc. Carlos Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El siguiente trabajo se enfoca en la implementación y la validación de dos algoritmos de inteligencia de enjambre: el algoritmo del murciélago y el de la luciérnaga. El objetivo fue explorar su habilidad para resolver problemas de optimización y planificación de trayectorias, que son temas de interés para la ingeniería mecatrónica como alternativas a otros métodos, como PSO y ACO.

Se implementaron los dos algoritmos en Matlab y se validaron con una variedad de funciones de costo que permitieron analizar su comportamiento en términos de convergencia y estabilidad. Luego, los algoritmos se usaron para una aplicación de planificación de trayectorias que se simuló en Matlab y Webots. En el caso de Webots, se usaron robots diferenciales para probar la validez de las trayectorias encontradas por los algoritmos. Por último, se usaron para una aplicación en la que se determinó la posición de unos sensores en un área determinada con el fin de optimizar su cobertura de área efectiva.

Los resultados indican que el algoritmo de la luciérnaga presenta mayor estabilidad y precisión, pero necesita más tiempo por iteración, mientras que el algoritmo del murciélago logra converger en menos tiempo, pero sus resultados tienen una mayor variabilidad. Los dos algoritmos demostraron que se pueden usar para planificar trayectorias viables que se pueden transferir a simulaciones con robots, lo que confirma su potencial para aplicaciones con robótica móvil.

Palabras clave: inteligencia de enjambre, algoritmo del murciélago, algoritmo de la luciérnaga, planificación de trayectorias, Webots, Matlab.

Abstract

The following work focuses on the implementation and validation of two swarm intelligence algorithms: the Bat Algorithm and the Firefly Algorithm. The objective was to explore their ability to solve optimization and trajectory planning problems, which are topics of interest in mechatronics engineering as alternatives to other methods such as PSO and ACO.

Both algorithms were implemented in Matlab and validated using a variety of cost functions that allowed the analysis of their behavior in terms of convergence and stability. Then, the algorithms were used for a trajectory planning application that was simulated in Matlab and Webots. In the case of Webots, differential robots were used to test the validity of the trajectories found by the algorithms. Finally, they were used for an application in which the position of several sensors in a given area was determined in order to optimize their effective area coverage.

The results indicate that the Firefly Algorithm presents greater stability and precision, but requires more time per iteration, while the Bat Algorithm manages to converge in less time, but its results have greater variability. Both algorithms demonstrated that they can be used to plan feasible trajectories that can be transferred to simulations with robots, confirming their potential for applications in mobile robotics.

Keywords: swarm intelligence, bat algorithm, firefly algorithm, path planning, Webots, Matlab.

CAPÍTULO 1

Introducción

La inteligencia de enjambre toma inspiración del comportamiento colectivo de ciertos grupos de animales que logran resolver problemas al intercambiar información entre sí. En el campo de la ingeniería mecatrónica, la inteligencia de enjambre se ha usado para resolver problemas como la planificación de trayectorias, la navegación de robots móviles y problemas de optimización.

Este trabajo busca alternativas a los algoritmos de enjambre trabajados en la Universidad del Valle de Guatemala, como *particle swarm optimization* (PSO) y *ant colony optimization* (ACO). Para esto, se implementaron y evaluaron el algoritmo del murciélago (*bat algorithm*) y el algoritmo de la luciérnaga (*firefly algorithm*). El objetivo principal fue determinar si estos podían ofrecer soluciones viables para resolver problemas de interés de la ingeniería mecatrónica, como la resolución de problemas de optimización y planificación de trayectorias.

Para el desarrollo del trabajo, se investigaron las bases teóricas de los algoritmos seleccionados y se implementaron en el entorno de Matlab para validar su funcionamiento con funciones de costo sencillas. Luego, se usaron para generar trayectorias en escenarios con complejidad variable. Para terminar, se realizó una validación en el entorno de simulación Webots, usando robots diferenciales para recorrer las trayectorias generadas con los algoritmos.

CAPÍTULO 2

Antecedentes

2.1. Aplicaciones de robótica de enjambre

La robótica de enjambre es un enfoque en el que múltiples robots trabajan de forma coordinada para logar un objetivo en común, inspirado en el comportamiento colectivo de insectos u otros animales como las abejas, las hormigas, las aves, entre otros. Sus aplicaciones abarcan diversos campos como en situaciones de busqueda y rescate en donde los enjambres de robots mapean los terrenos peligrosos, buscan sobrevivientes y operan en lugares que normalmente son innaccesibles para los humanos. En agricultura [1] se han usado enjambres de drones que ayudan a optimizar el uso de agua y fertilizantes al monitorear los cultivos. Para temas de medicina [2] se han desarrollado enjambres de nanorrobots capaces de ofrecer un tratamiento preciso y menos invasivo que los métodos tradicionales. Estas y otras aplicaciones demuestran el potencial de la robótica de enjambre papra resoler problemas complejos la cooperación de por robots por medio de un control descentralizado.

2.2. *Particle swarm optimization*

Particle swarm optimization (PSO) es un algoritomo de estocástico de busqueda basado en población que lo podemos ver utilizado en [3] se propone un algoritmo híbrido de PSO para la planificación de rutas de vehículos aéreos no tripulados (UAVs). Este enfoque combina la capacidad de exploración de PSO con la explotación de otros algoritmos de optimización, la cual mejora la eficiencia y precisión en la planificación de rutas en entornos complejos. El algoritmo aborda desafíos como la evasión de obstáculos y la optimización de la trayectoria, lo que garantiza rutas seguras y eficientes para los UAVs.

2.3. *Ant colony optimization*

Ant colony optimization (ACO) es una técnica probabilística que se utiliza para resolver problemas computacionales como se hace en [4], donde se presenta un algoritmo de ACO mejorado para la planificación de rutas de robots móviles, a partir de una estrategia basada en la optimización del punto más distante y con un enfoque multiobjetivo. Este enfoque busca mejorar la eficiencia del recorrido del robot al equilibrar múltiples criterios como la minimización de la distancia y el tiempo, así como la evasión de obstáculos. La combinación de estas técnicas promete mejorar el rendimiento en la navegación autónoma de robots en entornos complejos.

2.4. Aplicaciones e implementación de robótica de enjambre en UVG

En los últimos años, la Universidad del Valle de Guatemala (UVG) ha desarrollado diversas investigaciones relacionadas con ACO aplicada a la robótica de enjambre. En el trabajo desarrollado por Gabriela Iriarte [5], se investigó la aplicación de ACO para mejorar la navegación y cooperación en robots autónomos. El trabajo propone la integración de ACO con aprendizaje automático para optimizar la selección de rutas y la toma de decisiones en entornos dinámicos. Además, emplea computación evolutiva para ajusstar parámetros clave del algoritmo, lo que mejora su convergencia y adaptabilidad. En el trabajo de investigación de Daniela Baldizón[6], se toca el tema de la implementación de algoritmos de optimización por colonias de hormigas (ACO) y su aplicación práctica para robótica de enjambre. Durante el trabajo se modificó el ACO para mejorar su rendimiento en la planificación de trayectorias y la exploración de terrenos. El trabajo se centró en integrar este algoritmo en robots autónomos para que estos puedieran realizar tareas complejas de navegación y evasión de obstáculos. Y en el trabajo de investigación hecho por Jonathan Cardona [7], se evaluó el desempeño del algoritmo ACO en un entorno robótico real. Dentro del ecosistema Robotat, un sistema de localización y control de robots en tiempo real, el autor implementa y prueba ACO en enjambres de robots móviles para analizar su eficacia en la planificación de trayectorias y la navegación autónoma.

También se han realizado varias investigaciones relacionadas con el PSO. En el trabajo de investigación desarrollado por Aldo Aguilar[8] se desarrolló y evalúo una versión mejorada del PSO para su aplicación en sistemas robóticos. Su trabajo introduce modificaciones en la dinámica de actualización de partículas, lo que mejora la exploración y evita la convergencia prematura en óptimos locales. En el trabajo de investigación de Jonathan Cardona [7] además de sus aportes con ACO también se trabajó con PSO. Se analizo su desempeño en la navegación y control de enjambres de robots móviles. Durante el desarrollo de la tesis se utilizo el ecosistema Robotat para evaluar la capacidad de planificación de trayectorias y adaptación a entornos dinámicos. Y en el trabajo de investigación de Ana Barrientos [9] se mejoró la eficiencia de PSO para la navegación autónoma en entornos dinámicos. Su trabajo se enfoca en optimizar la convergencia del algoritmo y su capacidad de evasión de obstáculos mediante ajustes en la actualización de velocidades y la influencia de los vecinos dentro del enjambre.

CAPÍTULO 3

Justificación

Los algoritmos de inteligencia de enjambre son un área de la Inteligencia Computacional que toma inspiración del comportamiento colectivo de ciertas especies que se coordinan para realizar tareas que serían imposibles o tomarían mucho tiempo si las trataran de hacer por sí solas. A lo largo de los años se ha desarrollado una gran variedad de algoritmos, entre ellos PSO y ACO, los cuales han sido utilizados con éxito debido a su simplicidad y efectividad. A pesar de que estos algoritmos han sido utilizados con éxito para resolver una variedad de problemas de robótica de enjambre, existen otros algoritmos bioinspirados que no se han trabajado en la UVG. Estos nuevos algoritmos podrían ofrecer ventajas para algunas aplicaciones o simplemente ser igualmente viables.

Con este trabajo se logró implementar y validar dos algoritmos de inteligencia de enjambre como alternativas al PSO y ACO, demostrando su capacidad para resolver problemas de optimización y otros de interés para la Ingeniería Mecatrónica. Esto es importante porque de esta manera se estaría ampliando el repertorio de técnicas disponibles en la universidad para aplicaciones de robótica de enjambre.

Los resultados de esta investigación son relevantes porque muestran que es posible hacer uso de estos algoritmos en escenarios del mundo real. De esta manera, el trabajo no solo está demostrando que se tienen técnicas alternativas, sino que también está creando las bases para trabajos futuros que involucren experimentos con robótica colectiva o de control autónomo.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar y evaluar algoritmos de inteligencia de enjambre como alternativas del PSO y ACO para aplicaciones en Ingeniería Mecatrónica.

4.2. Objetivos específicos

- Investigar algoritmos de inteligencia de enjambre distintos a los usados en años anteriores y sus posibles aplicaciones en Ingeniería Mecatrónica.
- Implementar los algoritmos y realizar simulaciones simples en el entorno Matlab.
- Evaluar los algoritmos utilizando robots físicos simulados en el software Webots.
- Desarrollar escenarios prácticos para las distintas aplicaciones investigadas, y validar la implementación de los algoritmos mediante simulaciones realistas.

CAPÍTULO 5

Definición del problema

El presente trabajo de graduación se enfocó en la implementación y validación de algoritmos de inteligencia de enjambre distintos a PSO y ACO. Se trabajó con el algoritmo del murciélago y el algoritmo de la luciérnaga, con el fin de evaluar su desempeño en aplicaciones de Ingeniería Mecatrónica.

El alcance incluyó la investigación de los algoritmos seleccionados, su implementación en Matlab y su evaluación en pruebas simples. Primero se realizaron pruebas con funciones de costo para validar la capacidad de convergencia de los algoritmos. Luego, se aplicaron estos algoritmos para resolver problemas de generación de trayectorias en una variedad de escenarios con obstáculos. Por último, estos algoritmos de generación de trayectorias se validaron en el entorno simulado de Webots, mediante robots diferenciales. El trabajo se restringió a pruebas en simulación, sin llegar a pruebas en físico.

CAPÍTULO 6

Marco teórico

6.1. Inteligencia computacional de enjambre

La inteligencia de enjambre (*swarm intelligence*) parte de la idea de que múltiples agentes tratan de alcanzar un objetivo global a partir de un intercambio de información local. Este principio de cooperación logra que el sistema tenga un comportamiento más eficiente y robusto que sus partes individuales. Al trasladar este concepto al ámbito computacional, se utiliza el término *Computational Swarm Intelligence* (CSI) para referirse a algoritmos que modelan y aprovechan dicho comportamiento colectivo [10].

Los algoritmos de enjambre reciben su inspiración de fenómenos observados en la naturaleza, donde especies como hormigas, abejas, aves o peces muestran patrones de organización sin necesidad de un control centralizado. Estos patrones normalmente pueden traducirse a funciones matemáticas que guían a una población de soluciones en un espacio de búsqueda, lo que permite encontrar de manera aproximada la solución a un problema de optimización.

6.2. Algoritmo del murciélago

El algoritmo del murciélago (*bat algorithm*) es un algoritmo de optimización metaheurístico bioinspirado propuesto por Xin-She Yang, basado en la ecolocalización de los murciélagos y cómo la usan para comunicarse y moverse por su ambiente. El algoritmo está diseñado tanto para la exploración global del espacio de búsqueda como para la explotación local de las soluciones prometedoras [11].

El modelo propuesto consiste en las siguientes reglas o asunciones sobre el comportamiento de los murciélagos:

1. Todos los murciélagos usan ecolocalización para detectar distancias y distinguir entre obstáculos y presas.

2. Los murciélagos vuelan de forma aleatoria con una velocidad v_i hacia una posición x_i con una frecuencia de f, mientras se varía el volumen A_0 para buscar a su presa. También pueden ajustar de forma automática la frecuencia de sus pulsos y la tasa de emisión de pulsos $r \in [0, 1]$ a partir de qué tan cerca están del objetivo.
3. Se asume que el volumen de los pulsos varía desde un valor muy grande positivo A_0 hasta un mínimo constante A_{min}.

Cada murciélago está representado por una posible solución en el paso o instante t, que se actualiza a partir de las siguientes ecuaciones:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_*)f_i \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{3}$$

$\beta \in [0, 1]$ es un valor aleatorio que se toma de una distribución uniforme y x_* representa la mejor solución encontrada hasta ahora al comparar todas las soluciones de los murciélagos. Adicionalmente, se introduce una búsqueda local basada en pequeños desplazamientos aleatorios si se cumplen ciertas condiciones dependientes de la tasa de pulsos r_i:

$$x_{new*} = x_* + \epsilon A_i^t \tag{4}$$

En esta ecuación, $\epsilon \in [-1, 1]$ es un número aleatorio y A_i representa el volumen del murciélago, el cual decae gradualmente durante la ejecución del algoritmo. Si la nueva solución encontrada x_{new*} es mejor que la que la solución x_* que se tenía entonces se reemplaza $x*$ por x_{new*}. El volumen A_i y la tasa de pulsos r_i se actualizan a partir de las siguientes ecuaciones:

$$A_i^{t+1} = \alpha A_i^t \tag{5}$$

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)] \tag{6}$$

En donde $\alpha \in [0, 1]$ y $\gamma > 0$. Esto permite que los murciélagos exploren globalmente al inicio (cuando A_i es grande y r_i es pequeño) y gradualmente se enfoquen en zonas prometedoras a medida que se ajustan los parámetros. El pseudocódigo original se encuentra en el Anexo 15.1 como Algoritmo 2.

6.3. Algoritmo de la luciérnaga

El algoritmo de la luciérnaga (*firefly algorithm*) es otro algoritmo de optimización metaheurístico bioinspirado, también propuesto por Xin-She Yang. Está basado en la iluminación de las luciérnagas y cómo estas la usan para comunicarse entre sí [12].

El algoritmo parte de tres idealizaciones sobre el comportamiento de las luciérnagas:

1. Cualquier luciérnaga puede ser atraída por otra independientemente del sexo.
2. La atracción entre luciérnagas está determinada por su intensidad luminosa, que decrece con la distancia. Una luciérnaga menos brillante se moverá hacia una más brillante.
3. La intensidad de la luz es proporcional a la calidad de la solución, es decir, a la función objetivo del problema de optimización.

La intensidad de la luz $I(r)$ y la atracción $\beta(s)$ se modelan de esta manera:

$$I(r) = I_0 e^{-\gamma r^2} \tag{7}$$

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{8}$$

r es la distancia entre dos luciérnagas. γ es el coeficiente de absorción de luz. β_0 es la atracción máxima cuando la distancia es 0, e I_0 es la intensidad base. Es importante mencionar que, al implementarse el algoritmo como tal, la función de intensidad es reemplazada por una función de costo específica; en este caso, solo se define la función de intensidad porque la atracción de una luciérnaga es proporcional a la intensidad de luz.

La posición de una luciérnaga i al moverse hacia otra j más brillante se actualiza a partir de la siguiente ecuación:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r^2}(x_j^t - x_i^t) + \alpha \epsilon_i \tag{9}$$

ϵ_i es un vector aleatorio generado de una distribución gaussiana y α, el coeficiente de aleatoriedad. Este modelo garantiza que la exploración global este controlada por el parámetro α, mientras que la intensificación local se da cuando una luciérnaga se acerca a una más brillante. Este proceso les permite a las luciérnagas buscar zonas prometedoras dentro del espacio solución a partir de un equilibrio entre atracción y aleatoriedad. El pseudocódigo original se encuentra en el Anexo 15.1 como Algoritmo 3.

6.4. Funciones de costo

Para evaluar la efectividad de los algoritmos de enjambre, normalmente se utilizan funciones de costo. Estas pueden presentar un único mínimo o varios mínimos y máximos [10].

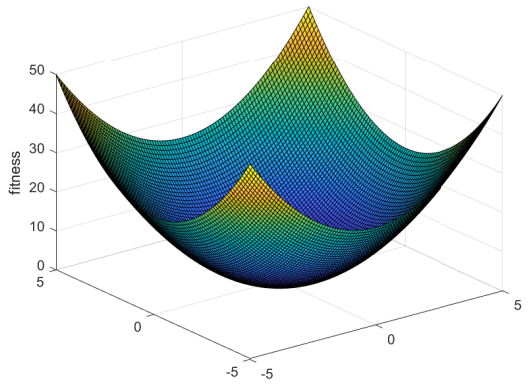
A cada uno de los agentes, en este caso los murciélagos o las luciérnagas, se les asigna un costo al evaluarlos en la función de costo. A partir de eso se puede determinar qué agente tiene el mejor costo o se encuentra más cerca del mínimo global, que normalmente es un dato muy importante a lo largo de las muchas iteraciones que los algoritmos de inteligencia de enjambre realizan. A continuación se muestran algunas funciones de costo:

6.4.1. Función Sphere

La función Sphere (Figura 1) se puede definir con la siguiente ecuación, en donde d representa la dimensión del problema, solo tiene un mínimo global y es una de las funciones de costo más sencillas y comunes:

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{10}$$

Figura 1. Función de costo Sphere.



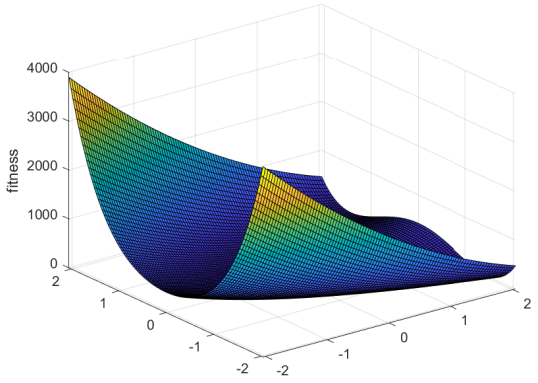
Nota. Elaboración propia.

6.4.2. Función Rosenbrock

La función Rosenbrock o valle (Figura 2) se presenta a continuación y tiene solo un mínimo global:

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{11}$$

Figura 2. Función de costo Rosenbrock.



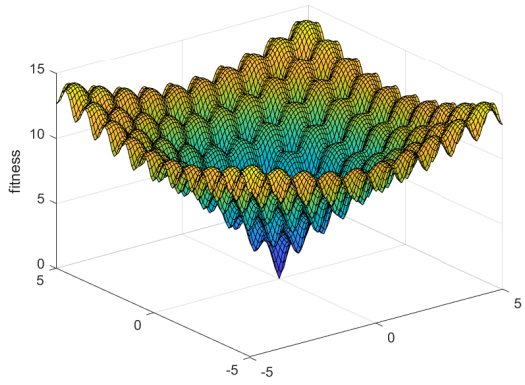
Nota. Elaboración propia.

6.4.3. Función Ackley

La función Ackley (Figura 3) también es bastante usada, ya que solo tiene un mínimo global, pero al mismo tiempo tiene muchos mínimos locales que ayudan a probar si los algoritmos se quedan atrapados en estos:

$$f(x) = -a(e^{-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}}) - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)} + a + e \tag{12}$$

Figura 3. Función de costo Ackley.



Nota. Elaboración propia.

6.4.4. Función Cross-in-Tray

La función Cross-in-Tray presenta múltiples mínimos globales, por lo que es bastante útil para ver si los algoritmos se confunden o no pueden converger debido a que hay más de una solución. Se puede ver en la Figura 22 que se encuentra en el Anexo 15.2.

$$f(x,y) = -0.0001\left(\left|\sin(x)\sin(y)e^{\left|100-\frac{\sqrt{x^2+y^2}}{\pi}\right|}\right|+1\right)^{0.1} \tag{13}$$

6.4.5. Función Eggcrate

La función Eggcrate también presenta múltiples mínimos locales distribuidos uniformemente, pero solo uno global, lo que es bastante útil para evaluar la habilidad de exploración de los algoritmos. Se puede ver en la Figura 23 que se encuentra en el Anexo 15.2.

$$f(x,y) = x^2 + y^2 + 25\left(\sin^2 x + \sin^2 y\right) \tag{14}$$

6.4.6. Función Beale

La función Beale tiene un mínimo global, pero varias regiones con pendientes suaves y curvas pronunciadas, lo que la hace útil para evaluar la capacidad de los algoritmos de escapar mínimos locales. Se puede ver en la Figura 24 que se encuentra en en el Anexo 15.2.

$$f(x,y) = \left(1.5 - x + xy\right)^2 + \left(2.25 - x + xy^2\right)^2 + \left(2.625 - x + xy^3\right)^2 \tag{15}$$

6.5. Robots móviles

Los robots móviles se caracterizan por su capacidad de moverse a través del entorno. Estos pueden ser robots caminadores, aéreos, acuáticos o con ruedas. Los robots móviles con ruedas son bastante usados para llevar a cabo pruebas y en investigación debido a que su diseño es fácil de modelar.

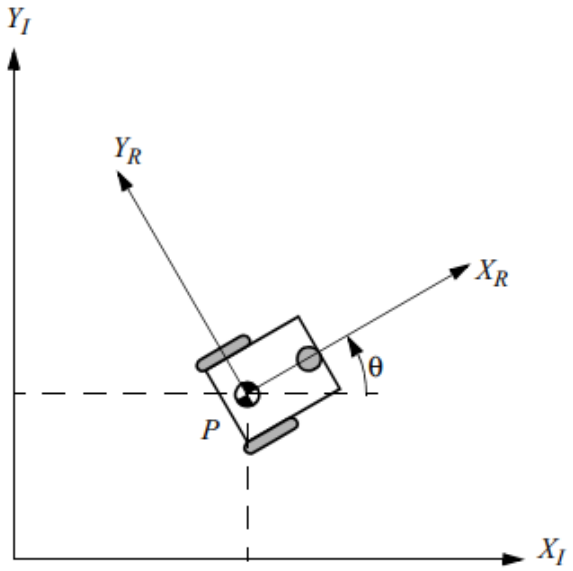
6.6. Modelado de robots móviles con ruedas

Los robots móviles con ruedas pueden modelarse sólo a partir de su cinemática. Este tipo de robot presenta restricciones no integrables. En otras palabras, si se integrara el recorrido, se perdería la información sobre el punto de partida. Estas restricciones también se les llama restricciones no holonómicas, y se presentan como restricciones de velocidad en la configuración. Como se quiere la cinemática del robot y al mismo tiempo establecer restricciones para la velocidad, se utiliza cinemática diferencial para los robots móviles [13].

Para los robots móviles no omnidireccionales, se tiene una condición de no deslizamiento, en la cual se establece que los robots no se pueden desplazar hacia los lados. Esto se puede ver en el marco de referencia del robot, donde las velocidades se representan de la siguiente manera:

$$B\dot{\xi} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \\ 0 \\ w \end{bmatrix} \tag{16}$$

Figura 4. Marco de referencia local del robot



Nota. El marco de referencia global y el marco de referencia local del robot. Imagen obtenida de [17].

En donde v es la velocidad lineal y w es la velocidad angular. Estas velocidades se refieren a las del marco de referencia del robot, las cuales se puede relacionar con las velocidades en el marco de referencia inercial mediante el jacobiano de θ [14]. Entonces, la cinemática diferencial del robot sería

$$I_{\dot{\xi}} = J(\theta)B_{\dot{\xi}} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 0 \\ w \end{bmatrix} \tag{17}$$

Al expandir esta expresión para cinemática diferencial, se obtiene el siguiente modelo:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\operatorname{sen}\theta \\ \dot{\theta} &= w \end{aligned} \tag{18}$$

El cual se conoce como el modelo uniciclo para cinemática de robots móviles.

6.6.1. Modelo uniciclo

El modelo de uniciclo se usa para describir el movimiento de un robot móvil en el plano. Se toma un punto ubicado en el centro del robot con coordenadas (x, y) y un ángulo de orientación θ. Este modelo es dinámico no lineal, y a partir de él se plantea una metodología de control en donde se le hace control al uniciclo y luego se mapea a un sistema más complejo [15]. El movimiento se describe en términos de la velocidad lineal v y la velocidad angular w. Este modelo toma en consideración las restricciones no holonómicas del robot, ya que la dirección de movimiento siempre está alineada con su orientación [16].

6.6.2. Modelo de robot diferencial

En este modelo se asume que el robot está compuesto por dos ruedas motrices paralelas de radio r, separadas una distancia L. También se asume que las ruedas giran con velocidades angulares $\dot{\varphi}_R$ (derecha) y $\dot{\varphi}_L$ (izquierda). Con esto se obtiene la velocidad lineal y angular del robot [17]:

$$\begin{aligned} v &= \frac{r(\dot{\varphi_R}+\dot{\varphi_L})}{2} \\ w &= \frac{r(\dot{\varphi_R}-\dot{\varphi_L})}{2L} \end{aligned} \tag{19}$$

Si se toma en cuenta lo anterior, se define el mapeo inverso que relaciona las velocidades del uniciclo con las del robot diferencial de la siguiente manera:

$$\begin{aligned} \dot{\varphi_R} &= \frac{v+Lw}{r} \\ \dot{\varphi_L} &= \frac{v-Lw}{r} \end{aligned} \tag{20}$$

6.7. Controladores de velocidad y posición de robots diferenciales

Debido a que el modelo es no holonómico, normalmente se emplean controladores basados en la cinemática del robot que actúan sobre la velocidad lineal v y la angular w. Entre las técnicas más comunes destacan los controladores PID, los cuales nos permiten controlar cada una de las velocidades a partir del error de posición y orientación del robot [18].

6.7.1. Control PID de posición y orientación

En este caso se define un controlador individual para cada una de las velocidades a partir de una posición deseada (x_g, y_g) y una orientación deseada θ_g [19]. El error de posición puede expresarse como:

$$e_p = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{21}$$

Y la orientación deseada y el error de orientación como:

$$\begin{aligned} \theta_g &= \arctan(\tfrac{y_g - y}{x_g - x}) \\ e_o &= \theta_g - \theta \end{aligned} \tag{22}$$

La implementación del control PID de posición y orientación en el dominio del tiempo con k_p, k_i y k_d como constantes se define de la siguiente manera:

$$\begin{aligned} v = PID(e_p) &= k_{p_p} e_p + k_{I_p} \int_0^t e_p(\tau) d\tau + k_{d_p} \dot{e_p} \\ w = PID(e_o) &= k_{p_o} e_o + k_{I_o} \int_0^t e_o(\tau) d\tau + k_{d_o} \dot{e_o} \end{aligned} \tag{23}$$

6.7.2. Control PID de acercamiento exponencial

En esta variación del control anterior se corrige la convergencia en forma de espiral que se ve en un controlador PID ordinario. El controlador para la orientación se mantiene igual, pero ahora para la velocidad lineal se agrega un coeficiente α que hace que la velocidad disminuya a medida que el robot se acerca a la meta [20]. La nueva ecuación para calcular la velocidad es:

$$v = k(e_p) e_p = \frac{v_0 (1 - e^{-\alpha e_p^2}) e_p}{e_p} \tag{24}$$

v_0 es la velocidad lineal máxima del robot.

6.8. Planificación de trayectorias y evasión de obstáculos

La planificación de trayectorias es un tema bastante común en la robótica móvil, ya que consiste en definir una ruta que un robot tendrá que seguir para ir de un punto hacia otro, mientras se toman en cuenta las restricciones del robot y del espacio en que se encuentra. Normalmente, para estas trayectorias se busca reducir la distancia recorrida, el tiempo de trayecto y la cantidad de movimientos, mientras se trata de evitar colisionar con objetos dentro del espacio de trabajo [14].

6.8.1. Conceptos generales

La planificación de trayectorias puede ser de dos tipos:

- Planificación global: el mapa o terreno es conocido antes de generar la trayectoria completa.

- Planificación local: en este caso, la trayectoria se genera por partes en tiempo real a medida que el agente avanza y no es necesario conocer el terreno de antemano.

También existen métodos híbridos que combinan los dos tipos, donde se integra los algoritmos de busqueda global en estrategias de control local. Algunos de los algoritmos para planificación local más conocidos son A*, D* y RRT (*Rapidly-exploring random tree*) [21].

6.8.2. Evasión de obstáculos

La evasión de obstáculos por lo general va de la mano con la planificación de trayectorias, ya que los obstáculos se usan para limitar las regiones en las que el robot se puede mover dentro del espacio de trabajo. Esta zona resultante se conoce como el espacio de configuración del robot, donde cada punto describe una posible configuración para el robot, y las configuraciones que implican colisión con un obstáculo se excluyen del conjunto de soluciones [21].

CAPÍTULO 7

Validaciones simples del algoritmo del murciélago y de la luciérnaga

En este capítulo se encuentra el procedimiento que se siguió para hacer la validación de los dos algoritmos. Esto incluye la elección de los parámetros óptimos, los escenarios en los que se evaluaron los algoritmos y los resultados de la evaluación. Las especificaciones de la computadora que se usó para todas las pruebas, experimentos y simulaciones se encuentran en el Cuadro 21, que se encuentra en el Anexo 15.3. Todo el código desarrollado durante este trabajo se encuentra en un repositorio cuyo link está en el Anexo 15.4.

7.1. Elección de parámetros

Para elegir los parámetros óptimos de cada algoritmo, se hicieron dos barridos de parámetros, el primero con los rangos de valores más amplios y el segundo más fino. Para cada barrido de parámetros, se ejecutó el algoritmo con cada posible combinación de parámetros, 5 veces para el primer barrido y 10 para el segundo, y se usaron 5 funciones de costo diferentes.

7.1.1. Barrido de parámetros para el algoritmo del murciélago

Los parámetros que se variaron durante el primer barrido y el rango en que se variaron se encuentran en el Cuadro 1. Y las funciones de costo que se usaron para este barrido fueron la función Sphere, la función Cross-in-Tray, la función Eggcrate, la función Beale y la función Ackley.

Para determinar la mejor combinación de parámetros, se calculó el promedio y la desviación estándar del costo obtenido por las 5 repeticiones que se hicieron por combinación. En este caso, como el algoritmo trata de minimizar, se busca que el promedio sea lo menor

posible, al igual que la desviación estándar, porque eso nos indica que la combinación de parámetros funciona consistentemente. Y luego, para poder comparar los resultados obtenidos al evaluar las combinaciones con distintas funciones de costo, estos se normalizaron. En el Cuadro 2 se pueden ver las 10 mejores combinaciones y, a partir de estos resultados, se redujeron los rangos y se aumentaron las repeticiones por combinación para el segundo barrido.

Cuadro 1. Rangos de barrido 1, algoritmo del murciélago.

Parámetro	Rango	Paso
α	(0.1,0.9)	0.1
γ	(0.1,0.9)	0.1
Cantidad de murcielagos	(20,80)	20

Nota. Valores variados en el primer barrido de parámetros para el algoritmo del murciélago. Elaboración propia.

Cuadro 2. Resultados de barrido 1, algoritmo del murciélago.

α	γ	Promedio	Desviación Estándar
0.5	0.9	0.018002	0.044814
0.7	0.8	0.022075	0.047158
0.7	0.7	0.022953	0.045724
0.6	0.5	0.024037	0.053239
0.7	0.3	0.030313	0.052565
0.3	0.6	0.030935	0.09012
0.4	0.9	0.033702	0.071882
0.7	0.9	0.035351	0.089265
0.5	0.5	0.037026	0.13323
0.6	0.6	0.037065	0.09457

Nota. Las 10 mejores combinaciones obtenidas del primer barrido de parámetros para el algoritmo del murciélago ordenadas por promedio y desviación estándar. Elaboración propia.

A partir de estos resultados se eligieron nuevos rangos para realizar el segundo barrido de parámetros, en donde ahora cada combinación se ejecutará 10 veces en lugar de 5. Estos rangos los podemos ver en el Cuadro 3.

Cuadro 3. Rangos de barrido 2, algoritmo del murciélago.

Parámetro	Rango	Paso
α	(0.5,0.9)	0.1
γ	(0.5,0.9)	0.1
Cantidad de murcielagos	(20,60)	20

Nota. Valores variados en el segundo barrido de parámetros para el algoritmo del murciélago. Elaboración propia.

Los resultados obtenidos en este segundo barrido de parámetros los podemos ver en el

Cuadro 4. A partir de esto podemos observar que el mejor valor para α es 0.6 y el mejor valor para γ es 0.7. En el caso de la cantidad de murciélagos, mientras sea más grande, se tendrán resultados más precisos, pero el tiempo de ejecución aumenta significativamente. Por esa razón, para la mayoría de las pruebas que se hicieron en los siguientes capítulos, se mantuvo la cantidad de murciélagos entre 20 y 40.

Cuadro 4. Resultados de barrido 2, algoritmo de murciélago.

α	γ	**Promedio**	**Desviación Estándar**
0.6	0.7	0.099546	0.13282
0.7	0.7	0.10393	0.11374
0.6	0.8	0.13818	0.25905
0.5	0.8	0.14097	0.28036
0.6	0.5	0.14796	0.1971
0.7	0.8	0.16153	0.2666
0.8	0.6	0.17056	0.19963
0.6	0.9	0.17248	0.22359
0.6	0.6	0.17595	0.2325
0.7	0.6	0.18642	0.24359

Nota. Las 10 mejores combinaciones obtenidas del segundo barrido de parámetros para el algoritmo del murciélago ordenadas por promedio y desviación estándar. Elaboración propia.

7.1.2. Barrido de parámetros para el algoritmo de la luciérnaga

Para determinar los mejores parámetros de este algoritmo, se siguió el mismo procedimiento que en la sección 7.1.1. Los parámetros a variar se presentan en el Cuadro 5.

Cuadro 5. Rangos de barrido 1, algoritmo de la luciérnaga.

Parámetro	**Rango**	**Paso**
α	(0.2,1.0)	0.2
γ	(0.01, 1.0)	10^x
θ	(0.85, 1.0)	0.5
Cantidad de Luciérnagas	(20,80)	20

Nota. Valores variados en el primer barrido de parámetros para el algoritmo de la luciérnaga. Elaboración propia.

Los resultados se pueden ver en el Cuadro 6. Igual que para el algoritmo anterior, se muestran las 10 mejores combinaciones a partir de su promedio y desviación estándar de las soluciones. A partir de estos resultados se redujeron los rangos y cada combinación se ejecutó 10 veces. Los nuevos rangos están en el Cuadro 7.

Los resultados de este segundo barrido se encuentran en el Cuadro 8. Por último, a partir de estos resultados, se puede ver que el mejor valor para α es 0.6, el mejor valor para γ es 0.01 y el mejor valor para θ es 1.0.

Cuadro 6. Resultados de barrido 1, algoritmo de la luciérnaga.

α	γ	θ	**Promedio**	**Desviación Estándar**
0.8	0.01	1.0	0.0023551	0.0052772
0.6	1.0	1.0	0.014778	0.055795
0.8	1.0	1.0	0.014967	0.056946
0.8	0.1	1.0	0.01526	0.056145
1.0	0.1	0.9	0.017643	0.036584
0.4	0.1	1.0	0.019052	0.047812
1.0	0.01	1.0	0.019932	0.071122
1.0	0.01	0.85	0.020431	0.036887
0.8	0.1	0.85	0.020712	0.026246
1.0	1.0	0.95	0.022163	0.057154

Nota. Las 10 mejores combinaciones obtenidas del primer barrido de parámetros para el algoritmo de la luciérnaga ordenadas por promedio y desviación estándar. Elaboración propia.

Cuadro 7. Rangos de barrido 2, algoritmo de la luciérnaga.

Parámetro	**Rango**	**Paso**
α	(0.6,1.0)	0.1
γ	(0.01, 1.0)	10^x
θ	(0.9, 1.0)	0.5
Cantidad de luciérnagas	(20,60)	20

Nota. Valores variados en el segundo barrido de parámetros para el algoritmo de la luciérnaga. Elaboración propia.

Cuadro 8. Resultados de barrido 2, algoritmo de la luciérnaga.

α	γ	θ	**Promedio**	**Desviación Estándar**
0.6	0.01	1.0	0.0069582	0.012538
0.9	0.01	1.0	0.017125	0.031282
0.7	0.01	1.0	0.026166	0.055181
1.0	0.01	1.0	0.034489	0.058106
0.8	0.01	0.9	0.042341	0.10463
0.9	0.1	1.0	0.050807	0.10534
0.8	0.01	0.95	0.06513	0.13692
0.6	1.0	1.0	0.069603	0.13692
1.0	0.1	1.0	0.074511	0.13753
0.9	1.0	1.0	0.076779	0.13712

Nota. Las 10 mejores combinaciones obtenidas del segundo barrido de parámetros para el algoritmo de la luciérnaga ordenadas por promedio y desviación estándar. Elaboración propia.

7.2. Validación con funciones de costo

Para realizar las validaciones de los algoritmos con los parámetros encontrados, se eligió una variedad de funciones de costo y se midió el tiempo y la cantidad de iteraciones que le tomó al algoritmo converger al mínimo global de las funciones de costo.

7.2.1. Pruebas con función Sphere

Cada uno de los algoritmos se ejecutó 10 veces con una cantidad de murciélagos y luciérnagas de 20. Para observar la cantidad de iteraciones que le toma al algoritmo llegar al objetivo, en este caso, se decidió tener un error del 5 %. Los resultados se pueden ver en el Cuadro 9.

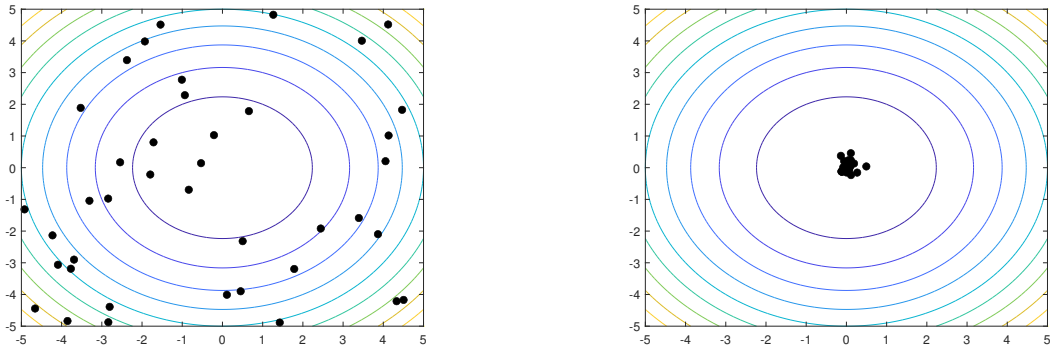
Cuadro 9. Resultados de prueba con función Sphere.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Iteraciones	Corrida	t(s)	Iteraciones
1	0.75	35	1	0.07	2
2	0.66	30	2	0.05	2
3	0.56	27	3	0.05	2
4	2.43	90	4	0.06	2
5	0.43	21	5	0.05	2
6	0.44	21	6	0.05	2
7	0.64	30	7	0.05	2
8	1.46	61	8	0.06	2
9	0.63	30	9	0.06	2
10	0.89	41	10	0.05	2

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con la función de costo Sphere. Elaboración propia.

De estas pruebas podemos observar que al algoritmo de la luciérnaga le tomó de forma consistente solo 2 iteraciones para converger y el tiempo no fue mayor a 0.07 segundos. En cambio, el del murciélago llegó a necesitar hasta 90 iteraciones para converger y tuvo un máximo tiempo de ejecución de 2.43 segundos. En la Figura 5 se muestra la posición inicial y final de los puntos en una de las pruebas con el algoritmo del murciélago y la función Sphere.

Figura 5. Ejemplos de convergencia con función Sphere y el algoritmo del murciélago.



Nota. Un ejemplo donde se muestra el estado inicial de los murciélagos comparado con cuando convergen en el mínimo global de la función Sphere. Elaboración propia.

7.2.2. Pruebas con función Ackley

Para estas pruebas se usarán los mismos parámetros que se usaron en la sección anterior. Los resultados de estas pruebas los podemos ver en el Cuadro 10.

Cuadro 10. Resultados de prueba con función Ackley.

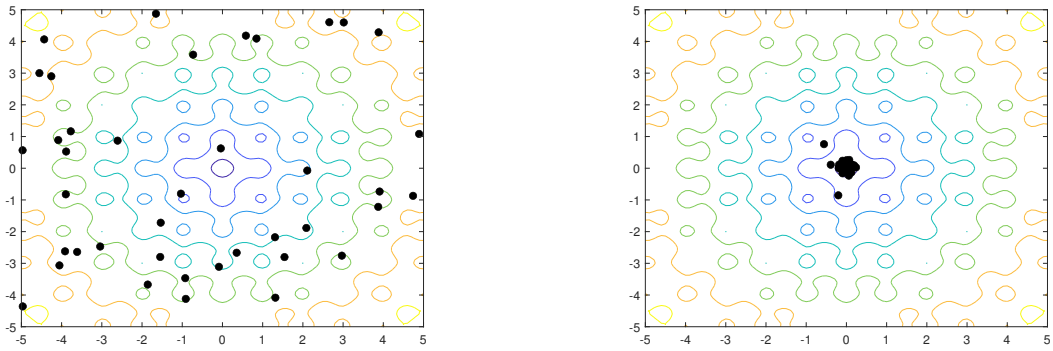
Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Iteraciones	Corrida	t(s)	Iteraciones
1	2.52	92	1	0.05	2
2	0.93	38	2	0.05	2
3	2.45	75	3	0.04	1
4	0.50	25	4	0.06	2
5	1.46	59	5	0.05	2
6	4.82	142	6	0.04	1
7	1.49	61	7	0.02	1
8	1.00	43	8	0.05	2
9	1.48	59	9	0.05	2
10	0.63	30	10	0.06	2

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con la función de costo Ackley. Elaboración propia.

En este caso, el promedio tanto de la cantidad de iteraciones como del tiempo de ejecución para el algoritmo del murciélago aumentó, pero eso es de esperarse, ya que la función Ackley presenta muchos mínimos locales que dificultan la convergencia de todos los murciélagos. Pero para el de la luciérnaga, el tiempo de ejecución y la cantidad de iteraciones necesarias para converger se mantuvo bastante similar y mejoró en algunos casos. En la Figura 6 se

muestra la posición inicial y final de los puntos en una de las pruebas con el algoritmo de la luciérnaga y la función Ackley.

Figura 6. Ejemplos de convergencia con función Ackley y el algoritmo de la luciérnaga.



Nota. Un ejemplo donde se muestra el estado inicial de las luciérnagas comparado con cuando convergen en el mínimo global de la función Ackley. Elaboración propia.

7.2.3. Pruebas con función Booth

La última función con la que se realizaron estas pruebas simples será la función Rosenbrock. Los parámetros para los dos algoritmos se mantienen igual que en las secciones anteriores. Los resultados se encuentran en el Cuadro 11.

Cuadro 11. Resultados de prueba con función Booth.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Iteraciones	Corrida	t(s)	Iteraciones
1	1.98	75	1	0.07	3
2	0.95	42	2	0.06	2
3	1.04	47	3	0.05	2
4	0.74	31	4	0.06	3
5	1.02	45	5	0.05	2
6	1.62	64	6	0.05	2
7	0.82	37	7	0.05	2
8	0.53	26	8	0.04	2
9	0.88	40	9	0.05	2
10	0.87	40	10	0.06	2

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con la función de costo Booth. Elaboración propia.

A partir de estas pruebas podemos ver que el comportamiento se mantiene bastante

similar. El algoritmo de la luciérnaga aún converge en 2 o 3 iteraciones y el del murciélago se mantiene en un rango de 2 o 3 segundos para converger y, la mayoría de las veces, no necesita más de 70 iteraciones para converger.

CAPÍTULO 8

Aplicación de los algoritmos de enjambre para planificación de trayectorias

En este capítulo se explica la lógica que se usó para la aplicación de planificación de trayectorias y las variaciones que se hicieron de la misma para tratar de mejorar los resultados que se obtuvieron inicialmente y lograr resolver problemas más complejos. También se incluyen las distintas pruebas que se hicieron con cada algoritmo, con las distintas variaciones que se desarrollaron para la aplicación de planificación de trayectorias y con una variedad de mapas.

8.1. Aplicación de planificación de trayectorias básica

Para esta aplicación se asume que se tiene una posición inicial, una posición objetivo y la cantidad de subdivisiones que tendrá la trayectoria. Si el mapa presentara obstáculos, también se necesitaría saber la posición de estos en el mapa. Con estos valores, ahora los murciélagos y las luciérnagas pasan de ser masas puntuales en el mapa a una secuencia de puntos que va de la posición inicial a la posición objetivo. La cantidad de puntos que tendrá cada trayectoria se determina a partir del número de subdivisiones y ahora el costo de cada uno de los agentes será la suma de la distancia entre cada uno de los puntos que conforman una trayectoria. Entonces, el algoritmo busca minimizar esta distancia.

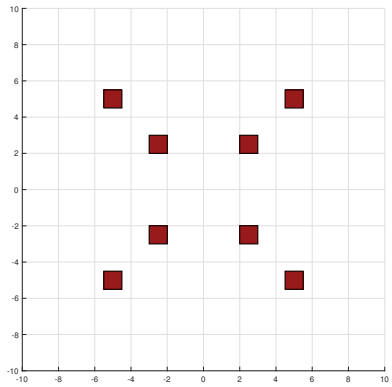
Por último, para evitar los obstáculos, se lleva una cuenta de la cantidad de veces que la trayectoria cruza a través de uno de los obstáculos y luego esta cuenta se multiplicará por un valor muy grande, en este caso se uso 10^6, con el fin de aumentar de forma artificial el costo de la solución y esto se suma a la distancia total. Esto se hace con el objetivo de que las trayectorias que tengan colisiones no se tomen en cuenta a pesar de presentar distancias más cortas al objetivo.

8.2. Pruebas básicas de planificación de trayectorias

8.2.1. Mapas usados en las pruebas

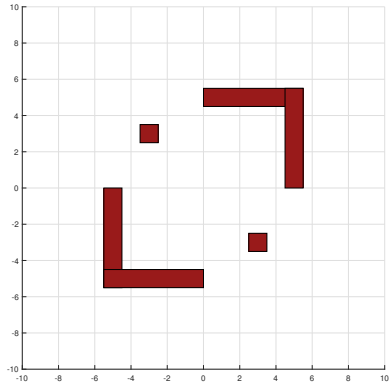
Para probar si el algoritmo funciona de manera satisfactoria, se diseñaron 3 mapas, los cuales se muestran a continuación. Todos los mapas usados en este trabajo para todas las pruebas de esta sección y las siguientes van de $(-10, 10)$ en los dos ejes. Estas son unidades de distancia arbitraria.

Figura 7. Mapa de pruebas A.



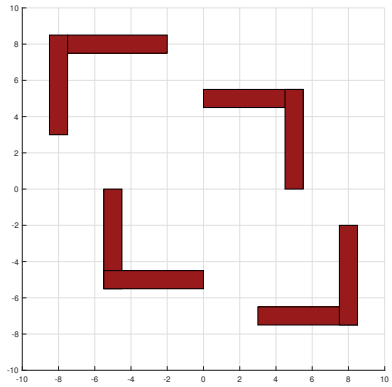
Nota. Mapa con obstaculos A usado en las pruebas para la aplicación de planificación de trayectorias. Elaboración propia.

Figura 8. Mapa de pruebas B.



Nota. Mapa con obstaculos B usado en las pruebas para la aplicación de planificación de trayectorias. Elaboración propia.

Figura 9. Mapa de pruebas C.



Nota. Mapa con obstaculos C usado en las pruebas para la aplicación de planificación de trayectorias. Elaboración propia.

8.2.2. Pruebas con el mapa A

Para estas pruebas se usaron los mismos parámetros para los dos algoritmos que se usaron en el capítulo anterior. La posición inicial es $(-9, -9)$, la posición objetivo es $(4, 4)$ y la cantidad de subdivisiones es 4. En el Cuadro 12, se muestran el tiempo que tomó generar y el costo de la trayectoria generada con el algoritmo del murciélago y el de la luciérnaga, 10 veces cada uno.

Cuadro 12. Resultados de prueba con mapa A.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	0.64	21.1	1	13.9	18.6
2	0.64	23.7	2	14.1	18.6
3	0.62	29.4	3	13.7	18.6
4	0.63	22.3	4	14.0	18.6
5	0.62	28.0	5	13.8	18.8
6	0.58	19.4	6	13.8	18.6
7	0.64	21.4	7	13.9	18.6
8	0.64	22.6	8	13.8	18.6
9	0.59	21.5	9	13.6	18.6
10	0.60	28.7	10	13.7	18.6

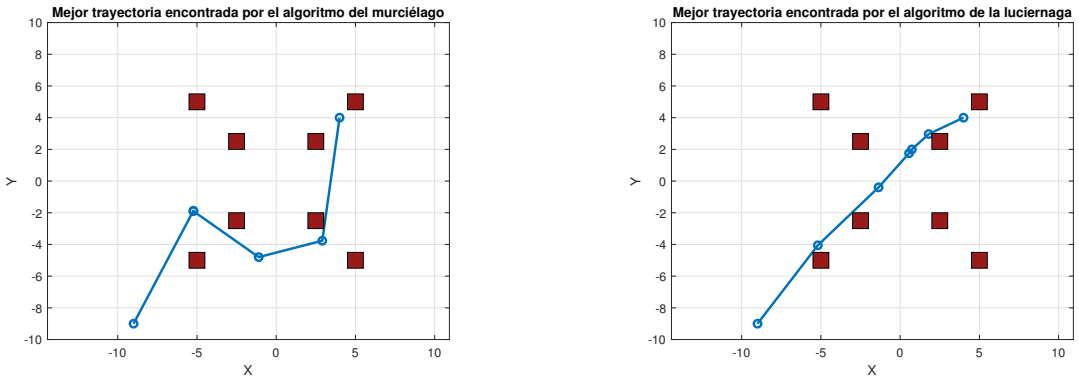
Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria para el mapa A de $(-9, -9)$ a $(4, 4)$. Elaboración propia.

Para estas pruebas se dejó correr los dos algoritmos por 200 iteraciones en lugar de definir un objetivo y parar cuando este se hubiera cumplido. Por esa razón, el tiempo de

ejecución del algoritmo de la luciérnaga aumentó significativamente, ya que ahora se realizan las iteraciones completas en lugar de parar al cumplir con el objetivo. En cuanto al costo, se puede apreciar que se mantuvo bastante consistente, casi sin variar.

En cuanto al algoritmo del murciélago, se puede ver que converge significativamente más rápido, pero presenta una mayor variación en cuanto al costo obtenido para las trayectorias. En la Figura 10 se muestran ejemplos de las trayectorias obtenidas por cada algoritmo para el mapa A.

Figura 10. Ejemplos de trayectorias para el mapa A.



Nota. Un ejemplo de las trayectorias encontradas para ir de $(-9, -9)$ a $(4, 4)$ en el mapa A por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

8.2.3. Pruebas con el mapa B

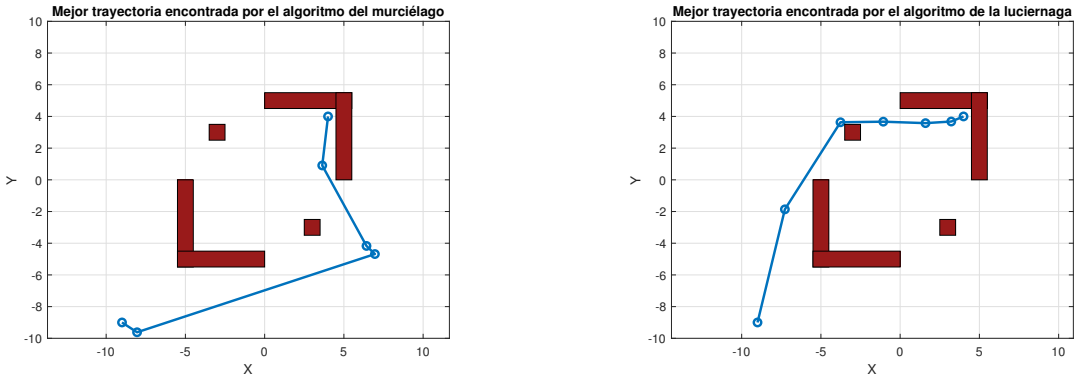
En estas pruebas se mantuvo la misma posición inicial, posición objetivo y cantidad de subdivisiones. En el Cuadro 13 se muestra el tiempo y costo de 10 trayectorias generadas con el algoritmo del murciélago y el de la luciérnaga.

Cuadro 13. Resultados de prueba con mapa B.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	0.49	19.9	1	10.5	19.9
2	0.46	31.5	2	10.8	19.9
3	0.55	30.1	3	10.9	19.9
4	0.48	32.6	4	10.6	19.9
5	0.48	22.3	5	10.6	19.9
6	0.52	24.8	6	10.6	32.6
7	0.49	28.3	7	10.8	19.9
8	0.47	39.8	8	11.7	19.9
9	0.51	22.8	9	10.8	20.0
10	0.48	20.2	10	10.4	20.0

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria para el mapa B de $(-9, -9)$ a $(4, 4)$. Elaboración propia.

Figura 11. Ejemplos de trayectorias para el mapa B.



Nota. Un ejemplo de las trayectorias encontradas para ir de $(-9, -9)$ a $(4, 4)$ en el mapa B por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

Después de estas dos pruebas se pueden notar ciertas tendencias, como que el algoritmo del murciélago converge significativamente más rápido, pero al mismo tiempo el costo obtenido de la trayectoria varía más de una ejecución a otra. Para que los resultados variaran menos entre sí, se podría aumentar la cantidad de murciélagos o la cantidad de iteraciones. Para estas pruebas se mantuvieron los mismos valores para que sean consistentes a lo largo de las pruebas. En cuanto al de la luciérnaga, se puede ver que este aún le toma mucho más tiempo en ejecutarse, pero el costo obtenido en las trayectorias básicamente no varía. En la Figura 11 se muestran ejemplos de las trayectorias obtenidas por cada algoritmo para el mapa B.

8.2.4. Pruebas con el mapa C

Igual que en la prueba anterior, se mantuvieron las mismas posiciones inicial, objetivo y cantidad de subdivisiones. En el Cuadro 14, se presentan los resultados de generar 10 trayectorias con el algoritmo del murciélago y el de la luciérnaga para resolver el mapa C.

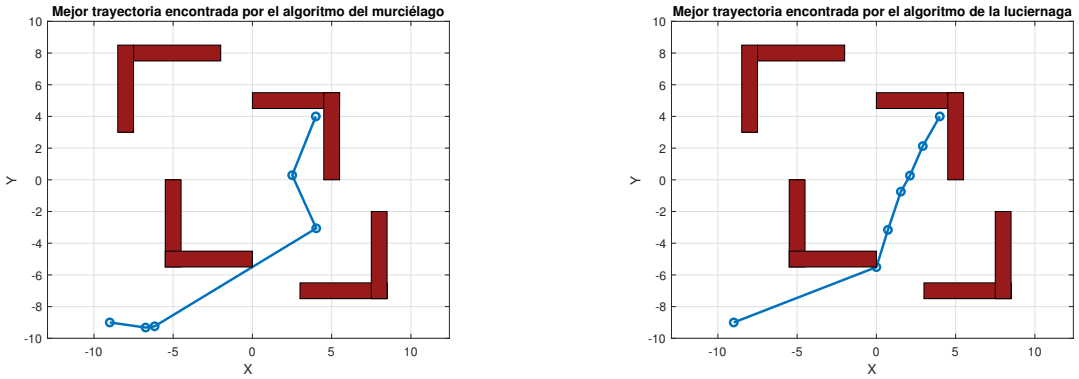
Cuadro 14. Resultados de prueba con mapa C.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	0.55	22.1	1	14.0	20.0
2	0.59	20.8	2	14.0	20.0
3	0.56	20.1	3	14.1	20.0
4	0.57	20.1	4	15.3	20.0
5	0.60	20.5	5	18.0	20.0
6	0.56	20.1	6	14.3	20.0
7	0.66	20.2	7	13.9	20.0
8	0.64	22.1	8	15.8	20.0
9	0.56	23.0	9	14.8	20.0
10	0.56	26.2	10	23.4	38.0

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria para el mapa C de $(-9, -9)$ a $(4, 4)$. Elaboración propia.

En estas pruebas, se ven resultados similares a los obtenidos con los otros mapas. El algoritmo del murciélago converge mucho más rápido que el de la luciérnaga, pero el costo de las trayectorias generado por el algoritmo del murciélago presenta más variación que los costos obtenidos por el de la luciérnaga. En la Figura 12, se puede ver ejemplos de las trayectorias por cada algoritmo para el mapa C.

Figura 12. Ejemplos de trayectorias para el mapa C.



Nota. Un ejemplo de las trayectorias encontradas para ir de $(-9, -9)$ a $(4, 4)$ en el mapa C por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

8.3. Planificación de subtrayectorias para escenarios más complejos

Al probar planificar trayectorias para mapas más complejos, la aplicación descrita en la sección anterior empezó a tener problemas para obtener una trayectoria sin colisiones y aumentar la cantidad de agentes, iteraciones o subdivisiones del trayecto no mejoraba las soluciones de forma significativa o no devolvía resultados consistentes. Para ayudar al algoritmo a poder resolver estos escenarios más complejos, se decidió que el problema se dividiría en varios problemas más pequeños. En términos simples, se dividiría la trayectoria en partes, por lo que se tendrán metas intermedias.

Aún se busca que la planificación de las trayectorias sea mayormente autónoma, pero ahora se necesitará definir una nueva variable, que es la cantidad de subtrayectorias que compondrán a la trayectoria completa. A partir de esto, el procedimiento para la planificación de las subtrayectorias se explica con el siguiente pseudocódigo:

Algoritmo 1 Planificación de subtrayectorias con puntos intermedios

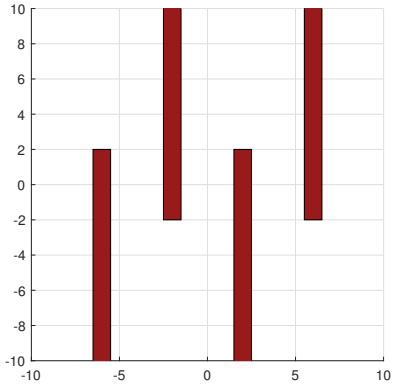
1: Determinar la distancia en x y y entre el punto inicial y el objetivo.
2: **Si** distancia en $x >$ distancia en y **Entonces**
3: Dividir el trayecto por el eje y.
4: **Sino**
5: Dividir el trayecto por el eje x.
6: **FinSi**
7: Para cada división, generar la coordenada faltante para determinar los puntos intermedios:
8: **Mientras** coordenada generada dentro de un obstáculo **Hacer**
9: Generar un nuevo número aleatorio dentro del mapa.
10: **FinMientras**
11: **Para** cada punto objetivo intermedio **Hacer**
12: Aplicar el método de planificación de trayectorias de la sección 8.1.
13: Cuando se alcanza un punto intermedio, este se convierte en el nuevo punto de inicio.
14: **FinPara**

8.4. Pruebas con aplicación de planificación de trayectorias compuestas

8.4.1. Mapas usados en las pruebas

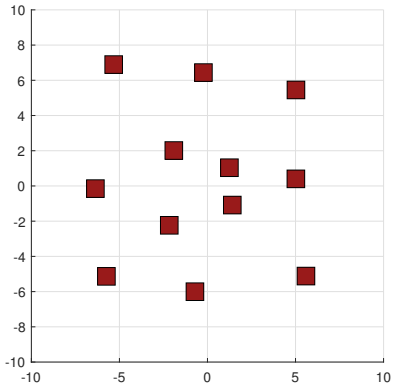
Para estas pruebas sólo se diseñaron 2 mapas. El primero muestra una serie de paredes que incentivan a un movimiento de zigzagueo (Figura 13) y el segundo presenta varios obstáculos pequeños que varían su posición con cada iteración (Figura 14). La variación es entre $(-1, 1)$ en los dos ejes.

Figura 13. Mapa de pruebas D.



Nota. Mapa con obstaculos D usado en las pruebas para la aplicación de planificación de trayectorias compuestas. Elaboración propia.

Figura 14. Mapa de pruebas E.



Nota. Mapa con obstaculos E usado en las pruebas para la aplicación de planificación de trayectorias compuestas. Elaboración propia.

8.4.2. Pruebas con el mapa D

Para estas pruebas se mantuvieron los mismos parámetros para los dos algoritmos: la posición inicial es $(-9, -9)$, la posición objetivo es $(9, 9)$, la cantidad de subdivisiones por subtrayectoria aún es 4 y el número de subtrayectorias para el mapa D será de 4. En el Cuadro 15 se muestran el tiempo que tomó generar y el costo de la trayectoria generada con el algoritmo del murciélago y el de la luciérnaga 10 veces cada uno.

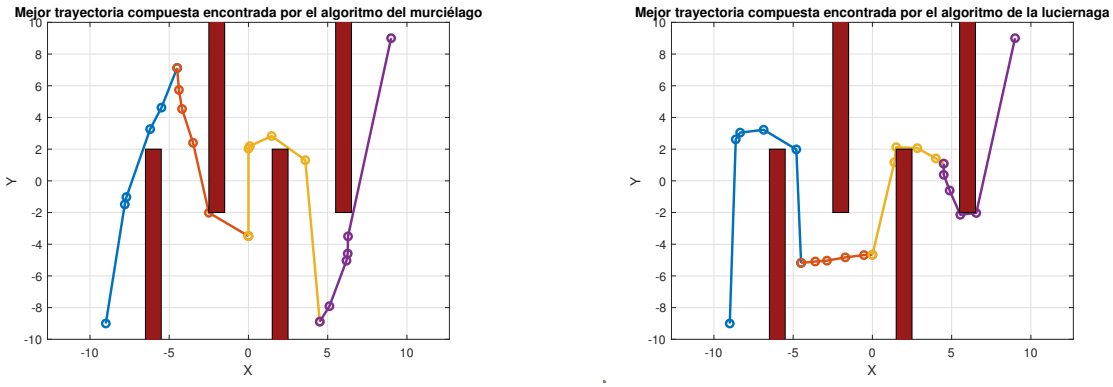
Cuadro 15. Resultados de prueba con mapa D.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	1.54	62.5	1	29.6	57.8
2	1.27	55.9	2	29.6	55.2
3	1.38	77.6	3	29.2	54.6
4	1.33	55.8	4	29.3	63.3
5	1.35	70.8	5	29.9	56.7
6	1.32	57.9	6	29.3	50.2
7	1.31	81.8	7	29.4	60.6
8	1.32	63.2	8	29.6	55.3
9	1.31	67.7	9	29.5	73.8
10	1.34	68.7	10	32.0	47.8

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria compuesta para el mapa D de $(-9, -9)$ a $(9, 9)$. Elaboración propia.

En estas pruebas, ninguno de los algoritmos mostró un comportamiento consistente en cuanto al costo total de las trayectorias. Esto tiene sentido debido al elemento aleatorio que se tiene al momento de seleccionar los puntos objetivo intermedios de las trayectorias. El algoritmo del murciélago aún se ejecuta mucho más rápido en comparación con el de la luciérnaga. También se observó que los dos algoritmos comenzaron a generar trayectorias con colisiones con mayor frecuencia en comparación con las pruebas hechas anteriormente. Este se podría mejorar al variar la cantidad de agentes, la cantidad de iteraciones o la cantidad de puntos que tendrá cada subtrayectoria, pero se mantuvieron los mismos parámetros que en las pruebas anteriores por consistencia. En la Figura 15 se muestran ejemplos de las trayectorias planificadas por el algoritmo del murciélago y el de la luciérnaga en esta prueba.

Figura 15. Ejemplos de trayectorias para el mapa D.



Nota. Un ejemplo de las trayectorias compuestas encontradas para ir de $(-9, -9)$ a $(9, 9)$ en el mapa D por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

8.4.3. Pruebas con el mapa E

Para estas pruebas, el punto de inicio es $(-9, -4)$, el punto objetivo es $(9, 4)$ y la cantidad de subtrayectorias es de 3 en lugar de 4; el resto de parámetros se mantuvo igual que en la prueba anterior. Los resultados se encuentran en el Cuadro 16.

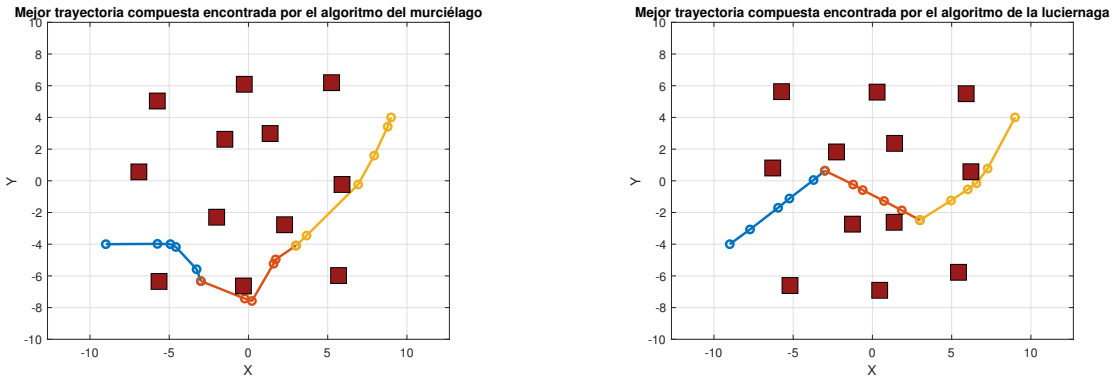
Cuadro 16. Resultados de prueba con mapa E.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	2.30	26.6	1	61.8	22.3
2	2.29	32.6	2	62.4	21.7
3	2.32	27.9	3	61.4	28.3
4	2.31	22.9	4	61.3	29.3
5	2.33	27.5	5	61.6	23.7
6	2.36	27.2	6	61.1	26.7
7	2.37	40.3	7	61.3	32.4
8	2.34	28.9	8	60.8	23.4
9	2.35	30.7	9	61.1	29.9
10	2.35	25.4	10	61.6	26.9

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria compuesta para el mapa E de $(-9, -4)$ a $(9, 4)$. Elaboración propia.

En esta prueba, el algoritmo para generar trayectorias compuestas presenta resultados que se parecen a los obtenidos en las pruebas con los mapas A, B y C. En estas pruebas, el costo de la trayectoria obtenido con el algoritmo de la luciérnaga se mantiene bastante consistente, pero le toma más tiempo converger y, en el caso del algoritmo del murciélago, el costo varía más de una corrida a la otra, pero le toma mucho menos tiempo en ejecutarse. Si se aumentara la cantidad de agentes e iteraciones, se obtendrían mejores resultados. En la Figura 16 se muestran ejemplos de las trayectorias generadas por el algoritmo del murciélago y el de la luciérnaga en esta prueba.

Figura 16. Ejemplos de trayectorias para el mapa E.



Nota. Un ejemplo de las trayectorias compuestas encontradas para ir de $(-9, -4)$ a $(9, 4)$ en el mapa E por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

Por último, para demostrar que se obtendrían mejores resultados con el algoritmo del murciélago si se aumentara la cantidad de murciélagos y la cantidad máxima de iteraciones. Para esto se repitió la prueba anterior, pero con 80 murciélagos y un máximo de 500 iteraciones. Los resultados se encuentran en el Cuadro 17.

Cuadro 17. Resultados de prueba con mapa E, al aumentar la cantidad de murciélagos y la máxima cantidad de iteraciones.

Algoritmo del murciélago		
Corrida	t(s)	Costo
1	25.6	30.4
2	26.5	25.8
3	25.7	23.5
4	23.0	27.8
5	22.9	22.2
6	23.5	28.2
7	26.0	27.4
8	22.8	39.5
9	25.0	26.7
10	22.5	22.7

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago con una trayectoria compuesta para el mapa E de $(-9, -4)$ a $(9, 4)$ con 80 murciélagos y una máxima cantidad de iteraciones de 500. Elaboración propia.

CAPÍTULO 9

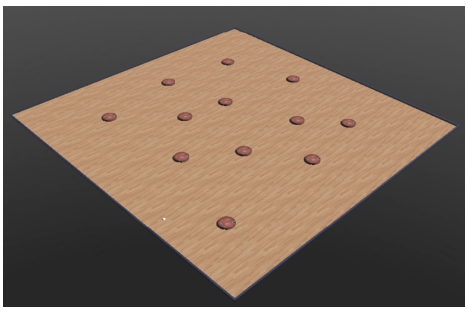
Pruebas con robots físicos simulados en Webots

En este capítulo se presentan resultados de pruebas similares a las presentadas en capítulos anteriores, pero ahora además de plainificar las trayectorias, se usará un controlador PID con acercamiento exponencial para controlar a un robot diferencial sencillo con el fin de que este recorra la trayectoria generada.

9.1. Mundo de Webots

Para el mundo de Webots se creó una plataforma de 20×20 m, pero desfasada -10 m en los dos ejes para que se parezca lo más posible a los mapas con los que se trabajó en los capítulos anteriores. Y los obstáculos, en lugar de ser cuadrados, serán barriles. En la Figura 17 se muestra el mapa E recreado en Webots.

Figura 17. Mapa de pruebas E en Webots.



Nota. Mapa con obstaculos E reecreado en el entorno de Webots. Elaboración propia.

9.2. Pruebas en Webots

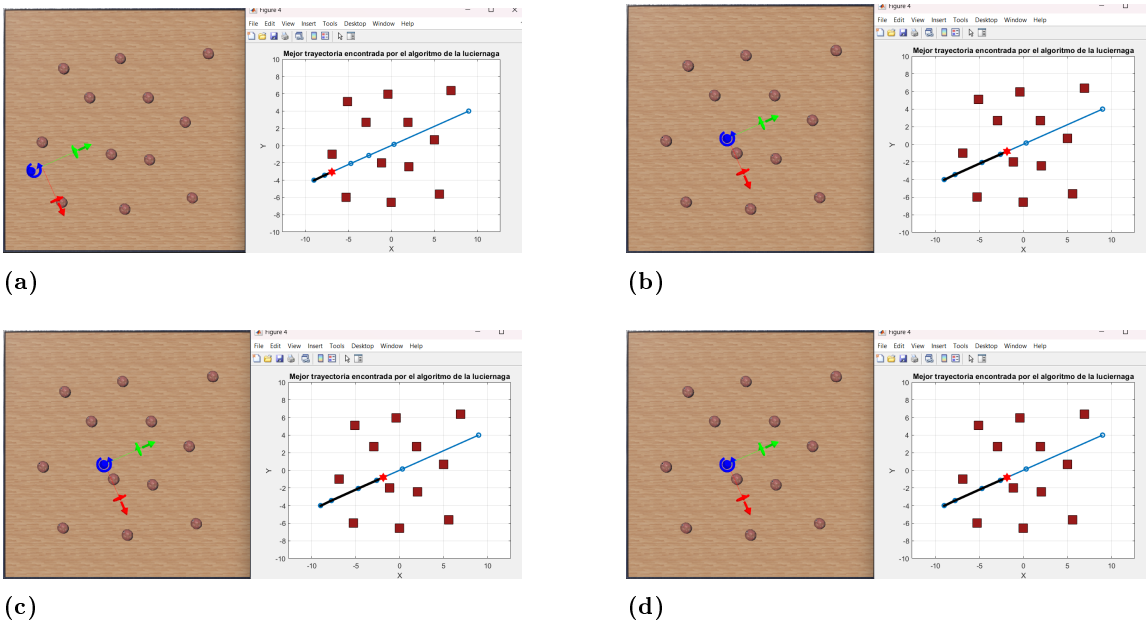
Para estas pruebas solo se usó el mapa E. Se tuvo una posición inicial de $(-9, -4)$, una posición objetivo de $(9, 4)$, 40 agentes, 5 subdivisiones por trayecto y, para las pruebas con la aplicación de planificación de trayectos compuestos, solo se harán 2 subtrayectos. Los demás parámetros para el algoritmo del murciélago y de la luciérnaga se mantienen iguales a los de las pruebas anteriores. En el Cuadro 18 se muestran los resultados para la aplicación de planificación de trayectorias sencillo, con la diferencia de que ahora el tiempo incluye también el tiempo que le toma al robot recorrer la trayectoria.

Cuadro 18. Resultados de prueba con mapa E en Webots, algoritmo sencillo.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	131.8	22.3	1	201.4	19.7
2	138.5	23.6	2	199.1	19.7
3	116.4	19.7	3	203.0	19.7
4	132.2	21.0	4	203.4	19.7
5	118.4	20.0	5	233.2	19.7
6	116.9	19.8	6	218.2	19.7
7	126.6	21.4	7	240.0	19.7
8	125.2	21.2	8	249.0	19.7
9	126.6	21.5	9	246.8	19.8
10	116.2	19.7	10	233.1	19.8

Nota. Las 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria para el mapa E en Webots de $(-9, -4)$ a $(9, 4)$. Elaboración propia.

Figura 18. Ejemplo de trayectoria para el mapa E en Webots.



Nota. Un ejemplo de la trayectoria encontrada para ir de $(-9, -4)$ a $(9, 4)$ en el mapa E por el algoritmo de la luciérnaga en Webots. Elaboración propia.

En la Figura 18 se puede ver un ejemplo de la trayectoria generada a partir del algoritmo del murciélago y cómo el robot se mueve por el mapa en Webots. Estas mismas pruebas se repitieron, pero con el algoritmo de generación de trayectorias compuestas; los resultados se muestran en el Cuadro 19.

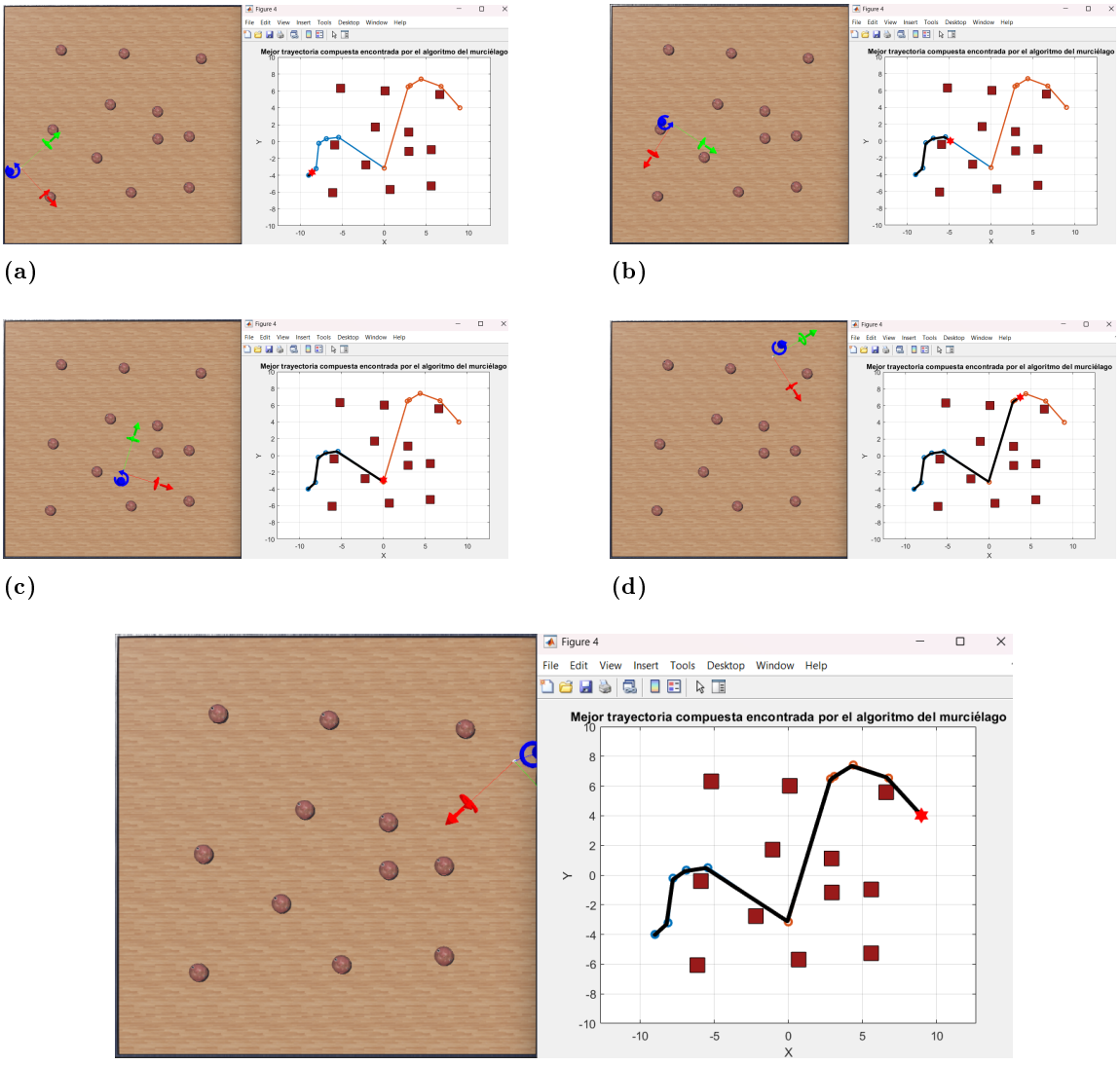
Cuadro 19. Resultados de prueba con mapa E en Webots con la aplicación de planificación de trayectorias compuestas.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	t(s)	Costo	Corrida	t(s)	Costo
1	138.5	22.6	1	304.4	22.1
2	123.2	20.0	2	376.1	24.4
3	162.4	26.9	3	398.4	20.6
4	157.9	25.9	4	407.8	21.6
5	136.1	21.7	5	384.6	20.0
6	151.6	24.0	6	404.7	21.1
7	190.1	31.0	7	303.9	21.1
8	132.2	20.9	8	395.6	21.4
9	152.3	24.3	9	397.6	19.7
10	168.3	27.2	10	412.4	21.3

Nota. Las 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga con una trayectoria compuesta para el mapa E en Webots de $(-9, -4)$ a $(9, 4)$. Elaboración propia.

Luego de realizar las pruebas con las dos variaciones de la aplicación de planificación de trayectorias, se observa que esta situación no era lo suficientemente complicada para justificar el uso del algoritmo de generación de trayectorias compuestas. En el caso del algoritmo del murciélago, el tiempo promedio de ejecución aumentó de 124.9 s a 151.3 s y el costo se mantuvo bastante similar. En el caso del de la luciérnaga, pasó de tomar un máximo de 4 minutos a casi 7 y el costo empeoró. En la Figura 19, se muestra un ejemplo de la trayectoria compuesta planificada a partir del algoritmo del murciélago y cómo el robot se mueve por el mapa en Webots.

Figura 19. Ejemplo de trayectoria compuesta para el mapa E en Webots.



(a) (b) (c) (d) (e)

Nota. Un ejemplo de la trayectoria compuesta encontrada para ir de $(-9, -4)$ a $(9, 4)$ en el mapa E por el algoritmo del murciélago en Webots. Elaboración propia.

CAPÍTULO 10

Aplicación de algoritmos para otros problemas de optimización

En este capítulo se muestran los resultados obtenidos para un problema de optimización diferente a la planificación de trayectorias. El problema consiste en encontrar la posición óptima para unos sensores de modo que estos cubran la mayor área posible sin desperdiciar cobertura al salirse del mapa o al tener intersección con la cobertura de otros sensores.

10.1. Aplicación para optimizar la cobertura de sensores

Para esta aplicación se asume que se sabe la cantidad de sensores que se tiene, el radio de cobertura de los sensores (todos tienen el mismo) y la posición de objetos en el mapa o área donde se quieren colocar los sensores.

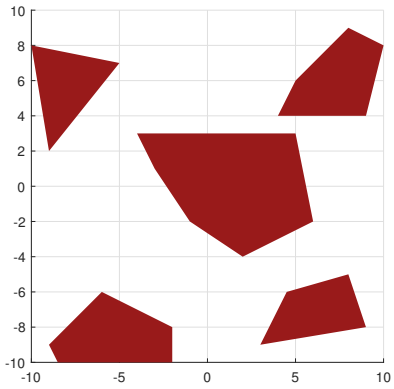
Similar a la aplicación de planificación de trayectorias (Sección 8.1), para esta aplicación se tiene un costo. En este caso es la diferencia entre el área que se puede cubrir y la cobertura efectiva de los sensores, y el costo se penaliza de forma artificial. El costo de esta aplicación se penaliza si parte del área de cobertura queda fuera del área de trabajo, si hay intersecciones entre las zonas de cobertura de los sensores y si el sensor queda colocado dentro de un obstáculo.

10.2. Pruebas con aplicación de optimización de cobertura de sensores

10.2.1. Mapa usado en las pruebas

Para estas pruebas solo se diseñó un mapa, el cual presenta varias formas irregulares, y también se agregaron bordes verdes en los límites del área de trabajo para que se pueda observar si el área de cobertura de los sensores quedó fuera de los límites. Se puede ver en la Figura 20.

Figura 20. Mapa de pruebas F.



Nota. Mapa con obstaculos irregulares F usado en las pruebas para la aplicación de optimización de cobertura de sensores. Elaboración propia.

10.2.2. Pruebas con el mapa F

Para estas pruebas se tienen 4 sensores con radio de 5 y el resto de parámetros se mantienen iguales a los que se usaron en la Sección 8.1. En el Cuadro 20 se muestra el costo de la mejor solución encontrada y el porcentaje de área cubierta si se usaran esas posiciones para los sensores. Tanto el algoritmo del murciélago como el de la luciérnaga se ejecutaron 10 veces cada uno.

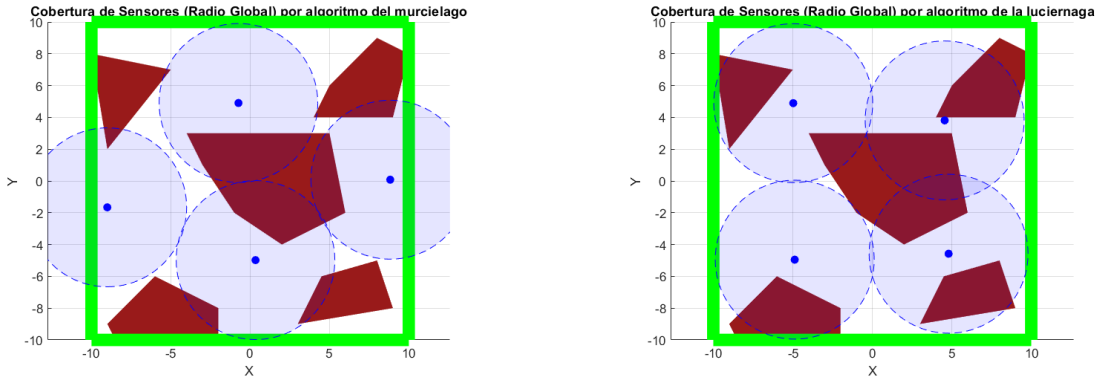
Cuadro 20. Resultados de prueba con mapa F.

Algoritmo del murciélago			Algoritmo de la luciérnaga		
Corrida	Costo	Área cubierta (%)	Corrida	Costo	Área cubierta (%)
1	419.1	63.3	1	206.2	67.5
2	753.8	56.1	2	193.6	68.4
3	1035.3	67.6	3	200.4	69.5
4	1058.8	59.3	4	422.6	64.4
5	737.9	61.7	5	417.1	64.0
6	565.2	54.2	6	416.5	64.2
7	527.3	61.3	7	202.6	68.8
8	753.3	56.3	8	342.5	65.3
9	741.3	60.5	9	220.9	68.2
10	1047.4	63.3	10	373.3	63.8

Nota. Los 10 resultados de las pruebas con el algoritmo del murciélago y el de la luciérnaga al optimizar la posición de 4 sensores con un radio de cobertura de 5 para el mapa F. Elaboración propia.

En estas pruebas se puede observar que los costos para los dos algoritmos son significativamente más altos. Esto tiene sentido cuando se toman en cuenta las restricciones del problema a resolver. En otras palabras, se trata de cubrir un área irregular con formas regulares, sin que estas se interlapen o se salgan del área de trabajo, y se restringió el área donde puede colocarse el radio de los círculos que representan la cobertura de los sensores. A pesar de esto, aún se puede ver que tanto el algoritmo del murciélago como el de la luciérnaga presentan tendencias similares a las que se han visto en las secciones anteriores. En la Figura 21 se muestran ejemplos de la posición de los sensores en el mapa F obtenidas con el algoritmo del murciélago y el de la luciérnaga en esta prueba.

Figura 21. Ejemplos de posiciones de sensores para el mapa F.



Nota. Un ejemplo de las posiciones de los 4 sensores con radio de cobertura 5 encontradas para el mapa E por el algoritmo del murciélago y el de la luciérnaga. Elaboración propia.

CAPÍTULO 11

Discusión de resultados

Los resultados obtenidos en los capítulos anteriores permitieron evaluar el desempeño del algoritmo del murciélago y de la luciérnaga en problemas de optimización y planificación de trayectorias. En las pruebas iniciales del Capítulo 7, ambos algoritmos fueron capaces de converger hacia la solución óptima, pero se pueden observar diferencias en su comportamiento. El algoritmo de la luciérnaga mostró una convergencia más rápida y estable. En contraste, el del murciélago presentó una convergencia más variable y necesitó más iteraciones. Hay que mencionar que, para estas pruebas, como se conoce el valor del costo que se busca, era posible detener la ejecución del algoritmo cuando el costo se acercara lo suficiente al valor real. En las pruebas realizadas en los otros capítulos no se tenía esta opción y por eso se observó que cuando los dos algoritmos tienen que realizar un número fijo de iteraciones, el algoritmo del murciélago es el más rápido de los dos.

En las pruebas de planificación de trayectorias en el Capítulo 8, los resultados confirmaron que ambos algoritmos pueden usarse para planificar trayectorias en escenarios con obstáculos, al generar rutas viables sin colisiones entre un punto inicial y un punto objetivo. Sin embargo, las trayectorias producidas por el algoritmo de la luciérnaga tendieron a tener un menor costo total y a tener resultados más estables de una corrida a la otra, mientras que los resultados del algoritmo del murciélago presentaron más irregularidades, pero tuvieron un tiempo de ejecución significativamente más bajo. Esto tiene sentido cuando se toma en cuenta el tipo de exploración que realiza cada algoritmo: el algoritmo de la luciérnaga tiene un enfoque más local, mientras que el del murciélago realiza una búsqueda global, pero es menos controlada.

La prueba en Webots del Capítulo 9 permitió comprobar los resultados teóricos en un entorno más realista. Los robots diferenciales fueron capaces de seguir las trayectorias planificadas por los dos algoritmos, lo que valida que ambos algoritmos se pueden aplicar a tareas de navegación autónoma.

Por último, en el Capítulo 10 se ejemplifica otra clase de problemas en donde se pueden aplicar estos algoritmos. Ambos algoritmos fueron capaces de resolver la situación propuesta y los resultados fueron bastante similares a los obtenidos en las otras pruebas realizadas. El algoritmo de la luciérnaga presentó menos varianza y más precisión, mientras que el del murciélago demostró ser una alternativa con mayor varianza, pero que requiere mucho menos tiempo de ejecución.

Estos resultados indican que los dos algoritmos son opciones viables como algoritmos de inteligencia de enjambre para aplicaciones de la Ingeniería Mecatrónica, aunque su desempeño dependerá del tipo de problema y de los valores de los parámetros.

CAPÍTULO 12

Conclusiones

- Se determinó que el algoritmo del murciélago y de la luciérnaga son alternativas viables como algoritmos de inteligencia de enjambre para resolver problemas típicos de la ingeniería mecatrónica, así como problemas de optimización.
- La implementación en Matlab confirma el funcionamiento de los dos algoritmos y su capacidad de alcanzar los mínimos globales de funciones de costo.
- El algoritmo del murciélago presentó variación en cuanto a sus resultados de iteración a iteración, pero su tiempo de ejecución se mantuvo consistentemente bajo.
- El algoritmo de la luciérnaga tuvo tiempos de ejecución bastante elevados, pero presentó mejores resultados, mientras que necesito menos iteraciones con poca variación de una ejecución a la otra.
- Ambos algoritmos fueron capaces de planificar trayectorias viables en mapas con obstáculos. Las trayectorias obtenidas fueron eficientes y evitaron colisiones, lo que valida su aplicabilidad para usarse en escenarios de navegación autónoma.
- Las simulaciones realizadas en Webots confirman que las trayectorias pueden implementarse junto a un controlador PID con acercamiento exponencial para llevar a un robot diferencial sencillo de un punto A a un punto B.
- Los escenarios prácticos diseñados permitieron analizar el desempeño de los algoritmos en escenarios más realistas. En estas pruebas, el algoritmo de la luciérnaga presentó un comportamiento más robusto independientemente del entorno, mientras que el del murciélago fue más eficiente en cuanto a tiempo de ejecución.
- El trabajo permitió ampliar el repertorio de algoritmos de inteligencia de enjambre con los que se ha trabajado en la universidad, lo que crea la base para futuros trabajos.

CAPÍTULO 13

Recomendaciones

- Ampliar las pruebas hacia implementaciones con robots físicos, al adaptar los códigos de Matlab y Webots para evaluar su desempeño ante factores más dificiles de simular, como el tiempo de respuesta.
- Optimizar la selección de parámetros de ambos algoritmos con el fin de reducir la variabilidad observada en los resultados y obtener soluciones más consistentes y estables en distintos escenarios.
- Incrementar la complejidad de los mapas utilizados en las pruebas, al usar mapas más grandes, con obstáculos irregulares, espacios estrechos o elementos con movimiento. Esto permitiría evaluar la escalabilidad de los algoritmos y su habilidad para adaptarse a escenarios todavía más realistas.
- Desarrollar nuevas aplicaciones de planificación de trayectorias al implementar un método como *occupancy grids*, para integrar la información del terreno en el proceso de planificación. Estos serían los pasos iniciales para eventualmente probar los algoritmos con métodos de mapeo en tiempo real.
- Comparar los resultados con los otros algoritmos que ya se han implementado en la universidad. Esto permitiría identificar las fortalezas y las limitaciones del algoritmo del murciélago y de la luciérnaga, lo que aportaría evidencia más sólida sobre su efectividad.
- Realizar pruebas en físico con los algoritmos dentro del Robotat de la Universidad del Valle de Guatemala como se ha hecho en años anteriores con los otros algoritmos de inteligencia de enjambre que se han trabajado.

CAPÍTULO 14

Referencias

[1] Agrocare Latinoamérica. «La robótica de enjambre adaptable podría revolucionar la agricultura inteligente.» es. Basado en el proyecto CASS de la Universidad Texas A&M, Agrocare Latinoamérica, visitado 12 de abr. de 2025. dirección: https://www.agrocarelatinoamerica.org/wp-content/uploads/2021/11/La-robotica-de-enjambre-adaptable-podria-revolucionar-la-agricultura-inteligente.pdf.

[2] E. Angulo, *Un ejército de nanorrobots logra reparar aneurismas cerebrales en conejos*, es, 7 de oct. de 2024. visitado 12 de abr. de 2025. dirección: https://elpais.com/salud-y-bienestar/2024-10-07/un-ejercito-de-nanorrobots-logra-reparar-aneurismas-cerebrales-en-conejos.html.

[3] Z. Yu, Z. Si, X. Li, D. Wang y H. Song, «A Novel Hybrid Particle Swarm Optimization Algorithm for Path Planning of UAVs,» *IEEE Internet of Things Journal*, vol. 9, págs. 22 547-22 558, 22 nov. de 2022, ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3182798.

[4] S. Wu, A. Dong, Q. Li, W. Wei, Y. Zhang y Z. Ye, «Application of ant colony optimization algorithm based on farthest point optimization and multi-objective strategy in robot path planning,» *Applied Soft Computing*, vol. 167, pág. 112 433, dic. de 2024, ISSN: 15684946. DOI: 10.1016/j.asoc.2024.112433.

[5] G. Iriarte, «Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica,» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2021.

[6] D. M. Baldizón, «Aplicaciones Prácticas para Algoritmmos de Inteligencia y Robótica de Enjambre,» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2022.

[7] J. M. Cardona, «Validación de los algoritmos de robótica de enjambre Particle Swarm Optimization y Ant Colony Optimization con sistemas robóticos físicos en el ecosistema Robotat,» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2023.

[8] A. S. Aguilar, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO),» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2019.

[9] A. L. Barrientos, «Optimización del algoritmo de robótica de enjambre Particle Swarm Optimization para su implementación con agentes robóticos físicos en escenarios con obstáculos en el ecosistema Robotat,» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2024.

[10] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2.ª ed. Chichester, England: John Wiley & Sons, 2008, ISBN: 9780470035610.

[11] X.-S. Yang, «A new metaheuristic bat-inspired algorithm,» en *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, ép. Studies in Computational Intelligence, J. González, D. Pelta, C. Cruz, G. Terrazas y N. Krasnogor, eds., vol. 284, Berlin, Heidelberg: Springer, 2010, págs. 65-74. DOI: 10.1007/978-3-642-12538-6_6.

[12] X.-S. Yang, «Firefly Algorithm, Stochastic Test Functions and Design Optimisation,» en *International Journal of Bio-Inspired Computation*, 2, vol. 2, Inderscience Publishers, 2010, págs. 78-84.

[13] B. Siciliano, L. Sciavicco, L. Villani y G. Oriolo, *Robotics: Modelling, Planning and Control*. London: Springer, 2010, ISBN: 978-1-84628-641-4.

[14] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki y S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005, ISBN: 978-0-262-03327-5.

[15] C. Canudas de Wit, B. Siciliano y G. Bastin, *Theory of Robot Control*. London: Springer, 1996, ISBN: 978-3-540-76219-1.

[16] K. M. Lynch y F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge: Cambridge University Press, 2017, ISBN: 978-1-107-16463-6. dirección: http://modernrobotics.org.

[17] R. Siegwart, I. Nourbakhsh y D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd. MIT press, 2011.

[18] M. W. Spong, S. Hutchinson y M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ: John Wiley & Sons, 2005, ISBN: 978-0-471-64990-8.

[19] J. Borenstein, H. R. Everett y L. Feng, *Where am I? Sensors and Methods for Mobile Robot Positioning*. Ann Arbor, MI: University of Michigan, 1996, ISBN: 978-99949-25-25-8.

[20] N. Correll, M. Wing, A. Schoellig y A. Martinoli, *Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators and Algorithms*. Cambridge, MA: The MIT Press, 2022, ISBN: 978-0-262-04772-2.

[21] S. Thrun, W. Burgard y D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005, ISBN: 978-0-262-20162-9.

CAPÍTULO 15

Anexos

15.1. Pseudocódigo de algoritmos de inteligencia de enjambre

Algoritmo 2 Algoritmo del murciélago (Bat Algorithm, BA) [11]

Inicializar la población de murciélagos x_i $(i = 1, 2, ..., n)$ con velocidades v_i
Definir la frecuencia f_i, el nivel de pulso r_i y la amplitud del sonido A_i
Mientras criterio de parada no cumplido **Hacer**
    Para cada murciélago i **Hacer**
        Actualizar la frecuencia f_i y la velocidad v_i
        Actualizar la posición x_i
        Si rand $> r_i$ **Entonces**
            Seleccionar una solución entre las mejores actuales
            Generar una nueva solución local alrededor de la mejor solución
        FinSi
        Generar una nueva solución por vuelo aleatorio con frecuencia f_i
        Si rand $< A_i$ **and** $f(x_i) < f(x^*)$ **Entonces**
            Aceptar la nueva solución
            Incrementar r_i y reducir A_i
        FinSi
    FinPara
    Ordenar y encontrar el mejor murciélago x^*
FinMientras
Retornar x^*

Algoritmo 3 Algoritmo de luciérnagas (Firefly Algorithm, FA) [12]

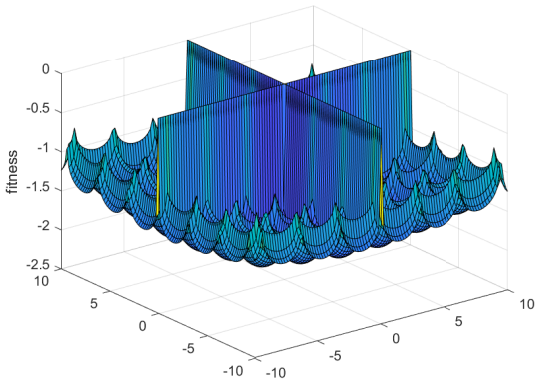
Inicializar la población de luciérnagas x_i $(i = 1, 2, ..., n)$
Definir el parámetro de absorción de luz γ, el coeficiente de atracción β_0 y el paso aleatorio α
Mientras criterio de parada no cumplido **Hacer**
    Para cada luciérnaga i **Hacer**
        Para cada luciérnaga j **Hacer**
            Si $f(x_j) < f(x_i)$ **Entonces**
                Calcular la distancia $r_{ij} = \|x_i - x_j\|$
                Calcular la atracción $\beta = \beta_0 e^{-\gamma r_{ij}^2}$
                Actualizar la posición:

$$x_i = x_i + \beta(x_j - x_i) + \alpha(\text{rand} - 0.5)$$

            FinSi
        FinPara
    FinPara
    Evaluar y ordenar las luciérnagas por brillo (valor de aptitud)
FinMientras
Retornar la mejor solución encontrada x^*

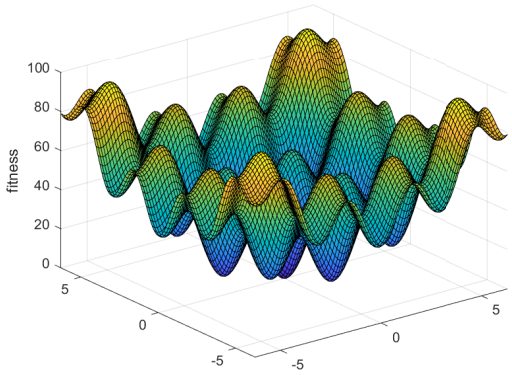
15.2. Figuras adicionales

Figura 22. Función de costo Cross-in-Tray.



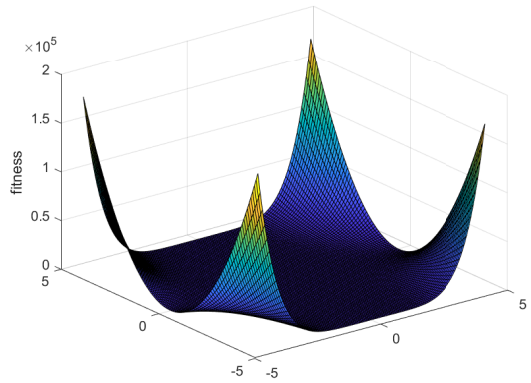
Nota. La imagen muestra a la función de costo Cross-in-Tray. Elaboración propia.

Figura 23. Función de costo Eggcrate.



Nota. La imagen muestra a la función de costo Cross-in-Tray. Elaboración propia.

Figura 24. Función de costo Beale.



Nota. La imagen muestra a la función de costo Cross-in-Tray. Elaboración propia.

15.3. Especificaciones de la computadora usada en las pruebas

Cuadro 21. Especificaciones de la computadora utilizada para las simulaciones.

Componente	**Detalle**
Procesador (CPU)	AMD Ryzen 7 7735HS with Radeon Graphics, 8 núcleos, 16 hilos, 5600 MHz
Memoria RAM	16 GB DDR5, 5600 MHz
Almacenamiento	SSD NVMe WD PC SN560, 1 TB
Sistema operativo	Windows 11, 64-bit

Nota. Especificaciones de la computadora usada en todas las pruebas realizadas en el trabajo. Elaboración propia.

15.4. Repositorio en Github

Puede acceder a los codigos usados en el siguiente repositorio:

`https://github.com/PJPE1227/19_MT_Pablo_Pe-a`