Escuela
Politécnica
Superior

# Computación evolutiva en comportamientos de robótica de enjambre

Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Autor:
Jasmina Rais Martínez
Tutor/es:
Fidel Aznar Gregori

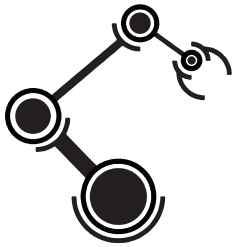
Universitat d'Alacant
Universidad de Alicante

Julio 2021

# Computación evolutiva en comportamientos de robótica de enjambre

**Autor**
Jasmina Rais Martínez

**Tutor/es**
Fidel Aznar Gregori
*Departamento de ciencia de la computación e inteligencia artificial*



Máster Universitario en Automática y Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

# Preámbulo

En la actualidad, la robótica de enjambre estudia cómo obtener comportamientos complejos a través de tareas individuales sencillas utilizando robots simples y económicos. Por este motivo, en este proyecto se van a estudiar diferentes conductas complejas de robótica de enjambre obtenidas a través del uso de estrategias evolutivas para el aprendizaje por refuerzo profundo. Para obtener las políticas óptimas de cada tarea se utilizará un simulador con física realista en 2D. Además, se utilizarán dos robots reales MBot Ranger para ejecutar una tarea de formación en cadena.

Por otra parte, se evaluarán las conductas obtenidas en algunos requisitos de la robótica de enjambre como la robustez o la flexibilidad del entorno. Esta evaluación se hará tanto en la conducta de formación en cadena, como en la conducta de mantenimiento de formaciones complejas en entornos con obstáculos.

Este proyecto ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (Spain), proyecto RTI2018-096219-B-I00. Proyecto co-financiado con fondos FEDER.

# Agradecimientos

En primer lugar, me gustaría darle las gracias a mi tutor, Fidel Aznar, por su valiosa ayuda tanto a lo largo del desarrollo de este proyecto como en el desarrollo de los artículos publicados derivados de este trabajo.

También me gustaría agradecer a mis amigos y compañeros, a quienes no nombro para no caer en olvidos innecesarios e incómodos, aunque ellos ya saben quiénes son, la ayuda y compañía que me han ofrecido cuando la he necesitado tanto en los buenos como en los malos momentos.

Por último, quiero realizar un agradecimiento especial a mi familia por el apoyo incondicional que me han proporcionado durante toda mi vida, incluyendo las situaciones difíciles presentes a lo largo de mis etapas educativas.

*¿Heredarán los robots la Tierra?*
*Sí, pero serán nuestros hijos*

Marvin Minsky

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de Códigos

# 1 Introducción

## 1.1 Resumen del trabajo

Este proyecto se centra en el aprendizaje de varias políticas de movimiento para un enjambre robótico. Se trata de comportamientos complejos dado que los robots a utilizar son simples y con poca capacidad de procesamiento o comunicación, siguiendo los preceptos de la robótica de enjambre. De manera concreta se intentarán aprender políticas que desarrollen conductas macroscópicas para la formación en cadena, donde un enjambre sigue al robot líder mientras mantiene la forma de una cadena y para el mantenimiento de una formación compleja mientras siguen al robot líder y esquivan los obstáculos del entorno. Por último, se ha empleado la conducta de mantenimiento de una formación compleja para la resolución de una tarea de transporte colectivo de objetos con la estrategia de empuje.

Para la resolución de ambas tareas se han utilizado estrategias evolutivas para el aprendizaje por refuerzo. Este tipo de aprendizaje requiere multitud de experimientos para aprender la política deseada. Por tanto se ha utilizado un simulador en 2D rápido con física realista del robot MBot Ranger, que será nuestra plataforma de pruebas final.

En nuestras pruebas se ha evaluado algunas características como la robustez frente al ruido o la escalabilidad con diferentes enjambres. Además, se han evaluado los resultados de diferentes estrategias evolutivas aplicadas a la obtención de una política de aprendizaje por refuerzo en robótica de enjambre.

## 1.2 Motivación

Actualmente, la robótica de enjambre es un campo de investigación, en el que se intentan conseguir comportamientos globales complejos, mediante tareas individuales simples, utilizando para ello, robots de bajo coste y con prestaciones limitadas. Por esta razón, el motivo principal del desarrollo de este proyecto es conseguir aprender conductas complejas de robótica de enjambre utilizando estrategias evolutivas para el aprendizaje por refuerzo. Además, se pretende evaluar qué estrategia evolutiva es la más óptima para resolver cada tarea en distintos enjambres.

## 1.3 Estado del arte

La robótica de enjambre presenta diferentes tipos de comportamientos muy variados donde se pueden utilizar distintos tipos de robots. En este proyecto se han utilizado estrategias

evolutivas para el aprendizaje por refuerzo para modelar el comportamiento de un enjambre. Este enjambre debe llevar a cabo la tarea de seguir los movimientos de un robot líder mientras mantienen la forma que se obtiene con sus posiciones iniciales y esquivan los obstáculos del entorno. En esta sección se va a mostrar tareas resueltas relacionadas con este problema.

Como se ha mencionado anteriormente, los robots deben esquivar los obstáculos del entorno, y para conseguirlo utilizarán un sensor láser para detectar la distancia de los objetos. Respecto a esta tarea, en la propuesta de Huang y cols. (2005) se utiliza Deep Q-learning para que un robot deambule por el entorno sin colisionar con los obstáculos. El robot de este artículo detecta los obstáculos mediante un sensor de infrarrojos y decide qué acción ejecutar mediante una red neuronal entrenada con Deep Q-learning.

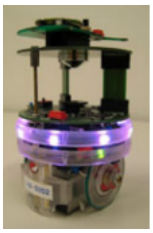
Por otra parte, la tarea de seguir al robot líder mientras mantienen una forma inicial dada tiene aspectos en común con las siguientes tareas de robótica de enjambre:

- **Movimiento coordinado**: En esta tarea un enjambre se desplaza a una posición objetivo utilizando determinados movimientos. El enjambre debe encontrar la forma de sincronizarse para llegar a la posición determinada al mismo tiempo.
- **Planificación de trayectorias**: En esta tarea un enjambre se desplaza a una posición objetivo decidiendo qué movimientos realizar para alcanzarla.
- **Formación en cadena**: En esta tarea el enjambre debe seguir los movimientos del robot "explorador" formando una cadena.

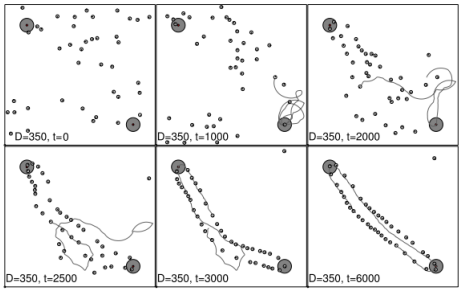
Las tareas de robótica de enjambre mencionadas anteriormente tienen en común comportamientos con la tarea principal a desarrollar en este proyecto porque al mantener una forma inicial el enjambre debe seguir al robot independiente de forma similar a una cadena, aunque en esta tarea los robots no tienen porqué formar una cadena al desplazarse. Además, deben desplazarse a una posición objetivo, que en cada instante será la posición que les corresponde según el movimiento y la posición del robot independiente respecto a su posición inicial. Por último, los robots deben coordinarse y desplazarse juntos para evitar perder la forma inicial en cada instante, aunque en este proyecto no se utilizarán algoritmos de sincronización para resolver este problema.

En la **formación en cadena** se distinguen dos roles: el explorador y la cadena. En cada enjambre hay un robot explorador, que se encarga de explorar el entorno y de liderar la cadena. Mientras que el resto de robots del enjambre son los robots que continúan la cadena siguiendo los movimientos del robot explorador. En Sperati y cols. (2011) utilizan un algoritmo evolutivo para la exploración y navegación de un enjambre en un entorno desconocido. Además, utilizan un modelo del robot E-puck y la comunicación del enjambre se realiza mediante luces LED de colores. Con este algoritmo se ha conseguido que los robots se desplacen hacia dos zonas determinadas formando una cadena.

En el **movimiento coordinado** los robots se mueven imitando el comportamiento de los peces o el de una bandada de aves. En Turgut y cols. (2008) utilizan un sensor de rumbo virtual que permite medir la dirección en la que se desplazan otros robots. Con esta información y con la distancia entre los robots vecinos, que se obtiene mediante un sensor de infrarrojos, consiguen alcanzar una dirección objetivo evitando los obstáculos. Mientras que en Ferrante



**(a)** E-puck robot

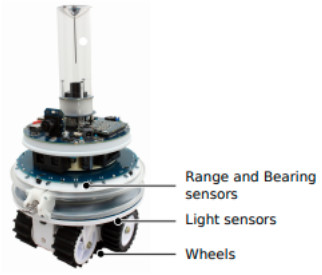



**(b)** Formación de una cadena al desplazarse entre dos zonas

**Figura 1.1:** Formación en cadena en Sperati y cols. (2011)

y cols. (2012) se propone un controlador de movimiento coordinado diferente a los utilizados en otros proyectos. En este artículo se describe cómo se desarrolla esta tarea con reglas de repulsión y atracción entre los robots.

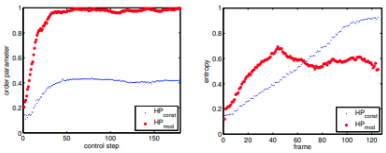
**(a)** Robot Kobot, robot utilizado en Turgut y cols. (2008)



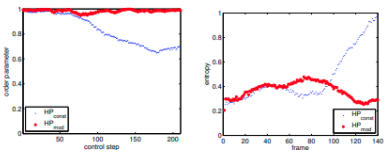

**(b)** Robot foot-bot, robot utilizado en Ferrante y cols. (2012)

**Figura 1.2:** Tipos de robots utilizados en la tarea de movimiento coordinado

En la **planificación de trayectorias**, un enjambre conoce su posición inicial y la posición que debe ser su posición final, es decir, la posición objetivo. Para alcanzar esa posición deben decidir qué movimientos realizar mientras se desplazan juntos. En Lin y cols. (2013) se utilizan algoritmos genéticos para conseguir alcanzar la posición ideal mediante una política óptima que permita mantener la forma del enjambre mientras siguen al robot líder. La función de recompensa tiene en cuenta la posición actual en la que están los robots y la posición objetivo intermedia que deberían tener en cada instante. Por lo tanto, en este artículo se calcula primero cuál es el camino que deben seguir los robots para alcanzar la posición deseada y después cuáles son los movimientos que deben realizar los robots del enjambre para seguir al robot líder manteniendo la distancia correcta. Esta tarea es la más parecida a la tarea principal que se desarrolla en este proyecto, ya que, en la planificación de trayectorias los robots siguen a un robot líder, que se podría considerar el robot independiente, con el objetivo de alcanzar una posición objetivo mientras mantienen las distancias correctas para conseguir obtener la forma deseada. De la misma forma, en la tarea principal de este proyecto, un enjambre debe



**(a)** Métricas de los resultados de la experimentación de la fase de alineación de los robots

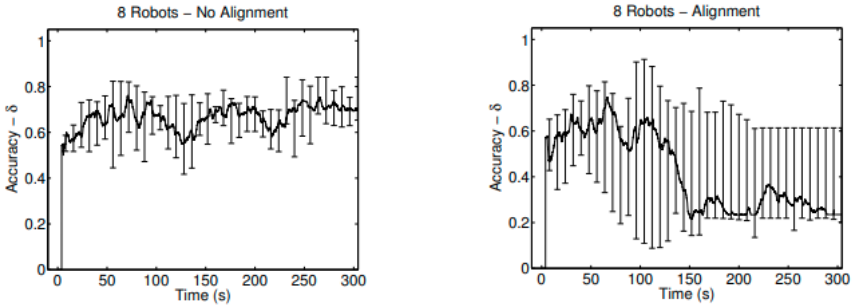


**(b)** Métricas de los resultados de la experimentación de la fase de evitar obstáculos


**(c)** Movimiento coordinado con robots reales y un obstáculo en el entorno

**Figura 1.3:** Resultados obtenidos con robots reales en Turgut y cols. (2008)

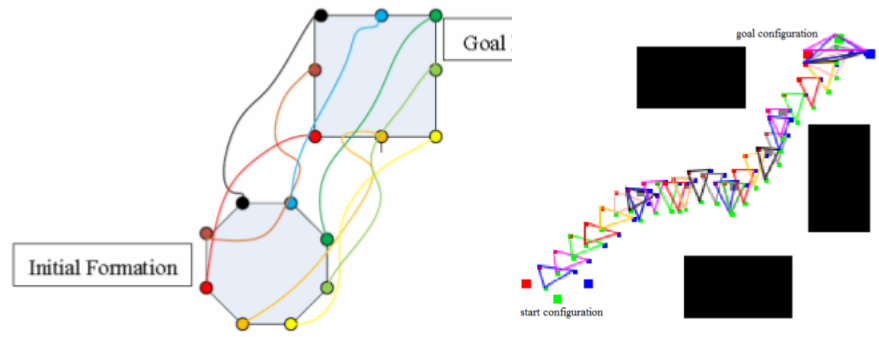
mantener la forma inicial dada en todo momento mientras siguen al robot independiente. La diferencia entre ambas tareas es que en la segunda tarea, es decir, en la desarrollada en este proyecto, los robots no tienen información sobre las posiciones intermedias que deben ocupar para alcanzar la posición objetivo porque el objetivo de este proyecto es que sigan al robot independiente sin perder la forma.




**(a)** Resultados obtenidos sin control de alineación

**(b)** Resultados obtenidos con control de alineación

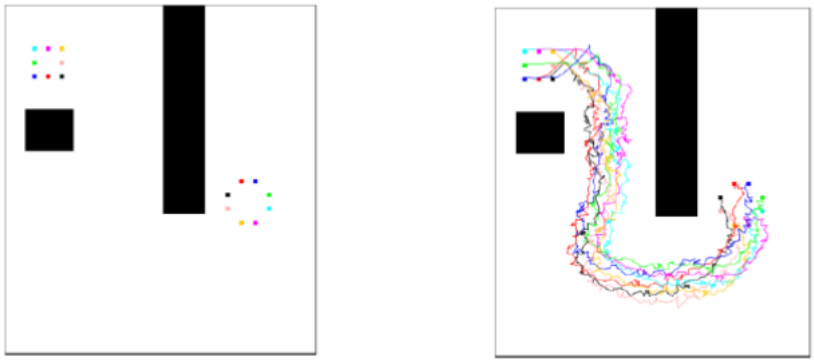
**Figura 1.4:** Resultados obtenidos en Ferrante y cols. (2012). En las gráficas se puede observar la evolución de la métrica de precisión de un enjambre de 8 robots con 2 robots que conocían la dirección a la que debían desplazarse.




**(a)** Posición inicial y deseada del enjambre

**(b)** Trayectorias realizadas con obstáculos

**Figura 1.5:** Posiciones objetivo que deben alcanzar los robots del enjambre en Lin y cols. (2013)



**(a)** Configuración inicial

**(b)** Trayectorias finales

**Figura 1.6:** Resultados obtenidos en Lin y cols. (2013) de un enjambre formado por 8 robots.

## 1.4 Propuesta y objetivos

En este proyecto se propone el aprendizaje de conductas complejas para sistemas robóticos de enjambre a través del uso de estrategias evolutivas para el aprendizaje por refuerzo. Para el desarrollo de este proyecto se establecen los siguientes objetivos:

- **Evaluación de eficiencia de las estrategias evolutivas**: Se pretende evaluar cuál es la estrategia evolutiva que obtiene una política óptima en menor tiempo de entrenamiento para la resolución de una tarea de enjambre. Además, se busca poder evaluar los resultados de las diferentes conductas aprendidas en entornos simulados.
- **Aprendizaje de una conducta en formación en cadena**: La conducta de formación en cadena consta de que un enjambre de robots mantengan la formación en cadena inicial mientras el robot líder se desplaza por un entorno. Se pretende resolver esta tarea en un entorno sin obstáculos.
- **Transferencia de una conducta aprendida a robots reales**: Uno de los objetivos de este proyecto es poder evaluar una conducta aprendida en un entorno real. Para poder resolver este problema se utilizará la conducta aprendida de formación en cadena.
- **Aprendizaje del mantenimiento de una formación en un entorno simple**: Una de las tareas que se desea resolver es el mantenimiento de una formación compleja inicial en un entorno simple sin obstáculos.
- **Aprendizaje del mantenimiento de una formación con evitación de obstáculos**: Esta conducta presenta una complejidad mayor que la anterior tarea, ya que, a la conducta de mantenimiento de una forma se le añade la tarea de evitación de obstáculos del entorno.
- **Análisis de métrica de las conductas de enjambre**: Otro objetivo importante de este proyecto es evaluar las métricas de las conductas. Las métricas que se analizarán son: Escalabilidad con distinto número de agentes del enjambre, robustez frente al ruido o a nuevos obstáculos y flexibilidad para poder resolver la tarea en distintos entornos.
- **Emplear el aprendizaje de una conducta para una tarea compleja de robótica de enjambre**: Otro objetivo es poder emplear una de las conductas aprendidas para la resolución de tareas complejas de robótica de enjambre. La tarea de enjambre que se resolverá es el transporte colectivo de objetos.

Para desarrollar este proyecto se han utilizado conocimientos adquiridos en las asignaturas: Robótica y Aprendizaje automático.

# 2 Computación evolutiva

En este proyecto se realiza un proceso de aprendizaje mediante computación evolutiva para llevar a cabo una tarea de enjambre. La técnica utilizada se llama Deep Reinforcement Learning y combina el uso de estrategias evolutivas para el entrenamiento de una red neuronal. Por ello, en este capítulo se va a explicar qué son las estrategias evolutivas y qué elementos están presentes en Deep Reinforcement Learning.

## 2.1 Introducción

La computación evolutiva es una rama de la inteligencia artificial que se basa en los procesos de evolución que ocurren en la naturaleza. Esta técnica se utiliza, principalmente, en tareas de optimización, diseño y control. Además, este proceso se puede utilizar para entrenar una red neuronal que controle determinadas acciones de los robots.

## 2.2 Redes neuronales

En primer lugar, las redes neuronales son funciones matemáticas inspiradas en las neuronas que constituyen el cerebro humano. En un nivel abstracto se pueden ver como la siguiente función:

$$f_\theta : x \to y \tag{2.1}$$

Donde $\boldsymbol{x}$ es la entrada que produce la salida $\boldsymbol{y}$ y su comportamiento está parametrizado por $\theta$.

### 2.2.1 Neuronas

Las redes neuronales están formadas por neuronas distribuidas en diferentes capas. La neurona es la parte más básica de una red neuronal y su resultado se puede describir mediante la siguiente fórmula:

$$f\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right) \tag{2.2}$$

Donde $\boldsymbol{x}$ es un vector de entrada que produce un valor escalar y está parametrizado con un

vector de pesos, $\boldsymbol{w}$ y un término bias, $\boldsymbol{b}$. Además, la salida depende de la función de activación utilizada.

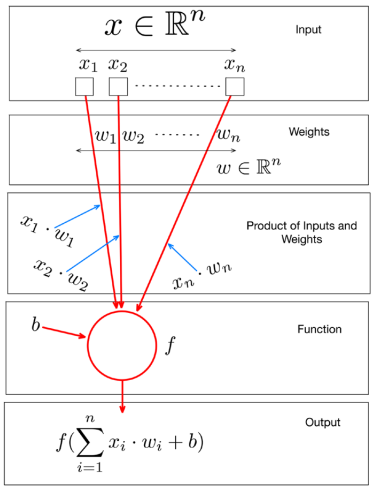



**Figura 2.1:** Esquema de una neurona

Las neuronas están distribuidas en capas, la última capa es la capa de salida, que está conectada a la última capa intermedia, y la primera capa es la capa de entrada. Además, todas las capas anteriores a la capa de salida deben estar conectadas con su capa anterior y la entrada de cada capa será la salida de la anterior, excepto en la capa de entrada. La anchura de cada capa es el número de neuronas correspondiente y la profundidad es el número de capas.

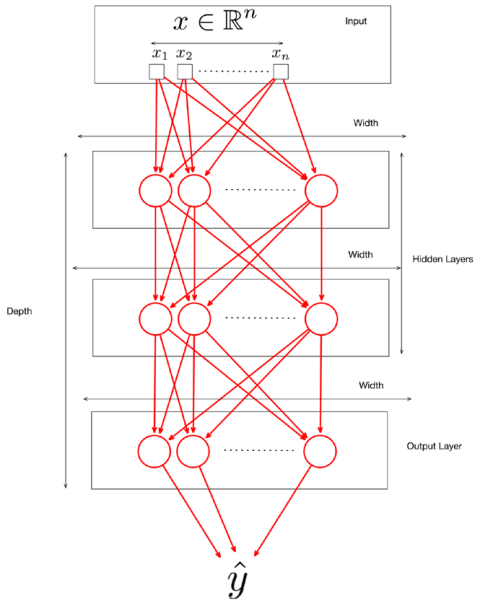



**Figura 2.2:** Esquema de una red neuronal

Para el entrenamiento de una red neuronal se utiliza el algoritmo de retropropagación para calcular el gradiente de una función objetivo.

### 2.2.2 Funciones de activación

Las funciones de activación más utilizadas en redes neuronales son las siguientes.

En primer lugar, se pueden utilizar **neuronas lineales**, estas neuronas son las más simples y su función de salida es: $y = w \cdot x + b$. Tienen un comportamiento lineal y permiten que el aprendizaje basado en gradientes sea una tarea más sencilla.

En segundo lugar, se pueden utilizar **neuronas sigmoideas**, su salida modela una distribución Bernoulli sobre $\boldsymbol{y}$ condicionada sobre $\boldsymbol{x}$. Se suele utilizar para problemas de clasificación binaria. El resultado de la salida viene dado por la siguiente función:

$$y = \frac{1}{1 + e^{-(w \cdot x + b)}} \tag{2.3}$$

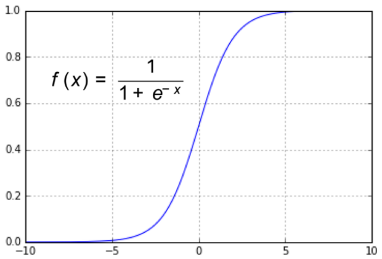



**Figura 2.3:** Función sigmoid

En tercer lugar, las **neuronas lineales rectificadas** o neuronas ReLU, se utilizan en conjunto con una transformación lineal y se suelen utilizar en las capas intermedias. La salida viene dada por la siguiente función:

$$y = max(0, w \cdot x + b) \tag{2.4}$$

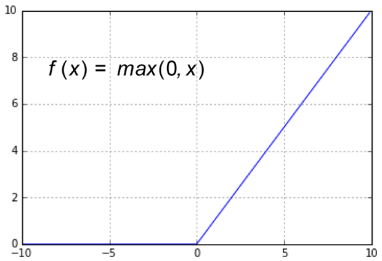



**Figura 2.4:** Función ReLU

En tercer lugar, la función **tangente hiperbólica**, tanh, es utilizada en las neuronas de las capas intermedias y su función de salida es la siguiente:

$$y = tanh(w \cdot x + b) \tag{2.5}$$

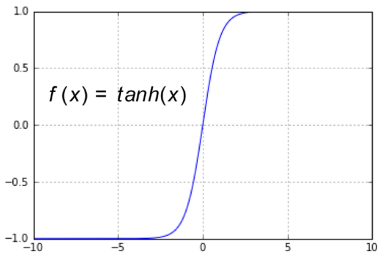



**Figura 2.5:** Función tanh

Por último, las capas **softmax** son capas utilizadas para la capa de salida de una multi-clasificación, esta capa normaliza las salidas de la capa anterior para que las neuronas en conjunto sumen 1. La capa softmax representa la probabilidad de cada clase.

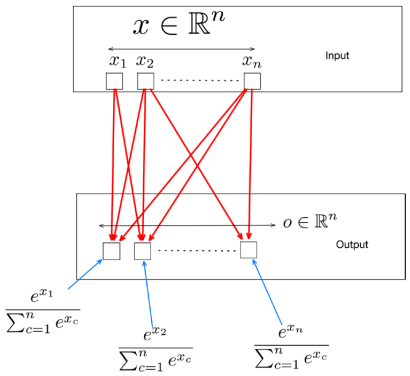



**Figura 2.6:** Capa softmax

## 2.3 Aprendizaje por refuerzo

El Aprendizaje por refuerzo (RL), es una rama del Machine learning, en el que un agente debe seleccionar las acciones en un entorno con el fin de alcanzar la máxima recompensa global. Este es uno de los tres paradigmas básicos del aprendizaje automático junto al aprendizaje supervisado y al aprendizaje no supervisado. Los algoritmos de aprendizaje por refuerzo buscan maximizar la recompensa obtenida mediante la exploración del entorno.

### 2.3.1 Proceso de decisión de Markov (MDP)

El Aprendizaje por refuerzo profundo (DRL) generalmente está modelado por el Proceso de desición de Markov (MDP). Este proceso describe matemáticamente el proceso del aprendizaje por refuerzo donde el entorno es observable. Formalmente un MDP está descrito por:

- Un conjunto finito de estados, S.
- Un conjunto finito de posibles acciones, A.
- Un modelo de transición, que especifica la probabilidad de pasar a un estado dado el estado presente y la acción, $P(s_t|s_{t+1}$, a)
- Una función de recompensa, que especifica el valor de ejecutar cierta acción a en el estado s, r(s, a).

La Propiedad de Markov indica que el resultado de una acción solo depende del acción y del propio estado, no de las acciones realizadas anteriormente, por esto se conoce como un proceso sin memoria. En este contexto se busca una política óptima, que es la política que maximiza la recompensa media esperada para las posibles secuencias de acciones que se puedan generar.

### 2.3.2 Aprendizaje por refuerzo profundo

DRL es la técnica que combina RL y redes neuronales para el modelado del comportamiento de un agente. Está técnica sigue el siguiente proceso de entrenamiento:

1. Inicializar una red neuronal de forma aleatoria.
2. Permitir que el agente interaccione con el entorno.
3. Obtener la recompensa resultante de cada acción generada

De esta forma, obtenemos un registro completo de secuencia de estados, acciones y recompensas. Las políticas que se utilizan en aprendizaje por refuerzo son estocásticas, ya que, solo calculan las probabilidades de ejecutar cualquier acción.

## 2.4 Estrategias evolutivas

Las redes neuronales se pueden utilizar en problemas complejos si se encuentran los parámetros adecuados. Sin embargo, en algunos problemas no se puede entrenar una red neuronal mediante el algoritmo de retropropagación porque no se llegaría a alcanzar el máximo global y el problema se quedaría estancado en un máximo local (Ha, 2017b). Mediante aprendizaje por refuerzo también se podría entrenar una red neuronal para realizar determinadas acciones que permitan obtener la máxima recompensa, pero, antes se deben calcular los gradientes de las recompensas de cada acción.

Por otra parte, el inconveniente de utilizar técnicas de aprendizaje basadas en información del gradiente es que generan poca información del entorno en problemas complejos en róbotica (Pagliuca y cols., 2020). Además, otro inconveniente de las políticas basadas en el gradiente es que se inicializan con políticas aleatorias, que en ocasiones obtienen soluciones no óptimas que generan una recompensa máxima porque realizan solo una acción determinada (Salimans y cols., 2017).

Como solución, se propone el uso de estrategias evolutivas. Una estrategia evolutiva es un algoritmo que proporciona una serie de candidatos de posibles soluciones a un problema dado. La evaluación se basa en una función objetivo y proporciona un valor de aptitud. Según el valor de aptitud el algoritmo producirá soluciones candidatas en las siguientes generaciones que será más probable que proporcionen mejores resultados que la generación actual. Este algoritmo sigue un proceso iterativo que finaliza cuando el usuario cree conveniente. Las soluciones candidatas de una estrategia evolutiva se muestrean a partir de una distribución cuyos parámetros están siendo actualizados en cada generación. Además, la ejecución de las poblaciones es fácilmente paralelizable, por tanto, disminuye el tiempo de entrenamiento.

A continuación, se describen una de las principales estrategias evolutivas más utilizadas en computación evolutiva.

### 2.4.1 CMA-ES

Estrategia evolutiva de adaptación de la matriz de covarianza (CMA-ES), es una estrategia que toma los resultados de cada generación y modifica el espacio de búsqueda para la siguiente generación. En cada generación se adapta $\mu$ y $\sigma$ y se calcula la matriz de covarianza completa.

El proceso de adaptación se basa en las $N_{best}$ soluciones y en los parámetros de entrada $\boldsymbol{x}$, $\boldsymbol{g}$ es la generación actual y $\boldsymbol{(g+1)}$ es la siguiente generación.

$$\mu_x^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} x_i \tag{2.6}$$

$$\sigma_x^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})^2 \tag{2.7}$$

Este algoritmo es el más popular para la optimización de gradientes. El único inconveniente es el rendimiento que puede tomar si el número de parámetros del modelo a optimizar es elevado, ya que, el cálculo de la matriz de covarianza es de orden $O(N^2)$, aunque actualmente hay aproximaciones que lo disminuyen a $O(N)$.

### 2.4.2 SES

Estrategia evolutiva simple (SES), se basa en muestrear un conjunto de soluciones mediante una distribución normal con una media $\mu$ y desviación estándar fija $\sigma$. Inicializamos $\mu$ al origen, en cada generación le asignamos el mejor resultado de la población y en las siguientes

generaciones se muestrea alrededor de esta nueva media. En general, este algoritmo funcionará en problemas simples. Debido a su naturaleza voraz, desecha todas las soluciones menos la mejor encontrada, de esta forma puede quedarse atascado en un máximo local en problemas complejos. Una posible solución a este problema sería seleccionar un conjunto de mejores resultados en vez de solo seleccionar el mejor de todos los individuos de la población.

### 2.4.3 Algoritmos genéticos

Los Algoritmos genéticos simples (GA) siguen un proceso natural que desarrolla la teoría de la supervivencia del más fuerte. Su propósito es optimizar un conjunto de parámetros con el fin de encontrar la mejor solución. Estos algoritmos están formados por una población de redes neuronales y en cada generación se selecciona el conjunto de redes que han proporcionado los mejores resultados. El algoritmo sigue el siguiente proceso iterativo:

Código 2.1: Proceso iterativo de un algoritmo genético

```
pop = init_population()

while True:
    evaluate(pop)
    parents = selection(pop)
    children = breed(parents)
    mutation(children)
    pop = parents + children
```

En el algoritmo anterior tenemos los siguientes pasos:

1. ***init_population***: Se genera la población inicial mediante individuos con parámetros aleatorios.
2. ***evaluate***: Se obtiene la recompensa de cada red neuronal.
3. ***selection***: Se seleccionan las redes neuronales que han obtenido mejores recompensas.
4. ***breed***: Obtenemos una lista de hijos para la siguiente generación que ha heredado los parámetros de sus padres.
5. ***mutation***: Se mutan algunos de los parámetros de las redes neuronales descendientes.
6. ***pop = parents + children***: Se forma la nueva población con los padres seleccionados y los hijos obtenidos, de esta forma se descartan las redes neuronales que no han proporcionado buenos resultados en la generación anterior.

Para llevar a cabo este proceso se deben configurar algunos parámetros como el número de individuos que se conservarán para la siguiente generación o la probabilidad de mutación.

Los algoritmos genéticos consiguen tener una variedad de soluciones óptimas para resolver un problema, pero, el elitismo, es decir, seleccionar las mejores redes neuronales, para evolucionar en las siguientes generaciones puede provocar que las soluciones converjan en un local óptimo.

### 2.4.4 PEPG

El objetivo de Gradientes de política de exploración de parámetros (PEPG) es maximizar el resultado de una solución muestreada y si el resultado obtenido es un buen resultado, la recompensa de la población muestreada será mejor. La implementación de esta estrategia que se utiliza en este proyecto está basada en Estrategias evolutivas naturales (NES) pero, incorporando el inverso de la matriz de información de Fisher en la regla de actualización del gradiente. Este algoritmo sigue el mismo esquema básico del resto de estrategias evolutivas, es decir, actualizan el valor de $\mu$ y $\sigma$ en cada generación con el fin de conseguir muestrear un conjunto de soluciones. Debido a que no se actualiza el parámetro de correlación, la eficiencia de este algoritmo es de orden $O(N)$, lo que indica que puede ser más rápido que CMA-ES.

### 2.4.5 OpenES

OpenAI Estrategia evolutiva (OpenES) se basa en establecer la desviación típica estándar, $\sigma$, a un número fijo y solo actualiza en cada generación la media, $\mu$. Además, añade ruido gaussiano a los pesos de la red, $\boldsymbol{w}$ para conseguir que las acciones del agente utilicen una política gaussiana.

## 2.5 Ventajas del uso de estrategias evolutivas en la computación evolutiva

A continuación, se enumeran las ventajas del uso de estrategias evolutivas para el aprendizaje por refuerzo que se demuestran en el estudio de Salimans y cols. (2017).

**No necesitan el algoritmo de retropropagación**, las estrategias evolutivas solo necesitan saber la recompensa final obtenida y no requieren la ejecución del algoritmo de retropropagación. Lo que hace que el código sea más corto y rápido en la práctica.

**Son altamente paralelizables**. Las estrategias evolutivas solo requieren que las generaciones se comuniquen unos cuantos valores escalares mientras que en el aprendizaje por refuerzo es necesario sincronizar los vectores de parámetros completos. Intuitivamente, esto se debe a que controlamos las semillas aleatorias en cada hilo, por lo que cada hilo puede reconstruir localmente las perturbaciones de los demás hilos. Así, todo lo que se necesita comunicar entre los hilos es la recompensa de cada perturbación. De esta forma, se agiliza el entrenamiento cuando se añaden más núcleos a la CPU.

Las estrategias evolutivas presentan **mayor robustez**. Varios hiperparámetros que son difíciles de configurar en el aprendizaje por refuerzo se evitan en las estrategias evolutivas.

Ofrecen una **exploración estructurada**. En algunos algoritmos de aprendizaje por refuerzo, en especial los de gradientes de políticas, se inicializan con políticas aleatorias. Esto puede provocar que manifiesten fluctuaciones aleatorias, es decir, que aparezca mucha diferencia entre los resultados de las generaciones. En cambio, en las estrategias evolutivas esto no ocurre porque usan políticas deterministas y logran una exploración consistente.

Permite la **asignación de recompensas a largo plazo**. Las estrategias evolutivas son una opción interesante especialmente cuando el número de pasos de tiempo en un episodio es largo, cuando las acciones tienen efectos de larga duración, o cuando no se dispone de buenas estimaciones de la función de valor.

Sin embargo, uno de los principales problemas para que las estrategias evolutivas funcionen es que la adición de ruido en los parámetros debe conducir a diferentes resultados para obtener alguna señal de gradiente. En el artículo de Salimans y cols. (2017) se descubre que el uso de la norma por lotes virtual puede ayudar a aliviar este problema, pero es necesario seguir trabajando en la parametrización efectiva de las redes neuronales para que tengan comportamientos variables en función del ruido.

# 3 Robótica de enjambre

La robótica de enjambre es un método de diseño de comportamientos robóticos en el que los robots trabajan en conjunto con el fin de desempeñar una tarea. Este método está inspirado en el comportamiento auto-organizado de los grupos de individuos en el reino animal. Los robots que pertenecen a un enjambre pueden llevar a cabo una tarea compleja mediante tareas individuales simples. Por lo tanto, en este capítulo se estudiará la robótica de enjambre y las diferentes tareas que puede llevar a cabo.

## 3.1 Características

La robótica de enjambre tiene las siguientes características:

- Los robots son autónomos.
- Los robots están situados en un entorno y pueden realizar acciones y modificarlo.
- Los sensores y actuadores de los robots son locales.
- Los robots no pueden tener acceso a un control centralizado o a información global.
- Los robots cooperan para llevar a cabo una tarea.

La principal inspiración de la robótica de enjambre proviene del comportamiento observado en aves, abejas, peces, en general a comportamientos del reino animal. El interés de estos animales sociales proviene del hecho de que exhiben una especie de inteligencia de enjambre. En particular, el comportamiento de estos animales parece ser escalable, robusto y flexible.

**Diseño robusto** es la capacidad de superar la pérdida de individuos. En animales sociales la robustez es promovida por la ausencia de un líder. Por esta razón, los robots no deben seguir órdenes de un sistema centralizado, ya que, la pérdida de este sistema podría causar la inutilidad del enjambre.

**Escalabilidad** es la capacidad de poder desempeñar una tarea con enjambres de diferentes tamaños. La pérdida o el aumento del enjambre no debería causar ningún cambio drástico en el comportamiento del enjambre. En el reino animal esto se puede llevar a cabo siguiendo una comunicación y percepción sensorial de forma local.

**Flexibilidad** es la capacidad de hacer frente a un amplio espectro de entornos y tareas. En el reino animal se consigue esta característica debido a la simplicidad de las tareas de cada individuo que pertenece al enjambre.

Los robots que pertenecen a un enjambre deben ser robustos, escalables y flexibles para poder comportarse como un enjambre, es decir, deben seguir la inteligencia de enjambre del

reino animal que caracteriza a los animales sociales.

## 3.2 Métodos de diseño de comportamiento

El diseño es la fase en la que se planea y desarrolla un sistema a partir de requisitos y especificaciones iniciales. En la robótica de enjambre no existe ninguna forma precisa de establecer un comportamiento individual que consiga el comportamiento colectivo deseado. Actualmente hay dos categorías de métodos de diseño de comportamientos: Basados en el comportamiento y método de diseño automático.

El método de diseño **basado en el comportamiento** tiene tres categorías principales: Diseño probabilístico de máquinas de estados finitos (PFSMs), diseño virtual basado en la física y otro tipo de diseño como programación en un medio computacional amorfo. Este último método considera que los individuos tienen capacidad de cálculo y pueden comunicarse con los individuos vecinos. En ocasiones, el método de diseño basado en el comportamiento sigue un proceso de prueba y error, ya que, hay que ajustar ciertos parámetros de los robots para que realicen la tarea indicada. Además, este método se inspira en el comportamiento de los animales sociales, lo cual es una ventaja porque existen modelos matemáticos y un comportamiento particular que definen esos comportamientos. Además, el proceso de diseño de este método suele seguir la estrategia *bottom-up*, a partir del comportamiento individual consiguen diseñar un comportamiento colectivo, aunque se puede utilizar la estrategia *top-down*, a partir del comportamiento colectivo deseado se consigue obtener el comportamiento individual.

Por otra parte, el método de diseño **automático** se utiliza para generar comportamientos sin necesidad de que ningún desarrollador intervenga en el proceso. En este método hay dos categorías: Aprendizaje por refuerzo y robots evolutivos. El uso de estas dos categorías se explicará más adelante.

## 3.3 Métodos analíticos de comportamiento

El análisis es una fase esencial en el proceso de la ingeniería. En esta fase se comprueba si el comportamiento obtenido es válido o no. El objetivo final es obtener un enjambre de robots reales que se comportan de la forma deseada y con las propiedades deseadas. Las propiedades de los comportamientos colectivos suelen analizarse mediante modelos microscópicos, macroscópicos o mediante el análisis de los robots reales.

Los **modelos microscópicos** tienen en cuenta a cada robot individualmente, la interacción robot-robot y la interacción robot-entorno. En el campo de la robótica de enjambre se han desarrollado varios niveles de abstracción. El más simple de todos es considerar cada robot como un punto de masa, en entornos 2D tienen fuerzas cinemáticas y en entornos 3D también tienen fuerzas dinámicas y los detalles de cada sensor y actuador son modelados. En los modelos microscópicos los robots son simulados y en las simulaciones se utilizan herramientas para verificar que el comportamiento colectivo obtenido es el correcto.

Los **modelos macroscópicos** consideran el comportamiento del enjambre en conjunto y los elementos de los robots individuales no se suelen tener en cuenta para la descripción del sistema en alto nivel. En este modelo se consideran tres categorías: trabajos que recurren a ecuaciones o diferenciales, trabajos donde se utiliza la teoría clásica de control y estabilidad para probar las propiedades del enjambre y otro tipo de enfoques como trabajos modelados que recurren a otro tipo de métodos matemáticos.

El **análisis de robots reales** es una tarea imprescindible, ya que, es inviable simular todos los aspectos del robot. En los entornos reales, los sensores y actuadores pueden percibir ruido de la información exterior, por este motivo, se debe comprobar la robustez de los sistemas de robótica de enjambre observando cuál es su tolerancia al ruido externo. Además, al trabajar con los robots reales podemos diferenciar entre comportamientos colectivos posibles en la práctica y entre comportamientos que son posibles siempre y cuando hagamos suposiciones no realistas.

## 3.4 Comportamientos colectivos

Estos comportamientos colectivos son comportamientos básicos de un enjambre que podrían combinarse con otras tareas más complejas. Estos comportamientos se pueden dividir en cuatro categorías principales: Organización del espacio, navegación, decisión-fabricación colectiva y otro tipo de comportamientos que no pertenecen a ninguna de estas categorías.

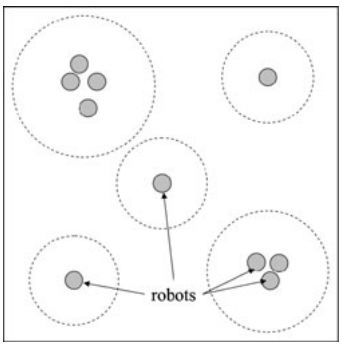
### 3.4.1 Organización del espacio

La **organización del espacio** es un tipo de comportamiento colectivo que se centra en distribuir robots y objetos en un espacio. Los robots se pueden distribuir de distintas formas como agregados, formando cadenas o patrones y estructuras de robots conectados físicamente.

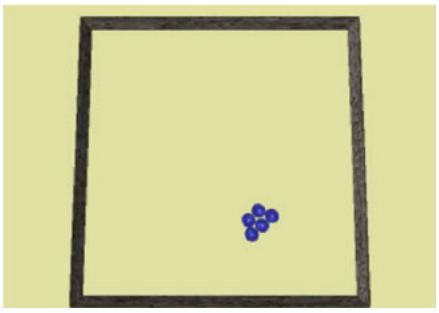
La tarea de **agregación** tiene como objetivo poder agrupar a un enjambre de robots en una región. A pesar de ser un comportamiento colectivo simple, forma un bloque de robots muy útil, ya que, mantiene a los robots muy cerca entre si para que puedan interactuar entre ellos. La agregación es un comportamiento muy común en la naturaleza y suelen realizarlo las bacterias, cucarachas, abejas, peces y pingüinos. En la robótica de enjambre se suele utilizar para el diseño de esta tarea PFSMs o evolución artificial.

La **formación de patrones** tiene como objetivo desplegar robots de manera repetitiva y regular. Los robots necesitan mantener ciertas distancias entre si para formar el modelo deseado. Esta tarea se puede encontrar tanto en la física como en la biología, los ejemplos biológicos son la disposición espacial de las colonias bacterianas y el patrón cromático de golondrinas de mar en el pelaje de algunos animales, mientras que en la física se puede encontrar en la distribución de las moléculas y en la formación de cristales. El método de diseño más utilizado en este comportamiento es el diseño virtual basado en la física. Este método utiliza las fuerzas virtuales para coordinar los movimientos de los robots.

En la **formación en cadena** los robots deben posicionarse en orden para conectar dos puntos. La cadena que forman se puede utilizar como guía para la navegación o la vigilancia.



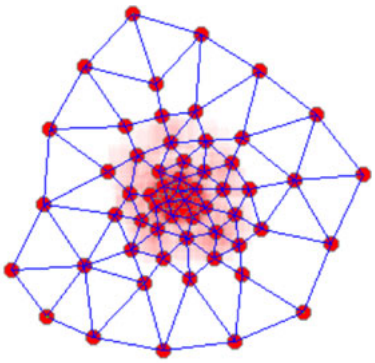

**(a)** De SOYSAL y cols. (2007)



**(b)** De Trianni y cols. (2003a)

**Figura 3.1:** Ejemplos de la tarea de agregación en robótica de enjambre

**(a)** De Spears y Spears (2012)

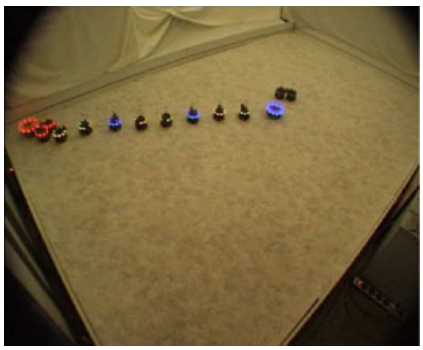


**(b)** De TShucker y Bennett (2007)

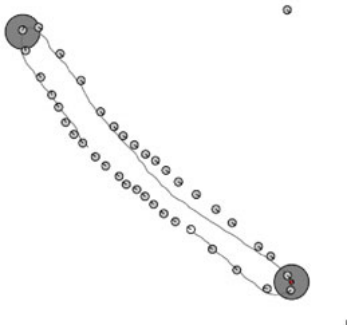
**Figura 3.2:** Ejemplos de formación en patrones en robótica de enjambre

Este comportamiento intenta imitar las cadenas que forman las hormigas. Los métodos de diseño más utilizado en este comportamiento son: PFSMs, diseño virtual basado en la física y evolución artificial. En PFSMs se tienen en cuenta 2 roles: Explorador, que es el primer miembro que aparece en la cadena y guía el movimiento del resto de robots, y miembro de la cadena, mientras que en el diseño virtual basado en físicas se utiliza la distancia entre los robots para modelar el comportamiento.

El **autoensamblaje** es el proceso por el cual los robots se conectan físicamente unos con otros. Este comportamiento tiene diferentes propósitos como aumentar la estabilidad al navegar en terrenos irregulares o para aumentar el poder de atracción entre los robots. La **morfogénesis** es el proceso que lleva a los robots a autoensamblarse después de formar un patrón determinado. Los robots pueden formar una estructura para realizar una determinada tarea como formar una línea para poder cruzar un puente. El autoensamblaje se inspira en algunas especies de hormigas que son capaces de conectarse físicamente, también se estudia en la auto-organización de las células para formar tejidos y órganos. Desde la perspectiva de la robótica de enjambre hay dos principales desafíos: cómo autoensamblarse en una estructura determinada, es decir, morfogénesis, y cómo controlar la estructura obtenida para abordar tareas específicas. Los trabajos que se centran en el primer desafío se basan en PFSMs y en



**(a)** De Nouyan y cols. (2009)



**(b)** De Sperati y cols. (2011)

**Figura 3.3:** Ejemplos de formación en cadena en robótica de enjambre

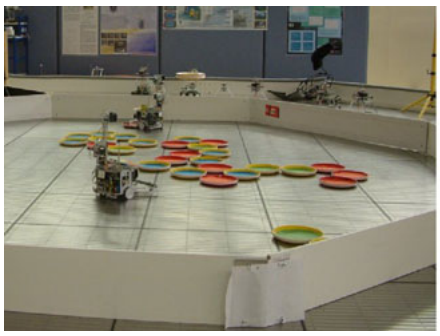
la comunicación para la coordinación. En cambio, los trabajos que se centran en el segundo desafío utilizan PFSMs o evolución artificial.

**(a)** De Christensen y cols. (2008)

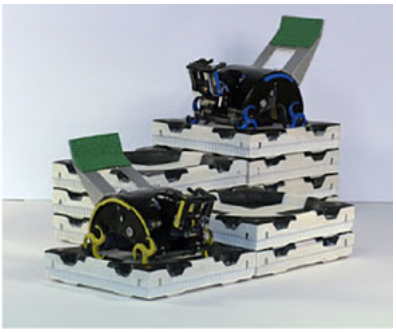
**(b)** De Mondada y cols. (2005)

**Figura 3.4:** Ejemplos de autoensamblaje y morfogénesis en robótica de enjambre

En el **agrupamiento y montaje de objetos** los robots están inicialmente dispersos en un entorno y pueden realizar dos tareas, agrupación o ensamblaje de los objetos, este comportamiento tiene como objetivo agrupar objetos. La diferencia entre agrupar y ensamblar los objetos es que en el primer caso los objetos no están conectados físicamente mientras que en el ensamblaje sí. Este comportamiento es fundamental en cualquier proceso de construcción. Este comportamiento está presente en el reino animal, como en las colonias de hormigas o termitas. En la robótica de enjambre, normalmente se aborda este problema utilizando PFSMs. Los robots exploran el entorno al azar y reaccionan de diferentes maneras al descubrir un objeto o parte de un cluster/ensamblaje de objetos ya formado. En muchos trabajos, esta tarea se realiza de manera secuencial, ya que, el realizarlo de manera paralelizada podría provocar colisiones o interferencias. Para evitar este problema, un robot evita que otros depositen objetos al mismo tiempo que él mediante comunicación o bloqueando físicamente el acceso al cluster.



**(a)** De Melhuish, Welsby, y Edwards (1999)



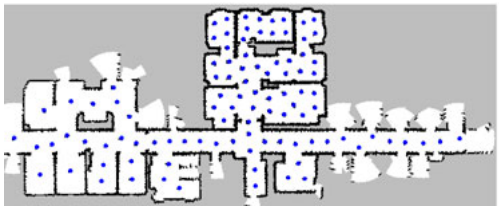
**(b)** De Werfel (2011)

**Figura 3.5:** Ejemplos de agrupamiento y montaje de objetos en robótica de enjambre

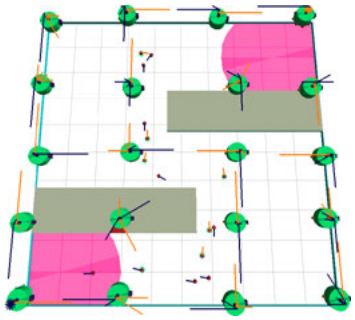
### 3.4.2 Navegación

La tarea de **navegación** se enfrenta al problema de coordinar los movimientos del enjambre.

La **exploración colectiva** es un comportamiento colectivo en el que los robots cooperan para explorar un entorno mientras navegan. Los comportamientos colectivos que pueden realizar esta tarea son: cobertura del área y navegación guiada por enjambres. El objetivo de cubrir el área es poder crear una cuadrícula regular o irregular de robots para que puedan comunicarse. La cuadrícula resultante se puede utilizar para monitorear el ambiente o para guiar a otros robots. Ambos comportamientos colectivos están fuertemente relacionados, por lo tanto, muchos trabajos se centran en diseñar los dos. La exploración colectiva está presente en el reino animal, por ejemplo, las hormigas usan rastros de feromonas para encontrar la ruta más corta entre dos puntos y las abejas comunican directamente los destinos mediante determinados movimientos. En la robótica de enjambre, es muy común utilizar el método de diseño virtual basado en físicas para poder obtener la cuadrícula de robots que cubre el entorno. Mientras que la navegación guiada se centra en la comunicación, por lo que utiliza PFSMs.



**(a)** De Howard y cols. (2002)

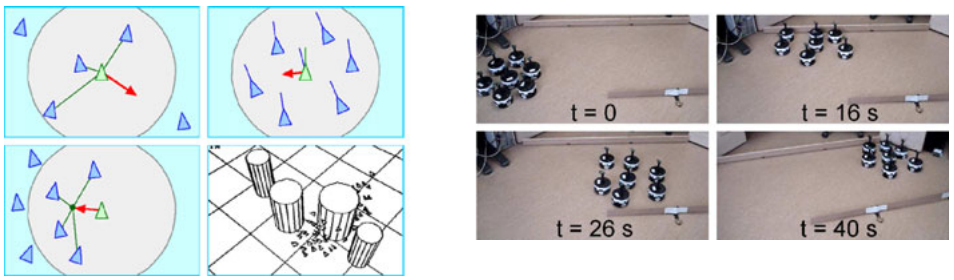


**(b)** De Ducatelle y cols. (2011)

**Figura 3.6:** Ejemplos de exploración colectiva en robótica de enjambre

El **movimiento coordinado** se utiliza para que los robots se muevan juntos como las

bandadas de pájaros o los bancos de peces. Este comportamiento es muy útil para navegar en un entorno con colisiones limitadas o sin colisiones entre los robots. Además, mejora las habilidades de detección del enjambre. Este comportamiento está inspirado en el observado en las bandadas de pájaros o en los bancos de peces y tiene la ventaja de aumentar la supervivencia del enjambre o aumentar la precisión de la navegación. Este comportamiento suele ser diseñado mediante el método de diseño virtual basado en físicas. Los robots deben mantener una distancia constante entre sí y una alineación uniforme mientras se mueven. Los movimientos coordinados también se pueden obtener mediante evolución artificial.




**(a)** De Reynolds (1987) **(b)** De Turgut y cols. (2008)

**Figura 3.7:** Ejemplos de movimiento coordinado en robótica de enjambre

En el **transporte colectivo** los robots cooperan para transportar un objeto que, en general, es demasiado pesado como para que lo pueda mover un solo robot. Los robots se tienen que poner de acuerdo en una dirección común para mover el objeto hacia un punto objetivo. Se inspira en el comportamiento de las hormigas, cuando estas encuentran un objetivo se ponen de acuerdo para empujarlo a una dirección y si ven que no cambia de posición cambian de dirección a la que empujar. En la robótica de enjambre, se puede diseñar este comportamiento mediante evolución artificial o PFSMs. La cooperación se obtiene mediante comunicación explícita de la dirección deseada o mediante comunicación indirecta, es decir, midiendo la fuerza aplicada al objeto transportado por los otros robots.

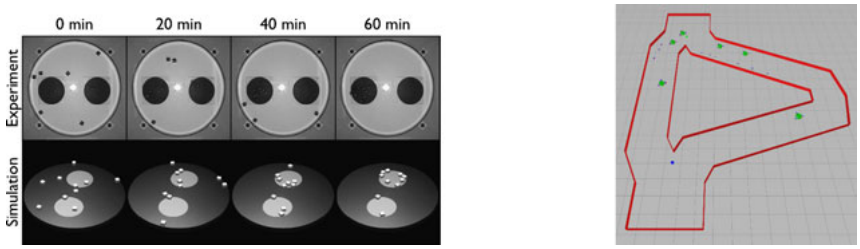
**(a)** De Campo y cols. (2006) **(b)** De Baldassarre (2006)

**Figura 3.8:** Ejemplos de transporte colectivo en robótica de enjambre

### 3.4.3 Decisión-fabricación colectiva

La tarea de **decisión-fabricación colectiva** trata de como los robots influyen entre si para tomar una decisión. Se puede utilizar para solucionar dos problemas opuestos: acuerdo y especialización. Hay dos tipos de tareas en las que se centra esta tarea: Toma consensuada de decisiones y asignación de tareas.

La **toma consensuada de decisiones** tiene como objetivo que los robots se pongan de acuerdo en tomar una decisión entre varias disponibles. La decisión suele ser la que maximiza el rendimiento del sistema. Esta tarea es generalmente difícil para un enjambre de robots, ya que, la mejor opción puede cambiar en el tiempo o los robots no pueden detectar cuál es debido a que sus capacidades de detección son limitadas. Este comportamiento se encuentra en algunas especies de insectos, como en las hormigas, que utilizan feromonas para decidir cuál de las dos rutas es más corta, también se puede encontrar en las abejas, que deciden cuál es el mejor alimento o la mejor localización disponible. En la robótica de enjambre hay dos categorías y dependen de la comunicación que se utilice. En la primera categoría se utiliza comunicación directa: cada robot puede comunicar su comunicación preferida o alguna información relacionada. En la segunda categoría se utiliza comunicación indirecta y la decisión se toma a través de ciertas pistas como la densidad de la población de los robots.




**(a)** De Garnier y cols. (2005)



**(b)** De Montes de Oca y cols. (2011)

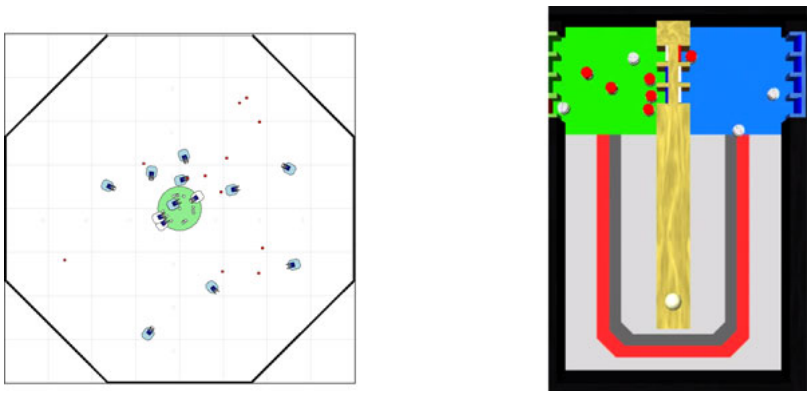
**Figura 3.9:** Ejemplos de toma consensuada de decisiones en robótica de enjambre

En la **asignación de tareas** los robots se distribuyen diferentes tareas con el objetivo de maximizar el rendimiento del sistema mediante la elección dinámica de las tareas que desempeñarán los robots. Este comportamiento está presente en las abejas y en las colonias de hormigas, sus decisiones, en principio, son fijas aunque, pueden cambiar en el tiempo. En la robótica de enjambre se suele obtener este comportamiento mediante PFSMs. Para promover la especialización, las probabilidades de seleccionar una de las tareas disponibles son diferentes entre los robots o pueden cambiar en respuesta de ejecución de tareas o mediante mensajes de otros robots.

### 3.4.4 Otro tipo de comportamientos

Hay otro tipo de comportamientos colectivos que no se pueden clasificar en los mencionados anteriormente. A continuación se describen algunos de ellos.

La **detección colectiva de fallos** tiene como objetivo que el enjambre detecte si uno de



**(a)** De Liu (2007) **(b)** De Pini (2011)

**Figura 3.10:** Ejemplos de asignación de tareas en robótica de enjambre

los robots presenta fallos o no. El trabajo que ha implementado Christensen y cols. (2009) funciona de la siguiente manera: Todos los robots emiten una señal de manera asíncrona, de esta forma, pueden detectar si un robot está defectuoso recibiendo su señal. Si un robot no está sincronizado, se asume que está defectuoso.

La **regulación del tamaño del grupo** es la capacidad de seleccionar o crear un grupo de un tamaño determinado. Esta tarea puede maximizar el rendimiento para realizar otras tareas. El trabajo que ha implementado Melhuish, Holland, y Hoddell (1999) está inspirado en las luciérnagas. Cada robot emite una señal en un periodo de tiempo aleatorio y después cuentan el número de señales recibidas. Este número puede servir para determinar el tamaño del enjambre.

La **interacción humano-enjambre** tiene como objetivo que un operario humano pueda recibir información del enjambre de robots para conocer el estado del sistema. McLurkin y cols. (2006) desarrolló un mecanismo simple en el que el robot podía proporcionar información mediante el uso de LEDs y sonido.

## 3.5 Robótica de enjambre con computación evolutiva

Como hemos comentado anteriormente, se puede modelar un comportamiento colectivo mediante deep learnig con métodos automáticos. Las dos técnicas más utilizadas son el aprendizaje por refuerzo y los robots evolutivos.

El **aprendizaje por refuerzo** es un conjunto de algoritmos de aprendizaje. En estos algoritmos un agente explora el entorno mediante una serie de acciones y recibe penalizaciones o recompensas por cada acción, obteniendo al final de cada episodio la puntuación total obtenida con las acciones del agente. En esta estrategia el robot recibe recompensas por sus acciones y el objetivo global es que el robot aprenda una política óptima. El comportamiento es óptimo en el sentido de que maximiza las recompensas recibidas del entorno. Es una tarea difícil poder considerar un problema de robótica de enjambre como un problema de aprendizaje por refuerzo, ya que, nos interesa modelar el comportamiento individual de los

robots pero con esta estrategia se establecen las recompensas y penalizaciones en base al comportamiento colectivo. Para solucionar este problema se deben descomponer la recompensa global en recompensas individuales. Aunque esta solución tiene dos problemas: El espacio de búsqueda al que se enfrenta este aprendizaje es grande y esto se debe a la complejidad del hardware de los robots y la complejidad de la interacción robot-robot. El otro problema es que la percepción del entorno es incompleta, esto provoca que la búsqueda del comportamiento sea más compleja.

Los **robots evolutivos**, o evolución artificial, aplican una técnica evolutiva computacional a un solo robot o a un sistema multi-robot. Este método está inspirado en el principio biológico Darwiniano de la selección y evolución natural. Dentro de la robótica de enjambre se han utilizado varias tareas de prueba para probar la efectividad de este método o como una herramienta para contestar preguntas científicas más fundamentales. Este método sigue los siguientes pasos.

1. Al principio, en la primera generación, se genera una población de individuos cuyos parámetros se han inicializado de forma aleatoria.
2. En cada iteración se ejecuta un número de experimentos por cada individuo.
3. El mismo comportamiento individual del experimento se ejecuta en todos los robots del enjambre.
4. En cada experimento se obtiene una función de fitness o recompensa del comportamiento colectivo como resultado de cada acción individual.
5. Una vez obtenemos las recompensas de todos los individuos de la población se seleccionan los individuos que han obtenido las mejores en la población, se cruzan para obtener individuos descendientes y se mutan.
6. Se vuelve a empezar el proceso desde el paso 2.

El comportamiento individual de los robots se puede representar mediante una máquina de estados finitos o mediante funciones de fuerzas virtuales. Normalmente, los métodos evolutivos se utilizan para encontrar los parámetros de una red neuronal.

Las desventajas del uso de robots evolutivos son las siguientes:

- La evolución es un proceso computacionalmente intensivo y no garantiza la convergencia a una solución.
- Las redes neuronales se comportan como una caja negra (black-box) y a veces es muy complicado entender su comportamiento.
- Desde un punto de vista de la ingeniería, se pueden obtener los mismos resultados diseñando el comportamiento manualmente.

# 4 Entorno de trabajo

En este capítulo se explicarán las librerías y entornos que se han utilizado durante el desarrollo del proyecto, tanto en entornos simulados como en entornos reales. Además, se explicarán las librerías utilizadas para el aprendizaje de las conductas obtenidas.

## 4.1 Raspberry Pi

Raspberry es una serie de ordenadores de placa única de bajo coste. La placa utilizada en este proyecto corresponde al modelo Raspberry Pi 3, modelo B+, y tiene instalado el sistema operativo Lubuntu 16.04, aunque en este modelo se pueden instalar otros sistemas. Además, esta placa está formada por una serie de puertos como los puertos HDMI, USB 2.0 o Ethernet. También dispone de conectividad Bluetooth 4.2, BLE y Wi-Fi a doble banda 2.4 Ghz y 5 Ghz.

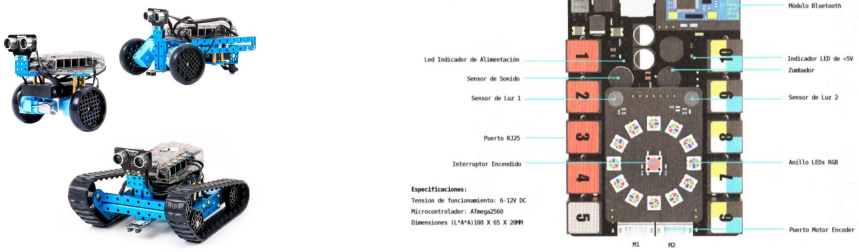
**Figura 4.1:** Placa base Raspberry Pi 3 modelo B+

## 4.2 Robot MBot Ranger

El robot Mbot Ranger es un robot educativo diseñado por Makeblock que dispone de una placa base Auriga y de una serie de sensores y actuadores internos o externos. Los motores que utiliza son motores con encoder de corriente continua que incluyen enconders ópticos en sus ejes principales que permiten monitorear la posición actual del eje. La velocidad se puede establecer en RPM, revoluciones por minuto. Además, se puede conectar una placa base Raspberry Pi, lo que facilita la conexión a otros sensores como puede ser un sensor de lectura láser.

Existen varias formas de programar el comportamiento de este robot que son: Mediante la aplicación móvil Makeblock, el programa mBlock para ordenador o mediante la librería

Aurigapy, creada por Fidel Aznar, que traduce los comandos de la placa Auriga a Python. Esta librería es la que se va a utilizar a lo largo del desarrollo del proyecto.




**(a)** Modelos bases de mBot Ranger **(b)** Placa base Auriga

**Figura 4.2:** Robot mBot Ranger

### 4.2.1 Sensores y actuadores

A continuación se describirán todos los sensores y actuadores del robot Mbot que se han utilizado en este proyecto.

**Sensor láser RPLidar**: Es un sensor de escaneo láser con una cobertura de 360 grados. Produce un escaneo omnidireccional de 360 grados y un rango de distancia de 0.15 a 12 metros. Además, se encuentra en rotación constante con velocidad regulable. El ángulo de apertura es de 1 grado y puede obtener 2000 muestras por segundo. Por último, este sensor se comunica con la placa Raspberry pi.


**Figura 4.3:** Robot mBot Ranger

**Motor con Encoder**: El robot dispone de dos motores de corriente continua, que funcionan a través de ser alimentados con un determinado valor de voltaje continuo entre sus terminales y producen un torque proporcional a dicha alimentación. Además incluyen encoders ópticos conectados a sus ejes principales lo cual permite monitorear la posición actual

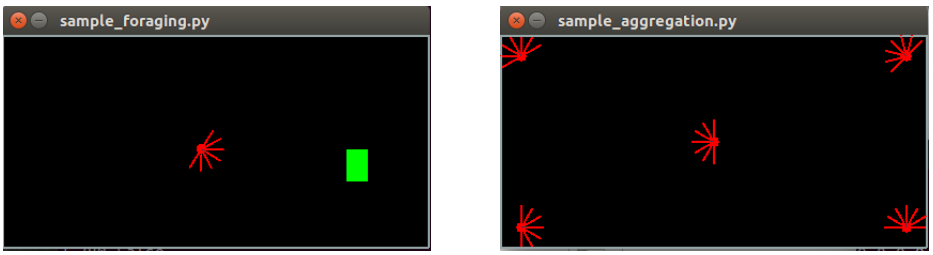
del eje. La dinámica de los encoder se basa en detectar el paso de luz a través de determinadas ranuras, a medida que el rotor va girando, con esta información se pueden calcular velocidades y posiciones absolutas de los ejes de los motores. A los motores se les puede indicar la velocidad de giro en revoluciones por minuto, RPM. Por último, estos motores se comunican con la placa Auriga del robot.


**(a)** Motor con cable para conexión **(b)** Zoom al eje del motor

**Figura 4.4:** Encoder Motor

## 4.3 Simulador SwarmSim

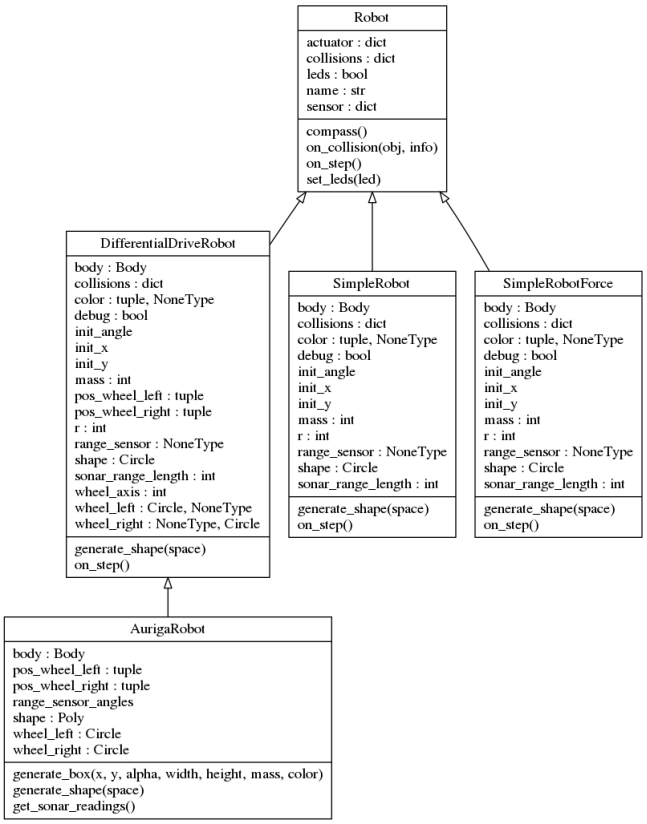
El simulador SwarmSim es un simulador en 2D del robot MBot Ranger escrito en Python con el framework Pymunk, esta librería permite añadir y utilizar físicas a un objeto en 2D. Este simulador incluye un modelo del robot con medidas y movimientos realistas, también utiliza física realista con el fin de simular correctamente los movimientos del robot. Además, contiene una implementación del láser, en el que se pueden definir tanto el número de rayos como el rango de lectura que se desea utilizar.



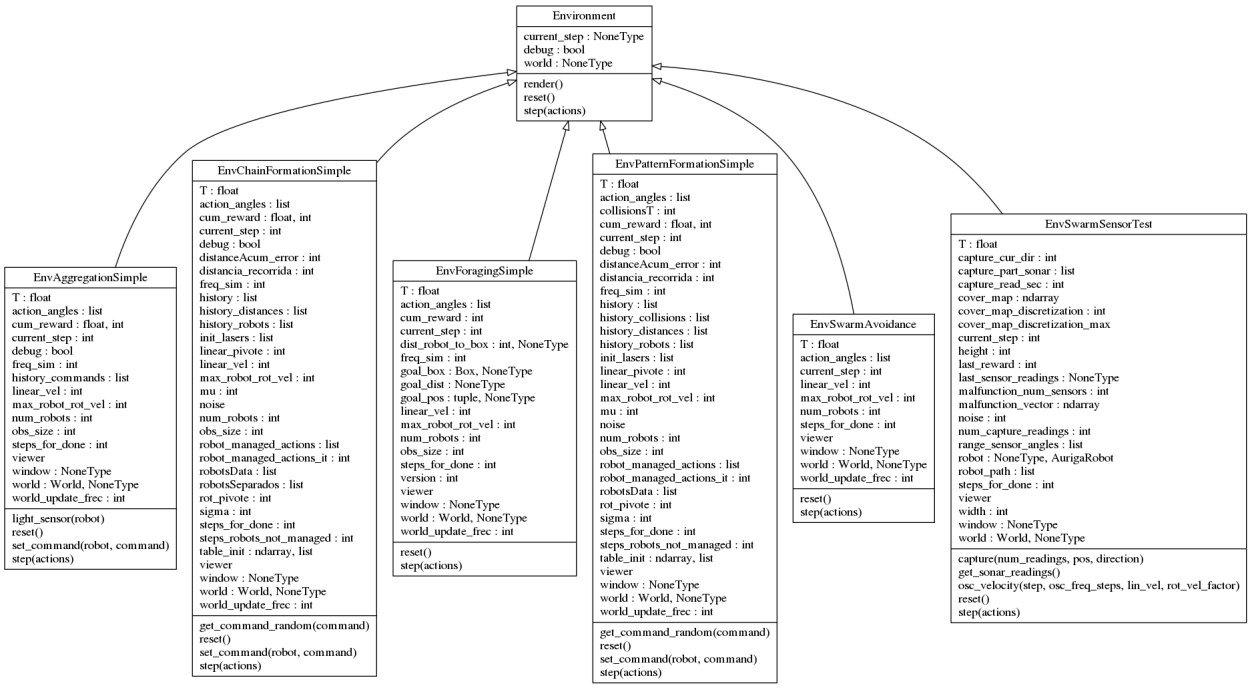

**(a)** Simulación de un robot en un escenario con un obstáculo **(b)** Simulación de un enjambre en un escenario sin obstáculos

**Figura 4.5:** Ejemplos de simulaciones con la librería SwarmSim

El simulador está formado por varias clases y entornos definidos. En la figura 4.6 se puede observar el diagrama de clases implementado para definir el Robot y los entornos. Los entornos creados para este proyecto son EnvChainFormationSimple y EnvPatternFormationSimple y se ha utilizado la clase SimpleRobotForce para definir los robots del enjambre.




**(a)** Diagrama UML de clases del robot. La clase de robot que se ha utilizado en este proyecto es SimpleRobotForce.




**(b)** Diagrama UML de clases del entorno. Los entornos creados para este proyecto son EnvChainFormationSimple y EnvPatternFormationSimple.

**Figura 4.6:** Diagramas UML de algunas clases utilizadas de la librería SwarmSim.

Por otra parte, se puede añadir más de un robot en la simulación, lo que significa es que el simulador permite la simulación de un enjambre de robots en un mismo entorno. Para el desarrollo de este proyecto se han implementado dos entornos adicionales (ver Fig. 4.7). El primer entorno es el utilizado en tarea de formación en cadena que está formado por un escenario cuadrado, simple sin obstáculos y 4 robots. El segundo entorno es el utilizado en la tarea de formación donde los robots del enjambre también esquivan obstáculos. Este entorno es un entorno cuadrado con 2 obstáculos cuya localización es aleatoria. Además, el enjambre está formado por 5 robots.

**(a)** Entorno utilizado para el aprendizaje de formación en cadena

**(b)** Entorno utilizado para el aprendizaje de formación con obstáculos

**Figura 4.7:** Ejemplos de entornos utilizados a lo largo de este proyecto.

## 4.4 Librería RLLib

RLLib es una librería de código abierto que se utiliza para el aprendizaje por refuerzo y es compatible con TensorFlow y PyTorch. Esta librería está escrita en Python, utiliza redes neuronales de la librería Keras y permite el uso de estrategias evolutivas como CMA-ES. Además, las ventajas que ofrece el uso de esta librería son:

- Paralelización del entrenamiento, es decir, ejecuta en paralelo las poblaciones con el fin de acelerar el proceso de entrenamiento.
- Almacenamiento de los resultados, tanto de la red neuronal entrenada como del historial de resultados
- Se puede restaurar el entrenamiento y continuarlo desde la última ejecución.
- Permite el entrenamiento de múltiples agentes, lo que nos facilita el entrenamiento de un enjambre.

Esta librería es compatible con la librería SwarmSim. Por lo tanto, para obtener una solución a un problema se crea un entorno de simulación con los robots deseados con SwarmSim y las acciones vendrán dadas según las recompensas obtenidas y los hiperparámetros de la red neuronal entrenada con RLLib.

### 4.4.1 Keras

Keras es una biblioteca de Redes Neuronales escritas en Python. Se puede ejecutar sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Esta librería está especialmente diseñada para posibilitar la experimentación en aproximadamente poco tiempo con redes de Aprendizaje Profundo. Sus puntos fuertes se centran en ser amigable para el usuario, modular y extensible. Entre las características de Keras se encuentra:

- Que contiene varias implementaciones de los bloques constructivos de las redes neuronales como por ejemplo las capas, funciones de activación u optimizadores matemáticos.
- Además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales Convolucionales y para las Redes Neuronales Recurrentes.
- Su código está alojado en GitHub y existen foros y un canal de Slack de soporte.

**Figura 4.8:** Logo de Keras

En este proyecto se ejecuta sobre TensorFlow y las versiones utilizadas de ambas librerías son Keras 2.3.1 y TensorFlow 2.1.0.

### 4.4.2 TensorFlow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Actualmente es utilizado tanto en la investigación como en los productos de Google. Además, proporciona una API de Python, así como APIs en lenguajes como C++ o Java.


**Figura 4.9:** Logo de TensorFlow

## 4.5 Hardware utilizado

A continuación, se muestra un resumen del hardware utilizado para el desarrollo del proyecto.

| | |
|---|---|
| CPU | Intel® Core™ i7-6500U 4 núcleos (8 hilos) |
| GPU | NVIDIA® GeForce® GTX 950M |
| RAM | 32 GB DDR4 |

**Tabla 4.1:** Especificaciones hardware

# 5 Estudio de estrategias evolutivas aplicadas al aprendizaje por refuerzo

En este capítulo se va a realizar un estudio del rendimiento de diferentes estrategias evolutivas para la optimización de políticas de aprendizaje de refuerzo profundo, DRL. La política se centrará en una importante tarea robótica de enjambre: la agregación de robots simples en un entorno sin obstáculos. Las estrategias que se van a estudiar son CMA-ES, PEPG, SES, GA y OpenES. Además, se utilizará un simulador rápido, basado en el robot diferencial Mbot Ranger. Una vez aprendida la tarea de agregación compararemos cada estrategia para diferentes tamaños de enjambre para analizar el tiempo de convergencia, la calidad de las políticas, su escalabilidad y su capacidad de generalización.

Por último, los resultados obtenidos en este capítulo se han publicado bajo el nombre *"Comparison of Evolutionary Strategies for Reinforcement Learning in a Swarm Aggregation Behaviour"* en la tercera edición del congreso **Conference on Machine Learning and Machine Intelligence (MLMI 2020)**.

## 5.1 Introducción

En este capítulo nos vamos a centrar en el modelado de la tarea de agregación. Esta tarea consiste en la agregación de un enjambre de robots disperso en el entorno utilizando la información de sus sensores locales. La tarea de agregación se estudia en diferentes temas como biología, robótica y teoría de control. En la naturaleza, la agregación es muy importante para la supervivencia de diferentes organismos como bacterias o insectos, ya que es la agregación de organismos la que permite a estos esconderse de los depredadores y encontrar a otros miembros de su misma especie. En algunos enjambres, los individuos agregan ciertos puntos caracterizados en el entorno como el nivel de temperatura en el caso de las abejas en los panales. En robótica, esta tarea puede considerarse como un requisito previo para abordar otra tarea como el movimiento coordinado, el autoensamblaje, la formación de patrones o el transporte colectivo de objetos.

El objetivo de este capítulo es comparar los resultados de las diferentes Estrategias Evolutivas en varios enjambres de diferente tamaño. Analizaremos cuál de ellas aprende más rápido y genera políticas de mayor calidad, teniendo en cuenta la escalabilidad y su capacidad de generalización.

## 5.2 Diseño del problema

La tarea de agregación se define en un entorno sin obstáculos para simplificar el comportamiento de los robots. Los robots son los principales componentes que interactúan con el entorno, que en este caso es desconocido, ya que, cada robot desconoce los movimientos que ejecutará el resto del enjambre. En cada paso, t, cada robot observa su estado, $s_t \in S$. Los posibles estados del problema están determinados por los datos capturados del sensor de luz, la brújula, el sensor de ultrasonidos y el comando de velocidad ejecutado anteriormente.

Una vez que los robots han percibido la información del entorno, se selecciona una acción $a \in A$, para pasar del estado $s_t$ al estado $s_{t+1}$ obteniendo una recompensa. Las posibles acciones que pueden realizar los robots son los movimientos: avanzar, girar a la derecha, girar a la izquierda y detenerse, y encender o apagar sus luces LED.

La función de recompensa se basa en la utilizada en la propuesta de SOYSAL y cols. (2007) ( 5.1).

$$Fitness = \frac{size(l)}{n_{robots}} \tag{5.1}$$

Donde $size(l)$ es el tamaño del cluster más grande que se ha formado. Para obtener los clusters formados por los robots, se calculan las conexiones entre los robots considerando el grupo como un grafo dirigido y calculando la conectividad con el algoritmo Floyd- algoritmo de Warshall, que encuentra el camino mínimo en un grafo ponderado. En este problema, los nodos son los robots y las aristas son la distancia entre cada par. Sin embargo, no todos los robots en el enjambre están conectados, es decir, considerarán las aristas de nodos vecinos. Además, se utilizarán las coordenadas (x, y) de cada robot para calcular las distancias entre sus vecinos. La función de recompensa se ha hecho de forma que no sea lineal, de esta forma premiando las mejores soluciones y penalizando las mediocres (5.2).

$$f(size(l)) = \begin{cases} (-10) & si \quad \frac{size(l)}{n_{robots}} = 1/n_{robots} \\ 10 \cdot \frac{size(l)}{n_{robots}} - 10 & si \quad \frac{size(l)}{n_{robots}} < 0.5 \\ 10 \cdot \frac{size(l)}{n_{robots}} & si \quad \frac{size(l)}{n_{robots}} \geq 0.5 \end{cases} \tag{5.2}$$

Como puede verse, consideramos la tarea de agregación como un Proceso de Decisión de Markov (MDP). Los MDP están bien adaptados para ser resueltos utilizando el Aprendizaje por Refuerzo.

La política de entrenamiento utiliza una red neuronal de clasificación para mapear la información de las entradas y salidas (ver Fig. 5.1). Esta red consta de 2 capas ocultas más las capas de entrada y salida, la función de activación de cada una de sus neuronas es tanh y obtiene la información del entorno discretizada para acelerar el proceso de acelerar el proceso de aprendizaje. Además, cada robot del enjambre ejecutará la misma red para conseguir un comportamiento colectivo generando los mismos criterios de decisión para las mismas tran-

siciones de estado. De este modo, buscamos obtener una política macroscópica que resuelva la tarea de agregación mediante una política microscópica.

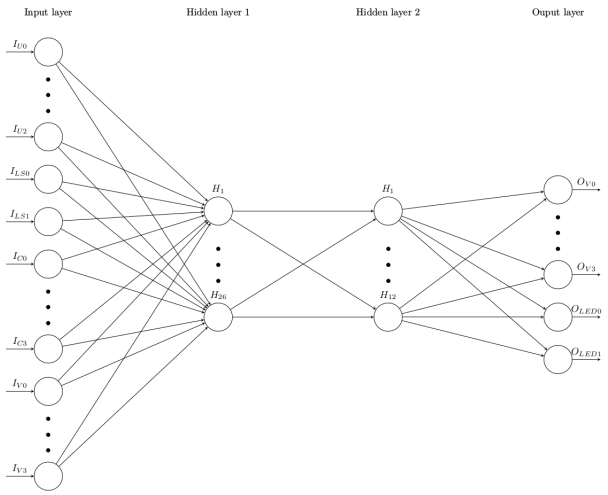



**Figura 5.1:** Red neuronal utilizada en este capítulo. Cada robot utilizará la misma red neuronal durante el entrenamiento, para obtener las acciones que debe realizar en cada paso.

## 5.3 Resultados

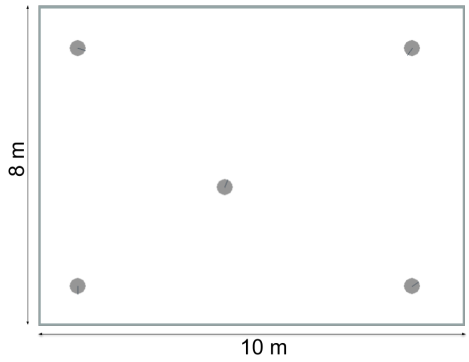
Para las pruebas se ha utilizado el simulador realista en 2D del Robot Mbot Ranger. Además, como se ha comentado anteriormente, estos robots utilizarán luces LED, sensores de luminosidad, sensores de ultrasonidos y una brújula para desarrollar esta tarea.

Al principio del entrenamiento, los robots estaban dispersos en un entorno cuadrado con los motores y las luces LED apagadas, como se puede ver en la figura 5.2. En todos los entrenamientos, el entorno tiene el mismo tamaño, 10x8 metros. Cada robot tiene 400 pasos para explorar el entorno y encontrar el enjambre agregado. Por lo tanto, a medida que aumenta el número de robots, aumenta el número de pasos de ejecución para resolver el problema.

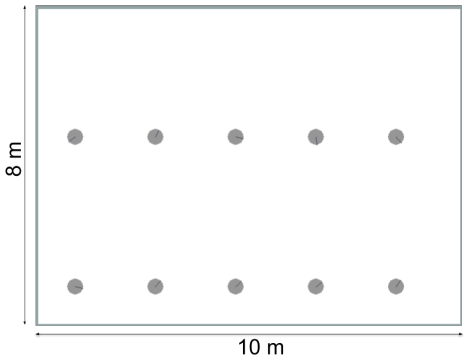
Un aspecto a tener en cuenta es que la resolución de este problema es más compleja cuando disminuye el tamaño del enjambre, ya que, en este caso, al agregarse ocupan una pequeña parte del entorno y los robots deben explorar una gran parte del mismo, para encontrar el enjambre. Por otro lado, al aumentar el tamaño del enjambre la resolución de esta tarea es menos compleja porque tienen que explorar menos terreno del entorno y así encuentran fácilmente a otros robots.

Como puede verse en la figura 5.3 todas las Estrategias Evolutivas consiguen obtener un comportamiento de agregación y alcanzar la máxima recompensa en enjambres de 5, 10, 20 y 40 robots.

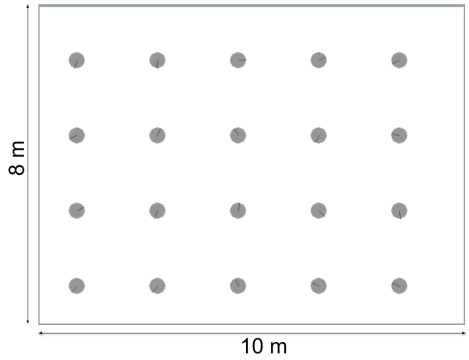
Por un lado, en los resultados obtenidos en las pruebas al ejecutar las redes neuronales entrenadas por el ES, se obtiene que las estrategias se diferencian en la estabilidad del cluster (figura 5.4). En algunas pruebas algunos robots del cluster agregado se separaron del grupo y



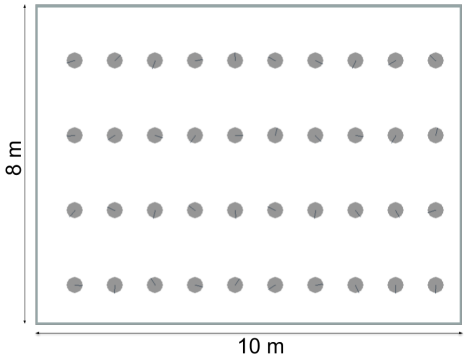

**(a)** Estado inicial de un enjambre de 5 robots




**(b)** Estado inicial de un enjambre de 10 robots




**(c)** Estado inicial de un enjambre de 20 robots




**(d)** Estado inicial de un enjambre de 40 robots

**Figura 5.2:** Posición de entrenamiento inicial. El escenario es un entorno cuadrado de 10x8 metros. En todas las figuras, los robots están dispersos por el entorno, de modo que si no se mueven durante la ejecución del problema, la recompensa sería mínima. Además, la posición de los robots es la misma en todas las ejecuciones, mientras que su orientación es aleatoria.

luego se agregaron de nuevo, esto podría llevar a cierta inestabilidad en el valor de la función de recompensa. Los resultados también dependen del tamaño del enjambre, es decir, el enjambre con 40 robots alcanza antes la recompensa máxima y el enjambre no se separa cuando se ha agregado completamente. Sin embargo, en el enjambre de 5 robots tarda más en agregarse el enjambre completo y la estabilidad del enjambre depende de la estrategia utilizada. En general, la estrategia que ofrece resultados más estables es CMA-ES y se obtiene más del 80% del enjambre agregado.

Por otro lado, en los resultados obtenidos en el entrenamiento, OpenAI-ES fue la estrategia evolutiva que encontró una política óptima en menos generaciones que las demás, en enjambres con 5, 10 y 20 robots. Por el contrario, en los enjambres de 40 robots, todas las estrategias evolutivas encontraron una política óptima en la primera generación.

Finalmente, en la figura 5.5 se representa la recompensa media obtenida para cada estrategia de ES y para cada tamaño de enjambre. En esta figura, se puede se observa que en los enjambres de 5, 10 y 20 robots se tarda más en resolver la tarea con la solución obtenida por la estrategia evolutiva estrategia evolutiva GA. Además, los resultados obtenidos por CMA-ES son superiores que las otras estrategias porque consigue resolver esta tarea en menos pasos de ejecución. En cambio, en un enjambre de 40 robots la estrategia que consigue los mejores resultados es SES, ya que resuelve en menos tiempo la tarea. Pero, la diferencia entre los resultados obtenidos es menor que en los enjambres anteriores.

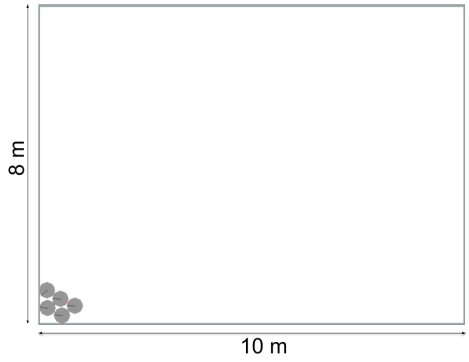
## 5.4 Conclusión

Se ha obtenido una política de comportamiento de agregación utilizando las estrategias evolutivas CMA-ES, PEPG, GA, SES y OpenAI-ES para el aprendizaje profundo por refuerzo. Esta política es capaz de agregar un enjambre robótico de diferente tamaño, con agentes robóticos muy básicos.

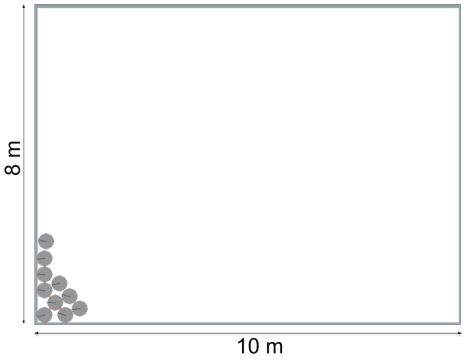
CMA-ES es la estrategia evolutiva que obtiene la mejor solución en enjambres de 5, 10 y 20 robots, ya que consigue resolver la tarea en menos pasos de ejecución. Por su parte, en enjambres de 40 robots la mejor solución la obtiene la estrategia SES. Además, a medida que aumenta el enjambre, la diferencia entre los resultados de las estrategias es menor.

En cuanto al entrenamiento, OpenAI-ES encontró una solución óptima en menos generaciones que el resto de ES en problemas complejos, es decir, en enjambres de 5, 10 y 20 robots. En cambio, en la agregación de enjambres de 40 robots, que puede considerarse una tarea más sencilla que con enjambres más pequeños, todos los ES alcanzaron una solución óptima desde el principio del entrenamiento.

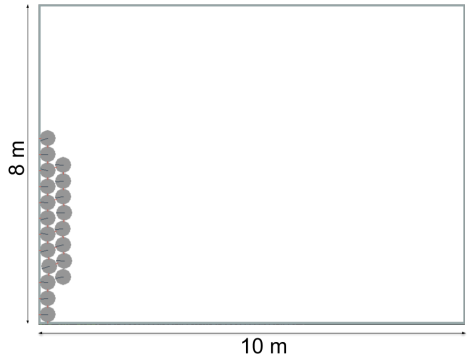
Finalmente, podemos concluir que en problemas menos complejos de resolver como una tarea de agregación con un enjambre de gran tamaño, las estrategias evolutivas que logran una política óptima son CMA-ES y SES. Por el contrario, en problemas complejos, como el entrenamiento con un enjambre de 5 robots, las estrategias que consiguen una política óptima son CMA-ES y OpenAI-ES. Aunque OpenAI-ES requiere menos tiempo de entrenamiento para encontrar una política óptima, la solución que ofrece requiere más pasos de ejecución para ejecución para resolver la tarea que las soluciones obtenidas con las estrategias CMA-ES o SES.



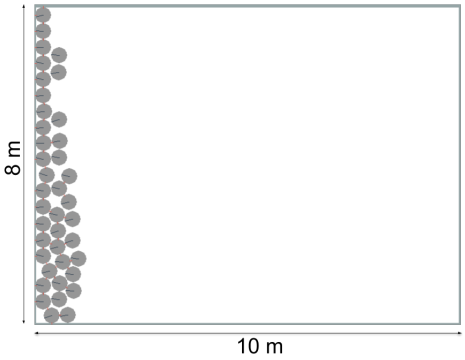

**(a)** Resultado en un enjambre de 5 robots




**(b)** Resultado en un enjambre de 10 robots

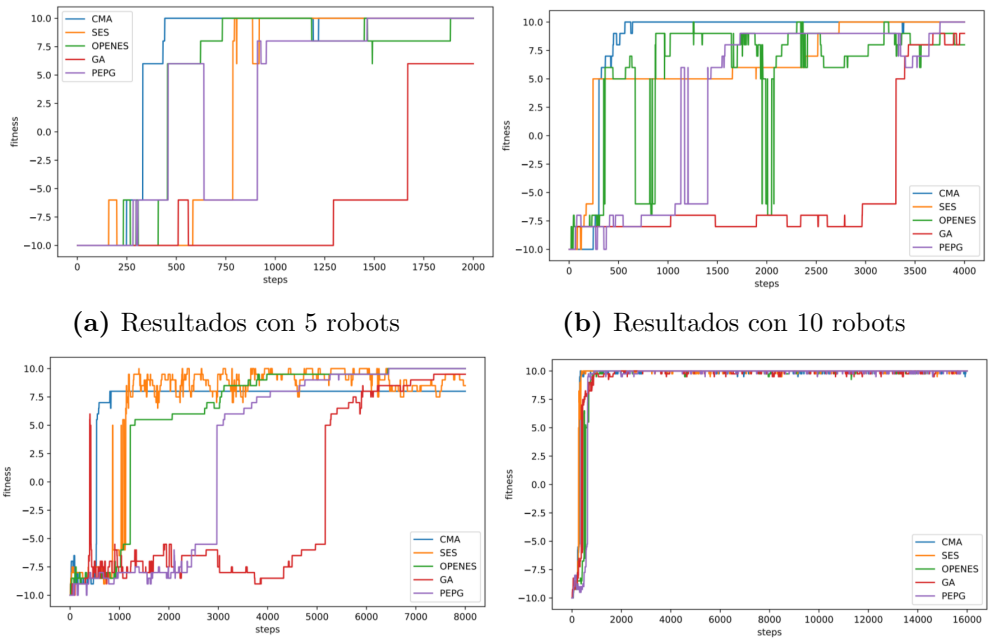



**(c)** Resultado en un enjambre de 20 robots




**(d)** Resultado en un enjambre de 40 robots

**Figura 5.3:** Posición final tras obtener el comportamiento de agregación en enjambres de 5, 10, 20 y 40 robots. El comportamiento resultante, en general, es que los robots giran hacia el Sur, colisionan en la pared inferior, luego giran hacia el Oeste y acaban colisionando en la esquina inferior izquierda. En todos los enjambres se alcanza la máxima recompensa, 10 puntos. Los resultados de las figuras se han obtenido con la solución CMA-ES. Sin embargo, las otras estrategias obtienen los mismos resultados y el mismo comportamiento.




**(a)** Resultados con 5 robots

**(b)** Resultados con 10 robots

**(c)** Resultados con 20 robots

**(d)** Resultados con 40 robots

**Figura 5.4:** Recompensas obtenidas en cada paso de ejecución durante la ejecución de las mejores soluciones encontradas con las estrategias evolutivas utilizadas. (a) En un enjambre con 5 robots CMA-ES consigue que los robots se agreguen en menos tiempo que utilizando las soluciones de las otras estrategias. (b) Del mismo modo, en un enjambre con 10 robots, es CMA-ES la que obtiene los mejores resultados, pero, con esta solución, los robots se separan del enjambre durante cortos periodos de tiempo. (c) En un enjambre de 20 robots la estrategia que obtiene la mejor recompensa antes que las demás es SES. (d) En un enjambre de 40 robots, las estrategias que ofrecen los mejores resultados antes que los obtenidos por las demás estrategias son CMA-ES y SES.

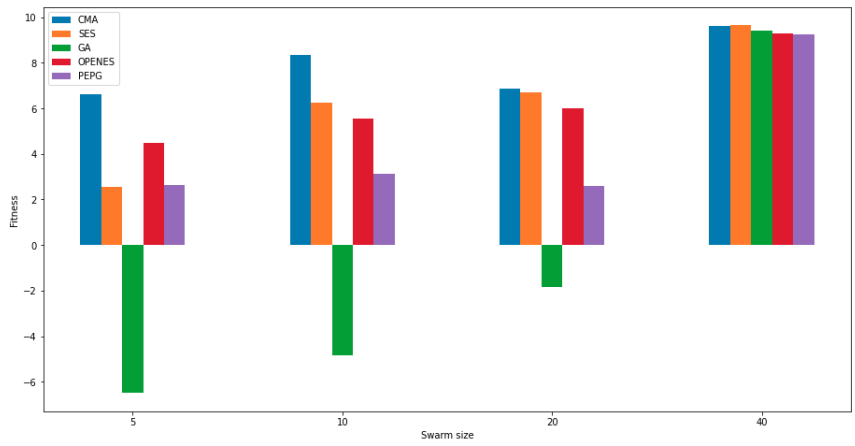



**Figura 5.5:** Resultados obtenidos al calcular la media entre las recompensas obtenidas y los pasos de ejecución. En un enjambre con 5, 10 y 20 robots la estrategia evolutiva que ofrece los mejores resultados es CMA-ES porque consigue agregar a todos los miembros del enjambre. Sin embargo, con GA se obtiene el peor resultado porque agrega al enjambre más tarde que con el resto de soluciones. Por su parte, en un enjambre de 40 robots, los mejores resultados se obtienen con la solución obtenida por SES. Además, al aumentar el número de robots en el enjambre disminuye la diferencia entre los resultados obtenidos por cada estrategia.

# 6 Aprendizaje de una conducta formación en cadena

En este capítulo se estudiará la formación en cadena de un enjambre, es decir, los robots deberán seguir al robot líder mientras mantienen la forma inicial de una cadena. Para realizar esta tarea los robots utilizarán los datos obtenidos del sensor láser para ver la distancia que hay entre el robot delantero y eligirán los valores de velocidad lineal y rotacional para desplazarse por el entorno. Se evaluará también el uso de acciones clasificadas para simplificar el problema. Además, para el proceso de entrenamiento se utilizará la estrategia CMA-ES para el aprendizaje por refuerzo. Por último, se aporta una prueba de esta conducta con el sistema real.

## 6.1 Introducción

El objetivo de este capítulo es conseguir aprender una conducta donde los robots del enjambre forman una cadena mientras siguen los movimientos del robot líder. En dicha tarea existen dos tipos de roles, el del robot de explorador o líder y el del robot cadena. La función del robot explorador es explorar el terreno y decidir qué movimientos realizar en cada instante, mientras que la función de los robots cadena es mantener la forma de cadena en cada instante siguiendo los movimientos del robot líder. Además, esta tarea tiene diversas aplicaciones, como puede ser el transporte colectivo de objetos o la exploración de un entorno.

Se ha elegido la tarea de formación en cadena para el desarrollo de este capítulo porque tiene características en común con la tarea principal del proyecto que es el mantenimiento de una forma dada inicialmente. Para empezar, en ambas tareas existe el rol del robot explorador y el del robot cadena. Además, en ambas tareas el principal objetivo es mantener la forma dada en el inicio de la ejecución del problema, con la diferencia de que en la tarea principal del proyecto se pretende mantener formaciones más complejas que la de la cadena.

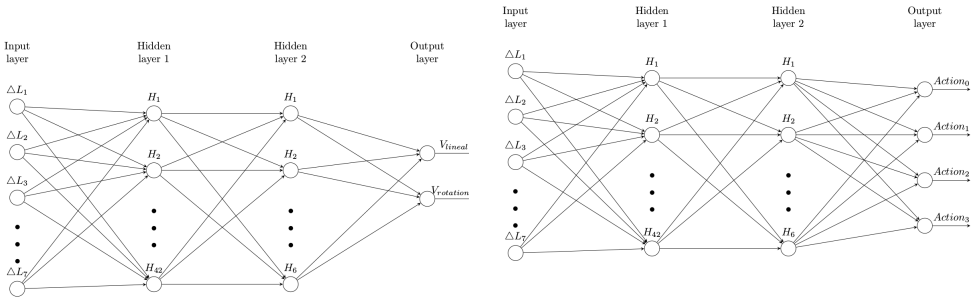
Por último, se ha utilizado la estrategia evolutiva CMA-ES porque en los resultados obtenidos en el Capítulo 5 podemos observar que dicha estrategia alcanza los mejores resultados al resolver tareas complejas, como se puede considerar a la tarea de formación en cadena.

## 6.2 Diseño del problema

Para el diseño de esta tarea se ha utilizado un entorno simple y sin obstáculos, con la finalidad de que los robots cadena solo tengan como función seguir al robot líder. Durante el proceso de aprendizaje, la ruta del robot líder es aleatoria, por lo tanto, los robots del

enjambre desconocen los movimientos que va a realizar el líder. Las acciones posibles que podrá ejecutar el robot líder son: ir hacia delante, ir hacia la derecha, ir hacia la izquierda y parar. La velocidad lineal o rotacional será un valor aleatorio delimitado entre un valor mínimo y máximo. En cada paso de ejecución, cada robot observará su estado $s_t \in S$ a través de un sensor láser de 180º de cobertura y 7 rayos con una separación de 30º. Los valores que se utilizarán del sensor láser es la diferencia que hay en las lecturas iniciales y actuales, con el objetivo de ver si los agentes están más cerca o lejos de los robots delanteros. A partir del estado actual, los agentes elegirán ejecutar una acción que maximice la recompensa global. Las posibles acciones $a_t \in A$ que podrán ejecutar vendrán dadas por los valores de velocidad lineal y rotacional de los motores. Aunque, para el problema con movimientos clasificados hay 4 posibles acciones a ejecutar: Avanzar, ir hacia la izquierda, ir hacia la derecha y parar. La velocidad de estos movimientos será constante y vendrá determinada por la acción a ejecutar. De esta forma, podemos considerar este problema como un Proceso de decisión de Markov (MDP), donde la posición de los robots delanteros pertenecen al espacio de estados, $S$, y los agentes deben elegir una acción que pertenezca al espacio de acciones, $A$, en función de un valor de recompensa $R_s$, que se explicará más adelante.

La política, $\pi$ que tomará las decisiones sobre qué acciones ejecutar en cada estado vendrá dada mediante el entrenamiento de una red neuronal por aprendizaje por refuerzo. Los pesos de la red serán obtenidos mediante la estrategia evolutiva CMA-ES. Además, para mapear la información del entorno se ha utilizado una red neuronal cuya arquitectura consta de 7 entradas, 2 capas ocultas y 2 salidas correspondientes a los valores de velocidad (ver Fig. 6.1). La conducta aprendida será una conducta homogénea, ya que, todos los robots del enjambre, a excepción del robot líder, ejecutarán la misma red entrenada.




**(a)** Red neuronal utilizada con movimientos definidos por las velocidades lineales y rotacionales.

**(b)** Red neuronal utilizada con movimientos clasificados en 4 acciones posibles.

**Figura 6.1:** Redes neuronales utilizadas durante el entrenamiento para el aprendizaje de una formación en cadena. Ambas redes constan de 7 entradas, que corresponden a la diferencia de las lecturas obtenidas entre el instante inicial y actual de la ejecución del problema. Además, contiene 2 capas ocultas con 42 y 6 neuronas, respectivamente.

### 6.2.1 Función de recompensa

En primer lugar, la función de recompensa inicial se basaba en la diferencia entre la distancia del láser leída al inicio de la ejecución y la distancia leída en el instante actual. Además, para evitar errores en la lectura del láser con los movimientos se ha tenido en cuenta la distancia con un margen de error, ya que, es más complicado que al desplazarse mantenga la mismas distancias exactas. Por lo tanto, para determinar si un láser ha mantenido la misma distancia desde el inicio se tiene en cuenta un margen de error de ± 5 cm. Si el láser se mantiene en ese margen de error, se considera que la lectura inicial y la lectura actual son iguales. Las siguientes fórmulas se calculan en cada robot del enjambre que sigue las acciones de la red neuronal.

$$distancia_{total} = \sum_{i=1}^{length(laser)} |l_{o_i} - l_{a_i}| \tag{6.1}$$

$$f(x) = \begin{cases} 1 & si \quad distancia_{total} \leq Margen\_error \\ -distancia_{total}/length(laser) & si \quad distancia_{total} > Margen\_error \end{cases} \tag{6.2}$$

Donde:

- $\boldsymbol{distancia_{total}}$: Es el total de la diferencia de la distancia leída de los rayos del láser.
- **length(laser)**: Es el número de rayos que contiene el láser.
- $\boldsymbol{l_o}$: Corresponde a las lecturas del láser en el instante inicial, es decir, antes de comenzar la ejecución.
- $\boldsymbol{l_a}$: Corresponde a las lecturas del láser en el instante actual.
- $\boldsymbol{Margen\_error}$: Es el margen de error permitido en centímetros para considerar que la lectura actual del láser es igual a la inicial.

Uno de los problemas que presentaba esta función de recompensa es que al leer la distancia entre los robots mediante los láseres, estos leían unas distancias distintas conforme el robot giraba. Por lo tanto, esta función solo sería útil si los robots solo avanzan o retroceden de manera lineal.

**(a)** Lectura inicial de los láseres de ambos robots. Donde el rayo que detecta el robot superior es el central.

**(b)** Lectura actual de ambos robots donde el robot inferior no contiene las mismas lecturas del láser.

**Figura 6.2:** Ejemplos de un posible error que causaría la función de recompensa inicial.

Como solución a este problema se propone una función de recompensa considerando que el enjambre forma un grafo no dirigido donde los robots son los nodos y las aristas son las distancias que hay entre ellos. De esta forma se obtiene una tabla donde en las filas y columnas se ubican los robots y el valor de cada elemento de la tabla es la distancia correspondiente. Entonces, la función de recompensa se obtiene comparando la tabla de distancias inicial con la del instante actual, manteniendo el margen de error de ± 5 cm. La distancia entre cada par de robots se calcula mediante la distancia euclidiana 6.3.

$$d = \sqrt{(X_{robot_i} - X_{robot_j})^2 + (Y_{robot_i} - Y_{robot_j})^2} \tag{6.3}$$

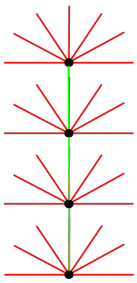


**Figura 6.3:** Grafo no dirigido formado por el enjambre. Los nodos son los robots y las aristas, representadas en color verde, son las distancias que hay entre ellos.

Las ecuaciones 6.4 y 6.5 se utilizan para calcular la función de recompensa.

$$\gamma = \sum_{i=1}^{N} \left( \sum_{j=1}^{N} |T_{o_{ij}} - T_{a_{ij}}| \right) \tag{6.4}$$

$$f(x) = \left(1 - \frac{\gamma}{\beta}\right) \tag{6.5}$$

Donde:

- $\boldsymbol{N}$:Es el número de robots del enjambre.
- $\boldsymbol{T_o}$: Es la tabla de distancias entre los robots en el instante inicial.
- $\boldsymbol{T_a}$: Es la tabla de distancias entre los robots en el instante actual.
- $\boldsymbol{\gamma}$: El resultado del sumatorio de las diferencias entre las tablas de distancias, es decir, la suma de todas las distancias que superan el margen de error de 5 cm. Se utiliza como penalización de la solución obtenida.
- $\boldsymbol{\beta}$: Es la distancia máxima de separación que puede haber entre los robots. Se utiliza para normalizar el resultado y que la penalización esté entre los valores 0 y 1.
- $\mathbf{f(x)}$: Es la función objetivo. Es el valor total obtenido después de las penalizaciones. Además, se resta un valor que simula el gasto de energía de cada paso de ejecución. La finalidad de restar ese gasto de energía es conseguir aumentar la velocidad del proceso de aprendizaje.

Con esta función objetivo se pretende generar valores más altos en las soluciones donde los robots permanecen con las mismas distancias que los separan en el instante inicial, en cada paso de ejecución del problema.

## 6.3 Experimentación

En esta sección se va a mostrar los resultados obtenidos con la conducta aprendida. El entrenamiento se ha realizado en un escenario con 4 robots formando una cadena y separados por 20 centímetros, donde el robot líder es el robot colocado en la zona superior del enjambre y el escenario es una superficie cuadrada de 12x8 metros sin obstáculos (ver Fig. 6.4). La distancia máxima de lectura del sensor láser durante el proceso de entrenamiento es de 50 centímetros. Para poder evaluar los resultados de la conducta se han realizado diferentes pruebas donde el robot líder contiene movimientos aleatorios, se agrega ruido gaussiano a las lecturas del sensor láser, se prueban diferentes longitudes del sensor láser y se prueban diferentes rutas que dibujan un Zig-Zag o un círculo.

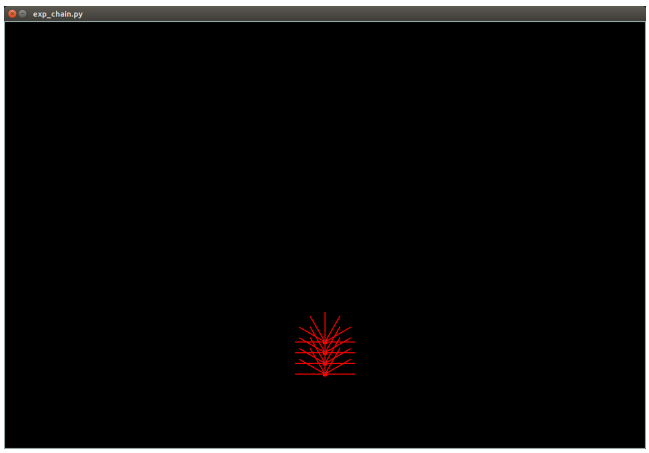



**Figura 6.4:** Instante inicial del problema. En el escenario se encuentran 4 robots del enjambre formando una cadena, donde el robot líder está situado al comienzo de dicha cadena. El escenario es una superficie cuadrada sin obstáculos.

En primer lugar, se han ejecutado las conductas aprendidas en 100 episodios donde la ruta del líder es aleatoria. En los resultados se muestran la distancia de error de separación de los robots relativa a su distancia total recorrida. Además, en otra gráfica se muestra cuántos robots se han separado de la cadena al finalizar la ejecución de cada episodio. Los resultados se pueden observar en las figuras 6.5 y 6.6, que muestran la distancia de error y el número de robots que se han separado de la cadena, teniendo en cuenta que la cadena está formada por 4 robots en total. Respecto a la conducta obtenida con la velocidad lineal y rotacional, en la primera imagen se puede observar que la distancia de error adquiere valores reducidos, por lo que se puede considerar que la conducta se ha aprendido con éxito. Además, el 90% de los episodios terminan con la cadena completa, es decir, sin ningún robot separado. Uno de los problemas que se ha observado en la conducta aprendida es que el robot que se separa de la cadena, no vuelve a unirse a lo largo del episodio. Otro problema es el hecho de que los robots se separen cuando la trayectoria del robot líder incluye giros bruscos, lo que dificulta que el sensor láser detecte al robot delantero y hace que los robots cadena se separen. Sin embargo, respecto a la conducta con acciones clasificados podemos observar que presenta peores resultados que la anterior, ya que, presenta mayor distancia de separación y mayor número de robots separados de la cadena. Por lo tanto, podemos concluir que la conducta con movimientos controlados con la velocidad rotacional y lineal proporciona una mayor flexibilidad y menor separación de la cadena a la hora de seguir nuevas trayectorias.

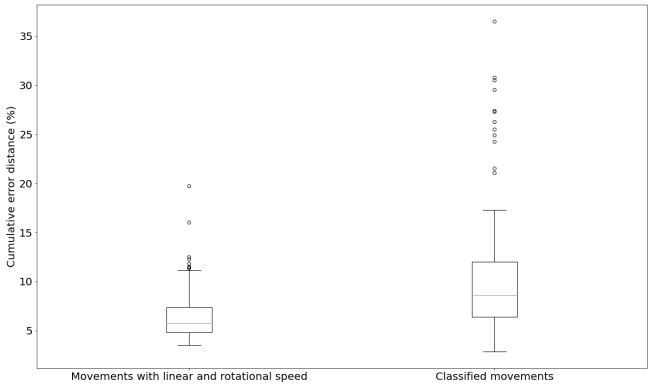



**Figura 6.5:** Media de distancia de error de separación relativa a la distancia total recorrida de los robots. Los resultados se muestran después de ejecutar la conducta en 100 episodios y con rutas aleatorias del robot líder.

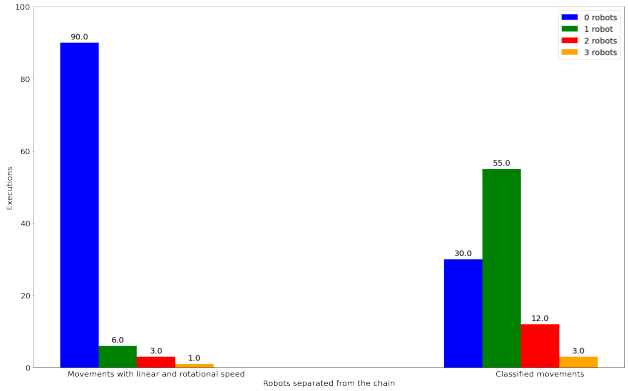



**Figura 6.6:** Número de episodios en los que los agentes han terminado con la cadena con un número de robots separados de ella. Como se puede observar en los resultados obtenidos, en un 90% de las ejecuciones, los robots han conseguido mantener la formación en cadena al finalizar el episodio.

En segundo lugar, centrándonos en la conducta con movimientos controlados por la velocidad lineal y rotacional, se ha agregado ruido gaussiano a la lectura de los láseres con la finalidad de comprobar su robustez frente al ruido. En los entornos reales es probable que la lectura de los sensores presente ruido, por lo que es muy útil comprobar la robustez del algoritmo. El ruido agregado presenta una distribución gaussiana con una media ($\mu$) de valor 0 y un valor de desviación típica ($\sigma$) variable entre 0, 0.1, 0.25 y 0.5. Como se puede observar en la figura 6.7 al aumentar el valor del ruido no afecta negativamente a la conducta aprendida, ya que, se presentan distancias de error similares.

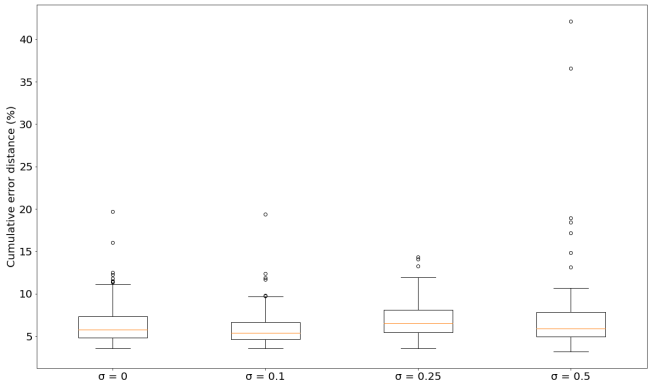



**Figura 6.7:** Media de distancia de error de separación relativa a la distancia total recorrida de los robots. En esta gráfica se muestra una comparación entre los valores obtenidos con distintos valores de $\sigma$ de la distribución gaussiana. Donde se puede observar que el aumento del ruido gaussiano no perjudica a la conducta aprendida porque los resultados son levemente superiores al aumentar el nivel de ruido.

En tercer lugar, se ha probado la conducta entrenada cambiando la distancia de cobertura del sensor láser. Se ha modificado la distancia máxima a 30, 50 y 100 centímetros. En la gráfica de la figura 6.8 se puede observar que cuando la longitud del láser es menor a la utilizada durante el entrenamiento, los robots cadena se separan más entre ellos, ya que, se pierden de la cadena debido al poco margen de cobertura que hay entre la separación con el robot delantero y la distancia máxima del láser (ver Fig. 6.9). Aunque, la conducta ofrece mejores resultados con una distancia mayor de la utilizada durante el proceso de aprendizaje, ya que, hay menos robots que se separan de la cadena y hay menor distancia de error durante la ejecución de los episodios.

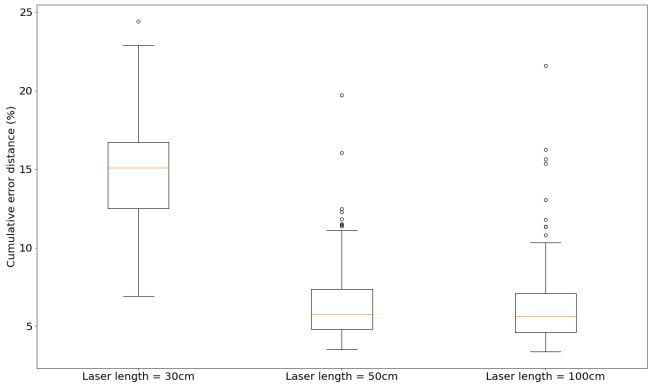



**Figura 6.8:** Media de distancia de error de separación relativa a la distancia total recorrida de los robots. En esta gráfica se muestra una comparación de resultados obtenidos con diferentes longitudes del sensor láser. Como se puede observar en la gráfica, a menor longitud del láser hay mayor distancia de error en la cadena formada. A diferencia de los resultados obtenidos con un láser de longitud de 100 cm, que obtiene una distancia de error ligeramente inferior a la obtenida con el láser de 50 centímetros de longitud.

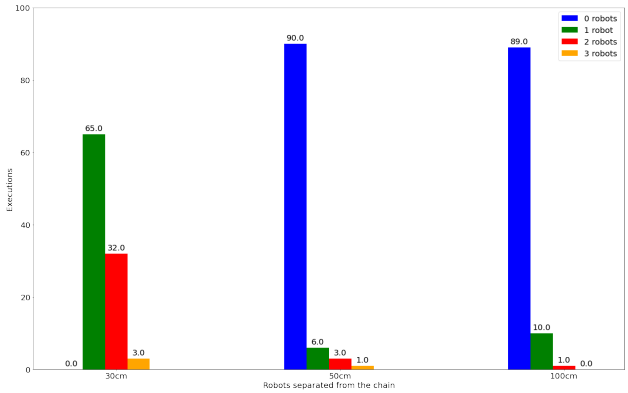



**Figura 6.9:** Número de episodios en los que los agentes han terminado con la cadena con un número de robots separados de ella. En esta gráfica podemos observar que la longitud del láser que ofrece menor separación de la cadena la del láser de 100 centímetros, ya que, tiene mayor cobertura de distancia y pueden volver con más facilidad a la cadena. Mientras que, con el láser de longitud de 30 centímetros es más fácil que se separen de la cadena porque tienen menor margen de cobertura.

Por otra parte, se ha evaluado la escalabilidad de la conducta aprendida con distinto número de robots cadena. En las gráficas 6.10 y 6.11 se pueden ver los resultados obtenidos en enjambres con 2, 4 y 6 robots, donde siempre uno de ellos actúa como robot líder. En ambas gráficas se puede observar que la conducta aprendida en un enjambre de 4 robots es flexible para enjambres de menor y mayor tamaño. Aunque, como se puede ver el la gráfica

6.11, la conducta presenta una separación de la cadena mayor en enjambres más grandes. Por lo tanto, la conducta se ve perjudicada cuando se aumenta el número de robots del enjambre dado durante el proceso de entrenamiento, ya que, hay más facilidad de que los robots cadena se separen de la formación.

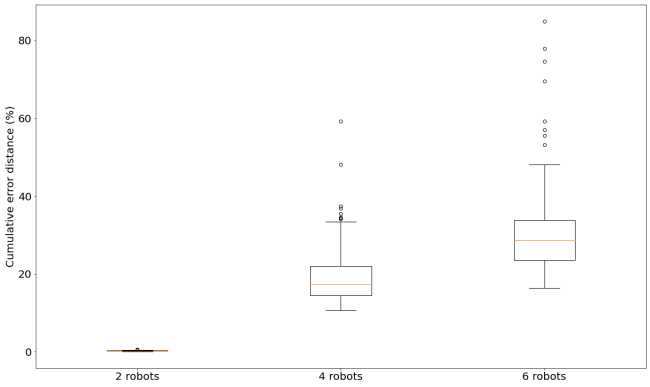



**Figura 6.10:** Distancia de error total de separación relativa a la distancia total recorrida de los robots. En esta gráfica se puede apreciar que cuando el número de robots del enjambre es menor, la distancia de error muy reducida. Mientras que, cuando el tamaño del enjambre es mayor, hay más robots que se separan de la distancia inicial dada.

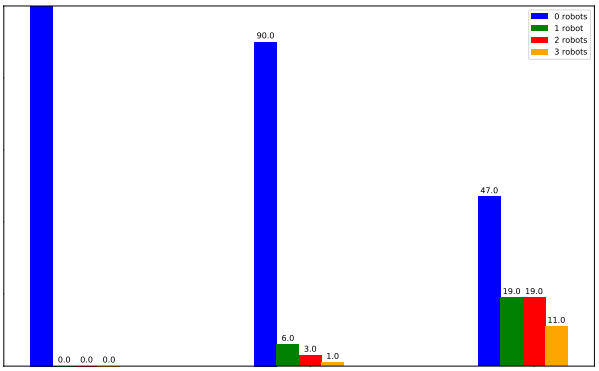
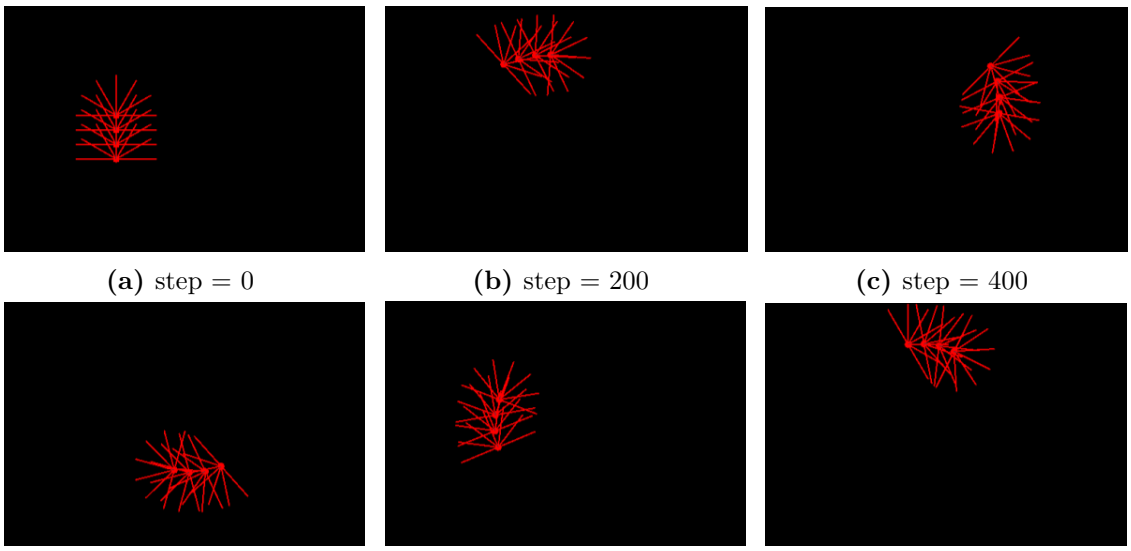



**Figura 6.11:** Número de episodios en los que los agentes han terminado con la cadena con un número de robots separados de ella. En esta gráfica podemos observar que los agentes de un enjambre de 6 robots se separan con mayor facilidad de la cadena que los pertenecientes a un enjambre de menor tamaño.

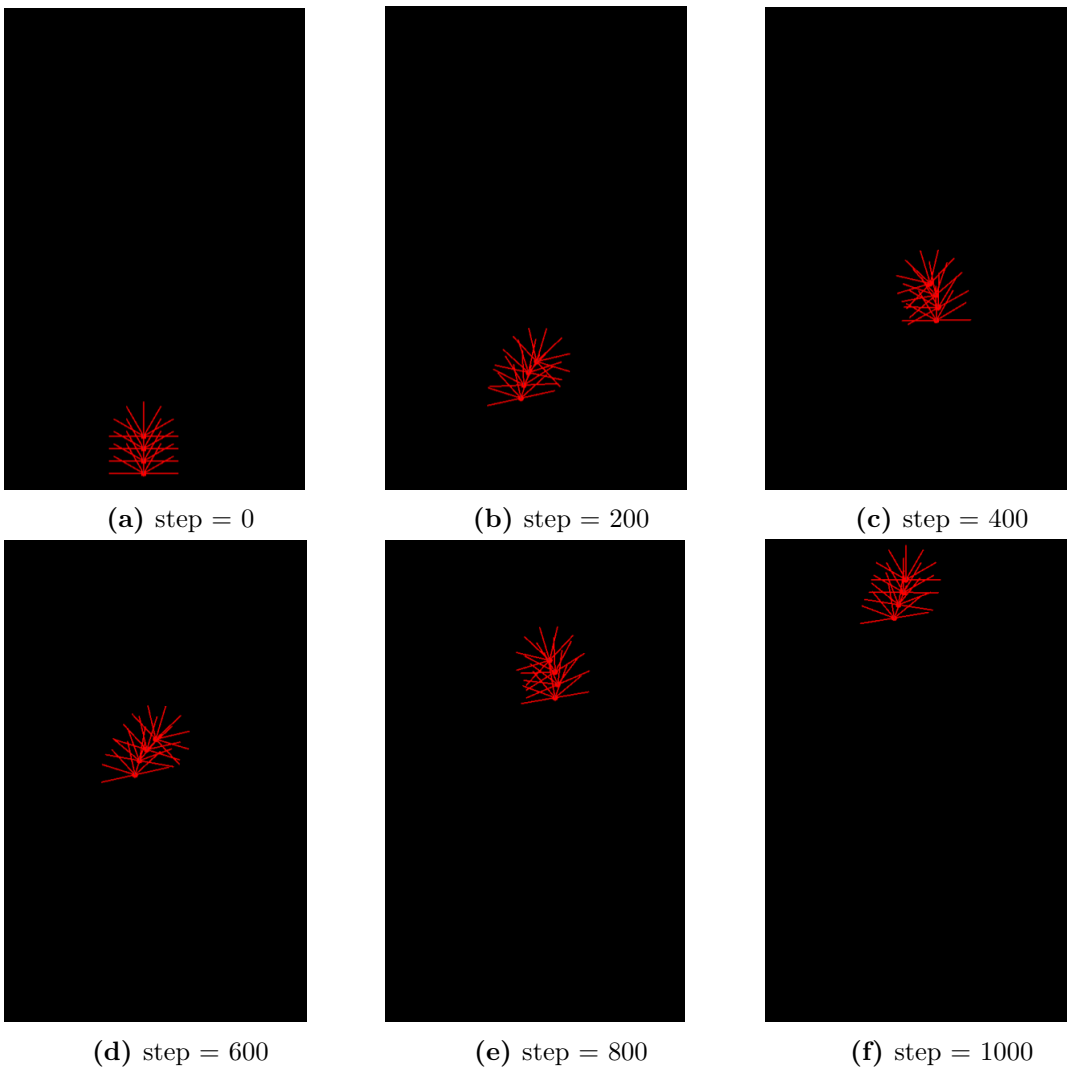
Por último, se ha evaluado la flexibilidad de la conducta obtenida en distintas rutas, como en una trayectoria circular o en una trayectoria de Zig-Zag. Donde en las figuras 6.12 y 6.13 se puede observar que la forma de cadena se mantiene en todo momento, hasta en las curvas más pronunciadas de las trayectorias.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 6.12:** Secuencia de una ejecución de la conducta aprendida donde los robots del enjambre siguen una trayectoria circular. En esta secuencia se puede observar como los robots mantienen la cadena hasta en las curvas de la trayectoria.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 6.13:** Secuencia de una ejecución de la conducta aprendida donde los robots del enjambre siguen una trayectoria en Zig-Zag. En esta secuencia se puede observar como los robots mantienen la cadena hasta en las diferentes curvas de la trayectoria.

## 6.4 Formación en cadena en un entorno real

El objetivo de esta sección es conseguir ejecutar una conducta de formación en cadena, que se ha obtenido previamente mediante aprendizaje por refuerzo, en un entorno real. Para la ejecución de la conducta en un entorno real se utilizará el robot Mbot Ranger que se conectará a una placa Raspberry Pi y a una placa Auriga. La placa Raspberry se comunicará con la placa Auriga y con el sensor láser para ejecutar la conducta aprendida mientras que la placa Auriga se comunicará con los motores de las ruedas para ejecutar los comandos de velocidad obtenidos de la conducta aprendida.

### 6.4.1 Especificaciones previas

Para el desarrollo de esta sección se han utilizado dos robots MBot Ranger, un robot cadena con la formación de la figura 6.14 y un robot líder con la formación de la figura 6.15. Respecto al robot cadena, el sensor láser RPLidar se encuentra en la zona superior del robot, este sensor está conectado a la placa base Raspberry Pi que se encuentra en la zona intermedia, que además, esta placa está conectada a la placa Auriga para enviar los comandos de velocidad a las ruedas del robot. Mientras que el robot líder solo necesita la placa base Auriga para ejecutar los movimientos de la ruta predefinida. Además, la placa Auriga del robot líder está a la misma altura que el sensor láser del robot cadena para que el primero sea detectable por el sensor del segundo robot.

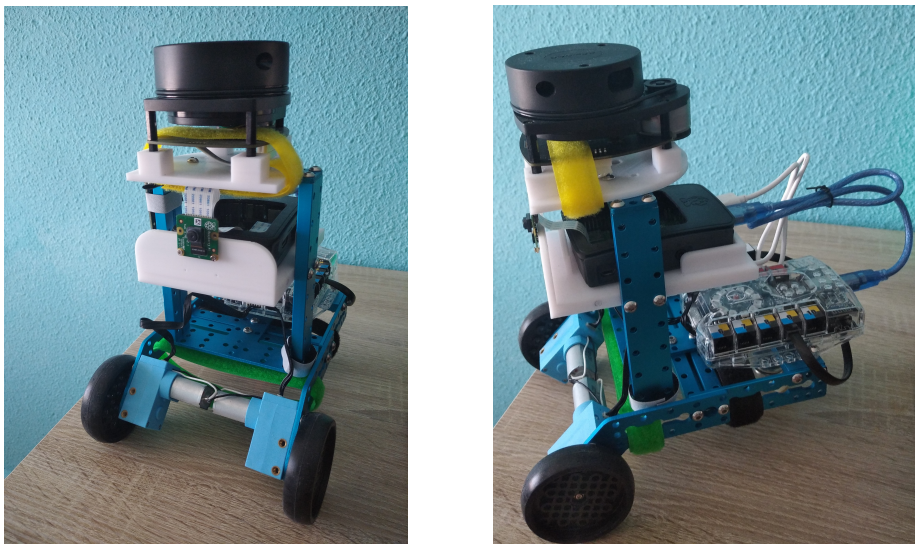


**Figura 6.14:** Formación del robot mBot Ranger para la realizar la tarea de formación en cadena. El sensor láser RPLidar se encuentra en la parte superior del robot. Este robot actúa como robot cadena que sigue los movimientos del robot líder.

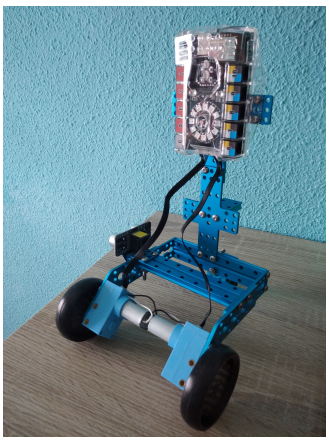
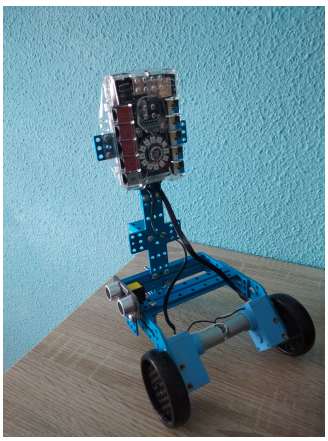



**Figura 6.15:** Formación del robot mBot Ranger que actúa como robot líder para la realizar la tarea de formación en cadena. No contiene sensor láser porque solo necesita la placa auriga para ejecutar los movimientos de la trayectoria predefinida.

Por otra parte, respecto a la conducta aprendida, en el comportamiento aprendido la red neuronal obtiene la velocidad lineal y rotacional del robot según las distancias leídas del sensor láser. Sin embargo, la librería AurigaPy utiliza las velocidades de cada rueda de forma individual o a través de comandos simples como el avance, giro a la derecha, giro a la izquierda y parar. Una solución para convertir las velocidades rotacionales y lineales obtenidas por la red neuronal es calcular las velocidades de cada rueda a través de las ecuaciones 6.6 y 6.7, donde actúan las siguientes variables:

- $L$: Es la distancia que hay entre las ruedas.
- $V_t$: Velocidad translacional o lineal.
- $V_r$: Velocidad rotacional.
- $V_{right}$: Velocidad de la rueda derecha.
- $V_{left}$: Velocidad de la rueda izquierda.

$$V_{left} = V_t + L \cdot \frac{V_r}{2} \tag{6.6}$$

$$V_{right} = V_t - L \cdot \frac{V_r}{2} \tag{6.7}$$

Sin embargo, a lo largo del desarrollo de este capítulo se utilizará la conducta obtenida con clasificación de movimientos para poder observar con más facilidad los movimientos resultantes de la red neuronal.

### 6.4.2 Configuración de la tarea con AurigaPy

En esta sección se explicará cómo desarrollar la tarea de formación en cadena con Keras, AurigaPy y la librería que recoge los datos del sensor láser, RPLidar-roboticia, utilizando como sistema central la placa Raspberry que controla los movimientos del robot y obtiene las lecturas del láser. La librería RPLidar cada segundo actualiza y muestra las lecturas obtenidas, aunque, en algunos casos, no se obtienen las lecturas de los 360º y se obtienen menos.

Antes de empezar a implementar la tarea se debe encontrar el rango del láser que corresponde con los 180º de cobertura delantera del robot. Para encontrar este rango se muestran las lecturas obtenidas del láser en 360º. Gracias a la gráfica de la figura 6.16 se observó que las lecturas de la zona delantera correspondían con el rango de [90º, 270º] con el rayo central situado en el grado 0º.

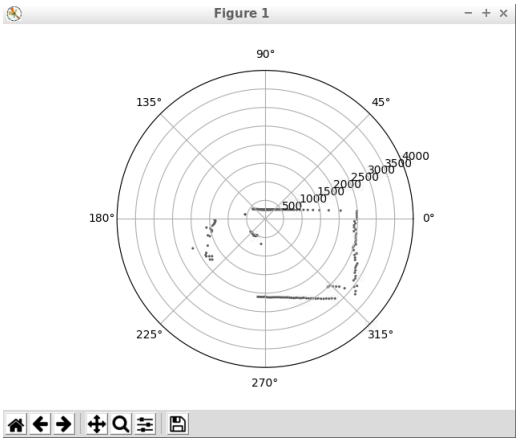



**Figura 6.16:** Lecturas del sensor láser en 360º obtenidas a través de la libreria RPLidar-roboticia

Cuando ya se han obtenido las lecturas correspondientes a la zona delantera del robot, se ejecuta el algoritmo con los siguientes pasos del diagrama de la figura 6.17. Este algoritmo se ejecuta completamente en la placa base Raspberry, ya que, la misma placa se comunica con la placa Auriga a través de la librería AurigaPy y con el sensor láser a través de la librería RPLidar.

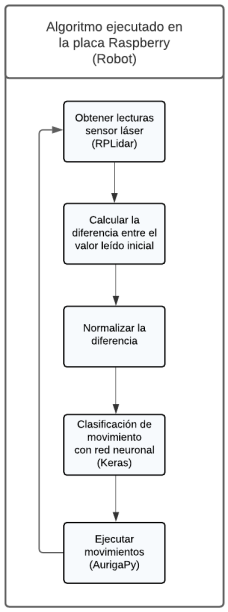



**Figura 6.17:** Diagrama de flujo del algoritmo ejecutado en la placa Raspberry.

Las desventajas de utilizar el algoritmo en la placa Raspberry con la librería de Roboticia, RPLidar, es que presenta un retardo en la lectura del láser que dificulta la ejecución del algoritmo en tiempo real. Por este motivo se resolverá la tarea en un entorno real utilizando ROS, como se explica en la siguiente sección.

### 6.4.3 Configuración de la tarea con ROS y AurigaPy

En esta sección se explicará la implementación de la tarea utilizando el framework ROS, para la lectura del sensor láser, y la librería AurigaPy para realizar los movimientos obtenidos con la red neuronal.

En primer lugar, ROS es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo, es decir, está organizado por una serie de nodos con distintas funciones. La ventaja de utilizar ROS es que obtiene los 360º del sensor láser sin problemas, lo que permite una ejecución en tiempo real más fluida. La versión instalada en la placa Raspberry es Ros Kinetic y la versión instalada de Python es la versión 2.7. El problema de estas versiones es que son incompatibles con la versión de Keras (2.3.1) utilizada para el entrenamiento. Por lo tanto, para poder utilizar Keras y ROS se utilizarán las librerías Roslibpy y Rosbridge. Roslibpy es una librería que permite recibir y publicar mensajes a través de los nodos de ROS en un equipo en remoto sin tener instalado ROS. Para realizar esta tarea se conecta a la dirección IP del equipo que tiene instalado ROS, en este caso la conexión se hace a la placa Raspberry, y se subscribe a los nodos necesarios. Mientras tanto, la librería Rosbridge sirve para publicar los mensajes de los nodos de ROS mediante websockets a los equipos clientes que se hayan suscrito. Por

lo tanto, en el robot se lanza Rosbridge para enviar las lecturas obtenidas del sensor láser, mientras que un equipo en remoto se encarga de recibir esas lecturas, tratarlas y enviarlas a la red neuronal de Keras y conectarse a la placa Auriga para ejecutar los comandos de velocidad obtenidos.

En segundo lugar, los nodos que se han utilizado para desarrollar la tarea de formación en cadena son los que aparecen en la figura 6.18. El nodo **rosbridge_websocket** es el encargado de comunicarse con el equipo local y enviar las lecturas del sensor láser. Por otro lado, el nodo **scan** es el que se encarga de capturar las lecturas del láser RPLidar.

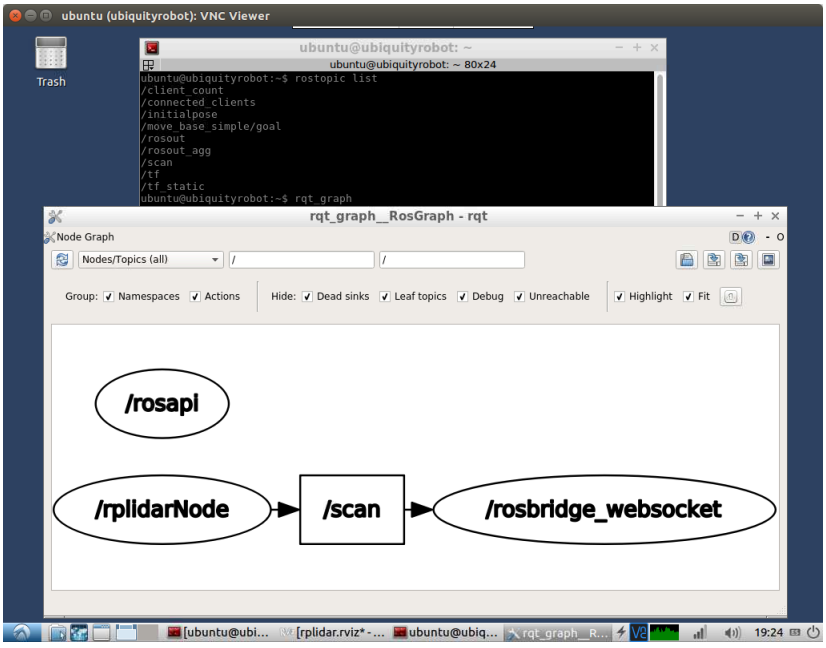



**Figura 6.18:** Nodos que se han utilizado para la lectura del láser y el envío mediante websockets de las lecturas.

En tercer lugar, las lecturas del láser que se capturan a través del topic **scan** se pueden ver con el visualizador RVIZ como aparece en la figura 6.19. Las lecturas que se utilizan para resolver la tarea son las correspondientes con los siguientes grados: 90º, 60º, 30º, 0º, 330º, 300º y 270º. Estas lecturas son las 7 lecturas que corresponden con la parte delantera del robot.

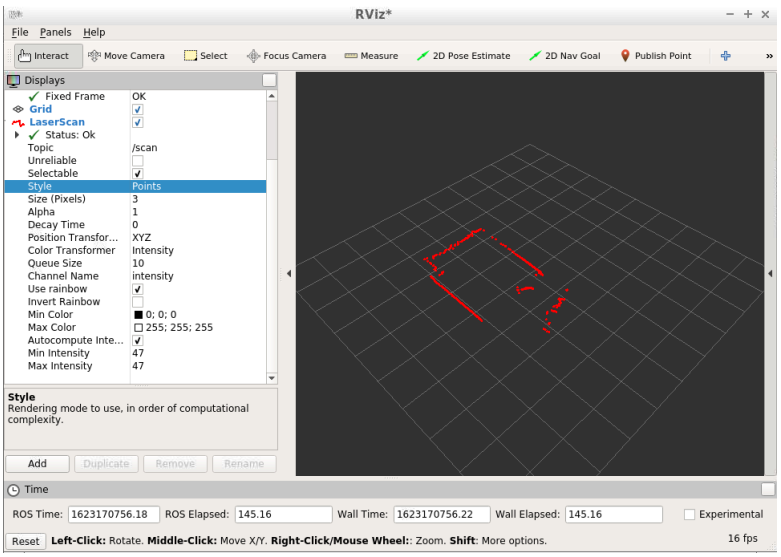



**Figura 6.19:** Lecturas del láser obtenidas por el nodo Scan mostradas con RVIZ.

Por último, respecto a la ejecución de la tarea en un entorno real disponemos de dos programas diferentes que se ejecutan en paralelo y de forma independiente. Por un lado, está el algoritmo del robot líder, que actúa de forma independiente del robot cadena y ejecuta los siguientes movimientos:

1. Avanzar hacia delante
2. Girar hacia la izquierda menos de 90 grados.
3. Avanzar hacia delante
4. Parar

Por otro lado, está el algoritmo del robot cadena que se ejecuta de forma independiente. Aunque, este algoritmo sí que tiene en cuenta la posición inicial del robot líder, ya que, forma parte de la lectura inicial. Este algoritmo sigue los pasos del diagrama de flujo de la figura 6.20. En esta figura se puede observar que ambos equipos, tanto el equipo local como el robot se comunican a través de las librerías Roslibpy y Rosbridge para recibir y enviar las lecturas del sensor láser. Además, en este diagrama se muestra que quién se comunica con el robot para ejecutar las acciones es el equipo local a través de la librería AurigaPy.

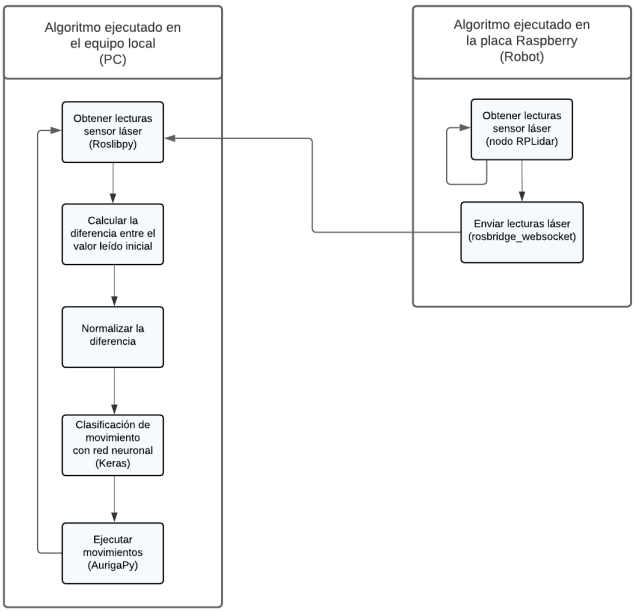
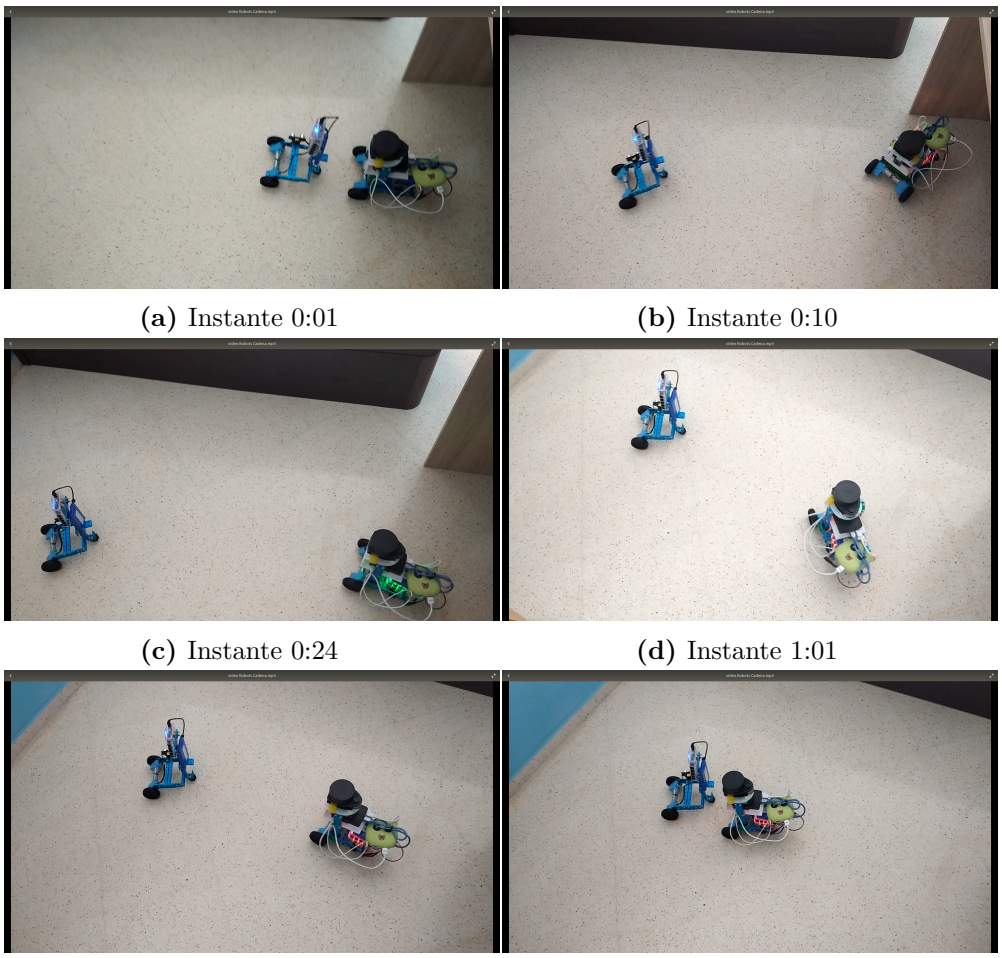



**Figura 6.20:** Diagrama de flujo del algoritmo ejecutado en la placa Raspberry y en el equipo local (PC).

Finalmente, los movimientos se envían a la placa auriga con la librería AurigaPy en el equipo local, que es el que se ha conectado previamente al robot. El entorno es una superficie rectangular con medidas de 1.90x1.60 metros. En la figura 6.21 se muestra una ejecución en un entorno real de la política obtenida en la conducta de formación en cadena. En esta secuencia de imágenes se puede observar como el robot gira hacia la derecha o izquierda hasta encontrar al robot líder.

Como resultado de estos algoritmos, obtenemos que el robot líder ejecuta una sencilla ruta, pero, el robot cadena tarda más tiempo en encontrarlo y seguirlo en forma de cadena. Este suceso se debe a que el robot líder es débilmente visible por el sensor láser del robot cadena, ya que la parte más visible del robot líder es la placa base auriga cuyas dimensiones son 6.5 centímetros de ancho y 2.5 centímetros de grosor. Por lo tanto, el robot líder presenta más dificultades a la hora de conseguir obtener las lecturas para el sensor láser.

Otro comportamiento del robot cadena es que cuando ha perdido las lecturas del láser inicial, gira hacia la derecha o izquierda completamente hasta encontrar la lectura del láser que corresponde con la posición del robot líder. Sin embargo, al final de la ejecución de la tarea, el robot cadena se dirige hacia el robot líder cuando ya lo ha detectado el sensor láser y se para cuando se encuentra a una distancia similar a la del inicio del programa.



**(a)** Instante 0:01 **(b)** Instante 0:10

**(c)** Instante 0:24 **(d)** Instante 1:01

**(e)** Instante 1:51 **(f)** Instante 2:06

**Figura 6.21:** Secuencia de imágenes de la ejecución de la tarea de formación en cadena en un entorno real.

## 6.5 Discusión

Para terminar, después del desarrollo de este capítulo podemos concluir que la conducta de formación en cadena se ha aprendido con éxito en trayectorias diseñadas con movimientos al azar del robot líder. Además, se ha obtenido una conducta robusta al ruido presente en los sensores y flexible a diferentes trayectorias del robot líder. Sin embargo, no podemos confirmar que sea una conducta escalable con un mayor número de robots cadena respecto a los proporcionados en el proceso de entrenamiento porque se presenta una mayor separación de la cadena que en otros enjambres de menor tamaño.

Por otra parte, una conclusión que se puede extraer, respecto a la prueba en un entorno real, es que el algoritmo presenta algunas dificultades cuando el robot líder es débilmente visible para el sensor láser del robot cadena. Este problema se puede solucionar creando una

estructura más visible del robot líder para facilitar la lectura de las distancias con el sensor láser. Además, en la ejecución en el entorno real mostrada anteriormente, el robot cadena tarda en seguir los pasos del robot líder porque no lo detecta con el sensor láser. Aunque, al final de la ejecución ya encuentra al robot líder y se dirige hacia él hasta que se posiciona a una distancia similar a la del inicio del programa.

Como líneas futuras, se podrían ampliar las pruebas realizadas en el entorno real, mejorando la estructura del robot líder para que se más visible para el sensor láser del robot cadena. Además, se podría comprobar la escalabilidad del algoritmo en un entorno real con más robots cadena.

# 7 Aprendizaje de una conducta de mantenimiento de una forma

En este capítulo se va a explicar cómo ha sido el proceso de aprendizaje que se ha realizado para que un enjambre de robots consiga mantener una forma inicial dada con sus posiciones iniciales mientras siguen los movimientos del robot líder y esquivan los obstáculos del entorno. El objetivo principal de este capítulo es encontrar una política óptima aplicando estrategias evolutivas para el aprendizaje por refuerzo. La finalidad de utilizar estrategias evolutivas es encontrar una política óptima que alcance un máximo global realizando un proceso automático de aprendizaje. Las estrategias que se van a utilizar son algoritmos genéticos simples, GA, y CMA-ES con el fin de comparar qué estrategia ofrece mejores resultados. Además, se va a estudiar la tarea de transporte colectivo de objetos obtenida a partir de una tarea de formación en un entorno simple.

Por último, los resultados de este capítulo sobre la conducta obtenida de mantenimiento de una forma y evitación de obstáculos han sido aceptados para su publicación en el congreso **XIX Conference of the Spanish Association for Artificial Intelligence (CAEPIA)** con el título *"Learning A Swarm Formation Policy using Deep Reinforcement Learning"*.

## 7.1 Introducción

Para comenzar, como se ha comentado anteriormente, la tarea se basa en que un enjambre se desplace por un entorno siguiendo los movimientos de un robot independiente manteniendo una forma dada sus posiciones iniciales. Para ello, se dispone de un robot independiente, cuyos movimientos se han definido antes del entrenamiento, y de un enjambre de robots que será el encargado de desplazarse por el entorno manteniendo la forma inicial. Para resolver este problema los robots contarán con la información de un sensor láser para detectar la distancia a la que se encuentran de los obstáculos. Además, para poder obtener información acerca de la posición y orientación relativa que tienen respecto al líder cuentan con un sensor de posicionamiento relativo en las coordenadas X e Y más la orientación.

Antes de comenzar el proceso de aprendizaje de esta tarea se debe definir la función de recompensa y se deben establecer cuales serán las entradas y salidas de la red neuronal. Los robots del enjambre seguirán las acciones determinadas por la red neuronal mientras que el robot independiente seguirá las acciones que se han determinado antes del entrenamiento. Se ha realizado el entrenamiento con diferentes entornos donde las posiciones de los obstáculos y el movimiento del líder son aleatorios. Aunque, las posiciones iniciales de los robots del enjambre son siempre fijas.

Esta tarea se basa en comportamientos de otras como la formación en cadena, el movimiento coordinado de un enjambre o la planificación de trayectorias. Debido a la dificultad que supone encontrar una política óptima que resuelva esta tarea mediante un método tradicional de robótica de enjambre, se utilizan estrategias evolutivas para el aprendizaje por refuerzo. De esta forma, se puede buscar una política óptima que resuelva el problema y que encuentre un máximo global mediante un proceso automático. A partir de los resultados obtenidos en el CAPÍTULO 5 se ha elegido proponer la estrategia CMA-ES para el proceso de entrenamiento, ya que, la tarea de formación con obstáculos en el entorno se puede considerar una conducta compleja. Además, en las propuestas de Sperati y cols. (2011) y Lin y cols. (2013), para las tareas de formación en cadena y planificación de trayectorias, se han utilizado soluciones basadas en algoritmos genéticos, por lo tanto, en este capítulo también se estudiará la eficacia de una solución basada en algoritmos genéticos en un entorno con obstáculos.

## 7.2 Diseño del problema

En la tarea principal de este artículo, un enjambre debe mantener la forma inicial dada en todo momento mientras siguen al robot independiente y esquivan los obstáculos del entorno. Este entorno es desconocido para los agentes porque no saben donde se encuentran los obstáculos ni qué recorrido hará el robot líder. Las acciones posibles que podrá ejecutar el robot líder son: ir hacia delante, ir hacia la derecha, ir hacia la izquierda y parar. En cada paso de ejecución, cada robot observará su estado $s_t \in S$ a través de un sensor láser de $180^{\text{o}}$ de cobertura y 7 rayos con una separación de $30^{\text{o}}$, como el utilizado en el CAPÍTULO 6, y de un sensor de posicionamiento relativo al robot líder que indica la posición X e Y más la orientación en radianes a la cual están situados. A través de una función de recompensa, que se explicará más adelante, cada robot deberá elegir qué acción ejecutar $a_t \in A$, que en este problema será a través de la velocidad lineal y rotacional que ejecutarán los motores. Los valores de velocidad serán proporcionados por la red neuronal entrenada. De esta forma, se puede considerar esta tarea como un Proceso de decisión de Markov (MDP), donde el espacio de observación, $S$, es el estado en el que se encuentra el entorno, es decir, la localización de los obstáculos o la posición del robot líder, y el espacio de acciones, $A$, es el conjunto de acciones que pueden desempeñar los robots del enjambre para esquivar los obstáculos y seguir los pasos del robot líder, en este caso, la velocidad rotacional y lineal.

Sin embargo, se ha realizado un entrenamiento previo en un entorno sin obstáculos con el fin de obtener una conducta que se centre en el mantenimiento de la forma del enjambre. Por lo tanto, en ese problema, el conjunto de $S$ solo se tendrá en cuenta la información obtenida del sensor de posicionamiento.

Por una parte, para mapear la información del entorno sin obstáculos se ha utilizado una red neuronal (ver Fig. 7.1) con 3 entradas, 2 capas ocultas y 2 neuronas de salida, que corresponden con la velocidad lineal y rotacional. Además, para el entrenamiento de esta red neuronal se ha utilizado la estrategia evolutiva CMA con aprendizaje por refuerzo, excepto en el entorno con obstáculos donde también se utiliza la estrategia basada en algoritmos genéticos simples, GA.

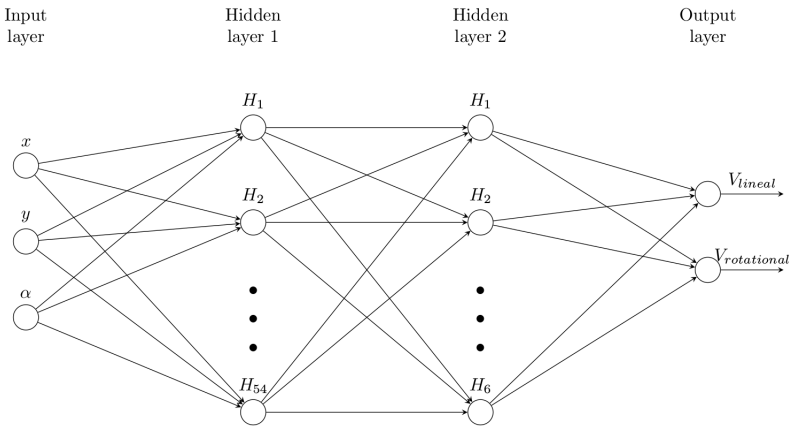



**Figura 7.1:** Red neuronal utilizada durante el entrenamiento. Contiene 3 neuronas de entrada. Las entradas x e y son la diferencia entre la distancia inicial y actual del robot del enjambre respecto al robot pivote en las coordenadas X e Y. La entrada $\alpha$ es la diferencia angular de las orientaciones inicial y final del robot del enjambre respecto al robot pivote. Además, contiene 2 capas ocultas de 54 y 6 neuronas respectivamente y su función de activación es tanh. Por último, se encuentra la capa de salida que contiene la velocidad lineal y rotacional que deberá ejecutar el robot según la información proporcionada.

Por otra parte, para mapear la información del entorno con obstáculos se ha utilizado una red neuronal (ver Fig. 7.2) con 10 entradas, 2 capas ocultas y 2 neuronas de salida, que corresponden a la velocidad lineal y rotacional.

Los agentes tomarán las decisiones que indiquen las redes neuronales en cada paso de ejecución. Con el objetivo de encontrar una política óptima se utiliza una función de recompensa que obtiene un valor sobre la acción ejecutada en cada estado. La función de recompensa diseñada para este problema se explica en la siguiente sección.

### 7.2.1 Función de recompensa

La función de recompensa de este capítulo se basa en la diseñada en el capítulo anterior. Aunque, para obtener una conducta donde los robots esquivan los obstáculos del entorno se modifica la ecuación 6.5 y se utiliza la ecuación 7.1 con el objetivo de penalizar las colisiones con los objetos del entorno. Además, en este capítulo, los robots forman un grafo distinto con sus posiciones iniciales (Ver Fig. 7.3). Para las tareas que tienen solo el sensor de posicionamiento, es decir, en las que no se esquivan obstáculos, se utilizará la función de recompensa 6.5. Mientras que, para las tareas que requieren evitar obstáculos mientras siguen al robot líder se utilizará la función de recompensa modificada 7.1.

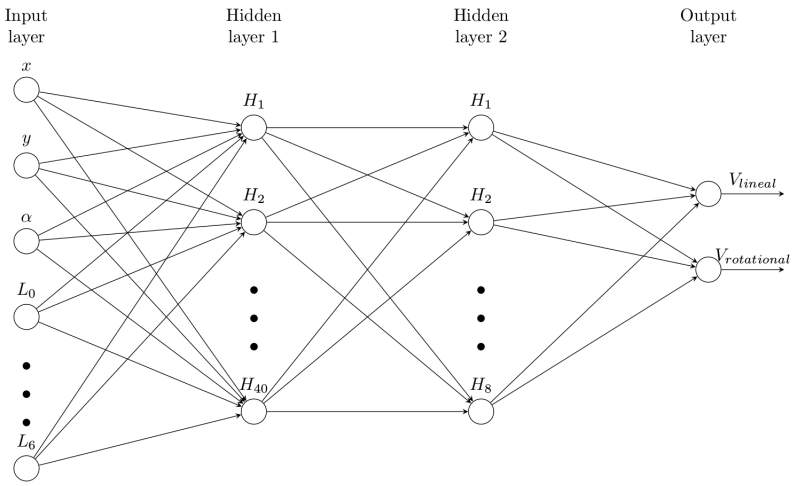



**Figura 7.2:** Red neuronal utilizada durante el entrenamiento. Contiene 10 neuronas de entrada. Las entradas x e y son la diferencia entre la distancia inicial y actual del robot del enjambre respecto al robot pivote en las coordenadas X e Y. La entrada $\alpha$ es la diferencia angular de las orientaciones inicial y final del robot del enjambre respecto al robot pivote. Las entradas $L_0$ a $L_6$ son los valores obtenidos de los rayos del sensor láser. Los valores de entrada están normalizados. Además, contiene 2 capas ocultas de 40 y 8 neuronas respectivamente y su función de activación es tanh. Por último, se encuentra la capa de salida que contiene la velocidad lineal y rotacional que deberá ejecutar el robot según la información proporcionada.

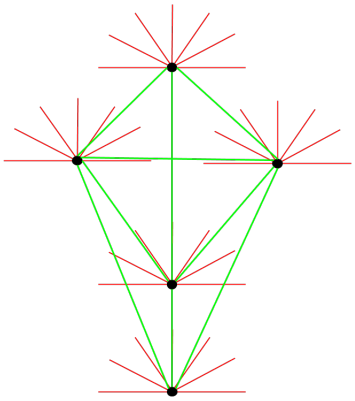


**Figura 7.3:** Grafo no dirigido formado por el enjambre. Los nodos son los robots y las aristas, representadas en color verde, son las distancias que hay entre ellos.

$$f(x) = \left(1 - \frac{\gamma}{\beta}\right) - \alpha \tag{7.1}$$

Donde $\alpha$: Es el número de colisiones de los robots con los obstáculos.

El objetivo de esta función de recompensa es que premie más las soluciones que ofrezcan menos diferencia de distancias leídas con el sensor de posicionamiento respecto a las leídas en el instante inicial de la ejecución. Por lo tanto, a mayor diferencia de distancia de la posición original, habrá menor recompensa. La recompensa se obtiene en cada paso de ejecución del programa. Además, con la penalización añadida se pretende dar más importancia a que no colisionen los robots, pero sin anular la recompensa obtenida de mantener sus formaciones.

## 7.3 Experimentación

En esta sección se va a describir qué pruebas se han realizado para resolver la tarea de una manera óptima.

En primer lugar, al inicio de la experimentación, se planteó la posibilidad de utilizar solo el sensor láser para determinar las posiciones de los robots del enjambre. El problema de esta propuesta es que durante el entrenamiento solo se conseguía obtener una conducta donde los robots mantenían la forma del enjambre en 3 posibles movimientos: Avanzar, retroceder o parar. Los giros no se aprendían de manera correcta por lo que se probó agregar la orientación relativa del pivote a la entrada de la red neuronal más las distancias de los láseres leídas. Al realizar este experimento no se obtuvo la conducta deseada porque los robots del enjambre seguían sin seguir al robot líder en los giros, por lo tanto, no conseguían mantener la forma inicial. Además, también se realizaron experimentos discretizando las distancias de los láseres y calculando los movimientos como las siguientes acciones: Avanzar, ir hacia la derecha, ir hacia la izquierda, retroceder y parar. El problema de discretizar los valores entrada y de salida es la pérdida de información.

Por lo tanto, después de obtener los resultados desfavorables en las pruebas anteriores, se decidió añadir un sensor de posicionamiento relativo al robot líder. Además, las distancias de los láseres y las del sensor de posicionamiento relativo se proporcionan a la red neuronal normalizadas. Las acciones de los agentes se obtendrán calculando la velocidad rotacional y lineal que proporcionará la red neuronal. De esta forma, se evita perder información del entorno y se obtienen más variedad de acciones.

Por otra parte, al utilizar la combinación del sensor láser más la del sensor de posicionamiento se puede agregar la tarea de esquivar obstáculos. De esta forma, se utiliza el sensor láser para detectar los obstáculos y el sensor de posicionamiento para poder localizar al robot líder. Además, la ventaja de utilizar esta combinación de sensores es que se puede obtener una conducta flexible al número de robots del enjambre, ya que, cada agente tendrá en cuenta solo su posición relativa y la localización de los obstáculos de su entorno.

Respecto a la configuración del entrenamiento de la red neuronal, el entrenamiento con CMA y algoritmos genéticos simples, GA, se ha obtenido en 1000 generaciones, donde cada generación constaba de una población de 32 individuos. La tasa de elitismo en el algoritmo genético utilizado es del 10% y este algoritmo no realiza ninguna mutación en los individuos. En cuanto a los parámetros de CMA, el valor inicial de $\sigma$ utilizado es $\sigma = 0,1$.

En esta sección se ha realizado el entrenamiento de un entorno con obstáculos y sin obstáculos (ver Fig. 7.4). En ambos entornos la ruta del robot líder es aleatoria y en el entorno

con obstáculos siempre hay 2 obstáculos con localización aleatoria que siempre están en el camino que debe seguir alguno de los robots del enjambre, con la finalidad de que consigan aprender a esquivarlos.

**(a)** Entorno sin obstáculos con rutas del líder aleatorias

**(b)** Entorno de entrenamiento con obstáculos cuya localización es aleatoria

**Figura 7.4:** Entornos utilizados durante el proceso de entrenamiento. El robot líder se encuentra en la posición superior del enjambre. En ambos entornos la ruta del robot líder es aleatoria y desconocida para los robots del enjambre.

Los resultados que se muestran a continuación son la media de distancia de error acumulada sobre la distancia recorrida total entre los 4 robots del enjambre que ejecutan la conducta obtenida mediante aprendizaje. La distancia de error es la distancia entre la posición real y la posición ideal que deberían ocupar respecto al robot líder de los robots del enjambre. Los resultados obtenidos en un entorno sin obstáculos y utilizando solo el sensor de posicionamiento relativo se muestran en la figura 7.5. Además, se ha añadido ruido gaussiano al sensor de posicionamiento de cada robot para comprobar su robustez frente al ruido externo (ver Fig. 7.6). Como podemos observar en ambos resultados, podemos considerar que la conducta se ha aprendido con éxito debido a la baja distancia de error que contienen las rutas de los robots del enjambre.

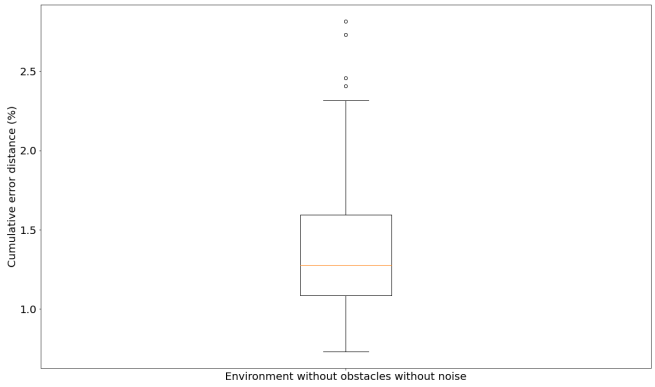



**Figura 7.5:** Distancia de error relativa a la distancia total en 100 ejecuciones en el entorno de entrenamiento. El entorno no contiene obstáculos y las rutas del robot líder son aleatorias.

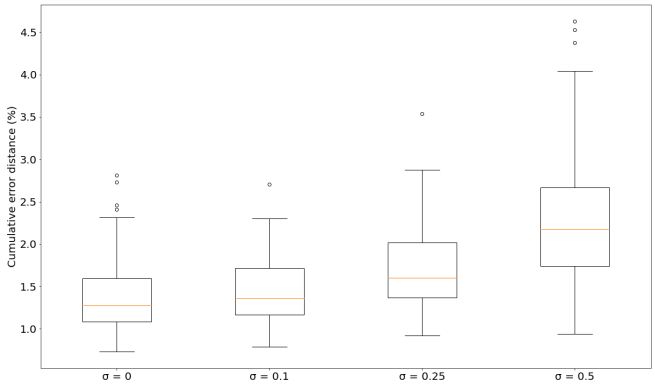



**Figura 7.6:** Distancia de error relativa a la distancia en 100 ejecuciones con distinto valor de ruido en el sistema de posicionamiento. En este gráfico se muestran los resultados obtenidos en el entorno de entrenamiento sin obstáculos pero, aplicando ruido gaussiano al sistema de posicionamiento relativo de los robots respecto al robot líder. Los valores del ruido gaussiano aplicado presentan una distribución normal con $\mu = 0$ y $\sigma = 0$, $\sigma = 0.1$, $\sigma = 0.25$, $\sigma = 0.5$.

Una vez que se ha solucionado el problema solo con el sensor de posicionamiento relativo, podemos añadirle al entorno obstáculos con la finalidad de que los robots los esquiven utilizando la información del sensor láser. Para solucionar este problema, se compararán los resultados obtenidos con las estrategias CMA y GA para ver qué estrategia ofrece mejores resultados. En cuanto a los resultados del entrenamiento, en la figura 7.7 podemos ver la convergencia de ambos algoritmos durante el entrenamiento a lo largo de 1000 generaciones.

Esta figura muestra que la estrategia CMA obtiene el máximo local antes que la estrategia GA.

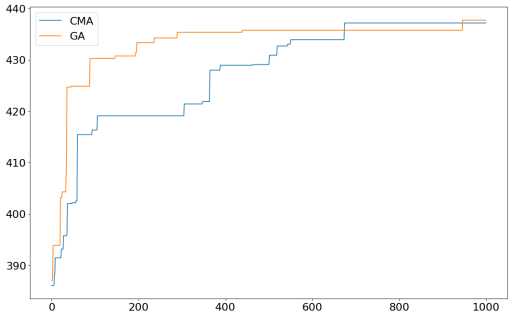
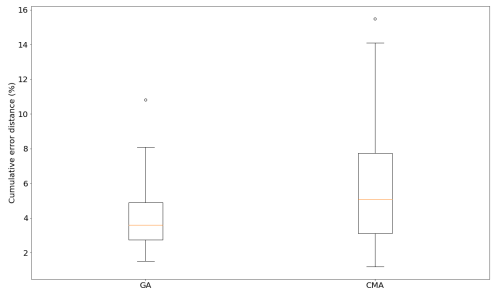
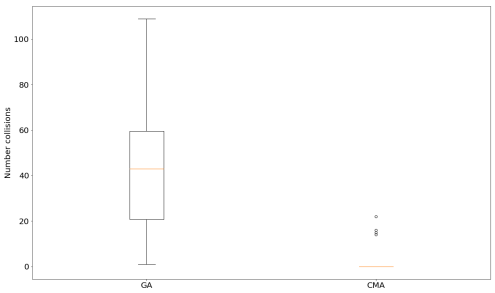



**Figura 7.7:** Recompensas obtenidas en el entrenamiento del proceso a lo largo de 1000 generaciones. La máxima recompensa global posible es de 450 puntos, ya que se realizan 450 pasos de ejecución. El entrenamiento se ha realizado en un entorno con 2 obstáculos situados aleatoriamente y un enjambre formado por 5 robots, donde el líder se encuentra en la zona superior central y su ruta contiene movimientos aleatorios.

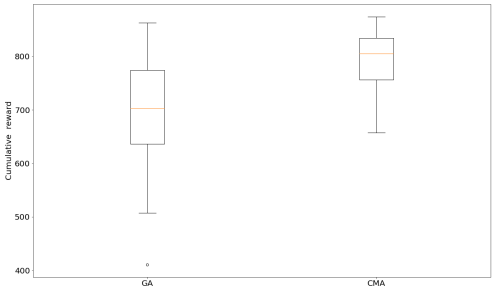
Para evaluar las conductas obtenidas por ambas estrategias, se ejecutarán las redes neuronales entrenadas en el entorno de entrenamiento con obstáculos. En estas pruebas se realizaron 900 movimientos en cada episodio, por lo que la recompensa máxima es de 900 puntos. Tras ejecutar las políticas obtenidas en 100 episodios, se obtienen los resultados de la figura 7.8. Esta figura muestra una comparación de las dos estrategias con 3 métricas. La primera métrica es la distancia media de error acumulada respecto a la distancia total recorrida del enjambre, la media se obtiene sobre los 4 robots del enjambre que ejecutan la política. La distancia de error es la distancia entre la posición real y la posición ideal que deberían ocupar con respecto al robot líder del enjambre. En esta métrica la estrategia GA proporciona mejores resultados que CMA, ya que la distancia de error es menor. La segunda métrica es el número total de colisiones en el episodio, donde CMA da mejores resultados que GA, ya que el comportamiento obtenido colisiona menos veces con los obstáculos. Por último, está la tercera métrica, la recompensa obtenida durante el episodio. En este gráfico se puede observar que la política CMA también ofrece mejores resultados que GA, ya que, la política generaba un mayor valor de recompensa. Por lo tanto, podemos concluir que la propuesta CMA ofrece mejores resultados que GA en 2 de las 3 métricas utilizadas.




**(a)** Distancia de error relativa a la distancia total recorrida en 100 ejecuciones.



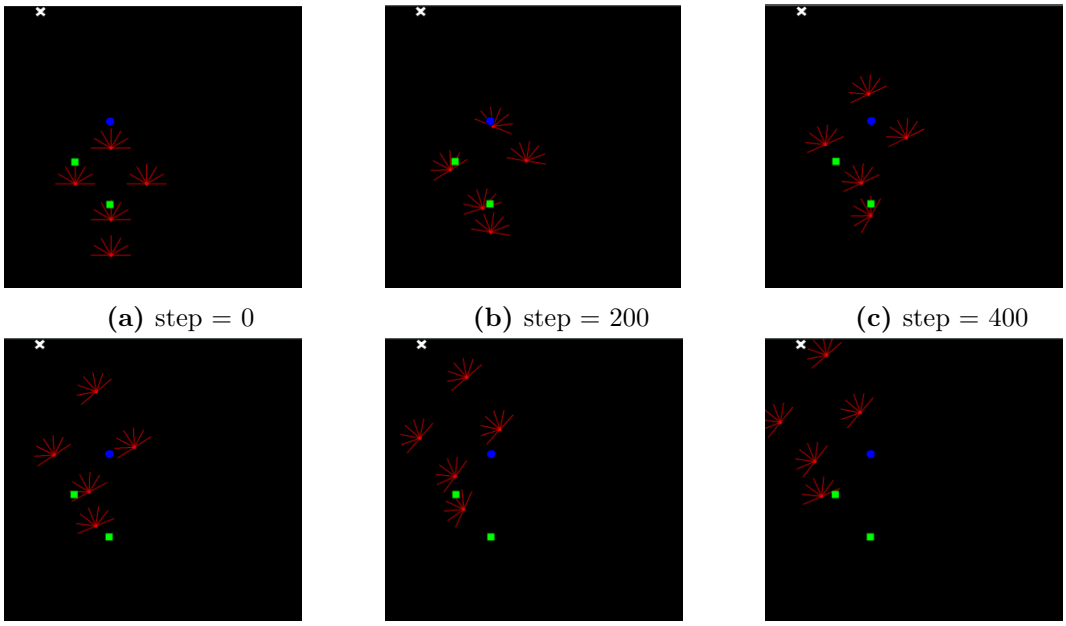

**(b)** Número total de colisiones durante un episodio en 100 ejecuciones.




**(c)** Recompensa obtenida acumulada en 100 ejecuciones.

**Figura 7.8:** Comparación de los resultados obtenidos con las estrategias CMA y GA. Las políticas se han ejecutado en el entorno de entrenamiento en 900 pasos de ejecución. La recompensa máxima global de estas ejecuciones es de 900 puntos.

Ahora bien, después de concluir que CMA ofrece mejores resultados que GA en las métricas mencionadas anteriormente, se evaluará si la conducta aprendida mediante CMA es robusta frente al ruido o a nuevos obstáculos. En la figura 7.9 se muestra un ejemplo de la conducta obtenida en el enjambre en un entorno con 2 obstáculos.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 7.9:** Secuencia de una ejecución de la conducta aprendida donde los robots del enjambre siguen al robot líder mientras esquivan los obstáculos. Donde (a) es el instante inicial y (f) es el instante final. La posición inicial del robot líder está representada mediante un punto azul y al posición final está representada mediante una cruz blanca. Los obstáculos están representados por los cuadrados verdes.

Se ha ejecutado la conducta aprendida en el entorno de entrenamiento (ver Fig. 7.10). Además, se ha realizado una comparativa con los resultados obtenidos con diferente número de obstáculos (ver Fig. 7.11) con el fin de comprobar que el comportamiento es robusto frente a un número variable de obstáculos. La conducta aprendida en un entorno sin obstáculos genera menos error acumulado, ya que, los robots no deben esquivar ningún obstáculo mientras siguen al robot líder. Esta distancia de error presenta un valor reducido, por lo tanto, se ha aprendido la conducta de manera satisfactoria. Por último, se ha añadido ruido gaussiano al sensor de posicionamiento de cada robot para comprobar su robustez frente al ruido externo (ver Fig. 7.12).

Una vez terminada la experimentación anterior hemos determinado que la conducta es robusta frente al ruido en el sistema de posicionamiento, ya que, al aumentar levemente el ruido no se perjudica gravemente a la conducta final. Asimismo, los robots no colisionan con los obstáculos y siguen al robot líder durante toda la trayectoria. Además, esta conducta también es robusta frente a distintos escenarios con diferente número de obstáculos y distintas rutas del robot líder.

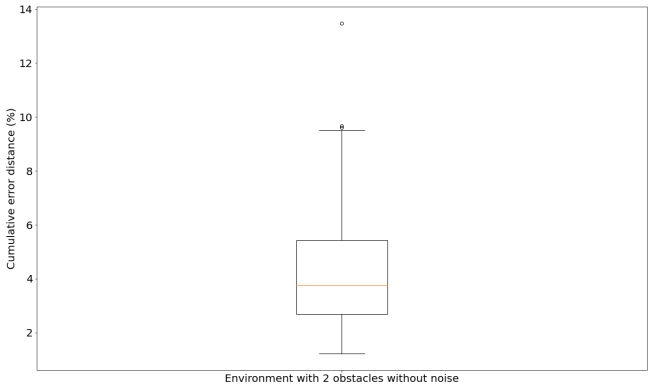



**Figura 7.10:** Distancia de error relativa a la distancia total recorrida en 100 ejecuciones en el entorno de entrenamiento. El entorno contiene 2 obstáculos de localizaciones aleatorias y las rutas del robot líder también son aleatorias.

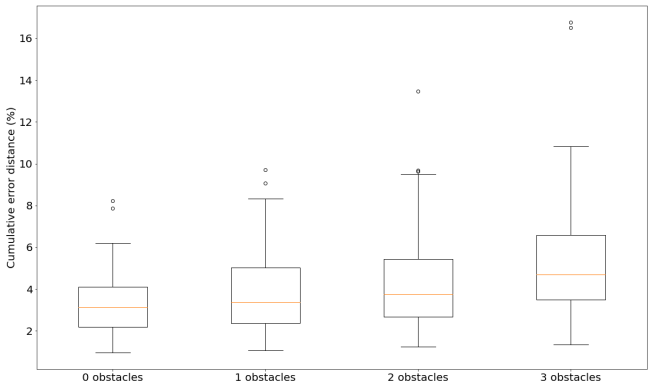



**Figura 7.11:** Distancia de error relativa a la distancia total recorridaen 100 ejecuciones con distinto número de obstáculos. En este gráfico se muestran los resultados obtenidos en entornos con 0, 1, 2 y 3 obstáculos aleatorios.

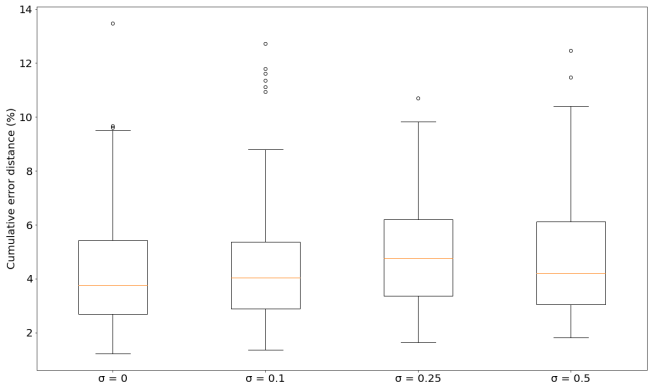



**Figura 7.12:** Distancia de error relativa a la distancia total recorrida en 100 ejecuciones con distinto valor de ruido en el sistema de posicionamiento. En este gráfico se muestran los resultados obtenidos en el entorno de entrenamiento con obstáculos pero, aplicando ruido gaussiano al sistema de posicionamiento relativo de los robots respecto al robot líder. Los valores del ruido gaussiano aplicado presentan una distribución normal con $\mu = 0$ y $\sigma = 0$, $\sigma = 0.1$, $\sigma = 0.25$, $\sigma = 0.5$.

## 7.4 Resultados obtenidos en diferentes entornos

En esta sección se mostrarán los resultados obtenidos en distintos enjambres con la finalidad de comprobar si la conducta aprendida a través de la estrategia CMA es escalable y robusta. La conducta que se ha evaluado es la obtenida en un entorno con obstáculos utilizando el sensor láser para detectarlos.

### 7.4.1 Resultados con diferente número de robots

Para comprobar que la conducta aprendida sea escalable se han realizado ejecuciones en entornos con distinto número de robots. Además, para evaluar solo las distancias de error que se generan al ejecutar la conducta aprendida se ha realizado la experimentación en entornos sin obstáculos (ver Fig. 7.13). En los resultados se muestra que la distancia de error total del enjambre es similar en todas las ejecuciones (ver Fig. 7.14), por lo tanto, la conducta no se ve afectada negativamente al modificar el número de robots del enjambre. Este suceso se debe a que cada robot tiene en cuenta su posición relativa al robot líder y no tiene información de los robots del entorno.

**(a)** Enjambre de 3 robots **(b)** Enjambre de 5 robots **(c)** Enjambre de 10 robots

**Figura 7.13:** Entornos utilizados para evaluar la flexibilidad de número de robots del enjambre.

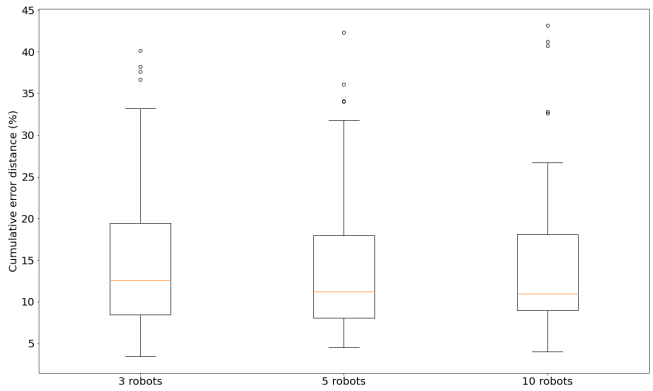



**Figura 7.14:** Distancia de error total relativa a la distancia total recorrida del enjambre en 100 ejecuciones en el entorno de entrenamiento. El entorno no tiene obstáculos y los enjambres contienen 3, 5 y 10 robots.

### 7.4.2 Resultados obtenidos en diferentes formaciones

En este apartado se ha evaluado la flexibilidad del algoritmo con diferentes formaciones iniciales. Se ha probado una ruta al azar en entornos con obstáculos para comprobar que los agentes los esquiven y, por otro lado, se han ejecutado 100 episodios con rutas al azar sin obstáculos para observar si siguen correctamente al líder en cada paso de ejecución manteniendo la forma inicial dada.

Por una parte, en la gráfica 7.15 se puede ver cómo los robots mantienen la forma inicial en todo momento mientras siguen la ruta del robot líder con una distancia de error reducida. Además, en la figura 7.16 se puede ver una secuencia de la trayectoria recorrida con la formación en media luna mientras algunos de los agentes esquivan los obstáculos.

Por otra parte, en la gráfica 7.17 también se puede observar que la distancia de error es reducida, lo que indica que los robots del enjambre mantienen en todo momento la forma de pentágono. Además, en la secuencia de la figura 7.18 se puede observar cómo los agentes

esquivan los obstáculos del entorno mientras siguen los movimientos del robot líder.

Finalmente, podemos concluir que el algoritmo es flexible frente a nuevas formaciones, nuevos obstáculos y nuevos robots líderes, ya que, estas formaciones no tienen la misma posición inicial de los robots líderes del enjambre.

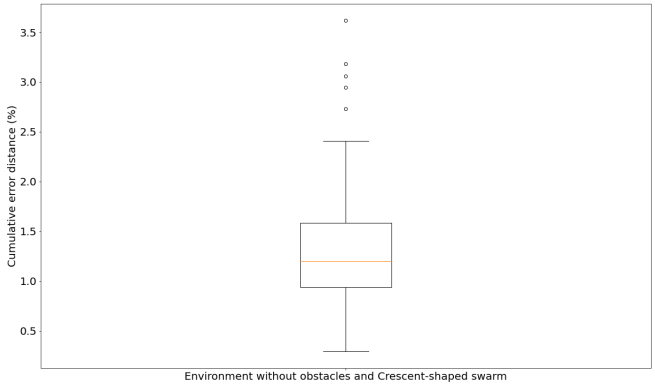
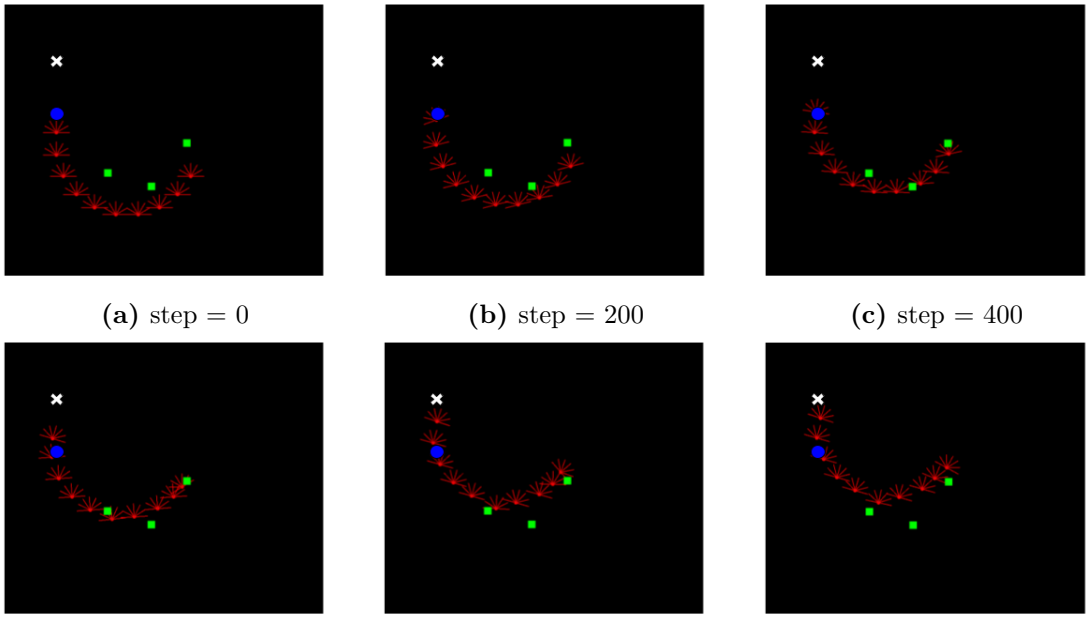



**Figura 7.15:** Distancia de error relativa a la distancia total recorrida en 100 ejecuciones en una formación de media luna. En este gráfico se muestran los resultados obtenidos de ejecutar la conducta en 100 episodios en un entorno sin obstáculos y con una formación de media luna.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 7.16:** Secuencia de un episodio donde los robots forman una media luna y siguen al robot líder mientras esquivan los obstáculos. El robot líder se encuentra en la zona superior izquierda.

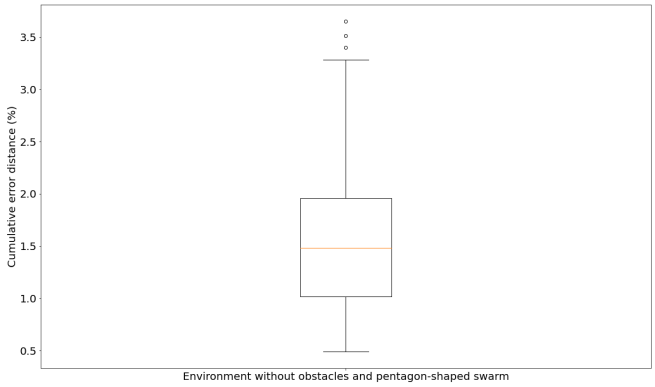
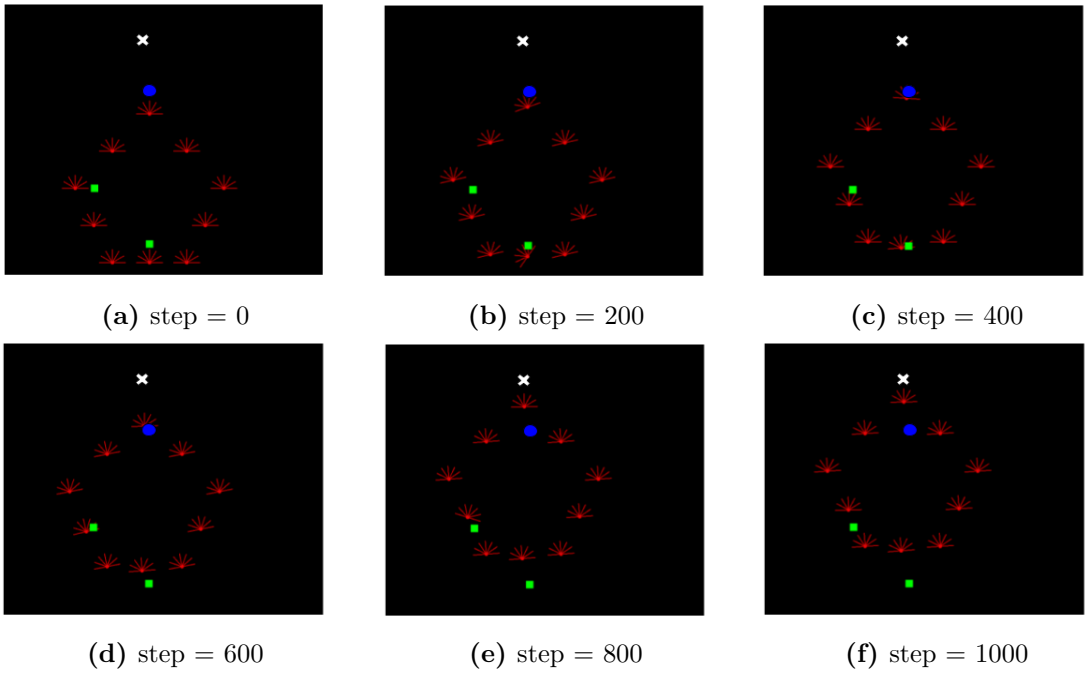



**Figura 7.17:** Distancia de error relativa a la distancia total recorrida en 100 ejecuciones en una formación de pentágono. En este gráfico se muestran los resultados obtenidos de ejecutar la conducta en 100 episodios en un entorno sin obstáculos y con una formación de pentágono.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 7.18:** Secuencia de un episodio donde los robots forman un pentágono y siguen al robot líder mientras esquivan los obstáculos. El robot líder se encuentra en la zona central superior.

## 7.5 Tarea de transporte colectivo de objetos

La tarea de transporte colectivo de objetos en robótica de enjambre es muy útil porque consigue desplazar objetos con un enjambre de robots simples que cooperan entre ellos. Las ventajas del uso de robótica de enjambre en esta tarea frente a sistemas multi-robot son:

- Escalabilidad para poder transportar un objeto con un número variable de robots, pudiendo superar la pérdida de alguno de los miembros del enjambre.
- Robustez frente a nuevos objetos, nuevos enjambres o ruido presente en el entorno.
- Flexibilidad frente a nuevos entornos.

El objetivo de esta sección es poder ejecutar transportar un objeto entre dos puntos mediante robótica de enjambre con una conducta aprendida previamente. Los resultados se obtendrán en un entorno simulado y la posición final del objeto vendrá determinada por la ruta de uno de los robots del enjambre que actuará como robot líder. La conducta que utilizarán los robots del enjambre será la aprendida en este capítulo, donde uno de ellos actuará como robot líder y ejecutará una ruta previamente definida. Mientras que el resto del enjambre utilizará la red neuronal entrenada anteriormente para seguir al robot líder manteniendo la forma inicial. La ventaja de utilizar esta conducta es que solo hace falta que uno de los robots conozca el destino del objeto. Además, utilizando esta conducta el enjambre mantiene la forma inicial de media luna mientras siguen al robot líder. La razón de utilizar este tipo de formación es que la media luna facilita el arrastre de objetos en cualquier dirección. Debido a

que los robots no deben esquivar el objeto, es decir, deben acercarse a él, no se puede utilizar la conducta de evitación de obstáculos a través del sensor láser. Por lo tanto, se utilizará la conducta aprendida utilizando solo el sensor de posicionamiento relativo como entrada de la red neuronal.

### 7.5.1 Estado del arte

En la propuesta de Tuci y cols. (2018) se recopilan diversas estrategias de transporte colectivo de objetos mediante sistemas multi-robot. A continuación, se enumeran las categorías que se recopilan en este artículo.

- Estrategia de empuje: Los robots no están unidos físicamente al objeto que deben desplazar y lo transportan empujándolo a través del entorno.
- Estrategia de agarre: Los robots están unidos físicamente al objeto y el transporte se realiza tirando o empujando el objeto.
- Estrategia de enjaulado: Esta estrategia es similar a la estrategia de empuje. Los robots se agrupan alrededor del objeto para simular una jaula y atraparlo entre el grupo. Para desplazar el objeto lo mantienen firmemente durante el transporte.

La tarea diseñada en esta sección pertenece a la categoría de estrategia de empuje, ya que, los robots no están unidos al objeto y lo desplazan de lugar arrastrándolo. Respecto a esta estrategia, en la propuesta de Alkilabi y cols. (2017) se utilizan grupos homogéneos de robots físicos E-puck para coordinar y sincronizar sus acciones para transportar un objeto. Para resolver esta tarea, el controlador del robot está compuesto por una red neuronal recurrente de tiempo continuo (CTRNN) entrenada mediante un algoritmo evolutivo simple basado en el mecanismo de selección de la ruleta. Este método de selección es un método de selección estocástico, en el que la probabilidad de selección de un individuo, en este caso una red neuronal, es proporcional a su valor de recompensa. Como resultado se obtienen estrategias robustas con respecto a las variaciones de las características de los objetos y relativamente escalables con respecto al tamaño del grupo.

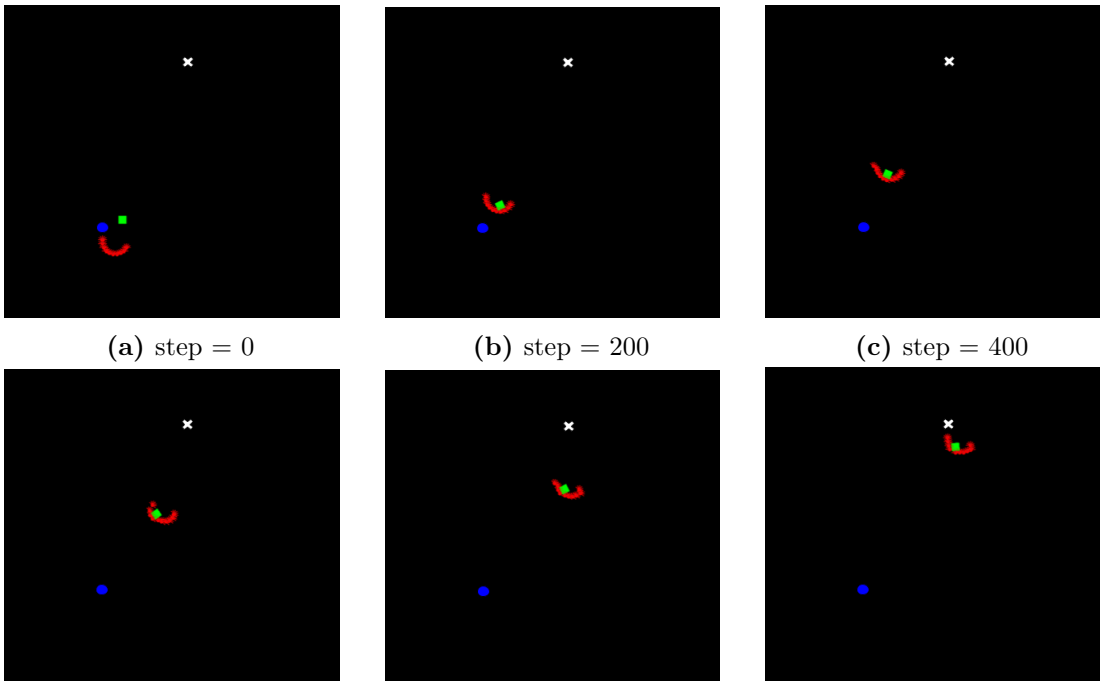
### 7.5.2 Ejecución de la tarea en un entorno simulado

En este apartado se utilizará la conducta aprendida en este capítulo con la estrategia evolutiva CMA-ES utilizando solo el sensor de posicionamiento relativo para poder transportar un objeto de un punto origen a un punto destino. Como la finalidad de esta tarea es arrastrar un objeto a un lugar determinado, no se ha empleado el sensor láser para evitar que al acercarse los agentes al objeto no lo esquiven. El objetivo de este apartado es mostrar el resultado de una de las posibles aplicaciones de la conducta aprendida.

Para desarrollar esta tarea se ha utilizado un escenario cuadrado con un objeto cerca de los robots en formación de media luna. En este problema se ha aumentado la velocidad de los robots con el objetivo de recorrer más terreno sin ralentizar la simulación. El objeto es un cuadrado de 20x20 centímetros, con una masa de 50 gramos y una fricción de 0.2. En la figura 7.19 se puede ver una secuencia de la ruta donde un enjambre de 10 robots llega hasta

el objeto y lo arrastra hasta un punto destino, resultante de una trayectoria del robot líder programada anteriormente. Se ha elegido ejecutar primero la conducta en un enjambre de 10 robots porque es con este número de robots es más sencillo rodear el objeto y trasladarlo a la posición destino. Una vez que se han obtenido estos resultados evaluará reducir el enjambre al mínimo número de agentes posible que realiza esta tarea sin problemas. Se ha ejecutado la conducta en enjambres de 3, 4 y 5 robots, siendo el último enjambre el que si conseguía realizar la tarea, como se muestra en la figura 7.20. En los enjambres con menor número de agentes los robots no podían arrastrar el objeto con las mismas características porque rodeaban una pequeña parte de él y no se lograba transportarlo.

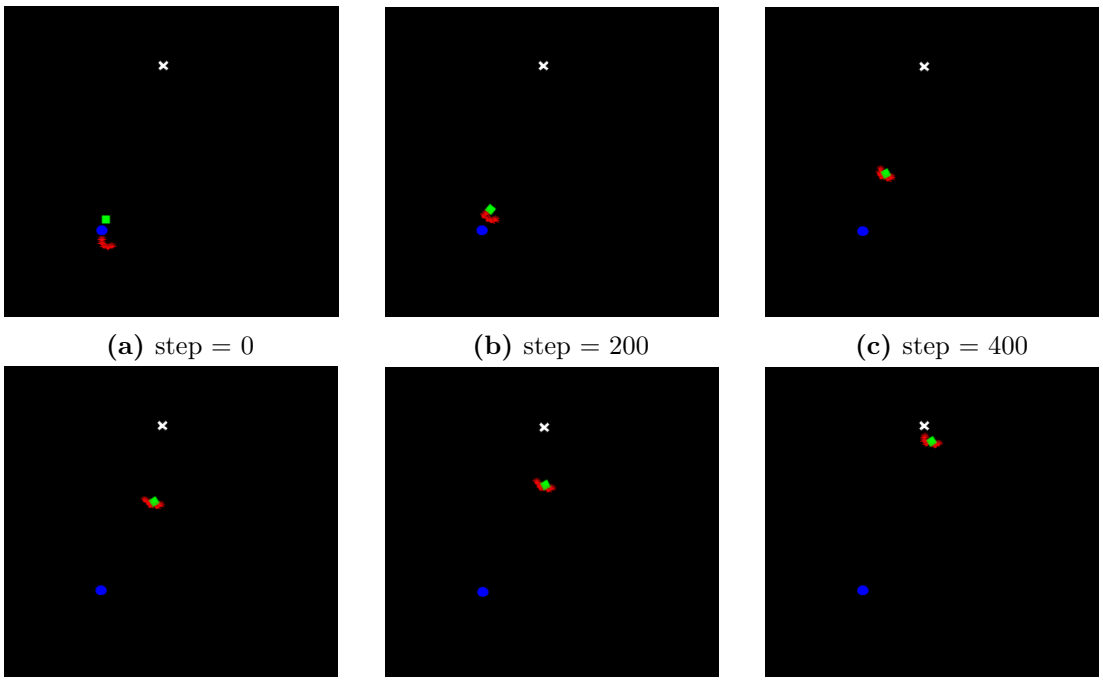
Por último, teniendo en cuenta que cada robot posee una masa de 25 gramos y un diámetro de 10 centímeros, mientras que el objeto está representado como un cuadrado de 20 centímetros y posee una masa de 50 gramos, un enjambre de 5 robots podría transportar un objeto de mayor tamaño que uno de los robots.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 7.19:** Secuencia de un episodio donde 10 robots forman una media luna y arrastran un objeto desde un punto origen hasta el punto destino. El punto azul indica la localización inicial del robot líder y la cruz blanca, la localización final. El objeto está representado por un cuadrado de color verde y de tamaño de 20x20 centímetros.



**(a)** step = 0 **(b)** step = 200 **(c)** step = 400

**(d)** step = 600 **(e)** step = 800 **(f)** step = 1000

**Figura 7.20:** Secuencia de un episodio donde 5 robots forman una media luna y arrastran un objeto desde un punto origen hasta el punto destino. El punto azul indica la localización inicial del robot líder y la cruz blanca, la localización final. El objeto está representado por un cuadrado de color verde y de tamaño de 20x20 centímetros.

## 7.6 Conclusión

Después del desarrollo de este capítulo, podemos concluir en que se ha obtenido una conducta de formación de un enjambre robusta frente al ruido, escalable frente a un número distinto de miembros del enjambre y flexible a nuevas rutas del robot líder. Además, a esta tarea se le puede agregar la tarea de esquivar obstáculos del entorno utilizando un sensor láser, cuya conducta ha sido obtenida por dos estrategias evolutivas: GA, cuyo algoritmo ya se ha utilizado en propuestas con tareas similares a la de este capítulo, y CMA, una estrategia que ha sido propuesta tras el resultado de un estudio sobre el rendimiento de varias estrategias evolutivas. En esta comparativa se ha obtenido que la estrategia CMA ofrece mejores resultados respecto a recompensas y número de colisiones con los obstáculos.

Por otro lado, la conducta obtenida mediante CMA tolera otras formaciones del enjambre como una formación de pentágono o media luna. Esta última formación es útil a la hora de realizar un transporte colectivo de objetos, ya que esta forma facilita el arrastre de objetos. Además, es robusta frente al ruido y a nuevas localizaciones de los obstáculos.

Por otra parte, respecto a la tarea de transporte de objetos con la política obtenida de mantenimiento de una formación compleja se puede concluir que la política obtenida es útil para poder transportar objetos mediante la estrategia de empuje. Además, al utilizar la forma

de media luna, el enjambre puede arrastrar el objeto con mayor facilidad hasta en los giros de las trayectorias. También se ha demostrado que esta conducta es escalable, ya que, se puede realizar la tarea tanto con 5 como con 10 robots. Aunque, se necesita un mínimo de 5 robots para realizar la tarea porque con un número inferior no se puede arrastrar un objeto que posee mayor masa y tamaño que uno de los robots del enjambre.

Como trabajo futuro, se podría incluir esta conducta en un entorno físico y para observar como se comporta el enjambre en un entorno real. Para realizar este experimento, se requiere un sensor de posicionamiento de los robots respecto al robot líder y un sensor láser para la localización de los obstáculos.

# 8 Publicaciones derivadas de este proyecto

En este capítulo se comentarán las publicaciones derivadas de este proyecto en dos congresos internacionales. Las publicaciones, que se explicarán con más detalle a continuación, corresponden a dos capítulos de este trabajo: Capítulo 5 y Capítulo 7.

## 8.1 Congreso MLMI 2020

El congreso MLMI es una conferencia que se centra en el aprendizaje automático y la inteligencia artificial. El tema principal de la conferencia es abordar y deliberar sobre los últimos avances y las tendencias recientes en la investigación y las aplicaciones del aprendizaje automático y la inteligencia artificial. El propósito de la conferencia es proporcionar una oportunidad a los científicos, ingenieros, industriales, académicos y otros profesionales de todo el mundo para interactuar e intercambiar sus nuevas ideas y resultados de investigación en campos relacionados.

El artículo que recoge los resultados del Capítulo 5 se publicaron en la tercera edición de la conferencia Machine Learning and Machine Intelligence (MLMI 2020). A continuación, se incluye el resumen del artículo publicado.

”This article studies the performance of different evolutionary strategies for deep reinforcement learning policy optimization. The policy will be centred in an important swarm robotic task: the aggregation of simple robots in the environment. The main inspiration for robotic swarm comes from the observation of social animals. Ants, bees, birds, and fish are some examples of how simple individuals can succeed when they gather in groups. In addition, is important to highlight that aggregation may be considered as a previous requirement to tackle another tasks.

Due the design of a swarm behaviour is a comprehensive process of experimentation, one of the current solutions is learn a policy able to control a robot. Gradient descent techniques have demonstrated their learning power in the field of neural networks and reinforcement learning, among others. But some of these techniques are difficult to be applied in the field of robotics because the requirements needed to calculate the gradient to be informative are not met. For that reason, in this article we are going to use and compare the evolutionary strategies CMA-ES, PEPG, SES, GA y OpenAI-ES. A fast simulator, based in the differential robot Mbot Ranger, will be used. Once the aggregation task is learned we will compare each strategy for different swarm sizes to analyse the convergence time, the quality of the policies, its scalability and its capacity to generalize.”

## 8.2 Congreso CAEPIA 2021

El Congreso de la Asociación Española de Inteligencia Artificial (CAEPIA) es un foro bienal abierto a investigadores de todo el mundo para presentar y discutir sus últimos avances científicos y tecnológicos en Inteligencia Artificial. En 2021 el CAEPIA se celebra en el contexto del 6th Spanish Congress on Computer Science (CEDI 2020), que incluye numerosos eventos y actividades relacionados con la Informática, con mayor visibilidad e impacto social.

El artículo que recoge los resultados del CAPÍTULO 7 ha sido aceptado para su publicación en la XIX Conference of the Spanish Association for Artificial Intelligence (CAEPIA) bajo el título *"Learning A Swarm Formation Policy using Deep Reinforcement Learning"*. Los resultados que se publicarán son los obtenidos por la conducta de mantenimiento de una forma mientras se esquivan los obstáculos del entorno. Además, se compararán los resultados obtenidos mediante la estrategia CMA-ES con los obtenidos a través de algoritmos genéticos simples. A continuación, se incluye el resumen del artículo publicado.

"The purpose of this paper is to solve a swarm robotics task where a swarm of robots manages to follow the movements of a robot leader (that will develop an independent behaviour), and they maintain its formation while avoiding the environment obstacles. For this task, the problem will be modelled as a Markov Decision Process (MDP) and learned using Deep Reinforcement Learning with an evolutionary optimization CMA-ES strategy. Thus, we propose an fitness function based on the relative distance difference between the initial positions and the current positions formed by the robots. All the swarm individuals will have a positioning sensor relative to the leader robot and a laser sensor to be able to locate obstacles in the environment. Training and results analysis will be carried out in a realistic simulated environment. Once the system has been modelled and the learning process has been completed, we will study the robustness of the learned policy. The obtained results show that it is possible to learn a single swarm control policy able to follow a random behaviour of the leader, with different number of obstacles or noise in the local positioning system. In addition, we will compare the results obtained using genetic algorithms with those obtained by CMA-ES."

# 9 Conclusiones

Una vez finalizado este proyecto podemos concluir que se han obtenido políticas de robótica de enjambre complejas utilizando estrategias evolutivas. Además, se ha utilizado un simulador en 2D del robot MBot Ranger que usa física realista. Este simulador se ha utilizado tanto para realizar el entrenamiento de la política como para evaluar las conductas obtenidas. También se ha ejecutado una conducta de robótica de enjambre compleja en un entorno real, es decir, se ha transferido la conducta de formación en cadena en un entorno real utilizando 2 robots MBot Ranger.

Por otra parte, cabe destacar que se han evaluado las conductas resultantes de cada tarea para verificar su robustez, flexibilidad o escalabilidad. Estas comparativas se han ejecutado en las conductas de formación en cadena y de mantenimiento de una forma compleja en un entorno con obstáculos.

Por último, se han cumplido los objetivos propuestos:

- **Evaluación de eficiencia de las estrategias evolutivas**: Se han evaluado diferentes estrategias evolutivas (CMA-ES, PEPG, GA, OPENAI-ES y SES) en enjambres de distintos tamaños. Además, se han evaluado tanto los resultados de entrenamiento como los obtenidos en test.
- **Aprendizaje de una conducta en formación en cadena**: Se ha conseguido obtener una política óptima mediante CMA-ES aplicado al DRL. Esta conducta ha sido obtenida en un entorno simple y se ha solucionado utilizando el sensor láser de los robots para detectar al robot delantero mientras siguen los pasos del robot líder. Además, se ha evaluado la robustez frente al ruido y la flexibilidad de esta conducta.
- **Transferencia de una conducta aprendida a robots reales**: Se ha conseguido ejecutar la política óptima obtenida de la conducta de formación en cadena en un entorno real con dos robots MBot Ranger, donde uno actuaba como robot líder y el otro como robot cadena. El robot cadena utilizaba una placa Raspberry Pi para obtener los datos del sensor láser. Por último, la conducta presentaba algunas dificultades a la hora de que el robot cadena siguiese los movimientos del robot líder, ya que, este último era débilmente visible para el sensor láser.
- **Aprendizaje del mantenimiento de una formación en un entorno simple**: Se ha realizado el aprendizaje de mantenimiento de una forma compleja en un entorno simple sin obstáculos utilizando un sensor de posicionamiento relativo al robot líder. En esta conducta se ha evaluado su robustez frente al ruido presente en el sistema de posicionamiento relativo. Para obtener esta conducta se ha utilizado la estrategia evolutiva CMA-ES.
- **Aprendizaje del mantenimiento de una formación con evitación de obstácu-**

**los**: Se ha obtenido una política óptima para la tarea de mantenimiento de una formación compleja en un entorno con obstáculos. Para resolver esta tarea se ha utilizado un sensor de posicionamiento relativo al robot líder para seguir sus movimientos manteniendo la forma inicial y se ha utilizado también un sensor láser para la lectura de los obstáculos para poder esquivarlos. Esta política también se ha obtenido con CMA-ES y con esta estrategia se han obtenido mejores resultados que utilizando algoritmos genéticos simples. Además, también se ha evaluado su robustez frente al ruido y su flexibilidad respecto a distintos escenarios.

- **Análisis de métrica de las conductas de enjambre**: Se han analizado distintas métricas como la robustez frente al ruido, flexibilidad a nuevos escenarios o escalabilidad respecto a distinto número de robots del enjambre. Para evaluar la conducta de formación en cadena se han comparado las diferencias entre las distancias de separación entre cada robot y el número de robots que se había separado de la cadena. En cambio, para evaluar la conducta de mantenimiento de formaciones complejas se han comparado las distancias de error relativas a las distancias totales recorridas y el número de colisiones, en la conducta de evitación de obstáculos.
- **Emplear el aprendizaje de una conducta para una tarea compleja de robótica de enjambre**: Por último, se ha utilizado la conducta de mantenimiento de una formación compleja en un entorno simple para realizar una tarea de transporte colectivo con una formación de media luna y utilizando una estrategia de empuje para transportar el objeto. De esta conducta se ha evaluado su escalabilidad siendo 5 el número mínimo de robots que debe estar en el enjambre para transportar un objeto determinado.

En conclusión, podemos afirmar que se han obtenido distintas conductas complejas de robótica de enjambre a través del uso de estrategias evolutivas para el aprendizaje por refuerzo, donde la política óptima obtenida cumple las características de robótica de enjambre. Además, para resolver estas tareas se ha utilizado robots simples y simuladores realistas para el entrenamiento.

Como futuras líneas se podría añadir más experimentación en sistemas reales, tanto en la conducta de formación en cadena como en la conducta de mantenimiento de una formación compleja con y sin obstáculos. Además, se podría investigar más en profundidad la aplicabilidad de estas conductas en otras tareas de robótica de enjambre, como en la tarea de transporte colectivo de objetos.

# Bibliografía


Alkilabi, M. H. M., Narayan, A., y Tuci, E. (2017). Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies. *Swarm intelligence*, *11*(3), 185–209.

Baldassarre, G. (2006). Evolution of collective behaviour: coordination object retrieval in groups of physically-linked simulated robots. Descargado 20/08/2019, de `http://laral.istc.cnr.it/baldassarre/demos/2003swarmobject/swarmobject.htm`

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

Campo, A., Nouyan, S., Birattari, M., Groß, R., y Dorigo, M. (2006). Enhancing cooperative transport using negotiation of goal direction. *Lecture notes in computer science: Vol. 4150. Proceedings of the fifth international workshop on ant colony optimization and swarm intelligence (ANTS 2006)*, 365–366.

Christensen, A. L., O'Grady, R., y Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, *13*(4), 754–766.

Christensen, A. L., O'Grady, R., y Dorigo, M. (2008). Swarmorph-script: a language for arbitrary morphology generation in self-assembling robots. *Swarm Intelligence*, *2*(2-4), 143–165.

Ducatelle, F., Di Caro, C. P. G. A., y Gambardella, L. M. (2011). Self-organized cooperation between robotic swarms. *Swarm Intelligence*, *5*(2), 73–96.

Ferrante, E., Turgut, A., Huepe, C., Stranieri, A., Pinciroli, C., y Dorigo, M. (2012). Self-organized flocking with a mobile robot swarm: a novel motion control method. *Adaptative behavior*, *20*, 460–477. doi: 10.1177/1059712312462248

Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G., y Theraulaz, G. (2005). Aggregation behaviour as a source of collective decision in a group of cockroach-like robots. *Lecture notes in artificial intelligence, Advances in artificial life*, *3630*, 169–178.

Ha, D. (2017a). Evolving stable strategies. *blog.otoro.net*. Descargado de `http://blog.otoro.net/2017/11/12/evolving-stable-strategies/`

Ha, D. (2017b). A visual guide to evolution strategies. *blog.otoro.net*. Descargado de `http://blog.otoro.net/2017/10/29/visual-evolution-strategies/`

Howard, A., Matarić, M. J., y Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem. *Proceedings of the 2002 international symposium on distributed autonomous robotic systems (DARS 2002)*, 299–308.

Huang, B.-Q., Cao, G.-Y., y Guo, M. (2005). Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. , *1*, 85-89. doi: 10.1109/ICMLC.2005.1526924

Ketkar, N. (2017). *Deep learning with python: A hands-on introduction*. Apress. doi: 10.1007/978-1-4842-2766-4

Lin, C.-C., Chen, K.-C., Hsiao, P.-Y., y Chuang, W.-J. (2013). Motion planning of swarm robots using potential-based genetic algorithm. *International Journal of Innovative Computing, Information and Control*, *9*(1), 305–318.

Liu, W. (2007). Modelling of adaptive foraging in swarm robotic systems. Descargado 20/08/2019, de `http://www.brl.ac.uk/researchthemes/swarmrobotics/swarmroboticsystems.aspx`

McLurkin, J., Smith, J., Frankel, J., Sotkowitz, D., Blau, D., y Schmidt, B. (2006). Speaking swarmish: human–robot interface design for large swarms of autonomous mobile robots. *AAAI spring symposium*, 72–75.

Melhuish, C., Holland, O., y Hoddell, S. (1999). Convoying: using chorusing for the formation of travelling groups of minimal agents. *Robotics and Autonomous Systems*, *28*(2-3), 207–216.

Melhuish, C., Welsby, J., y Edwards, C. (1999). Using templates for defensive wall building with autonomous mobile ant-like robots. *Proceedings of towards intelligent and autonomous mobile robots*, *99*.

Mondada, F., Bonani, M., Guignard, A., Magnenat, S., Studer, C., y Floreano, D. (2005). Superlinear physical performances in a swarm-bot. , *In Lecture notes in computer science: Vol. 3630. Proceedings of the VIIIth European conference on artificial life*, 282–291.

Montes de Oca, M. A., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., y Dorigo, M. (2011). Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making. *Swarm Intelligence*, *5*(3-4), 305–327.

Nouyan, S., Groß, R., Bonani, M., Mondada, F., y Dorigo, M. (2009). Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, *13*(4), 695–711.

Pagliuca, P., Milano, N., y Nolfi, S. (2020). Efficacy of modern neuro-evolutionary strategies for continuous control optimization. *Frontiers in Robotics and AI*, *7*.

Pini, G. (2011). Task partitioning in swarms of robots an adaptive method for strategy selection. Descargado 20/08/2019, de `http://iridia.ulb.ac.be/supp/IridiaSupp2011-003/index.html`

Reynolds, C. W. (1987). Flocks, herds and schools: a distributed behavioral model. *Computer Graphics*, *21*(4), 25–34.

Salimans, T., Ho, J., Chen, X., Sidor, S., y Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schmickl, T., Hamann, H., y Karl-Franzens. (2016). Beeclust: A swarm algorithm derived from honeybees. derivation of the algorithm, analysis by mathematical models and implementation on a robot swarm. , 1–45.

SOYSAL, O., BAHÇECİ, E., y ŞAHİN, E. (2007). Aggregation in swarm robotic systems: Evolution and probabilistic control. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, *15*(2), 199–225.

Spears, W. M., y Spears, D. F. (2012). Physics-based swarm intelligence.

Sperati, V., Trianni, V., y Nolfi, S. (2011). Self-organised path formation in a swarm of robots. *Swarm Intelligence*, *5*, 97–119.

Trianni, V., Groß, R., Labella, T. H., Şahin, E., y Dorigo, M. (2003a). Evolving aggregation behaviors in a swarm of robots. *Lecture notes in artificial intelligence. Advances in artificial life: 7th European conference—ECAL*, *2801*(4), 865–874.

Trianni, V., Groß, R., Labella, T. H., Şahin, E., y Dorigo, M. (2003b). Evolving aggregation behaviors in a swarm of robots. *ECAL 2003, LNAI 2801*, 865–874.

TShucker, B., y Bennett, J. K. (2007). Scalable control of distributed robotic macrosensors. *Distributed autonomous robotic systems*, *6*, 379–388.

Tuci, E., Alkilabi, M. H., y Akanyeti, O. (2018). Cooperative object transport in multi-robot systems: A review of the state-of-the-art. *Frontiers in Robotics and AI*, *5*, 59.

Turgut, A. E., Çelikkanat, H., Gökçe, F., y Şahin, E. (2008). Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, *2*(2-4), 97–120.

Werfel, J. (2011). Distributed multi-robot algorithms for the termes 3d collective construction system. Descargado 20/08/2019, de `http://www.eecs.harvard.edu/ssr/publications/`

# Lista de Acrónimos y Abreviaturas

| | |
|---|---|
| **CMA-ES** | Estrategia evolutiva de adaptación de la matriz de covarianza. |
| **DRL** | Aprendizaje por refuerzo profundo. |
| **GA** | Algoritmos genéticos simples. |
| **MDP** | Proceso de desición de Markov. |
| **NES** | Estrategias evolutivas naturales. |
| **OpenES** | OpenAI Estrategia evolutiva. |
| **PEPG** | Gradientes de política de exploración de parámetros. |
| **PFSMs** | Diseño probabilístico de máquinas de estados finitos. |
| **RL** | Aprendizaje por refuerzo. |
| **SES** | Estrategia evolutiva simple. |