TRABAJO FIN DE GRADO

Técnicas de pastoreo multi robot

AUTOR

Darío Martín Sendra

DIRECTORA

Rosario Aragüés Muñoz

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020/2021

AGRADECIMIENTOS

En primer lugar, a mi directora *Rosario Aragüés* por su ayuda en la planificación, información y organización en este Trabajo de Fin de Grado.

En segundo lugar, a mi familia y a mis amigos que han estado a lo largo de toda mi carrera apoyándome en todo momento y animándome a seguir adelante

RESUMEN

En el mundo de la robótica existe un desarrollo continuo desde hace años. Esto ha llevado a que se necesite una mayor especialización de los trabajos, formándose múltiples disciplinas. Una de las disciplinas que se va a considerar en este trabajo es la multi-robot. Los sistemas multi-robot son sistemas de dos o más robots que disponen de capacidad de comunicaciones entre ellos y se intercambian información para realizar una tarea con un objetivo común. El trabajo se va a centrar en un sistema multi-robot con un objetivo particular: poder controlar y llevar a un punto objetivo un grupo de animales (ovejas, vacas…). El control se realiza en base a un algoritmo de un artículo científico y se realizan simulaciones en las que existen dos tipos de robots: los robots cooperativos (dogs) y los robots no cooperativos (sheeps). En estos últimos se introducirán una serie de movimientos para simular el comportamiento de los rebaños. El trabajo explica paso a paso cómo ha ido construyéndose el control. También las diversas simulaciones que se realizan para verificar el comportamiento del algoritmo y sus resultados. Empieza por el movimiento de una única oveja, realizando el control para llevarla a un punto objetivo. Una vez completado, se introducen dos perros, en un principio moviéndose al son de la oveja a una posición deseada. Y una vez conseguido, moviéndose con su propio regulador. Después se realizan los ajustes necesarios para generalizar a m perros y a n ovejas. Dejando lo realizado hasta el momento a parte, se realiza un algoritmo de reagrupamiento de rebaño para las n ovejas. Por último, se introduce esta parte en el conjunto y se realiza el algoritmo final.

Índice

Contenido

1. Introducción 1

1.1 Motivación 1

1.2 Contexto 2

1.3 Objetivos 3

1.4 Organización del documento 4

2. Descripción del problema 5

3. Modelo con una única oveja y modelo monociclo del sistema 6

3.1 Modelo repulsión 6

3.2 Controlador proporcional del Offset-Point (p) 7

3.3 Control de sistema monociclo mediante Offset-Point 9

3.3.1 Ajustes del controlador 12

4. Movimiento una oveja y varios perros 17

4.1 Movimiento con perros pasivos 17

4.2 Movimiento con perros activos 21

4.2.1 Asignación posiciones deseadas respecto a posiciones anteriores 21

4.2.2 Generalización a m perros 23

5. Movimiento varias ovejas 24

5.1 Colocación simple (en circulo) 24

5.2 Algoritmo de rebaño 25

5.3 Movimiento varias ovejas y varios perros 27

6. Simulaciones finales 28

6.1 Primera simulación 29

6.2 Cambio de posiciones iniciales 32

6.3 Variación de Kd y Ks 33

6.4 Variación del radio r 35

6.5 Variación número de perros y ovejas. 36

7. Conclusión y trabajo futuro 37

7.1 Conclusión 37

7.2 Trabajo Futuro 37

8. Bibliografía 38

ANEXO (Código) 39

Capítulo 1

1. Introducción

1.1 Motivación¡Error! Marcador no definido.

La robótica es en la actualidad una de las ramas de conocimiento que más importancia y desarrollo está teniendo en los últimos años. A día de hoy, son innumerables los procesos que necesitan ser robotizados, ya sea por ser peligroso para el ser humano, ser repetitivo o necesitar una precisión y eficiencia solo al alcance de estos. Los procesos han variado a lo largo del tiempo. En los comienzos solo se veían en grandes fábricas. A día de hoy los podemos encontrar en restaurantes, hospitales, e incluso en nuestros propios hogares (robots de limpieza autónoma).

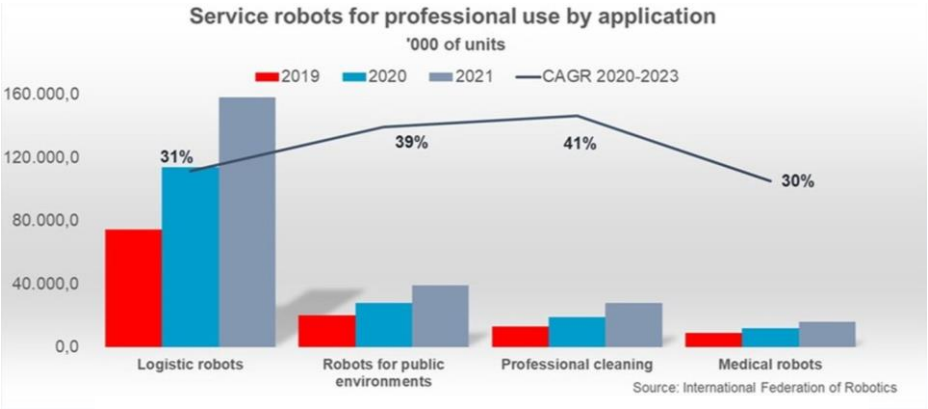

Figura 1.1: Evolución del número de robots en servicio en distintas áreas
Fuente: IFR International Federation of Robots

Como se puede observar en la figura 1.1, la evolución del uso de robots va creciendo incluso fuera del propio uso industrial, donde sigue creciendo bastante.

Cada vez las actividades a realizar son más complejas, por lo que es necesario que esos robots sean capaces de reaccionar a su entorno. Son los llamados robots autónomos, que mediante sensores son capaces de tomar decisiones en función de la información que poseen.

Otra de las modalidades sobre las que versa ese trabajo, son los llamados sistemas multi-robot. Es decir, los formados por dos o más robots que comparten espacio de trabajo e información. Lo hacen de forma directa o indirectamente, para poder cumplir con los objetivos. Hay una gran variedad de aplicaciones en las que se usan estos sistemas, y las principales son:

- Operaciones de vigilancia, como la realizada por un grupo de drones sobre una zona para la detección de incendios [1] O la obtención de imágenes aéreas dentro de naves industriales mediante drones, evitando los riegos de la vigilancia en persona [2] .

- Búsqueda, exploración y rescate. Por ejemplo, búsqueda de submarinos en alta mar, búsqueda y rescate en desastres naturales, o búsqueda de personas extraviadas en entrono hostiles como la alta montaña [3]

- Tareas de mapeo que consisten en introducir a un pequeño grupo de robots en zonas de las que se desee obtener un mapa. En especial en zonas de difícil acceso o peligroso para los seres humanos. Un ejemplo claro es el caso de la agencia espacial europea que ya ha diseñado un robot para poder explorar las cuevas de la luna. Si bien por ahora solo hay uno, podrían ser varios robots trabajando en cooperativo [4] .

- Tareas de coordinación para transporte de objetos, clasificación, etc., donde se aplican estrategias multi-robot para clasificar y operar grandes almacenes, como por ejemplo Amazon, una de las empresas más punteras en el uso de estas estrategias [5] .

- Realización de tareas agrónomas, donde la utilización de estas técnicas ayuda a la automatización, eficiencia y sostenibilidad de estas actividades.[6] [6] [6] [6] [6]

- Entretenimiento y publicidad, una de las ramas que más rápido ha crecido, consiste en realizar figuras, palabras o cualquier otra cosa con cientos de drones que se iluminan con luces, con fines de entretenimiento o incluso con fines de publicidad, creando pancartas gigantes. Un ejemplo claro ocurrió en la ciudad de Shanghái donde 1500 drones crearon un código QR para descargar un videojuego [7] Más reciente aún, el espectáculo de drones en la inauguración de los juegos Olímpicos de Tokio 2020 [8] a cargo de una de las empresas más punteras en este ámbito, INTEL.

1.2 Contexto

En este trabajo se van a abordar simulaciones multi-robot con 2 tipos de robots. Unos robots tipo ***reactivo*** (las ovejas), que reaccionan a la presencia de otros robots. Otros robots tipo ***guiado*** (los perros) que serán los encargados de llevar a los anteriores hasta su objetivo. Se considerará un problema de pastoreo no cooperativo, al igual que los perros se coordinan para mover el rebaño.

Se va a proponer una estrategia de control basada en el artículo [9] donde los perros se coordinarán entre sí para encerrar parcialmente a la manada y usar la fuerza de repulsión para poder conducir el rebaño por el entorno. Las estrategias de control se realizarán paso a paso, donde se empezará con una oveja, se simulará el comportamiento del equipo de robots mediante un robot cinemático monociclo, y luego se irá introduciendo la estrategia de control de los perros.

Se obtendrá en primer lugar una estrategia de control lineal simple para el sistema monociclo, con el que se modela el conjunto oveja-perros. Se añadirá la ley de control de retroalimentación para llevar a los perros a sus posiciones ideales. Después esa estrategia se generalizará para un número arbitrario de perros y 1 oveja. Por último, también se generalizará a un número arbitrario de ovejas.

A lo largo del trabajo se va a utilizar la terminología perro-oveja, pero, los “perros” son agentes robóticos de los cuales tenemos control, mientras que las “ovejas” pueden ser animales de pastoreo biológicos u otros robots que se comportan como animales de pastoreo.

Las ovejas responden a los perros con un campo potencial repulsivo que se usa comúnmente para modelar la respuesta de los animales de pastoreo a las amenazas percibidas. La estrategia de control que se plantea será útil en el control de vida silvestre, así como en otras aplicaciones [10] . Dentro del mundo de la ganadería se ve un claro futuro, pues con el uso de drones se pueden realizar las tareas de agrupamiento más rápido y eficiente, además de ser más seguro, por ejemplo, en el caso de Australia, al haber grandes extensiones de terreno usan helicópteros para reagrupar animales. Se trata de una técnica muy peligrosa debido a la baja altitud a la que tienen que volar los aparatos [11] .

A parte de para controlar animales, también se podría usar esta estrategia de control para evacuaciones de emergencia en multitudes humanas.

Se considerará este trabajo como un problema de múltiples robots no cooperativos, ya que el objetivo de los perros es guiar a las ovejas, pero las ovejas no están activamente inclinadas ni se oponen a ser dirigidas, todo en un entorno plano.

Para modelar la dinámica del rebaño de robots no cooperativos, además de los términos repulsivos entre los robots, se consideran también términos atractivos para proporcionar cohesión. Estos términos se usan habitualmente en las propuestas de pastoreo robótico no cooperativo, como en el trabajo pionero de Vaughan [12] , en el que se utiliza un solo robot para pastorear patos en un campo de experimentos de diseño especial.

En este trabajo, para modelar la dinámica de la manada, utilizaremos el artículo del ICRA 2013 [13] . En este artículo, se propone un algoritmo de agregación de enjambres descentralizado novedoso para sistemas de múltiples robots con una evitación de obstáculos integrada, aunque esta última no se realizará. En este marco, la interacción entre robots se limita a su vecindario de visibilidad, es decir, robots que están dentro del rango de visibilidad de los demás.

1.3 Objetivos

Los objetivos del trabajo consisten en implementar técnicas de pastoreo multi robot, basándonos en dos artículos científicos presentado en ICRA (2015 y 2013)[9] [13] . La intención es poder simular el movimiento de animales no cooperativos ("sheeps"), a través de modelos de repulsión y poder controlar estos, realizando las acciones a través de robots cooperativos ("dogs"). Se desea obtener así una mayor eficiencia en las labores de pastoreo y otras actividades. Se consideran técnicas con uno o más robots no cooperativos y con varios robots cooperativos.

Para el desarrollo del trabajo, partimos del documento realizado por Alyssa Pierson y Mac Schwager. Se modela el movimiento propuesto para el robot no cooperativo, y se implementan, ajustan y validan las estrategias de control de un equipo con dos robots cooperativos en un entorno de simulación. Después se consideran equipos con más miembros y acabamos introduciendo la dinámica de rebaño.

Para la realización se va a utilizar PyCharm, que es un entorno de desarrollo integrado que se utiliza para programación informática, donde programaremos en lenguaje Python. En concreto, los objetivos específicos que se plantean en este trabajo son los siguientes:

- Lectura y comprensión del artículo científico en el que se basa el trabajo.

- Implementación del modelo de repulsión básico para un "sheep".

- Modelo de monociclo unificado para el conjunto de robots.

- Controladores para los robots "dogs" y ajuste de parámetros.

- Modelos de repulsión-atracción de los "sheep" para modelar el rebaño "herd".

- Simulaciones y pruebas finales y elaboración de la documentación.

1.4 Organización del documento

Este documento se organizará de la manera que se explica a continuación:

- Capítulo 1: El capítulo en el que se encuentra, donde se introducirá y se pondrá en contexto el proyecto, así como los objetivos a alcanzar.

- Capítulo 2: En este capítulo se describe la problemática a resolver y algunos detalles sobre la realización.

- Capítulo 3: Implementación del sistema monociclo con una única oveja.

- Capítulo 4: En este capítulo se incluye la participación de los perros dentro del sistema.

- Capítulo 5: En este capítulo se realizan varios modelos de agrupación de rebaño de ovejas.

- Capítulo 6: Se realizan varias simulaciones y pruebas del algoritmo final.

- Capítulo 7: Se desarrollan las conclusiones y se habla sobre las posibilidades de los proyectos que sucedan a éste.

- Bibliografía

- ANEXO: Se encuentra el código desarrollado.

Capítulo 2

2. Descripción del problema

En este trabajo, se han ido desarrollando los diferentes componentes que integran el sistema final, realizando pruebas de validación de cada componente por separado para facilitar la implementación y la comprensión de cada proceso.

La implementación del algoritmo se realizó siguiendo las distintas fases de esté tal como se muestra en el artículo [9] , desarrollando las partes por separado, e integrándolas una vez se ha comprobado su correcto funcionamiento.

El objetivo es llevar a la oveja o al centro del rebaño a una región alrededor del objetivo (goal) a través de fuerzas de repulsión ejercidas por los perros, determinando las leyes de control para los perros de forma que el guiado se produzca de forma efectiva. Todo está desarrollado en un plano 2D. También existirán unas fuerzas de atracción repulsión entre las propias ovejas de tal forma que se comporte como un rebaño [13] .

Se usará la nomenclatura de perro (dog, d) y oveja (sheep, s) para referirnos a los dos tipos de robots, los que se mueven en base al controlador, y los que se mueven en función del movimiento de los otros a través de un campo de repulsión. Las posiciones 2D del dog j y de la sheep i se representa respectivamente con d_j y s_i. En este trabajo se decide que las posiciones de los perros sea de forma circular entorno a la oveja o centroide del rebaño, siendo el radio r constante y suficientemente grande como para poder albergar todo el rebaño.

En las figuras 2.1 y 2.2 se puede ver de forma esquemática la colocación de los perros respecto a las ovejas, así como las fuerzas de repulsión (f_r) y de atracción (f_a).

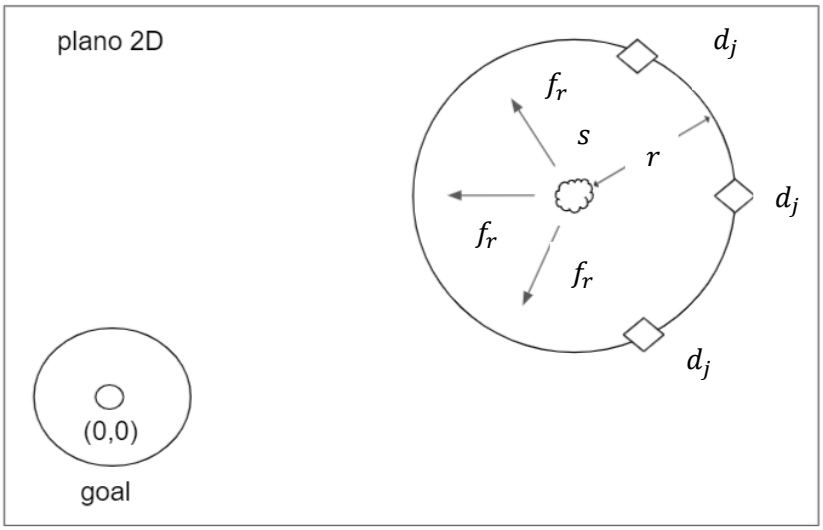

Figura 2.1: esquema general sheep-dogs

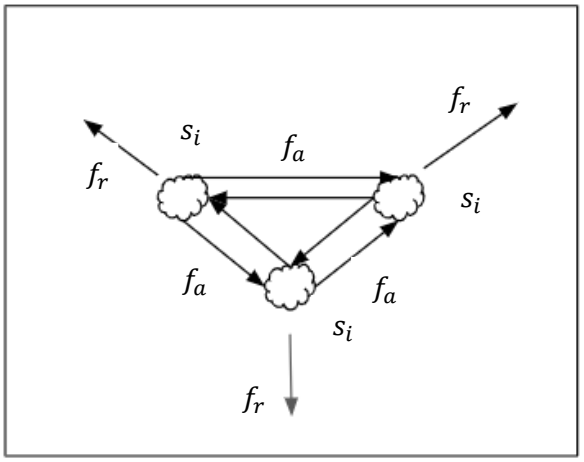

Figura 2.2: esquema fuerzas de rebaño

Capítulo 3

3. Modelo con una única oveja y modelo monociclo del sistema

Se empezó por implementar el movimiento de una única oveja, en la que se introducirá el movimiento de monociclo tal como está descrito en el artículo [9] y el modelo de repulsión encargado del movimiento de estas una vez estén ya incluidos los perros.

3.1 Modelo repulsión

La primera parte que se va a implementar, es el modelo de repulsión entre la oveja y el perro, aunque no se utiliza hasta la última parte, se decidió implementarlo y validar su correcto funcionamiento como campo de repulsión para el movimiento de las ovejas.

Dicha implementación es fácil, pues únicamente es implementar la fórmula:

$$\dot{s}_i = \sum_{j=1}^{m} \frac{-(d_j - s_i)}{\|d_j - s_i\|^3} \qquad (1)$$

Donde

j = nº de perro

i = nº de la oveja

d_j = posición del perro nº j

s_i = posición de la oveja nº i

$\dot{s}_i$ = dinámica de la oveja

Con la ecuación (1) representamos una posible dinámica de la oveja debida a la repulsión que tienen estas sobre la presencia de los perros y su distancia. Este modelo está utilizando un campo de potencial artificial común en modelos de animales de pastoreo biológico, como en bancos de peces, aves, manadas de mamíferos y otros enjambres [14] .

Se realiza una simulación sencilla donde se introduce un perro y una oveja, el perro se mueve en línea recta hacia delante, mientras que la oveja situada delante del perro se irá moviendo en relación a la dinámica impuesta por el campo de potencial.

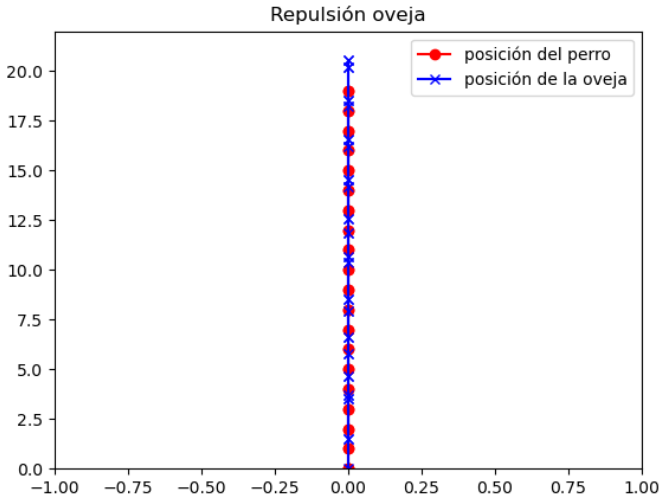

Figura 3.1: Posición x, y perro y oveja

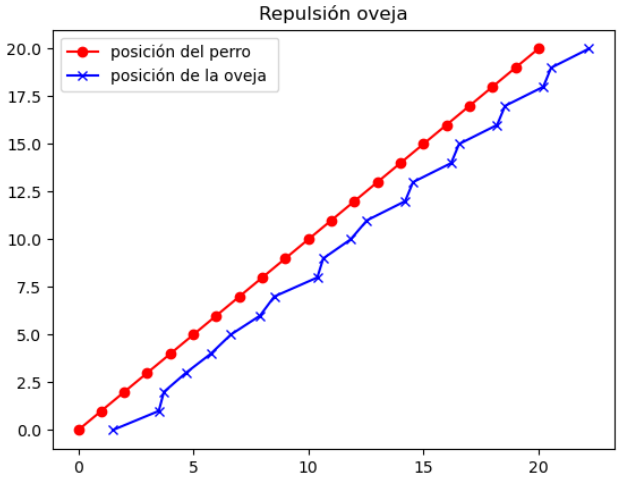

Figura 3.2: Coordenadas y perro-oveja evolución con el tiempo

Como se puede observar en estas primeras gráficas, la posición de los perros avanza de uno en uno, tal como se ha indicado, mientras que la posición de la oveja varia de distinta forma, dependiendo a qué distancia del perro se encuentra. La posición inicial del perro es $(0,0)$ mientras que la de la oveja es $(0,1)$.

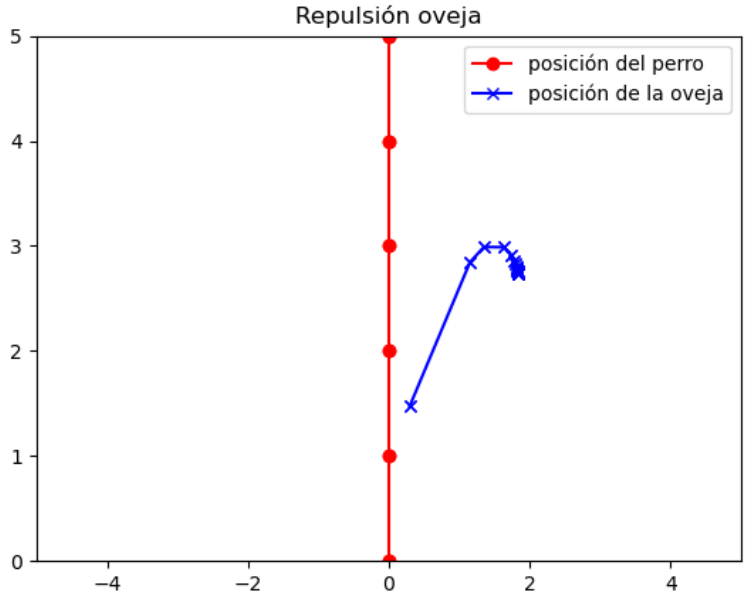

Figura 3.3: posición x, y perro y oveja

En esta gráfica se puede ver mejor el funcionamiento de este campo de repulsión, donde se puede observar como la oveja se aleja de la posición del perro y una vez este se va alejando, el movimiento de la oveja es cada vez más lento, pues a medida que el perro se aleja, la oveja tiende a quedarse quieta. En este caso la oveja empieza en $(0.2, 1)$, de ahí que se desvíe.

Con esta última gráfica podemos suponer que la implementación del modelo de repulsión es correcta, aunque se tendrá que volver a validad una vez se incorpore en el algoritmo final.

3.2 Controlador proporcional del Offset-Point ($\dot{p}$)

En referencia al trabajo de Pierson, (Proposición 1)[9] se demuestra que, si los dogs se colocan de forma circular alrededor del sheep, entonces el conjunto oveja-perros se comporta como un monociclo. Para controlar este sistema monociclo, se propone un controlador que conduce hasta el origen un punto p, denominado offset-point, situado a una distancia fija del centro del monociclo (Figura 3.4). A continuación, se explica cómo implementar el controlador para el offset-point p. En la siguiente sección, se explica cómo incluir este controlador para guiar al sistema monociclo. Se propone un controlador que conduce un punto de desplazamiento del sistema similar al monociclo hasta el origen. Dada alguna velocidad deseada que controla el sistema ideal hacia la región objetivo, se puede calcular las posiciones deseadas para los perros a lo largo de la circunferencia del círculo. En la sección 4 empleamos un controlador de seguimiento para llevar a los perros a las posiciones deseadas. Para nuestro análisis, asumimos

que la dinámica del sistema similar a un monociclo ideal es significativamente más lenta que la dinámica para llevar a los perros a sus posiciones ideales.

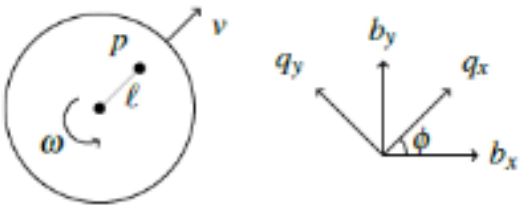

Figura 3.4: Modelo de vehículo monociclo

El controlador proporcional está formado por una constante k:

$$\dot{p} = -kp \quad (2)$$

Para ver el funcionamiento de este controlador, se realiza un pequeño programa, donde se coloca un punto en una posición determinada, se calcula la posición del punto p a una distancia l de éste, en dirección al objetivo, que en este caso será el $(0,0)$ y se realiza una pequeña simulación donde previamente se ha tenido que discretizar el controlador para poder ser utilizado.

$p_{punto}(t) = -K * p(t)$ donde $p_{punto}(Kt)$ aproxima a $(p(k+1)T) - p(kT)/T$ (3)

por lo tanto, queda:

$$(\mathrm{p}((\mathrm{k}+1)\mathrm{T}) - \mathrm{p}(\mathrm{kT}))/\mathrm{T} = -\mathrm{K} * \mathrm{p}(\mathrm{kT}) \quad (4)$$

Se quita kT para trabajar con la iteración k y queda finalmente:

$$\mathrm{p}(\mathrm{k}+1) = \mathrm{p}(\mathrm{k}) - \mathrm{K} * \mathrm{T} * \mathrm{p}(\mathrm{k}) \quad (5)$$

Una vez implementado en la simulación, se puede observar en la gráfica 3.5 cómo funciona el control proporcional modificando la velocidad del sistema en función de la distancia a la que se encuentra del objetivo, por lo que conforme se acerca al objetivo, la velocidad disminuye que es lo que se quiere para llevar las ovejas a su objetivo.

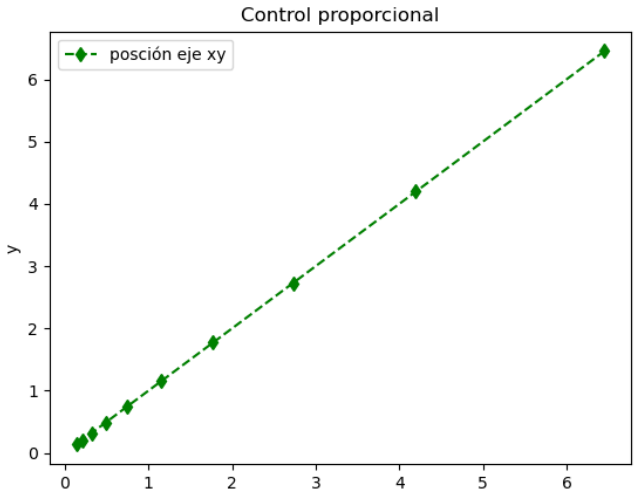

Figura 3.5: Trayectoria punto p, de posición inicial (10,10) a objetivo

3.3 Control de sistema monociclo mediante Offset-Point

Una vez obtenido el controlador del offset-point p, se aborda el problema de controlar el sistema monociclo guiándolo al origen. Para ello, se busca el rumbo con Φ^* (phi deseada) y la velocidad v^* ideal. Se empezará diseñando la dinámica del monociclo de la que se ha estado hablando anteriormente.

En algunas de las fórmulas y expresiones del documento [9] , había cierta ambigüedad sobre si se debía usar la orientación actual del monociclo o su orientación deseada. Por tanto, se implementaron y evaluaron diferentes alternativas para esta sección, que permitieron aclarar estos conceptos y garantizar que en las siguientes secciones se usan las expresiones adecuadas para controlar el sistema.

Para la realización del primer programa se le da un valor inicial al ángulo phi, siendo este ángulo el que marca la dirección del sistema monociclo sheep-dog respecto a la referencia mundo, por ejemplo, la oveja se encuentra en el punto (10,0) y el ángulo phi (Φ) es 90º, por lo que el punto p está situado en $(10, l)$ siendo l la distancia del punto s al offset-point p como se ve en la figura 3.6.

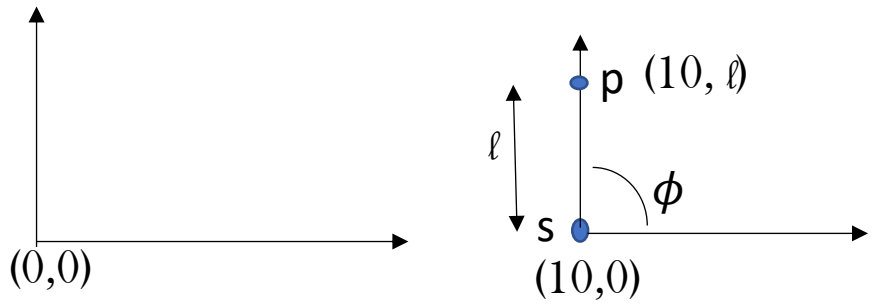

Figura 3.6: Posición punto s y punto p

El modelo del monociclo para calcular los valores de velocidad lineal y angular vienen dado por la ecuación (6), donde la derivada de p viene de la ecuación (2). v y ω son las velocidades lineal y angular del monociclo.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\frac{\sin(\phi)}{l} & \frac{\cos(\phi)}{l} \end{bmatrix} * \dot{p} \quad (6)$$

Es importante tener en cuenta que el ángulo phi se refiere a la orientación del conjunto sheep-dogs en coordenadas globales tal y como se muestra en la figura 3.6. Nótese que para llegar a esta conclusión se probaron algunas alternativas para el ángulo phi (este era el ángulo que tenía que recorrer el monociclo para poder colocarse en la dirección correspondiente a su objetivo, en este caso (0,0)). Estas interpretaciones alternativas se comprobaron enseguida que no eran correctas, ya que los valores de ω no eran los correctos, pues cuando se llegaba a un ángulo $\phi = 0$, siguiendo el entendimiento que se había tenido del ángulo phi, esta velocidad ω debería ser 0. En ese momento el monociclo está alineado con la dirección deseada y ya solo tendría que avanzar hacia delante para encontrar el objetivo, pero como pudimos observar esto no sucedía y $\omega \neq 0$. Por lo tanto, se descartó esta otra alternativa.

Una vez se tuvieron claros los conceptos y se entendió el funcionamiento del monociclo, se pasó a simular el modelo.

```
#inicialización datos
s_x = 10
s_y = 0
phi = 90 # valor de phi en grados
phi = phi * np.pi / 180
s = np.array([s_x, s_y])
l = 0.1   #distancia de s a p
p = np.array([0, 0])
p = s + [np.cos(phi) * l, np.sin(phi) * l]

#variables de control
k = 2

#parámetros de simulación
dt = 0.01
MAXITER = 500
i = 0
```

Una vez introducidos los datos iniciales, se pasa a realizar el bloque de simulación.

```
while i < MAXITER:
    p_k1 = - k * p

    #controlador y modelo cinemático
    m_cinema = ([np.cos(phi), np.sin(phi)], [-np.sin(phi) / l, np.cos(phi) / l])
    v,w = m_cinema @ p_k1

    #simulación del monociclo
    s_x_k1 = s_x + dt * np.cos(phi) * v
    s_y_k1 = s_y + dt * np.sin(phi) * v
    phi_k1 = phi + dt * w
    p = np.array([s_x_k1 + np.cos(phi_k1) * l, s_y_k1 + np.sin(phi_k1) * l])


    phi = s_tita_k1
    s_x = s_x_k1
    s_y = s_y_k1

    i = i + 1
```

Como se observa, la simulación comprende los siguientes pasos: calculamos la dinámica del punto p, obtenemos el modelo cinemático y los valores de la velocidad lineal y velocidad angular, y una vez obtenidos calculamos el movimiento del punto s con dichas velocidades y ángulo phi. Como resultado, al modificarse la posición y orientación del monociclo, cambia también la posición del offset-point p.

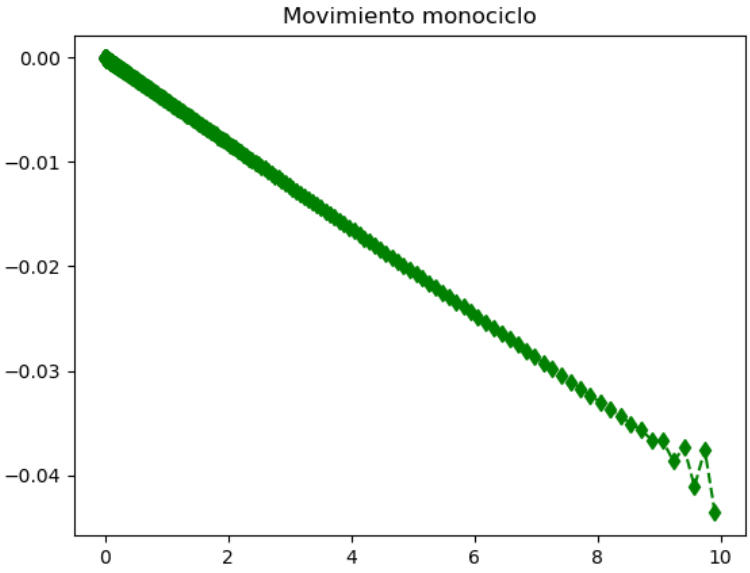

Figura 3.7: movimiento punto p (px, py)

Como se puede observar en la figura 3.7, el control funciona correctamente, aunque oscila un poco al principio. El principal efecto de esta oscilación se ve mejor en la siguiente gráfica.

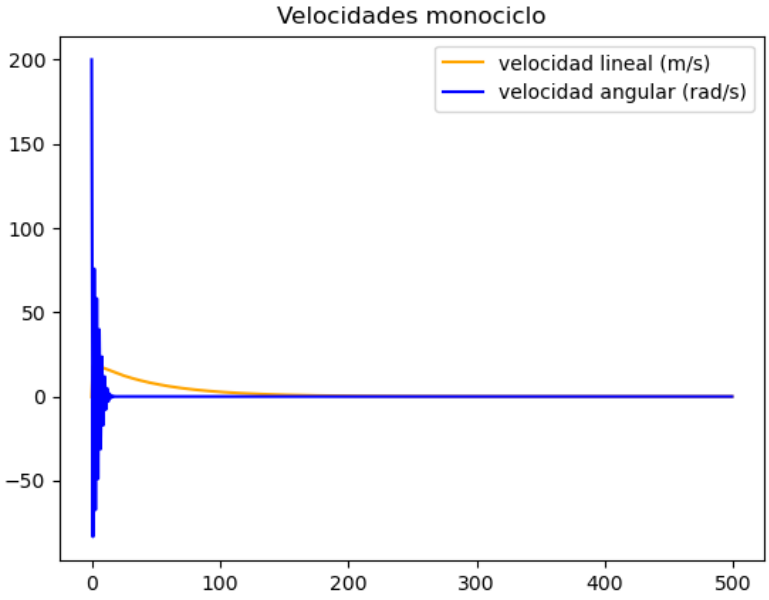

Figura 3.8: velocidades del monociclo

Como podemos observar en la figura 3.8, se pueden ver las velocidades lineal y angular durante la simulación. Ahora se puede afirmar, que la oscilación inicial es debida al alto valor de velocidad angular que hay en el principio de la simulación. Para generar movimientos más suaves, se limita la velocidad angular. El valor concreto seleccionado se podría elegir en aplicaciones reales teniendo ya ahí sí en cuenta el tipo de sistemas que son, tipo de movimientos, etc.

3.3.1 Ajustes del controlador

Como se observa en la figura 3.8 existe una oscilación inicial que podría generar problemas en el control, por lo que se realizaron una serie de ajustes para adecuar el modelo a una simulación más real.

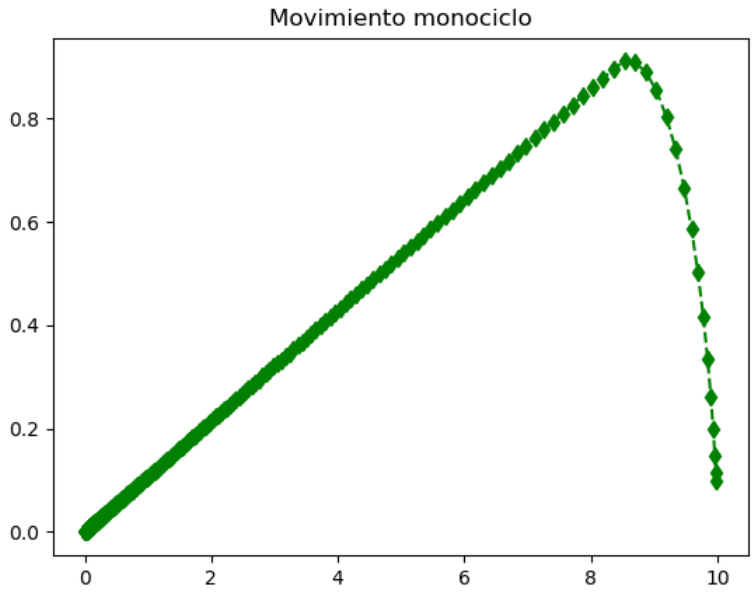

Figura 3.9: movimiento punto p (px, py) sin oscilación

Como se puede observar ahora en la gráfica, la posición del monociclo no es tan oscilante, aunque se observa que se aleja del punto (0,0) llegando a 0.8, esto es debido a que la dinámica del monociclo está empezando con un ángulo phi de 90 º, esto quiere decir que los primeros movimientos se realizarán hacia arriba, de ahí que aparezca esa curva.

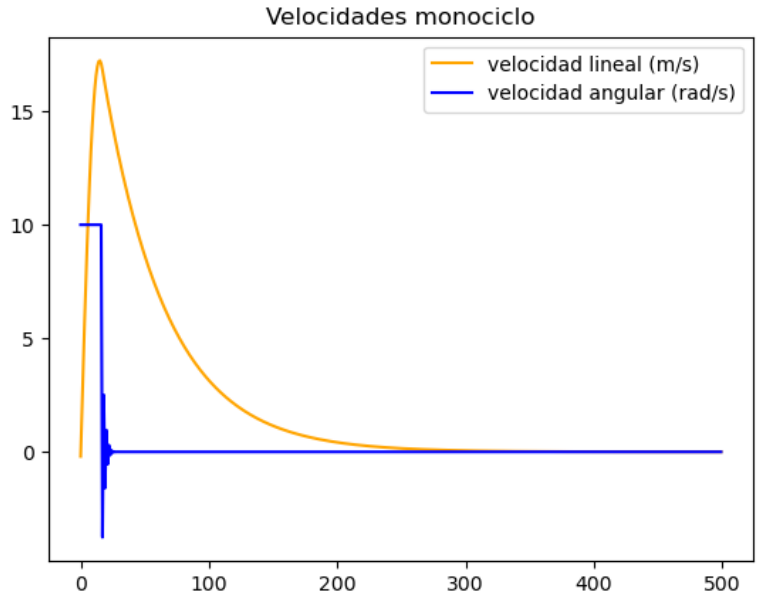

Figura 3.10: velocidades del monociclo

Ahora se pude ver en la figura 3.10 como la velocidad angular satura durante el principio al valor elegido de 10, y como ahora oscila bastante menos que antes. Aunque la mejoría es evidente, se puede conseguir mejorar el comportamiento para que la caída de la velocidad angular no sea tan brusca, y que esta velocidad angular no oscile en ningún momento.

Para conseguir dicha mejora se modifica el controlador, teniendo en cuenta que ω es proporcional a p (mayor cuanto más lejos del objetivo está). Esta decisión se toma en el artículo para demostrar estabilidad, pero que el efecto práctico es mejorable. Puede tener sentido para v, para que vaya más rápido si está lejos del objetivo. Pero para ω, se prueba aquí una alternativa: se usa el vector unitario derivada (p) en general, y sólo se usa la versión proporcional cuando se está muy cerca del objetivo (distancia menor que 1). Con esto conseguimos normalizar el controlador y obtener otro que no dependa de la distancia, el cual se aplicará únicamente para el cálculo de ω.

```
if np.linalg.norm(p_k1) < 1:
    p_nor = p_k1
else:
    p_nor = p_k1 / np.linalg.norm(p_k1)

p_k2 = k * p_nor #normaliza

_,w = m_cinema @ p_k2
```

Por último, y para poder tener un parámetro manejable con el que poder ajustar el sistema, se introduce una constante $k\omega$. Esta constante estará multiplicando a ω justo en el momento de ser aplicada para calcular el valor de phi en la simulación. Con esto conseguimos realizar la dinámica de giro más rápida o más lenta según nos convenga, solo modificando un parámetro.

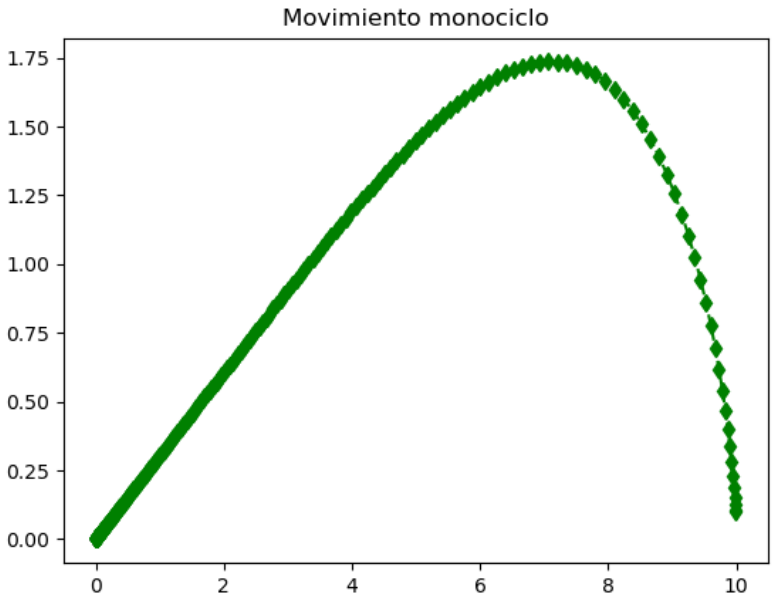

Figura 3.11: movimiento punto p (px, py) modificación de ω

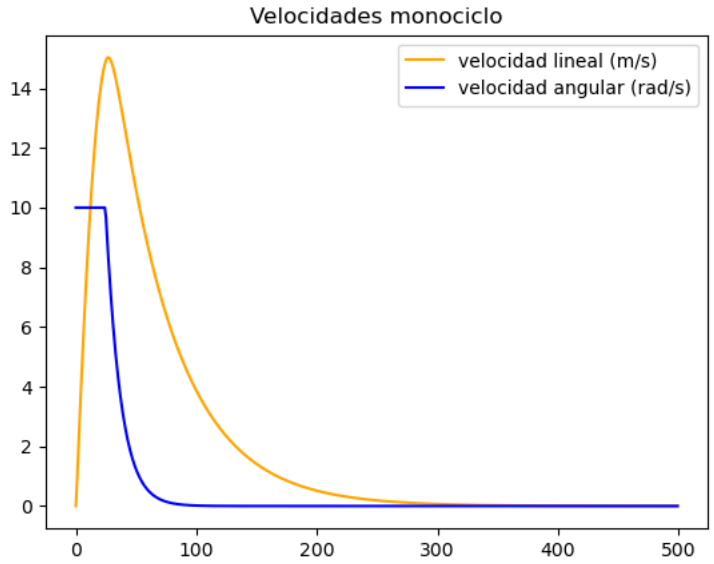

Figura 3.12: velocidades del monociclo sin oscilaciones

Como se puede ver, con las modificaciones realizadas, se consigue que phi varíe más lento y con ello que la curva sea más abierta, como se observa en la gráfica de posición, donde ya no hay una curva tan cerrada como antes.

En la gráfica de velocidades se puede corroborar que la velocidad angular ya no oscila. Satura al principio y desciende sin oscilar hasta llegar a 0 donde se mantiene durante casi todo el recorrido.

Una vez incorporados estos retoques se realizaron varias pruebas para verificar el buen funcionamiento del monociclo, y se observó un comportamiento llamativo justo cuando el valor de phi = 0, es decir cuando la dinámica del monociclo está apuntando en la dirección opuesta al objetivo. En este caso particular los resultados de las gráficas son los siguientes.

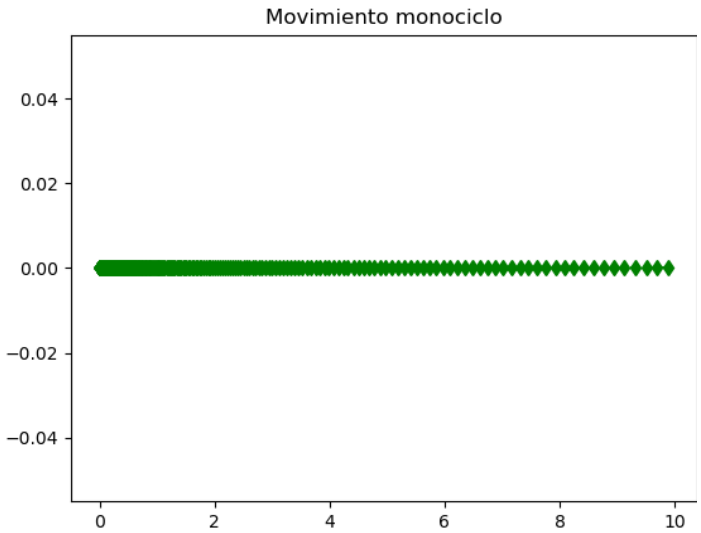

Figura 3.13: movimiento punto p (px, py)

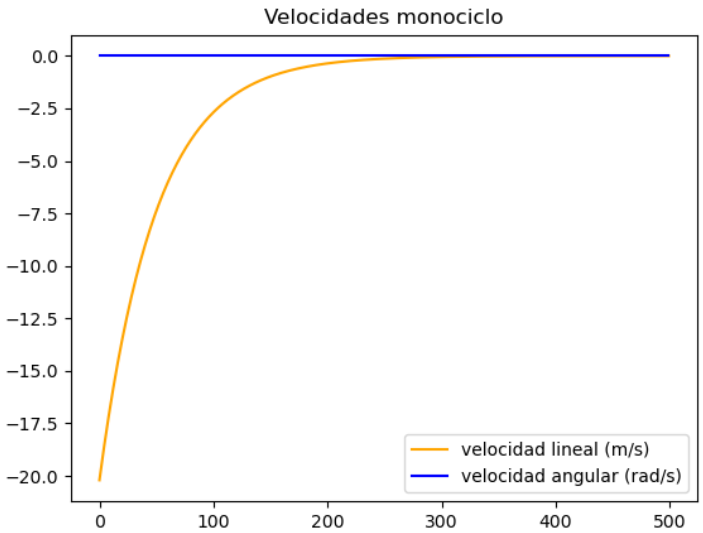

Figura 3.14: velocidades del monociclo

Como se observa en la gráfica de posición, el monociclo va directo al objetivo sin realizar ninguna curva, y si nos fijamos en la gráfica de velocidades, vemos como la velocidad angular es siempre 0 y que la velocidad lineal empieza siendo negativa hasta llegar al 0. Esto quiere decir que el monociclo va hacia el objetivo, pero hacia atrás. Dado que en el caso que se aborda, el monociclo representa el conjunto sheep-dogs, se toma la decisión de modificar su comportamiento para asegurar que siempre se mueve hacia delante.

Para solucionar esta problemática, bastará con obligar a que la velocidad lineal sea solo positiva. En este caso, se decide poner la velocidad lineal v a cero si ésta es negativa, permitiendo al monociclo girar sobre sí mismo. En el caso concreto de que ω sea también 0 (como por ejemplo en el anterior), se le da un valor para que pueda girar en el momento que la velocidad es negativa, evitando que el monociclo se quede parado para siempre.

```
if v < 0:
    v = 0
    if w == 0:
        w = 1
```

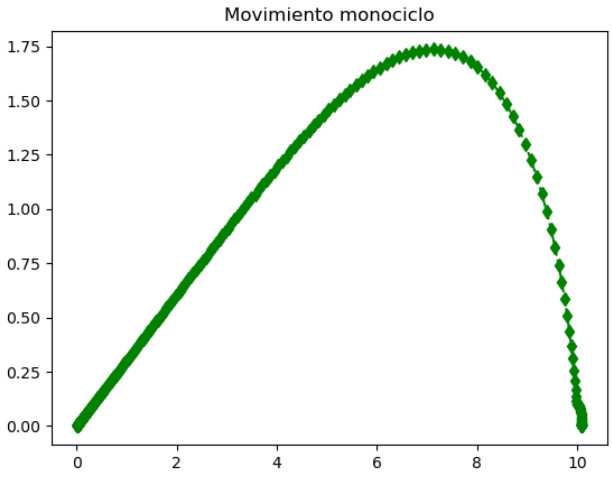

Figura 3.15: movimiento punto p (px, py)

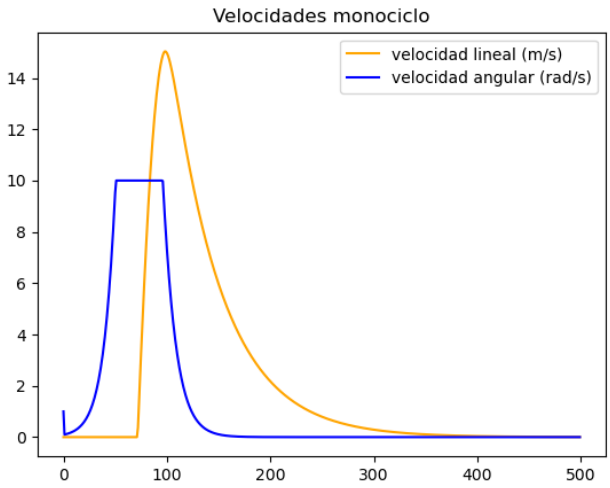

Figura 3.16: velocidades monociclo

Como se ve ahora en las gráficas, al principio de la simulación hay un momento donde solo existe velocidad angular y la velocidad lineal es 0, hasta que llega un momento donde empieza a moverse y realiza un movimiento muy parecido al anterior. Ahora se podría decir que funciona, pero se tiene un giro brusco al principio donde se pasa de 0 a 90º en muy poco recorrido, lo que tampoco será recomendable para lo que viene más adelante, por lo que hay que abrir esa curva para que no sea tan cerrada.

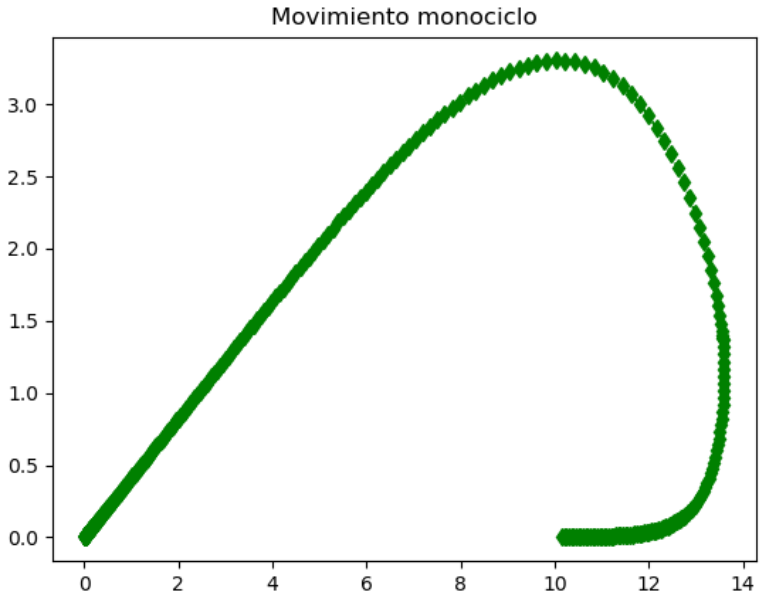

Figura 3.17: movimiento punto p (px, py) final

Cambiando los datos de v y ω dando un valor de velocidad en vez de que sea 0 cuando la velocidad es negativa, se consigue que el giro del principio ya no sea tan brusco. Aunque todos estos parámetros se irán modificando a lo largo que avanza la programación y se van incluyendo más partes.

Capítulo 4

4. Movimiento una oveja y varios perros

Una vez se ha desarrollado y ajustado el control del sistema sheep-dogs monociclo basado en offset-point, el siguiente paso es la introducción de los perros. En una primera parte se introducirán de modo pasivo, es decir, sin que la posición del sheep se vea afectada por la presencia de perros, para verificar su correcta colocación alrededor del sheep. En la segunda fase, se introducen ya en modo activo, es decir, generando repulsión sobre la oveja. En esta segunda fase, se sustituye el modelo ideal del monociclo por el comportamiento de repulsión de las ovejas (sección 3.1) y se implementa un controlador para llevar los perros a la posición deseada.

4.1 Movimiento con perros pasivos

El primer paso para la introducción de los perros, es el cálculo de su posición deseada tal como se indica en la elaboración del algoritmo en el documento de Pierson, A. y Schwager [9] .

Para empezar, se va a suponer que todos los perros están situados a una distancia r de la oveja, y separados un ángulo que depende de la velocidad deseada del monociclo v, velocidad que sale de la ecuación (6). Esta relación viene dada por la ecuación (7).

$$v = \frac{\sin\left(\frac{m\Delta}{2-2m}\right)}{r^2 * \sin\left(\frac{\Delta}{2-2m}\right)} \quad (7)$$

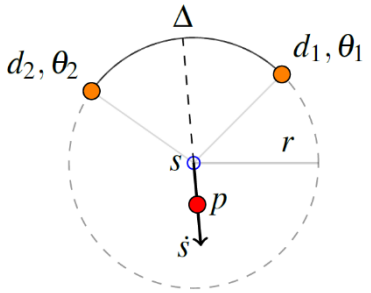

Figura 4.1: Configuración 2 perros y una oveja

De esta ecuación obtenemos una posibilidad infinita de valores para Δ dado un valor de v, pero el rango de Δ está limitado a $(0,2\pi)$ por lo que, para una velocidad dada, obtenemos su correspondiente Δ.

Resolver la expresión anterior para obtener el valor de Δ dado un valor concreto de v no es trivial. En este trabajo, se ha decidido optar por generar una tabla con los resultados de v variando los valores de Δ grado a grado. Una vez obtenida la tabla, que se realizará al principio del programa, dentro del bloque de la simulación, se comparará en cada ciclo la velocidad del monociclo con las velocidades guardadas en la tabla, y se escogerá el valor de delta emparejado con el valor de velocidad que más se acerque al valor de velocidad que tiene el monociclo en ese momento. La precisión que se obtiene en esta parte será de un grado, pues es la máxima precisión que se ha elegido para los valores de delta.

Una vez se ha obtenido el valor de delta correspondiente a la velocidad del monociclo, habrá que asignar un valor distinto de este a cada uno de los perros que hay en la simulación, pues dependiendo del número que haya estos estarán más juntos o más separados. Dichos valores de delta vienen dados por la fórmula (8).

$$\Delta_j = \Delta * \frac{(2j - m - 1)}{(2m - 1)} \quad (8)$$

Siendo m el número de perros que hay y j el número del perro del que se esta calculando delta en ese momento.

Una vez se tienen los valores de delta adecuados para cada uno de los perros, ya se puede calcular la posición deseada para los perros. En la referencia [9] se da la fórmula (9) que presenta problemas y es por este motivo que se proponen dos versiones alternativas que se discuten a continuación.

$$d_j^* = s + r * \begin{bmatrix} \cos(\phi^* + \Delta_j^*) \\ -\sin(\phi^* + \Delta_j^*) \end{bmatrix} \quad (9)$$

El **primer método** de los dos que se van a probar, está relacionado con las matrices de traslación y rotación generales, especificando para dos perros, donde se introducirán dos matrices homogéneas $GTs = loc(xs_k, ys_k, phi_k)$ que relaciona la posición del objetivo final con la posición y orientación de la oveja (s), y otra matriz sTd que va desde la posición de la oveja, a la posición deseada de los perros.

Las matrices de traslación y rotación son las fórmulas (10) y (11) respectivamente.

$$Tranl(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \quad (10) \qquad Rot(z, \theta) = \begin{bmatrix} \cos\theta & -sen\,\theta & 0 \\ sen\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

Por lo que las matrices que relacionan la posición objetivo con la posición de los perros quedan:

$$GTd1 = GTs * sTd1 = transl(xs_k, ys_k) * rotz(\phi_k) * transl(-r * cos(\Delta j), +r * sin(\Delta j)) \quad (12)$$

$$GTd2 = GTs * sTd2 = transl(xs_k, ys_k) * rotz(\phi_k) * transl(-r * cos(\Delta j), -r * sin(\Delta j)) \quad (13)$$

Con esta configuración se consigue que la posición deseada de los perros este justo posicionada en el lado contrario al que deben ir las ovejas, es decir como si estos perros cerraran el camino para que las ovejas tomen el correcto.

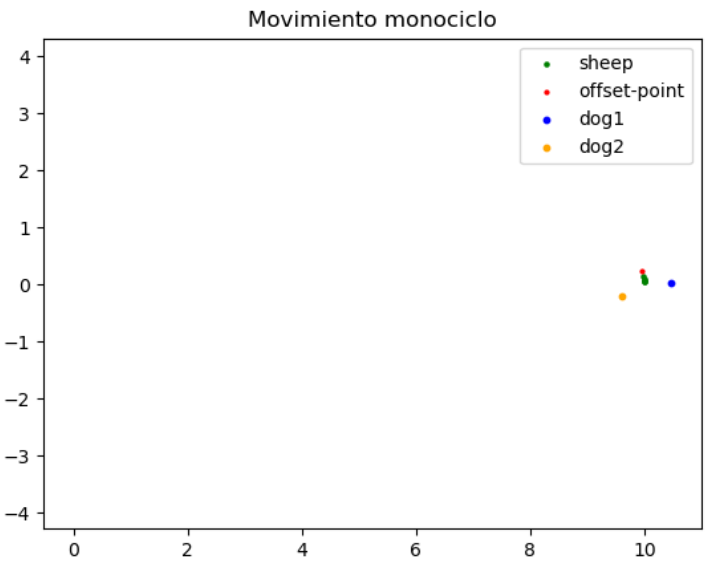

Figura 4.2: Primeras posiciones $i = 5$

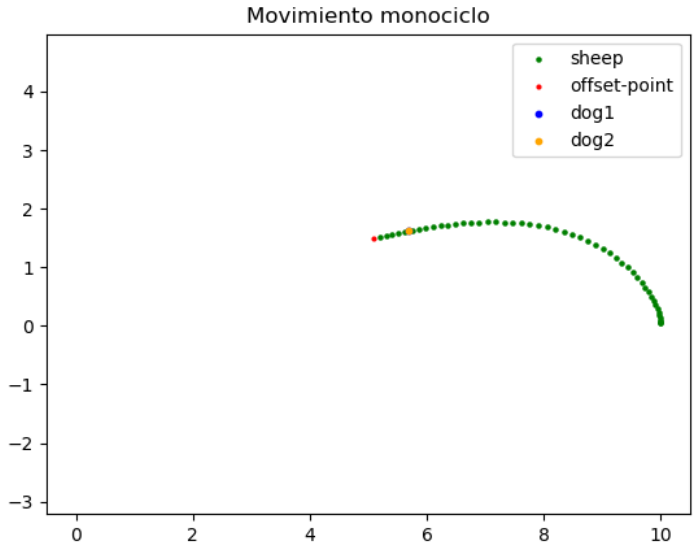

Figura 4.3: Movimiento $i = 50$

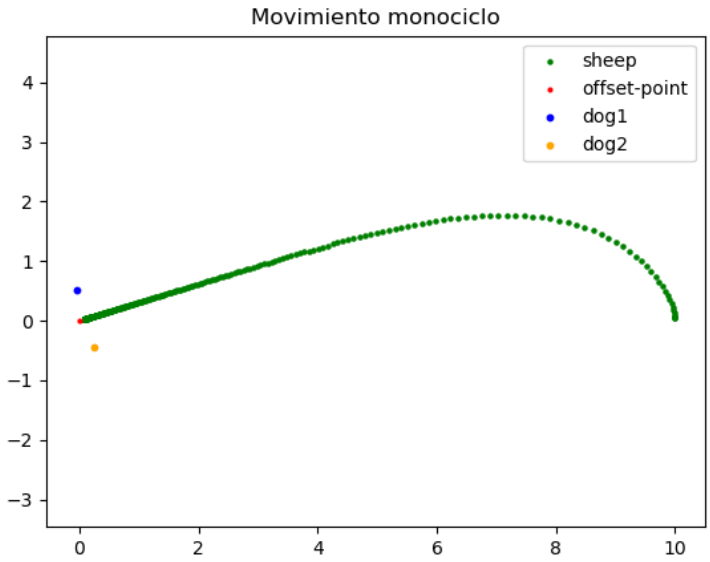

Figura 4.4: Posiciones finales $i = 400$

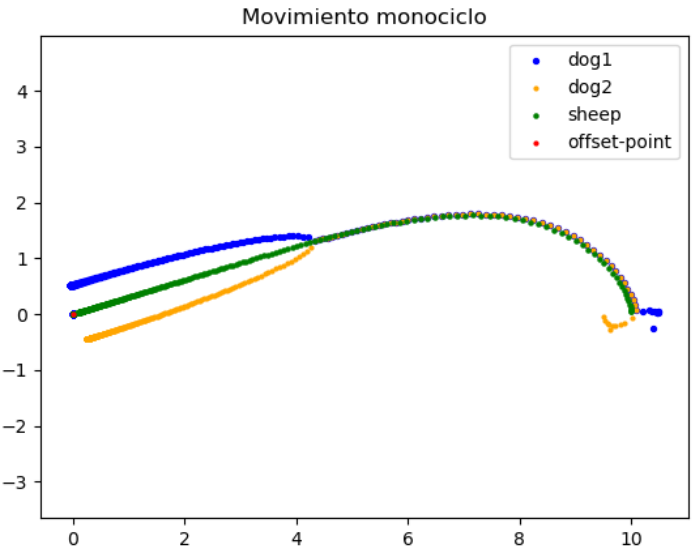

Figura 4.5: Movimiento completo perros

Como se puede apreciar en la figura 4.2, en las primeras posiciones los perros se colocan formando un semicírculo. Una vez se va avanzando, se ve como en la figura 4.3 solo se puede ver un único perro. Esto es debido a que la velocidad del monociclo es máxima, y entonces la separación entre ambos perros es nula para que la repulsión sea máxima.

Una vez se está llegando al objetivo, las posiciones de los perros se abren. Como se puede observar en la figura 4.5, los perros se van separando poco a poco hasta colocarse uno enfrente del otro, anulándose las fuerzas de repulsión entre sí para dejar quieta a la oveja.

En la **segunda** configuración que se probó, se utilizan las mismas matrices que en la anterior configuración, pero ahora, la posición de los perros deseada será de modo de barrera para que la oveja se mueva hacia la posición objetivo en este caso el (0,0).

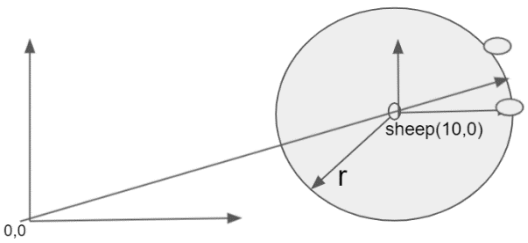

Figura 4.6: Configuración de las posiciones sheep-dog

Para poder realizar este caso, se necesitará obtener el ángulo deseado en todo momento para poder ponerlos en torno a la línea que une el objetivo de la oveja, siempre estando los perros por detrás de ella.

Para poder determinar ese ángulo se usa la formula $atan2$ (14).

$$\phi_{deseada} = atan2(ys_k, xs_k) \qquad (14)$$

Una vez se tiene el ángulo phi deseado las matrices quedan:

$$\begin{aligned} \mathrm{GTd1} &= \mathrm{GTs} * \mathrm{sTd1} = \mathrm{transl}(\mathrm{xs_k}, \mathrm{ys_k}) * \mathrm{transl}(\mathrm{r} * \cos(\phi_{\mathrm{deseado}} + \Delta\mathrm{j}), \mathrm{r} * \sin(\phi_{\mathrm{deseado}} + \Delta\mathrm{j})) \\ \mathrm{GTd2} &= \mathrm{GTs} * \mathrm{sTd2} = \mathrm{transl}(\mathrm{xs_k}, \mathrm{ys_k}) * \mathrm{transl}(\mathrm{r} * \cos(\phi_{\mathrm{deseado}} - \Delta\mathrm{j}), \mathrm{r} * \sin(\phi_{\mathrm{deseado}} - \Delta\mathrm{j})) \end{aligned} \qquad (15)$$

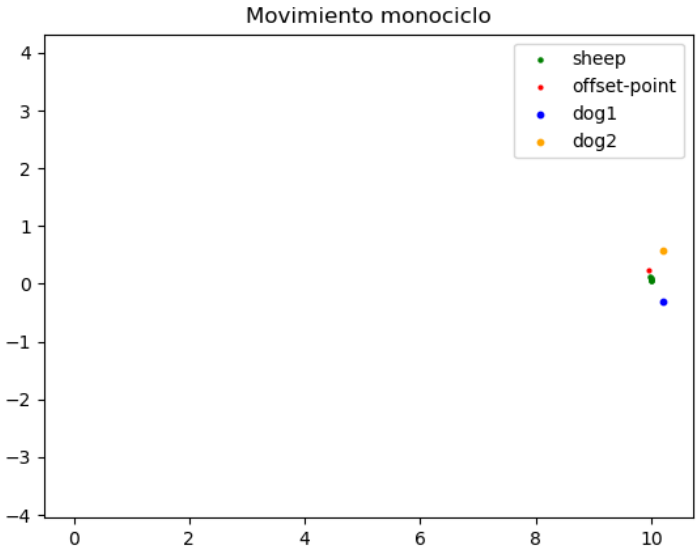

Figura 4.6: Primeras posiciones $i = 5$

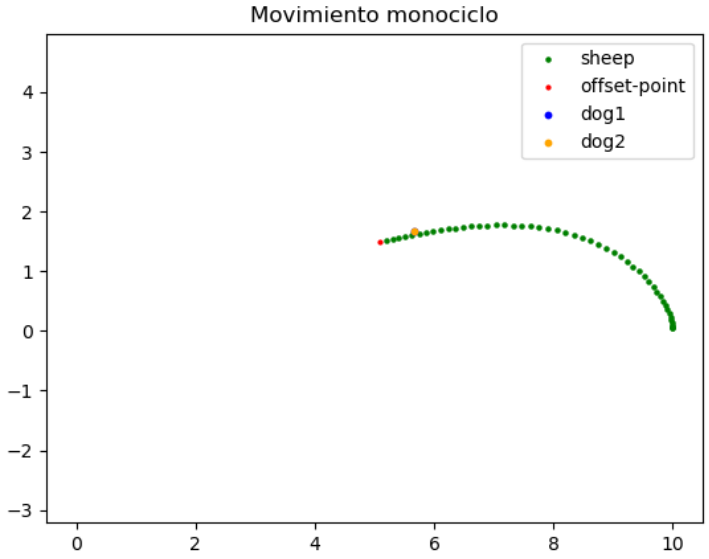

Figura 4.7: Movimiento $i = 50$

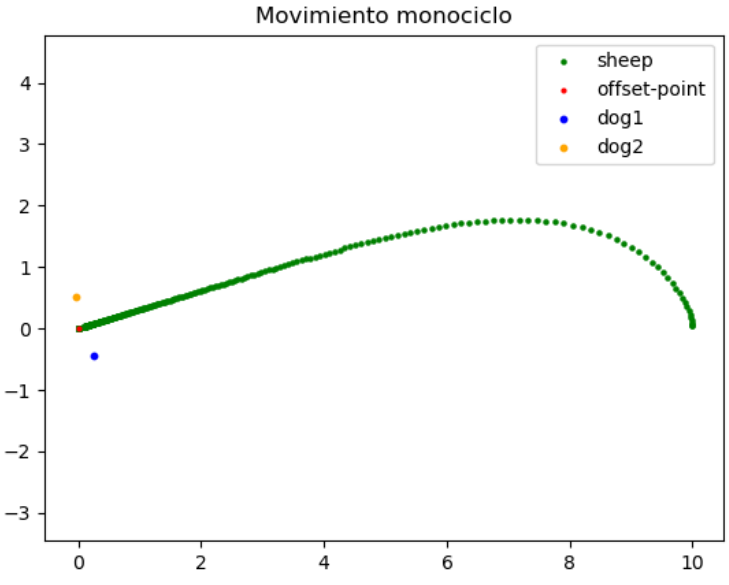

Figura 4.8: Posiciones finales $i = 400$

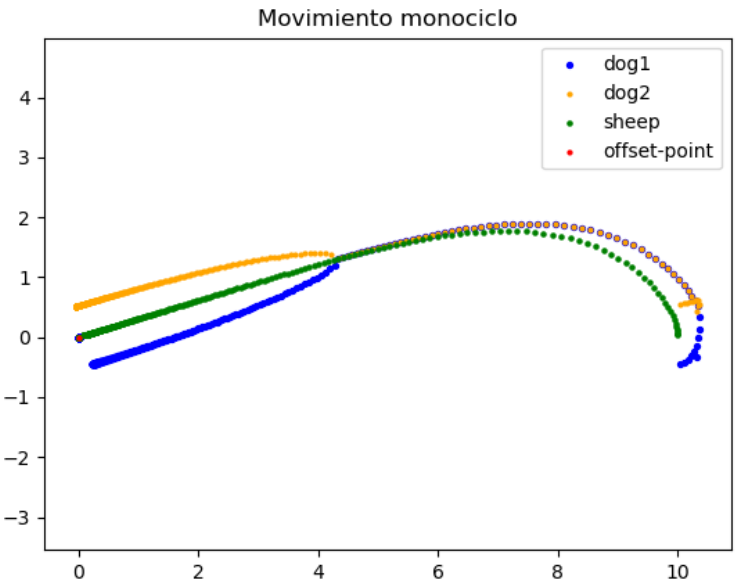

Figura 4.9: Movimiento completo perros

Como se puede ver en la figura 4.6, la posición de los perros es igual a la forma esquemática que aparece en la figura 4.6 donde los perros hacen de barrera en la dirección contraria al objetivo. En la figura 4.7 al igual que en la anterior 4.3 se puede apreciar cómo solo aparece un perro ya que ambos están en la misma posición. La diferencia con el anterior es que están ligeramente por encima del recorrido de las ovejas. Las posiciones finales prácticamente son iguales y en el movimiento completo 4.9 se puede apreciar mejor la diferencia con respecto al anterior, donde al principio la dinámica es algo más lenta al estar más alejados de la oveja.

Una vez realizadas las dos opciones se optó por utilizar esta última a la hora de completar el programa.

4.2 Movimiento con perros activos

Una vez se ha obtenido la posición deseada de los perros en base al movimiento deseado del sistema monociclo del conjunto sheep-dogs, se debe tener en cuenta que el modelo monociclo se usa en el artículo [9] para representar cómo se comporta el conjunto resultante y para proporcionar pruebas teóricas de su correcto funcionamiento. Sin embargo, en el algoritmo definitivo, se cuenta únicamente con la oveja, que reacciona repulsivamente a la presencia de los perros. En cualquier caso, para calcular las posiciones deseadas en las que se deben colocar los perros, se hará referencia a variables relacionadas con el sistema monociclo mencionado en los apartados anteriores.

Ahora el movimiento de los perros estará controlador con la posición deseada y la posición real del perro, como se ve en la fórmula (16).

$$\dot{d}_j = Kd * \left(d_j^* - d_j\right) \quad (16)$$

En el cálculo de la dinámica del perro j, d_j^* es la posición deseada de los perros, d_j la posición real de los perros, es decir la posición en la que se encontraban en el instante anterior, y Kd la constante de movimiento, la que decidirá si la dinámica de movimiento de los perros es más rápida o más lenta y cuyo parámetro se irá ajustando.

Para realizar el movimiento con el controlador mencionado anteriormente, hay que eliminar el movimiento propio del monociclo con el que se movía la oveja, pues ahora este movimiento se realizará a través de la fórmula de la repulsión tratada anteriormente (1).

El controlador del monociclo va seguir funcionando, no realizando el movimiento de la oveja, pero si calculando las velocidades, para poder usarlas luego en el cálculo del ángulo delta (Δ) que marca la separación que hay entre los perros dentro del radio establecido.

```
s_k1 = s - ks * ((d_1 - s) / ((np.linalg.norm(d_1 - s)) ** 3) + (d_2 -
s) / ((np.linalg.norm(d_2 - s)) ** 3)) * dt
```

4.2.1 Asignación posiciones deseadas respecto a posiciones anteriores

Una vez se han realizado varias pruebas se observa un comportamiento anómalo respecto al comportamiento esperado.

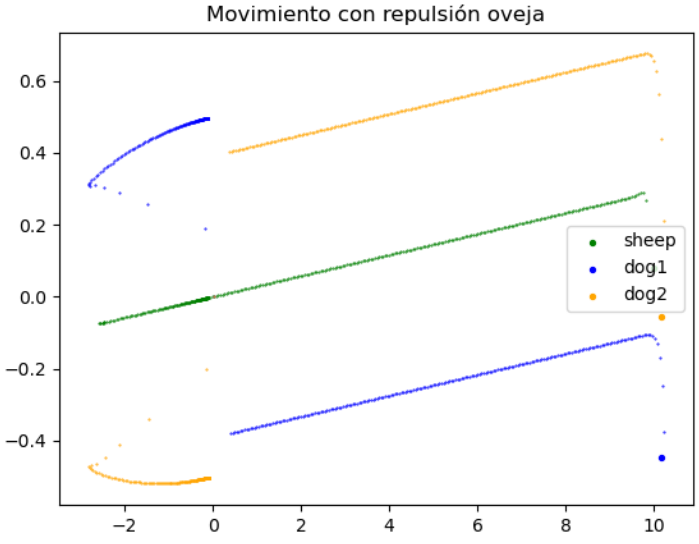

Figura 4.10: Movimiento anómalo de los perros

Como se ve en la figura 4.5 de una de las pruebas realizadas, los perros han llegado al objetivo y la oveja no para en ese punto. Entonces, las posiciones de los perros se cruzan para devolver a la oveja al punto de origen. Esto puede ser perjudicial cuando en vez de una oveja haya varias en rebaño, pues cuando los perros se cruzan para cambiar las posiciones, pasan muy cerca de las ovejas. Podrían dispersar el rebaño de forma que ya no fuera posible volver a juntar las ovejas de nuevo.

Para poder solucionar esta problemática se implementó un algoritmo de vecino más próximo, donde una vez calculada la posición deseada, se mira que perro estaba más cerca de esa posición y al que esté más cerca se le asigna esa posición siguiente. Así podemos evitar que los perros se crucen ya que a esa posición deseada se le asignará el perro que más cerca este.

Una vez implementada la dinámica de la oveja y arreglado el problema mencionado el algoritmo queda como se puede observar en la figura 4.11.

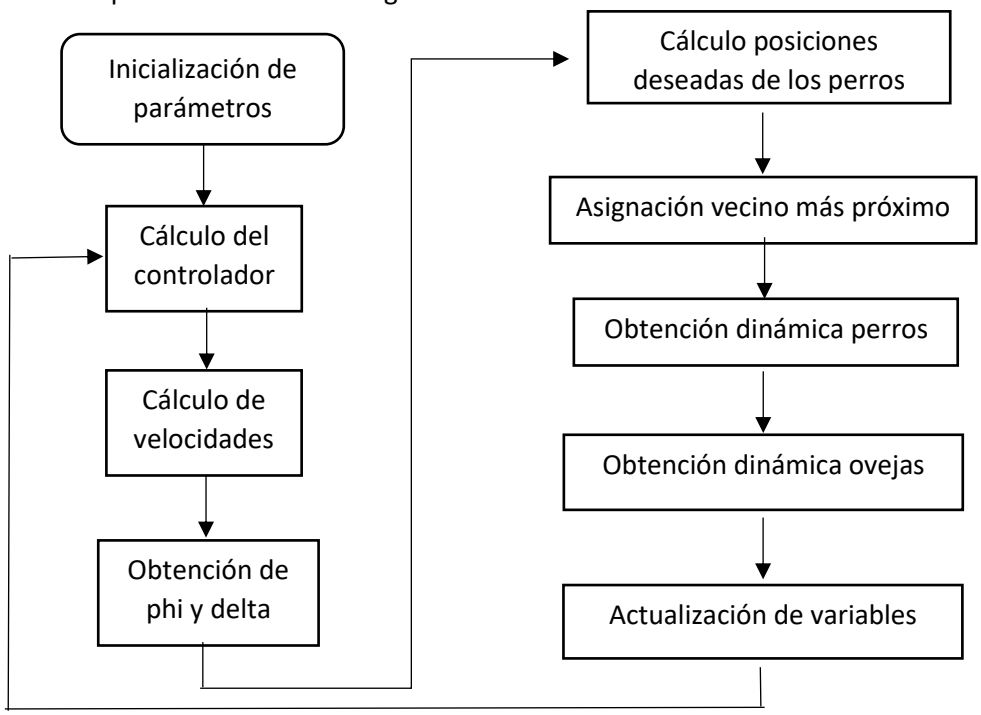

Figura 4.11: Diagrama de flujo del algoritmo

El cálculo del controlador se explica en la sección 3.2 a través de la ecuación (2) y (5). La obtención de las velocidades se indica en la sección 3.3 con el uso de la ecuación (6). La obtencion de los valores de delta(Δ) se explica en la sección 4.1 junto con las fórmulas (7) y (8), y la obtención de phi con la fórmula (14). El cálculo de las posiciones deseadas se explica en esta misma sección y se usará la ecuación (15). Se realizará la asignación al vecino más cercano como se observa en el código (anexo). La dinámica de los perros se calcula a través de la ecuación (16) y la dinámica de la oveja a través de la ecuación (1) como se explica en la sección 3.1.

4.2.2 Generalización a m perros

Una vez realizadas varias pruebas y tras haber comprobado el funcionamiento del sistema, se va a generalizar para poder introducir m perros. Estos se pondrán en torno a la oveja a una distancia predefinida r, en torno a un semicírculo y separados un ángulo en función de la velocidad, igual que antes, pero teniendo en cuanta el número de perros en la simulación para calcular el valor de delta con las fórmulas (7) y (8).

Para realizar una programación que permitiera modificar el número de perros introducidos dentro de la simulación y que esta modificación fuera posible cambiando únicamente la variable m que indica el número de perros, se realizó una programación orientada a objetos.

Para poder realizar la programación orientada a objetos, se ha introducido una clase Dog(). En esta clase se han introducido una serie de atributos, como la posición y la posición deseada, así como todos aquellos atributos necesarios para la realización del algoritmo, y cada uno de los métodos aplicados a la elaboración del algoritmo con el fin de que el código quede lo más limpio posible, permitiendo su reusabilidad (código en el anexo).

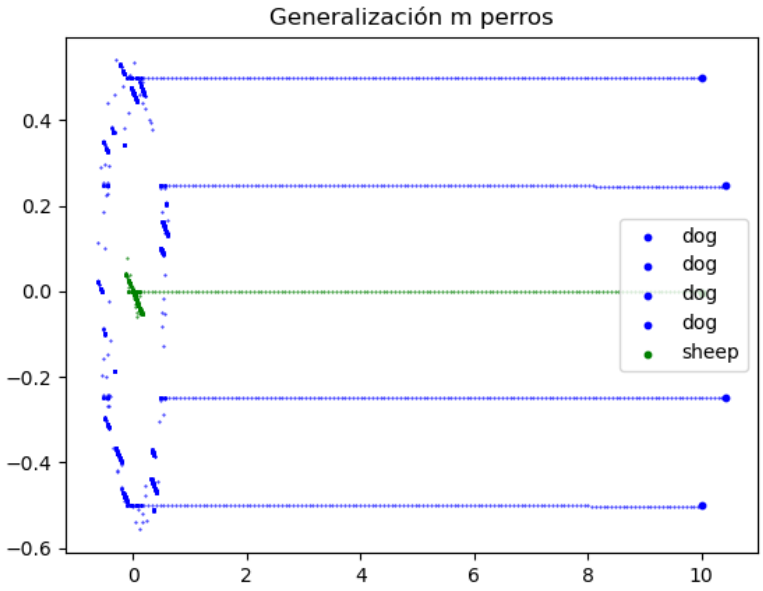

Figura 4.12: Movimiento con varios perros

Como se puede ver en la figura 4.12, el movimiento de las ovejas es rectilíneo. Esto es debido a que una vez introducida la dinámica de la oveja a través de la repulsión que los perros ejercen en ellas, su dinámica cambia al instante, de ahí que los ajustes realizados en la sección 3.3.1 no vayan a tener relevancia de aquí en adelante.

Capítulo 5

5. Movimiento varias ovejas

Una vez se tiene en funcionamiento el algoritmo de control con una oveja y múltiples perros, se empezó a pensar en la forma de introducir varias ovejas a modo de rebaño con lo que darle un toque más realista a la simulación.

Para poder introducir varias ovejas a forma de rebaño dentro de la simulación se pensó en dos posibles formas para poder generalizar, una de ellas muy simple y otra más complicada en la que las ovejas sí que participan activamente en el algoritmo de simulación.

5.1 Colocación simple (en circulo)

La primera forma para poder introducir más ovejas en la simulación, es simplemente introducir alrededor de la oveja con la que se realizan los cálculos ya realizados (en esta sección se trata de una oveja virtual) el número de ovejas que se elija formando un círculo alrededor de esta, siempre y cuando el circulo sea siempre menor al radio que forman los perros con la oveja con la que se realizan los cálculos pertinentes para llevar a cabo el guiado.

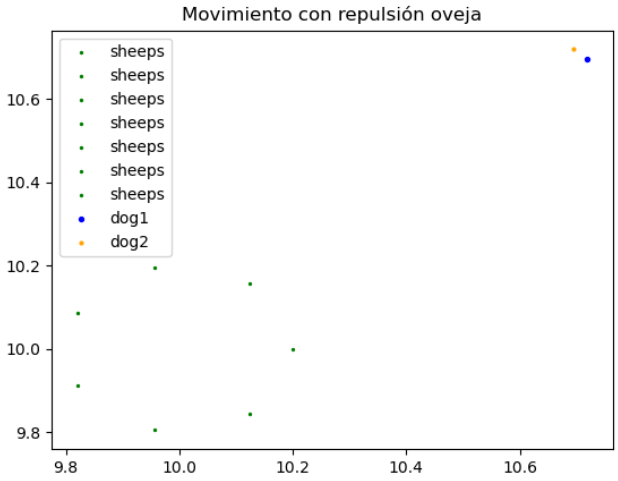

Figura 5.1: Posición inicial perros y ovejas

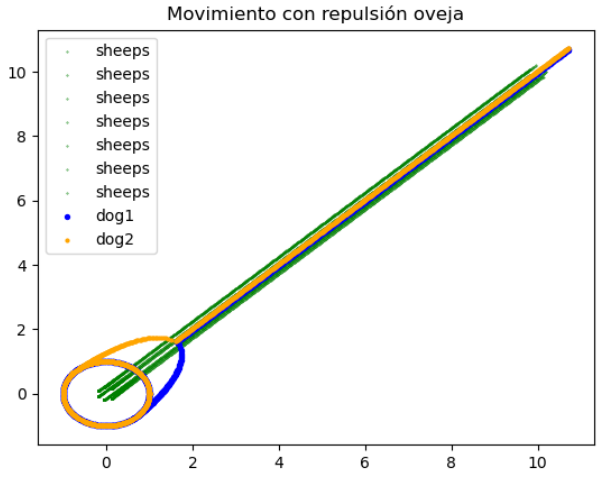

Figura 5.2: Movimiento completo

Como se puede apreciar en la figura 5.1, en un primer momento las ovejas se colocan formando un círculo alrededor de la localización en la que estaría la oveja virtual con la que se realizan los cálculos del control. En la figura 5.2 podemos observar el movimiento del controlador con dos perros, y cómo llegan al objetivo sin modificar la formación de las ovejas pues es una simple puesta en escena.

5.2 Algoritmo de rebaño

Para la segunda forma de implementación del rebaño sí que se va a tener en cuenta la posición de las ovejas y el número de ellas, pues en este caso sí que van a tener una implicación en el algoritmo de control, ya que van a ejercer una fuerza de repulsión y atracción entre ellas.

Para la implementación se partirá de otro documento, en este caso de un artículo científico presentado en el ICRA en 2013, "A Swarm Aggregation Algorithm based on Local Interaction with Actuator Saturations and Integrated Obstacle Avoidance" [13] , aunque en este caso se centrará únicamente en enjambre basado en interacción local y se dejará sin implementar la evitación de obstáculos. Dejar claro que es un agrupamiento por parejas, en el que interactúan las ovejas una con otra.
Se empieza con la dinámica que van a tener cada uno de los robots i en nuestro caso simulando que son ovejas.

$$\dot{x_\iota} = \frac{\sum_{j \in N_i} \gamma_{ij} * g\,(x_i - x_j)}{\sum_{j \in N_i} \gamma_{ij}} \qquad (17)$$

Donde γ_{ij} es el factor de ponderación entre cada uno de los robots i y j definido como la ecuación (19).

$$\gamma_{ij} = \frac{1}{\|x_i - x_j\|^{\alpha}} \qquad (19)$$

Donde $\alpha \geq 1$

La función $g()$ es la función de interacción definida como se ve en la ecuación (19).

$$g(y) = \frac{y}{\|y\|}\left(g_r(\|y\|) - g_a(\|y\|)\right) \qquad (19)$$

Con g_a y g_r como las funciones de atracción y repulsión respectivamente. Tener en cuenta que esta función es impar, por lo que $g(y) = -g(-y)$ Por lo que las funciones de repulsión y atracción están definidas como en las ecuaciones (20) y (21) respectivamente.

$$g_a(\|y\|) = a - (1 - \Phi(\|y\|)) \quad (20)$$

$$g_r(\|y\|) = b * \Phi(\|y\|) \qquad (21)$$

Donde $a, b > 0$ son constantes que toman valores entre 0 y 1.

Para elegir la función $\Phi()$ hay que tener en cuenta una serie de suposiciones para poder elegir una función que cumpla con los objetivos.

Existe una distancia única δ en que se equilibran las funciones de atracción y repulsión (22).

$$g_a(\delta) = g_r(\delta) \quad (22)$$

Además, es $g_a(\|y\|) \geq g_r(\|y\|)$ para $\|y\| \geq \delta$ y $g_a(\|y\|) < g_r(\|y\|)$ para $\|y\| \geq \delta$, lo que es lógico, pues si están muy cerca se quiere que se alejen y si están muy lejos se quiere que se junten.

Para elegir la función $\Phi()$ hay una serie de propiedades que debe satisfacer:

- Tiene que ser una función monótona
- $\lim_{\|y\| \to 0} \Phi(\|y\|) = 1$
- $lim_{\|y\| \to \infty} \Phi(\|y\|) = 0$

La función elegida para introducir es la que se propone como primer ejemplo en el documento [13] como se indica en la ecuación (23).

$$\exp(-\frac{\|y\|^{\beta}}{c}) \; con \; \beta \geq 1 \; y \; c > 0 \qquad (23)$$

Una vez se tenía claro el funcionamiento del sistema que se iba a utilizar, se comenzó a implementar un algoritmo capaz de realizar lo descrito anteriormente.

Antes de introducir esta parte en el grueso del sistema junto con el movimiento de control de los perros, se decidió hacer alguna prueba a parte para poder verificar que efectivamente el control crea unas fuerzas de repulsión y de atracción en función de si dos ovejas se encuentran más lejos o más cerca.

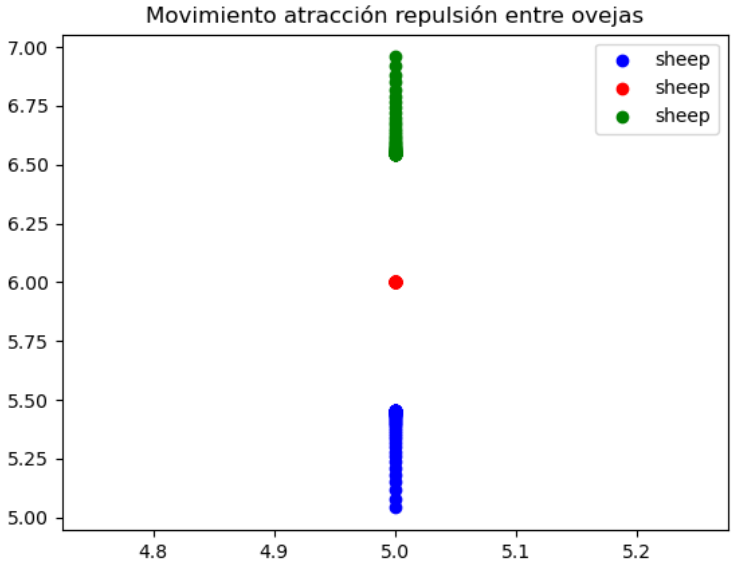

Figura 5.3: Repulsión-atracción de las ovejas

Como se puede ver en la figura 5.3 donde tenemos tres ovejas, vemos como la del centro esta quieta y como las de los laterales se acercan hasta un punto y una vez en ese punto reculan hacia atrás hasta volver a sentirse atraídas. También se puede comprobar como no van a llegar a tocarse nunca. También se realizaron una serie de pruebas para poder escoger los valores adecuados a las constantes que aparecen dentro de la implementación.

5.3 Movimiento varias ovejas y varios perros

Una vez se ha realizado el algoritmo de enjambre, se realiza la generalización a n ovejas como se ha hecho con la generalización a m perros, donde se ha creado una clase Sheep() donde irán al igual que en la clase Dog(), todos los atributos de la clase oveja y todos los métodos necesarios para el algoritmo.

Para incluir esta última parte del control de repulsión y atracción de las propias ovejas entre ellas, lo único que se ha modificado una vez se ha calculado la dinámica de repulsión y de atracción de cada una de ellas, ha sido introducir dentro de la dinámica global de la oveja ese valor sumando a la dinámica de repulsión que los perros introducían en las ovejas. También se introduce el calculo de la posición y la velocidad media del conjunto del rebaño de las ovejas para poder realizar los cálculos de una forma mas sencilla. Para calcular los valores de delta se usará la velocidad media del rebaño, así como para calcular el ángulo deseado (phi_deseada).

```
sheeps[i].pos_k = sheeps[i].pos - ks * (dindog + sheeps[i].dinsheep) *dt
```

Como se puede ver en esta línea de código, para calcular justo la posición siguiente de la oveja j necesitamos su posición anterior, el valor de la constante Ks ya explicada; $dindog$, que es la suma de las dinámicas de repulsión producidas por cada perro en la oveja j y; por último, $dinsheep$ propia de la oveja j donde están las dinámicas producidas por las otras ovejas en la oveja j. Para la colocación de los perros se usa el centroide de las ovejas, como se muestra en el código en el anexo.

Capítulo 6

6. Simulaciones finales

Una vez implementado el algoritmo completo se realizaron una serie de simulaciones. Se realizarán una serie de combinaciones con la variación de distintos parámetros del algoritmo. Primero se van a comentar un poco los parámetros más relevantes que aparecen y se les va a dar un valor. Después, en las distintas simulaciones, se variarán aquellos parámetros considerados de mayor interés.

$n \rightarrow$ número de ovejas.

$m \rightarrow$ número de perros.

$l \rightarrow$ longitud entre la localización de la oveja y el punto p con el que se realiza todo el cálculo del monociclo.

$r \rightarrow$ distancia a la que se encuentran los perros de las ovejas. Formando un círculo y dejando a las ovejas en el centro.

$dt \rightarrow$ tiempo de paso entre cada simulación.

$k \rightarrow$ constante del controlador proporcional del offset-point del monociclo.

$Kd \rightarrow$ constante de movimiento. Multiplica la dinámica del movimiento del perro, obteniendo que su movimiento sea más rápido o más lento.

$Ks \rightarrow$ constante de movimiento. Multiplica la dinámica del movimiento de la oveja, al igual que Kd conseguimos una dinámica más rápida o más lenta de la oveja.

A los parámetros que no van a variar debido a que no se consideran suficientemente relevantes se les pondrá un valor. Este será el mismo en cada una de las pruebas que se realicen.

Para la variable de l se ha considerado una distancia de **0.1**, elegida al principio del proceso de realización, aunque podría variar en función de la escala en la que se use.

El paso de simulación (dt) es de **0.01** y la constante del control proporcional (k) es **2**. Estos valores fueron seleccionados al igual que l al principio, siendo validado su correcto funcionamiento.

Para los parámetros de regulación del comportamiento de rebaño se han elegido valores del artículo [13] quedando de la siguiente forma:

$$Beta\ (\beta) = 1$$

$$Alfa\ (\alpha) = 1$$

$$c = 1$$

$$a = 1$$

$$b = 1$$

6.1 Primera simulación

En esta primera prueba se va a realizar una simulación donde se han sacado distintas gráficas para ver el comportamiento del sistema. Los valores de los parámetros y las variables que se aplican en este apartado seguirán siendo los mismos en los apartados siguientes a no ser que se especifique lo contrario.

En este primer caso utilizaremos los siguientes valores:

$$Kd = 100$$

$$Ks = 1$$

$$n = 4$$

$$m = 3$$

$$r = 0.5$$

Los parámetros de las constantes de movimiento son los mencionados para que el perro vaya directo a la posición inicial debido a que, si $Kd = \frac{1}{dt}$, como está implementado el algoritmo ecuación (24), la posición siguiente es la posición deseada.

$$d1\_k1 = d\_1 + Kd * (d\_1\_deseada - d\_1) * dt \quad (24)$$

$$d1_{k1} = d_1 + \frac{1}{dt} * (d\_1\_deseada - d\_1) * dt$$

$$d1\_k1 = d\_1\_deseada$$

Las coordenadas de inicio de las ovejas son $(5{,}5)$.

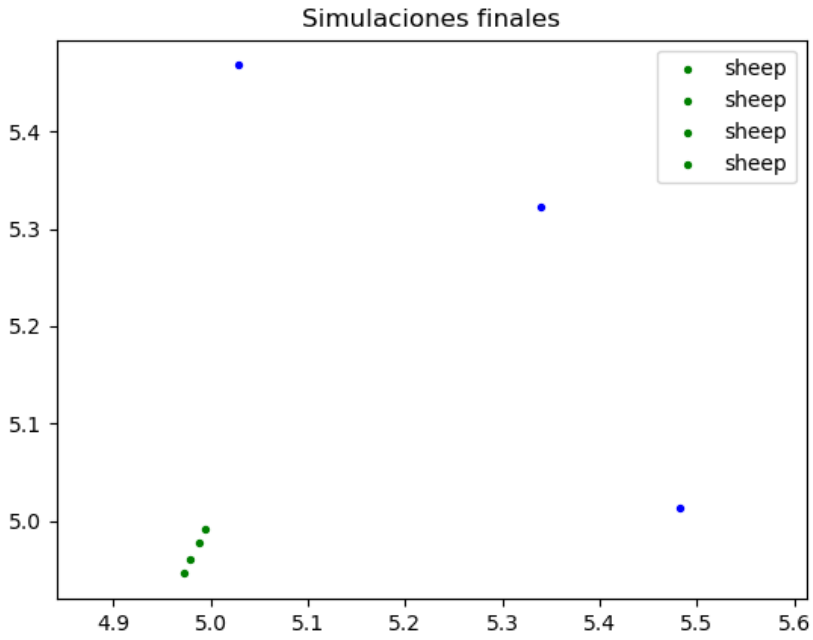

Figura 6.1: Posiciones iniciales $i = 1$

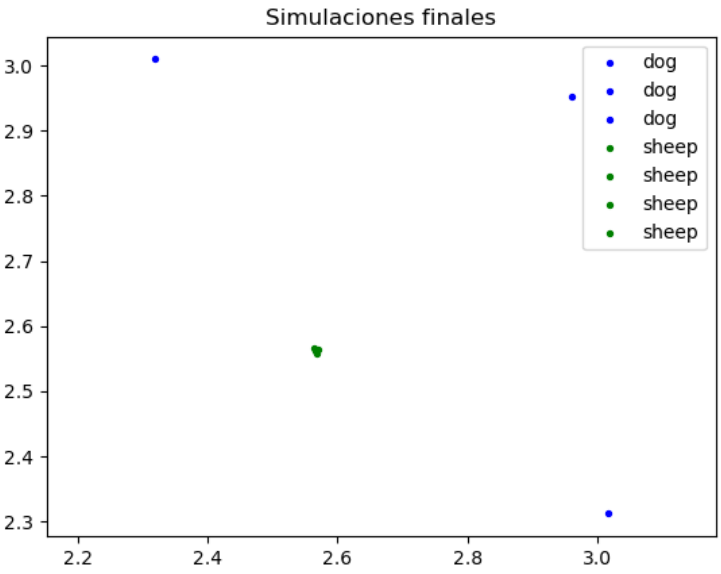

Figura 6.2: Posiciones $i = 50$

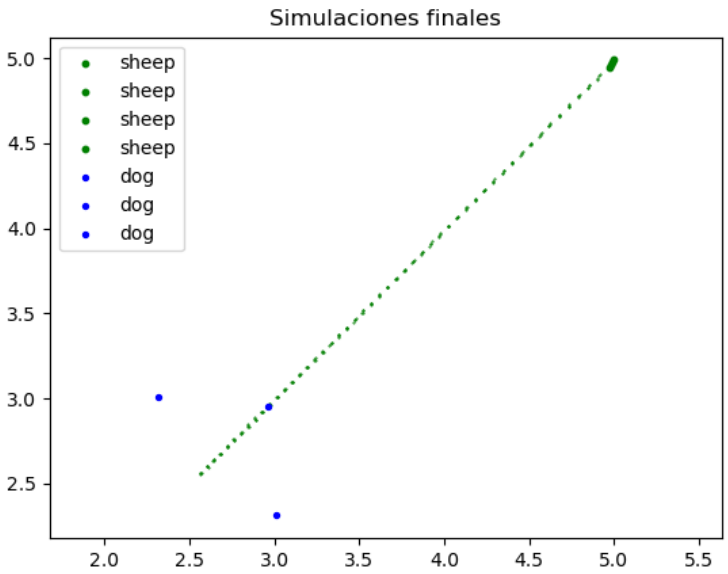

Figura 6.3: Posiciones con recorrido completo de las ovejas $i = 50$

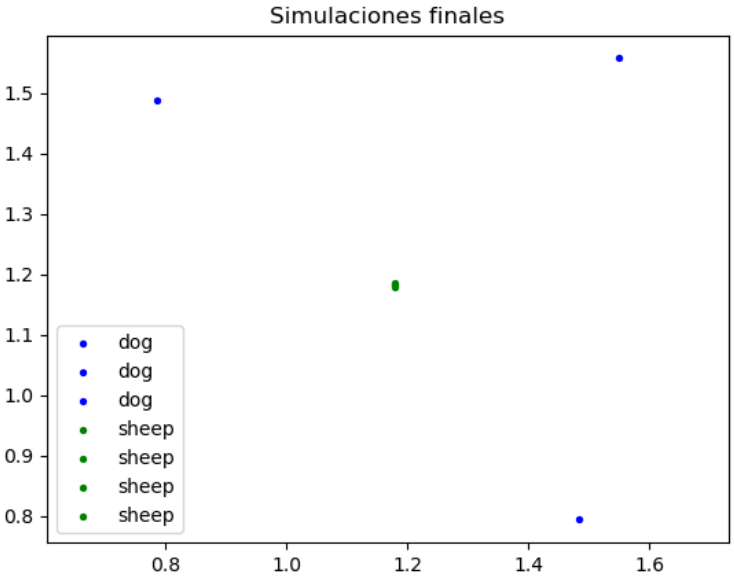

Figura 6.4: Posiciones $i = 100$

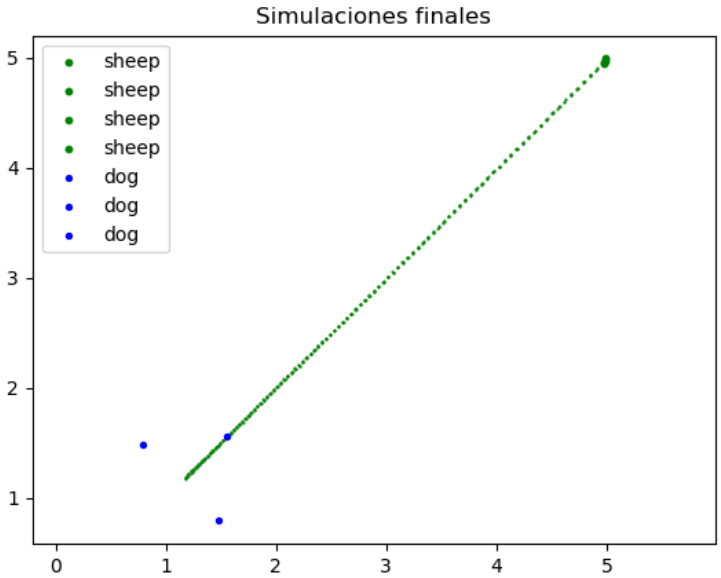

Figura 6.5: Posiciones con recorrido completo de las ovejas $i = 100$

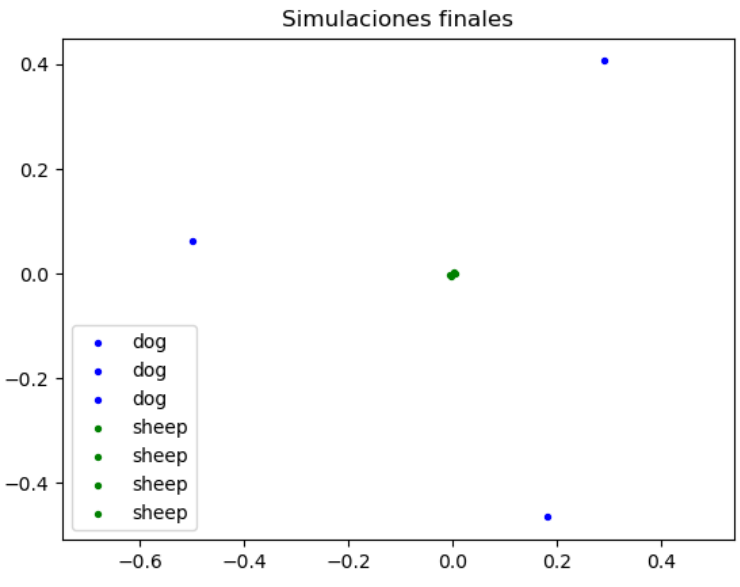

Figura 6.6: Posiciones $i = 300$

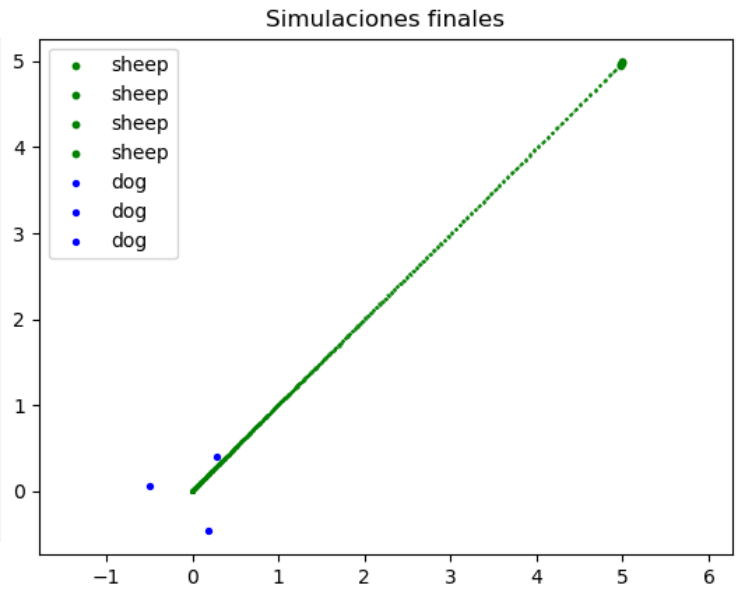

Figura 6.7: Posiciones con recorrido completo de las ovejas $i = 300$

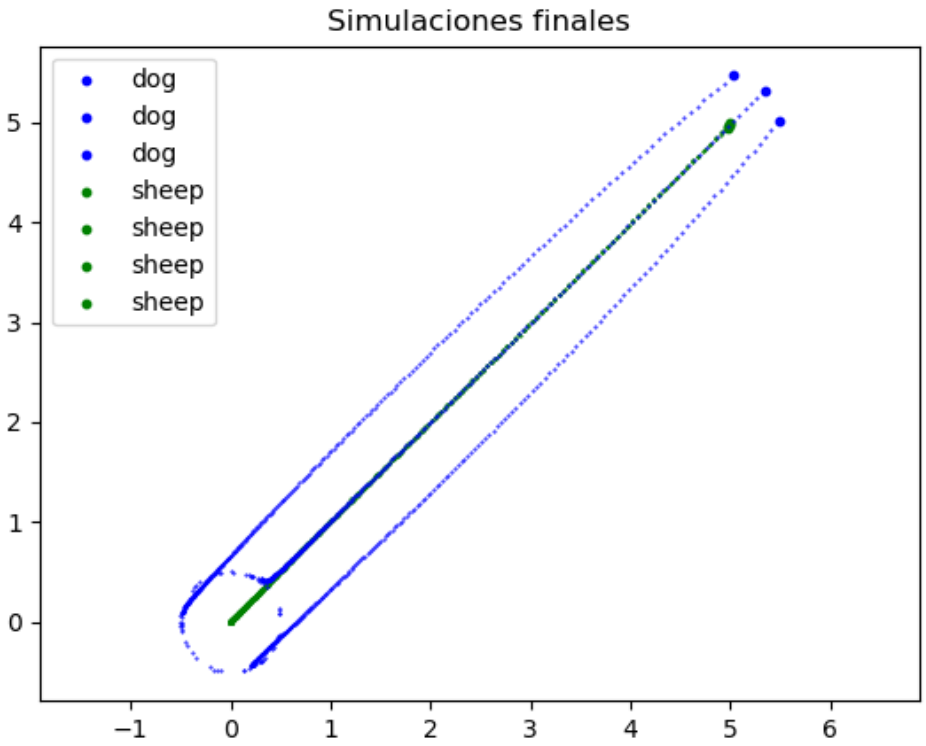

Figura 6.8: Recorrido final realizado $i = 300$

En la figura 6.1 se pueden ver las posiciones iniciales que toman los perros y las ovejas. En las siguientes seis figuras, se observa el movimiento de las ovejas en los instantes marcados, además del recorrido completo que ha realizado hasta el momento la oveja. Se ve cómo los perros se colocan formando un semicírculo de radio 0.5 alrededor de las ovejas. En la última figura (6.8), podemos ver el recorrido completo de perros y ovejas. Vemos como los perros

realizan movimientos en círculos alrededor del objetivo, esto lo hacen para evitar que se salgan de ahí.

6.2 Cambio de posiciones iniciales

Para esta simulación se usarán los valores obtenidos anteriormente con la única modificación de la posición de inicio de los perros y las ovejas. Se realizan pruebas en los tres cuadrantes restantes.

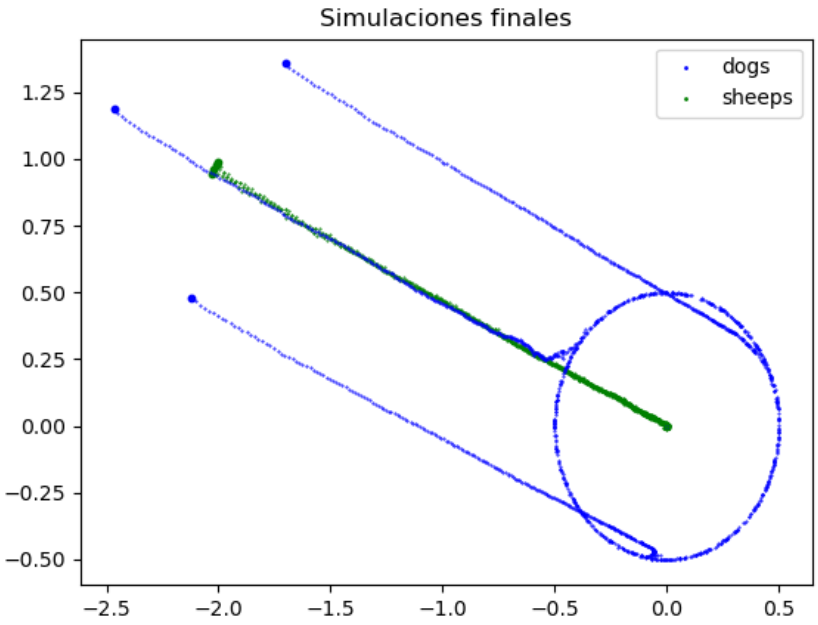

Figura 6.9: Movimiento cuarto cuadrante.
Posición inicial $(-2,1)$

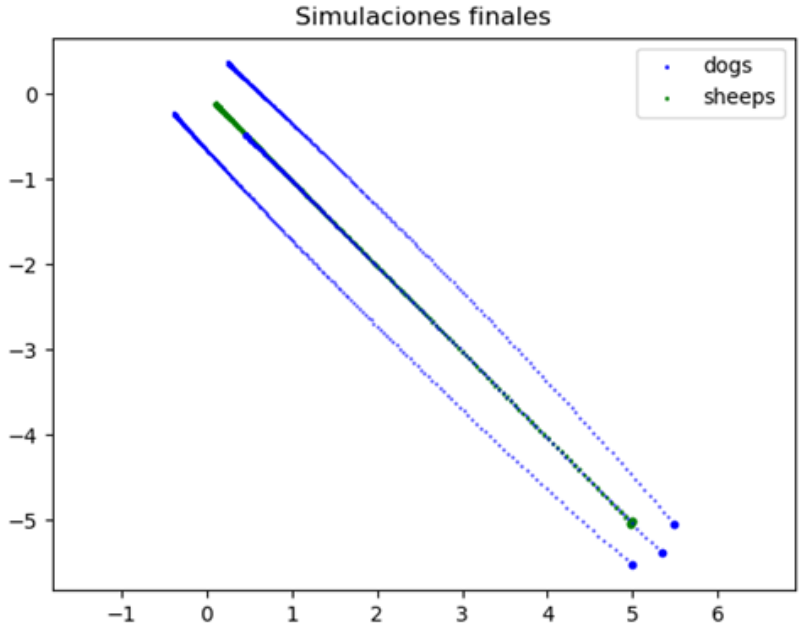

Figura 6.10: Movimiento segundo cuadrante.
Posición inicial $(5,-5)$

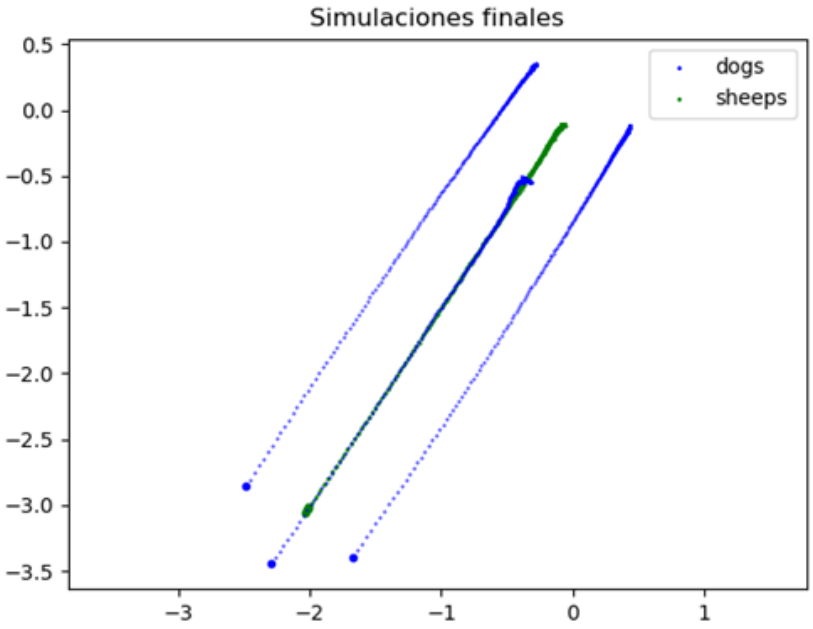

Figura 6.11: Movimiento tercer cuadrante.
Posición inicial $(-2,-3)$

Como podemos comprobar, en los tres cuadrantes las ovejas llegan al objetivo. Todas las figuras están realizadas con el mismo número de iteraciones (300). Con esto podemos afirmar que en los cuadrantes segundo y tercero costará más llegar al objetivo.

6.3 Variación de Kd y Ks

Las constantes Kd y Ks son las encargadas de regular la dinámica de los movimientos de las ovejas y los perros. Es por eso que es necesario que haya una relación entre estas, pues si la dinámica de las ovejas es mucho más alta que la de los perros, estos nunca llegarán a alcanzarlas.

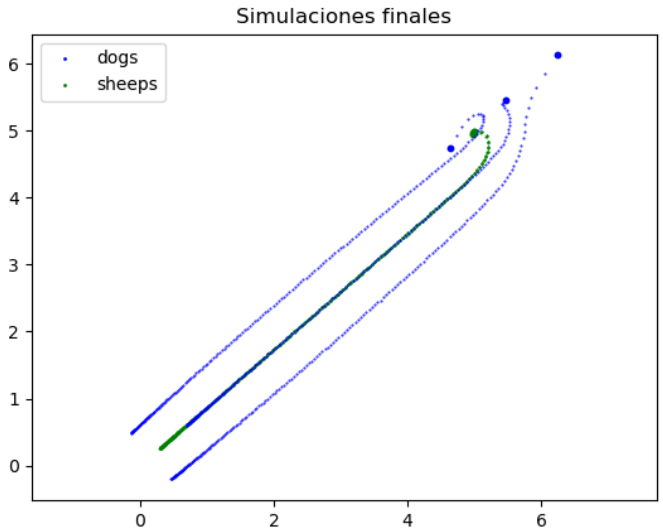

Figura 6.12: $Kd = 25$ $Ks = 1$ $i = 300$

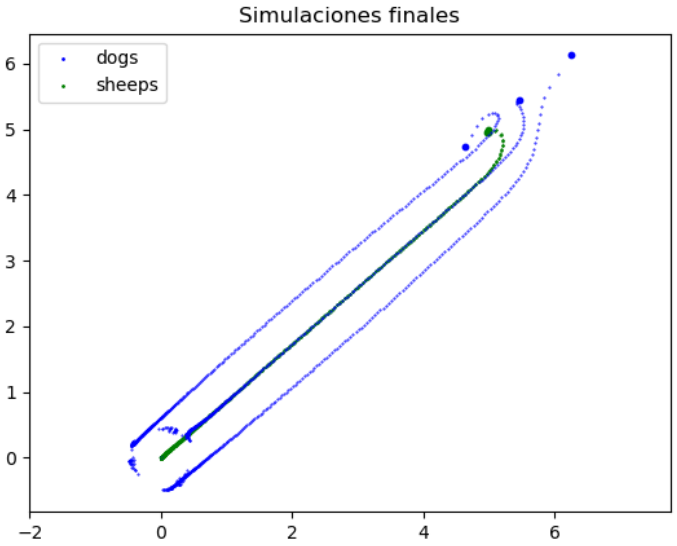

Figura 6.13: $Kd = 25\ Ks = 1\ i = 400$

Como podemos ver en la figura 6.12 la posición inicial es la misma que con la que se realizaron las pruebas en la sección 6.1. En este caso, se ha modificado la constante Kd que ha pasado de ser 100 a 25. En la gráfica 6.12 vemos como aun realizando las mismas iteraciones que en el apartado anterior (300), no consigue llegar al (0,0). Lógico ya que ahora la dinámica es más lenta. Aumentando el número de las iteraciones conseguimos que llegue al objetivo.

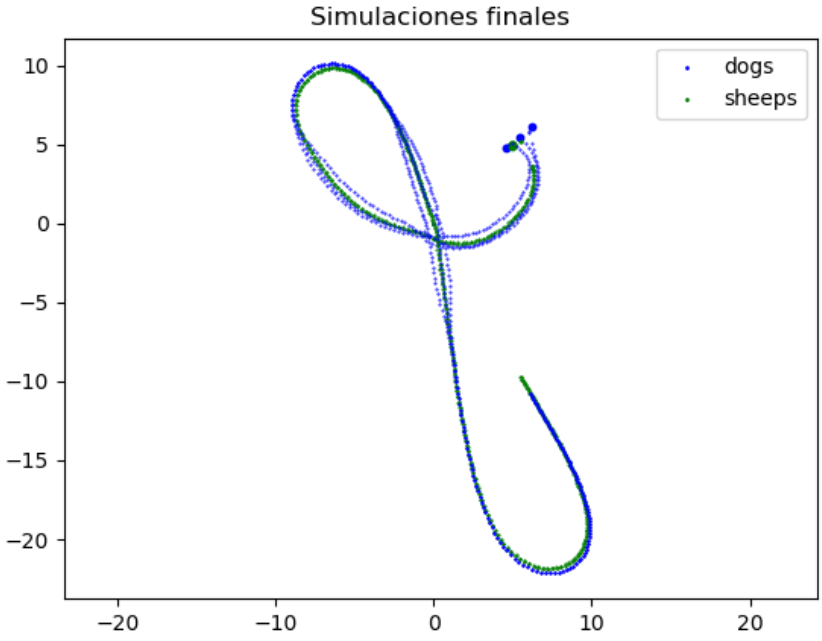

Figura 6.14: $Kd = 25\ Ks = 10\ i = 300$

En la figura 6.14, existe un claro ejemplo en el que la dinámica de los perros no es suficientemente rápida en comparación con la dinámica de las ovejas. Es por esto que nunca se podrán quedar quietas en el objetivo, aunque siempre intenten volver hacia él.

6.4 Variación del radio r

Para la realización de este apartado volvemos a cambiar los valores de Kd y Ks, dejándolos como estaban anterior mente 100 y 1. Lo que se va a variar ahora es el radio. En el artículo [9] existe un apartado donde se puede obtener un radio que varíe en función del número de ovejas. Esto se deja para trabajo futuro, y aquí simplemente se realizarán algunas pruebas dando valores distintos al radio.

Las pruebas anteriores se han realizado con un radio de 0.5 por lo que se utilizará ahora un radio más grande y otro más pequeño.

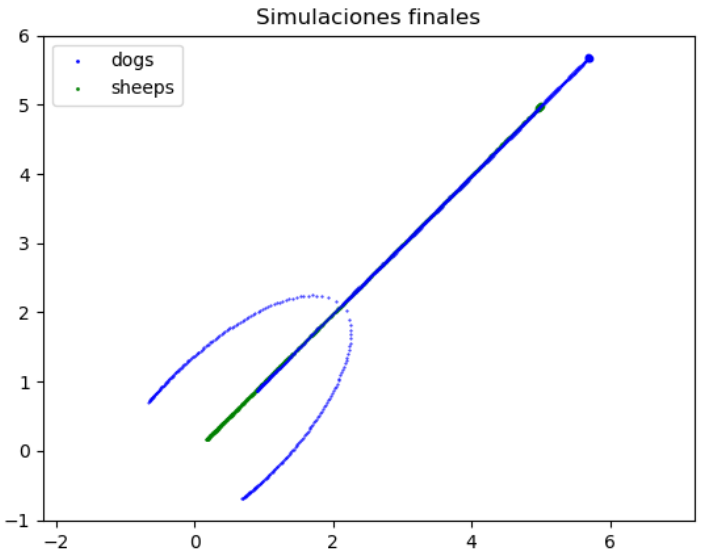

Figura 6.15: $r = 1 \;\; i = 300$

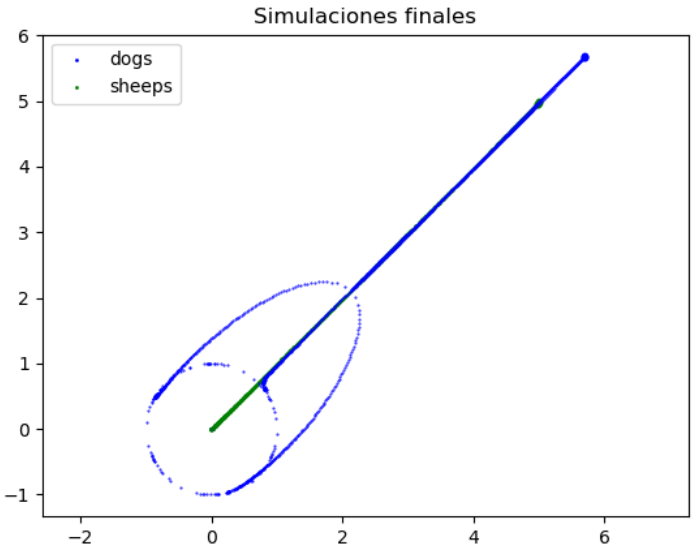

Figura 6.16: $r = 1 \; i = 400$

Al estar más alejados de las propias ovejas, las fuerzas de repulsión son menores. Como vemos en las figuras 6.15 y 6.16 los perros están juntos desde un principio para que la repulsión sea máxima y se van separando conforme llegan al objetivo. Aun así, necesitan más tiempo para llegar al objetivo.

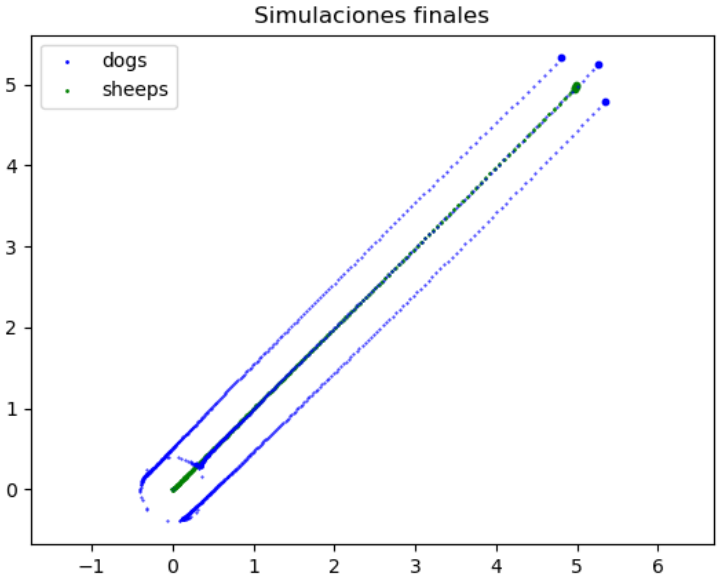

Figura 6.17: $r = 0.3 \;\; i = 300$

Como podemos ver en la figura 6.17, al tener un radio más pequeño las fuerzas de repulsión son más altas, por lo que no es necesario que los perros se junten para ejercer más repulsión. Además, la dinámica es más rápida. De ahí que, en el mismo número de iteraciones ($i = 300$), esta vez sí que llegue al objetivo.

6.5 Variación número de perros y ovejas.

En este último apartado, ser realizarán una serie de combinación con distintos números de perros (m) y ovejas (n) para ver la disposición que estos toman.

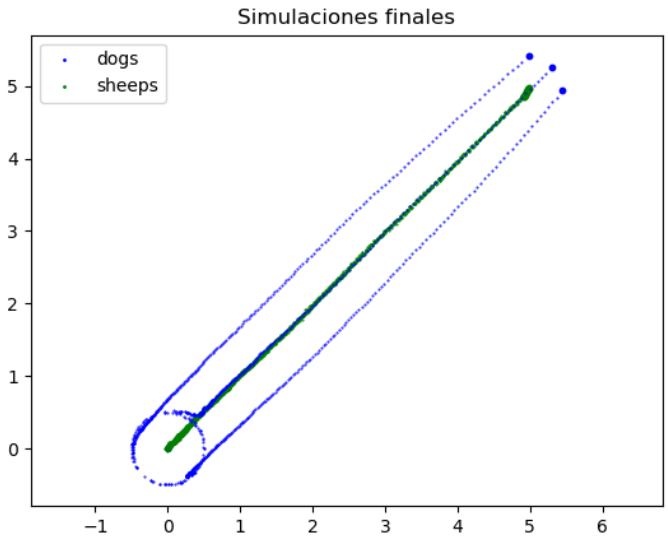

Figura 6.18: $n = 10\ m = 3$

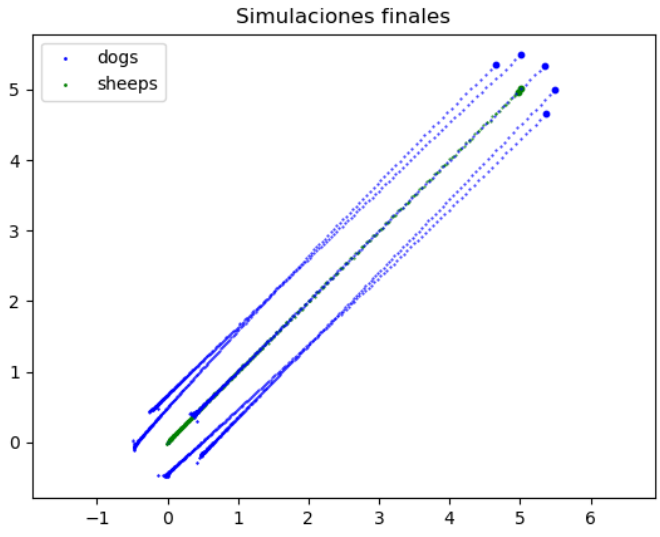

Figura 6.19: $n = 2\ m = 5$

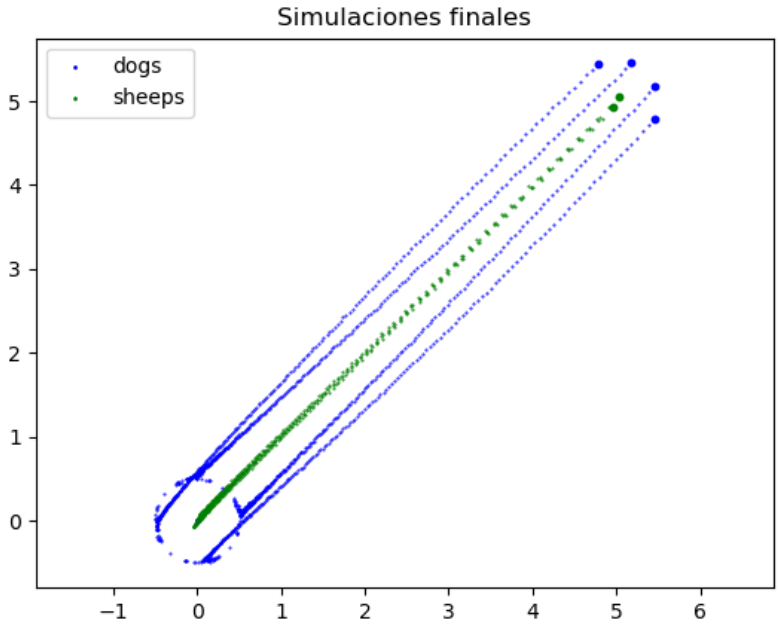

Figura 6.20: $n = 4\ \ m = 4$

Capítulo 7

7. Conclusión y trabajo futuro

En este capítulo se va a proceder a evaluar si se han cumplido los objetivos iniciales propuestos al comienzo del trabajo y, en caso afirmativo, hasta qué punto se ha hecho.

Por otro lado, se hablará, en base al trabajo realizado, de cómo puede seguir mejorándose en un futuro.

7.1 Conclusión

Tras la realización del trabajo, podemos afirmar que se han cumplido los objetivos previstos. Desde una mejor familiarización con el lenguaje Python y su entorno, la lectura y compresión de los artículos científicos en los que se basa el trabajo, la realización del algoritmo por partes para poder entender bien el procedimiento de este, hasta concluir con las simulaciones completas del algoritmo con todos los parámetros y la elaboración de este documento.

Este trabajo también ha supuesto una mejora en las técnicas de análisis y depuración en la elaboración de código, ya que no se ha seguido al pie de la letra lo establecido en los distintos documentos y se han tenido que solucionar muchos problemas, tanto problemas puros de programación y compilación, como simples problemas de geometría. Además de repasar técnicas de control de posición de robots monociclo al tener que usar esas técnicas en el transcurso del trabajo.

Por último, se ha conseguido una mejor familiarización con las técnicas multi-robot, las cuales serán muy importantes en un futuro muy cercano, como ya se están empezando a ver, además de ganar protagonismo en el ámbito en el que se ha realizado el trabajo como es el pastoreo robótico, pues llegará un momento que los grandes propietarios de ganado utilicen estas técnicas para ganar eficiencia y reducir gastos.

7.2 Trabajo Futuro

Con la realización de este trabajo, se han conseguido implementar técnicas de control para pastoreo robótico. De esta manera, en futuros proyectos se podría implementar alguna de estas técnicas dentro de robots reales, tanto terrestres como aéreos, como pueden ser drones. Los robots aéreos serían los más adecuados para la realización de estas tareas pues su movimiento no depende tanto del entorno como podría suceder en los robots terrestres.

Otra posibilidad de futuro trabajo podría ser la implementación de la detección de obstáculos explicada en el segundo documento utilizado en el trabajo [13] y que ayudaría a que el algoritmo fuera más robusto. También se podría jugar con el tamaño del radio r al que se colocan los perros de las ovejas. Desde el posicionamiento inicial que no pase por el centro del rebaño, sino que se muevan por los bordes del círculo o incluso que se alejen primero para no espantarlas. Hasta modificar su radio en función del número de ovejas del rebaño.

Capítulo 8

8. Bibliografía

[1] Madridano Á., C. S.-K. (2020). Vehículo aéreo no tripulado para vigilancia y monitorización de incendios. *Revista Iberoamericana de Automática e Informática industrial, 17*(3), 254-263. Obtenido de https://polipapers.upv.es/index.php/RIAI/article/view/11806

[2] Lòpez, J., Peréz Losada, D., Sanz, R., & Paz, E. (2009). *Sistemas de vigilancia de edificios basado en robots móviles. Workshop Robot'09*

[3] UAV & UAM. (28 de Junio de 2019). Funcinamiento y aplicaciones con UAVs en enjambre. Recuperado el 2021, de https://www.embention.com/es/news/funcionamiento-y-aplicaciones-con-uavs-en-enjambre/

[4] Ortega, E. (25 de 03 de 2021). La ESA presenta al sorprendente robot que explorará las cuevas de la Luna. Obtenido de https://computerhoy.com/noticias/tecnologia/esa-presenta-sorprendente-robot-explorara-cuevas-luna-836195

[5] Ackerman, E. (26 de 05 de 2021). What's Going on With Amazon's "High-Tech" Warehouse Robots? *IEEE*. Obtenido de https://spectrum.ieee.org/whats-going-on-with-amazons-hightech-warehouse-robots

[6] CSIC Comunicación. (26 de Marzo de 2021). Sistemas multi-robot para realizar operaciones agrícolas de precision y automatizas.

[7] Rus, C. (18 de 04 de 2021). En China los drones forman códigos QR gigantescos para que los espectadores se descarguen un videojuego. *Xataca*. Obtenido de https://www.xataka.com/drones/china-drones-forman-codigos-qr-gigantescos-espectadores-se-descarguen-videojuego

[8] Actualidad Aeroespacial. (26 de 07 de 2021). Un total de 1.874 drones iluminaron el cielo de Tokio en la inauguración de los JJOO. Obtenido de https://actualidadaeroespacial.com/un-total-de-1-874-drones-iluminaron-el-cielo-de-tokio-en-la-inauguracion-de-los-jjoo/

[9] Pierson, A., & Schwager, M. (2015). Bio-Inspired Non-Cooperative Multi-Robot Herding. *IEEE International Conference on Robotics and Automation (ICRA), pp. 1843-1849* Washington.

[10] CONtexto ganadero. (23 de 12 de 2020). Drones para pastoreo, el futuro de la ganadería según empresario israelí. Obtenido de https://www.contextoganadero.com/ganaderia-sostenible/drones-para-pastoreo-el-futuro-de-la-ganaderia-segun-empresario-israeli

[11] Lane, M. (16 de 2 de 2011). Helicopter cowboys of Australia's Outback. *BBC News*. Obtenido de https://www.bbc.com/news/world-asia-pacific-12408888

[12] Vaighan, R., Sumpter, N., Henderson, J., Frost, A., & Cameron, S. (2000). *"Experiments in automatic flock control," Robotics and Autonomous Systems* 31(1-2), 109-117

[13] Laccese, A., Gasparri, A., Priolo, A., Oriolo, G., & Ulivi, G. (2013*). A Swarm Aggregation Algorithm based on Local Interaction with actuator saturations and integrated obstacle*

avoidance. *IEEE International Conference on Robotics and Automation (ICRA), pp. 1865-1870,* Karlsruhe, Germany,.

[14] C. M. Breder, "*Equations descriptive of fish schools and other animal aggregations*," Ecology, vol. 35, no. 3, pp. 361–370, Jul. 1954.

ANEXO (Código)

```
import numpy as np
import matplotlib.pyplot as plt

######## funciones ########

def list_v_delta(m, r):       # se calculan en función del radio y del número de perros
    v_list = np.zeros(360)    #las velocidades asociadas a los valores delta de 0 a 360
grados.
    delta = np.zeros(360)
    i = 0
    grad = 1
    while i < 360:
        delta[i] = np.deg2rad(grad)

        v_list[i] = np.sin((m * delta[i]) / (2 - 2 * m)) / (r ** 2 * np.sin(delta[i] /
(2 - 2 * m)))
        i = i + 1
        grad = grad + 1

    return v list


def emparejamiento(v_list, v):   # función con la que asignaremos una delta en función
    if v > v_list[0]:            # del valor de la velocidad de la oveja calculado
anteriormente
        delta = np.deg2rad(1)
    elif v < v_list[359]:
        delta = np.deg2rad(360)
    else:
        delta = np.deg2rad((np.abs(v_list - v)).argmin() + 1)
    return delta


def trans(x, y):
    mat = np.array([[1, 0, x], [0, 1, y], [0, 0, 1]])
    return mat


def rot(tita):
    mat = np.array([[np.cos(tita), -np.sin(tita), 0], [np.sin(tita), np.cos(tita), 0],
[0, 0, 1]])
    return mat


def cinematica(phi, l):
    cinema = ([np.cos(phi), np.sin(phi)], [-np.sin(phi) / l, np.cos(phi) / l])
    return cinema

##### funciones para cálculo de repulsión atracción ovejas #####

def fi(y, beta, c):
    phi = np.exp(- (np.linalg.norm(y))**beta/c)
    return phi

def g_y(y, g_r, g_a):                  # función para calcular la dinámica conjunta
    g_y = y/np.linalg.norm(y) * (g_r - g_a)
    return g_y

def gamma(y, alfa):                          # función relacionada con la repulsión
    gamma = 1 / (np.linalg.norm(y)**alfa)
```

```
    return gamma

######## variables inicio ########

n = 4  # número de ovejas
phi = 90 # ángulo inicial de la oveja
l = 0.1  # distancia de la posición de la oveja al offset-point de cálculo de monociclo
m = 3  # número de perros
r = 0.5  # distancia a la que tienen que estar los perros de la oveja

v_list = list_v_delta(m, r) #calculamos los valores de velocidades para luego comparar

######## parámetros de simulación ########

dt = 0.01 #paso de simulación
s = 0 #iteraciones

####### variables de control ########

k = 2    #constante control proporcional
kd = 100 #constante proporcional movimiento de los perros
ks = 1   #constante proporcional movimiento de las ovejas

######## parámetros repulsión atracción ovejas ########

beta = 1 #zona de atracción, tiene que ser mayor o igual a 1
alfa = 1 #tiene que ser mayor o igual a 1
c = 1   #tiene que ser mayor que 0
a = 4   #g_a, proporcional a la fuerza de atracción g_r cogerán valores entre el rango
[-a, b]
b = 1   #proporcional a la fuerza de repulsión


######## clase dog ########

class Dog():
    """ Clase usada para representar los perros"""

    def __init__(self, pos):
        self.pos = pos
        self.pos_k = ""
        self.pos_deseada = ""
        self.delta = ""
        self.r = r
        self.plotx = []
        self.ploty = []
        self.dis = ""

    def vecino_prox(self, m, available, v_dogs):
        """ Cálculo del vecino más próximo a la posición deseada siguiente
        para evitar cruzamientos.

        Parámetros:
            m : número de perros.

            available : vector de booleanos utilizado para indicar si ya está
            cogida una posición.

            v_dogs : vector con todos los perros

        Devuelve:
            best_ind : índice donde se encuentra la mejor posición deseada para el perro
            del que se está calculando.

            available : vector actualizado.

        """
        best_dis = np.inf
        best_ind = -1
        for j in range(m):
            if available[j] == True:
                self.dis = np.linalg.norm(v_dogs[j].pos_deseada - self.pos)
                if self.dis < best_dis:
                    best_dis = self.dis
                    best_ind = j
```

```
        available[best_ind] = False

        return best_ind, available


    def calculo_pos_deseada(self, m, delta, i, sheep_x_media, sheep_y_media,
phi_deseada):
        """ Calcula la posición deseada para poder aplicar luego la dinámica.

        Parámetros:
            m : número de perros.

            delta : ángulo de separación entre los perros en radianes.

            i : dog al que se le está aplicando el cálculo (d1, d2...)

            sheep_x_media, sheep_y_media : posiciones medias de las ovejas

            phi_desada : ángulo deseado en radianes
        """
        self.delta = delta * (2 * (i + 1) - m - 1) / (2 * m - 2)
        pos_deseada = trans(sheep_x_media, sheep_y_media) @ trans(self.r *
np.cos(phi_deseada + self.delta), self.r * np.sin(phi_deseada + self.delta))

        self.pos_deseada = np.array([pos_deseada[0, 2], pos_deseada[1, 2]])

    def plot_dinamico_perros(self, s):
        if s == 0:
            punto = 10
            if s == 0:
                etiqueta2 = "dog"
        else:
            punto = 0.2
            etiqueta2 = ""

        plt.scatter(self.pos[0], self.pos[1], color="blue", label=etiqueta2, s=punto)
```

```
######## clase sheep ########

class Sheep():
    """ Clase usada para representar las ovejas"""
    def __init__(self, pos, phi):
        self.pos = pos
        self.pos_k = ""
        self.phi = phi
        self.phi_deseada = ""
        self.p = self.pos + np.array([np.cos(self.phi) * l, np.sin(self.phi) * l])
        self.p_k1 = ""
        self.p_k2 = ""
        self.v = ""
        self.w = ""
        self.plotx = []
        self.ploty = []
        self.dinadog = 0
        self.dinsheep = 0
```

```
    def control_sheep(self, sum_x, sum_y, sum_v):
        """ Realiza el control de cada una de las ovejas,
            para obtener la velocidad y la posición.

            Parámetros:
                sum x : la suma de la posición X de cada oveja

                sum_y : la suma de la posición Y de cada oveja

                sum_v : la suma de las velocidades de cada oveja

            Devuelve :
                los parámetros actualizados
        """
        self.p = self.pos + np.array([np.cos(self.phi) * l, np.sin(self.phi) * l])
        self.p_k1 = - k * self.p
        if np.linalg.norm(self.p_k1) < 1:
            p_norm = self.p_k1
        else:
            p_norm = self.p_k1 / np.linalg.norm(self.p_k1)
        self.p_k2 = - k * p_norm
        m_cinema = cinematica(self.phi, l)
        self.v, _ = m_cinema @ self.p_k1

        if self.v < 0:
            self.v = -self.v

        # cálculo de la media, necesaria para phi_desada
        sum_x = sum_x + self.pos[0]
        sum_y = sum_y + self.pos[1]
        sum_v = sum_v + self.v

        return sum_x, sum_y, sum_v

    def dinamica_ovejas(self, i, n, a, b, c, alfa, beta, v_sheeps):
        """ Cálculo de la dinámica producida por las ovejas
        debida a su propia repulsión-atracción.

        Parámetros:

            i : número de la oveja con la que se están realizando
            los cálculos.

            n : número de ovejas.

            a, b, c, alfa, beta : parámetros control algoritmo.

            v_sheeps : vector con todas las ovejas.

        Devuelve:

            rep_atrac : la dinámica a aplicar en las ovejas, que se sumará a la que
realizan
            los perros en estas.

        """
        num = 0
        den = 0
        for j in range(n):
            if i != j:
                y = self.pos - v_sheeps[j].pos
                g_a = a * (1 - fi(y, beta, c))
                g_r = b * fi(y, beta, c)
                num = num + gamma(y, alfa) * g_y(y, g_r, g_a)
                den = den + gamma(y, alfa)

        rep_atrac = num / den

        return rep_atrac
```

```
    #ploteo dinámica

    def plot_dinamico(self, s):
        if s == 0:
            punto = 10
            if s == 0:
                etiqueta1 = "sheep"

        else:
            punto = 0.2
            etiqueta1 = ""

        plt.scatter(self.pos[0], self.pos[1], color="green", label=etiqueta1, s=punto)


######## inicializamos ovejas  ########

default_pos = np.array([5, 5])
default_phi = phi * np.pi / 180
sheeps = []
for i in range(n):
    sheeps.append(Sheep(default_pos - i * np.array([0.01, 0.02]), default_phi))

######## inicializamos perros ########

dogs = []
for i in range(m):
    pos_inicial = sheeps[0].pos - np.array([0.5, 0.5]) + i*np.array([1, 1])
    dogs.append(Dog(pos_inicial))

######## simulación ########

while s < 500:
    sum_x = 0
    sum_y = 0
    sum_v = 0
    for i in range(n):
        sum_x, sum_y, sum_v = Sheep.control_sheep(sheeps[i], sum_x, sum_y, sum_v)

    phi_deseada = np.arctan2(sum_y / n, sum_x / n)
    delta = emparejamiento(v_list, sum_v / n)  #calculamos el valor de delta con la
velocidad media del rebaño


     #### cálculo posición deseada perros ####

     for i in range(m):
        Dog.calculo_pos_deseada(dogs[i], m, delta, i, sum_x / n, sum_y / n, phi_deseada)

     for i in range(n):
        sheeps[i].dinadog = 0   #se hace cero la dinámica para que no se sobrescriba


     #### cálculo dinámico perros ####

    available = []
    for i in range(m):
        available.append(True)

    for i in range(m):
        near, available = Dog.vecino_prox(dogs[i], m, available, dogs)
        dogs[i].pos_k = dogs[i].pos + kd * (dogs[near].pos_deseada - dogs[i].pos) * dt
        for j in range(n):
              sheeps[j].dinadog = sheeps[j].dinadog + ((dogs[i].pos - sheeps[j].pos) /
((np.linalg.norm(dogs[i].pos - sheeps[j].pos)) ** 3))


     #### cálculo dinámica oveja ####
```

```
    if n > 1:
        for i in range(n):
            rep_atrac = Sheep.dinamica_ovejas(sheeps[i], i, n, a, b, c, alfa, beta,
sheeps)
            sheeps[i].dinsheep = rep atrac

     #### aplicamos repulsión a las ovejas ####

     for i in range(n):
        sheeps[i].pos_k = sheeps[i].pos - ks * (dindog + sheeps[i].dinsheep) * dt
        sheeps[i].pos = sheeps[i].pos_k

    for i in range(m):
        dogs[i].pos = dogs[i].pos_k
        dogs[i].plotx.append(dogs[i].pos_k[0])
        dogs[i].ploty.append(dogs[i].pos_k[1])

        #ploteo dinámica
        Dog.plot_dinamico_perros(dogs[i], s)

    for i in range(n):
        sheeps[i].plotx.append(sheeps[i].pos_k[0])
        sheeps[i].ploty.append(sheeps[i].pos_k[1])

        # ploteo dinámico
        Sheep.plot dinamico(sheeps[i], s)

    s = s + 1

###### gráficas ######
plt.title('Simulaciones finales')
plt.axis('equal')
plt.legend()
plt.show()
```