Escuela
Politécnica
Superior

Desarrollo de comportamientos de enjambre mediante computación evolutiva

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Mario Almela Rubio
Tutor/es:
María del Pilar Arques Corrales

Universitat d'Alacant
Universidad de Alicante

Julio 2021

Desarrollo de comportamientos de enjambre mediante computación evolutiva

Ampliación sobre el trabajo del año pasado

Autor
Mario Almela Rubio

Tutor/es
María del Pilar Arques Corrales

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

Preámbulo

Este trabajo se enmarca en el ámbito de la robótica de enjambre y la inteligencia artificial. Su objetivo principal es la investigación de técnicas de computación evolutiva para desarrollar comportamientos colectivos complejos, así como la ampliación y actualización del trabajo previo realizado a este menester agregando nuevos comportamientos más complejos.

Los comportamientos emergentes son uno de los fenómenos más fascinantes de la naturaleza, y la causa más probable de que estemos hoy sobre la faz de este planeta. Cuando múltiples comportamientos simples se unen, pueden surgir consecuencias inesperadas. Es por esta razón que entender y replicar comportamientos colectivos supone un avance significativo y abre las puertas a una posible revolución en la robótica. Este trabajo constituye un primer paso en este camino.

Primero, se analizará el estado del arte tanto en materia de robótica de enjambre como en computación evolutiva, con la idea de adquirir una base de conocimientos sobre la que construir la propia investigación.

A continuación, se definirán los objetivos concretos del trabajo y se procederá a la actualización y ampliación del trabajo realizado anteriormente, añadiendo nuevos comportamientos e incluyendo una guía de instalación y uso.

Cabe destacar que este proyecto ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (Spain), proyecto RTI2018-096219-B-I00. Proyecto cofinanciado con fondos FEDER.

Agradecimientos

Este trabajo no hubiera sido posible sin la ayuda de mi tutora, María del Pilar Arques Corrales, que me ha proporcionado todo el material necesario para su realización y cuya contribución es altamente apreciada. También debo agradecer a mi familia y amigos por todo su apoyo en los tiempos difíciles que han acaecido últimamente.

Esa mezcla de orden y anarquía es lo que hoy llamamos comportamiento emergente.

Steven Johnson.

Índice general

Índice de figuras

1. Introducción

El objetivo final de la robótica es la creación y programación de máquinas autónomas (robots) capaces de ayudar a las personas en diferentes tareas sin demasiada o sin ninguna supervisión por parte del operador humano, capaces de interpretar su entorno y tomar decisiones para optimizar la ejecución de la tarea o adaptar su comportamiento a las circunstancias del entorno. Dentro de esta premisa, existen múltiples formas de lograr este objetivo. Una de ellas consiste en el desarrollo de máquinas muy complejas dotadas de sistemas perceptivos y cognitivos de última generación, diseñadas para ser lo más inteligentes y adaptables posibles. No obstante, este enfoque resulta muy costoso a nivel económico y técnico. Se requieren múltiples profesionales y años de investigación para lograr crear un robot de tales características. Además, un único robot puede tener fallos en alguno de sus subsistemas, comprometiendo todo el conjunto y la correcta ejecución de la tarea. Otra posibilidad, no obstante, surge al estudiar los comportamientos emergentes: comportamientos complejos que surgen de la agregación o combinación de múltiples comportamientos simples de forma no obvia (como si el conjunto fuese "algo más" que la suma de sus partes).

Un comportamiento emergente se basa en la idea filosófica de la emergencia, la capacidad de un sistema de manifestar propiedades no directamente reducibles a las de sus partes constituyentes [2]. Los sistemas emergentes son aquellos que manifiestan estos comportamientos, es decir, pueden llegar a resolver problemas sin presencia de una inteligencia centralizada ni jerárquica, sino ascendente, emergente, a partir de masas individuales sin inteligencia aparente (Fig. 1.1) [12]. En ausencia de leyes concretas que lo fuercen, el sistema empieza a mostrar un comportamiento colectivo característico de un nivel de organización superior a medida que el número de unidades individuales aumenta [3]. Ejemplos de sistemas emergentes pueden ser las ciudades, el cerebro humano o las colonias o enjambres de insectos [2].

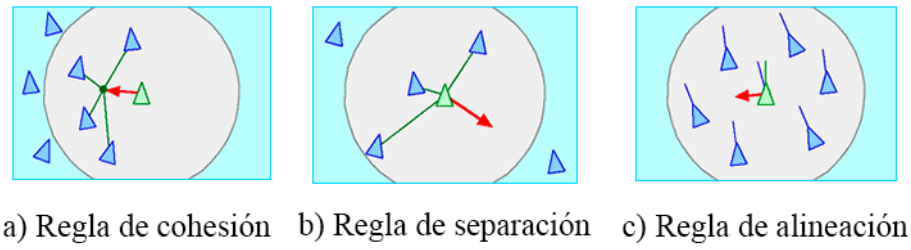

Figura 1.1: El comportamiento de una bandada de pájaros puede ser descompuesto en tres reglas simples

Aplicando esta idea a la robótica, se puede lograr un comportamiento aparentemente complejo mediante la adición de comportamientos simples. Tomando como estandarte el último ejemplo mencionado, las colonias de insectos, surge la **Robótica de Enjambre**, una rama de la robótica centrada en lograr dichos comportamientos emergentes. Este nuevo paradigma aporta beneficios respecto al anterior el cual está centrado en la creación de un único robot de elevada complejidad, como por ejemplo capacidad de paralelización, reutilización, mayor escalabilidad, tolerancia a fallos y eficiencia y un menor coste (estos beneficios se explican con mayor profundidad en la Sección 2: Marco Teórico). El enjambre está formado por multitud de unidades simples, que son más sencillas de fabricar que los robots complejos, además de más económicas. Los comportamientos emergentes permiten al enjambre entero desarrollar tareas más complejas de las que un único integrante sería capaz por separado, y al tener una gran tolerancia a fallos, en caso de estropearse alguna unidad individual estas pueden ser retiradas, reparadas y reinsertadas en el enjambre con una perturbación mínima al comportamiento general de este. Además, el número de unidades del enjambre puede aumentarse y reducirse sin problema, pudiendo dividir el enjambre en sub-enjambres capaces de realizar múltiples tareas en paralelo, aumentando así la eficiencia.

El principal inconveniente de la robótica de enjambre es la compleja programación que requieren las unidades para posibilitar su coordinación. Se requieren múltiples sensores y un sistema de comunicación, ya sea explícito, pasivo o mediante estigmergía, para sincronizar los movimientos de los robots [4]. Los robots deben ser capaces de adaptarse a su entorno, aunque sea de forma sencilla. En este sentido, existen varios paradigmas de programación dependiendo del grado de interacción de los robots con su entorno. En primer lugar, existe la programación en bucle abierto o sin realimentación: este tipo de programación carece de sensores capaces de medir o captar información del exterior, por lo que las máquinas funcionan 'a ciegas', sin reaccionar a sucesos externos. El siguiente paradigma es la programación en blucle cerrado o con realimentación: en este segundo caso, existen sensores que miden magnitudes del entorno y se utiliza esta información para reaccionar a eventos o corregir las acciones de la máquina para mantenerse en unos determinados márgenes de funcionamiento. Este segundo tipo es más complejo pero mucho más eficiente y preciso.

No obstante, todavía se puede ir más allá. Una de las características de la robótica de enjambre es que las unidades deben ser lo más sencillas y económicas posible, por tanto suelen contar con pocos sensores y actuadores en comparación con otras máquinas, lo que limita los comportamientos que cada unidad es capaz de realizar. Este problema se puede solventar mediante el uso de inteligencia artificial, un paradigma de programación que dota a los robots de la capacidad de aprender de la experiencia o de conjuntos de ejemplo. La inteligencia artificial permite que cada unidad pueda mejorar su rendimiento y reaccionar a más sucesos a medida que interactúa con el entorno. En este trabajo, se pretende equipar a cada unidad del enjambre robótico con un algoritmo de inteligencia artificial perteneciente a la sub-categoría del aprendizaje por refuerzo [1].

2. Marco Teórico

Antes de empezar a realizar cualquier solución, conviene primero explorar las herramientas y tecnologías disponibles actualmente. En esta sección se expone el abanico de soluciones ya implementadas a partir de las cuales se ha desarrollado la propia. Existen dos categorías o ramas que conviene investigar: por un lado, los enjambres robóticos; por el otro, las técnicas de computación evolutiva y aprendizaje por refuerzo que se emplearán para programarlos.

2.1. Enjambres robóticos

En esta sección se detallan varios aspectos de los enjambres robóticos, desde el equivalente biológico que da origen a la idea, pasando por su comparación con otras alternativas disponibles, sus ventajas y aplicaciones, y los tipos de robots de enjambre presentes actualmente en el mercado.

2.1.1. Equivalente biológico

La idea de la robótica de enjambre, al igual que muchas otras tecnologías, proviene de la observación directa de la naturaleza. Ya se ha hablado con anterioridad de los fenómenos y comportamientos emergentes, pues la idea teórica detrás de estos proviene de la observación de elementos naturales, como las neuronas, determinados conjuntos de partículas o sustancias, o, más comúnmente, manadas, colonias o enjambres propiamente dichos de animales.

En este sentido, existen multitud de especies que forman grupos de cooperación. En primer lugar, muchos invertebrados como hidrozoos y antozoos (conocidos popularmente como corales) viven en colonias de múltiples individuos fusionados y, en ciertos casos, especializados en funciones concretas (alimentación, defensa, reproducción, etc.). Entre las colonias de seres pluricelulares se encuentran las formadas por insectos eusociales (hormigas, abejas, termitas, avispas, etc.) con diferentes organizaciones jerárquicas y división en castas (Fig. 2.1); o los conjuntos formados por animales como castores, pingüinos, mejillones, etc [5].

En este sentido, el término ”robótica de enjambre” hace referencia a las colonias de insectos. De todas las colonias biológicas, estas son las más similares a lo que se pretende con la robótica de enjambre: son homogéneas, pueden realizar múltiples tareas en paralelo y la inteligencia general del conjunto solo puede extraerse del propio conjunto, pues cada insecto por sí solo apenas puede tomar decisiones y no sobrevive fuera de la colonia.

Figura 2.1: Una colonia de hormigas formando un puente. (Igor Chuxlancev, CC BY 4.0 <https://creativecommons.org/licenses/by/4.0>, via Wikimedia Commons)

2.1.2. Robótica de enjambre frente a otras alternativas

Ya se ha hablado ligeramente de las ventajas de un enjambre robótico frente a un único robot. En este apartado se ahonda en dichas diferencias, así como en la comparación entre enjambres robóticos y otros tipos de sistemas multirobot.

En primer lugar, como ya se ha comentado, un único robot programado para realizar una determinada tarea debe contar con todas las tecnologías necesarias para poder desarrollar el comportamiento deseado sin problemas. Es decir, se precisa que el robot adquiera un comportamiento complejo por sí mismo. Esto requiere soluciones caras y de dificultad exponencial que solo pueden implementarse invirtiendo grandes cantidades de tiempo, recursos e ingenio. Algunos comportamientos son tan complejos que ni siquiera es posible replicarlos con total precisión, como el habla o la gesticulación.

Comportamientos tan extremos no suelen ser necesarios en la mayoría de tareas, no obstante, un enjambre robótico es capaz de replicar comportamientos equivalentes a los presentes en robots complejos con una programación individual mucho más simple. En general, los enjambres robóticos presentan las siguientes ventajas:

- **Paralelización:** un enjambre de robots es capaz de dividirse en subconjuntos para realizar tareas en paralelo, mientras que un único robot solo podría realizar una única tarea a la vez.
- **Reutilización:** los enjambres son flexibles y fácilmente adaptables a nuevas tareas, mientras que los robots individuales requerirían cambios en su software y/o hardware, dificultando la adaptación.
- **Escalabilidad:** los enjambres robóticos pueden ser fácilmente escalados, es decir, se

pueden agregar o retirar robots sin que esto afecte al conjunto completo y sin tener que modificar ningún parámetro. Esto se debe a la estructura descentralizada del enjambre.

- **Robustez:** a diferencia de los robots individuales, un fallo en un miembro de un enjambre no compromete la realización de la tarea: el enjambre puede prescindir de una unidad dañada y continuar funcionando.

- **Bajo coste:** los miembros de un enjambre suelen ser mucho más simples y baratos que sus equivalentes individuales. Incluso en grandes números, un enjambre robótico puede resultar más económico que un robot individual.

- **Eficiencia:** las unidades del enjambre consumen mucha menos energía que los robots individuales equivalentes. Incluso en grandes números, la eficiencia se nota en que las unidades que no son necesarias para la tarea pueden ser desactivadas sin comprometer al resto, por lo que el enjambre puede adaptar su potencia a la tarea, a diferencia de los robots individuales.

2.1.3. Aplicaciones

Los enjambres robóticos, dada su constitución, resultan idóneos para aplicaciones que requieran cubrir grandes terrenos. Por ejemplo, en tareas de búsqueda y rescate, donde varios robots patrullan y se dispersan para explorar grandes áreas. También son útiles en tareas de mapeado, debido a que todos los robots del enjambre pueden contribuir al mapa global, logrando reducir considerablemente el tiempo necesario. En general, cualquier aplicación que requiera una gran superficie o elevada paralelización es apta para un enjambre robótico, como tareas de exploración de zonas escarpadas o peligrosas para el ser humano, la localización de víctimas en tareas de rescate o la gestión logística [18]. A pesar de esto, la robótica de enjambre continúa en estudio, y todavía no está enteramente integrada en la totalidad de la industria. No obstante, algunas empresas empiezan a apostar por esta tecnología (un ejemplo de estas empresas se puede observar en [18]), por lo que en el futuro es posible que se extienda rápidamente a medida que avanza su desarrollo.

Dentro de la robótica de enjambre, se pueden distinguir varios tipos de aplicaciones [1]:

- **Organización espacial:** Tareas que requieran un posicionamiento espacial preciso para ocupar la máxima superficie posible, imitar un patrón o forma concreta o crear una malla compacta de robots para monitorizar o mantener una zona.

- **Exploración:** Los enjambres robóticos resultan especialmente útiles para tareas de exploración y mapeado de entornos desconocidos e intrincados, como lugares afectados por catástrofes naturales o incluso la superficie de otros cuerpos celestes. El hecho de que todos los miembros del enjambre contribuyan y utilicen el mismo mapa acorta el tiempo necesario para la exploración y aumenta la calidad del mapa resultante.

- **Cooperación:** Básicamente tareas paralelizables que se beneficien de múltiples robots coordinándose entre sí para realizar una acción. Por ejemplo, varios robots pequeños

coordinándose para transportar varios objetos más grandes que ellos o para mantener organizado un inventario.

- **Entretenimiento:** Si bien la robótica de enjambre todavía no se ha asentado en la industria, sí que es conocida y apreciada en el ámbito del entretenimiento. Los enjambres bien coordinados resultan en un interesante espectáculo.

2.1.4. Robots más utilizados

Existen multitud de robots diseñados para trabajar como enjambre. La mayoría de robots son similares entre sí: todos suelen ser pequeños robots móviles con sensorización básica, sencillos y baratos. No obstante, entre ellos hay ciertas diferencias que se detallan en el cuadro 2.1.

Nombre	**Actuadores**	**Sensores**	**Comunicación**
Khepera IV	Ruedas	IR, Cam, US, IMUs	WiFi, Bluetooth
e-puck	Ruedas	IR, Mic, Cam, IMU	WiFi, Bluetooth
Alice	Ruedas	IR, Cam	Radio
Jasmine	Ruedas	IR	Infrarroja
Kilobot	Patas vibratorias	IR, LDR	Infrarroja
S-Bot	Ruedas, pinzas	Prox, Cam, Mic, IMU	WiFi
Kobot	Ruedas	IR, Cam	Zigbee
SwarmBot	Ruedas	IR, LDR, Cont, Cam	Infrarroja

Tabla 2.1: Comparativa entre diferentes robots de enjambre *(IR: Infrarrojos, Cam: Cámara, US: Ultrasonidos, IMU: Unidad de Medición Inercial, Mic: Micrófono, LDR: Fotorresistor, Prox: Sensor de Proximidad, Cont: Sensor de Contacto)*

2.2. Computación evolutiva

En esta sección se muestra la otra mitad del marco teórico de este trabajo, relacionada con la computación evolutiva. Se tratan desde sus orígenes, hasta sus variantes más conocidas, pasando por sus diferentes componentes. También se habla acerca de su relación con la robótica y las limitaciones reales que emergen de esta relación.

2.2.1. Orígenes biológicos

Al igual que la robótica de enjambre en sí misma, la computación evolutiva también encuentra sus orígenes en el mundo natural. Concretamente, en la evolución darwiniana. Siendo el mundo un lugar tan complejo y lleno de variables, la evolución ha logrado crear infinidad de especies capaces de sobrevivir en casi cualquier entorno. Su capacidad de optimizar la genética y las características de los seres vivos para adaptarlos a las condiciones del medio es indudable y poderosa. No resulta extraño, pues, que desde hace años se intente imitar este

proceso aplicado a algoritmos artificiales, con la esperanza de encontrar soluciones perfectamente adaptadas a un conjunto de restricciones.

Básicamente, el proceso evolutivo natural puede modelizarse del siguiente modo: Un **entorno** determinado con sus propias características es ocupado por una cierta **población** de individuos compitiendo por la supervivencia y la reproducción. El propio entorno determina la validez (**fitness**) de cada individuo en base al éxito a la hora de cumplir estos objetivos. Esto se traduce en una **probabilidad** de supervivencia y reproducción distinta para cada individuo. Aquellos con mayor probabilidad de sobrevivir transmitirán sus características (**genoma**) a la siguiente generación, mientras que los menos válidos tendrán mayor probabilidad de morir sin descendencia. De este modo, los mejores genes perduran y poco a poco se adaptan al medio. Trasladando este modelo al contexto de la resolución de problemas basada en prueba y error estocástica, la población se compone de posibles soluciones. Su validez (cuán bien resuelven un determinado problema) determina su probabilidad de ser usadas como semillas de nuevas soluciones [6].

2.2.2. Definición y componentes principales

A pesar de que existen múltiples variaciones de algoritmos evolutivos, todos ellos tienen una serie de características en común. Simplificando al máximo, dada una población de individuos, la competición entre ellos por los recursos naturales provocará la selección natural de los más aptos. Esta selección a su vez provoca un aumento en la validez media de la población. Matemáticamente, dada una función objetivo a maximizar, la población se obtiene generando soluciones candidatas aleatorias en el dominio de dicha función. A continuación se aplica una función de calidad que asigna una validez (fitness) a cada solución candidata. En base a esta validez, los mejores candidatos son elegidos para producir la siguiente generación. Esto se consigue aplicando recombinación y mutación. El proceso continúa hasta que se alcanza un determinado límite de validez u otra condición previamente establecida [6].

No todos los tipos de algoritmos evolutivos funcionan del mismo modo, pero en general, los elementos básicos de cualquier algoritmo evolutivo son:

- **Representación del individuo:** El primer paso a la hora de crear un algoritmo de computación evolutiva es encontrar un modo de representar a cada individuo. Esto implica modelar el problema real y sus posibles soluciones de una manera compatible con la computación. En este contexto, se suele distinguir entre las características reales que categorizan a una posible solución como tal (denominadas **fenotipo** de la solución) con su representación abstracta en el espacio computacional (referidas como **genotipo** de la solución). El primer paso para diseñar un algoritmo evolutivo es encontrar un modo de mapear el fenotipo en un genotipo capaz de representarlo.
- **Función de fitness:** La función de fitness es la encargada de definir los requisitos que los individuos deben cumplir para prosperar. Define la dirección en la que un individuo se considera mejor que otro. Matemáticamente, es una función que asigna un valor de calidad o validez a un determinado genotipo.

- **Población:** La población es un conjunto de individuos (es decir, de posibles soluciones), representadas mediante su genotipo. La función de la población como ente es servir como unidad de evolución. Los individuos en sí son invariantes en el tiempo, sin embargo, cada **generación** de la población es diferente: la población es la que se adapta y evoluciona, no los individuos. La característica más importante referente a la población es su **diversidad**, pues esta definirá que tan probable es que se alcance la solución óptima. La diversidad, no obstante, es difícil de conceptualizar y más aún de medir.

- **Mecanismo de selección de progenitores (mating):** Su función principal es decidir qué individuos se encargarán de engendrar la siguiente generación. Debe discriminar a los individuos en base a su fitness y solo permitir la reproducción a aquellos con mayor calidad. No obstante, la selección de progenitores es habitualmente probabilística, es decir, incluso las soluciones con peor calidad tienen una mínima, pero existente, probabilidad de ser elegidas como semillas. De este modo, se evita la **endogamia** del sistema, es decir, una meseta en la que los individuos no evolucionan por ser demasiado similares entre sí.

- **Operadores de variación:** Los operadores de variación son los encargados de generar nuevas soluciones a partir de las anteriores. Estos operadores son divididos dependiendo de a cuantos individuos afectan: los operadores unarios son habitualmente llamados **mutaciones**, mientras que los *n-arios* son habitualmente referidos como **recombinaciones** (Fig. 2.2).

- **Mecanismo de selección de supervivientes:** Este elemento se encarga de mantener el tamaño de la población constante. Funciona de forma similar al mecanismo de elección de progenitores, solo que se aplica en un momento distinto del proceso evolutivo: una vez ya se ha generado la descendencia. Este mecanismo elige las soluciones menos adaptadas y las elimina hasta reducir la población de nuevo a su tamaño original. Su equivalente biológico sería la **selección natural**. A diferencia del mecanismo de selección de progenitores, el mecanismo de selección de supervivientes también puede tener en cuenta la **edad** de las soluciones: puede permitir a algunos progenitores sobrevivir en la siguiente generación, o puede sustituir a todos los progenitores por su correspondiente descendencia.

2.2.3. Variantes más conocidas

No existe una forma única de realizar un algoritmo evolutivo. Aquí se expondrán las variantes principales y más conocidas de esta rama de la computación:

- **Algoritmos genéticos:** Los más conocidos en la rama de computación evolutiva, y una de las variantes más populares y antiguas. Las soluciones se representan mediante genotipos en forma de cadenas binarias. Estas cadenas son habitualmente recombinadas aleatoriamente, eligiendo el mismo bit en dos cadenas distintas e intercambiando ambos valores. La probabilidad de mutación es muy baja, y la función de selección de

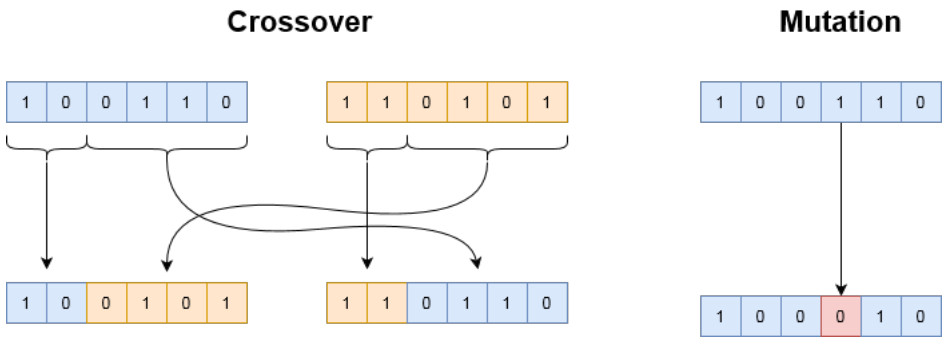

Figura 2.2: Esquema de funcionamiento de los operadores de variación: recombinación (izquierda) y mutación (derecha)

progenitores es proporcional (una especie de "ruleta" en la que el fitness de cada solución representa su probabilidad de ser elegida). A pesar de que estos algoritmos son los más populares, no cuentan con ciertas mejoras introducidas posteriormente, como el elitismo.

- **Estrategias evolutivas:** Las estrategias evolutivas poseen genotipos en forma de vectores de números reales. La variación se realiza añadiendo un número real aleatorio (elegido mediante una distribución Gaussiana) a cada elemento del vector, y conservando la solución si resulta más válida que la progenitora. El principal operador de variación, por tanto, es la mutación, teniendo un tipo de reproducción "asexual". Las estrategias evolutivas, además, fueron pioneras en un nuevo tipo de evolución: la evolución de los propios parámetros del algoritmo. Esto quiere decir, que los genes de cada solución contienen los parámetros del algoritmo que los ha generado, y estos evolucionan junto a la solución en sí.

- **Programación evolutiva:** Su principal objetivo es el desarrollo de sistemas de inteligencia artificial. Se busca un comportamiento adaptativo, por lo que la predicción del entorno es crucial. Por esta razón, cada individuo es representado como una máquina de estados finitos. También es exclusivamente mutágeno, es decir, no emplea recombinación, sino únicamente mutación.

- **Programación genética:** Esta variante novedosa se emplea sobretodo en *machine learning*. Sus individuos son modelos representados en forma de árboles. El objetivo principal de estos sistemas no es encontrar un comportamiento específico ni ajustar una serie de parámetros, sino encontrar el mejor modelo para representar una situación.

- **Learning Classifier Systems (LCS):** Es una combinación entre un clasificador y un algoritmo de aprendizaje. Los individuos son reglas que mapean determinados *inputs* con ciertos *outputs*, o a veces, conjuntos enteros de reglas que constituyen modelos completos. El algoritmo evolutivo evalúa estas reglas y modelos asignándoles un fitness en base a su capacidad de predicción, conociendo el resultado real de una determinada acción y comparando este resultado real con el propuesto por la regla evaluada. Se trata por tanto de un sistema de aprendizaje supervisado. Si los individuos son conjuntos de

reglas, estos se desglosan, evalúan, y categorizan en base al fitness medio de cada una de sus reglas.

- **Evolución diferencial:** La principal característica de esta variante es su novedoso operador de variación: a cada individuo (representado por un vector de números reales) se le suma otro vector resultado de la diferencia escalada de otros dos individuos. Es decir, la magnitud de la mutación está relacionada con la diversidad del conjunto, y todos los individuos son capaces de influirse entre sí. A partir de una lista de individuos X, se genera una nueva lista de mutantes V, los cuales sobrescribirán a sus homólogos en la misma posición en X si resultan ser más aptos.

2.2.4. Robótica evolutiva

A pesar del potencial de la computación evolutiva, no es una rama demasiado presente en el ámbito de la robótica. A pesar de esto, se pueden encontrar ejemplos que integran algoritmos evolutivos en plataformas robotizadas para una gran variedad de funciones: funciones de apoyo, donde el algoritmo no controla directamente a los robots pero modifica su comportamiento para optimizar su tarea [8]; funciones de control, donde el algoritmo modifica el comportamiento del robot hasta lograr el éxito [9]; o incluso algoritmos evolutivos que optimizan el proceso de diseño del propio robot [7]. Los algoritmos evolutivos proveen a los robots de la capacidad de modificar sus controladores para adaptarse a diferentes tareas o para cambiar su comportamiento frente a diversas adversidades o entornos parcialmente desconocidos. La robótica evolutiva, además, permite a la computación evolutiva interactuar en un entorno real, físico, en vez de limitarse a un modelo matemático. La robótica evolutiva, además, posee individuos **activos**, es decir, no solo se adaptan al entorno sino que son capaces de modificarlo, a diferencia de la mayoría de algoritmos evolutivos donde los individuos son **pasivos**. Como tal, no basta con adaptar a los robots a un determinado entorno, se busca que los propios robots sean capaces de modificarlo de una forma concreta, es decir, se busca un **comportamiento** concreto [6].

El comportamiento del robot, por definición, es un ente dinámico que se ve influenciado no solo por el robot en sí, sino por sus interacciones con el entorno. Esto implica que el diseño del robot no es suficiente para garantizar el comportamiento deseado, y que la influencia de los parámetros de diseño en el comportamiento final es débil y ruidosa. El espacio de soluciones también es dinámico y multiobjetivo, a veces incluso contradictorio. No obstante, los algoritmos evolutivos suelen brillar en este tipo de situaciones, debido a que no necesitan una definición exacta de cada situación para funcionar, y se desenvuelven bien en entornos con múltiples variables dinámicas, ruidosas y hasta contradictorias.

Uno de los aspectos claves en la robótica evolutiva es el momento en el que se producen los cambios. La mayoría de robots evolutivos emplean la llamada **evolución offline**, es decir, la evolución se realiza en un simulador para poder monitorizar y corregir el comportamiento en caso de ser erróneo. Una vez que el programa está optimizado, se introduce en el robot y se pone en marcha. El enfoque más atractivo de la robótica evolutiva es la **evolución online**, es decir, la evolución del agente una vez ya es operativo, y mientras realiza su tarea. Este tipo de evolución es paralelo a la evolución natural, constante e incesante, y supone un gran avance

con respecto a la evolución offline en capacidad de adaptación en entornos desconocidos. No obstante, la evolución online viene ligada a problemas difíciles de solucionar actualmente [6]:

- **Ruido:** El ruido es omnipresente en el mundo real. Mínimas variaciones en ciertos parámetros pueden llevar a comportamientos totalmente distintos, y dichas variaciones tienden a ser difíciles de modelar y evitar.
- **Coste:** La evolución online es computacionalmente costosa. Además, tomar diferentes medidas o probar el robot en diferentes entornos puede llevar mucho más tiempo que modificar parámetros en una simulación.
- **Complejidad:** El controlador de un robot únicamente puede funcionar a bajo nivel. Solo puede recibir señales eléctricas de los sensores y controlar los motores de los diferentes actuadores. No obstante, el objetivo principal de la robótica evolutiva es observar un comportamiento general concreto. Relacionar cada posible comportamiento con acciones a bajo nivel requiere de sistemas altamente complejos y un análisis separado del genotipo y fenotipo. La complejidad aumenta aún más cuando varios robots deben comunicarse entre sí.
- **Evaluación de la validez:** En entornos simulados las condiciones son fácilmente modificables y ajustables. No obstante, en un entorno real, las variables pueden cambiar a cada segundo y el entorno puede transformarse en otro totalmente distinto. En este tipo de situación, ¿qué función de fitness sería la ideal? No existe una respuesta clara, ni una función de fitness concreta capaz de representar todo el espectro de posibilidades.
- **Líneas rojas:** Por último, el problema más evidente e importante de la evolución online. En un entorno simulado o en un algoritmo evolutivo clásico, malas soluciones solo pueden llevar a malos resultados. No obstante, en entornos reales, las malas soluciones pueden llevar a la destrucción o la inutilización del robot. Resulta inviable construir un nuevo robot cada vez que el algoritmo falla, así como resulta inviable indicar al algoritmo la lista completa de posibles malas soluciones a evitar.

Por estas razones, la evolución online no parece una opción prometedora hoy en día. No obstante, resulta una opción a considerar a largo plazo, a medida que los diferentes problemas que presenta se vayan solventando. De momento, la evolución offline parece más que suficiente para el ámbito de este trabajo y es la que se empleará en su implementación.

2.3. Elección de la solución

2.3.1. CMA-ES

En las secciones anteriores se han explorado todas las posibilidades que ofrece este campo de estudio. Se han expuesto los tipos de algoritmos evolutivos que existen así como los robots de enjambre más utilizados. Una vez planteadas todas las posibilidades, llega el momento de elegir una de ellas para esta aplicación.

Para este caso, el tipo de algoritmo evolutivo elegido es una estrategia evolutiva. Esta elección parte de la premisa de que se pretende lograr un comportamiento específico, sin más opción que la prueba y error para determinar la secuencia correcta de acciones. En este tipo de problemas, uno de los enfoques más directos es el **aprendizaje por refuerzo**. No obstante, esta técnica requiere del estudio de un determinado gradiente para encaminar al sistema hacia el máximo global de la función a optimizar. Es complicado, además, conocer el origen de cada recompensa debido a que se produce una sola en cada iteración, sin demasiada información relacionada a su origen o al por qué de su valor. Por este motivo, resulta más interesante ignorar análisis alguno del gradiente y centrarse en técnicas más "ciegas". Y una de estas técnicas es la estrategia evolutiva [10].

El uso de estrategias evolutivas aplicadas al aprendizaje por refuerzo no es algo nuevo [19] [20]. En una primera instancia, se propuso el uso de estrategias evolutivas como optimizadores sin gradiente para buscar los parámetros óptimos de políticas que maximizasen una cierta función [19]. Las estrategias evolutivas actúan como cajas negras que permiten ignorar el análisis del gradiente en aplicaciones de aprendizaje por refuerzo, así como optimizar la política que gobierna estas aplicaciones [19] [21]. El aprendizaje por refuerzo y la estrategia evolutiva no son pues antagónicos ni tampoco sinónimos, sino que actúan en simbiosis para adaptar el aprendizaje por refuerzo a campos en los cuales no se conoce (o no se desea involucrar) el gradiente.

Dentro de las propias estrategias evolutivas, existen múltiples planteamientos diferentes. Uno de los más simples se dedicará a muestrear soluciones candidatas de una distribución normal con una media y desviación típica fijas. Una vez que las soluciones candidatas han sido evaluadas y tienen asociado su respectivo valor de fitness, se escoje la mejor solución y se desplaza la media de la distribución a ese punto, de modo que en la siguiente generación las soluciones se muestrearán desde allí. El problema de este planteamiento es que únicamente tiene en cuenta la mejor solución de cada generación, por lo que es posible que el sistema se atasque en un máximo local en problemas complejos. Además, dado que la desviación típica de la distribución no varía, es difícil que las soluciones converjan una vez alcanzado el máximo global.

La solución a estos problemas es un tipo de estrategia evolutiva llamada **Covariance-Matrix Adaptation Evolution Strategy**, a partir de ahora referida como CMA-ES. La potencia de esta estrategia se debe a su capacidad de adaptar la desviación típica de la distribución de cada generación para ampliar o reducir el espacio de búsqueda de forma dinámica. Es decir, puede expandir el espacio de muestreo de las soluciones para cubrir más terreno y así encontrar soluciones más allá de los máximos locales, pero también puede reducir dicho espacio para afinar las soluciones y concentrarlas en un único punto cuando se tiene la certeza de haber alcanzado el máximo global. El resultado es un sistema que explora rápido y es robusto frente a máximos locales, al mismo tiempo que preciso una vez se ha alcanzado el clímax.

La joya de la corona de las CMA-ES, que les posibilita este comportamiento tan suculento, es la matriz de covarianza. Matriz, que no solo modifica la media y la desviación típica de la distribución normal de cada generación, sino que se extiende a todo el espacio de los

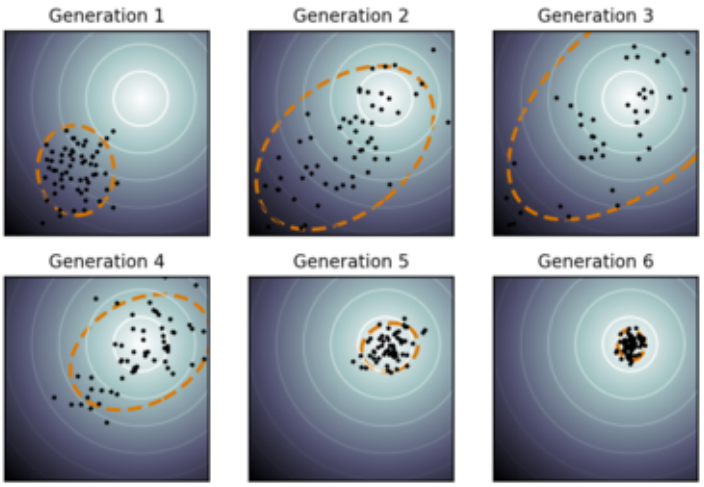

Figura 2.3: Adaptación dinámica del espacio de búsqueda en una CMA-ES

parámetros. Como ejemplo, a continuación se muestran las fórmulas para crear una matriz de covarianza sencilla de dimensiones 2x2, con dos parámetros considerados, las coordenadas de cada solución:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2$$

$$\sigma_y^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_y)^2$$

$$\sigma_{xy}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Las CMA-ES modifican las fórmulas anteriores de un modo inteligente. Primero, escogen las N mejores soluciones de la generación actual y las utilizan para calcular la media de la siguiente generación. A continuación, se emplean esas N mejores soluciones para construir la matriz de covarianza de la siguiente generación, no obstante el valor de la media utilizado en esos casos es el actual, no el siguiente:

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

$$\mu_y^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} y_i$$

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

$$\sigma_y^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (y_i - \mu_y^{(g)})^2$$

$$\sigma_{xy}^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})(y_i - \mu_y^{(g)})$$

Por último, una vez construida la matriz de covarianza, se puede muestrear una nueva generación. De este modo se consigue que el sistema aumente su espacio de muestreo cuando la solución óptima es lejana, y además lo disminuya cuando esté cerca. El único inconveniente de estos algoritmos es su coste $O(N^2)$ a la hora de crear la matriz, no obstante ya se han creado aproximaciones que reducen este coste a $O(N)$ [10]. Esto convierte a las CMA-ES en uno de los algoritmos de optimización sin análisis del gradiente más populares y empleados de todos, y en la solución elegida para esta aplicación.

2.3.2. Funcionamiento de la solución previa

El código desarrollado para esta aplicación en el trabajo previo a este [1] ya contiene una implementación de CMA-ES, por lo que resulta prometedor emplearlo como base para el diseño de los comportamientos deseados. Para esto, es menester conocer a fondo el funcionamiento de este código.

El primer elemento a comprender es el propio algoritmo CMA-ES. Este se ha implementado mediante la librería Gym, una librería ampliamente usada que contiene entornos diseñados para aplicaciones de aprendizaje por refuerzo. Estos entornos cuentan con sus propias entradas y salidas (acciones que modifican el entorno y el nuevo estado del mismo, respectivamente), así como con una función de *fitness* para valorar cuánto se ha acercado cada individuo a la solución general del entorno. Estos entornos pueden emplearse como cajas negras, siendo irrelevante su funcionamiento y proporcionando todas las herramientas necesarias para desarrollar una aplicación CMA-ES [1].

No obstante, los algoritmos CMA-ES se caracterizan por optimizar una determinada función. En este caso, la función que representa a cada robot es una red neuronal. Las redes neuronales resultan muy versátiles debido a su capacidad de aprendizaje, y permiten representar funciones desconocidas a priori (como la función interna de los robots). De este modo,

los parámetros de la red neuronal no son modificados por algoritmos como *Back Propagation*, sino por el algoritmo CMA-ES en sí. Esta red neuronal se ha implementado mediante la librería Keras, con una pequeña modificación del autor [1].

El funcionamiento básico de esta solución comienza con la inicialización de tres elementos fundamentales: el **entorno**, proporcionado por Gym, que cuenta con tantas salidas como entradas tenga la red neuronal y viceversa, y es el encargado de asignar un valor de *fitness* a cada red; la **red neuronal**, que representa a cada individuo de la población y, por tanto, es la función a optimizar; y el **optimizador CMA-ES**, encargado de modificar los parámetros de la red neuronal en base al *fitness* obtenido por la misma. Una vez inicializados estos elementos, se repite el siguiente bucle: primero, el algoritmo CMA-ES genera n soluciones (configuraciones de la red neuronal); segundo, cada configuración se aplica a un miembro de la población (de tamaño n) y se prueba en el entorno, generándose un *fitness* para cada configuración; y tercero, los valores de *fitness* son empleados por el algoritmo CMA-ES para ajustarse y generar n nuevas soluciones.

Para aplicar este algoritmo a la simulación de enjambres de robots, se emplea el simulador *swarmsim*. Este simulador cumple la misma función que el entorno generado por Gym, no obstante, en vez de ser una caja negra que funciona por sí misma, es necesario especificar la función de control y de *reward* (equivalente a la función de *fitness*) manualmente. Modificando esta última es como se modifica el comportamiento del enjambre, por tanto este trabajo se centrará en desarrollar nuevas funciones de *reward* para ampliar el número de comportamientos implementados previamente.

3. Objetivos

Una vez analizado el estado del arte, la elección de objetivos para este trabajo resulta mucho más diáfana. El análisis ha mostrado un gran bagaje teórico ya estudiado previamente en lo que respecta a estrategias de computación evolutiva, así como múltiples robots diseñados para trabajar como enjambre. Debido a la gran concentración de material ya disponible, este trabajo pretende dedicar un porcentaje importante de sus páginas a la investigación de estas soluciones. Este objetivo ha sido cumplido principalmente en la sección de **Marco Teórico** (2), donde se han expuesto las principales variantes, ramas y líneas de investigación actuales sobre computación evolutiva; así como los robots de enjambre más importantes desarrollados en la actualidad.

No obstante, esto no basta como objetivo final. Dado el gran avance que se produjo el año pasado en esta misma materia [1], el objetivo final de este trabajo no es sino ampliar lo ya logrado anteriormente. Este objetivo comprende múltiples subobjetivos: la actualización del sistema del año pasado a una plataforma distinta, y la documentación de las dependencias que requiere y sus diferentes versiones; el uso de la infraestructura generada el año pasado para desarrollar comportamientos de enjambre propiamente dichos, principalmente comportamientos complejos como *flocking*; y la corrección de posibles errores que hayan quedado remanentes en el código y que puedan surgir durante el desarrollo.

El primer subobjetivo pretende solucionar la falta de información con respecto a las dependencias y la instalación del software del año pasado. La solución completa se puede descargar fácilmente, pero aún así es complicado hacerla funcionar sin el conocimiento que aporta este primer subobjetivo. Las librerías y dependencias en este caso concreto a menudo causan problemas de compatibilidad, el simulador empleado necesita de ajustes especiales para ser compatible con la solución, y las versiones necesitan ser muy específicas si se quieren evitar fragmentos de código no definidos. El primer subobjetivo plantea registrar y documentar todo el proceso de instalación y selección de versiones con la intención de actuar como un aglutinante para todos los pedazos sueltos disponibles.

A continuación, el segundo subobjetivo tiene la intención de generar comportamientos complejos de enjambre. El programa desarrollado el año pasado carece de comportamientos complejos de enjambre, por lo que se pretende desarrollar una nueva área del proyecto centrada en los comportamientos. Con este propósito se intenta expandir este campo de conocimiento y aportar aunque sea una mínima contribución para que este trabajo pueda servir de guía para el futuro así como el trabajo del año pasado lo fue para este.

Por último, el tercer subobjetivo trata de solucionar cualquier posible error de funcionamiento que se detecte, depurando y optimizando el código anterior lo máximo posible. Los

programas informáticos siempre causan más errores de los esperados. Algunos de estos errores no se manifiestan hasta mucho después de terminada la tarea, al cambiar la plataforma o incluso la máquina en la que se ejecuta. Es por esto que una corrección final no resulta perjudicial en ningún ámbito, de hecho, contribuye a aumentar la portabilidad y la universalidad del código permitiendo que funcione en un mayor número de entornos.

4. Metodología

En este capítulo se muestran los procesos y experimentos realizados durante el desarrollo del trabajo, empezando por la selección de materiales y continuando con las actividades elegidas para completar cada uno de los subobjetivos mencionados en el capítulo 3. Se detallan las pruebas realizadas, los errores solucionados y los resultados parciales obtenidos de cada fase.

4.1. Lista de materiales empleados

4.1.1. Hardware

- **Controlador *auriga* (Fig. 4.1):** Este controlador desarrollado por Makeblock suele incorporarse en kits completos que contienen todas las piezas necesarias para construir robots prototipo. En este caso, el producto físico no se ha utilizado durante la experimentación, pero el modelo de robot presente en el simulador se basa en este controlador. El modelo viene incluido con el propio simulador, por lo que sería ineficiente crear un nuevo modelo de cero.

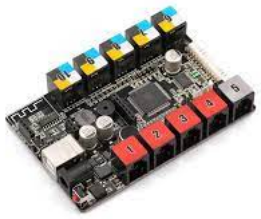

Figura 4.1: Controlador auriga

- **Sensores de ultrasonidos (Fig. 4.2):** Para cada robot del enjambre se emplean tres sensores de ultrasonidos colocados en la parte delantera y en ambos laterales de este. Al igual que la entrada anterior, no se han usado objetos físicos como tal, sino que el comportamiento de este tipo de sensores se ha modelado para la simulación del enjambre.

4.1.2. Software

- **VMWare:** se trata de un software de creación de máquinas virtuales desarrollado por VMware Inc., filial de EMC Corporation, capaz de replicar diferentes sistemas operativos Windows y Linux [13]. Este software se ha empleado para generar el entorno de

Figura 4.2: Sensor de ultrasonidos

Ubuntu 18.04 necesario para desarrollar la aplicación, debido a que múltiples herramientas como el simulador necesitan Ubuntu para funcionar.

- **Visual Studio Code:** es un editor de código fuente desarrollado por Microsoft para Windows, Linux y MacOS, con soporte para depuración y para Git [14]. Se ha empleado para el desarrollo del código del proyecto y el análisis del código del año pasado, debido a que es gratuito y compatible con el lenguaje Python.

- **Overleaf:** se trata de un editor colaborativo, basado en tecnologías en la nube, del lenguaje LaTeX, empleado para escribir y editar documentos científicos [15]. Dada la pulcritud que se puede lograr para cualquier documento, resulta idóneo para escribir este trabajo, además permite cargar la plantilla oficial de la Escuela Politécnica Superior para asegurar que no falta información.

- **Software desarrollado en el trabajo previo:** consiste en el sofware empleado previamente para esta misma aplicación, que se me ha facilitado para la correcta realización de este trabajo. Incluye, entre otros elementos, el código fuente del sistema CMA-ES, la definición de la red neuronal que gobierna a cada robot y varios archivos de testeo.

- **Gym:** es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo [16]. Su principal fortaleza es su inclusión de entornos de aprendizaje por refuerzo completos y listos para funcionar, por lo que resulta muy útil para la depuración fácil y sencilla de algoritmos en entornos simples.

- ***rltrainer:*** se trata de un gran paquete, basado en Gym, que contiene diferentes entornos e implementaciones de algoritmos de aprendizaje por refuerzo, como CMA-ES. Se me facilitó para la correcta ejecución de este trabajo.

- ***swarmsim:*** de manera similar al anterior, *swarmsim* es un simulador diseñado en la Universidad de Alicante, capaz de soportar enjambres robóticos y otros sistemas multirobot. También se me entregó para poder realizar correctamente este trabajo, debido a la escasez de simuladores gratuitos que soporten múltiples robots.

- **Microsoft Excel:** es una hoja de cálculo desarrollada por Microsoft para Windows, MacOS, Android e iOS [17]. Es el software elegido para la realización de las tablas y gráficas que resumen cada experimento, debido a su condición de estándar y su potencia frente a otras opciones.

4.2. Guía de instalación del código del trabajo previo

Como ya se ha explicado, el código realizado previamente para esta misma aplicación ya cuenta con una buena implementación de la solución, con un sistema CMA-ES en el que cada individuo es una red neuronal independiente [1]. Para poder trabajar con esta solución como esqueleto, se ha actualizado y revisado la implementación para asegurar su funcionamiento.

4.2.1. Elección del sistema e instalación de dependencias

El sistema operativo elegido para este trabajo es Ubuntu 18.04, por ser un sistema fiable utilizado en el pasado con buenos resultados. Se ha elegido Python 3.6.9 como plataforma de desarrollo debido a su gran flexibilidad y soporte de múltiples librerías para sistemas de inteligencia artificial. El código disponible cuenta con dos partes: una en la que se prueba el algoritmo CMA-ES de forma ajena al enjambre robótico, y otra en la que se aplica a un enjambre de robots en un simulador. El paquete empleado para desarrollar el algoritmo CMA-ES básico ha sido *rltrainer*, mientras que para la simulación se ha empleado *swarmsim*. Ambos paquetes necesitan dependencias específicas y versiones concretas de las mismas para no incurrir en incompatibilidades.

En el caso de *rltrainer*, la lista de dependencias es la que sigue:

- **Numpy:** Dependencia indispensable para cualquier aplicación. Proporciona soporte para cálculos matriciales necesarios para desarrollar CMA-ES. En este caso se instaló la última versión, a fecha de realización de este documento, la 1.19.5.
- **Matplotlib:** El código del año pasado contiene gráficas y diagramas que sin esta dependencia serían imposibles de mostrar. Esta librería puede exigir la versión 3.7 o superior de Python, negándose a funcionar con la versión 3.6.9, por lo que se recomienda instalarla al principio y sin tener Python 3.7 instalado en el equipo. Se instaló la versión 3.3.4.
- **Tensorflow:** Dependencia principal en lo que a redes neuronales se refiere. Actualmente ya contiene una implementación de Keras, no obstante, el *readme* del propio paquete *rltrainer* recomienda instalar ambas y con versiones muy concretas. En el caso de Tensorflow, la versión 1.9.0.
- **Keras:** Esta dependencia puede causar problemas de compatibilidad con Tensorflow, debido a que las últimas versiones de esta última ya contienen una definición de Keras. Por tanto, se recomienda seguir las instrucciones del *readme* de *rltrainer*, que indican instalar la versión 2.2.2.
- **Ipyparallel:** Necesaria para poder paralelizar la ejecución de la aplicación y así lograr mayor eficiencia. Se instaló la versión 6.3.0, no obstante, se necesita la librería IPython para que la paralelización funcione.

- **IPython:** A pesar de instalar Ipyparallel, la paralelización no funcionó hasta que no se instaló esta dependencia. Concretamente, el comando *ipcluster* no está definido sin ella. Se instalaron la librería principal en su versión 7.16.1, así como los paquetes *ipython[all]*, *ipython[terminal]* y *ipython[parallel]*. Es necesario reiniciar la máquina después de la instalación para actualizar el comando *ipcluster*.
- **Sortedcontainers:** Aporta un tipo de contenedor necesario para el funcionamiento de otras librerías. Se instaló la última versión, a esta fecha, 2.3.0.
- **Gym:** Librería central para el desarrollo de entornos de inteligencia artificial, como redes neuronales o algoritmos CMA-ES. En este caso se empleó la versión 0.15.4.

Con estas dependencias, y a fecha de realización de este documento, no se han producido errores de dependencias en cuanto a *rltrainer* respecta. A continuación, la lista de dependencias para el simulador *swarmsim*:

- **Pymunk:** Esta librería proporciona un motor de físicas para la simulación. Permite crear figuras y formas y asignarles parámetros dinámicos como masa o inercia. Se empleó la versión 6.0.0.
- **Ray:** Esta librería es necesaria para simular rayos y haces (como los de los sensores de los robots) y detectar sus colisiones con otros objetos. Es una librería extensa, por lo que además de la librería en sí (versión 1.3.0) se intalaron los paquetes *ray[default]*, *ray[tune]* y *ray[rllib]*.
- **Dm-tree:** Dependencia necesaria para la construcción de árboles dentro de Ray. Dado que el nombre del paquete es simplemente *tree*, no se debe confundir con la librería Tree. Se instaló la versión 0.1.6.
- **Pandas:** Dependencia empleada para el análisis de datos y *datasets*. Se instaló la versión 1.1.5.

Al igual que con *rltrainer*, con estas dependencias no se han producido a fecha de redacción de este documento errores relacionados con las mismas.

4.2.2. Corrección de errores

A pesar de instalar las dependencias adecuadas, todavía han aparecido algunos errores que ha habido que solucionar. El primero de ellos ha consistido en una súbita desaparición de uno o varios miembros de la población en cada generación. Una lista de tamaño 160 se emplea para contener a los 160 miembros de cada generación del CMA-ES. No obstante, a partir de la generación 40, la lista empieza a reducir su tamaño a 159, perdiendo un miembro de forma súbita. Este error se ha solucionado comprobando que la lista tenga el tamaño correcto en cada generación, y en caso de no ser así, añadiendo copias del último elemento hasta completar el número de individuos correcto.

Una vez solucionado este problema, el simulador dio otro error importante. Resulta que la versión de *swarmsim* empleada en el TFG del año pasado fue modificada por el autor, añadiendo una función adicional necesaria para el correcto funcionamiento del enjambre. En la versión oficial no existía dicha función, por lo que se tuvo que emplear la ingeniería inversa para reconstruirla. El propósito de la función es ayudar a los robots a diferenciar a los demás robots del enjambre de los obstáculos estáticos repartidos por el entorno. En un primer momento se rescató una antigua versión del TFG del año pasado con una primera implementación de la función faltante. Esto resultó en un buen punto de partida, pero todavía faltaba un parámetro de salida del cual no se conocía su naturaleza ni función. Tras un profundo análisis del código, se llegó a la conclusión de que se debía añadir una lista de tres enteros a la función, inicializada a -1, y modificar dicho valor con el índice del robot detectado en caso de cumplirse las condiciones de detección (cada robot contiene un índice entero positivo que sirve como identificador).

A continuación, se resolvió un error con la función de *numpy*, *matmul*, encargada de la multiplicación matricial en cálculos importantes para las redes neuronales en cada robot. En la definición de la red neuronal, las matrices a multiplicar no tenían las dimensiones adecuadas para la multiplicación, difiriendo en una única fila o columna, en un caso muy similar al primero de este apartado. El problema se ha resuelto asegurando que las matrices tienen el tamaño adecuado antes de la multiplicación, y en caso contrario, añadiendo o eliminando las filas o columnas necesarias para garantizarla.

Por último, se ha completado la función de guardado del modelo de la red neuronal en formato *json*, la cual no estaba correctamente implementada.

4.3. Desarrollo de comportamientos

En este punto del trabajo conviene repasar el principio básico del funcionamiento del código, para aclarar las posibles dudas y facilitar la comprensión de los apartados siguientes.

En primer lugar, el funcionamiento básico de este código emplea una estrategia evolutiva, concretamente una **CMA-ES** para simular la evolución de los individuos. La **población** del sistema está formada por un conjunto de robots, cada uno gobernado por una **red neuronal**. El algoritmo CMA-ES debe determinar qué configuración de parámetros de la red neuronal es óptimo para cada tarea. Para ello, se generan n configuraciones de redes diferentes, siendo n el número de individuos de la población. Es importante tener en cuenta que en este paso del sistema todavía no se está produciendo ninguna simulación, por lo que el número de individuos de la población puede diferir del número de robots que finalmente conformarán el enjambre.

Una vez generadas n configuraciones de posibles redes neuronales, estas se prueban en un **entorno** determinado. El entorno determina cuán válidas son estas configuraciones mediante una función de **fitness**. Dependiendo de la librería empleada para generar dicho entorno, este y su función de fitness cuentan con distinta naturaleza. Una vez asignado el valor de fitness de cada configuración, el algoritmo CMA-ES genera n nuevas soluciones en base a las configura-

ciones previas más prometedoras. Esto constituye un episodio, y el ciclo se repite hasta llegar a un número predefinido de episodios (en este caso, se ha definido el número de episodios de cada experimento en 50). Posteriormente, una vez completado este entrenamiento, se genera una **ventana de simulación** donde un enjambre de robots (independiente a los individuos de la población) opera empleando la configuración de red neuronal más prometedora obtenida durante el entrenamiento.

La librería Gym proporciona entornos ya construidos con su propia función de fitness oculta. Estos entornos son invariantes, y generan comportamientos concretos definidos en la propia librería (como por ejemplo, el entorno Car-Pole, que simula una barra sólida colocada en vertical y en equilibrio sobre un carro móvil). Estos entornos no permiten generar comportamientos correspondientes a enjambres robóticos, por lo que Gym se ha empleado únicamente con propósito de testeo del sistema CMA-ES.

Para poder generar comportamientos de enjambre, el **entorno** en sí debe ser redefinido. En este punto es donde entra *swarmsim*, un simulador capaz de proporcionar dichos entornos para robótica de enjambre. Una de las características de los entornos de *swarmsim* es que permiten modificar/definir funciones de fitness propias (llamadas funciones de *reward*, o de recompensa, dentro del simulador). Estas funciones discriminan los buenos comportamientos de los erróneos, definiendo por tanto el comportamiento final que se desea para cada individuo del enjambre. Por tanto, la ampliación del trabajo anterior consiste en el diseño de **nuevas funciones de recompensa (fitness)** para generar nuevos comportamientos. Este proceso de diseño tendrá como principales mecanismos: la prueba y error de diferentes condiciones, de diferentes formas de modificar la recompensa en base a diferentes acciones y de penalizar comportamientos erróneos. En los siguientes apartados se refinarán y analizarán las funciones de recompensa ya existentes y se crearán algunas nuevas.

Como último dato, es conveniente entender el concepto de las **tres leyes de la bandada**, que de ahora en adelante serán empleadas para referenciar diferentes comportamientos. Se trata de tres leyes sencillas que modelizan el comportamiento característico de las bandadas de pájaros sin necesidad de ninguna organización jerárquica ni centralizada (Fig. 1.1) [12]. Los comportamientos de enjambre recreados el año pasado corresponden de algún modo con estas reglas:

- **Regla de Cohesión:** esta regla obliga a la bandada (o al enjambre robótico) a acercarse a los demás miembros, concentrando al enjambre en una región concreta del espacio. Por sí sola, esta regla define el comportamiento de **agregación**.
- **Regla de Separación:** esta regla previene a los miembros del enjambre de colisionar entre ellos o con obstáculos cercanos. Por sí sola, constituye el comportamiento de **evitación de obstáculos** y en combinación con la regla anterior, produce comportamientos de agregación más optimizados y seguros donde los robots se acercan entre sí pero sin llegar a chocarse.
- **Regla de Alineación:** la más difícil de implementar, esta regla obliga a los miembros del enjambre a seguir la misma dirección, determinando una trayectoria general

para todo el conjunto. La combinación de las tres reglas juntas constituye un tipo de comportamiento de ***flocking***.

4.3.1. Separación de comportamientos y creación de un selector

La primera tarea a realizar ha sido la separación de los diferentes comportamientos realizados hasta la fecha. El trabajo del año pasado, al menos la versión disponible para trabajar, únicamente contaba con una función de recompensa, correspondiente al comportamiento de *flocking*. Para mejorar la flexibilidad del programa, se ha creado un nuevo archivo de código llamado *behaviour.py* donde se han implementado las diferentes funciones de recompensa correspondientes a diferentes comportamientos, empezando por los implementados con anterioridad: agregación, agregación combinada con evitación de obstáculos, *flocking*, y una función especial de test, que consiste en una versión del *flocking* sin ninguna modificación, tal cual se implementó el año pasado.

A continuación, para poder aprovechar la presencia de diferentes comportamientos, se ha programado en el mismo fichero *behaviour.py* un selector que permite al usuario elegir el comportamiento que se desea para el enjambre antes del entrenamiento. El comportamiento elegido debe escribirse como una cadena de texto y debe existir en la lista de comportamientos disponibles (mostrada al principio del proceso de selección). Si la cadena introducida no existe, se informa del error y se reinicia el selector, hasta que se introduzca un comportamiento válido. Es entonces cuando la función de recompensa se selecciona automáticamente y el enjambre entrena con ella de forma normal. De este modo se evita tener que modificar la misma función de recompensa cada vez que se desea cambiar de comportamiento.

4.3.2. Pruebas y ajuste de comportamientos simples

Una vez implementado el selector de comportamiento, se han realizado pruebas de ajuste con los comportamientos más simples, empleando como base las hipótesis y funciones de recompensa planteadas el año pasado. Primero se han extraído los parámetros clave de los comportamientos, como la influencia de la evitación de obstáculos en el cálculo de la recompensa final, la cual se ha parametrizado como una variable porcentual cuyo valor representa la importancia de este proceso en el cálculo global de la recompensa (Un valor bajo permite a los robots acercarse más a los obstáculos, mientras que un valor alto los aleja de ellos). También se ha parametrizado el índice mínimo de los robots (el menor identificador posible) y la distancia mínima a la que se considera a dos robots colindantes como tal (Básicamente, la distancia a la que dos robots se consideran agregados).

A continuación, se han realizado pruebas modificando estos parámetros para obtener el mejor resultado posible, entrenando cada solución hasta 50 episodios, guardando la solución en un fichero *json* y probándola mediante el fichero *test_solution.py*. Para cada experimento se ha generado una gráfica que muestra la recompensa promedio obtenida durante cada episodio de entrenamiento. De manera general para todos los experimentos realizados, se pretende maximizar la recompensa. Por esta razón, las gráficas por lo general empezarán en un valor bajo e irán subiendo hasta alcanzar un equilibrio, un valor de recompensa estable

que no varía demasiado de un episodio a otro. Este valor estable se considera pues el máximo obtenible por el experimento, y depende de la configuración de la función de recompensa, por lo que cada experimento puede estabilizarse en valores diferentes. Además de esto, los resultados más prometedores se han grabado en vídeo para mostrar el comportamiento real. Este comportamiento real no tiene por qué corresponderse con la gráfica obtenida, pudiendo ser esta muy prometedora pero generando comportamientos erróneos (este hecho se explica mejor en el capítulo 6).

4.3.2.1. Comportamiento de Agregación

El primer comportamiento ajustado ha sido el comportamiento de agregación puro, es decir, comportamiento de agregación sin tener en cuenta la evitación de obstáculos, correspondiente a la regla de cohesión de la bandada (para más información, consultar la Fig. 4.5). Al igual que en los demás comportamientos, estos experimentos se han entrenado hasta el quincuagésimo episodio, y en cada uno de ellos se han modificado los parámetros más relevantes del algoritmo para observar su influencia. En este caso el único parámetro relevante es el índice mínimo que pueden tener los robots. Este valor debe mantenerse constante a lo largo de los experimentos pues constituye un parámetro de diseño del modelo de los robots *auriga*. Lo que quiere decir que una vez determinado su valor óptimo, no será necesario volver a cambiarlo, y permanecerá con el mismo valor en los siguientes experimentos. La razón de considerar este parámetro en vez de mantener el valor por defecto asignado en el trabajo previo es que la función que hace uso de este tuvo que ser reconstruida de forma artificial sin conocimiento de su funcionamiento concreto ni del código original. Al haber modificado esta función, es posible que el valor óptimo del parámetro haya cambiado, por lo que se considera para este experimento.

Dicho esto, el primer experimento se ha realizado considerando que el índice de los robots del enjambre empieza en 1, valor por defecto asignado en el trabajo previo, obteniendo los siguientes resultados (Fig. 4.3):

Figura 4.3: Resultados obtenidos en el primer experimento de Agregación

Como se puede observar, la recompensa media asciende ligeramente hasta estabilizarse sobre el episodio 30. Aún así se producen oscilaciones más allá de este punto, pero la tendencia ascendente desaparece. La recompensa a nivel absoluto no es muy elevada y a nivel relativo no existe demasiada pendiente durante la ascensión, cuyo rango comprende menos de 5 unidades. Esto indica que el comportamiento en sí no resulta demasiado variable, no ofrece demasiadas posibilidades explotables por el aprendizaje.

A continuación, se ha realizado un segundo experimento considerando que el índice de los robots empieza en 0 (Fig. 4.4). No tiene sentido probar más valores, pues el análisis del código demuestra que los índices de los robots se asignan de forma ascendente al ser incluidos en la escena, y estos son enteros positivos.

Figura 4.4: Resultados obtenidos en el segundo experimento de Agregación

Como ya se ha teorizado, ambos resultados son muy similares, y presentan un rango muy bajo de variación en la recompensa promedio. Si se observa el vídeo correspondiente, se llega a una conclusión paradójica, donde el enjambre apenas parece agregarse, o la agregación se disuelve al poco tiempo de lograrse. Esta situación puede tener una explicación simple: el comportamiento es demasiado simple. La función de recompensa es tan sencilla que no da pie a ningún tipo de aprendizaje. No da pie a que los robots actúen de diferentes maneras. Esto puede explicar el ínfimo rango de variación en las recompensas, que son prácticamente las mismas siempre. Además, los robots carecen de capacidad para evitar obstáculos por lo que son muy fácilmente bloqueados por estos. Los robots bloqueados detrás de obstáculos actúan como atractores para otros robots, alejándolos de posiciones previamente prometedoras. Los robots carecen del concepto de zona, por lo que alejarse mínimamente unos de otros implica la ruptura de la formación.

En cuanto al análisis de la influencia del parámetro, no existe una diferencia importante en ambos experimentos que decante el valor del parámetro. No obstante, el segundo experimento se muestra más rápido, alcanzando valores cercanos al valor final en menos episodios que el primero. En el primer experimento, se sobrepasa la recompensa de 6 en el episodio 23, mientras que en el segundo experimento se alcanza este valor de recompensa en el episodio

16. Es por esta razón que de ahora en adelante se considerará el valor 0 como índice inicial del enjambre de robots. El video mostrando el comportamiento del último experimento se puede ver en el siguiente enlace: agregación pura.

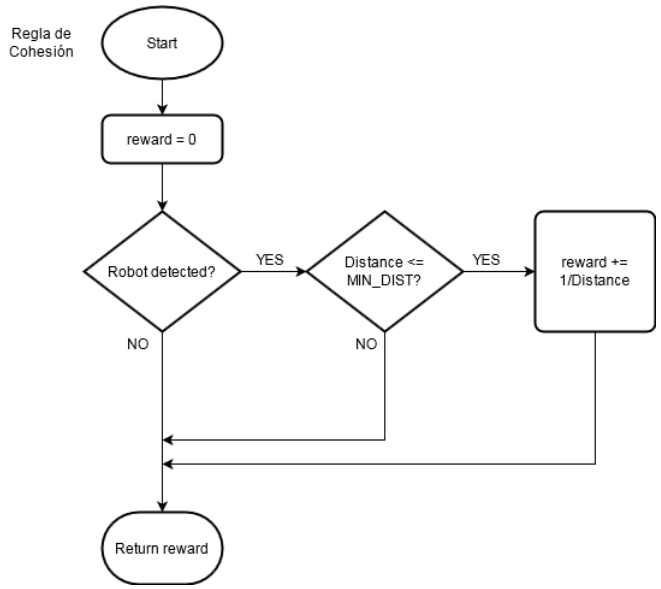

Figura 4.5: Funcionamiento básico de la regla de Cohesión

4.3.2.2. Comportamiento de Agregación y Evitación de Obstáculos

El primer comportamiento de Agregación se ha realizado como toma de contacto con el software y para la determinación del índice mínimo que se puede asignar a los robots. Una vez cumplido este objetivo, se ha añadido la segunda regla de la bandada: la regla de separación (más información en la Fig. 4.8). Como se ve en el vídeo correspondiente, la adición de esta regla modifica de forma positiva el resultado final, observándose un comportamiento de agregación más cercano al óptimo. En este comportamiento se ha estudiado el equilibrio entre ambas reglas. Un exceso de separación causa que los robots no lleguen a formar un grupo compacto, mientras que un exceso de cohesión puede provocar choques entre robots y obstáculos. Para este menester se ha definido un parámetro porcentual que expresa el nivel de influencia de la regla de separación frente a la regla de cohesión. Un valor bajo implica que la importancia de la separación es escasa, por lo que los robots tienden a aproximarse. Un valor alto, por el contrario, otorga más peso a la separación en el cálculo final de la recompensa, por lo que los robots tienden a separase. El primer experimento realizado ha mantenido el valor original de este parámetro, asignado el año pasado: un 50% (Fig. 4.6).

En la figura 4.6 se muestra un comportamiento estable y con más rango de variación que en el caso de la agregación pura. En este caso la recompensa empieza en un valor negativo y asciende hasta 0. Esto no hace este comportamiento mejor o peor que el anterior, sino que lo verdaderamente importante es la pendiente, la presencia de ascensión. Esta ascensión indica

Figura 4.6: Resultados obtenidos en el primer experimento de Agregación y Evitación de Obstáculos

que la recompensa está siendo maximizada hasta alcanzar un valor estable, el valor absoluto de cada recompensa depende de la implementación particular de la función correspondiente y no está relacionado con la progresión, que es lo que realmente interesa. En este caso la progresión comprende un rango mayor que en el experimento anterior, lo que indica un aprendizaje mayor. A pesar de esto, como ya se ha explicado, el comportamiento capturado en video puede aun así ser erróneo. En este primer experimento, los robots no llegan a acercarse todo lo que se debería.

A continuación, se ha realizado un segundo experimento, modificando el parámetro en cuestión hasta un 25% (Fig. 4.7).

Figura 4.7: Resultados obtenidos en el segundo experimento de Agregación y Evitación de Obstáculos

A simple vista se observa cómo el segundo experimento es ligeramente más rápido que el primero. Además, la recompensa en el primer caso no llega a superar -45 puntos, mientras que en el segundo caso llega alrededor de -25. Esta subida de recompensa se ve además res-

paldada por el vídeo correspondiente a este comportamiento (visible en el siguiente enlace: agregación y evitación de obstáculos), donde los robots logran formar un grupo cohesionado. Se puede ver cómo los robots distantes empiezan a moverse hacia el grupo principal tan pronto como lo detectan, no obstante es posible que ciertos robots queden aislados debido a los obstáculos del entorno, incapaces de detectar otros robots. Los robots que sí logran agregarse, por otra parte, son capaces de rectificar y volver al grupo si se alejan demasiado de este.

A diferencia del primer comportamiento de agregación, donde los robots fácilmente pueden perder de vista a los demás o chocar contra obstáculos y carecen del concepto de **zona**, en este comportamiento los robots aprenden a mantenerse cerca unos de otros incluso aunque no se detecten directamente, fomentando la toma constante de medidas sensoriales y el escaneo frecuente del entorno para asegurar el mantenerse en el grupo sin salir de él.

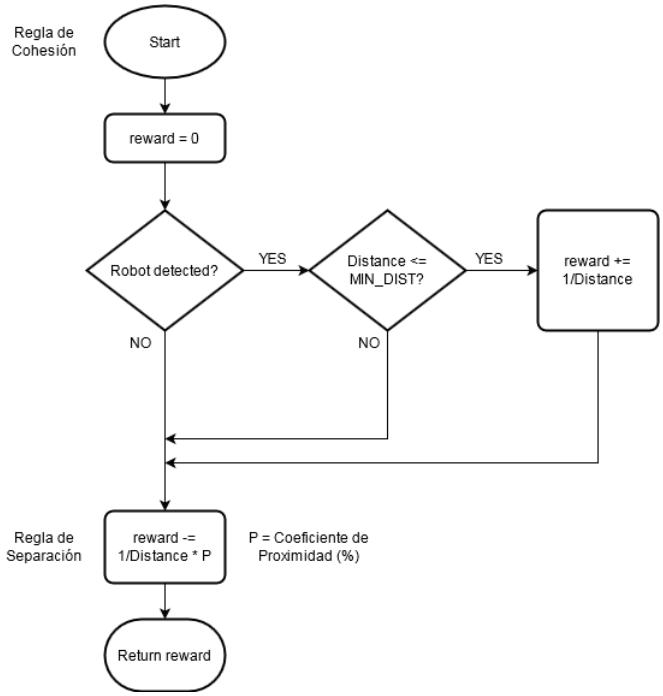

Figura 4.8: Funcionamiento básico de las reglas de Cohesión y Separación

4.3.3. Comportamiento de Flocking

El siguiente paso lógico es añadir la última regla de las tres: la regla de alineación, que permite al enjambre robótico moverse coordinadamente en una dirección concreta. Esta regla es la más difícil de implementar a nivel conceptual, haciendo del comportamiento de *flocking* el más complicado de implementar de todos los comportamientos relacionados con las reglas de la bandada.

4.3.3.1. Prueba preliminar

En el trabajo previo, este comportamiento está prácticamente ausente, siendo la única aportación una hipótesis de función de recompensa consistente en sumar un pequeño valor adicional a los robots que sigan una dirección predeterminada (información más detallada en la Fig. 4.10). En este caso, seguir hacia delante. El siguiente experimento consiste pues en una prueba de esta hipótesis, obteniendo los siguientes resultados (Fig. 4.9):

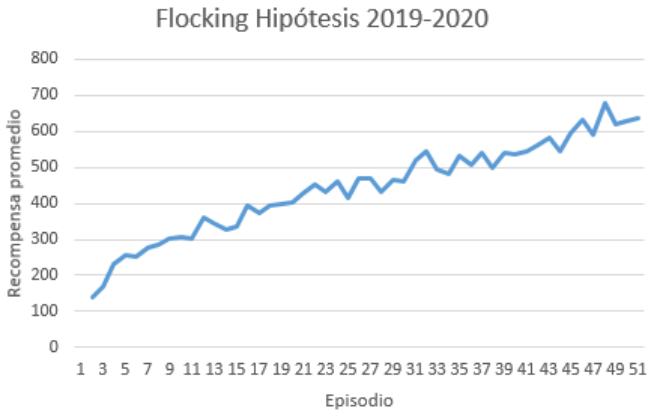

Figura 4.9: Resultados obtenidos según la hipótesis planteada en el trabajo previo

Si se observa la gráfica, el primer detalle en llamar la atención es la ausencia de estabilidad. Con anterioridad se ha mencionado que la presencia de un valor estable indica la presencia de un máximo global en la función, por lo que un comportamiento inestable como este muestra que la función de recompensa no es capaz de alcanzar ningún máximo. Esto puede deberse a un fallo de diseño en esta función o a una falta de entrenamiento. En este caso, no obstante, la función de recompensa añade de forma arbitraria recompensa adicional sin condiciones específicas (si el robot se mueve hacia delante, se añade una pequeña recompensa), por lo que la recompensa promedio aumenta sin llegar a la estabilidad.

El resultado final al ver la simulación del enjambre, acompañando a la gráfica, deja mucho que desear (simulación disponible en: hipótesis del año pasado). El hecho de premiar una acción determinada (avanzar hacia delante) la convierte en la única prometedora: las demás acciones se ignoran porque no añaden recompensa de forma inmediata y a la larga, las reglas de cohesión y separación no son suficientes para frenar este aumento. Además, la dirección de desplazamiento depende de la orientación inicial (aleatoria) de cada robot, por lo que no constituye una dirección uniforme para todo el enjambre. Los robots se mueven de forma caótica sin la apariencia característica de bandada que se espera en este tipo de comportamiento. Estos resultados dependen de la implementación en sí de la regla de alineación más que del ajuste de parámetros, por lo que se debe rehacer esta regla de una forma más precisa.

Más allá de una breve hipótesis, el trabajo previo no proporciona más información o desarrollo del comportamiento de *flocking*. El comportamiento ideal está lejos y es objetivo

primordial de este trabajo el desarrollarlo y brindarle la investigación y experimentos que se merece. Como paso preventivo y a modo de centrarse únicamente en el enjambre, se han eliminado los obstáculos del entorno de simulación para favorecer la interacción robot-robot y así poder depurar el comportamiento más eficientemente.

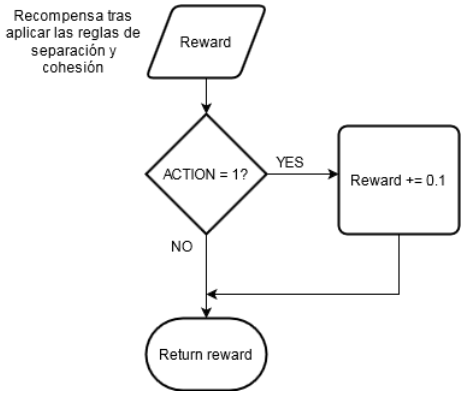

Figura 4.10: Funcionamiento básico de la regla de Alineación planteada en el trabajo previo

4.3.3.2. Primera versión

Uno de los problemas más evidentes de premiar una dirección concreta es que no se tiene en cuenta la orientación de los robots, por lo que el movimiento rectilíneo no logra la coordinación. Con tal de resolver este problema, se ha modificado la función de recompensa del comportamiento de *flocking*, añadiendo como parámetros el robot actual que calcula la recompensa y el vector de orientaciones detectadas en los demás robots. Estos parámetros permiten calcular la diferencia entre la orientación actual del robot en cuestión y la orientación de los robots que ve. Este principio ha sido empleado para programar una nueva regla de alineación que otorga una recompensa fija de 100 a los robots cuya orientación sea la misma que la del primer robot detectado, considerando un pequeño umbral (más información en la Fig. 4.12). El resultado de este experimento puede verse en: Experimento 1, y los resultados del mismo se encuentran en la figura 4.11.

Como se puede observar, los resultados son altamente inestables, oscilando en gran medida sin una recompensa final clara, además de ser recompensas notablemente más altas que en el resto de experimentos. Esto es debido al aumento constante de recompensa cada vez que robots colindantes alcanzan orientaciones parecidas, evento que puede suceder varias veces durante el entrenamiento de cada episodio. Si se observa el video, se ve que los robots tienen un comportamiento errático, centrado en obtener la misma orientación que sus colindantes sin preocuparse por la agregación. Esto genera un comportamiento erróneo pero curioso donde los robots se agrupan en filas pequeñas de dos o tres robots cada una, y avanzan así hasta chocar con una pared. La naturaleza inestable y variable de la recompensa impide conocer de antemano cómo se comportará el enjambre, pues en un episodio la recompensa puede ser astronómica y en el siguiente descender en varios miles. El enjambre también puede decidir

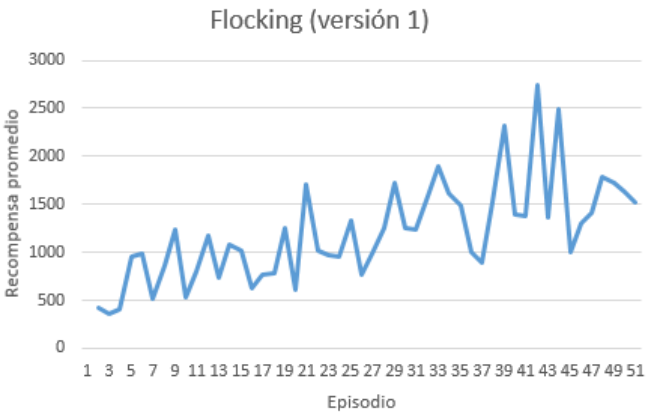

Figura 4.11: Resultados obtenidos en el primer experimento de *flocking*

enfocar su comportamiento únicamente a la regla de alineación, ignorando las otras dos debido a que la primera genera ya de por sí suficiente recompensa como para opacar a las demás reglas. Además, la elección de un umbral fijo para determinar si dos robots siguen la misma orientación es también delicado, y requeriría de múltiples experimentos largos y tediosos para determinarse experimentalmente.

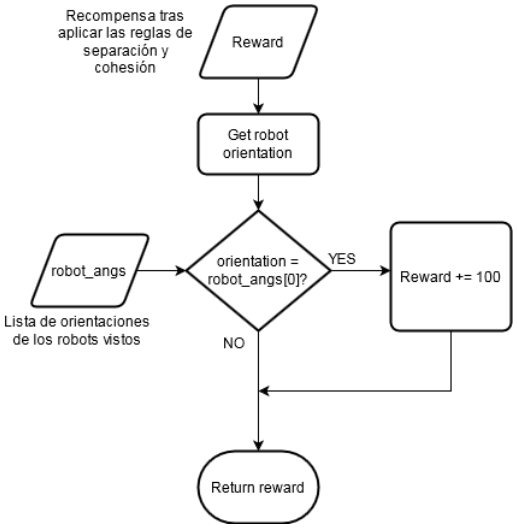

Figura 4.12: Funcionamiento básico de la regla de Alineación (versión 1)

4.3.3.3. Segunda versión

Es por las razones mencionadas en la subsección anterior que para el siguiente experimento, se ha sustituido la recompensa fija por una recompensa variable, inversamente proporcional a la diferencia de orientación entre el robot en cuestión y el primer robot detectado (dia-

grama de flujo en la Fig. 4.14). En cuanto a las reglas de cohesión y separación, se sigue el comportamiento de agregación y evitación de obstáculos de forma idéntica al mostrado en su respectivo experimento, siendo iguales todos los parámetros. La modificación introducida ha generado los resultados observables en la figura 4.13 y en el video: Experimento 2.

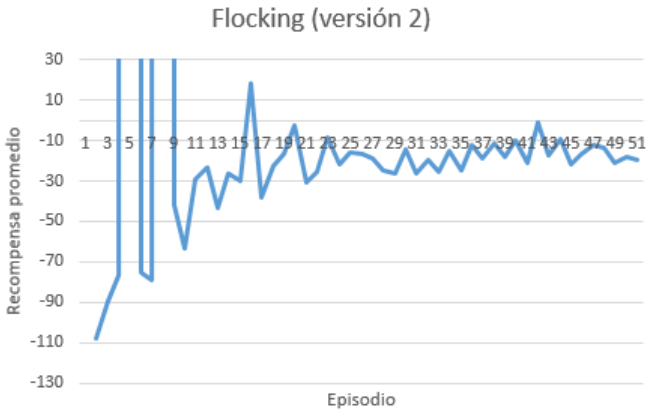

Figura 4.13: Resultados obtenidos en el segundo experimento de *flocking*

Si se observa la gráfica, se puede observar que el comportamiento en general se vuelve más estable, acercándose a un valor entre -10 y -20. El hecho de que la recompensa sea proporcional en vez de sumar un valor fijo impide que oscile de manera tan impredecible como en el experimento anterior. A pesar de esto, se pueden observar extraños picos en los primeros episodios. Estos picos de valor descomunal (que se han recortado para poder observar el resto del gráfico) se deben a que la recompensa es inversamente proporcional a una diferencia. Se ha tenido en cuenta la situación en la que la diferencia de orientación sea 0 para evitar un caso de división por 0, no obstante cuando la diferencia es mínima, el valor de la recompensa puede alcanzar valores inconmensurables.

Dicho esto, analizando el video de comportamiento se puede observar como los robots no parecen seguir un comportamiento específico al principio, simplemente deambulando, para después terminar generando filas de robots similares a las vistas en el experimento anterior y distribuyéndose a lo largo de los límites del entorno. La principal conclusión a extraer de esto es que el comportamiento de agregación está siendo ignorado por el enjambre, pues al final de la simulación los robots no podrían estar más separados. Resulta evidente que ha de aumentarse el peso de la regla de cohesión para obligar a los robots a permancer en el grupo en vez de dispersarse.

4.3.3.4. Tercera versión

Existen múltiples modos de aumentar el peso de la regla de cohesión respecto a las otras dos. Un movimiento directo sería aumentar la recompensa otorgada por esta regla multiplicándola por un factor, y posteriormente ajustar este factor mediante experimentos sucesivos.

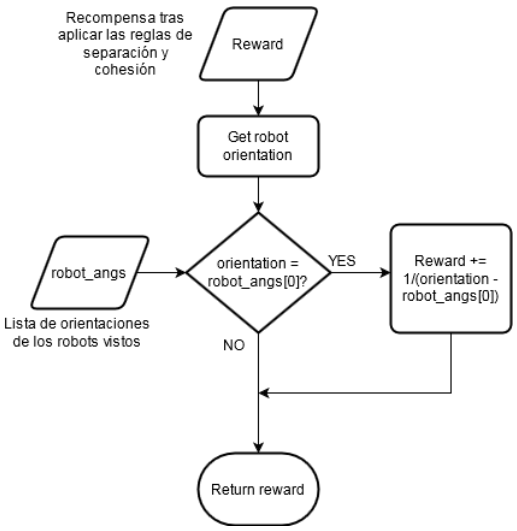

Figura 4.14: Funcionamiento básico de la regla de Alineación (versión 2)

No obstante, esto requeriría una cantidad de tiempo elevada y en general resultaría ineficiente. Es por eso que la solución propuesta es otra. Se trata de restringir la aplicación de la regla de alineación hasta que el enjambre esté agregado. Esto resulta más complejo de decir que de hacer debido a que los robots no cuentan con un sistema de comunicación explícita que permita conocer el estado global del enjambre, sino que cada robot solo puede saber si él se encuentra agregado al enjambre (cerca de otro robot) o no. Este problema es la causa de que de vez en cuando los robots se agurpen en múltiples *clusters* en vez de en un solo grupo, y dificulta la elección de un momento concreto para empezar a aplicar la regla de alineación. Por el momento, se ha añadido un nuevo parámetro al algoritmo, denominado **Distancia de *Flocking*** y predeterminado a 5. Si un robot se encuentra a menos de esta distancia de otro robot, se considerará a sí mismo agregado al enjambre, y por tanto pasará a aplicar la regla de alineación (para ampliar, véase Fig. 4.16). Con esta medida se pretende diferenciar esta regla de las reglas de cohesión y separación, que se aplican si el robot se encuentra lejos de los demás. Esta modificación ha generado el siguiente comportamiento (véase Fig. 4.15 y Experimento 3).

Observando la gráfica de la Fig. 4.15, se puede observar que la recompensa se estabiliza de nuevo en un valor cercano a -20 aunque ligeramente inferior que en el experimento anterior. De nuevo se pueden observar picos recortados que corresponden a diferencias de orientación mínimas. En general, el comportamiento del enjambre parece ser muy similar al obtenido en el segundo experimento y, observando el vídeo del resultado, todavía se reafirma este enunciado. El enjambre parece predispuesto a agregarse en momentos concretos del inicio de la simulación, no obstante, a medida que pasan los segundos los robots abandonan los grupos para acercarse a los bordes del entorno, de forma idéntica a lo ocurrido en el anterior experimento. Esto parece demostrar que la medida tomada realmente no es la solución correcta.

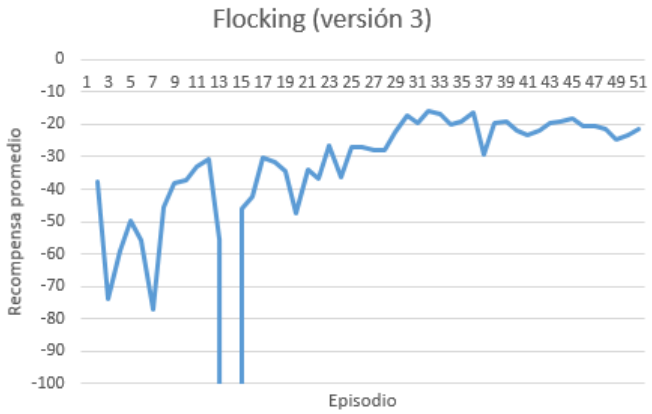

Figura 4.15: Resultados obtenidos en el tercer experimento de *flocking*

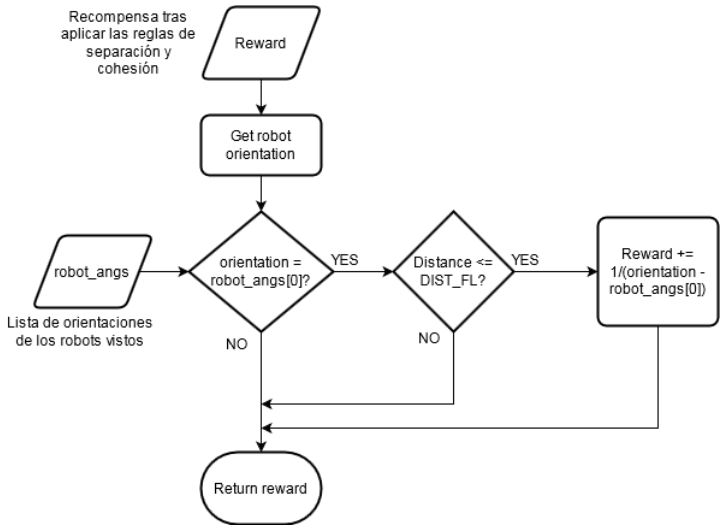

Figura 4.16: Funcionamiento básico de la regla de Alineación (versión 3)

4.3.3.5. Cuarta versión

En este punto, el hecho de que los robots prefiriesen acercarse a los bordes del entorno en vez de agregarse, contradiciendo las reglas que supuestamente proporcionan más recompensa, resulta realmente confuso. No obstante, esta confusión momentánea es el caldo de cultivo para un descubrimiento nimio a la par que crucial para este y otros futuros comportamientos. Un dato que hasta este momento había pasado desapercibido debido a la presencia de obstáculos en las etapas iniciales de esta experimentación. Resulta que debido a la ausencia de obstáculos no se había considerado importante la distinción entre robots y obstáculos, y la implementación de la regla de alineación se planificó inicialmente sin esta comprobación. No obstante, el propio borde del entorno también es detectado por los robots como un obstáculo, por tanto, en ausencia de distinción, los robots tratan de acercarse a los bordes, ignorando a los demás robots porque estos permiten una alineación casi perfecta, con una diferencia de

orientación cercana a 0. Esto puede explicar los picos de recompensa vistos con antelación, además de explicar la tendencia anómala del enjambre. Para probar esta hipótesis, se modificó la regla de alineación para aplicarse únicamente cuando los robots se encuentren cercanos a otros robots, no a los bordes del entorno (véase Fig. 4.18). Este cambio ha quedado registrado en: Experimento 4, así como en la figura 4.17.

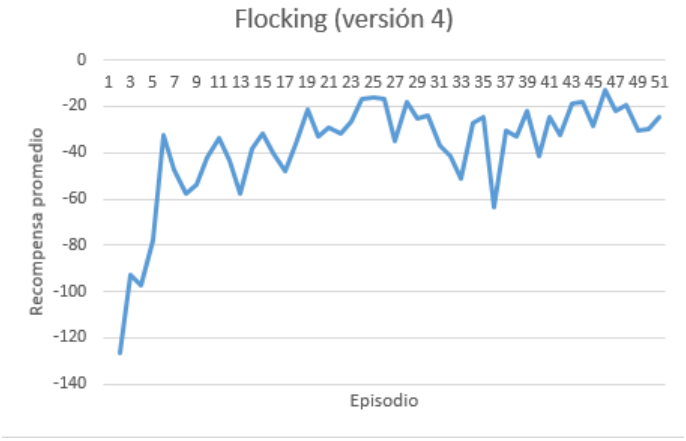

Figura 4.17: Resultados obtenidos en el cuarto experimento de *flocking*

Un vistazo rápido a la Fig. 4.17 muestra que los picos de recompensa desaparecen tal y como la hipótesis predijo. Si se observa el video, se puede observar cómo el enjambre ahora forma un par de *clusters* bien diferenciados, para poco a poco converger en un único grupo. A lo largo de toda la simulación, ningún robot se atasca contra el borde del entorno como en los experimentos anteriores. Esto proporciona oportunidades hasta ahora veladas de observar el funcionamiento de la regla de alineación. En el tiempo en el que los robots forman agrupaciones, algunos de ellos ya están aplicando la regla de alineación, copiando la orientación del primer robot que ven. Esto tiende a alargar los agrupamientos y a colocar a los robots en fila, lo cual supone uno de los tipos de *Flocking* más comunes.

A pesar de esto, los robots pueden disolver la formación fácilmente al momento de observar un nuevo robot moviéndose en una dirección diferente. El primer robot en verlo tratará de copiar su orientación, y este a su vez provocará cambios en la orientación de todos los que le vean. Los cambios de orientación también ocasionan que el primer robot visto (el que ocupa el índice 0 del vector de orientaciones de los demás robots) cambie debido a la rotación del robot observador, modificando aún más la orientación final de este. En general, el sistema de copiar la orientación del primer robot visto puede producir resultados más o menos satisfactorios en ciertas configuraciones colectivas derivadas del azar, no obstante, no resulta robusto y es incapaz de considerar a todos los robots del enjambre a la vez.

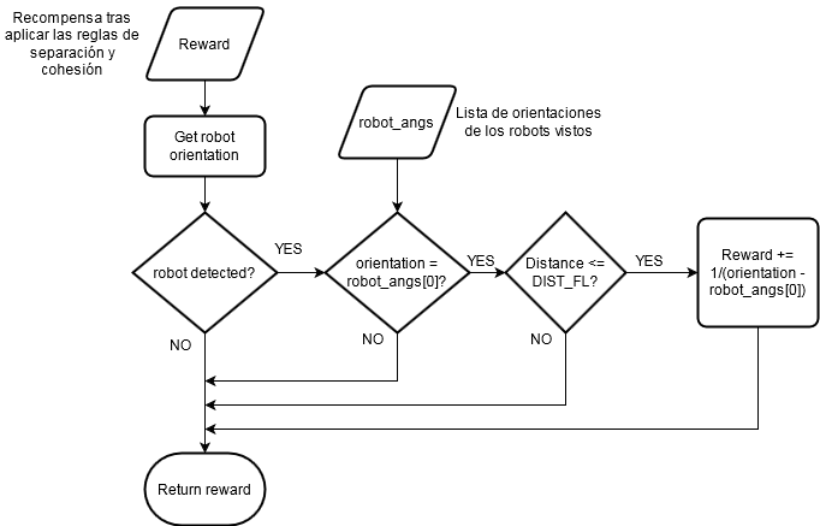

Figura 4.18: Funcionamiento básico de la regla de Alineación (versión 4)

4.3.3.6. Quinta versión

Por las razones anteriormente expuestas, el objetivo de los siguientes experimentos es tratar de producir un **consenso** entre varios robots del enjambre, eligiendo una orientación común que se aplica al robot observador. El primer intento de consenso tiene en cuenta la media aritmética de las orientaciones de todos los robots vistos, siendo esta media la orientación que el robot observador obtiene como referente (diagrama de flujo disponible en Fig. 4.20). Esta aproximación ha proporcionado los siguientes resultados (Fig. 4.19, Experimento 5).

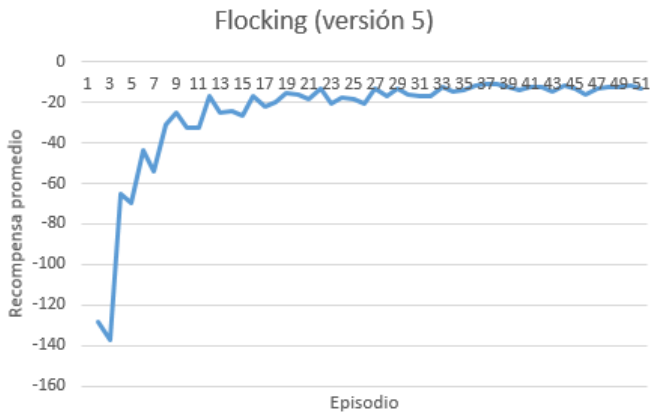

Figura 4.19: Resultados obtenidos en el quinto experimento de *flocking*

Este quinto experimento muestra una curva de recompensa muy prometedora: sin demasiado ruido, rápida y estable, se puede observar cómo la recompensa apenas varía en los últimos 20 episodios. No obstante, si se observa el video de comportamiento, el enjambre apenas se

agrupa, manteniendo a los robots en grupos pequeños y separados que nunca llegan a formar una sola bandada. Hay que recordar que la división entre la regla de alineación y las demás mediante distancia todavía persiste, así como la comporbación de que los objetos observados sean realmente robots y no obstáculos. Por estas razones, el comportamiento descrito por el enjambre resulta anómalo.

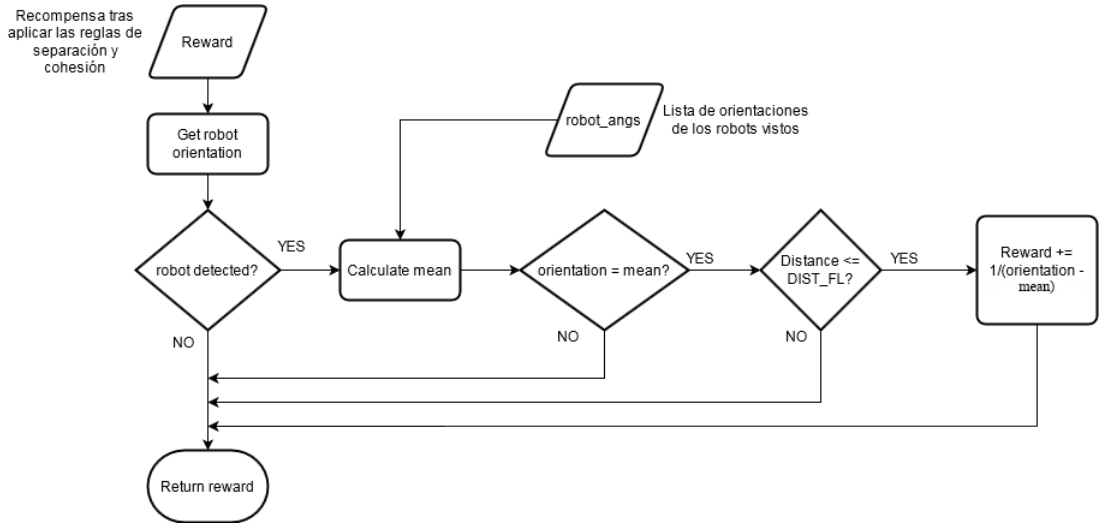

Figura 4.20: Funcionamiento básico de la regla de Alineación (versión 5)

4.3.3.7. Sexta versión

La explicación más loable al resultado del quinto experimento es que el consenso produce demasiada inestabilidad en la recompensa y por tanto los robots aprenden a ignorarlo, es decir, no se acercan lo suficiente porque aplicar la regla de alineación supone recompensas no prometedoras. El hecho de que el consenso elegido sea la media aritmética hace que cualquier robot pueda influir en ella. Otro dato a tener en cuenta es que la acción de no moverse en absoluto no existe, los robots avanzarán hacia delante, hacia atrás o girarán sobre sí mismos siempre, sin poder detenerse por completo. Esto causa que múltiples robots girando sobre sí mismos alteren la media del consenso a un nivel equiparable a un sistema caótico, forzando a los robots a no acercarse entre sí con tal de evitar tal destino. Esta es la única razón que parece explicar la involución del enjambre (de formar satisfactoriamente *clusters* en el cuarto experimento a no formar ninguno en el quinto), pues la función de consenso es el único cambio realizado (no se han modificado las reglas de cohesión ni de separación, ni ningún parámetro adicional).

En vista de lo ocurrido, se necesita un sistema de consenso que no dependa tanto de la orientación instantánea de los robots, al mismo tiempo que permita al enjambre cambiar de dirección y adaptarse al entorno. Tratando de encontrar este equilibrio, se propone un nuevo sistema que en vez de tener en cuenta la media, tiene en cuenta la **moda** de las orientaciones

de los robots vistos, es decir, el robot observador copiará la orientación que más se repita entre todos los robots que vea.

Este nuevo sistema ha requerido una nueva función encargada de calcular el valor del consenso. Primero, las orientaciones de los robots participantes en el consenso se aproximan a 8 direcciones cardinales correspondientes a los ángulos de 0º, 45º, 90º, 135º, 180º, 225º, 270º y 315º, generando un nuevo vector de igual tamaño que contiene las orientaciones aproximadas. A continuación, se calcula la moda de dicho vector [11], analizando los posibles casos en los que la moda es imposible de calcular. Si la moda es viable, esa será la orientación que el robot observador tomará como referente. Si la media no existe, se elegirá una dirección cardinal al azar y se pospondrá el consenso (para más información, consultar la Fig. 4.22).

Este sistema ha generado los resultados siguientes (Fig. 4.21, Experimento 6).

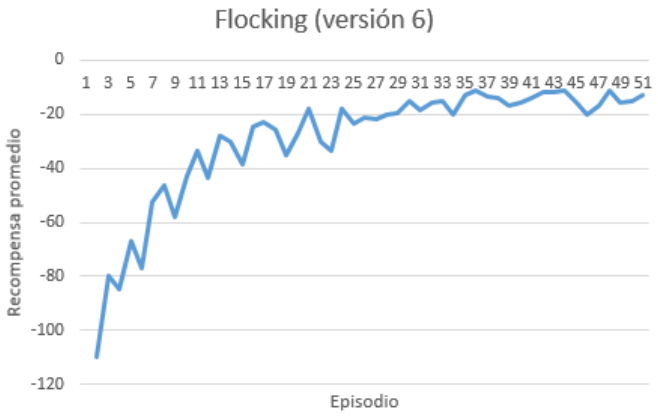

Figura 4.21: Resultados obtenidos en el sexto experimento de *flocking*

A simple vista, la curva representada corresponde a la del quinto experimento, pero con más cantidad de ruido y oscilaciones y un poco más lenta. Aún así, el video de comportamiento muestra una mayor intención de agregación, formando *clusters* de forma recurrente durante la simulación. El sistema de consenso también se observa en ciertos puntos colocando los robots en hileras o generando pequeños grupos que se desplazan en la misma dirección.

A pesar de esto, todavía no se logra un comportamiento de bandada natural. El principal problema es que algunos robots aplican su regla de alineación antes que otros, por lo que su orientación queda fijada y (recordemos que los robots no pueden quedarse totalmente quietos) con la orientación constante lo único que pueden hacer los robots es avanzar. Dado que los robots no aplican la alineación a la vez, los robots alineados avanzan antes de que otros robots estén siquiera agregados y antes de que otros robots realicen su consenso, por lo que se generan múltiples direcciones y se desmenuza el *cluster* en pequeños grupos siguiendo estas direcciones. Dado que el enjambre no avanza como conjunto, sino que se disuelve, en un punto dado los robots alineados dan la vuelta y tratan de reconstruir el *cluster*, dando lugar al com-

portamiento observado en el que el grupo se dispersa y se agrega de forma más o menos cíclica.

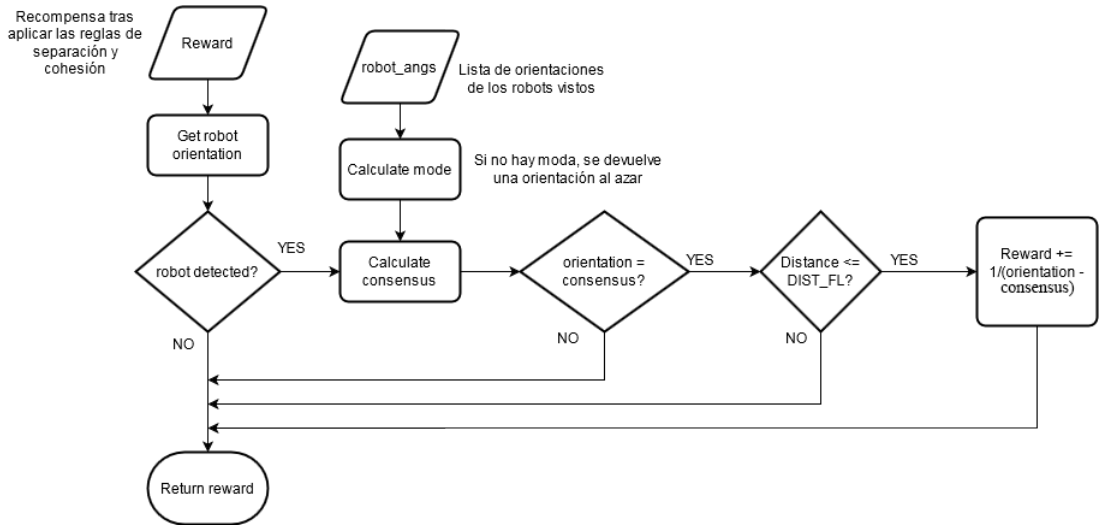

Figura 4.22: Funcionamiento básico de la regla de Alineación (versión 6)

4.3.3.8. Séptima versión

Los siguientes experimentos tienen por objetivo, pues, favorecer la coordinación entre robots y la sincronización de la fase de alineación. Para ello se toman varias medidas concernientes a la velocidad de los robots con respecto a otros y a las posibles maneras que tienen de esperarse entre sí. Además, se debe mejorar la toma de decisiones del consenso, tratando de reducir la influencia que la orientación instantánea de los robots tiene sobre este. Es por esto por lo que los siguientes experimentos se centran en medidas como modificación selectiva de la velocidad o adición de una nueva acción que permita a los robots no moverse en absoluto. El primero de estos experimentos se centra en añadir una nueva función que modifica la velocidad de los robots en función de su distancia relativa. De este modo, cuando los robots están lejos se mueven más rápido que cuando están cerca (se puede visualizar esta función en contexto en la Fig. 4.24). El objetivo es favorecer la agregación rápida de los robots distantes en el enjambre mientras los robots ya agregados son ralentizados para así esperar a los más distantes. Los resultados de este experimento son los mostrados en la Fig.4.23 y en el siguiente enlace: Experimento 7 (50 episodios)

La gráfica muestra un comportamiento muy similar a los anteriores, sin nada demasiado relevante que aportar más allá de los primeros episodios que rápidamente escalan hasta orbitar el valor final estable. En el vídeo, no obstante, se puede ver a los robots hacer *clusters* separados que posteriormente se agregan en uno más grande. También se aprecian apariciones de la regla de alineación en ciertos momentos aislados. No obstante, la diferencia de velocidades que se pretende observar no es evidente en el vídeo. Una posible explicación es que las distancias requeridas para ello son demasiado pequeñas. Al margen de este experi-

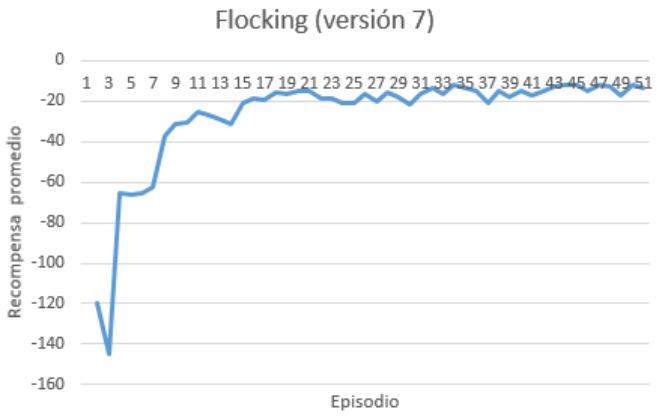

Figura 4.23: Resultados obtenidos en el séptimo experimento de *flocking*

mento, se grabó otro vídeo de la misma solución, pero con solo 20 episodios de entrenamiento. Esto es debido a un comportamiento prometedor durante la previsualización del simulador: Experimento 7 (20 episodios). El video demuestra que, realmente, no hay demasiada diferencia más allá de una mayor torpeza dado el menor entrenamiento.

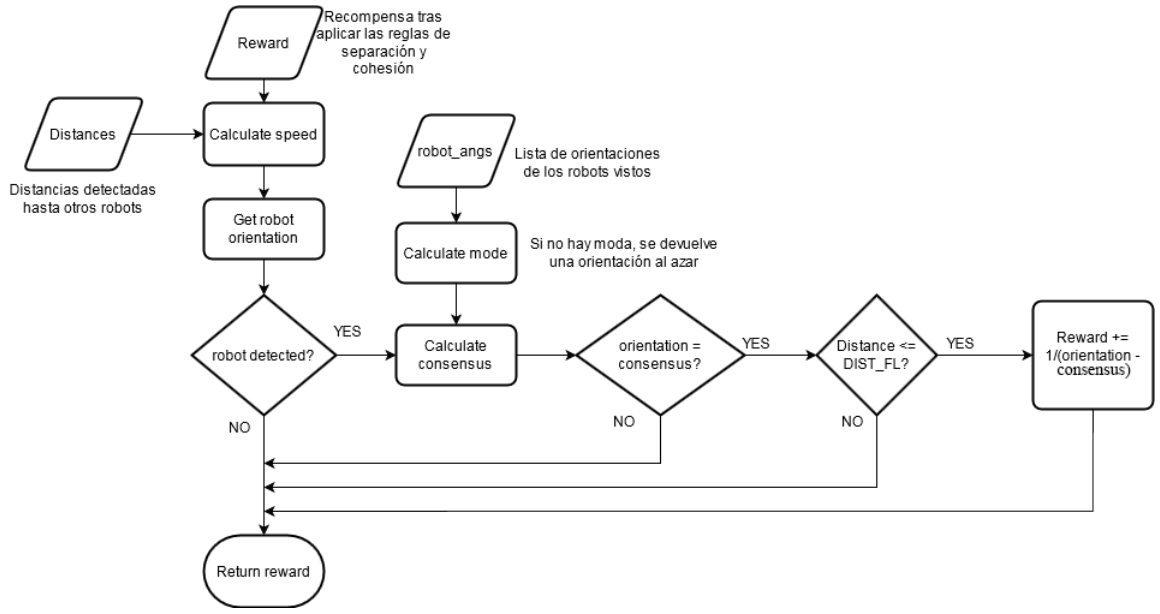

Figura 4.24: Funcionamiento básico de la regla de Alineación (versión 7)

4.3.3.9. Octava versión

El siguiente experimento ha consistido en un aumento de la distancia de *flocking*, parámetro que determina cuándo se inicia la alineación y por tanto, cuándo se modifica la velocidad

(el diagrama de flujo correspondiente se encuentra en la Fig. 4.26). El objetivo de esta medida es apoyar el cambio de velocidad y hacerlo más apreciable, al mismo tiempo que se relaja el tamaño de los *clusters* y se permite que la lineación empiece antes, para favorecer aún más una rápida agregación.

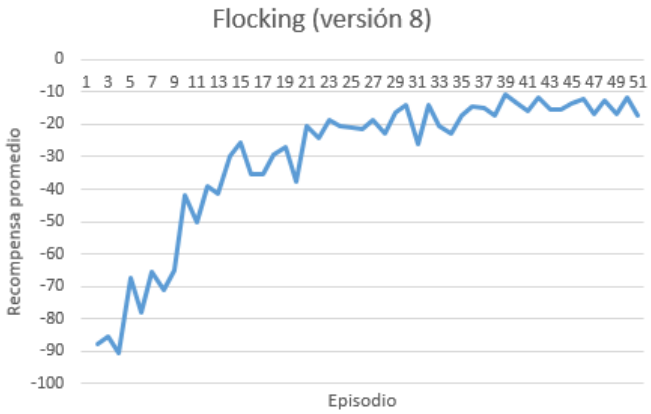

Figura 4.25: Resultados obtenidos en el octavo experimento de *flocking*

Como se puede ver en la Fig. 4.25 y en el enlace: Experimento 8, aumentar la distancia también aumenta los picos de la gráfica, haciendo el comportamiento más errático. En el vídeo, no obstante, los cambios de velocidad son extraordinariamente sutiles, aunque apreciables. En general el comportamiento favorece una rápida agregación y la regla de alineación se hace visible sobre todo en los compases finales de las simulaciones, donde los robots se desplazan ligeramente formando organizaciones con forma de flecha o una rudimentaria fila. Todavía hay robots descolgados del enjambre, sin embargo, se puede considerar esto como un pequeño paso al frente, incluso como un comportamiento prometedor si se le da más tiempo de simulación. Aún así, hacen falta muchas mejoras para lograr el comportamiento perfecto. Los siguientes experimentos tratan de mejorar este resultado, sin embargo, ahora la experimentación se vuelve algo más caótica, centrándose en cubrir posibilidades y ver qué sucede. En caso de no lograr mejoras significativas, este comportamiento demuestra ya un mínimo avance respecto a la situación inicial.

4.3.3.10. Novena versión

Dicho esto, el noveno experimento consiste en dotar a los robots de la capacidad de detenerse completamente, mediante la adición de una nueva acción a tomar. Se espera que esta medida permita a los robots esperar o dejar de moverse una vez han alcanzado una posición intermedia óptima, de este modo no influyen en futuros consensos realizados por nuevos robots que se agreguen al enjambre (esta medida no afecta al diagrama de flujo, por lo que el del Experimento 8 sigue vigente). Anteriormente, los robots empezaban a girar sobre sí mismos cada vez que alcanzaban una posición interesante, por lo que los consensos, especialmente el basado en la media, se volvían locos gracias a la constante rotación. Con esta medida se

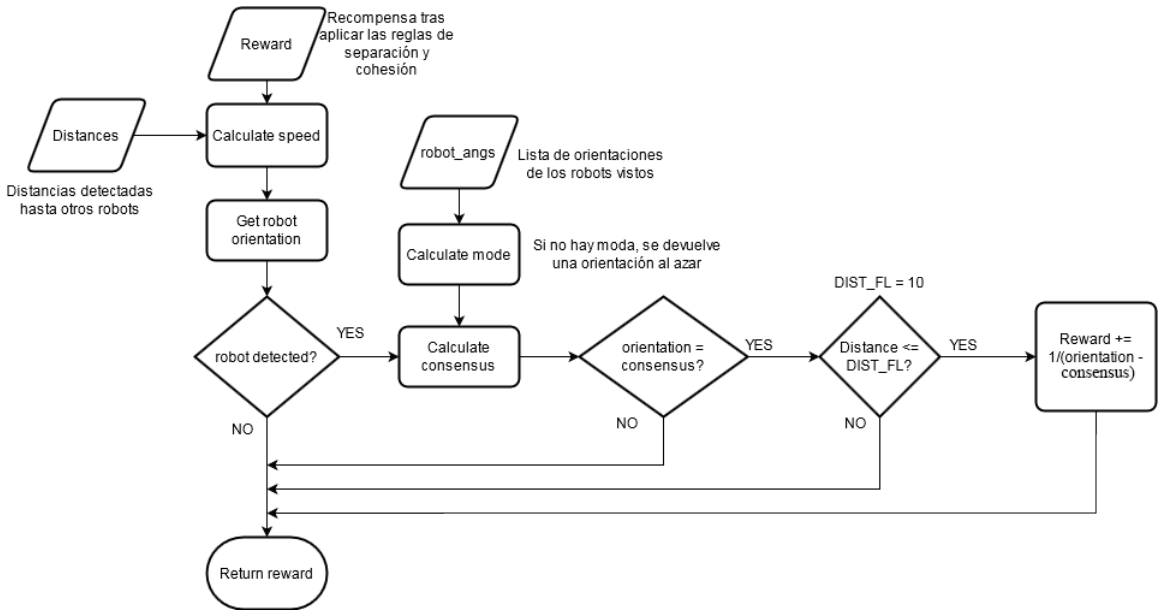

Figura 4.26: Funcionamiento básico de la regla de Alineación (versión 8)

espera mantener la orientación de los robots algo más constante para favorecer nuevos consensos. Los resultados son los mostrados a continuación: (Fig. 4.27 y Experimento 9).

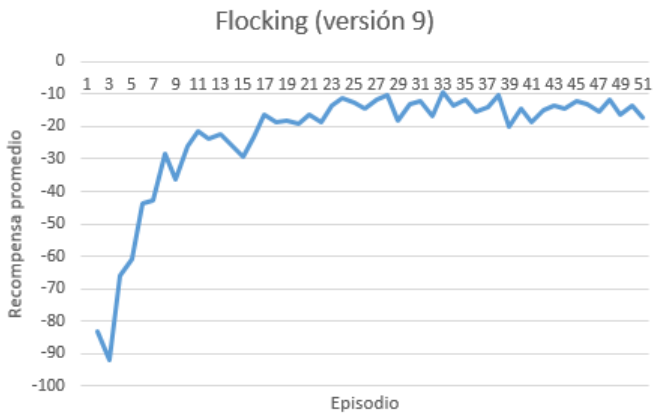

Figura 4.27: Resultados obtenidos en el noveno experimento de *flocking*

Desde un punto de vista aislado, se puede considerar este experimento como un éxito debido a que no empeora el comportamiento y sin embargo se puede ver de forma evidente en el video cómo los robots se paran por completo y esperan hasta formar un gran *cluster*. Este cluster luego se deforma y estira hasta formar una elipse que ya muestra síntomas de dirección. No obstante, desde el punto de vista global, este experimento no ha aportado demasiado al comportamiento, aunque no ha sido perjudicial.

NOTA: La creación de una nueva acción para los robots implica la creación de una nueva salida en cada red neuronal, haciendo incompatibles las soluciones previas a este experimento (más información en la Fig. 4.28). Si se desea probar soluciones correspondientes al experimento 8 o anteriores, se debe modificar el archivo test_solution.py para generar redes neuronales de únicamente 4 salidas (de ahora en adelante se considerarán redes de 5 salidas)

```
#MODELO NN
nn = nn.RNNModel(8,4,[8,6,4])
nn.set_model_params(nn.get_random_model_params(1))
fitness_list=sim(True)

#MODELO NN
nn = nn.RNNModel(8,5,[8,6,5])
nn.set_model_params(nn.get_random_model_params(1))
fitness_list=sim(True)
```

Figura 4.28: Modificación realizada a la declaración de cada red neuronal. La versión de arriba es compatible con redes del Experimento 8 o anteriores, mientras que la versión de abajo es compatible con redes del Experimento 9 o posteriores

4.3.3.11. Décima versión

Dado que la respuesta milagrosa no se encuentra en la nueva acción, se ha retomado la tendencia anterior de aumentar la distancia de agregación (diagrama de flujo en Fig. 4.30). Se espera que este nuevo aumento aporte algo más de direccionalidad al enjambre, permitiendo que más tiempo de simulación se dedique a la alineación en vez de a la formación de *clusters*. En este caso la distancia de *flocking* se ha aumentado a 15, lo que reduce el espectro de decisión de cada robot a agregarse o alinearse. El espacio intermedio que antes existía entre ambas opciones se ha eliminado, y los resultados han sido los que se muestran (Fig. 4.29, Experimento 10):

Como ya se había comprobado antes, el aumento de distancia ha comportado un aumento de ruido en cuanto a las recompensas, siendo los picos de la gráfica más pronunciados que en el experimento anterior. El video confirma que esta nueva solución no es demasiado prometedora. Se puede ver cómo los robots involucionan respecto a su comportamiento, ignorando la agregación y quedándose congelados en mitad de las simulaciones sin ningún orden aparente. También se observa cómo los robots terminan pegados a las paredes de forma totalmente antagónica a lo mostrado en los experimentos 8 y 9.

A pesar de este resultado, en los dos siguientes experimentos se ha tratado de subsanar el mal comportamiento obtenido en este. Se han realizado medidas tratando de forzar a los robots a un consenso y se ha tratado de evitar el escenario en el que todos los robots quedan congelados en pantalla.

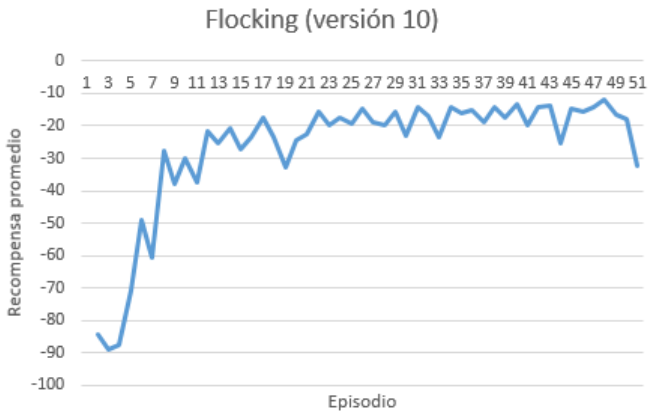

Figura 4.29: Resultados obtenidos en el décimo experimento de *flocking*

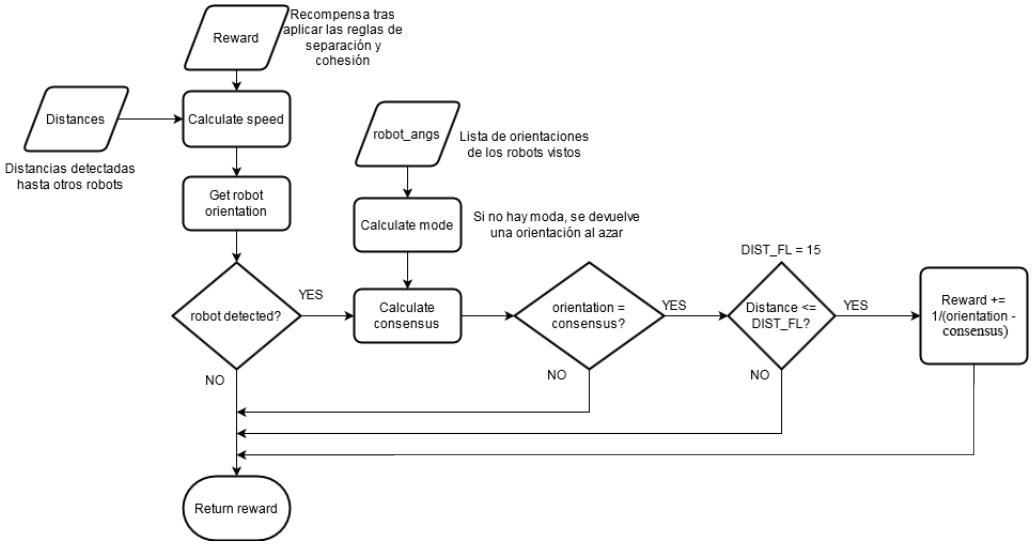

Figura 4.30: Funcionamiento básico de la regla de Alineación (versión 10)

4.3.3.12. Undécima versión

En este undécimo intento, se ha intentado reforzar la función de consenso al eliminar la elección aleatoria. Originalmente, si el conjunto de orientaciones de los robots cercanos no poseía una moda clara, es decir, ninguna orientación se repetía, se tomaba una orientación al azar como decisión final. En este caso, si el conjunto no tiene una moda clara, se realiza la media de las orientaciones vistas (diagrama de flujo en Fig. 4.32). Los resultados pueden observarse en (Fig. 4.31, Experimento 11):

Como se observa, esta medida no ha modificado en gran medida el comportamiento del experimento anterior. La gráfica de la Fig. 4.31 es muy similar a la de la Fig. 4.29 y el vídeo muestra el mismo tipo de comportamiento. Los robots actúan de forma errática, ignoran la agregación, no son capaces de formar *clusters* consistentes y terminan también congelados en

mitad del entorno.

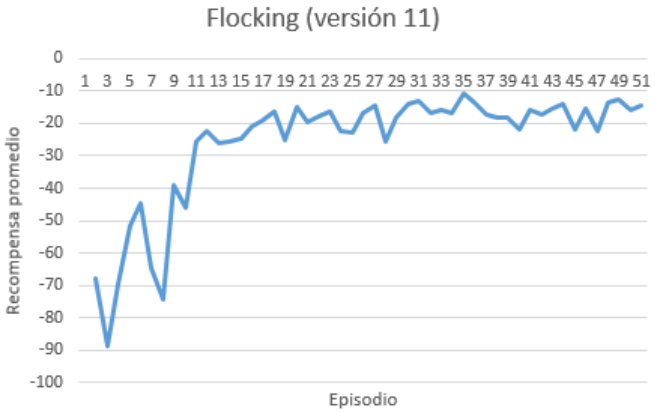

Figura 4.31: Resultados obtenidos en el undécimo experimento de *flocking*

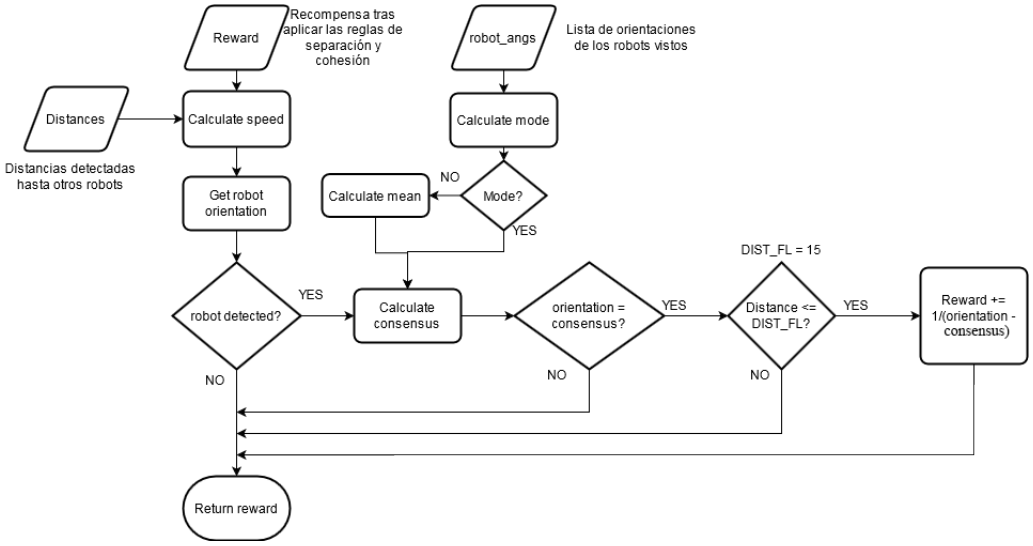

Figura 4.32: Funcionamiento básico de la regla de Alineación (versión 11)

4.3.3.13. Duodécima versión

Como última prueba en esta línea, se ha tratado de limitar el uso de la nueva acción mediante una pequeña penalización de recompensa. Si un robot decide elegir la acción de quedarse quieto, recibirá una penalización de -0,1. Con esto se pretende impedir que los robots se queden quietos durante largos periodos de tiempo, si bien todavía pueden emplear esta acción en momentos puntuales pues la penalización es pequeña (diagrama de flujo en Fig. 4.34). Los resultados obtenidos se muestran en la Fig. 4.33 y el enlace Experimento 12.

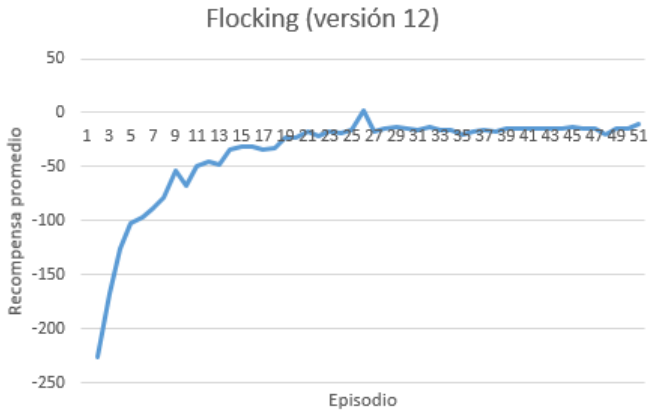

Figura 4.33: Resultados obtenidos en el duodécimo experimento de *flocking*

Sorpresivamente, la de la Fig. 4.33 es una de las gráficas más lineales obtenidas en este trabajo. No obstante, el comportamiento mostrado en el video, pese a ser mejor que en el experimento anterior, no está a la altura de anteriores experimentos. Los robots han abandonado prácticamente su nueva acción, y tardan en formar *clusters* consistentes. Es por esto por lo que se decide restaurar los parámetros del experimento a sus correspondientes en el experimento 9, experimento que mejor ha funcionado hasta ahora (junto con el 8).

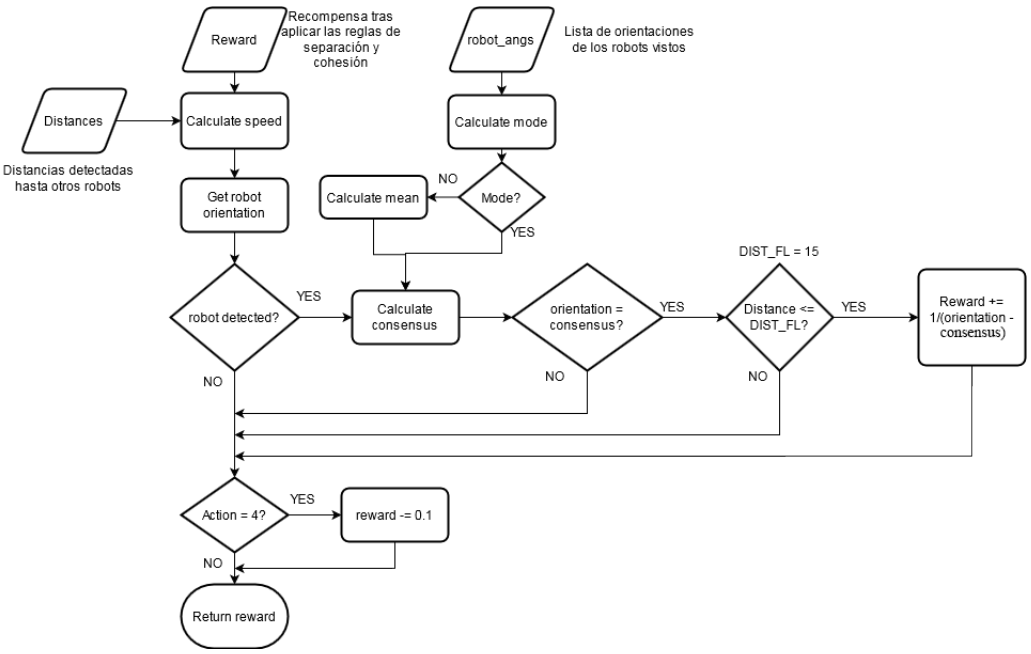

Figura 4.34: Funcionamiento básico de la regla de Alineación (versión 12)

4.3.3.14. Consideraciones finales

En retrospectiva, hasta ahora los experimentos que más rendimiento han aportado son los de la sección central, del 6 al 9 con especial énfasis en el 8 y 9. Estos experimentos muestran una buena agregación con una ligera intención de consenso, se pueden ver varios robots avanzando en dirección similar, o en general se observa cómo el *cluster* principal se alarga en una dirección determinada. El comportamiento de *flocking* todavía no se extiende a todos los robots del enjambre, sino que se aprecia esporádicamente en pequeños grupos de robots y no dura demasiado. Los robots solo tienen 3 sensores y la cantidad de vecinos que pueden ver es limitada, lo cual reduce mucho las posibilidades de consenso a gran escala. No obstante, en grupos pequeños lo suficientemente cercanos y alejados de otros robots que puedan interferir en el consenso, el comportamiento puede apreciarse satisfactoriamente, aunque sea por breves instantes.

5. Resultados

A este punto, una vez realizada la investigación, es el momento de analizar los objetivos planteados en el capítulo 3 y determinar si se han cumplido o no y en qué medida. La existencia del trabajo del año pasado ha sido al mismo tiempo bendición y maldición, pues aunque ha marcado un camino claro a seguir y proporcionado la mayoría de herramientas necesarias para este; la comparación entre ambos es inevitable y añade presión a la hora de investigar. No obstante, dado el objetivo principal de ampliación del trabajo realizado el año pasado, este documento no se ha desviado de su propósito en ningún momento.

El primer subobjetivo planteado es la actualización del trabajo a una nueva plataforma y la documentación de dependencias y del proceso de instalación en sí. En este sentido, el proceso entero ha sido escrito y mostrado, por lo que no se puede considerar incumpido. Las dependencias a instalar han sido incluidas, con sus respectivas versiones e instrucciones adicionales previendo posibles errores sucedidos durante la realización de este mismo trabajo y cómo evitarlos. Si bien con esto no se garantiza la completa ausencia de errores de dependencias o su falta, el proceso de instalación y funcionamiento del proyecto ha quedado más claro que tal y como estaba al inicio.

Permítaseme la licencia de considerar el tercer subobjetivo antes que el segundo, dada la diferencia de complejidad entre ambos. El tercer subobjetivo consiste en la corrección de errores misceláneos que pudiesen surgir dirante el desarrollo. Este objetivo consumió más tiempo del necesario, pues el código previo disponible falló bastante más de lo esperado. El punto más costoso fue la reconstrucción de una función mediante ingeniería inversa, presente en el TFG del año pasado pero no en los archivos disponibles para este (al menos, no la versión final de la función). El error, hasta lo que se sabe, ha sido solucionado y el programa funciona sin mayor complicación. Aún así, como este subobjetivo en sí mismo ejemplifica, nuevos errores pueden surgir en el futuro si el código cambia de plataforma una vez más. En este sentido, se ha hecho todo lo posible para que esto no sea necesario, eligiendo una nueva plataforma universal y conocida, que todo el mundo puede configurar fácilmente y depurando el código hasta la ausencia de errores.

Por último, el segundo subobjetivo ha sido el más largo y complicado de los tres. El segundo subobjetivo consiste en la ampliación del trabajo en sí, añadiendo nuevas funcionalidades y comportamientos a los presentes anteriormente. En este sentido, se han añadido funcionalidades como un selector que permite elegir el comportamiento a entrenar nada más iniciar el programa, en vez de tener que modificar la función de recompensa a mano cada vez que se desea cambiar de comportamiento. El selector está vinculado a un archivo que contiene las funciones de recompensa necesarias para varios comportamientos, pudiendo seleccionar una de ellas para entrenar un comportamiento concreto. En cuanto al número de comportamien-

tos, cabe destacar que los del año pasado eran muy rudimentarios, reduciéndose prácticamente a evitación de obstáculos y agregación. A pesar de que una función de *flocking* se escribió el año pasado, esta función se demostró que no producía un comportamiento correcto y era por tanto una hipótesis. Este trabajo ha dedicado la mayor parte del tiempo otorgado al segundo subobjetivo a perfeccionar y mejorar esta función de *flocking*, siendo este el comportamiento más complejo posible pues los robots carecen de manipuladores que permitan el *foraging*. En este sentido, el comportamiento de bandada dista mucho de ser perfecto, no obstante ha mejorado respecto a la implementación realizada en el trabajo previo. Es por esto que el segundo subobjetivo debería considerarse al menos parcialmente cumplido, siendo tan vasto como puede ser una ampliación.

Dicho esto, los resultados materiales de este trabajo se pueden resumir en los siguientes. Para el primer subobjetivo, consúltese el capítulo 4, concretamente el apartado 4.2.1. El tercer subobjetivo se encuentra a continuación, en el apartado 4.2.2. Por último, el tercer subobjetivo abarca la totalidad de la sección 4.3 y se extiende hasta el final del capítulo. El último apartado 4.3.3 cuenta con 12 experimentos, de los cuales los mejores resultados se han obtenido en los experimentos del 6 al 9, cuyos resultados se muestran en las figuras Fig. 4.21, Fig. 4.23, Fig. 4.25 y Fig. 4.27; así como en los enlaces: Experimento 6, Experimento 7, Experimento 8 y Experimento 9.

6. Conclusiones

En última instancia, un capítulo final concluye este viaje. Como conclusión, se recalcan varios aspectos importantes de lo que ha supuesto realizar este trabajo.

Como ya se ha explicado antes, la existencia de este mismo trabajo en años anteriores tiene un carácter de doble filo. Por un lado, proporciona una muy buena infraestructura y guía de trabajo, hasta el punto de considerar que realmente está todo ya hecho. Esto es muy útil a la hora de determinar un camino y objetivos a seguir, así como de disponer de herramientas funcionales que sirven de base para la realización. En ese sentido, he apreciado la ventaja de ideas que me proporcionó con respecto a algunos compañeros, que en ciertos instantes se sintieron más perdidos al carecer de esta guía. No obstante, el lado negativo de esta colaboración resulta en la comparación. Es imposible no comparar este trabajo con el realizado el año pasado, y a pesar de que los objetivos de cada uno son diferentes, no existe certeza de que este trabajo derrote al anterior si se llegasen a producir dichas comparaciones. Esto ha supuesto una gran presión y la sensación de que el trabajo nunca está terminado del todo a pesar de cumplir con los objetivos propuestos. En este sentido, los compañeros que no contaron con trabajos de años pasados han podido determinar su propio camino sujeto únicamente a sus propias convicciones.

La conclusión más directa que se extrae de los resultados es que la forma de la gráfica y el comportamiento final observado no siempre van en concordancia. El análisis de la evolución de la recompensa media resulta útil para determinar la estabilidad general del comportamiento, cuando las recompensas se mantienen en torno a un mismo valor durante varios episodios. Esto indica que el comportamiento ha alcanzado un valor máximo de recompensa que no debería aumentar en el futuro, por lo que entrenar más no va a suponer un cambio apreciable. No obstante, la estabilidad por sí sola no garantiza un comportamiento idóneo.

Si se observa la Fig. 6.1, se puede observar como la mayoría de las versiones terminan estabilizándose en mayor o menor medida. La primera versión ni siquiera aparece en la gráfica debido a que sus valores son tan elevados que se salen de la escala de las demás versiones. A pesar de lo que pueda parecer, las versiones más estables no se corresponden con los mejores comportamientos. La duodécima versión aparenta ser la más rápida y carente de ruido, sin embargo su video correspondiente (Experimento 12) no muestra el mejor comportamiento. Un comportamiento estable puede consistir en detener a todos los robots del enjambre, lo cual no puede considerarse válido. Los comportamientos en los que los robots chocan contra las paredes del entorno pueden tener evoluciones y gráficas también prometedoras, y sin embargo son situaciones a evitar. Esto no quiere decir que las gráficas sean inútiles: una gráfica inestable o errática produce comportamientos más erráticos e impredecibles que difícilmente satisfarán las expectativas. La estabilidad de las gráficas es por tanto una condición necesaria,

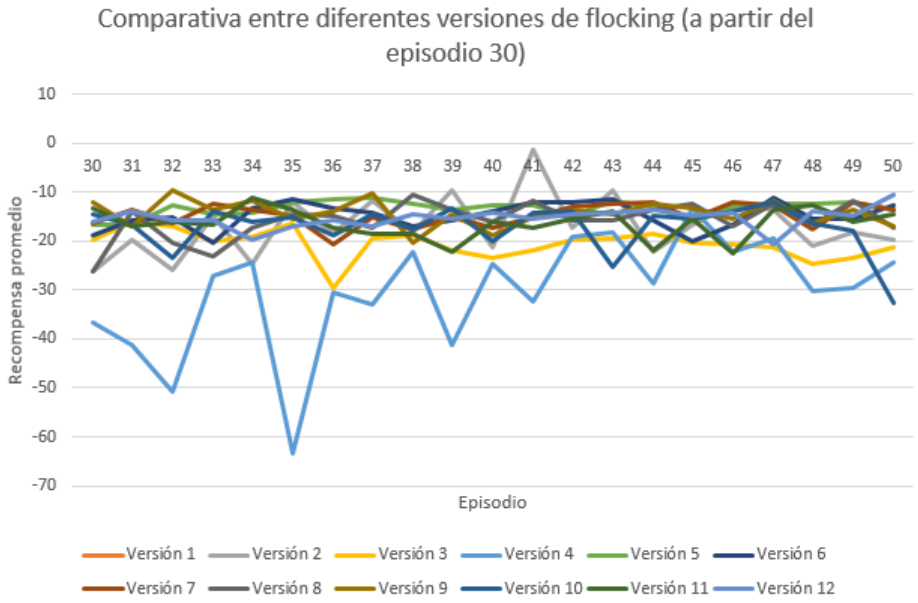

Figura 6.1: Comparativa entre diferentes versiones del comportamiento de *flocking*, del episodio 30 en adelante

aunque no suficiente, para lograr los comportamientos deseados.

Otro aspecto a considerar ha sido la limitación física de los robots. Es común colocar toda la presión en el código, pensando que el aprendizaje automático es una herramienta milagrosa capaz de convertir hasta una tostadora en un robot de última generación. Y a pesar de que hasta cierto punto puede ser posible, los robots simulados cuentan con grandes limitaciones de hardware que han dificultado la tarea de generar un comportamiento coordinado. El comportamiento de enjambre exige sencillez, y este trabajo ha tratado de mantenerse fiel a ese principio. No obstante, la ausencia de más sensores o de diferentes tipos en los robots limita mucho la cantidad que un observador es capaz de percibir (los robots Khepera IV, sin ir más lejos, cuentan con un anillo entero de 8 sensores, en vez de los tres que poseen los robots simulados). Esto a su vez reduce la capacidad de consenso del enjambre y acaba generando pequeños conjuntos descoordinados entre sí y algunos choques debido a que los robots ni siquiera pueden ver sus espaldas. Este resultado es lo que más se aprecia en los videos, y hace que surja la pregunta de si realmente estos robots simulados están diseñados exclusivamente para trabajar en pequeños grupos y no como gran enjambre. Sea así o no, el trabajo en pequeños subconjuntos parece la opción óptima para este tipo de robot.

Dicho esto, este trabajo ha resultado altamente informativo y útil. El aprendizaje recibido es invaluable, y a pesar de las múltiples limitaciones ha logrado cumplir los objetivos planteados. Es por esto que no considero esta experiencia como negativa en absoluto. La naturaleza perfeccionista puede ser una gran carga en ocasiones, y de verse cegado por ella, este trabajo podría no terminar nunca. Es importante entender las limitaciones no como algo negativo, sino como punto de partida para mejorar en el futuro y como trampolín para desarrollar

soluciones creativas.

6.1. Trabajos futuros

Todo trabajo es susceptible de ser mejorado en el futuro, y este no es una excepción. Existen múltiples mejoras capaces de paliar algunos de los problemas más importantes encontrados durante este trabajo. La primera de estas mejoras es el aumento del número de sensores de los robots de enjambre. Como una posible ampliación, se podría definir un nuevo robot virtual en el simulador, dotado de más sensores que los actuales *auriga* y adaptar las funciones de recompensa a estos nuevos robots. Al tener más sensores, es posible que los robots puedan alcanzar mejores consensos y por tanto realizar las tareas de *flocking* de forma más coordinada. Además, se podrían equipar los nuevos robots con pinzas para poder manipular obstáculos y hacer tareas de *foraging*.

Otra mejora interesante es el refinamiento del comportamiento de *flocking* en sí, probando nuevas funciones de recompensa o cambiando el algoritmo en su totalidad, implementando las leyes de la bandada de una forma distinta. Se podría diseñar además un sistema de comunicación para los robots, ya sea de forma explícita o de otra, para dar paso a comportamientos más complejos. También se podría probar el auto-ensamblaje de robots, combinando varios robots simples para formar una unidad más grande y versátil.

La robótica de enjambre ofrece múltiples posibilidades que todavía están por explorar. Las posibilidades son vastas y todavía se pueden descubrir nuevas aplicaciones para los enjambres robóticos que actualmente no se hayan considerado aún. Espero que este trabajo pueda servir de guía a aquellos que me sucedan.

Bibliografía

[1] Trabajo de Fin de Grado del año 2019-2020. DE LA TORRE GARCÍA, JUAN: *Desarrollo de comportamientos de enjambre mediante computación evolutiva*

[2] Página de Wikipedia sobre la idea filosófica de la emergencia. `https://es.wikipedia.org/wiki/Emergencia_(filosof%C3%ADa)`

[3] Página de Wikipedia sobre sistemas emergentes. `https://es.wikipedia.org/wiki/Sistema_emergente`

[4] Diapositivas del tema 1 de la asignatura Sistemas Multirobot de Ingeniería Robótica. ARQUES CORRALES, MARÍA DEL PILAR: *Sistemas Multirobot. Introducción*

[5] Página de Wikipedia sobre colonias biológicas. `https://es.wikipedia.org/wiki/Colonia_(biolog%C3%ADa)`

[6] Libro sobre computación evolutiva. A. E. EIBEN, J. E. SMITH: *Introduction to Evolutionary Computing*

[7] Artículo acerca de la aplicación de algoritmos evolutivos al diseño cinemático de robots. O. Chocron and P. Bidaud, 'Evolutionary algorithms in kinematic design of robotic systems' Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS '97, Grenoble, France, 1997, pp. 1111-1117 vol.2, doi: 10.1109/IROS.1997.655148.

[8] Artículo acerca del uso de algoritmos evolutivos para reducir la huella de carbono de una cadena de producción robotizada. `https://www.sciencedirect.com/science/article/abs/pii/S0959652617307394`

[9] Artículo acerca del empleo de algoritmos evolutivos para el diseño de controladores robóticos. R. A. Watson, S. G. Ficiei and J. B. Pollack, 'Embodied evolution: embodying an evolutionary algorithm in a population of robots', Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 1999, pp. 335-342 Vol. 1, doi: 10.1109/CEC.1999.781944.

[10] Blog acerca de Estrategias Evolutivas. `https://blog.otoro.net/2017/10/29/visual-evolution-strategies/`

[11] Entrada del foro *Stack Overflow* que explica maneras de calcular la moda estadística de una lista. `https://stackoverflow.com/questions/10797819/finding-the-mode-of-a-list`

[12] Página de Wikipedia acerca del comportamiento de *flocking*, donde se exponen las leyes de la bandada. `https://en.wikipedia.org/wiki/Flocking_(behavior)`

[13] Página de Wikipedia acerca del software VMWare. https://es.wikipedia.org/wiki/VMware

[14] Página de Wikipedia acerca del software Visual Studio Code. https://es.wikipedia.org/wiki/Visual_Studio_Code

[15] Página de Wikipedia acerca del software Overleaf. https://en.wikipedia.org/wiki/Overleaf

[16] Página de OpenAI acerca de la librería Gym. https://gym.openai.com/docs/

[17] Página de Wikipedia acerca del software Microsoft Excel. https://es.wikipedia.org/wiki/Microsoft_Excel

[18] Video que muestra un enjambre robótico encargándose de la logística en un almacén de Amazon. https://www.youtube.com/watch?v=4D9k3tO4LDA

[19] Página con información acerca de las estrategias evolutivas y su relación con el aprendizaje por refuerzo. https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html

[20] Paper científico acerca de la aplicación de estrategias evolutivas al aprendizaje por refuerzo. https://arxiv.org/abs/1106.0221

[21] Método llamado *log-derivative trick* para la estimación del gradiente, altamente empleado en *Machine Learning* http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/

A. Anexo: Enlaces de interés

En este anexo se incluyen enlaces a todos los archivos de interés del proyecto, desde hojas de cálculo con los datos de cada experimento, pasando por todos los archivos de código fuente a archivos en formato *json* con los modelos de redes neuronales empleados en cada experimento.

Enlace a la carpeta que contiene todos los videos de comportamiento: `https://drive.google.com/drive/folders/1e3ktBO4M8-ew5WMaIbzgrplofcKOWZz8?usp=sharing`

Enlace al resto de archivos (modelos, datos de experimentos, código fuente): `https://drive.google.com/drive/folders/1B6gAuZSgzRZq8XIn7nMP0wyigZxDoZC0?usp=sharing`

Enlaces a videos individuales:

- Experimento final de agregación pura sin evitación de obstáculos: `https://drive.google.com/file/d/11yNGtcR1DpFac-y0ymJcifAEqhI9i_ZY/view?usp=sharing`
- Experimento final de agregación y evitación de obstáculos: `https://drive.google.com/file/d/1PiDP8Imw7YRYNRoOAWEaz2E82GFKcFVc/view?usp=sharing`
- Experimento de *flocking* empleando la hipótesis del año pasado `https://drive.google.com/file/d/1Q2xbmM6w2rG2c7rMtPbRzBxtmiwGeqbV/view?usp=sharing`
- Primera versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1OwVdNbpkvVULhG5n7fCZBdtzP6hmEj3H/view?usp=sharing`
- Segunda versión del comportamiento de *flocking*: `https://drive.google.com/file/d/117AzPbXYAfLxpYRYIjY9vkxHAZOGdhb-/view?usp=sharing`
- Tercera versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1P2_wTM4rBVqJMUAJTg-VylhNSSW-kFJa/view?usp=sharing`
- Cuarta versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1vyAcdz1MphDMVo_X2jVTtzsvn2A_2yw3/view?usp=sharing`
- Quinta versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1VaGN2dCFUlBN3DuRdZdD4Vo_6rQr1uz-/view?usp=sharing`
- Sexta versión del comportamiento de *flocking*: `https://drive.google.com/file/d/12GTM5-VR-loB726W3hg5kom6gfiw2d4q/view?usp=sharing`
- Séptima versión del comportamiento de *flocking* (20 episodios): `https://drive.google.com/file/d/1YsdPg476gq4syxehWLBoMqmCDj-fw9YG/view?usp=sharing`

- Séptima versión del comportamiento de *flocking* (50 episodios): `https://drive.google.com/file/d/1Ua2SYqxVix6kNHxdE8TdedrTk5XebqoH/view?usp=sharing`
- Octava versión del comportamiento de *flocking*: `https://drive.google.com/file/d/13FJvQJyTCEXjHDLAaXkWiLVhhyucHMDd/view?usp=sharing`
- Novena versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1qhwExm2tj9dA5ZWv01XlqzKmA9YQiSf8/view?usp=sharing`
- Décima versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1rTA6JmXwT084DuFCTyM6qOmLl6dAKxRl/view?usp=sharing`
- Undécima versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1E_YAYSc44KPCzVlZVuySsRTBNLY7LaHR/view?usp=sharing`
- Duodécima versión del comportamiento de *flocking*: `https://drive.google.com/file/d/1Z1RcCfp6x92eGhZnKjBCdej5JqSVyfd6/view?usp=sharing`