UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Formaciones adaptativas para la evasión de obstáculos mediante *flocking* aplicado a la robótica móvil

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A N

Moisés Sebastián González Chávez

Alejandro Ruiz Esparza Rodríguez

DIRECTOR DE TESIS

Dr. Víctor Javier González Villela

Ciudad Universitaria, Cd. Mx., 2019

Contenido

Índice de figuras

Índice de ecuaciones

Índice de tablas

1. Agradecimientos

Agradecemos en lo que corresponde a la DGAPA, por el apoyo brindado para la realización de este trabajo, a través del proyecto UNAM-DGAPA-PAPIIT IN118117: "Investigación sobre robótica topofixadaptable aplicada a robots móviles híbridos, que operan en ambientes inteligentes estructurados, en tareas de sujeción, traslación y orientación de objetos con cierto grado de asimetría"

De Moisés:

A mi mamá, por su apoyo incondicional, soporte y entendimiento a lo largo de mi trayectoria académica en la Facultad de Ingeniería y, en general, durante toda mi vida. Le debo todo, por lo que parte de este trabajo es igualmente de ella.

A mi hermana que, a pesar de todo, ha sabido sobreponer la importancia de la familia antes de cualquier interés personal. Igualmente gracias, por su actitud a lo largo de mi vida.

A Guillermo Ruiz, por ser parte fundamental de mi familia y ser parte también de este gran logro.

A Alejandra Vega López, debido a que con el amor que en estos años me ha brindado, siempre ha sabido cómo sacar lo mejor de mí, incluso en los momentos más complicados.

A la familia Vega López, por su apoyo e incondicional amistad a lo largo de estos años.

Al Dr. Víctor J. González Villela por el apoyo durante la realización del trabajo y compartir con nosotros la experiencia y conocimientos necesarios para completarlo exitosamente.

A Alex, por acompañarme en este muy interesante y lleno de sorpresas camino de la realización de este trabajo, así como a todos mis amigos que hicieron de la estancia en la Universidad Nacional Autónoma de México una experiencia increíble.

De Alex:

A Dios, por llenar mi vida de personas y momentos maravillosos, por acompañarme siempre y por concederme la alegría de ver realizado uno de mis más grandes sueños.

A toda mi familia, por el amor y apoyo que me ha brindado a lo largo de mi vida entera.
A mis papás, por motivarme siempre a dar lo mejor de mí y por facilitarme todo lo necesario para lograr alcanzar esta meta, por el cariño, comprensión, consejo y ayuda que me dan.
A mi hermanito, por darme su buena compañía y alegrar mis días.

A mi novia Lesliee, por el amor y cariño que me da, por motivarme y tenerme paciencia.
A todos mis amigos, por acompañarme en este camino y hacerlo más ameno y satisfactorio.
A Moisés, por ser un excelente compañero de tesis y por todo el trabajo realizado.

Al Dr. Víctor J. González Villela, por la orientación, paciencia y apoyo en la realización de este proyecto, por el tiempo y recursos dedicados para su desarrollo.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería, por la excelente formación que me han proporcionado y por tantas oportunidades a lo largo de mi trayectoria académica.

2. Resumen

En este trabajo se presenta el desarrollo de un modelo para el control de un enjambre robótico mediante la integración del algoritmo de campos potenciales y un método de inteligencia de enjambre conocido como *flocking*. Estos se adaptan en campos de velocidades para ser combinados eficientemente y poder ser aplicados directamente en el control de robots móviles. El sistema reactivo resultante tiene la capacidad de lograr el desplazamiento del enjambre robótico evadiendo obstáculos del entorno y evitando colisiones entre los diferentes agentes que lo conforman, además, genera formaciones que se adaptan a las condiciones del ambiente. Finalmente, se analiza el desempeño del modelo mediante simulaciones computacionales e implementándolo en un banco de pruebas físico, controlando un sistema conformado por robots diferenciales en pruebas de desplazamiento a un objetivo fijo y de seguimiento de trayectorias.

Abstract

In this thesis we present the development of a model for the control of a robotic swarm by integrating the algorithm of potential fields and a swarm intelligence method known as flocking. These are adapted in speed fields in order to be combined efficiently and to be applied directly in the control of mobile robots. The resulting reactive system can achieve the displacement of the robotic swarm evading obstacles from the environment and avoiding collisions between the different agents that compound it, in addition, it generates formations that adapt to the conditions of the environment. Finally, the performance of the model is analyzed through computer simulations and with the implementation in a physical test bench, controlling a system made of differential robots in tests of displacement to a fixed objective and trajectory tracking.

3. Introducción

Hoy en día la implementación de sistemas que utilizan inteligencia artificial es fundamental debido al incremento de interacciones entre sistemas autónomos, así como la interacción con su entorno físico. A su vez, se ha incrementado la búsqueda de modelos y algoritmos simples que faciliten la implementación práctica de los diversos sistemas. Por otro lado, siguiendo la línea de investigación del *Mechatronics Research Group* de la Facultad de Ingeniería de la UNAM (MRG) enfocada en el desarrollo de modelos para la robótica móvil, se busca encontrar un modelo modular, sencillo y fácilmente aplicable para el control de diversas cantidades de robots móviles en un ambiente controlado.

En el presente trabajo se propone un sistema basado en visión artificial que haga uso de cómputo distribuido simulado, a partir de una computadora central, para investigar la implementación de dos grandes áreas de la física artificial: campos potenciales y *flocking*. Así mismo, se propone estudiar los efectos resultantes de la interacción sencilla entre agentes, sus topologías y tiempos de estabilización al desplazarse a un objetivo determinado, tomando como base un sistema de esquemas de motorización propuesto por Arkin y ecuaciones derivadas de las investigaciones en campos potenciales en el *MRG*.

Esto se pretende alcanzar utilizando los principios rectores de la inteligencia de enjambre descritos en la sección 4.2 de este trabajo: escalabilidad, flexibilidad, localidad y robustez. Además, se propone un campo de velocidad alrededor del agente caracterizado principalmente por tres regiones:

- ***Atracción**. Sección del campo potencial en la que cualquier agente cercano se sentirá atraído por otro agente en el entorno.*
- ***Repulsión**. Sección en la cual el agente se sentirá repelido de manera que los agentes mantengan una distancia segura entre ellos.*
- ***Orientación**. El agente sólo sentirá una influencia rotacional para conducirse a una posición parecida al resto.*

El trabajo se organizó de la siguiente manera: En la sección 4 se hace un estudio y análisis de los trabajos relacionados con la línea de investigación de inteligencia de enjambre y robótica móvil. Posteriormente, en la sección 5 se presenta el planteamiento del modelo propuesto que incluye la definición teórica sobre la cinemática del robot móvil y la definición del campo potencial artificial a implementar, así como su integración con el algoritmo de *flocking* que, en su conjunto, llevarán a los agentes de un punto inicial hacia el objetivo. Después, en la sección 6 se describe el desarrollo de un simulador para validar los modelos teóricos anteriormente planteados, así como la implementación de un banco de pruebas físico para obtener conclusiones sobre el desempeño del algoritmo en un entorno no idealizado. A continuación, en las secciones 7 y 8 se presentan las pruebas realizadas, así como los resultados y su respectivo análisis para, posteriormente en la sección 9, generar conclusiones relevantes con la información desarrollada, así como proponer posibles trabajos a futuro para la continuación de esta línea de investigación en la sección 10.

4. Estado del arte

4.1 Formaciones adaptativas

Uno de los objetivos principales de este trabajo es el desarrollo de un sistema para evadir obstáculos que tenga la capacidad de mantener una formación entre los agentes involucrados en el ambiente de trabajo. El interés en los sistemas multiagente y la capacidad para poder controlarlos eficazmente se basa en su eficiencia, reducción de costos y la robustez en su control [1]; ya que, al contar con información parcial o global sobre su entorno, los sistemas multiagente son capaces de mantener, reforzar y adaptar su formación, para llevar a cabo su tarea en un tiempo finito. Para diversas tareas resulta más eficiente tener varios agentes actuando simultáneamente en lugar de uno solo, además, la falla de alguno de los agentes no implica una falla completa del sistema, sino que el resto de ellos pueden continuar para cumplir el objetivo. Todo esto se puede lograr mediante diferentes tipos de algoritmos como: métodos basados en el comportamiento, métodos de control descentralizado y leyes de control locales, incluyendo al control mediante campos potenciales [2] que se adopta en este trabajo. Las formaciones adaptativas son parte del estudio de sistemas multiagente en los cuales se involucran dos o más elementos de un ambiente para correlacionarse, moverse y colaborar, al mismo tiempo que consideran y reaccionan a la información espacial parcial o global de su entorno.

Es importante aclarar que la palabra *centralizado* puede ser utilizada para referirse al sistema de adquisición de datos o al método de control de la formación. Haciendo referencia al sistema de adquisición de datos puede hablarse de sistemas centralizados y sistemas descentralizados.

1. Los sistemas de adquisición de datos globales o centralizados son aquellos con la capacidad de observar la totalidad del espacio de trabajo y dirigen los movimientos de los agentes desde un mismo sistema de procesamiento.
2. Los sistemas de adquisición de datos descentralizados, por otro lado, son capaces de obtener datos locales de los elementos cercanos para, posteriormente, reaccionar según los cálculos realizados con datos parciales del entorno.

Con respecto al método de control para una formación se puede hablar de los siguientes tipos:

1. *Centralizado*. Son aquellos que tienen un agente encargado de mantener el orden en la dirección de la formación [3]. También se pueden encontrar trabajos donde no es un agente el encargado de la dirección general del grupo, sino que se encarga de ello un punto específico que finalmente será declarado el centro de la formación. Uno de los métodos más comunes es el *leader-follower*[1], [3].
2. *Descentralizado*. En este tipo de control no existe un líder claro de la formación, sino que, en ocasiones, se programan las mismas capacidades en cada uno de los elementos para que sean capaces de actuar reactivamente durante la ejecución del experimento y moverse a la par. Esta situación es útil para la evasión de obstáculos dado que los diferentes agentes involucrados son capaces de moverse independientemente, pero, a pesar de ello, mantenerse unidos durante o después del proceso de evasión del obstáculo. Un ejemplo de esto se da en el artículo presentado por J. Li [4], donde se busca que cualquiera de los

elementos sea capaz de detectar el obstáculo, reaccionar ante él y transmitir la información al resto, lo cual provoca una mayor flexibilidad en la formación.

Existen diversos trabajos dedicados al estudio de formaciones y grupos de múltiples agentes robóticos, incluyendo casos de investigaciones en los que se presenta un control de agentes basado en información local, donde a partir de la información detectada por cada uno de los sistemas robóticos móviles se determina la acción a realizar [1]. En este trabajo se ocupa el sistema de control centralizado de formación *leader-follower*, en el cual uno de los agentes es declarado como aquél que guiará al resto hacia el punto objetivo, mientras que los demás actuarán en consecuencia detectando la cercanía entre sus pares y manteniendo una formación estable hasta llegar al punto indicado. En el trabajo *Animal group behavioral model with Evasion Mechanism* [3] se presenta una estrategia local donde el objetivo es simular el movimiento de una multitud de agentes, esto en función de economizar tiempos de animación utilizando un sistema basado en el comportamiento natural de los animales de repulsión contra un depredador (Figura 1).

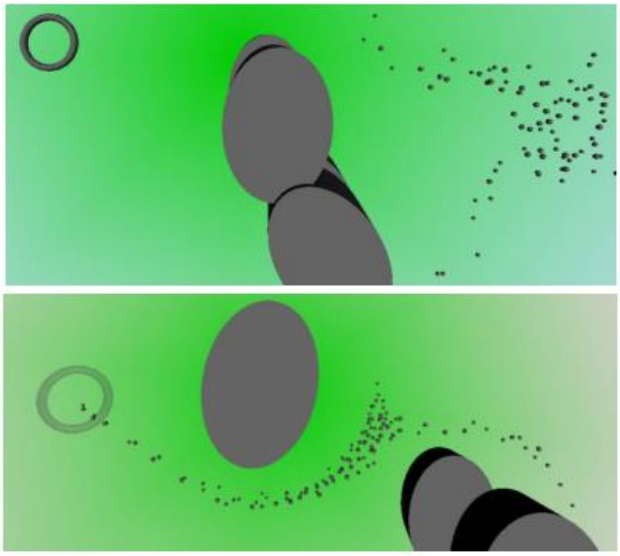


Figura 1. Elementos animados utilizando el algoritmo de evasión planteado en [3]

Algunas investigaciones hacen uso de métodos de control de formaciones basadas en adquisición de datos centralizada utilizando un sistema global que detecta la posición de los agentes, de modo que un solo sistema de procesamiento de datos sea el encargado de calcular trayectorias y distancias. Sin embargo, esto hace muy complicada la expansión y la escalabilidad del experimento, así como también, al incrementar el número de agentes involucrados en la formación, se provoca un incremento exponencial en la capacidad computacional necesaria para poder controlar a los agentes correctamente sin errores en la transmisión de datos. Por lo anterior, los sistemas computacionales descentralizados, en los que se realizan cálculos locales sobre posición global, velocidad y posición relativa con respecto a sus pares, se han hecho más comunes y es un área de intensa investigación [5].

Como se mencionó anteriormente, una de las técnicas más comunes para el control de formaciones adaptativas enfocadas en la evasión de obstáculos es el modelo *leader-follower* que se puede encontrar en muchos proyectos [1]–[3], [6]. Este ejemplo forma parte de un modelo rígido de

distancias predefinidas para la formación en sistemas multiagente, pero también se encontraron algunas investigaciones que ocupan líderes virtuales que son un ejemplo de una aproximación al problema de formaciones en el ámbito de topologías adaptables y control descentralizado de los agentes [7]. Una parte interesante de este tipo de experimentos es que tanto el cómputo de los valores de priorización entre agentes como el movimiento de cada uno de ellos es realizado de forma descentralizada, garantizando su independencia y una alta adaptabilidad en entornos cambiantes [7].

Por otro lado, autores como Gouzheng et al. [2] comentan el surgimiento del método mediante campos potenciales como elemento de control de sistemas multiagente, debido a su simpleza y la economía computacional que involucra en sistemas reactivos que se corren en tiempo real. Este método se contrapone a propuestas deterministas de planeación de trayectorias debido a que dicho algoritmo modifica la trayectoria del agente reactivamente a lo largo de la ejecución del experimento, mientras que apoya a la descentralización de la información generando un entorno plenamente adaptable, creando agentes independientes sin líderes absolutos ni virtuales. Este método de control se estudiará más adelante debido a su intensivo uso en diferentes artículos de robótica móvil [6], [8]–[12].

En este punto vale la pena hacer la distinción entre dos tipos distintos de sistemas de control de robots: los sistemas deliberativos y los sistemas reactivos. [13]

Sistemas deliberativos: Se caracterizan por determinar el comportamiento del robot a partir de un conocimiento global del entorno en el que se va a desarrollar. Por ejemplo, para planear una ruta o trayectoria se parte de que son conocidas las características del terreno o ambiente en el que se van a mover los robots, así como de los obstáculos u objetivos presentes en el camino. A partir de esta información se puede determinar la mejor trayectoria de acuerdo con el objetivo particular que se tenga. En los sistemas deliberativos el control se hace de forma predictiva y planeada tanto temporal como espacialmente.

Sistemas reactivos: Por otro lado, los sistemas reactivos determinan las acciones de los robots con base en la información que va llegando de forma continua del entorno. Es decir, en este caso no se tiene un conocimiento completo del ambiente, sino que se van utilizando los sensores en tiempo real para ir modificando y decidiendo el comportamiento de los robots. Considerando que la robótica de enjambre busca que cada uno de los agentes responda de manera individual al entorno y al comportamiento del enjambre, se habla de que no existe un control centralizado y tampoco un conocimiento previo de todo el entorno que rodea a los robots, por lo que se trata principalmente de un sistema reactivo que va a comportarse de acuerdo con las señales que provean los sensores de los agentes.

4.2 Inteligencia de enjambre

4.2.1 Antecedentes generales

La inteligencia de enjambre se centra en el estudio del comportamiento colectivo de un grupo de elementos que realizan tareas o actividades en conjunto, mejorando las capacidades que tendría un

único elemento por sí mismo. En principio, la inspiración para el desarrollo de esta clase de algoritmos surge a partir de la observación de muchos tipos distintos de animales e insectos que presentan una organización y comportamiento específico que les permite interactuar en grupos muy numerosos y lograr sus objetivos, como puede ser transportarse de un lugar a otro en conjunto, la búsqueda de alimento o provisiones y la exploración de algún espacio.

Este tipo de inteligencia colectiva se presenta en un gran número de ejemplos en la naturaleza y ha sido observado desde hace mucho tiempo, sin embargo, alrededor de 1980 comienzan a desarrollarse trabajos con la intención de explicar estos comportamientos con una base matemática y de modelar sus principios y funcionamiento. En el artículo [14] publicado por *D.J.T. Stumpter* se presentan los resultados y ejemplos del análisis del comportamiento colectivo, desde el estudio de grupos de aves, el movimiento de las hormigas generando rutas para regresar al hormiguero, e incluso se presentan actividades humanas que pueden ser modeladas mediante algoritmos de inteligencia de enjambre como es el caso de los aplausos que se generan entre la multitud tras alguna función. Cada uno de estos comportamientos, entre muchos otros, tienen un principio de funcionamiento que permite la organización del grupo sin la necesidad de que exista algún líder en particular, si no que cada uno de los elementos encuentra la manera de seguir la organización colectiva desde su propia percepción individual. Por ejemplo, en el caso de las hormigas, se ha descubierto que se guían con base en algún tipo de feromona que van dejando a su paso tras descubrir alguna fuente de comida. De esta manera, las siguientes hormigas siguen el camino de feromonas dejado por las primeras hasta llegar a la comida y, al mismo tiempo, refuerzan la señal para las hormigas que vendrán a continuación. Así mismo, resultan ser los caminos más cortos los que son más reforzados y que presentan una señal de feromonas mayor, de tal manera que al final la mayoría de las hormigas termina encontrando la comida y tomando los caminos más cortos para llegar a ella [14]. Por otra parte, el movimiento de grupos de aves o de peces se rige de una forma completamente diferente, ya que puede ser explicado por tres reglas básicas, la repulsión por parte de los elementos vecinos más cercanos, la orientación con los elementos que se encuentran a su alrededor y la atracción por parte del resto del grupo para evitar quedar aislado del mismo.

4.2.2 Características de la inteligencia de enjambre.

De acuerdo con el libro *Design and Control of Swarm Dynamics* publicado por R. Bouffanais en el año 2016 [15], la inteligencia de enjambre debe caracterizarse por los siguientes aspectos :

i) No existe un líder general que controle el comportamiento del enjambre a partir de un conocimiento global del sistema.
ii) Se tiene una percepción local del entorno por parte de los individuos, que puede llevar a un cierto nivel de conocimiento global a partir del intercambio compartido de información.
iii) Un alto grado de adaptabilidad ante diferentes circunstancias que pueden ser a su vez cambiantes.

Así mismo, se tienen los siguientes principios que denotan también ventajas de este tipo de sistemas:

i) *Robustez*: El sistema continúa funcionando aún con elementos faltantes o con un mayor número de elementos.
ii) *Flexibilidad*: Capacidad de adaptación rápida a entornos cambiantes.
iii) *Localidad*: Capacidad de los agentes para sensar su entorno y tomar decisiones a partir de lo detectado, sin depender de un elemento externo.
iv) *Escalabilidad*: Los mismos principios pueden emplearse para enjambres conformados por una cantidad mayor o menor de agentes.

De igual manera, se menciona que, para poder comprender correctamente el comportamiento colectivo, es necesario hacer uso de los siguientes conceptos:

- *Interacciones entre agentes*: hacen referencia a las diferentes formas de intervenciones físicas que llegan a presentar unos agentes con otros, por ejemplo, el contacto mecánico o mediante interacciones térmicas o de algún otro tipo de relación física.
- *Intercambio de información*: en el comportamiento de enjambre de un grupo de elementos, más aún si se trata de un enjambre de agentes artificiales, resulta muy importante la transmisión de información entre ellos, ya sea de forma unidireccional o bidireccional. Mediante estos intercambios de información es posible generar un conocimiento mayor a partir de la suma de las perspectivas de varios agentes que permitan tomar una mejor decisión de comportamiento para el grupo.
- *Procesamiento de información*: la información, ingresada a partir del intercambio de información con otros agentes del grupo o por medio de un sensado directo del ambiente, es procesada por los agentes para evaluar las condiciones y el estado en el que se encuentra el mismo agente y el ambiente que lo rodea.
- *Algoritmo de comportamiento y respuesta*: la información adquirida y procesada es utilizada finalmente por el agente para determinar el comportamiento que tendrá a continuación con base en las características del propio agente, ya sea su información genética y biológica para los enjambres de seres vivos o los algoritmos programados en el caso de los enjambres de agentes artificiales.

Existen diferencias significativas entre un tipo de comportamiento de enjambre y otro, así como entre diferentes tipos de agentes. Dichas diferencias radican principalmente en 3 importantes aspectos. El primero de ellos es la capacidad de sensado que tiene cada uno de los agentes, es decir, la información que puede recibir de manera individual cada uno de los agentes que integran el enjambre sobre el entorno que lo rodea. Un segundo aspecto que caracteriza los comportamientos colectivos son las habilidades cognitivas que poseen los agentes y como tercer aspecto la manera en que responden a la información recolectada y procesada. [15]

4.2.3 Antecedentes en la robótica

La inteligencia de enjambre aplicada en robots, conocida como robótica de enjambre [16], se basa en la conjunción de robots, de tal suerte que no desarrollan actividad intelectual únicamente de forma individual cada uno de ellos, sino que se encuentran capacitados para generar inteligencia colectiva que provoque a su vez un comportamiento inteligente en la generalidad del enjambre.

Dentro de dicho comportamiento de enjambre se pueden localizar según Dudek et al. [17] cinco áreas de investigación principal:

- Tamaño del enjambre.
- Rango de comunicaciones.
- Topología de la comunicación.
- Ancho de banda de la comunicación.
- Reconfiguración del enjambre y capacidad de procesamiento de la unidad del enjambre.

Al variar cualquiera de ellas, se puede dar un cambio en el comportamiento general del enjambre robótico que puede ayudar a completar tareas referentes a conflictos en la búsqueda, rastreo, fundamentos de cooperación, aprendizaje y configuraciones geométricas [16]. Mientras que, aunado a lo anterior, la inteligencia de enjambre aplicada a robótica también es útil para dar características de robustez, flexibilidad y escalabilidad (vistas como parámetros fundamentales en la inteligencia de enjambre en la sección anterior). Por otro lado, L. Bayindir et al. [18] mencionan que lo más adecuado es dejar de lado el tamaño y la complicación de la comunicación debido a que rompe con los principios de la inteligencia de enjambre y en vez de ello, utilizar una comunicación abierta en forma de *broadcast* (emisión abierta sin destinatario), ya que dota al sistema de anonimidad en sus interacciones dejando a la robótica de enjambre subdividida en el siguiente orden jerárquico:

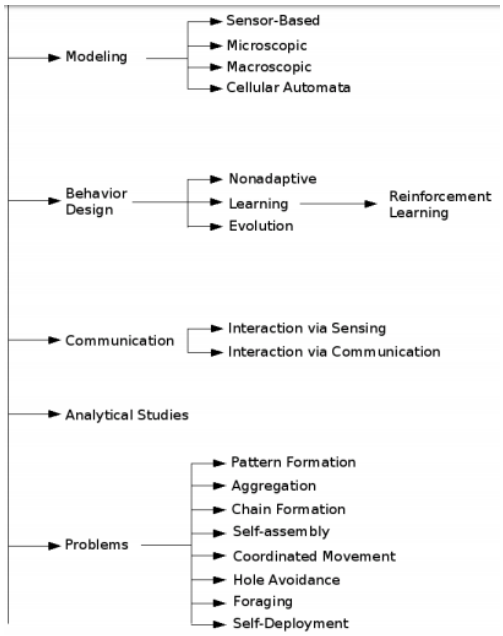


Figura 2. Jerarquización de las investigaciones realizadas sobre robótica de enjambre según L. Bayindir. [18]

La jerarquización anterior hace entonces evidente cinco ramas principales de investigación:

1. Modelado de sistemas de enjambre.
2. Diseño del comportamiento.
3. Comunicación
4. Estudios analíticos
5. Problemas

Estas ramas de investigación son especialmente importantes debido a que dan pauta para conocer el estado del arte detrás de la robótica de enjambre, las investigaciones realizadas al respecto y las posibles tendencias en este ámbito.

Modelado

En este tipo de investigaciones se trata de encontrar las ecuaciones matemáticas que modelan de forma representativa el comportamiento de los robots bajo algoritmos de enjambre. Una de las principales ventajas de desarrollar este tipo de modelos es que el problema de escalabilidad para validar los resultados bajo experimentos que involucran una multitud de agentes robóticos se soluciona, ya que se puede validar mediante el uso de elementos de software que facilitan la simulación siendo elementos virtuales en lugar de elementos físicos, cuya manufactura involucra un problema de recursos sabiendo que para validar algunos experimentos de escalabilidad es necesario involucrar hasta a cien agentes dentro de un mismo entorno [18]. Por otro lado, se sabe que los modelos en computadora pueden no tomar en consideración algunos elementos de la vida real, pero existen elementos como simuladores de ruido en los sensores o ambientadores que pueden llevar a la simulación a una situación realista.

Diseño del comportamiento

Existen investigaciones en el ámbito de la robótica de enjambre en las cuales el principal objeto de estudio es el comportamiento adaptativo de los agentes, basándose en el comportamiento natural de enjambres de animales que modifican su comportamiento según su entorno. Con base en esto se han generado investigaciones con control manual, inteligente (aprendizaje) y evolutivo [18].

Considerando lo anterior, también ha habido investigadores que han enfocado el estudio en el control del comportamiento centralizado y distribuido, siendo este último de donde se derivan los modelos basados en campos potenciales [16], y de lo cual se hablará más adelante en este trabajo.

En el mismo eje de la investigación, se han encontrado métodos para evaluar y controlar el comportamiento de un conjunto de robots basándose en redes neuronales y algoritmos genéticos que generan información local del entorno de los agentes para que, en su conjunto, se genere información generalizable para el resto del enjambre, con lo que es posible generar una mejor adaptación del comportamiento.

Comunicación

En entornos de trabajo con múltiples agentes físicos involucrados, la forma en la que son capaces de comunicarse y percibir su entorno es fundamental para entender el comportamiento de enjambre. Un ejemplo de ello son las investigaciones de S. Camazine et al. [19] donde se ejemplifica la comunicación mediante el entorno, tomando como modelo el comportamiento de las hormigas, las cuales son capaces de modificar químicamente su entorno de modo que pueden dejar un rastro que, posteriormente, agentes pares pueden detectar e interpretar. También se han dado casos de

investigaciones donde el medio de comunicación principal funciona mediante broadcast, el cual consiste en la emisión de un mensaje general a todo aquel agente dentro de un rango de recepción del mensaje, con lo que se busca una anonimidad por parte del emisor. Estas investigaciones dan una ramificación sobre la forma en la que los agentes robóticos son capaces de transmitir información entre sus pares [18]:

- Interacción por sensado.
- Interacción por comunicación

Interacción por sensado

Este tipo de comunicación se basa en el principio de *reconocimiento de parentesco* que ha sido descrito en la naturaleza como un método por medio del cual los animales en manada son capaces de colaborar y protegerse entre ellos para combatir depredadores y buscar comida. El reconocimiento de parentesco se aplica en la robótica de enjambre dotando a los agentes de la capacidad de discriminar a sus pares con respecto a quienes no lo son. Esto es benéfico debido a que en diferentes investigaciones se hace uso de esta propiedad para mejorar el funcionamiento del algoritmo de enjambre, como en el artículo escrito por O. Soysal [20] en el cual se busca una diferenciación entre los límites del espacio de trabajo del enjambre y los agentes involucrados para una interacción eficiente al resolver la tarea en conjunto mediante un algoritmo probabilístico.

Se puede notar que, en la comunicación entre agentes en este tipo de relación, no se busca entablar una transmisión de datos agente – agente de forma deliberativa, sino simplemente realizar un sensado de su entorno y, con ello, generar algoritmos de comportamiento.

Interacción por comunicación

Este tipo de comunicación no se recomienda para la robótica de enjambre debido a que limita la escalabilidad y flexibilidad del sistema [18], ya que también se requiere de un sistema de comunicación más complejo que considere la comunicación agente – agente y su habilidad de enviar mensajes tipo *broadcast*.

Existen diferentes trabajos que ejemplifican esto [21], [22], donde se utiliza un sistema de envío de datos visual en el cual los agentes son capaces de recibir información sobre el estado de sus pares por medio de una luz LED colocada en su cuerpo. Esta luz cambia de color dependiendo de si el robot se encuentra anclado a el enjambre, se encuentra en el "nido" (home) o si se encuentra en ruta para incorporarse a alguna de las dos posiciones anteriores.

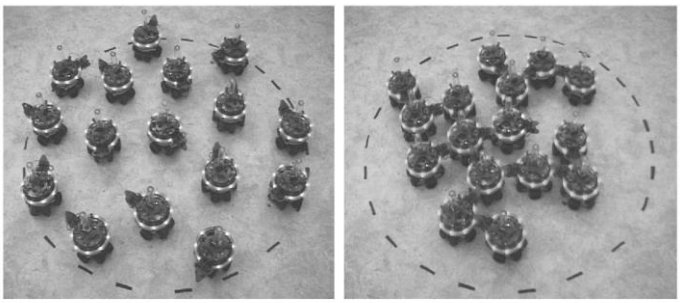


Figura 3. Ejemplo de enjambre autoensamblable. [22]

Estudios analíticos

En este tipo de estudios se ha dejado en un segundo plano el desarrollo en hardware y se han desarrollado más bien algunos artículos, como por ejemplo el escrito por Balch [23], en donde se hablan de algunas métricas para cuantificar el desempeño de los enjambres en el desarrollo de una tarea específica. En dicho artículo se menciona la importancia de tomar en cuenta la *Entropía Social* para medir la diversidad de un grupo de robots.

Problemas

Finalmente, algunos de los problemas que se han investigado utilizando la robótica de enjambre son los siguientes [18]:

- *Formación de patrones*. Estos artículos hablan sobre la formación de patrones generales a partir de la interacción local de los agentes y cómo la modificación de los parámetros involucrados para la interacción entre agentes puede modificar parámetros de la formación global.
- *Agregación de agentes*. Debido al principio de escalabilidad de la inteligencia de enjambre, la posibilidad de poder agregar agentes de forma sencilla es una parte fundamental de los estudios. Si se tiene un control centralizado es sencillo agregar o quitar agentes del entorno pero, como se pueden mostrar en algunas investigaciones [22], [24], al tener un control distribuido el problema no es trivial. En estas aproximaciones al problema se utiliza un patrón probabilístico sobre la agregación de agentes; mientras que en otro trabajo [25] se realiza una solución del problema mediante el uso de una red neuronal.
- *Formación en cadena*. Este problema consiste en la generación de una línea de agentes que puede ser de utilidad al tener que pasar por una trayectoria definida y limitada. En el trabajo *Chain Formation in a Swarm of Robots* [21] se aborda este problema, de modo que cada uno de los agentes tenga que buscar el final de la cadena y mantenerse en esa posición hasta que el tiempo de exploración termina.
- *Auto ensamblaje*. Este problema consiste en la generación de patrones complejos a partir de unidades relativamente simples bajo reglas locales [18].
- *Movimiento coordinado*. Este problema consiste en la generación y mantenimiento de un patrón global mientras el enjambre se mueve. Fue abordado mediante un algoritmo de evasión de obstáculos utilizando el cambio del centro de masa del enjambre y aplicando a dicho centro una velocidad relativa al sensado local de cada uno de los agentes. [26]
- *Búsqueda*. En este problema, el objetivo es buscar a una presa predefinida para el enjambre y llevarla hasta la base o home de los agentes. Este problema se ha abordado utilizando un sistema auto organizado [26] y con un sistema de aprendizaje basado en el método Monte Carlo [27].
- *Auto desarrollo*. Este problema consiste en que los agentes en el ambiente, distribuidos aleatoriamente, sean capaces de desarrollarse en su entorno de forma autónoma. Siendo que los robots sólo cuentan con información ambiental local [18]. Este problema fue abordado por Howard et al. aplicando teoría de campos potenciales para maximizar el área explorada y midiendo porcentaje de cobertura del ambiente y tiempo de estabilización de los agentes [28].

Como se puede apreciar hasta este punto, los avances en la inteligencia de enjambre dejan ver como conclusión, que la tendencia en los trabajos referentes a la robótica de enjambre se basa en la optimización de la colaboración de los agentes de modo que puedan desarrollar sus tareas eficientemente. Al mismo tiempo, el respeto a los principios fundamentales como escalabilidad y localidad de la información son factores que guían a las investigaciones de modo que se trata de encontrar nuevas formas de control a partir de información parcial del entorno. A su vez, también se puede observar que existen pocos parámetros cuantitativos propios del área que sirvan para medir la eficacia de los algoritmos.

También cabe mencionar que los autores, en la mayoría de los textos, hacen especial énfasis en la comunicación y el método por el cual los agentes se mantienen ligados y comparten información sobre su entorno, así como la importancia y la influencia de lo anterior sobre el diseño del comportamiento. Esto debido a que, al tener una masa de información basada en la localidad de los agentes, los patrones globales que emanan de su interacción ayudan a incrementar el desempeño respecto a un solo agente actuando independientemente.

De acuerdo con la tesis doctoral escrita por Luisa y Tortosa [29], para que el sistema del enjambre de robots cumpla realmente con las características de la inteligencia de enjambre, por ejemplo, que sea robusto contra fallos y que pueda ser escalable a un número mayor o menor de elementos, se debe cumplir lo siguiente.

- Los robots que forman el enjambre deben ser robots autónomos situados en el entorno.
- El enjambre debe estar formado por un gran número de robots que podrán dividirse en grupos más pequeños de robots homogéneos.
- Los robots serán relativamente simples.
- Los robots deberán disponer de sensores locales y capacidades de comunicación limitadas.

Además, señala las principales ventajas que ofrece el uso de la robótica de enjambre como las siguientes:

- Los sistemas robóticos son robustos y tolerantes a fallos, ya que pueden seguir en funcionamiento ante el fallo de alguna unidad.
- Son sistemas con una alta escalabilidad, donde el tamaño del enjambre puede ser aumentado o disminuido según la tarea lo requiera.
- Enfatizan el uso de paralelismo, donde un conjunto de robots puede llevar a cabo una tarea más rápidamente que un único robot, descomponiendo la tarea en subtareas y ejecutándolas de manera simultánea.
- Los sistemas de enjambre suelen ser más económicos.

4.2.4 Algoritmo de *flocking*

Ahora bien, dentro del ámbito de la inteligencia de enjambre existe el algoritmo de *flocking* (proveniente del término congregar en inglés). Éste consiste en un método mediante el cual es posible coordinar grandes volúmenes de agentes de forma que se preserve la integridad individual

de los involucrados al no colisionar y también se mantenga la integración como enjambre al tener en cuenta que los individuos tiendan a estar juntos.

Este algoritmo tiene sus principios en la etología [13], una rama de la biología que se encarga del estudio del comportamiento de los animales para la obtención de modelos matemáticos que explican la organización y funcionamiento de su comportamiento para que, posteriormente, puedan ser aplicados en diferentes ámbitos de la ingeniería siguiendo como ejemplo el siguiente diagrama de flujo [13].

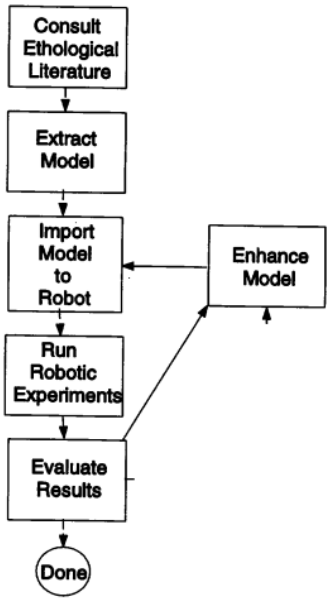


Figura 4. Método para la obtención de un modelo a partir de la etología según Arkin. [13]

Siguiendo dicha tendencia, Sumpter [14] menciona algunos algoritmos basados en la interacción entre insectos y animales sociales entre los cuales es de destacar, para propósitos de este trabajo, el que se basa en el comportamiento de algunas aves y peces. Se trata de un algoritmo que marca zonas de influencia en las cuales los agentes involucrados (peces o aves) son capaces de distinguir entre tres diferentes acciones: repulsión, atracción u orientación. Con esos tres comportamientos básicos un conjunto de animales es capaz de mantener las siguientes propiedades:

i) Alejarse de los demás individuos localizados en las cercanías.
ii) Adoptar la misma dirección de los demás individuos cercanos.
iii) Evitar quedar aislados o separados del grupo.

Como se ha visto, este comportamiento se puede agrupar en la inteligencia de enjambre debido a que cada agente, localmente, es capaz de captar su entorno y reaccionar en consecuencia, con lo cual, es posible delimitar las zonas de influencia:

1. *Zona de repulsión*. Mantiene la integridad de los individuos al evitar colisiones en el enjambre
2. *Zona de atracción*. Mantiene la integridad del enjambre como un todo y evita que los agentes involucrados se alejen entre sí.

3. *Zona de orientación*. En esta zona cada uno de los agentes ajusta su dirección en un promedio ponderado de la orientación de los agentes en su vecindad.

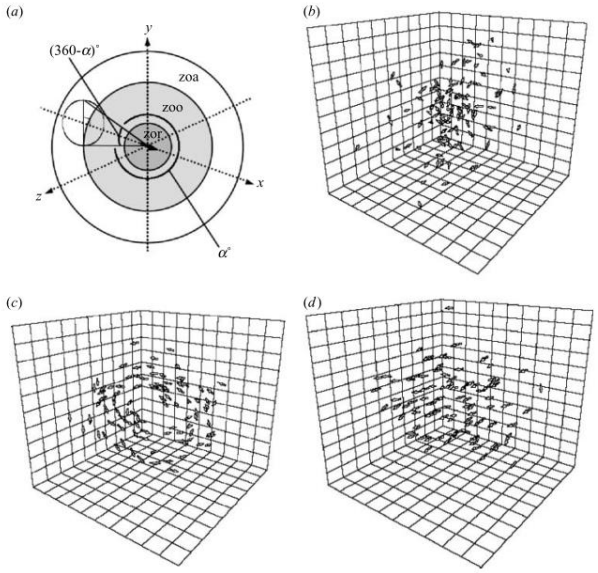


Figura 5. (a) Zonas de influencia de los agentes. (b) Inicio del algoritmo con localización de los agentes aleatoriamente. (c) Trayectoria de los agentes siguiendo las reglas de flocking. (d) Formación paralela en movimiento. [14]

4.3 Campos potenciales

Los campos potenciales artificiales se postularon por Khatib en 1985 [30], quien propone que, si se coloca un objeto en un campo de fuerza producido a partir de la atracción de su meta o punto objetivo y fuerzas repulsivas por parte de obstáculos, los cuales se encuentran modelados como superficies repulsivas, es posible que dicho objeto pueda moverse a través del espacio sin colisionar con los obstáculos. Dicho procedimiento se efectúo en un manipulador estático (PUMA 560) [30], aunque en décadas recientes dicha investigación se ha ampliado en aplicaciones utilizando robots móviles como agentes del espacio de trabajo [9]–[12], [31].

Una metáfora que resulta interesante [12] describe al campo potencial artificial como una manta, colocando a la meta en el punto más bajo, mientras que los obstáculos se representan como elevaciones de la manta. Al dejar caer un balín, es posible percatarse como, del punto inicial al punto final, el balín esquivará las elevaciones de la manta (obstáculos) hasta llegar al punto más bajo del espacio de trabajo (meta u objetivo).

La técnica de campos potenciales artificiales (CPA) se basa en la suma vectorial de la atracción de una meta y la repulsión de los n obstáculos que pueden presentarse en el espacio de trabajo del robot. La definición matemática de dichas fuerzas varía según los objetivos de la investigación, pero en general se implementa un sistema proporcional a la distancia entre el agente y la meta, e

inversamente proporcional a la distancia de la superficie del obstáculo. De esta forma, para cada punto del espacio, es posible asociar una fuerza resultante correspondiente a la que el agente está sujeto. Un ejemplo de la definición del campo vectorial, se muestra a continuación, siendo el utilizado en [30] y [9], entre otras investigaciones, para generar el campo vectorial.

$$\boldsymbol{F}(\boldsymbol{q}) = \boldsymbol{F}_{att}(\boldsymbol{q}) + \boldsymbol{F}_{rep}(\boldsymbol{q}) \qquad (1)$$

$$\boldsymbol{F}_{rep}(\boldsymbol{q}) = \nabla U_{rep} = \begin{cases} \eta \sqrt{\left(\frac{1}{d} - \frac{1}{d_0}\right)} \left(\frac{\boldsymbol{q} - \boldsymbol{q_o}}{|q - q_o|^3}\right), & si\ d \le d_0 \\ 0, & si\ d > d_0 \end{cases} \qquad (2)$$

$$\boldsymbol{F_{atr}}(\boldsymbol{q}) = -\boldsymbol{\nabla} U_{atr} = -\xi(\boldsymbol{q} - \boldsymbol{q_a}) \frac{1}{\boldsymbol{q} - \boldsymbol{q_a}} \qquad (3)$$

Donde se puede observar que la fuerza total del campo vectorial $\boldsymbol{F_q}$ será la suma de la fuerza atractiva $\boldsymbol{F_{atr}}$ y la fuerza repulsiva $\boldsymbol{F_{rep}}$, siendo q la posición del robot, d_0 la distancia de influencia del obstáculo y q_a la posición del objetivo. Considerando η como una constante proporcional para la fuerza de repulsión y ξ la correspondiente para la fuerza de atracción. Siendo ésta, de nuevo, la aproximación clásica al problema de la planeación por campos potenciales.

Cabe destacar que diferentes investigadores como D.H. Kim [31] proponen modificaciones sustanciales a la definición original del campo potencial, debido a diferentes problemas que se presentan con este modelo, haciendo una definición que proporcione un tiempo de estabilización menor o que considere la inclusión de varios robots al mismo tiempo generando un sistema multiagente [6], en el cual se propone un sistema líder-seguidor para controlar el movimiento de varios agentes en un entorno para llegar del punto A al punto B. Algunas deficiencias del método se encuentran en las suposiciones que se hacen, en las cuales la trayectoria del líder debe ser planeada previamente y considerada como libre de obstáculos para poder llevar al resto de agentes a una orientación adecuada en el punto final. Este tipo de algoritmos es un híbrido entre un comportamiento de agentes deliberativo y reactivo dada la volatilidad en la decisión de evadir un obstáculo al encontrarlo sin conocer de antemano su presencia en el espacio.

Algunos autores como González, V. [12] mencionan diferencias sustanciales entre las técnicas basadas en la planeación de trayectorias y el método de campos potenciales artificiales, realizando una marcada diferencia entre las investigaciones dedicadas a la planeación y las que se dedican a la evasión de obstáculos. Debido a que en el caso de la planeación se involucra un conocimiento preconcebido del espacio de trabajo, lo cual implica complicaciones como la estimación de tiempos de llegada, posibles colisiones con obstáculos mal calculados y la baja capacidad de escalabilidad al no poder involucrar fácilmente un sistema multiagente utilizando dichos métodos; por esto, se concluye que un sistema de control robótico robusto es aquel que conjuga tanto un sistema de planeación de trayectoria como un sistema reactivo de evasión de obstáculos como lo es el de campos potenciales artificiales. Un ejemplo de ello se puede ver en el artículo de López Padilla[11] donde se ocupa un sistema de robótica móvil diferencial compuesto por un solo agente al cual se preprograma una trayectoria específica, como una línea recta del punto inicial al punto final o una serie de rectas formando polígonos regulares. A dicho sistema de planeación de trayectorias se le incorpora un detector láser para medir la distancia a la que se encuentra de un posible obstáculo y,

de esa manera, calcular mediante un campo potencial artificial una nueva trayectoria que lo evada. En esta investigación se puede observar la mezcla que menciona González, V. sobre el comportamiento deliberativo y reactivo trabajando en conjunto para generar un sistema de navegación autónomo robusto. Otro ejemplo donde se conjuga un sistema de planeación con un sistema reactivo de evasión de obstáculos lo tenemos en la coexistencia multiagente de la que se habló anteriormente [6].

El sistema reactivo de CPA es un creciente campo de investigación debido a que presenta una fácil implementación así como alta flexibilidad al tener la capacidad de trabajar en entornos cambiantes y complejos [11], [30] que, hoy en día, es una capacidad fundamental que deben tener y considerar los agentes robóticos en ambientes destinados a la coexistencia multiagente, tanto con humanos como con agentes robóticos. El CPA, por lo tanto, según Osorio-Comparán [9], es el método más utilizado en la implementación robótica para el trazado de trayectorias así como el más convenientes según Zhou, L. [8] debido a que se trata de un método computacionalmente ligero, simple y adecuado en aplicaciones de simulación en tiempo real. Además, otros métodos como gradilla o encuadre requieren de una cantidad de datos por proceso enorme para poder realizar una simulación reactiva eficaz.

A pesar de tener diversas ventajas sobre otros métodos de control para trayectorias, también existen algunas desventajas [12] como el hecho de que el modelado y la simulación del robot en el entorno de trabajo debe realizarse suponiendo que el agente se trata de un punto el cual es afectado por la fuerza resultante del campo vectorial en el que se encuentra inmerso. Dicho punto, usualmente se trata del centro geométrico o de masa del robot y a su vez, para efectos prácticos, se puede generar una circunferencia de protección alrededor del punto, la cual marca los límites permisibles para el robot, modelando así al agente, como un disco. El hecho de modelar al agente con formas irregulares también es posible, pero complica el análisis del entorno [10], [12]. Esto hablando sobre los efectos del agente robótico, pero también existen algunas desventajas referentes a la definición del campo potencial.

También se da la posibilidad de la existencia de múltiples regiones mínimas además de la meta derivadas de la definición de campo. A estas condiciones se les conoce como mínimos locales y tiene un enorme efecto en el desempeño del algoritmo, debido a que tienen el efecto de frenar el avance del agente hacia el objetivo y detener el desarrollo del experimento provocando un tiempo de llegada al objetivo infinito. Para este problema existen diferentes perspectivas desarrolladas para evitar que el agente robótico se detenga. Entre ellas, existen investigadores que han tratado de solucionar el problema agregando un detector de estancamiento en el sistema observador, de modo que al detectar un momento en el cual el agente se detiene antes de llegar a la meta, se adiciona una fuerza de escape además de las ya definidas en el campo, lo que provoca que el mínimo local se mueva de posición de manera que sea posible sacar del estancamiento al agente. Dicho método tiene la desventaja de presentar tiempos de asentamiento elevados y puede llevar al agente a trayectorias poco convenientes. Además, existen configuraciones de obstáculos irregulares, como los mínimos locales provocados por formas de U (figura 6), en las cuales dicho método al ser aplicado no resuelve el problema de mínimo local.

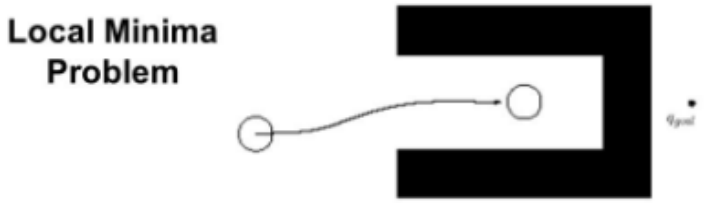


Figura 6. Ejemplificación de un mínimo local debido a un obstáculo en forma de U. [8]

También, se puede aplicar un sistema de navegación a partir de un algoritmo de campos potenciales artificiales involucrando un sistema de evasión de mínimos locales (LMA, *Local Minima Avoidance*) [9]. Éste se basa en un algoritmo de vorticidad de modo que, si se detecta un estado de estancamiento en el agente, se ponga en marcha el sistema de evasión para salir del estado de estancamiento. Una de las principales ventajas es que, sin importar las características físicas del obstáculo como su forma o localización en el entorno inteligente, el agente podría ser capaz de evadirlo. Existen algunos problemas en el artículo de Osorio-Comparán ya que el algoritmo de evasión de mínimos locales no se explica a profundidad; mientras que, discutiendo un poco el algoritmo del campo potencial, se ocupa la definición estándar implementada en 1985 por Khatib. En comparación con lo anterior, Zhou, L. [8], propone la modificación de la definición de la fuerza repulsiva del campo potencial utilizando como definición de la energía de repulsión una función Gaussiana con la cual fue posible documentar, con ayuda de una simulación, que dicho algoritmo es capaz de lograr evadir todas las presentaciones documentadas de la generación de los mínimos locales que se presentan en la figura 7. Los ejemplos anteriores se consideraron como un caso claro de métodos que, utilizando una reconfiguración en la definición matemática del campo potencial artificial, buscan evadir el problema de mínimos locales.

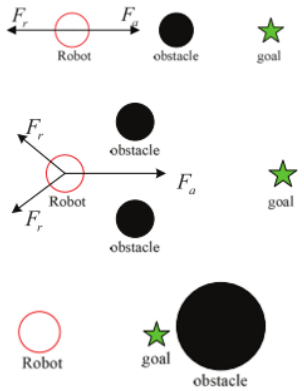


Figura 7. Presentaciones documentadas de aparición de mínimos locales. (Superior) Agente, obstáculo y meta colineales. (Medio) Dos obstáculos colocados simétricos al eje que une al agente y la meta. (Inferior) Localización de la meta en el área de influencia del obstáculo. [8]

Por otro lado, existen ejemplos que modifican reactivamente la configuración del campo para evadir este problema de mínimos locales. Un ejemplo de ello es utilizando el ángulo que forma el agente con respecto a la meta para proponer una forma distinta del campo repulsivo del obstáculo de modo que se eviten los mínimos locales. En dicha investigación [31] atribuyen la aparición de este problema al hecho de que, en el común de los casos, la definición circular del campo repulsivo

provoca que no exista una suficiente adaptabilidad por parte del entorno para lograr evadir eficazmente el problema de mínimos locales. Además, menciona Hun, D., K., et al en su artículo las complicaciones computacionales que existen al implementar modelos reactivos utilizando redefiniciones del campo potencial, debido a lo cual, se propone una redefinición reactiva que cambia la forma del campo repulsivo de un círculo a una forma adaptada específicamente a la no aparición de mínimos locales en el tránsito del agente y cuya orientación se encuentre definida por la localización relativa entre el agente y el obstáculo (figura 8).

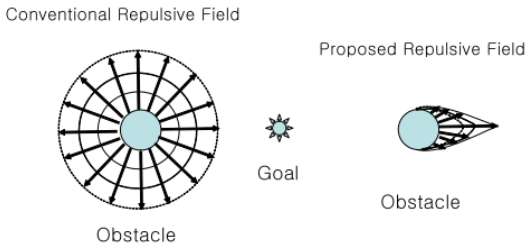


Figura 8. Campo repulsivo convencional (izquierda) contra el campo repulsivo alternativo (derecha). [30]

Así como el método mencionado anteriormente, también existen propuestas que hacen uso de una modificación reactiva del campo repulsivo para la evasión de mínimos locales, como es el caso de Rezaee, H. et al [32] donde se utiliza un campo repulsivo rotacional para que, al momento de que un agente se aproxima al obstáculo, éste sea repelido y movido al mismo tiempo, con lo que se evita que caiga en un estado de estancamiento. Simultáneamente, el campo potencial artificial mantiene en dirección al objetivo a un conjunto de agentes basados en una repulsión en forma de carga eléctrica entre ellos para evitar colisiones en el transcurso de la simulación. Cabe destacar que dicha investigación se validó utilizando una simulación computacional y no un sistema físico.

Los diferentes métodos en los cuales se aplican los campos potenciales artificiales en la robótica colaborativa, y con un solo agente, resultan interesantes y, como lo mencionan diversos autores, se trata de un método de control de trayectorias relevante debido a la simpleza de su aplicación, ya que los recursos computacionales exigidos por el método son menores que en el caso de diversos métodos de planificación deliberativa de la trayectoria como gradilla o encuadre. De igual forma, la evasión de obstáculos se puede resolver utilizando el mismo método, implementando un método reactivo para la navegación segura de agentes en un ambiente posiblemente cambiante y dinámico. Por otro lado, la investigación sobre la evasión de mínimos locales resulta de especial importancia debido a su aparición indeseada en el momento de la definición de campos potenciales artificiales, y es, en general, un tema que tocan una gran cantidad de autores.

4.4 Aplicaciones de *Flocking* y Campos potenciales

La aplicación de los campos potenciales, con ciertas reservas con respecto a las inconveniencias derivadas de la aparición de mínimos locales, resulta una de las técnicas más investigadas y una tendencia en la investigación sobre la navegación en la robótica móvil. Así mismo, también es importante mencionar la creciente necesidad de utilizar sistemas multiagente debido a que un solo

ente en un ambiente es incapaz de captar la totalidad de su entorno y, como lo han revisado ciertos autores como Spears et al. [33], existen algunas tareas como la vigilancia, defensa y exploración de terrenos desconocidos donde no sólo es más conveniente, sino más bien, necesaria, la implementación de un sistema colectivo de robots.

Como se ha observado, se han presentado algunas de las investigaciones sobre aplicaciones de navegación utilizando campos potenciales, así como la importancia y la creciente presencia de la inteligencia de enjambre como base de un cómputo distribuido de los agentes dentro de un ambiente inteligente. A pesar de ello, se notó, que no existen muchas investigaciones que logren conjugar ambas técnicas de modo que los robots móviles sean capaces de reaccionar reactivamente a su entorno y al mismo tiempo, utilizando el mismo principio de reactividad, sean capaces de mantenerse en formación. Se pretende además que lo hagan sin hacer un uso excesivo de recursos computacionales como al utilizar técnicas deliberativas como gradilla o encuadre de agentes, que resultan en ocasiones poco prácticas debido a la escasa capacidad que se tiene para ser escalables o flexibles ante cambios inevitables en el entorno de trabajo.

Existen algunas investigaciones [34] donde se utiliza un algoritmo rígido para evadir obstáculos basado en el giro a la izquierda, conocido como método de laberinto, sin embargo, se ha visto que puede llegar a provocar tiempos de exploración elevados a pesar de la reactividad en su desarrollo. En esta investigación se puede observar la presencia del algoritmo de *flocking* para mantener cohesionado al grupo, pero el enfoque de evasión de obstáculos sigue siendo simple y dirigido a una solución poco práctica.

Por el otro lado, una investigación más aproximada, llevada a cabo por Fatih, G. en [35] habla sobre los pros y contras de llevar a cabo el algoritmo de *flocking* en migraciones a gran escala. Dicha investigación hace una fuerte referencia a la biomimética abordada anteriormente ya que investiga, con el uso de robótica, el comportamiento natural de algunas aves. En esta investigación se observa la declaración de un campo potencial artificial que simula el campo magnético terrestre, el cual, se propone como unidireccional. Con esto, combinado con el algoritmo de *flocking*, logran llevar a una parvada de robots desde un punto A hasta una dirección que ellos denominan "*home direction"* a la cual se encuentra dirigido el campo vectorial que afecta a los robots. Un factor interesante es que se demuestra, utilizando los tiempos de estabilización y la eficiencia en la topología, que los mejores resultados se obtienen con la implementación de 3 robots en el espacio de trabajo, además de demostrarse la eficiencia en la implementación de *flocking* debido a que la sección de orientación y fuerzas de atracción y repulsión internas de cada robot lleva a un mejoramiento en los tiempos de estabilización del sistema completo.

Otro autor que ha mencionado las ventajas en la implementación de campos potenciales junto con *flocking* es Bleach, T. et al. [36] quien propone un método modificado de *flocking* variando algunos parámetros como: el ángulo que son capaces de formar los agentes entre sí y la distancia entre ellos, logrando modificar la topología de la formación reactivamente entre línea horizontal, vertical y diamante. A su vez, es importante mencionar que el autor, al principio de la investigación, insiste en que la implementación del cómputo distribuido es una ventaja debido a que ayuda a implementar escalabilidad, flexibilidad y robustez en los sistemas robóticos multiagente, así como aumentar el desempeño de dichos sistemas para tareas determinadas (navegación y exploración). También menciona, que la definición del campo potencial artificial está relacionada con la propuesta

por Khatib [30], así como una serie de esquemas de motor propuestos por Arkin R., los cuales funcionan como métodos programados cuyos resultados son vectores que finalmente son superpuestos para formar un solo motor o reacción resultante de todo el sistema, esto se puede observar en la figura 9.

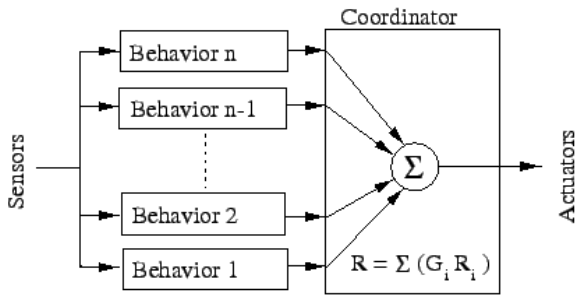


Figura 9. Ejemplificación del sistema de "motor schemas" propuesto por Arkin R. [13]

Con lo anterior Bleach, T. fue capaz de desarrollar un sistema que guiaba de manera global a los agentes de una línea de salida hasta una línea de meta a través de un ambiente con un 5% ocupado por obstáculos [36] en un tiempo determinado y finito, mientras que los agentes reportaron una formación con topología controlable por medio de los parámetros mencionados anteriormente.

Como se puede observar, la investigación utilizando los sistemas de campos potenciales como un director reactivo para la evasión de obstáculos en sistemas multiagente, y *flocking* para mantener su formación, es un área que no se ha explorado completamente, mientras que diferentes autores como Spears, W. et al [33] mencionan la importancia del desarrollo de la física artificial debido a la elegancia en su implementación, ya que, con parámetros generalizados y pequeños sistemas de cómputo distribuido programados con algoritmos simples, se pueden lograr interacciones complejas, así como sucede naturalmente con las reglas físicas generales. A esto le denomina fisicomimética.

5. Desarrollo teórico del modelo

5.1 Modelado robótico

Para la implementación del sistema de validación del algoritmo es fundamental poder saber cómo controlar los agentes robóticos involucrados en estas pruebas, para ello se decidió ocupar un robot de tipo (2,0) o diferencial como base para la investigación. Esto debido a que se trata de una de las configuraciones de robot móvil más sencilla de analizar pero que, a final de cuentas, funciona para los fines del presente trabajo ya que se busca la validación del algoritmo propuesto sobre una base física y no tanto una validación del robot como tal.

Para poder realizar un análisis a profundidad, se verán las características básicas del sistema robótico propuesto para determinar su modelado cinemático, poniendo especial interés en obtener una función de la velocidad de las llantas con respecto a la velocidad angular y lineal del robot, las cuales serán utilizadas para traducir el modelo de campos potenciales al movimiento o la trayectoria.

5.1.1 Modelado de un robot diferencial

Un robot (2,0) se trata de un elemento, con cualquier forma geométrica, que cumple con la característica de tener dos llantas motorizadas, una en cada lado y una llanta no motorizada al frente que da estabilidad a la plataforma. Para realizar el modelo cinemático es importante considerar que las llantas se encuentran en un mismo eje y la velocidad total de la plataforma se restringe a la velocidad a la que giran las ruedas. Aunado a lo anterior se realizan las siguientes consideraciones [37]:

- El robot se mueve en una superficie plana y horizontal, por lo que la energía potencial gravitatoria es constante.
- Los ejes de guiado son perpendiculares al suelo.
- Todos los elementos del robot son rígidos, incluyendo las ruedas. No existen deformaciones dinámicas.
- Se considera que solo existe un punto de contacto entre la rueda y el suelo.
- No existe desplazamiento entre la rueda y el suelo.

Con dichas consideraciones se puede tomar el análisis realizado por el Dr. González, V. [38] para los robots móviles y considerar una postura del robot como se muestra en la siguiente figura.

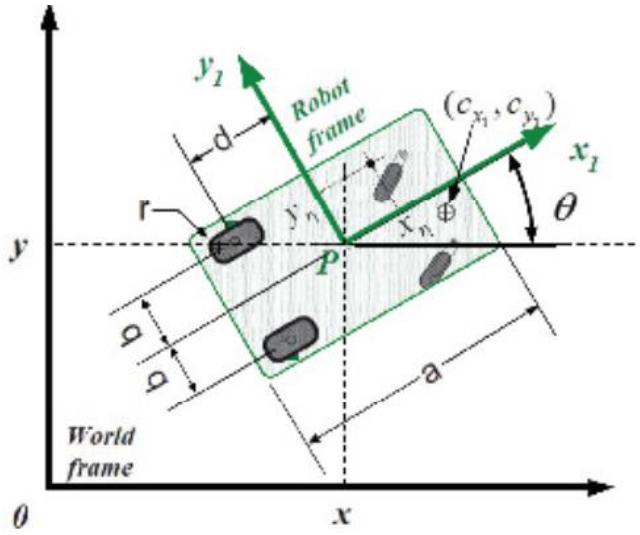


Figura 10. Esquema de la postura del robot móvil diferencial. [38]

En esta imagen es fácil apreciar que existen dos marcos de referencia que son de especial importancia para el estudio cinemático y el control de los movimientos del robot diferencial: por un lado, se encuentra el marco de referencia inercial o absoluto dentro del cual se está desenvolviendo el robot y, por otro lado, se encuentra un sistema de referencia montado sobre la plataforma del robot móvil. La descripción completa de la postura del robot se puede establecer mediante las siguientes variables:

$$q = [x \quad y \quad \theta \quad \emptyset wr \quad \emptyset wl]^T \qquad (4)$$

Donde *q* representa las coordenadas de P(x, y) del sistema de referencia {*x1, y1*} [39] y el ángulo que forma dicho sistema de referencia con respecto al sistema inercial absoluto, mientras que las últimas dos variables describen las posiciones angulares de las llantas tanto derecha (wr) como izquierda (wl). A su vez, para transformar el sistema de referencia inercial en el sistema de referencia que se encuentra montado en la plataforma del robot móvil, se tiene la siguiente matriz de transformación, que en sí misma se trata de una matriz de rotación en el eje Z, perpendicular al plano de trabajo del robot móvil.

$$R(\theta) = \begin{bmatrix} cos(\theta) & sen(\theta) & 0 \\ -sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5)$$

Considerando lo anterior, se ocupa la herramienta de propagación de velocidades para referir la velocidad del sistema de referencia montada en el robot a las dos ruedas diferenciales en los ejes motorizados, de tal manera que sea posible encontrar una relación entre las variables del sistema.

$$\omega_{r\_r} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \omega_{1_1} + \emptyset' wr * \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad (6)$$

$$v_{r\_r} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \left(v_{1_1} + \omega_{1_1} \times P_{1_{wr}}\right) \qquad (7)$$

Como se puede observar, la matriz de rotación asociada a estas ecuaciones es la matriz de identidad, esto debido a que el sistema de referencia colocado en el punto P, a partir del cual se realiza la propagación de velocidades, no se encuentra rotado con respecto al sistema colocado o propagado en la llanta del robot diferencial. Por otro lado, se tiene velocidad angular y velocidad lineal inicial representadas por ω_{1_1} y v_{1_1} los cuales representan las velocidades lineales (V) y angulares (omega) del sistema colocado en P que, por inspección, resulta ser la velocidad lineal de la plataforma con sus componentes en X, Y; y la velocidad angular de la misma plataforma (eje Z). A su vez, el vector P_{1_wr} representa el desplazamiento desde el punto P hasta el sistema colocado en la llanta, el cual como se puede observar en la figura anterior, se divide en dos componentes: -d en X y +b o -b respectivamente dependiendo de a cuál llanta se realiza la propagación en Y.

Dicho procedimiento debe ser realizado en ambas llantas para después definir las restricciones cinemáticas de cada una de ellas y, finalmente, obtener 6 ecuaciones, tres por cada llanta que representan matemáticamente el movimiento de cada una de ellas.

$$WR := \begin{pmatrix} x' \cdot \cos(\theta) + b \cdot \theta' + y' \cdot \sin(\theta) - r \cdot \varphi'_{fr} \\ y' \cdot \cos(\theta) - x' \cdot \sin(\theta) - d \cdot \theta' \\ 0 \end{pmatrix} \qquad WL := \begin{pmatrix} x' \cdot \cos(\theta) - b \cdot \theta' + y' \cdot \sin(\theta) - r \cdot \varphi'_{fl} \\ y' \cdot \cos(\theta) - x' \cdot \sin(\theta) - d \cdot \theta' \\ 0 \end{pmatrix} \quad (8)$$

Con esto es posible obtener la matriz A correspondiente al sistema del robot diferencial (2,0) que se muestra a continuación:

$$\begin{pmatrix} -\sin(\theta) & \cos(\theta) & -d & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & -d & 0 & 0 \\ \cos(\theta) & \sin(\theta) & b & -r & 0 \\ \cos(\theta) & \sin(\theta) & -b & 0 & -r \end{pmatrix} \quad (9)$$

Con esta matriz es posible darse cuenta de que sólo existen en ella tres ecuaciones que son linealmente independientes, contra 5 variables a obtener (vector q), por lo cual dos de las variables deberán ser dadas para resolver las demás.

En este caso, debido a que más adelante el método de control así las requerirá al estar definido por un vector de comportamiento para el robot móvil, se decidió que las dos variables dadas fueran la velocidad lineal de avance y la velocidad angular del robot móvil (velocidad y ángulo que forma el vector de comportamiento). Obteniendo, una vez resuelto el sistema de ecuaciones, la siguiente relación entre las variables independientes o de control (velocidad lineal de avance y angular) y el resto de las variables de velocidad del robot:

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \varphi'_{fr} \\ \varphi'_{fl} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -d\cdot\sin(\theta) \\ \sin(\theta) & d\cdot\cos(\theta) \\ 0 & 1 \\ \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{pmatrix} \cdot \begin{pmatrix} x' \\ \theta' \end{pmatrix} \quad (10)$$

Donde x' es la velocidad de avance y Θ' la velocidad angular. A partir de dicha relación es fácil observar que las velocidades angulares de los motores (dos últimas variables del vector de incógnitas) se encuentra expresadas en función de las variables independientes, lo cual será de gran utilidad al momento de controlar al robot. El análisis completo para la obtención de estas ecuaciones se incluye en el anexo 1, donde es posible encontrar las ecuaciones desglosadas y basadas en el análisis realizado por González, V. y también basado en lo desarrollado en la tesis de Dorantes, E. [37].

5.2 Definición del campo potencial artificial implementado

5.2.1 Campo potencial general

Basándose en las propuestas realizadas por Khatib [30] para definir el campo potencial artificial que domina los movimientos del agente hacia el objetivo evadiendo obstáculos, se decidió implementar conceptualmente una idea semejante. Tomando como principal variable de control la distancia del agente al objetivo es posible generar el campo de atracción hacia el mismo, de tal forma que el agente se vea atraído hacia el objetivo con una fuerza proporcional a la distancia que lo separa de él, esto es, entre más alejado se encuentre el agente del objetivo mayor será la fuerza de atracción y viceversa.

Una modificación a lo establecido en el artículo anterior es que el campo potencial artificial propuesto por dicho autor se toma como un campo de fuerza, mientras que para el caso de la presente propuesta se considera un campo potencial artificial de velocidades. Esto hace que el modelo utilizado sea congruente con el sistema implementado físicamente, ya que, como se explicará más a detalle posteriormente en las especificaciones de los materiales empleados, se utilizó un controlador para los motores que incluye un sistema automático de regulación de velocidad. De tal forma que la variable a controlar por el modelo diseñado será más directamente la velocidad en lugar de una fuerza que actúa sobre los agentes. Por lo anterior, a cada valor del campo potencial le corresponderá un respectivo vector de velocidad cuyas características principales serán la magnitud de la velocidad y ángulo de dicho vector. Estas características serán las que se tomen para su posterior simulación e implementación en el sistema real, además resultan congruentes con las entradas requeridas por el modelo cinemático de los agentes para su control.

Para la definición del campo potencial se considera un espacio de trabajo de *n x m* [m] con un sistema coordenado colocado cerca del centro del espacio, en el cual se definen coordenadas en pares ordenados tales que:

$$(x, y) \in R^2, \quad -n < x < n, \quad -m < y < m \qquad (11)$$

En dicho espacio de trabajo existe un punto denominado como objetivo, (x_{obj}, y_{obj}), éste genera una atracción sobre todo el espacio con una magnitud equivalente a la distancia euclidiana hacia él. Adicionalmente, se pueden considerar una cantidad *j* de obstáculos que generan repulsión modificando el campo de atracción producido por el objetivo. Es importante mencionar que, idílicamente, el punto objetivo se comporta como una partícula atractiva sin masa y sin volumen, mientras que los obstáculos se consideran elementos sólidos circulares con un cierto volumen a evadir.

A partir de la definición general anterior sobre el espacio de trabajo, se reconoce la presencia, en cada punto del espacio, de una velocidad repulsiva y una velocidad atractiva generada por los obstáculos y el objetivo respectivamente. Dichos vectores de velocidad actúan simultáneamente en el mismo espacio y, por lo tanto, es posible superponerlos entre sí y sumarlos para generar un vector resultante que actuará en cada punto del espacio. De esta manera, el vector correspondiente al campo de velocidades para un punto específico será equivalente a la suma vectorial del vector de velocidad del campo de atracción hacia el objetivo en dicho punto, más el vector de velocidad del campo de repulsión generado por cada uno de los obstáculos presentes en el espacio de trabajo.

$$V(x, y)_{total} = V(x, y)_{atractiva} + V(x, y)_{repulsiva\ total} \qquad (12)$$

Donde (x, y) corresponde a la coordenada a evaluar, para todo par ordenado localizado dentro del espacio de trabajo; $V(x, y)_{total}$ denota el vector de velocidad que actúa en el punto (x, y) del espacio; $V(x, y)_{atractiva}$ denota el vector de velocidad correspondiente a la atracción hacia el objetivo; y $V(x, y)_{repulsiva\ total}$ denota el vector de velocidad resultante de la suma de la repulsión de todos los obstáculos actuando en el espacio de trabajo.

A raíz de esta definición se identifican dos elementos clave para la generación del campo de velocidades que regulará el comportamiento de los agentes en el entorno. Por un lado, tenemos una velocidad atractiva y, por otro lado, una velocidad repulsiva.

La definición de velocidad atractiva y velocidad repulsiva se da siguiendo el siguiente estándar:

- *Repulsiva*: Aquella velocidad que es paralela y con dirección contraria al vector generado que va del punto (x, y) evaluado dentro del espacio de trabajo, al centro del objeto (objetivo u obstáculo) con respecto al que se mide la distancia.

- *Atractiva*: Aquella velocidad que es paralela y en la misma dirección al vector generado que va del punto (x, y) evaluado dentro del espacio de trabajo, al centro del objeto (objetivo u obstáculo) con respecto al que se mide la distancia.

5.2.2 Campo de atracción al objetivo

Una vez ubicadas estas dos componentes se procede a definir la función de la velocidad atractiva. Dicha función, al ser proporcional a la distancia entre el punto evaluado y el objetivo del agente, queda definida, de manera general, de la siguiente forma:

$$V(x,y)_{atractiva} = \mu * d_{obj} \qquad (13)$$

Donde μ representa una constante positiva de atracción y d simboliza la distancia euclidiana al objetivo definida de la siguiente manera:

$$d_{obj} = \sqrt{(x_{obj} - x)^2 + (y_{obj} - y)^2} \qquad (14)$$

Por lo anterior, la definición queda como se muestra a continuación:

$$V(x,y)_{atractiva} = \mu\sqrt{(x_{obj} - x)^2 + (y_{obj} - y)^2} \qquad (15)$$

Siendo las únicas variables las coordenadas (x, y) debido a que los demás parámetros quedan determinados por el sistema.

Es importante mencionar que, debido al tamaño del espacio de trabajo y a la manera en que está definida la fuerza atractiva, se podría dar el caso de que, al agente, colocado en algún punto del espacio, le corresponda una magnitud de velocidad que resulte demasiado elevada como para producir fallas en el sistema o incluso que sea mecánicamente imposible de alcanzar por los agentes empleados. Es por ello por lo que se optó por definir también un valor máximo para la velocidad, V_{atrMax}, que será la máxima velocidad presente en el sistema. Quedando la velocidad atractiva definida, finalmente, de la siguiente forma:

$$V(x,y)_{atractiva} = \begin{cases} \mu\sqrt{(x_{obj} - x)^2 + (y_{obj} - y)^2} & , \quad V(x,y)_{atractiva} < V_{atrMax} \\ V_{atrMax} & , \quad V(x,y)_{atractiva} \geq V_{atrMax} \end{cases} \qquad (16)$$

Cabe destacar que la constante de proporcionalidad, en este caso, también cumple la función de hacer la conversión de valores de distancia en valores de velocidad deseada en el punto (x, y).

5.2.3 Campo de repulsión de los obstáculos

Por otro lado, para definir la velocidad repulsiva provocada por los obstáculos presentes en el espacio de trabajo se considera que, ésta, será inversamente proporcional a la distancia entre el punto en el espacio evaluado y el borde del obstáculo más cercano al agente. Debido a que se consideran obstáculos sólidos circulares, como se mencionó en el planteamiento del problema, se debe tomar en cuenta que la distancia entre cada punto del espacio y el obstáculo será la distancia hasta su centro menos el radio del obstáculo, debido a la forma de éstos.

$$d_{obs,i} = \sqrt{\left(x_{obs,i} - x\right)^2 + \left(y_{obs,i} - y\right)^2} - r_{obs,i} \qquad (17)$$

$$donde\ i = 1, 2, \dots, j$$

En la ecuación anterior, se puede observar que la distancia al obstáculo estará definida por el punto a evaluar del espacio de trabajo, las coordenadas del obstáculo que se consideran estáticas durante el transcurso de la prueba y el radio del obstáculo. Como lo indica el subíndice en cada uno de los términos referentes al obstáculo, debe realizarse este cálculo para cada uno de los obstáculos presentes en el espacio de trabajo.

La intención de este proyecto es crear un sistema en el que los agentes puedan actuar de forma reactiva, es decir, que no requieran tener un conocimiento absoluto del ambiente que los rodea para actuar, sino que sean capaces de reaccionar a su entorno conforme lo van explorando. Con esto en mente, una vez determinada la distancia del punto del espacio a cada obstáculo respectivamente, se debe tomar en cuenta que, para considerar una acción reactiva por parte del agente dentro del ambiente, dicho agente no debería tener la capacidad de percibir un obstáculo en cualquier punto del espacio de trabajo, sino únicamente cuando éste se localice a una distancia finita definida del mismo. Por ello, es necesario definir una distancia denominada *radio de alcance del obstáculo* ($r_{alcance}$), en la cual, de estar dentro de dicho radio algún agente, la velocidad repulsiva se hará presente y actuará sobre ellos, de lo contrario, la fuerza repulsiva deberá ser cero como si el obstáculo no estuviera presente.

Considerando lo anterior es posible describir matemáticamente el comportamiento de dicha velocidad de repulsión. A su vez, se añade una normalización del vector de modo que exista un valor máximo de velocidad de repulsión para que, de ese modo, al igual que pasó con la velocidad de atracción, la velocidad exigida en un punto dado en la periferia del obstáculo no sea demasiado grande como para hacer fallar al sistema. De esta manera la velocidad de repulsión será igual a la velocidad de repulsión máxima cuando la distancia entre el agente y el obstáculo sea cero, y será igual a cero si la distancia es mayor o igual al radio de alcance del obstáculo.

$$V(x,y)_{repulsiva,i} = \begin{cases} \frac{r_{alcance} - d_{obs,i}}{r_{alcance}} * V_{repMax} & , \quad d_{obs} \leq r_{alcance} \\ 0 & , \quad d_{obs} > r_{alcance} \end{cases} \qquad (18)$$

La ecuación anterior regirá la velocidad de repulsión que un obstáculo *j* proyecta sobre una coordenada (x, y) dada dentro del espacio de trabajo. También es importante notar que, mientras la distancia entre el punto evaluado y el obstáculo se reduce, la velocidad resultante va en aumento hasta llegar a una distancia de cero, en cuyo caso la velocidad repulsiva se presentará como máxima.

El cálculo de la velocidad mencionada anteriormente es solo individual y referente exclusivamente al efecto que tiene un único obstáculo por cada uno de los puntos del espacio. Por lo tanto, para poder generar una velocidad repulsiva total es necesario sumar todos los vectores presentes en un mismo punto debido a los efectos del resto de los obstáculos para cada uno de los puntos del espacio como se muestra a continuación:

$$V(x,y)_{repulsiva\ \ total} = \sum_{i=1}^{j} V(x,y)_{repulsi\ ón,i} \qquad (19)$$

Donde, *j* representa la cantidad total de obstáculos presentes en el espacio de trabajo, como se había mencionado anteriormente.

5.2.4 Unión de campo atractivo y repulsivo

Una vez que se tiene el vector de velocidad resultante de atracción y repulsión total para un punto dado del espacio de trabajo (x, y), es posible determinar el efecto de dichos fenómenos sobre ese punto utilizando el principio de superposición de efectos presentado en la ecuación 12.

Utilizando dichas definiciones, es posible graficar, para cada punto dentro de un espacio determinado, una representación del campo vectorial de velocidades de modo que fuera posible discutir su viabilidad y futura aplicación en conjunto con el algoritmo de *flocking* propuesto.

En las figuras 11 y 12 se muestran los campos de velocidades producidos por el objetivo y un obstáculo respectivamente de forma independiente. Como resulta evidente en estas representaciones gráficas el campo generado por el objetivo tiene influencia en todo el espacio, mientras que el campo generado por el obstáculo únicamente afecta una sección del espacio que se encuentra dentro del área delimitada por el radio de alcance del obstáculo, que para este ejemplo de aplicación se estableció como $r_{alcance} = 1\ [m]$.

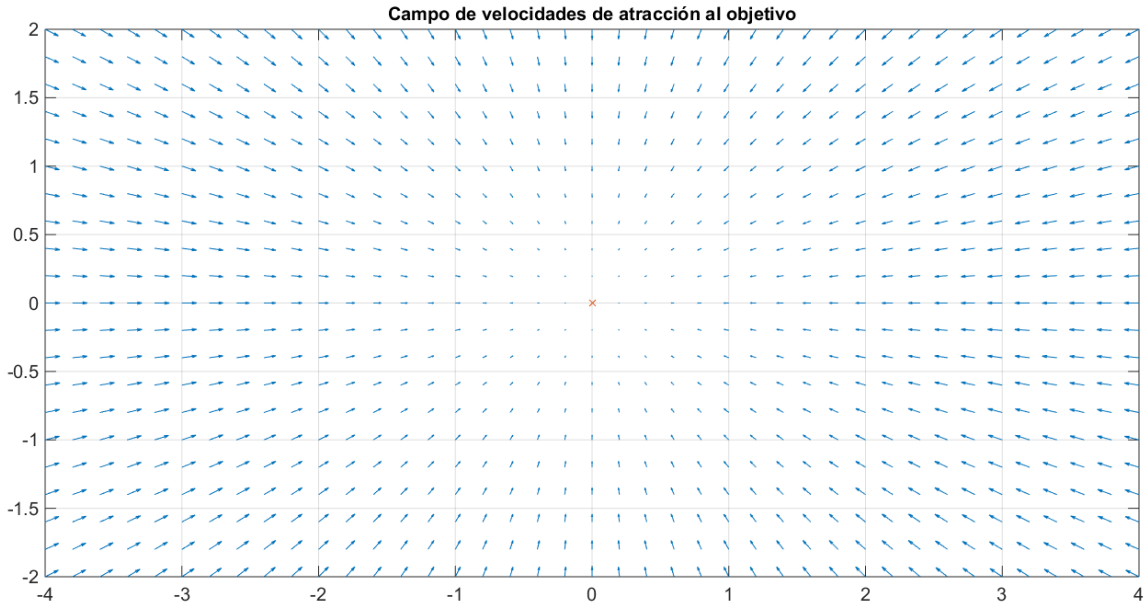


Figura 11. Campo de velocidades de atracción generadas por un objetivo. (Unidades en metros)

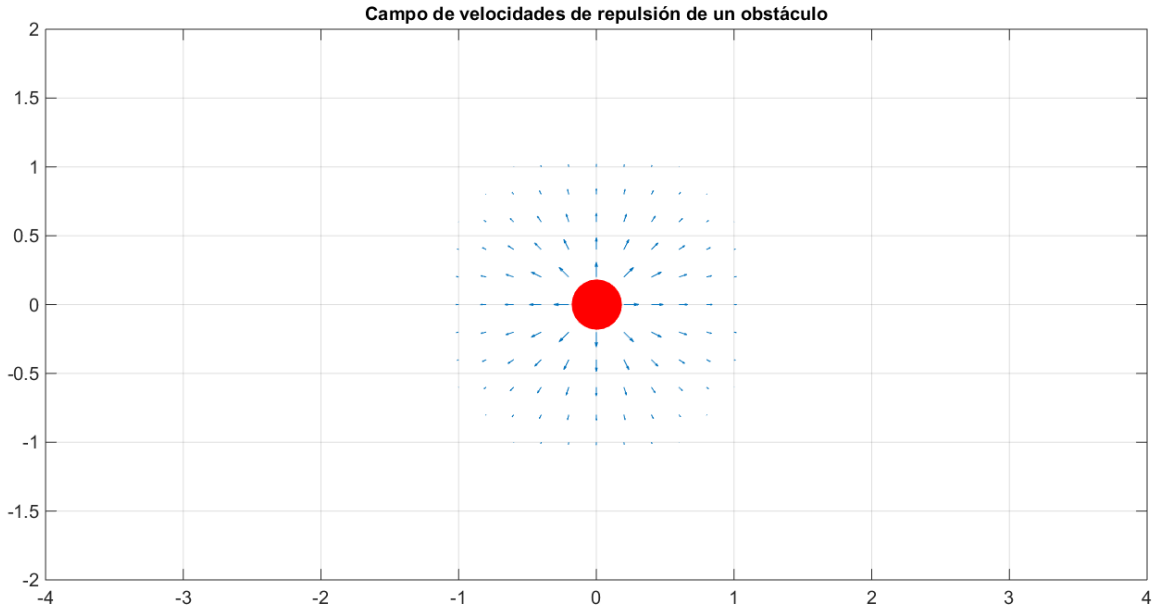


Figura 12. Campo de velocidades de repulsión generado por un obstáculo. (Unidades en metros)

Proponiendo un entorno en el que el objetivo se ubique en las coordenadas (3,0) y en el cual se localicen, además, dos obstáculos en (0,0) y (1.5,-0.2) respectivamente, tras realizar la superposición de los campos generados se obtiene la representación mostrada a continuación.

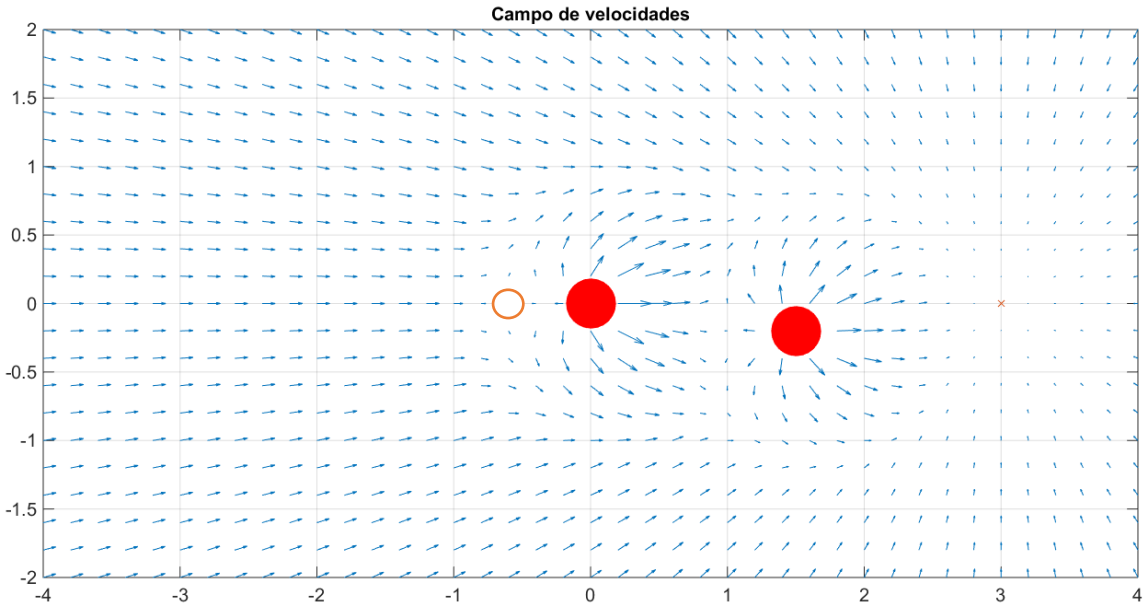


Figura 13. Campo de velocidades con un objetivo y dos obstáculos estáticos. Es posible observar un mínimo local claramente identificado (círculo naranja). (Unidades en metros)

En la figura 13 se observa el funcionamiento y aplicación de los campos potenciales generados para la atracción hacia un objetivo definido y la evasión de obstáculos mediante campos de repulsión. En estas gráficas se presentan, mediante flechas de color azul, las velocidades para diferentes puntos en el espacio de trabajo. La dirección de esta flecha establece, respectivamente, la dirección de la velocidad que regirá el comportamiento del agente en dicho punto, mientras que el tamaño de ésta representa, gráficamente, la magnitud de la velocidad. Este campo de velocidades representa ya la superposición de los campos de atracción y repulsión y pueden considerarse propios de cada punto del espacio, siendo independientes de la presencia de agentes.

A partir de la gráfica es posible observar que el campo de velocidades generado es congruente con el comportamiento que se pretende implementar en los agentes, ya que las flechas azules del campo se encuentran direccionadas al punto objetivo. El punto objetivo es representado, en este caso, por una pequeña cruz ubicada del lado derecho de la imagen cerca de las coordenadas (3, 0). Al mismo tiempo, para los puntos cercanos a las posiciones de los obstáculos, este campo se ve deformado por la acción del campo de repulsión, cuyo efecto se ve reflejado como una tendencia a evadir el obstáculo ajustando la trayectoria hacia el objetivo. También es significativo observar la identificación clara de un mínimo local, resaltado por un círculo naranja externo a la simulación, en el cual existe un punto de estancamiento en el campo potencial. Esto se debe a que en este punto se presenta un equilibrio particular entre las velocidades atractivas y repulsivas produciendo una velocidad del campo total de magnitud cero. A pesar de que éste no se localiza en un punto cerca del objetivo, por la naturaleza del funcionamiento del sistema, los agentes pueden tender a detenerse ahí sin llegar a cumplir su propósito de desplazarse hasta el punto objetivo.

Esta simulación gráfica da una idea clara del comportamiento que tendría un agente influenciado directamente por las velocidades de atracción y repulsión generadas por el objetivo y los obstáculos respectivamente. Aquí ya se toman en cuenta las consideraciones conceptuales para la generación del campo potencial definidas en esta sección. Al momento de implementar este campo de

velocidades sobre un agente el resultado será el desplazamiento de éste con la velocidad correspondiente a las coordenadas en las que se encuentre y, por lo tanto, tenderá a desplazarse en dirección al objetivo a la vez que se aleja y evade los obstáculos. Ahora bien, este comportamiento resulta exitoso para un único agente desplazándose en el espacio de trabajo, sin embargo, al momento de introducir un mayor número de agentes se presenta también la posibilidad de que haya colisiones entre ellos. Debido a esto se hace necesaria la implementación de un algoritmo que solucione las interacciones entre los diferentes agentes y que pueda ser sobrepuesto a la velocidad de atracción y repulsión establecidas previamente. Esto se pretende resolver mediante la implementación del algoritmo de *flocking* que se propone a continuación.

5.3 Integración del algoritmo de *flocking* al campo potencial

5.3.1 Campo de interacción entre agentes

En la sección previa se explicaron los campos de velocidades de atracción hacia el objetivo y repulsión de los obstáculos que regirán a grandes rasgos el comportamiento de un agente que se localice dentro del espacio. Sin embargo, hasta este punto no se ha considerado la situación en la que haya un mayor número de agentes actuando simultáneamente y, por lo tanto, interactuando entre sí en el mismo entorno. A diferencia de los campos generados por el objetivo o por obstáculos estáticos, que podrían considerarse fijos y constantes, la interacción entre agentes será dinámica ya que éstos, a su vez, van cambiando de posición y velocidad constantemente y de manera simultánea.

Para dotar al espacio de la capacidad de soportar varios agentes sin que estos choquen y, al mismo tiempo, tengan la habilidad para cohesionarse ordenadamente en una formación, es necesario generar un algoritmo que utilice la información local para determinar el comportamiento del agente. Para ello, se utilizará el modelo de Arkin de modo que se genere un vector resultante de la interacción entre agentes que se sume y se pondere utilizando una constante de peso para cada uno de los modelos, de tal manera que tanto el efecto del campo potencial generado en la sección anterior como el efecto de los agentes se refleje en el comportamiento del robot con una rapidez y dirección determinada.

La propuesta consiste en generar un campo de velocidades específico para representar la interacción entre los diferentes agentes localizados en el entorno y combinar los efectos de éste con los efectos del objetivo y obstáculos, de manera que en conjunto determinen el comportamiento de los agentes. Es decir, la superposición de los campos generados por los agentes con el campo explicado en la sección anterior regirá el comportamiento de éstos.

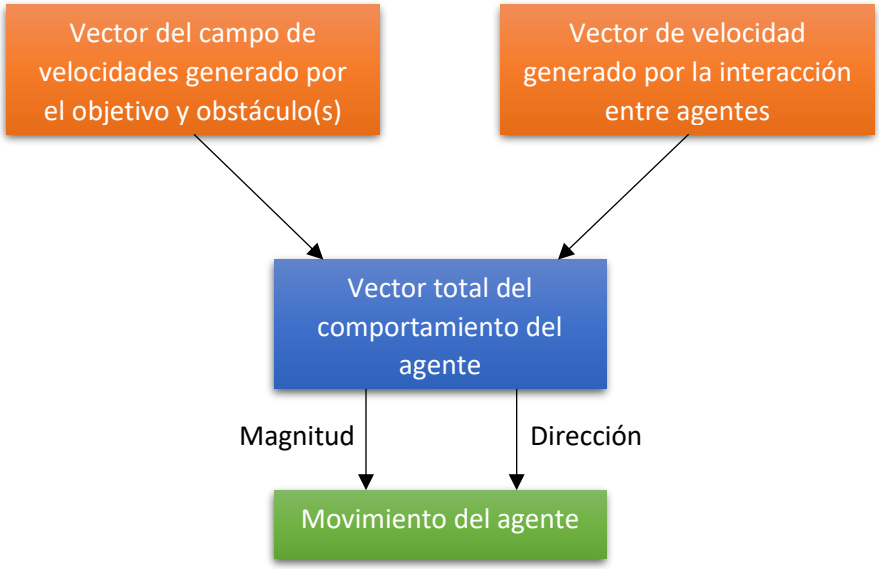


Figura 14. Diagrama conceptual del funcionamiento del sistema integrando la interacción entre agentes.

El vector de interacción entre agentes se calculará localmente, considerando únicamente aquellos otros agentes que se localicen a una distancia menor que el radio de detección del agente. De esta forma se pretende respetar el comportamiento reactivo del sistema, permitiendo su correcto funcionamiento aún sin el conocimiento absoluto de las posiciones del resto de los agentes y abriendo la posibilidad de un control descentralizado. Para regular la interacción entre agentes se hará uso del algoritmo de *flocking* analizado en la sección 4.2.4. Por lo que los rangos de interacción entre agentes se definen de forma similar a como los define Reynolds en [40] considerando tres zonas fundamentales: atracción, repulsión y orientación.

El problema para determinar la interacción entre agentes queda formulado de la siguiente manera. Se tiene un agente dentro del espacio de trabajo definido en *X* desde *-n* hasta *n* y en *Y* desde *-m* hasta *m*, tal y como se definió el espacio para el campo potencial artificial. El agente se encuentra ubicado en las coordenadas *(X, Y)* dentro del mismo espacio de trabajo y tiene la posibilidad de percibir su entorno y ubicar a aquellos agentes que se ubiquen a una distancia menor que un radio de interacción establecido. Alrededor del agente, dentro del radio de interacción, se encuentra una cantidad *t* de agentes a diferentes distancias de él, con diferentes posiciones y orientaciones. Debido a que esta interacción será un comportamiento reactivo y local para cada agente, no es posible generar un campo vectorial completo que incluya a todos los agentes en el espacio e implementarlo sobre todos ellos, sino que es necesario que cada agente interprete la propia interacción que generan los agentes a su alrededor sobre sí mismo. Es decir, cada agente debe identificar los agentes a su alrededor, y calcular la interacción generada por cada uno de ellos con base en la distancia a la cual se localizan.

En la figura 15 se presentan las zonas de interacción entre agentes con base en lo propuesto por el algoritmo de *flocking*.

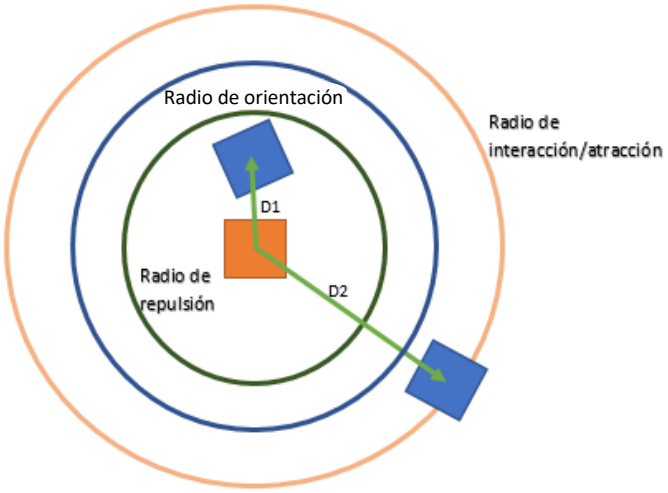


Figura 15. Se muestra el agente de interés (o local) en color naranja, agentes dentro del área de interacción (azules) y los tres radios que delimitan áreas de interacción.

Para la definición de las diferentes zonas de interacción entre agentes se hace uso de tres nuevas variables que funcionarán como radios límites. Un primer radio denominado *radio de repulsión*, que será el menor de los tres y, en consecuencia, el de menor distancia al agente local, definirá una zona identificada como zona de repulsión. Un segundo radio de mayor magnitud que el primero denominado radio de orientación definirá la zona de orientación. Y finalmente, el mayor de los tres, llamado radio de atracción (o también identificado anteriormente como el radio de interacción) definirá la zona de atracción, además de delimitar también el alcance de percepción del robot a su entorno.

En las diferentes áreas que delimitan estos radios se tendrán diferentes comportamientos:

1. En el área de atracción delimitada entre el radio de orientación y el radio de atracción se tendrá una velocidad atractiva proporcional a la distancia del agente local y el agente que se encuentra en dicha región.
2. En el área de orientación, entre el radio de repulsión y el radio de orientación, se presentará únicamente una velocidad angular con tendencia a orientar al agente local en la misma dirección que los agentes externos que se ubican en dicha zona.
3. En el área de repulsión que se encuentra dentro del radio de repulsión se tendrá un comportamiento parecido al obstáculo entre el agente de estudio y el agente en dicha área, presentándose una velocidad repulsiva inversamente proporcional a la distancia entre los agentes involucrados.

Cabe resaltar, como se mencionó anteriormente, que cada agente se encarga de calcular localmente los efectos que generan los agentes ubicados en su entorno. De esta manera no es necesaria la transferencia de información entre agentes, sino que cada uno es capaz de realizar su propio procesamiento independiente del resto de los miembros del enjambre. Además, esto permite que cada uno de los agentes sea programado de la misma manera individualmente pero que, al mismo tiempo, sean estas interacciones entre ellos las que produzcan un comportamiento global de enjambre.

5.3.2 Efecto de atracción y repulsión entre agentes

Una vez entendido esto se despliegan las ecuaciones que regirán la formación de un campo de velocidades alrededor del agente, dependiente de la distancia relativa a la que se encuentre el agente local respecto al resto. Para, finalmente, obtener un vector resultante de la suma de todas las interacciones con los *t* agentes presentes en el radio de interacción. En este caso es más sencillo definir por partes la magnitud y el ángulo resultante del vector para cada agente.

Por simplicidad se generarán de forma independiente las velocidades angulares de la zona de orientación, por lo que en esta primera representación se calculan únicamente las velocidades en las zonas de atracción y repulsión, haciéndose cero la magnitud en el lugar correspondiente a la velocidad angular de orientación.

$$\left|Va\left(d_{i,a}\right)_i\right| = \begin{cases} \frac{R_{rep}-d_{i,a}}{R_{rep}} * Va_{max,rep} & , \quad d_{i,a} \leq R_{rep} \\ 0 & , \quad R_{rep} < d_{i,a} < R_{ori} \\ \frac{d_{i,a}}{R_{atr}} * Va_{max,atr} & , \quad R_{ori} \leq d_{i,a} \leq R_{atr} \\ 0 & , \quad d_{i,a} \geq R_{atr} \end{cases} \qquad (20)$$

Esta ecuación describe la magnitud de la velocidad del *i-ésimo* agente, mientras que se puede apreciar la definición de los tres radios:

- R_{rep}: Radio de repulsión
- R_{ori}: Radio de orientación
- R_{atr}: Radio de atracción o radio de interacción

Todos estos valores son predefinidos, positivos e invariables a lo largo del proceso. Otros parámetros que se encuentran predefinidos serán:

- $Va_{max,rep}$*:* Velocidad máxima de repulsión entre agentes
- $Va_{max,atr}$*:* Velocidad máxima de atracción entre agentes

Haciendo especial énfasis que $d_{i,a}$ es la distancia entre el agente local y el agente externo en cuestión que se calcula de la siguiente forma:

$$d_{i,a} = \sqrt{(x - x_i)^2 + (y - y_i)^2} \qquad (21)$$

Donde (x_i, y_i) serán las coordenadas del i-ésimo agente dentro del radio de interacción desde i = 1 hasta *t* agentes.

Con el objetivo de evitar la no linealidad en la implementación de las interacciones y evitar cambios bruscos de comportamiento entre los agentes se realizó un ajuste de tal forma que los valores mínimos en las diferentes zonas corresponden a una magnitud de cero en cada una de las fronteras. Considerando esto, para la zona de repulsión se tiene una magnitud igual a $Va_{max,rep}$ cuando la distancia entre los agentes es nula y ésta va disminuyendo hasta llegar a cero en la frontera correspondiente a una distancia igual al radio de repulsión. Para el caso de la zona de atracción se presenta el fenómeno opuesto, la magnitud máxima ($Va_{max,atr}$) se presentará a la mayor distancia entre agentes, es decir, a una distancia igual al radio de atracción, y ésta disminuirá hasta llegar a una magnitud de cero en la frontera correspondiente al radio de orientación. Aunque aún no se incluye el efecto de la zona de orientación este tendrá un comportamiento similar, con la particularidad de que en ambas fronteras (radio de atracción y radio de orientación) la magnitud de la velocidad será cero y el valor máximo se ubicará en el punto medio entre estas dos distancias.

Para implementar esto se modifica la ecuación de la interacción entre agentes de la siguiente manera:

$$\left|Va\left(d_{i,a}\right)_i\right| = \begin{cases} \frac{R_{rep}-d_{i,a}}{R_{rep}} * Va_{max,rep} & , \quad d_{i,a} \leq R_{rep} \\ 0 & , \quad R_{rep} < d_{i,a} < R_{ori} \\ \frac{d_{i,a}-R_{ori}}{R_{atr}-R_{ori}} * Va_{max,atr} & , \quad R_{ori} \leq d_{i,a} \leq R_{atr} \\ 0 & , \quad d_{i,a} \geq R_{atr} \end{cases} \qquad (22)$$

Una vez definida la magnitud de la velocidad, la siguiente parte es determinar la dirección que tendrá dicha velocidad de modo que pueda ser representada de forma vectorial e interpretada por todo el sistema, así como también, ser operada como elemento vectorial, lo cual será de especial importancia cuando se reúnan los diferentes esquemas de comportamiento al final de este proceso.

La dirección de la fuerza estará íntimamente relacionada, al igual que la magnitud, con la región en la que se encuentra el agente invasor del área de influencia, dado que, dependiendo de su localización relativa a estas regiones, la dirección podrá ser del agente invasor al agente local, en caso de ser una velocidad atractiva, o al contrario en caso de encontrarse en la región repulsiva. Para el caso de la zona de orientación, debido a que no se genera ninguna velocidad lineal que influya al agente, la dirección es irrelevante y se establece como cero para esta aplicación en particular. Considerando lo anterior se plantea la siguiente ecuación que describe la dirección del vector de velocidad.

$$\angle Va\left(d_{i,a}\right)_i = \begin{cases} \arctan\left(\frac{y-y_i}{x-x_i}\right) & , \quad d_{i,a} \leq R_{rep} \\ 0 & , \quad R_{rep} < d_{i,a} < R_{ori} \\ \arctan\left(\frac{y_i-y}{x_i-x}\right) & , \quad R_{ori} \leq d_{i,a} \leq R_{atr} \\ 0 & , \quad d_{i,a} \geq R_{atr} \end{cases} \qquad (23)$$

Con esta información es posible obtener las componentes de las diferentes velocidades correspondientes a la influencia que ejercen cada uno de los agentes circundantes al agente de estudio y así realizar la suma vectorial para obtener el vector total de interacción entre agentes.

$$V_{interacci\ ón} = \sum_{i=1}^{t} Va(d_{i,a})_i \qquad (24)$$

Tomando como referencia las velocidades determinadas anteriormente, es posible generar un campo de velocidades que corresponde a la influencia que tendría un agente externo sobre un agente local ubicados a $d_{i,a}$ de distancia entre ellos. Como primera iteración para analizar el funcionamiento de este modelo se consideró a los agentes como partículas, por lo que no es necesaria aún la implementación de la zona de orientación ya que, al ser considerados como partículas, los agentes aún no tienen una dirección determinada. El resultado de esta simulación, considerando radios de repulsión, orientación y atracción como 0.3, 0.5 y 1 [m] respectivamente, se presenta en la figura 16.

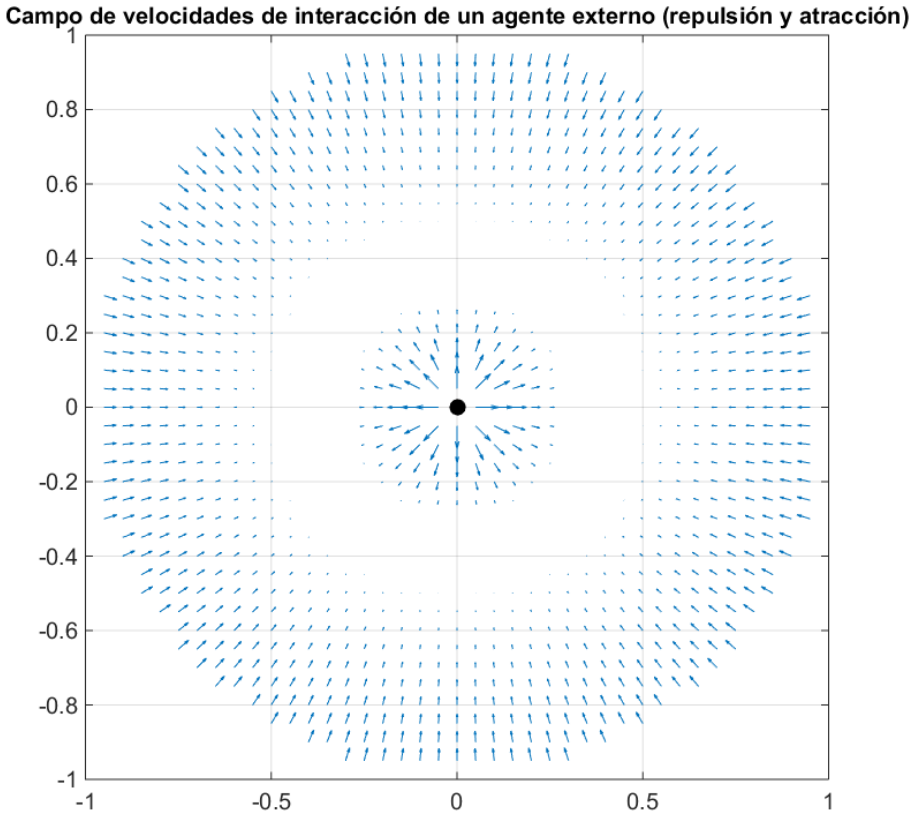


Figura 16. Campo vectorial generado desde un agente externo considerado como una partícula. (Unidades en metros)

En esta imagen se pueden observar los diferentes radios de influencia, de atracción, orientación y repulsión, existentes en la definición del campo vectorial, lo cual, corresponde con la implementación de las ecuaciones presentadas anteriormente. El modelo propuesto soporta, además, la localidad del algoritmo al permitir a cada agente hacer su procesamiento correspondiente con base en la información local que posea del sistema y del entorno que lo rodea.

Así mismo, se rescata el principio de escalabilidad y robustez, debido a que no es necesario definir la cantidad total de agentes interactuando en el espacio, sino que basta con definir aquellos agentes que se encuentran dentro de algún radio de influencia, permitiendo el incremento o decremento del número de agentes manteniendo la funcionalidad del sistema implementando los mismos principios. De igual manera se tiene un sistema flexible debido a que con dicha definición del modelo el sistema tiene la capacidad de adaptarse y actuar en reacción al entorno cambiante que se pueda presentar, por ejemplo, en el caso de la interacción con los demás agentes presentes en el medio y que cambian constantemente sus condiciones.

Otro punto importante para rescatar de este modelo es la creación natural de formaciones por parte de los agentes que se encuentran interactuando en un mismo entorno. Debido al modelo del campo de velocidades de interacción entre agentes, es de suponerse que los agentes que se encuentran más alejados entre sí tenderán a acercarse, mientras que aquellos que se encuentran demasiado cerca se alejarán unos de otros. Esto lleva a los agentes a aproximarse a un estado en el que las velocidades repulsivas y atractivas entre ellos se anulen y terminen creando una formación específica. Es de esperarse que las características de estas formaciones dependan de los valores establecidos para los radios de interacción, así como también, en cierta medida, del número de agentes que se encuentran interactuando en el entorno.

Para validar estas aseveraciones se implementaron una serie de simulaciones donde es posible demostrar cómo los agentes, preprogramados con el algoritmo dominado por las ecuaciones anteriores, se organizan sin colisionar y sin perder la cohesión entre ellos. Para la realización de estas pruebas de formación entre agentes todavía no se hace el acoplamiento del campo vectorial generado por el objetivo y obstáculos con el efecto de la interacción entre agentes, sino que únicamente se consideran las velocidades de interacción entre agentes. En estas simulaciones se consideraron distinto número de agentes ubicado en diferentes posiciones dentro del campo de interacción unos de otros, de tal forma que al programar en cada uno el comportamiento de interacción mostrado anteriormente se desplazarán hasta llegar a una formación estable.

Las figuras 17, 18 y 19 presentan algunas de las pruebas realizadas con formaciones de agentes como partículas.

Con 3 agentes:

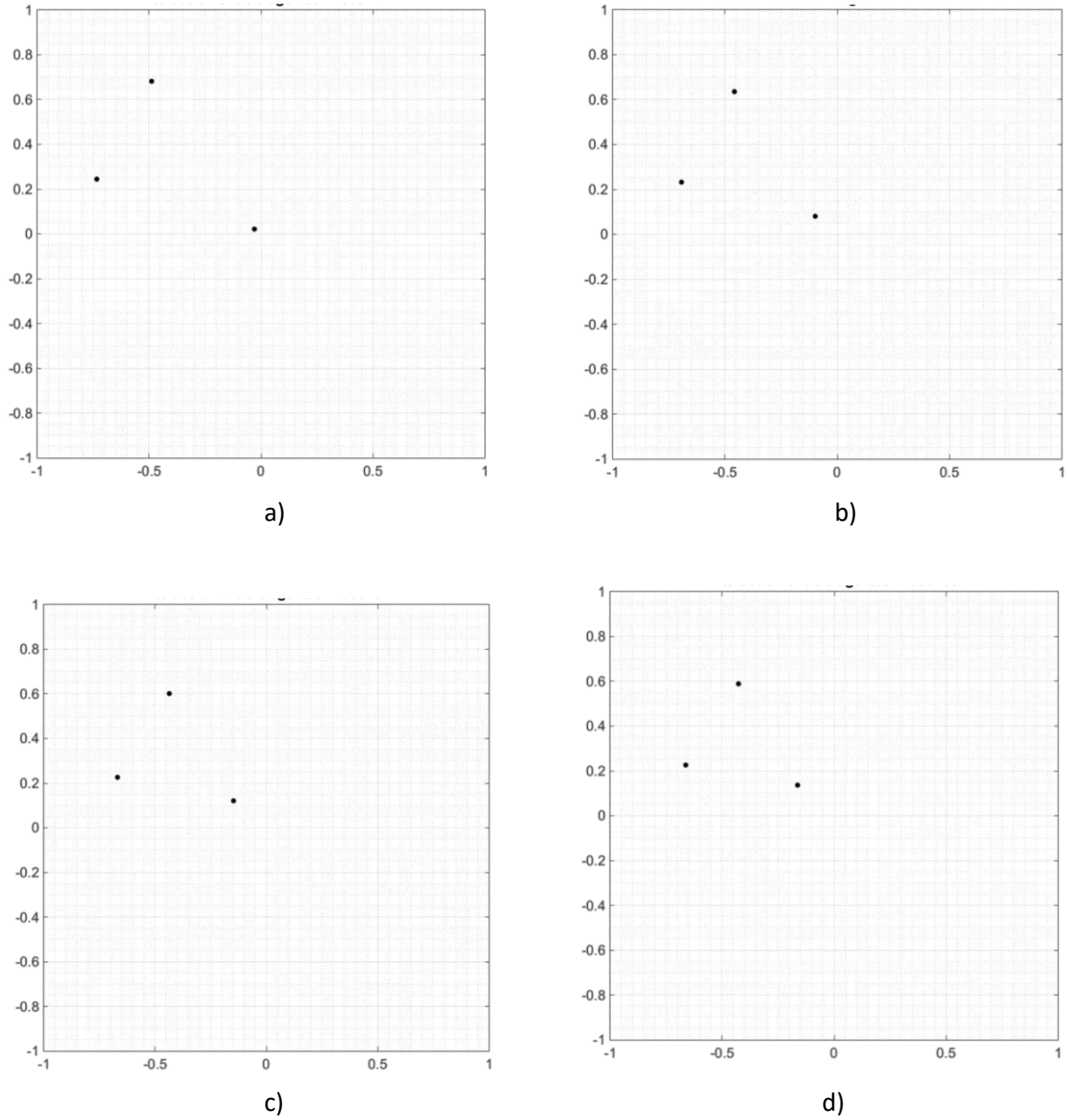


Figura 17. Formación generada de la interacción de 3 agentes considerados como partículas. Se muestran los siguientes pasos de simulación: a) paso 4, b) paso 15, c) paso 34, d) paso 53. (Unidades en metros)

Con 7 agentes:

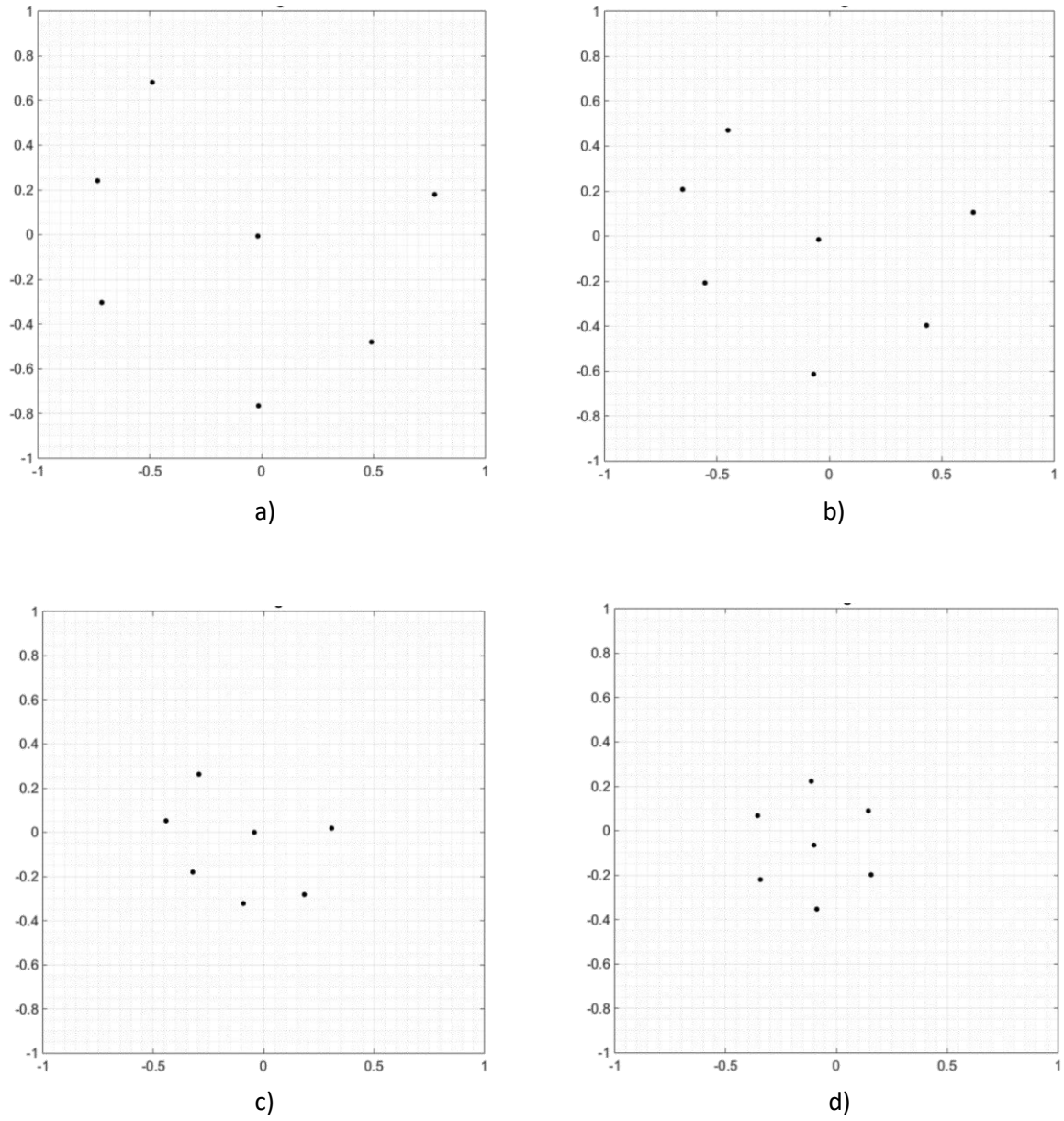


Figura 18. Formación generada de la interacción de 7 agentes considerados como partículas. Se muestran los siguientes pasos de simulación: a) paso 4, b) paso 26, c) paso 52, d) paso 182. (Unidades en metros)

Con 10 agentes:

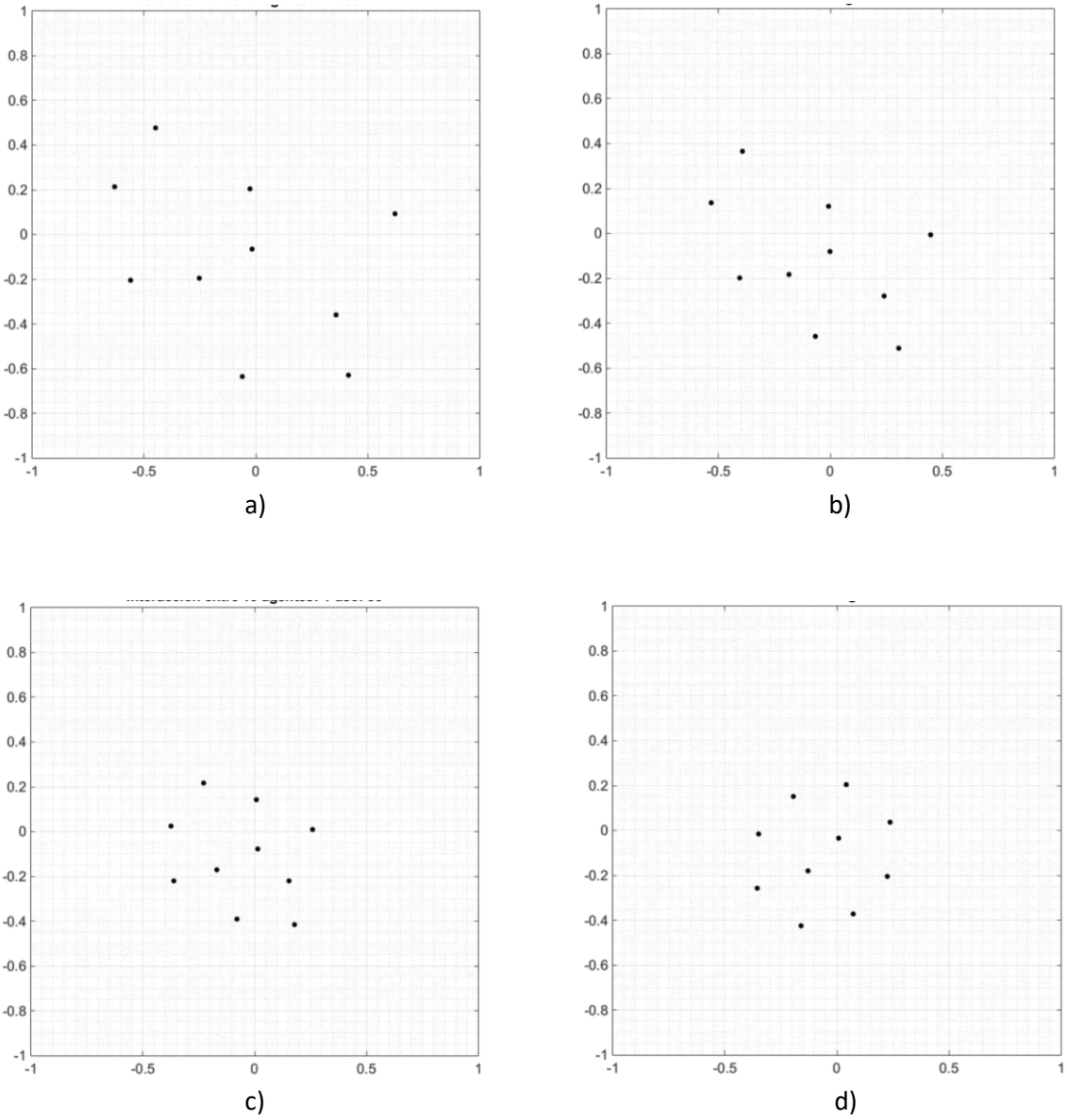


Figura 19. Formación generada de la interacción de 10 agentes considerados como partículas. Se muestran los siguientes pasos de simulación: a) paso 14, b) paso 24, c) paso 36, d) paso 168. (unidades en metros)

Con las simulaciones presentadas anteriormente se pueden observar patrones geométricos casi simétricos en las formaciones generadas entre los agentes, lo cual es de esperarse debido a que cada uno de ellos está programado para reaccionar de igual manera a su entorno y ante el resto de los agentes que lo rodean. Se aprecia también que el algoritmo mantiene la cohesión entre los integrantes del espacio de trabajo, mientras que, al mismo tiempo, los mantiene a una distancia segura. Es interesante y de especial relevancia destacar que la distancia de seguridad entre los agentes no es un parámetro de control en el sistema establecido directamente en el programa, por lo cual, la distancia resultante entre ellos es derivada de la interacción de los campos locales de velocidad de cada uno de los agentes que están interactuando. En las pruebas realizadas se considera el promedio de todas las distancias existentes entre los agentes debido a que se presentan variaciones dependiendo de cuáles agentes se tomen para realizar la medición. A esta distancia entre los agentes, se le denominará *"distancia de equilibrio"* debido a que, en la formación final en la que se mide, la suma de fuerzas de repulsión y atracción entre los agentes alcanza un estado de estabilidad entre ellos de modo que no se provoca ningún movimiento. Además, el tiempo que tardan los agentes en llegar a ese estado, como se menciona en diferentes artículos, se le conocerá como *tiempo de estabilización*.

Analizando las distancias de equilibrio en las pruebas realizadas con formaciones de distinto número de agentes se obtuvieron los resultados mostrados en la Tabla 1.

Tabla 1. Comparación de la distancia de equilibrio de formaciones con diferente número de agentes.

Número de agentes	Distancia promedio final entre agentes [m]	Distancia mínima entre agentes [m]
2	0.5000	0.5000
3	0.4739	0.4249
4	0.4059	0.2999
5	0.3705	0.2781
6	0.3811	0.2794
7	0.3900	0.2885
8	0.3965	0.2558
9	0.3980	0.2079
10	0.4001	0.2036

En esta tabla se presentan dos datos relacionados con la distancia de equilibrio, como primer dato se presenta la distancia promedio, calculada como el promedio de todas las distancias entre agentes al llegar a la posición de equilibrio de la formación; como segundo dato, la distancia mínima entre agentes, esto corresponde con la menor de todas las distancias entre agentes. Este otro dato es relevante también ya que se debe mantener siempre una distancia mayor a cero entre los agentes para evitar colisiones en el sistema.

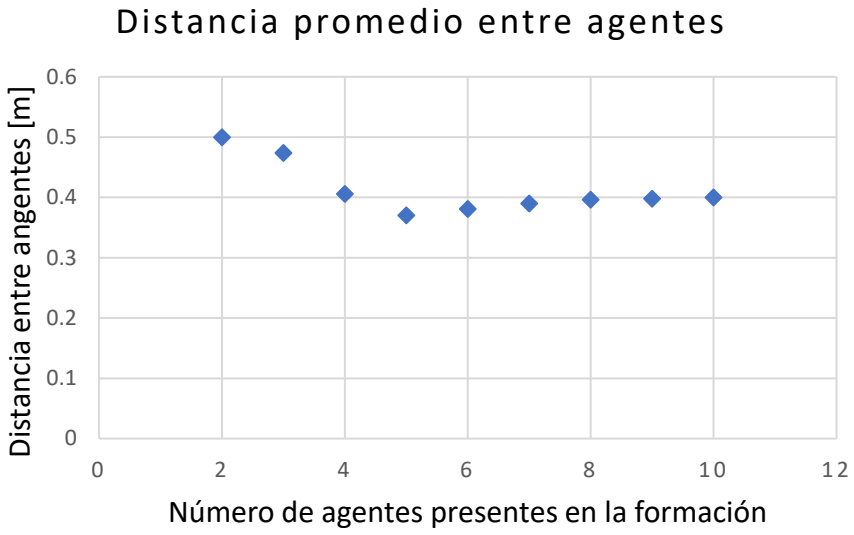


Figura 20. Distancia promedio entre agentes en formaciones.

Aunque para un número bajo de agentes la distancia promedio entre ellos es mayor, se observa que al incrementarse la cantidad de agentes interactuando en la formación, esta distancia promedio no sufre variaciones al incrementar su tamaño (cantidad de agentes), tendiendo una distancia promedio de 0.4 [m] para las condiciones establecidas en estas pruebas (definidas en correspondencia con el campo de interacción de la figura 16). Analizando el coeficiente de variación de los datos se observa que, considerando formaciones a partir de 6 agentes, tiene un valor de 1.96%. Ya que la variación es muy baja, se puede considerar que a partir de este punto la distancia promedio entre agentes se mantiene constante.

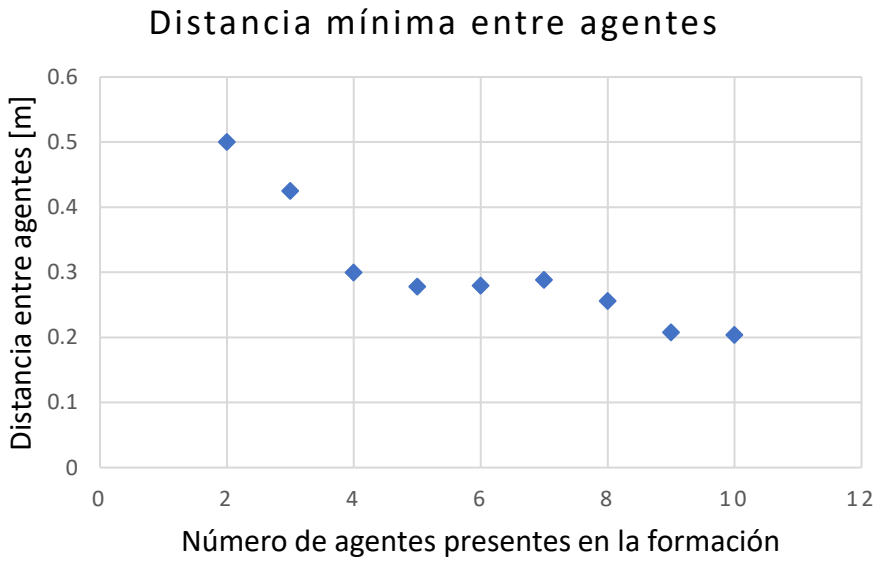


Figura 21. Distancia mínima entre agentes en formaciones.

Por otro lado, analizando la distancia mínima entre agentes se aprecia una disminución de ésta conforme se incrementa el número de agentes. Esto se debe a que, mientras el efecto repulsivo entre agentes actúa a distancias cortas (definidas por el radio de repulsión) y, por ende, exclusivamente entre los agentes más próximos entre sí, el efecto de atracción presenta un mayor alcance y, por tanto, afecta a un mayor número de los agentes presentes. Esta distancia mínima debe tenerse en cuenta, debido a que es la que definirá el número de agentes que soportará la formación sin que se presente colisión alguna entre ellos.

5.3.3 Efecto de orientación entre agentes

Finalmente, para completar el modelo de *flocking* hace falta incluir el efecto de la interacción entre agentes correspondiente a la zona de orientación. Si bien este no produce efecto alguno al considerar los agentes como partículas, sí tendrá efecto al momento de implementarse sobre agentes reales con dimensiones y orientación definidas.

Para implementar la parte del algoritmo correspondiente a la orientación se requiere tener como referencia la orientación que presentan los agentes, ya que, en principio, se pretende que éstos tiendan a orientarse hacia una misma dirección. De igual manera como se hizo para la atracción y repulsión entre agentes, el efecto de orientación únicamente se considerará con aquellos agentes externos que se localicen dentro de la zona correspondiente, para este caso entre el radio de repulsión y el radio de orientación.

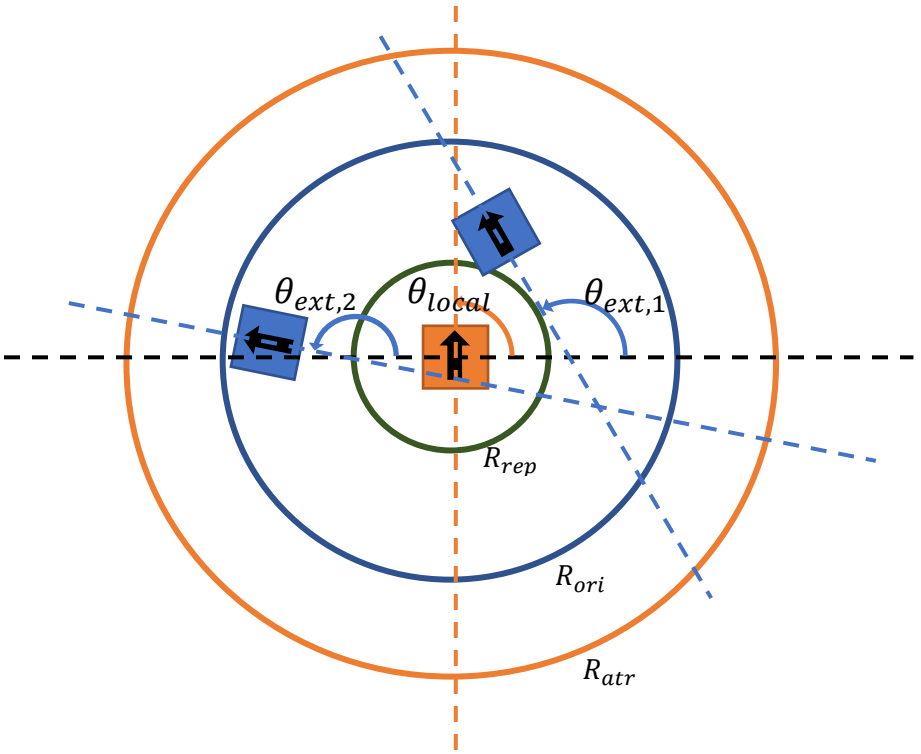


Figura 22. Agente local con dos agentes externos localizados en la zona de orientación.

Para determinar la magnitud del efecto de orientación se tomará como referencia la diferencia entre la orientación del agente local y el promedio de las orientaciones de aquellos agentes que se localicen dentro de la zona de orientación. En primera instancia es necesario conocer la orientación de los agentes, la cual se considerará como el ángulo formado entre una referencia horizontal paralela al eje x (0°) y la dirección que lleva cada uno de los agentes. En la figura 22 se muestra la orientación del agente local (θ_{local}) y la orientación de dos agentes externos ($\theta_{ext,1}$, $\theta_{ext,2}$). La orientación de 0° se estableció como referencia, sin embargo, el efecto de orientación dependerá exclusivamente de la diferencia entre las orientaciones de los agentes, por lo que sería posible determinarlo también a partir de orientaciones relativas con referencia al agente local en cuestión.

Se deberá calcular primero el valor promedio de las orientaciones de los agentes externos ubicados en la zona de orientación.

$$\theta_{ext,prom} = \frac{1}{n}\sum_{i=1}^{n} \theta_{ext,i} \qquad (25)$$

Donde n corresponde al número de agentes externos localizados dentro de la zona de orientación.

Como se pretende que el efecto de orientación entre agentes produzca exclusivamente un cambio en la orientación de los agentes, sin implicar un desplazamiento, se considera que produce únicamente una velocidad angular en lugar de una velocidad lineal como en el caso de los campos generados anteriormente. Esta velocidad angular se generará exclusivamente en la zona de orientación, por lo que también puede quedar definida matemáticamente en forma de una ecuación por partes. La diferencia de orientación ($\delta_{i,a}$) queda definida como se muestra en la ecuación 26.

$$\delta_{i,a} = \theta_{ext,prom} - \theta_{local} \qquad (26)$$

Considerando que se genera una velocidad angular que es proporcional a la diferencia de orientación se llega a la siguiente expresión matemática.

$$\omega a\left(\delta_{i,a}\right)_i = \begin{cases} 0 & , \quad d_{i,a} \leq R_{rep} \\ \gamma * \delta_{i,a} & , \quad R_{rep} < d_{i,a} < R_{ali} \\ 0 & , \quad d_{i,a} \geq R_{ali} \end{cases} \qquad (27)$$

De forma similar a como se ajustaron las velocidades repulsiva y atractiva para evitar las no linealidades en el comportamiento de interacción entre agentes en las fronteras de las distintas zonas, en este caso se realizará un ajuste similar de tal forma que, en las fronteras, correspondientes al radio de repulsión y al radio de orientación, la magnitud de la velocidad angular generada por el modelo sea cero. Además, se presentará el valor máximo de la velocidad angular en el punto medio entre estas dos fronteras.

$$\omega a(\delta_{i,a})_i = \begin{cases} 0 & , \quad d_{i,a} \leq R_{rep} \\ \frac{\frac{1}{2}(R_{ali}-R_{rep})-abs(d_{i,a}-\frac{R_{rep}+R_{ali}}{2})}{\frac{1}{2}(R_{ali}-R_{rep})} * \gamma * \delta_{i,a} & , R_{rep} < d_{i,a} < R_{ori} \\ 0 & , \quad d_{i,a} \geq R_{ori} \end{cases} \qquad (28)$$

Con esta velocidad angular de orientación queda definido el modelo de *flocking* que regirá el comportamiento de enjambre entre los agentes. Este último efecto de orientación se tratará independiente del efecto atractivo y repulsivo debido a que no se trata de una velocidad lineal, sino de una velocidad angular y, en consecuencia, también debe ser considerada de forma distinta al introducirse al modelo cinemático de los agentes.

5.3.4 Superposición del algoritmo de *flocking* y el campo potencial artificial

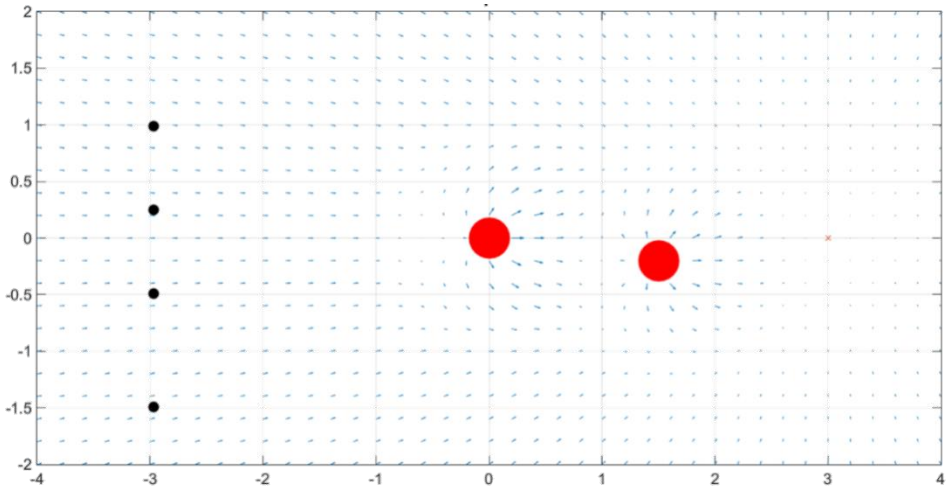
Una vez descritos los modelos que se implementarán para generar el sistema y lograr un comportamiento eficiente entre los agentes, es necesario unirlos adecuadamente y probar su funcionamiento en conjunto para comprobar que se siguen cumpliendo los objetivos resueltos por cada modelo de manera independiente.

Como se mencionó anteriormente, cada uno de los métodos descritos en la sección anterior resuelve parcialmente un mismo problema. Por un lado, el campo potencial artificial resuelve el problema de interacción del agente con el entorno del espacio de trabajo, describiendo su capacidad para alcanzar un objetivo dado; mientras que, por otro lado, el método de *flocking* ayuda a los agentes a mantener distancias seguras y una formación flexible gracias a la información localizada y la interacción entre ellos. Debido a que la salida de ambos sistemas, tanto el de campos potenciales artificiales como del sistema de *flocking*, se trata de vectores de velocidad, es posible superponer ambos efectos mediante la suma de los vectores finales de cada subproceso para así obtener un vector que dirigirá el comportamiento del agente. De esta manera, siempre que el agente tenga la información requerida por los modelos, podrá calcular y determinar cuál es el mejor comportamiento para lograr desplazarse al punto objetivo, a la vez que reacciona a su entorno y al comportamiento de los agentes que se ubican alrededor de él.

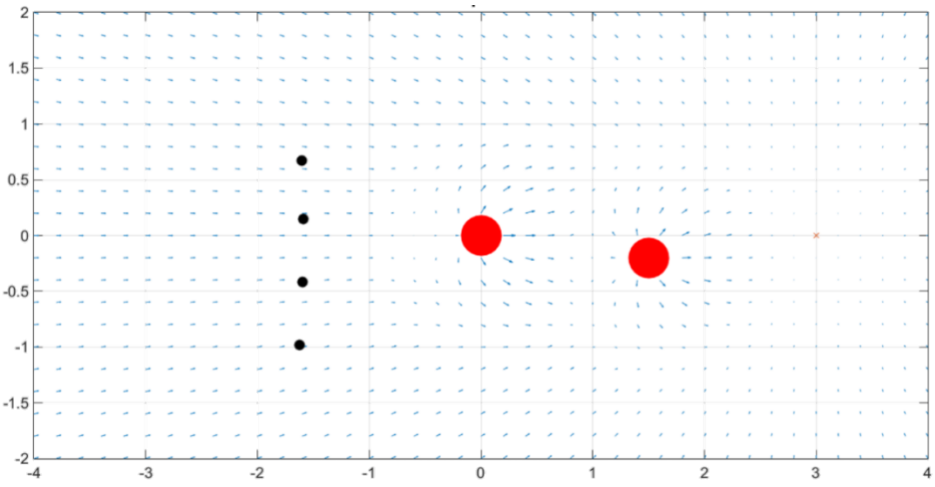
$$V_t = \rho * V(x,y)_{total} + \sigma * V_{interacción} \qquad (29)$$

La velocidad de salida resultante V_t será una función tanto de la localización del agente dentro del espacio como de la posición relativa de éste con respecto al resto de los agentes en el mismo espacio. Siendo ρ y σ constantes de peso del efecto del campo potencial y del efecto de la interacción entre agentes respectivamente; dichas constantes ayudan a controlar el peso relativo entre estas dos variables a lo largo de la implementación del algoritmo y deben ser valores numéricos reales y positivos.

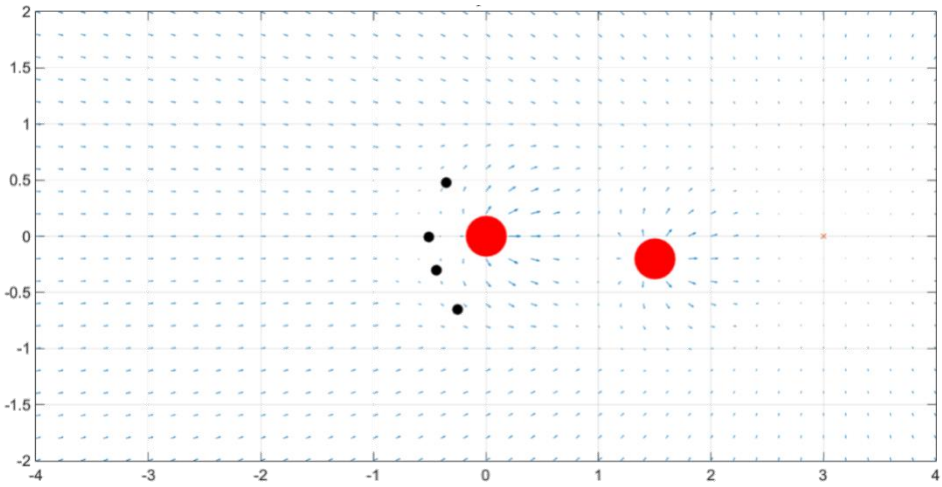
Una vez delimitados dichos parámetros fue posible diseñar una simulación mediante la cual se valide la aplicación de los dos métodos mencionados anteriormente conjugados en la suma de los vectores resultantes de cada uno de ellos.



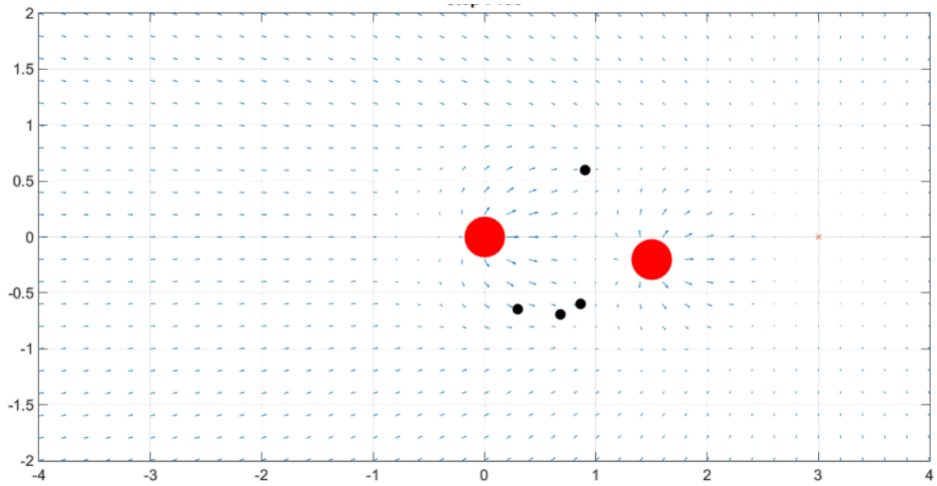
a) Paso de simulación: 0



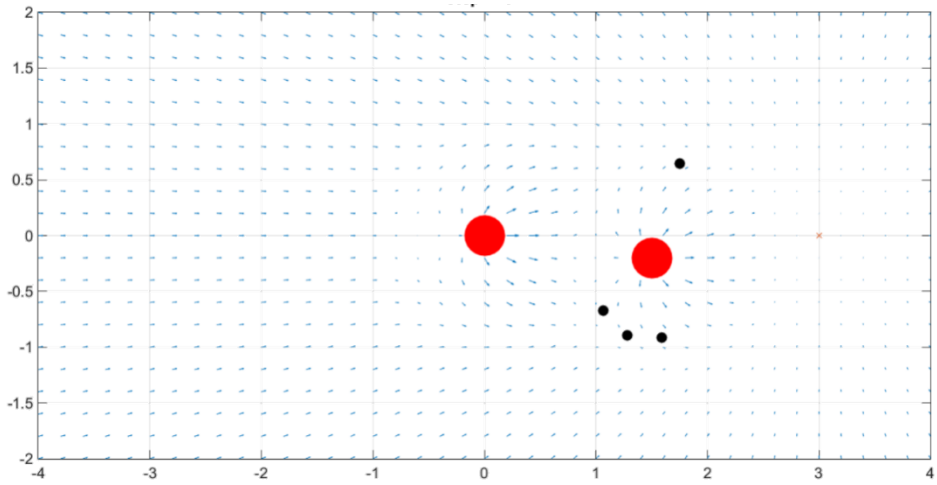
b) Paso de simulación: 50



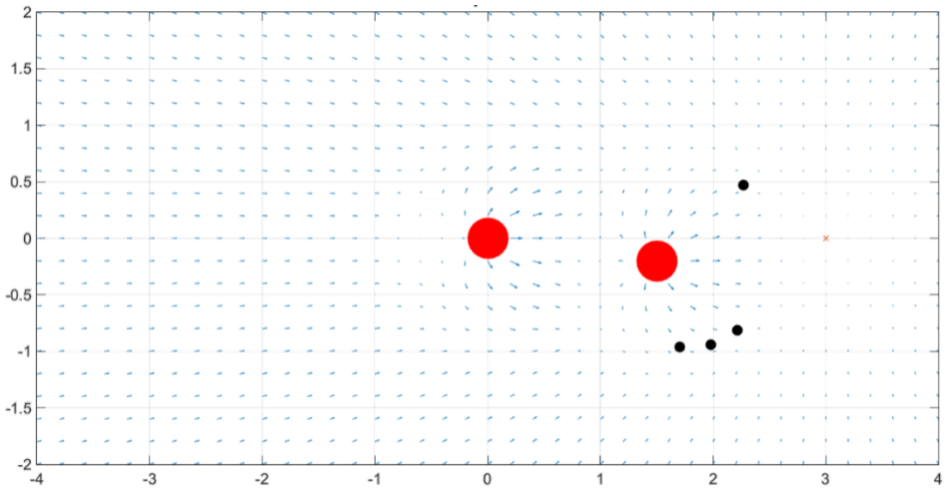
c) Paso de simulación: 100



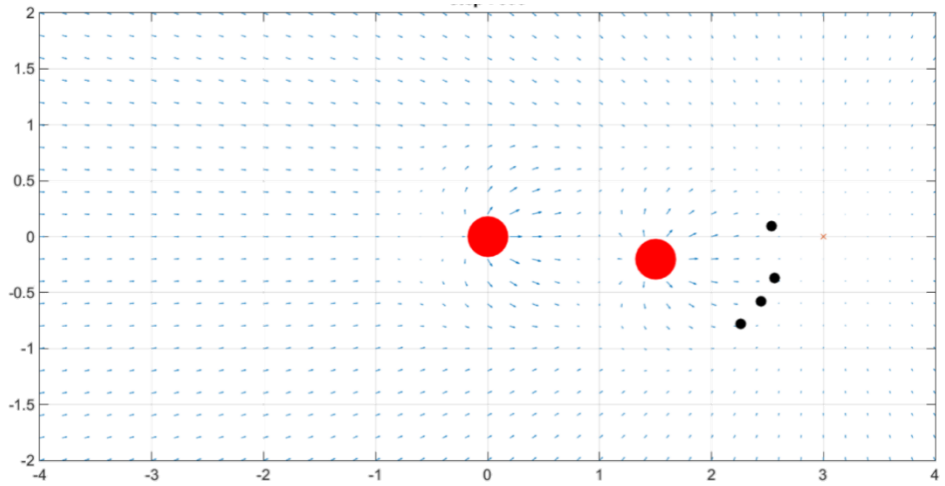
d) Paso de simulación: 150



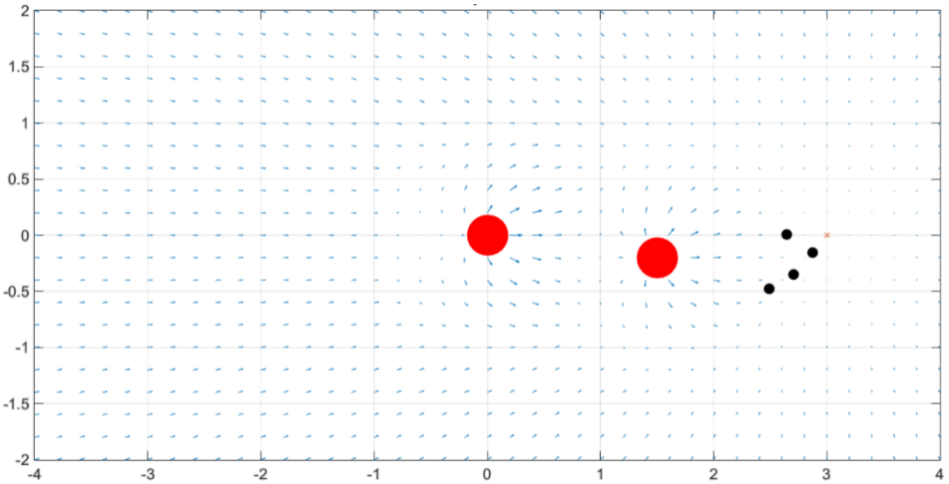
e) Paso de simulación: 200



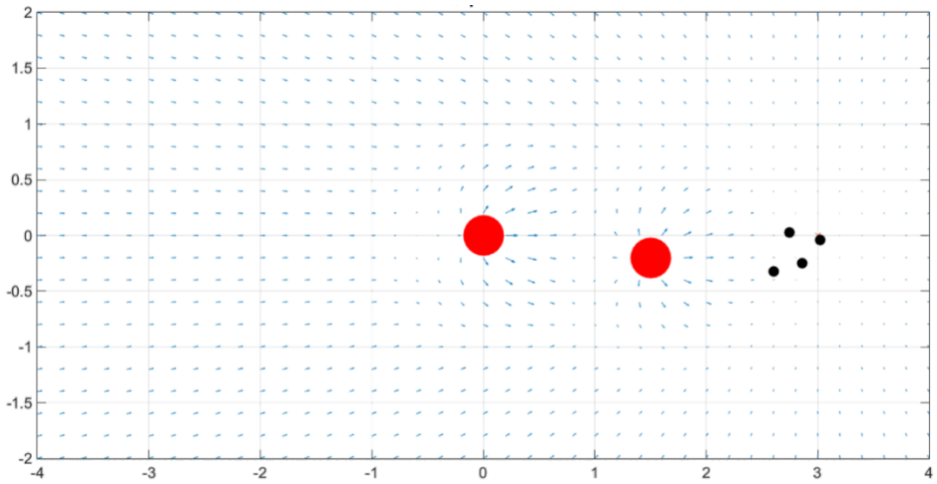
f) Paso de simulación: 250



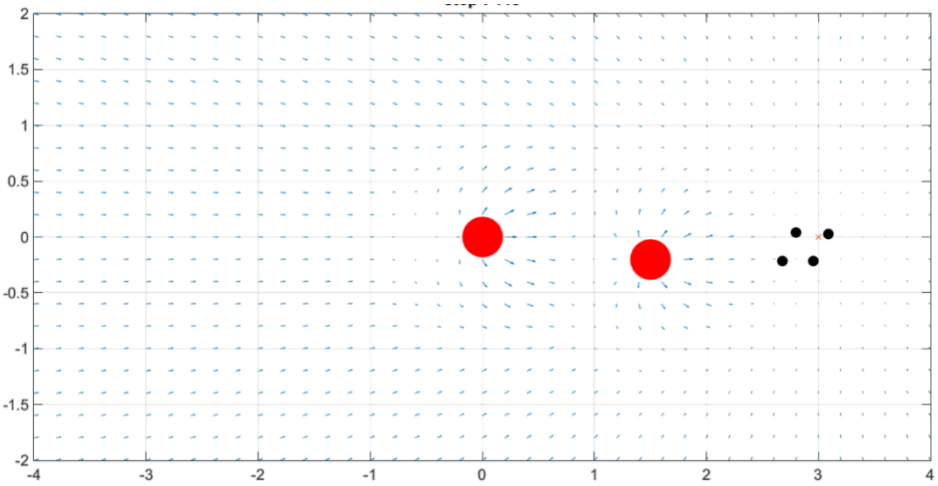
g) Paso de simulación: 300



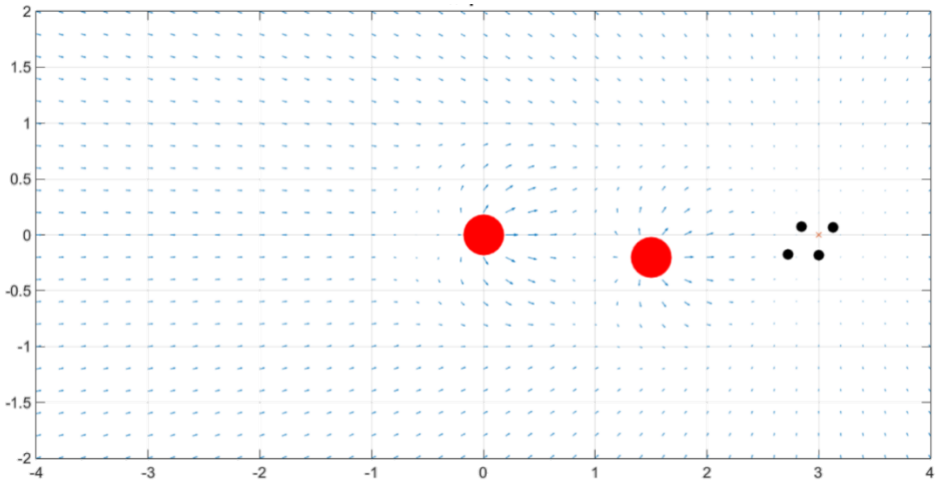
h) Paso de simulación: 350



i) Paso de simulación: 400



j) Paso de simulación: 450



k) Paso de simulación: 500

Figura 23. Secuencia del desarrollo de la simulación con la superposición de modelo de campo potencial y de interacción entre agentes mediante flocking. (Unidades en metros)

Con base en estos resultados se observa que la aplicación del algoritmo de *flocking* junto con el de campos potenciales hacen posible el tránsito de una cantidad *n* de agentes dentro de un espacio de trabajo hasta un punto objetivo dentro del ambiente en un tiempo finito, sin chocar entre ellos ni contra los obstáculos, por lo cual se valida la metodología para su posterior aplicación en agentes robóticos reales. Cabe resaltar que hasta este punto en las simulaciones se siguen considerando los agentes como partículas y aún no se integra el modelo cinemático de los mismos. Con el objetivo de analizar este modelo con condiciones más apegadas a un enjambre robótico real, se diseña y desarrolla a continuación un banco de pruebas para la realización de la experimentación correspondiente con agentes robóticos reales, además de implementar simulaciones que consideran ya la cinemática de los robots.

6. Desarrollo del banco de pruebas

Una vez planteados los modelos que se utilizarán como base para el control del enjambre de agentes robóticos, se decidió implementar el algoritmo y el sistema generado en un grupo de robots móviles, de tal forma que se compruebe su funcionamiento con un sistema físico real que valide la propuesta realizada. Por practicidad, y debido al área efectiva de trabajo limitada con la que se contaba, además de trabajos previos en el grupo MRG que validan su implementación, se decidió utilizar un conjunto de robots móviles diferenciales. Sin embargo, el modelo cinemático del robot resulta independiente del modelo de campos potenciales y de interacción entre agentes, por lo que el algoritmo presentado podría ser implementado sobre distintos tipos de robots móviles modificando la parte correspondiente a la cinemática del robot, siendo una configuración omnidireccional o cualquier topología de motorización diferente a la diferencial que en este trabajo se utilizará.

Antes de proceder con la aplicación en el sistema físico se desarrolló una simulación que permite incluir la parte del modelo cinemático de los agentes, de tal forma que ya no se consideraran únicamente como partículas, sino que ya se comportaran de la manera en que actuaría un robot móvil real. De esta manera, una vez validado el comportamiento, se tendría la capacidad y la seguridad de poder implementar el algoritmo en los robots móviles reales.

En términos generales, el sistema robótico que se busca implementar estará conformado como se muestra en la figura 24. La parte del cálculo de los campos potenciales y de interacción entre agentes estudiados en la sección anterior se ubican dentro de la parte denominada *procesamiento vectorial*. Debido a que se busca simplificar el desarrollo de tecnología a un nivel conceptual para validar el algoritmo propuesto, para la implementación física del sistema en cuanto a la capacidad de sensado de los agentes, se utilizará en el banco de pruebas un sistema de visión que hará el papel de los sensores que deberían tener montados cada agente para la detección del entorno en el que se encuentra (obstáculos y otros agentes). A pesar de esto, el sistema será programado de tal forma que cada agente pueda utilizar en su procesamiento únicamente la información que podría obtener directamente desde sus sensores, es decir, solo de los obstáculos y agentes que se encuentran dentro de su radio de detección. Con esto se pretende mantener el funcionamiento descentralizado del sistema, que resulta ser una característica importante para el comportamiento de enjambre. Así, el sistema de visión envía los datos de las coordenadas y ángulos (x, y, α) para que se realice con ellos el procesamiento vectorial, como resultado de este procesamiento se obtienen las velocidades lineales y angulares que se pretenden implementar en cada uno de los agentes. Después, se procede a hacer el análisis de la cinemática de los robots para obtener los valores buscados para aplicar directamente en los robots, en este caso las velocidades angulares de cada una de las llantas, y finalmente, se envía esta información a los robots para su implementación.

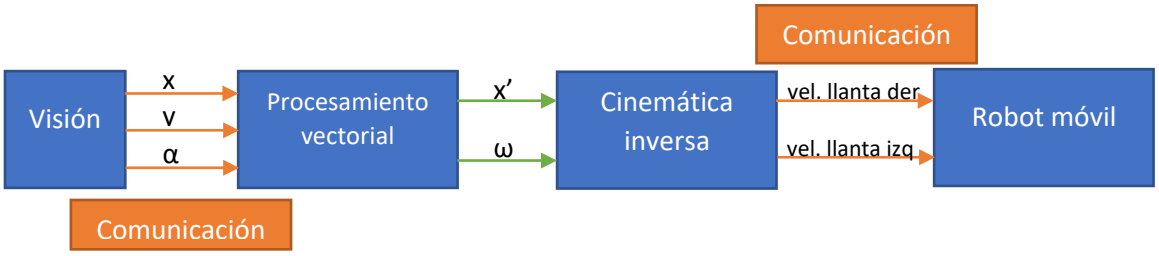


Figura 24. Diseño conceptual del sistema robótico.

6.1 Simulador

6.1.1 Desarrollo de un simulador del espacio de trabajo

Uno de los primeros pasos para poder validar los modelos matemáticos planteados anteriormente fue la generación de un simulador de condiciones reales que incluyera ya la cinemática de los robots, de modo que los parámetros a modificar o constantes dentro del modelo descrito pudieran ser probadas para una posterior aplicación en la realidad. Además, con este simulador es posible realizar pruebas preliminares sin poner en riesgo los robots físicos y sin gastar energía y tiempo innecesariamente. De este modo se obtiene un patrón de resultados con condiciones ideales (sin fricción o deslizamiento de los robots, sin inclinación y sin fallas o errores en el sistema de adquisición de datos), de modo que pueden presentar diferencias en comparación con los resultados homólogos que se obtengan en el sistema físico. Aun así, estos resultados preliminares de la simulación resultan ser una buena guía y orientación para determinar la viabilidad del modelo que se va a utilizar.

Para dichos propósitos se ocupó como base el software de *Simulink* de *Matlab* para generar el código necesario y poder ejecutar las simulaciones. Se seleccionó trabajar sobre este software debido a que en trabajos previos desarrollados en el MRG se presentó la viabilidad del programa para este tipo de aplicaciones con robots móviles, además de que incluye una gran cantidad de bloques útiles para las pruebas como para el procesamiento y transmisión de los datos. El funcionamiento del programa implementado para esta simulación se explica más detalladamente en el diagrama de flujo de la figura 25.

En el bloque denominado *"cinemática del robot móvil"* se programaron las ecuaciones vistas en la sección del desarrollo teórico, por lo que, correspondientemente, la entrada del bloque es la magnitud de las velocidades lineal y angular que se pretenden implementar en el agente. Éstas se obtienen tras efectuar todo el análisis correspondiente a los campos potenciales y la interacción entre agentes que se presentó previamente. De esta forma, el bloque de la cinemática del robot toma las velocidades lineal y angular deseadas generadas a partir del modelo, y produce como salidas las velocidades lineales y angulares que tendría el agente robótico en el sistema coordenado; por lo tanto, para poder determinar la posición resultante del agente dentro del mismo bloque, considerando sus características físicas, es necesario integrar las diferentes velocidades y así obtener sus coordenadas cartesianas en el sistema bidimensional. Con esto es posible efectuar una simulación gráfica en la que se aprecie el comportamiento que tendrían los agentes en determinada situación.

En el diagrama se muestra también, enmarcado con un recuadro rojo, la parte correspondiente al cálculo de los campos potenciales y la interacción entre agentes. Se observa, finalmente, que este diagrama representa un ciclo que estará actualizando constantemente las coordenadas de las posiciones de los agentes, generando así una simulación gráfica del comportamiento que tendrían los robots móviles con este modelo. Si se grafican en cada iteración del ciclo las posiciones de los agentes, obstáculos y objetivo es posible obtener la simulación.

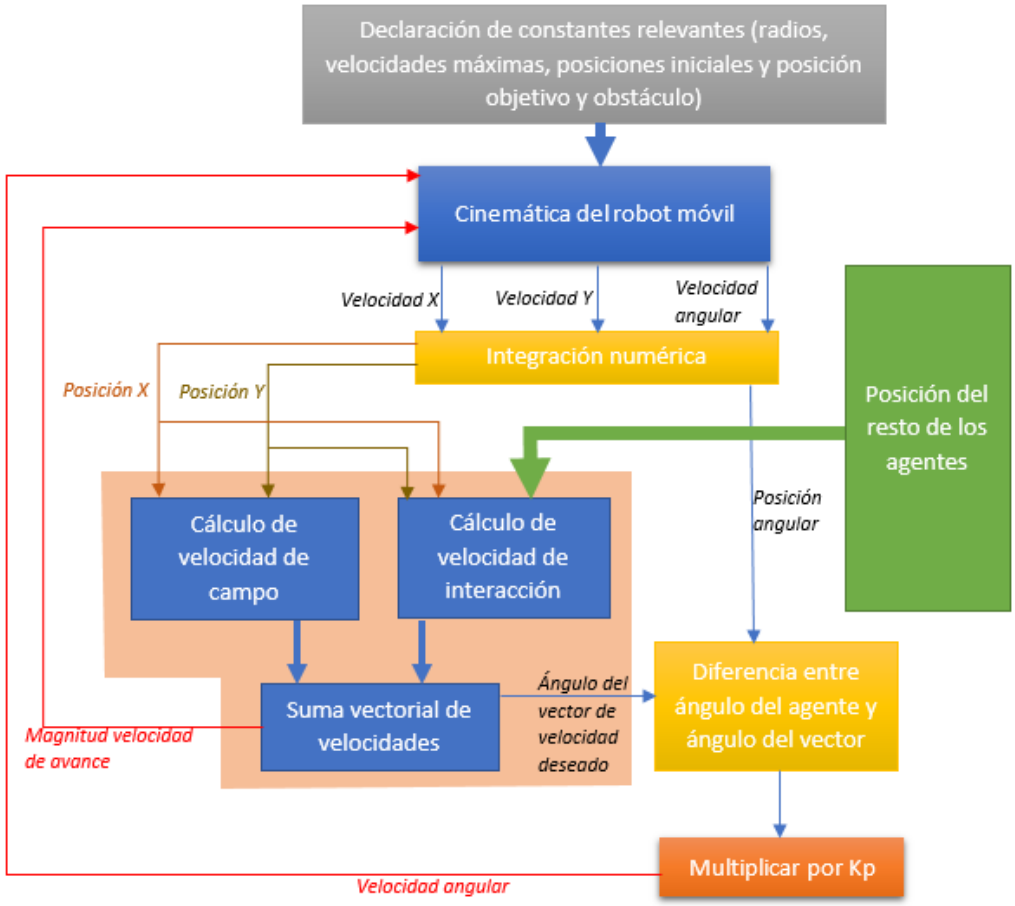


Figura 25. Diagrama de flujo para la simulación computacional del sistema robótico.

La justificación para la cantidad de agentes que deberían representarse en el entorno de simulación se basa en la capacidad computacional y la relevancia de los resultados. Se busca utilizar un número bajo de agentes, ya que, de lo contrario, los recursos computacionales se elevarían demasiado haciendo la simulación muy lenta, sin embargo, con uno o dos agentes no se aprecia claramente el comportamiento de enjambre que es justamente el elemento de estudio en este trabajo. Es por esto por lo que se decidió ocupar un mínimo de tres agentes.

Es importante mencionar que hasta este punto la simulación se realizó ocupando el comportamiento cinemático de un robot diferencial (2,0) descrito en la sección 4.1.1. En el caso de que se pretenda utilizar distintas configuraciones robóticas será necesario modificar la cinemática del robot, por ejemplo, por la de un robot omnidireccional. El cambio entre estos dos modelos, debido a la forma modular de la programación, sería tan sencillo como cambiar la matriz contenida en el bloque de cinemática del robot móvil por la matriz que describe el comportamiento cinemático del robot omnidireccional.

Con el fin de mantener la localidad y escalabilidad del algoritmo, el código generado se programó de la misma manera en cada uno de los agentes involucrados, por lo que cada uno de ellos tendrá su comportamiento individual con el mismo funcionamiento que el resto de los agentes. Finalmente, se tradujo el diagrama presentado a códigos y diagramas en *Matlab* y *Simulink* para generar la simulación.

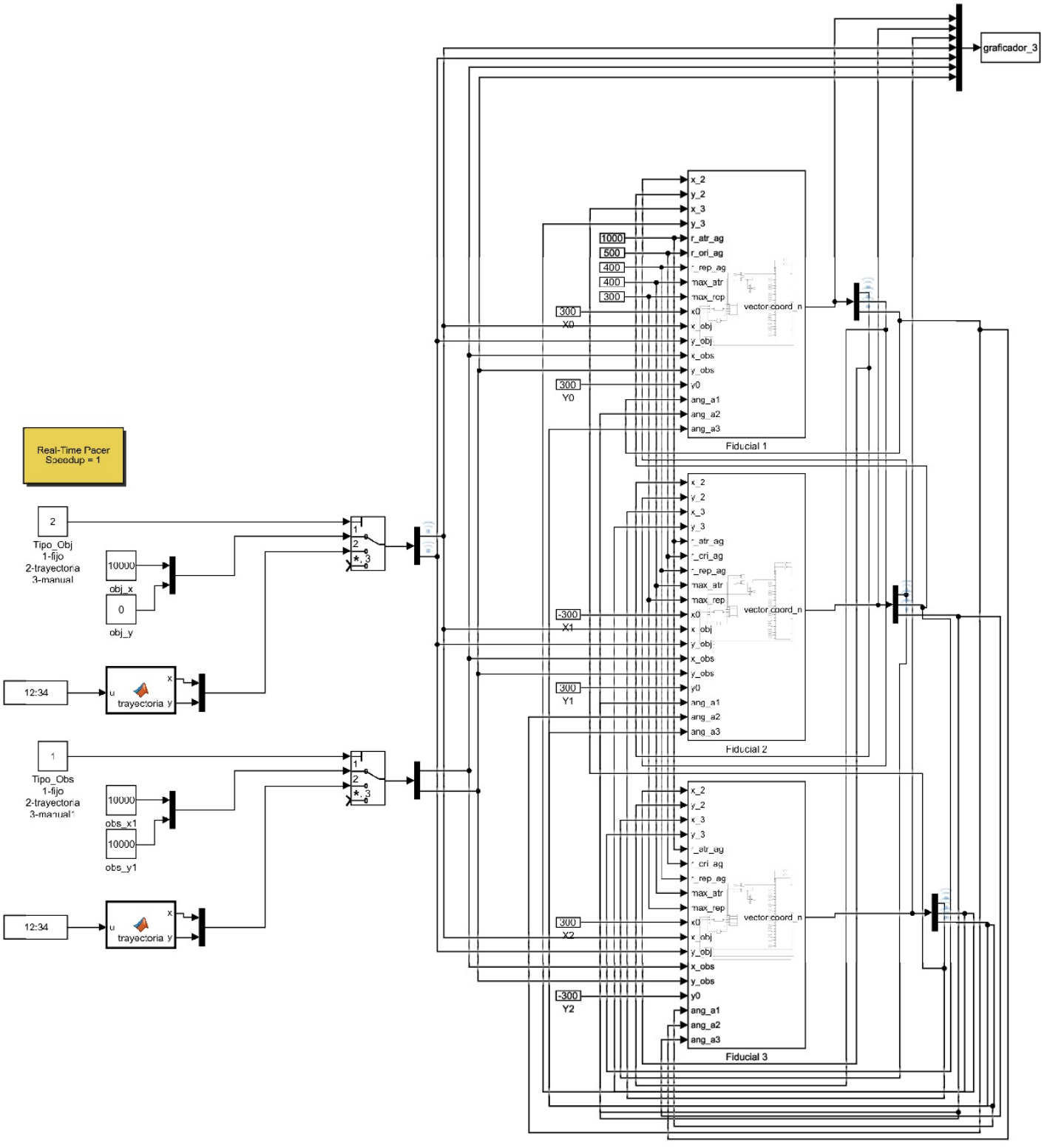


Figura 26. Diagrama de bloques para simulación computacional en Simulink.

En el diagrama anterior es posible observar tres grupos que representan la programación de cada uno de los agentes involucrados en la simulación. También se observan algunos bloques de constantes representando la declaración de condiciones iniciales, posiciones trascendentes y constantes relativas a las ecuaciones de control del campo potencial y la metodología de *flocking*. Es importante mencionar que cada uno de los recuadros (de color gris) se encuentra igualmente codificado y dentro de cada uno de ellos está contenido, a su vez, el diagrama de ejecución descrito en la figura 27, que representa la programación individual de cada agente. En esta figura se explica la analogía de la programación con el diagrama de flujo presentado en la figura 25.

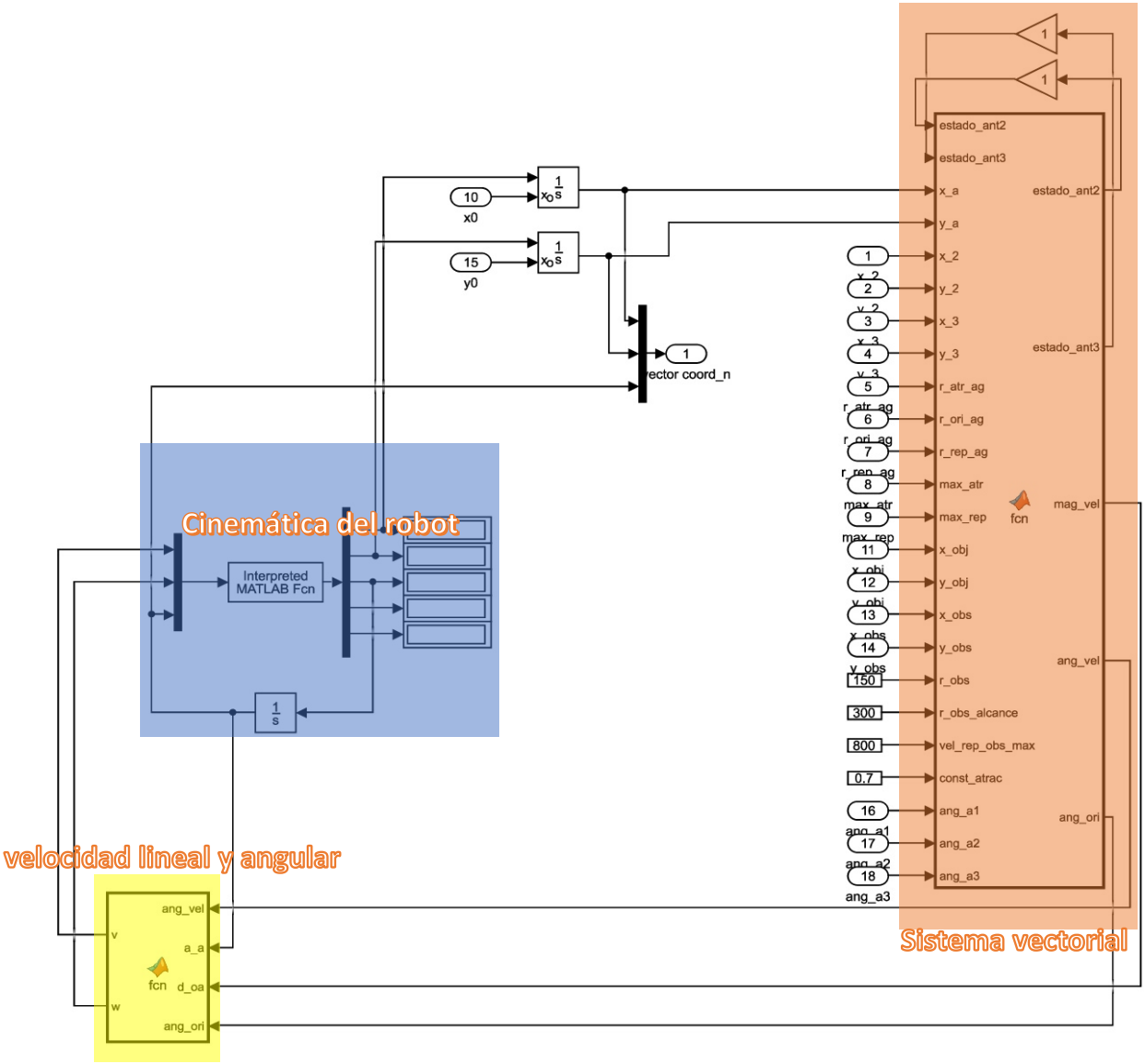


Figura 27. Diagrama de bloques de cada agente para la simulación en Simulink.

En la figura 27 se aprecia con mayor detalle la manera en la que fue implementado el modelo. En la sección identificada en color rojo se realiza el procesamiento requerido para el modelo de campos potenciales y el modelo de *flocking*, aquí está contenida una función de *Matlab* en la que se programó la velocidad de atracción al objetivo y la velocidad de repulsión de los obstáculos, así como también los efectos de atracción, repulsión y orientación generados por la interacción entre agentes. Por otro lado, la sección marcada con color azul utiliza la información generada por el modelo en conjunto con la cinemática del robot para calcular las nuevas coordenadas y ángulo del agente y actualizar así, paso a paso, la simulación del sistema. Debido a que se decidió modelar la cinemática del robot en función de la magnitud de la velocidad lineal ($x_r{}'$) y la velocidad angular (θ'), se incluyó otro elemento, marcado de color amarillo en la imagen, que calcula estos valores a partir del vector de velocidad generado por el modelo y las coordenadas y ángulo actuales del agente. Finalmente, la información respectiva de cada agente es enviada a un nuevo bloque de código que se encarga de transformar esto en una representación gráfica y generar la simulación conforme se van actualizando los datos.

Debido a las limitaciones físicas de los robots diferenciales y a su forma de desplazamiento, resulta poco estable el sistema de velocidades implementado en torno al punto de equilibrio o de estabilidad. Los robots deben girar para alcanzar la orientación deseada y poder desplazarse hacia su objetivo, por lo que al momento de que uno de ellos sobrepasa este punto de equilibrio deberá dar un giro completo para orientarse a su destino y, en consecuencia, se mantendrá en movimiento hasta que el agente se ubique exactamente en el punto objetivo. Para evitar esto se propone establecer un área alrededor del objetivo y considerar que dentro de ella el agente ya ha llegado a su destino y, por tanto, detenga su movimiento. De esta manera, se puede estabilizar el sistema sin necesidad de ubicarse los agentes exactamente en el punto definido como objetivo y evitar así que se mantengan en constante movimiento cerca del destino. Al tener un enjambre constituido por más de un agente, considerando que todos ellos se encuentran programados de la misma manera y con el mismo punto objetivo, es imposible que todos se ubiquen directamente sobre el mismo punto objetivo establecido ya que el modelo de flocking evita que haya colisiones y, por tanto, tiende a separar a los agentes entre sí. En este caso, ninguno llegará exactamente al punto objetivo, sino que será el centroide de la formación el punto que tienda a ubicarse sobre él, quedando todos los demás agentes bajo el efecto de la atracción al objetivo en equilibrio con todos los efectos de interacción entre agentes.

Para determinar el área que se considerará como objetivo alcanzado se estableció el término de distancia de protección (d_p). Cuando alguno de los agentes se ubique a una distancia al objetivo menor a la distancia de protección éste dejará de responder al efecto de atracción del objetivo, quedando únicamente influenciado por el resto de los agentes que se ubiquen alrededor de él. Esto se puede imaginar también como una circunferencia con centro en el objetivo y radio d_p dentro de la cual la velocidad de atracción al objetivo es cero (Figura 28). Adicionalmente, se optó por implementar una idea semejante a la distancia de protección, pero ahora para evitar que los efectos del modelo de *flocking* impidan la estabilidad en el sistema. En este caso en lugar de definir un área se estableció una velocidad mínima, de tal forma que, si la velocidad del efecto del modelo de *flocking* es menor que dicha velocidad, ésta se hará cero a fin de permitir al sistema estabilizarse.

Con estas consideraciones se desarrolló el sistema para hacer las simulaciones y se efectuaron varias pruebas para analizar el comportamiento de los agentes, considerando condiciones cinemáticas de posibles agentes reales, y proceder con la aplicación en un entorno físico.

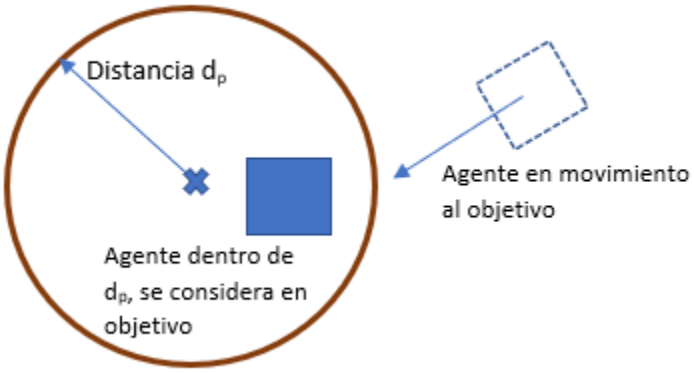


Figura 28. Ejemplificación de la distancia de protección.

6.1.2 Diagramas de simulación con cinemática diferencial

A continuación, se presenta el análisis de una de las simulaciones realizadas con este sistema. Las tablas siguientes muestran las condiciones consideradas en dicha simulación para el ambiente y los agentes implementados.

Tabla 2. Condiciones estándar del ambiente para simulación computacional.

Configuración de ambiente [1 Obstáculo]			
Coordenadas objetivo [mm]		**Coordenadas obstáculo[mm]**	
X: -400	Y: 100	X: 200	Y: -200
Límite del espacio		**R obstáculo[mm]**	**R alcance(obs)[mm]**
X [-1000, 1000]	Y [-1000, 1000]	100	300
V máx. rep.[mm/s]	600	**μ = 1**	
V máx. atr. [mm/s]	300		

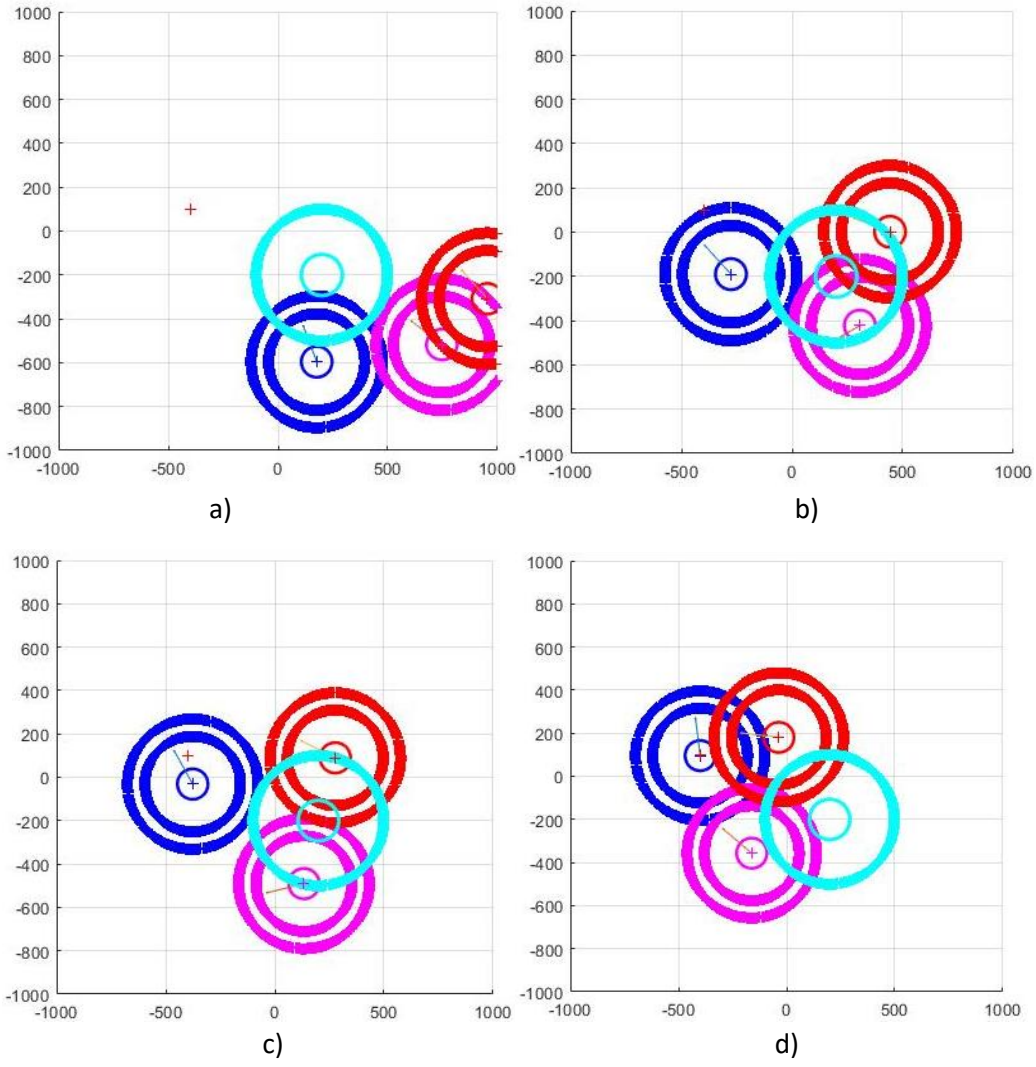
En la tabla 2 se presentan las condiciones del ambiente en el que se simuló el sistema, en este caso se presenta un obstáculo con radio de 100 [mm] en las coordenadas (200,-200) cuyo radio de alcance es de 300 [mm]. Mientras que el objetivo se ubica en las coordenadas (-400,100).

Para las configuraciones correspondientes a los agentes, después de varias pruebas se observó que los agentes tenían un comportamiento adecuado (tiempo de estabilización finito, sin vibraciones excesivas en su movimiento y una función continua de velocidad) con los valores presentados en la tabla 3.

Tabla 3. Configuración de los agentes para simulación computacional.

Configuración de interacción [3 Agentes]					
Coordenadas iniciales[mm]					
Agente 1		**Agente 2**		**Agente 3**	
X: 800	Y: -500	X: 200	Y: -600	X: 1000	Y: -300
R atr. [mm]	**R rep. [mm]**	**R ali. [mm]**			
400	320	350			
V máx. rep.[mm/s]	**V máx. atr. [mm/s]**				
200	100				

Con lo anterior se realizó la simulación mostrada en la figura 29 con la ayuda de *Simulink*.



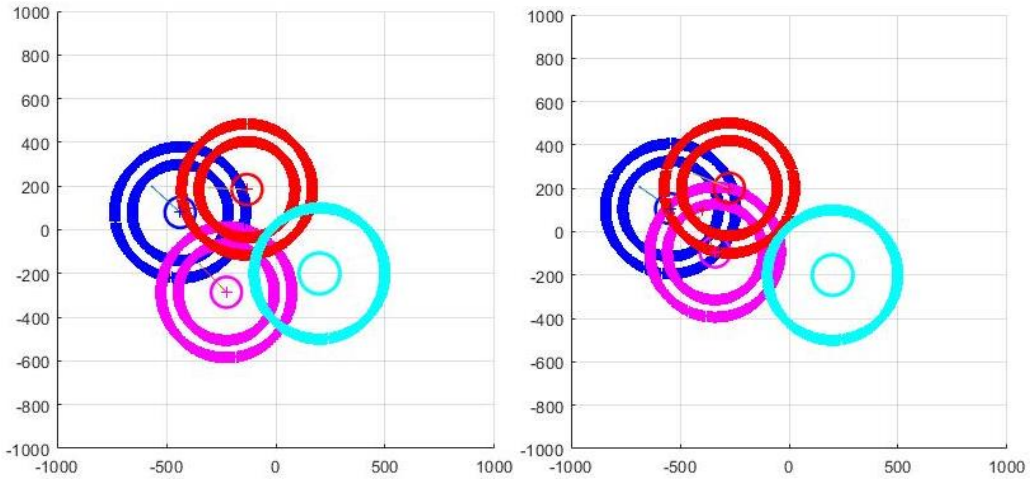


Figura 29. Secuencia de imágenes de simulación de tres agentes con un obstáculo. (Unidades en milímetros)

En estas imágenes es posible observar que los agentes se desplazan siguiendo un vector de velocidad producido por la interacción entre el campo potencial y la definición de velocidades de interacción entre ellos. Es importante observar cómo los agentes evitan chocar con el obstáculo y se desplazan hasta llegar a una distancia aceptable del objetivo en donde se quedan quietos tras un tiempo de 5.6 [s] de acuerdo con la simulación realizada en *Simulink.*

Los colores en la simulación representan la diferenciación entre agentes (azul y magenta) y el obstáculo (cian). En el caso del obstáculo, el círculo concéntrico interior representa la barrera física del obstáculo definida por el radio del obstáculo (R_{obs} en la tabla de datos), mientras que el círculo exterior representa la distancia a la que los agentes son capaces de comenzar a detectar el obstáculo. Esto es importante debido a que este radio puede depender, en aplicaciones prácticas, del tipo y calidad del sensor ocupado para detectar el obstáculo en los agentes. En el caso de los agentes, los círculos representan los radios principales, tanto de interacción/atracción como de repulsión, exterior e interior, respectivamente.

También es posible observar flechas que representan la orientación de cada uno de los robots móviles en los momentos indicados, es decir, la dirección de desplazamiento hacia adelante de cada uno. Dicho vector proviene del centro del agente (el cual representa el punto *P* del robot móvil analizado en la sección 4.1.1) y se puede observar que representa la tendencia clara del movimiento del agente. Se incrementa cuando se encuentra más lejos del objetivo, mientras que al acercarse su magnitud va disminuyendo hasta la última imagen de la secuencia, lo cual indica un asentamiento del sistema.

Retomando un tema visto en párrafos anteriores es importante recalcar que, al ser un sistema multiagente, se trata de una llegada al objetivo no completa, sino que cada uno de los agentes tratará de llegar lo más cerca posible de dicho objetivo hasta el punto en el que se equilibren las velocidades del sistema. En consecuencia, la llegada al objetivo se representa por la llegada del centroide de la formación, no por la llegada de alguno de los agentes en particular, lo cual también respeta la no centralidad que plantea la idea de la inteligencia de enjambre para la formación adaptativa de la robótica móvil debido a que no se trata de un enjambre enfocado en un agente

líder, sino que todos tienen la misma meta de llegar al objetivo. En este caso, el centroide de la formación puede obtenerse considerando las posiciones de todos los agentes que lo conforman y calculando un promedio de éstas. En la figura 30 se observa la condición final de llegada del enjambre robótico al objetivo, que se localiza prácticamente en el centroide de la formación.

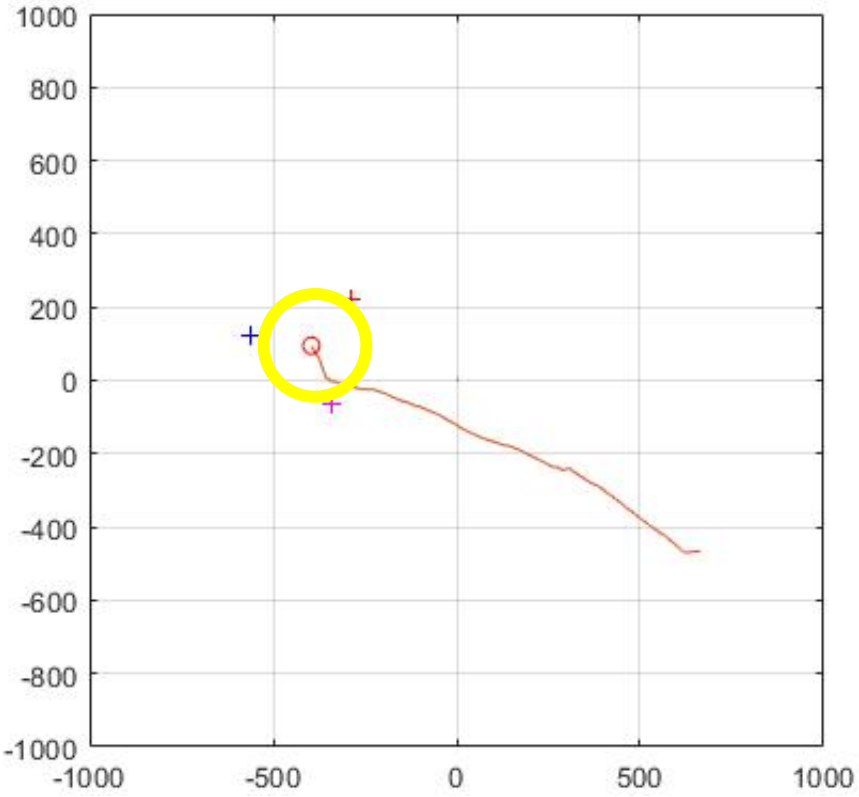


Figura 30. Posicionamiento del centroide de la formación en el objetivo. Distancia promedio entre agentes de 289.17 mm (diámetro del círculo amarillo). Línea roja marcando la trayectoria del centroide durante el experimento. (Unidades en milímetros)

Con el fin de verificar que se cumple esta tarea de desplazar el enjambre robótico hasta el punto objetivo se toman las coordenadas reportadas por el sistema de simulación de los agentes en los instantes después de los 6.9 [s] que se tardaron en estabilizarse, a partir de los cuales es posible calcular el centroide de la formación y compararlo con el punto objetivo.

Coordenadas del agente 1	
X: -340.52 mm	Y: -62.43 mm

Coordenadas del agente 2	
X: -560.88 mm	Y: 123.68 mm

Coordenadas del agente 3	
X: -289.73 mm	Y: 223.29 mm

Utilizando un promedio de las coordenadas presentadas anteriormente, es posible obtener la coordenada (X, Y) que representa el centroide de la formación, así como la distancia promedio final entre ellos.

$$d_{prom} = 289.17\ mm$$

$$P_{centroide} = (-397.04,\quad 94.84)\ mm$$

Con dichos datos, es posible determinar cuál es el error entre el centroide de la formación presentado en $P_{centroide}$ y el punto objetivo. La distancia entre estos dos sería:

$$Dist_{centroide} = 5.95\ mm$$

Esta distancia es de 0.59 cm, lo cual para las dimensiones que se están manejando con un espacio de trabajo del orden de magnitud de 1 [m] se trata de un error relativamente bajo. La existencia del error de la ubicación del enjambre con respecto al objetivo se debe al conjunto de consideraciones de protección para favorecer la estabilización del sistema; tanto la distancia a la cual la atracción del campo se vuelve cero, como la velocidad total a partir de la cual se considera que el agente se deja de mover.

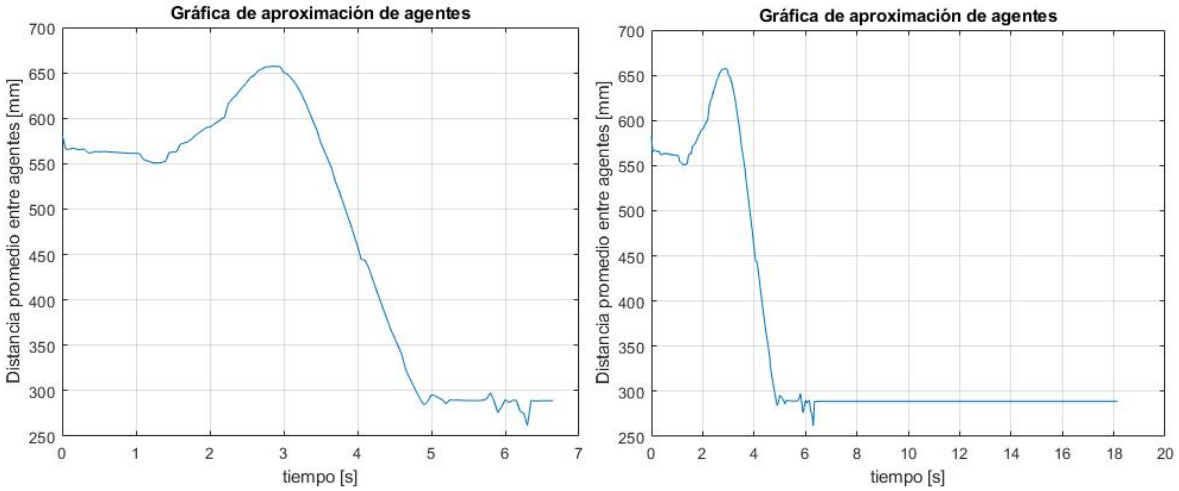


Figura 31. Gráfica de aproximación entre 3 agentes.

Otro punto relevante de análisis es la distancia entre agentes a lo largo de la prueba que se muestra en la figura 31. En esta gráfica se puede apreciar un aumento en la distancia entre agentes cerca de 1.5 segundos después de haber comenzado el experimento debido a que, en ese momento, los agentes toman la decisión de separarse al encontrarse en el camino de la formación al obstáculo. Posteriormente, una vez evadido el obstáculo, tienden a dirigirse hacia el objetivo, reduciendo la distancia entre ellos paulatinamente. A su vez, es importante hacer hincapié en el hecho de que la estabilización se alcanza una vez que los agentes no se mueven, no necesariamente cuando mantienen una distancia estable entre ellos, ya que pueden seguir desplazándose hacia el objetivo a pesar de mantener la misma distancia entre sí.

Siguiendo esta idea, es posible determinar cuantitativamente el instante en el que se alcanza la estabilización observando la velocidad en las llantas de los agentes, las cuales se calculan a partir del bloque de cinemática inversa. Dichas velocidades se observan tanto en la llanta derecha como en la izquierda, ya que con su interrelación se producen los giros y avance del agente. Para su estudio se presentan las gráficas de la figura 32.

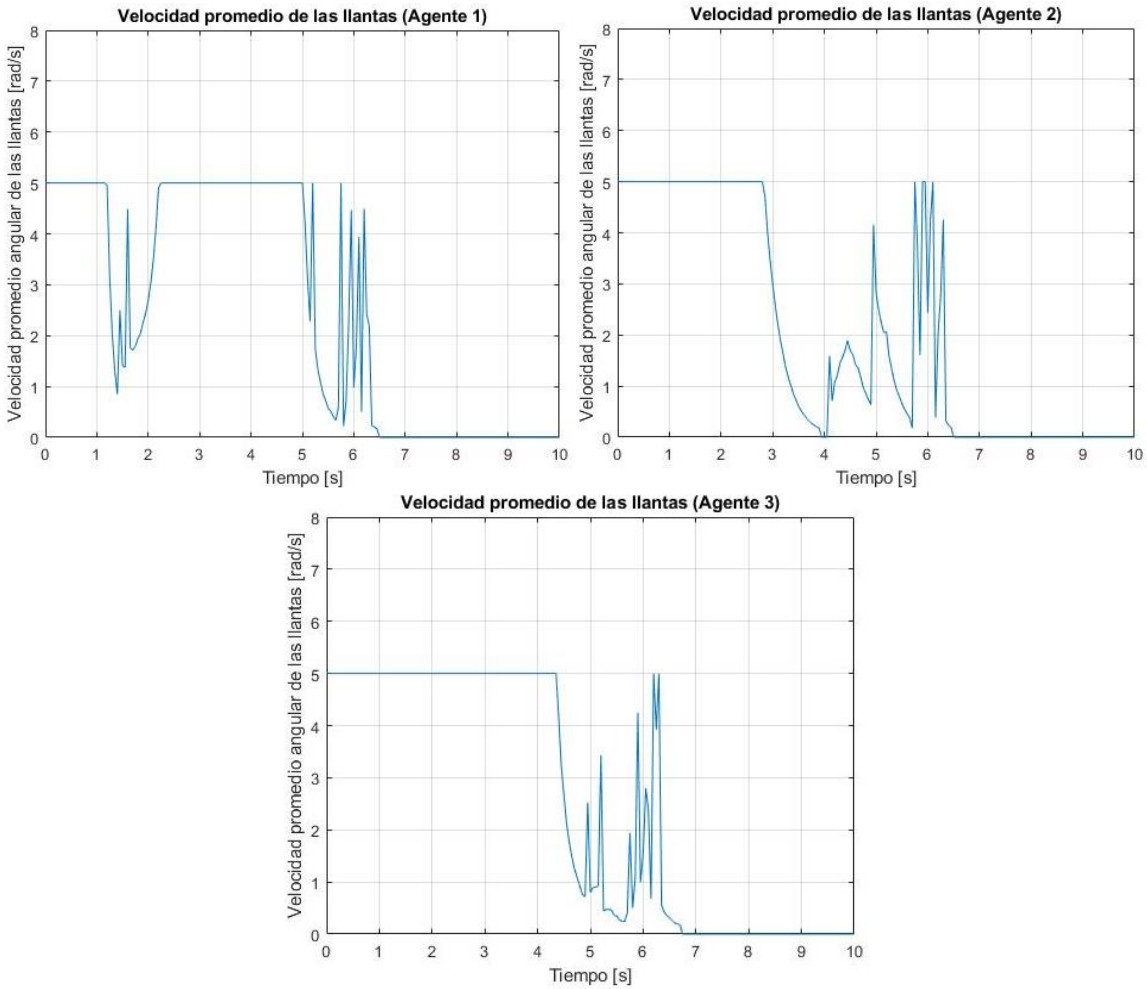


Figura 32. Gráficas de velocidad en llantas de los agentes.

En la figura 32 es posible observar el comportamiento del promedio de las velocidades de las llantas de cada uno de los agentes durante el experimento. Dicho valor nos aporta la velocidad neta de avance del agente durante su trayectoria en dirección al objetivo. Las gráficas anteriores presentan variaciones y picos significativos que representan cambios en el comportamiento de los agentes, en los intervalos de tiempo en los que el agente se desplaza con una velocidad constante se observa una línea constante en la gráfica, mientras que al cambiar la rapidez de desplazamiento o al cambiar de dirección esto se ve reflejado como cambios también en la gráfica de velocidades de los motores. Las variaciones se presentan principalmente debido a la interacción de los agentes, tanto con el resto de ellos, como con el obstáculo. Estas variaciones se presentan en frecuencias relativamente altas, por lo cual, se puede suponer que en un sistema físico los motores de los agentes reaccionarán de alguna manera como filtros pasa bajas y dicho efecto tenderá a ralentizar la reacción en las llantas favoreciendo un movimiento más suave y estable de los robots en el experimento, aunque de manera general, la hipótesis es que el comportamiento tenderá a ser similar en la implementación física. Otro hecho relevante en estas gráficas que apoya la realización y validación del algoritmo es que los agentes llegan a un estado en el que no se mueven o, en otras palabras, la velocidad neta de avance es cero. Se observa en las gráficas que dicho estado es alcanzado aproximadamente entre el segundo 6.5 y 6.8 en cada una.

Tras analizar estas simulaciones se determinó que el modelo y algoritmo es aplicable y se procedió a diseñar su implementación en un ambiente real con el uso del enjambre de robots móviles físicos. Cabe destacar que este tipo de sistemas resultan ser caóticos, es decir, que con variaciones mínimas en las condiciones iniciales se presentan cambios muy significativos en el desarrollo del sistema. Adicionalmente, la implementación en un sistema físico real implica que se involucren algunas variables no consideradas en las simulaciones, como el posible deslizamiento de alguna de las llantas de los robots, limitaciones en el funcionamiento mecánico de los robots o incluso alteraciones en el proceso de sensado y de transmisión de datos. Es por esto por lo que las simulaciones presentarán, seguramente, variaciones significativas con respecto de cada una de las pruebas efectuadas en el entorno real. Sin embargo, aunque no se presente exactamente el mismo desarrollo que en la simulación, es posible hacer pruebas que validen el correcto funcionamiento del algoritmo planteado.

Para la implementación del algoritmo se diseñó en primera instancia un ambiente inteligente en el que se desarrollarán los robots, que posea las respectivas capacidades de sensado, procesamiento y transmisión de la información. Además, debe tener la capacidad de la aplicación del modelo en un conjunto de robots móviles diferenciales.

6.2 Ambiente inteligente

6.2.1 Sistema de visión

Para dotar al sistema de la capacidad de obtener la localización de los agentes se basó su implementación en la desarrollada por Gálvez, J. [41] en la cual se relata el uso de *ReacTIVision* como base para la identificación de agentes dentro del espacio de trabajo. *ReacTIVision* se trata de un software de *Open Source* que goza de licencia libre para utilizarla con fines educativos que fue desarrollado por Martin Kaltenbrunner y Ross Bencia [42]. Este sistema se basa en la identificación de símbolos por medio de un dispositivo de adquisición de imágenes (cámara digital) para realizar su localización con respecto al sistema coordenado del mismo dispositivo. Los símbolos se encuentran codificados utilizando números enteros y existen 255 de ellos, cada uno único por su dibujo representativo denominado *fiducial,* o amiba por su traducción al español. Un ejemplo se muestra en la figura 33.

Figura 33. Ejemplo de un fiducial.

El sistema *ReacTIVision*, como se menciona en el trabajo de Gálvez J. [41], es extremadamente intuitivo y fácil de utilizar debido a que funciona bajo el concepto de *plug* and *play*, ya que no es necesaria una configuración adicional, sino que sólo es necesario conectar la cámara o el sistema de adquisición de imágenes y correr el programa para que comience a funcionar.

Por otro lado, también es importante mencionar, para fines de este trabajo, que el programa *ReacTIVision* comunica la información de localización y ángulo del *fiducial* a través del protocolo TUIO para objetos multimedia. Dicho protocolo de comunicación cuenta con dos clases de mensajes principales: [43]

- *SET*. Se utiliza para enviar la información relacionada con el estado de los objetos, por ejemplo, la posición y la orientación.
- *ALIVE*. Indica cuáles son los objetos que se encuentran presentes o que existen en la visualización.

Estas dos funciones dentro del protocolo TUIO permiten la actualización de la información de *fiducials* entrantes en el entorno de trabajo, así como la actualización de posición y ángulo cuando éstos se mueven.

Para que *ReacTIVision* tenga acceso a mandar la información a aplicaciones externas, el sistema cuenta con un cifrado de datos con base en el protocolo OSC (*Open Sound Control*) y manda la información mediante UDP.

El protocolo UDP (*User Datagrams Protocol*) es utilizado por diversas aplicaciones para la transferencia de información debido a que se trata de un protocolo sencillo, práctico y veloz para comunicar interfaces. La unidad mínima de envío de datos se trata del datagrama, en el cual, cada una de las partes involucradas en la comunicación debe hacerse cargo de empaquetar y desempaquetar respectivamente dependiendo del lado de la comunicación en la que se encuentre. Se trata de un protocolo para envío de cadenas de texto, la cual puede ir codificada con instrucciones o datos de un programa a otro. Dicho protocolo se encuentra descrito en la RFC 768 y no consume cantidades importantes de recursos debido a que no está pendiente de la calidad de la recepción de la información, sino que el receptor, debe garantizar por sus propios medios que el segmento haya llegado adecuadamente para poder ser procesado.

La transmisión mediante UDP se realiza usualmente en aplicaciones de audio o video donde la fiabilidad en la transmisión en tiempo real no tiene gran trascendencia; si de un instante a otro se pierde un componente en la cadena de audio no resulta importante debido a que un instante después el mismo segmento de audio habrá cambiado, lo importante en este protocolo es que, en condiciones generales e imperfectas de recepción de datos, el mensaje pueda ser comprensible. Por otro lado, el sistema UDP tiene mucho que ver con el modelo de red OSI (*Open System Interconnection*) el cual maneja también en una de sus capas la identificación de servidores y host por identificadores únicos llamados *IP´s*. La explicación completa de cómo se relaciona este último con el protocolo UDP es posible encontrarlo en el trabajo de tesis de Gálvez, J tomado como referencia [41].

La comunicación OSC proveniente de *ReacTIVision* es posible desempaquetarla por medio de diferentes recursos. En el caso de la referencia [41] se utiliza java como puente intermedio para conectar *ReacTIVision* y desempaquetar el protocolo OSC, enviando la información correspondiente a *Simulink* para su posterior análisis y procesamiento de datos en tiempo real. Sin embargo, en diferentes trabajos desarrollados dentro del *Mechatronics Research Group* se ha reportado que dicho programa presenta algunas deficiencias entre las que destacan las siguientes:

1. El área de trabajo se limita al uso de una sola cámara.
2. El sistema basado en Java tiene bajas prestaciones para lograr la fácil escalabilidad del sistema, ya que es necesario recodificar el script para adaptar la cantidad de agentes dentro del espacio de trabajo. El programa utilizado previamente obliga la presencia necesaria de un determinado número de fiducials específicos para su correcto funcionamiento.
3. El sistema tarda en procesar la información de ReacTIVision, lo que, para aplicaciones reactivas en tiempo real puede provocar fallas en el sistema.

Según la documentación de *ReacTIVision*, es posible desempaquetar la información por diferentes medios, incluyendo: *C++, Java, C#, Processing, Pure Data y Max/Msp.*

Para mejorar las prestaciones que ofrece el programa, así como ajustarlo a los requerimientos particulares de este trabajo, se decidió modificar el programa base ofrecido por *ReacTIVision* para generar nuestra propia versión. En este caso se seleccionó trabajar sobre el programa de *C#* debido a que es el lenguaje de programación con el que se tenía mayor experiencia. Dentro de la documentación de *ReacTIVision* se incluye un proyecto de Microsoft Visual Studio, por lo que se utilizó este mismo software para editar el código del programa.

6.2.2 Desempaquetamiento en *C#* y ampliación del espacio de trabajo

En diversas tesis realizadas en el *Mechatronics Research Group* de la Facultad de Ingeniería de la UNAM se ha estudiado el movimiento de robots móviles a partir de un sistema de visión artificial basado en *ReacTIVision* y un *Middle Worker*[41] específico dependiendo de la implementación requerida. En dichos trabajos[37], [39], [44] se han reportado entornos de visión con dimensiones en torno a 100x100 cm que para aplicaciones con un solo robot resulta suficiente según las conclusiones de dichos trabajos. A pesar de ello, existen algunos otros trabajos como en [45] donde el autor menciona que el espacio es reducido para realizar pruebas enfocadas al análisis del

desempeño de robots móviles, por lo que, en este caso, el autor se ve en la necesidad de modificar el comportamiento del robot para que no llegara a valores límite del alcance del dispositivo de adquisición de imágenes.

Debido a lo comentado en el párrafo anterior, la necesidad de ampliar las dimensiones del espacio de trabajo para dar mayor libertad al análisis del desempeño se consideró como un tema importante. Esto con la finalidad de poder realizar los diferentes experimentos con múltiples agentes de forma eficiente. Además, debido a la necesidad de mantener a más de un agente en el mismo espacio de trabajo, se hace evidente el requerimiento de tener un espacio mucho más amplio para poder operar sin que los agentes salgan del campo de visión de las cámaras, que serán los elementos encargados de transmitir su posición al sistema de control. Para poder solventar este problema la estrategia fue aumentar el campo de visión con la implementación simultánea de dos cámaras que se conecten al mismo tiempo al sistema de adquisición de datos. (Figura 37)

La integración de las dos cámaras se realizó basándose en el hecho de que *ReacTIVision* se liga con alguna de las entradas de video conectadas a la computadora. Estas especificaciones del funcionamiento del programa son fácilmente configurables dentro del archivo denominado "camera.xml" ubicado en la carpeta donde se encuentra el ejecutable del programa (destacándose además que el programa de *ReacTIVision* es portable, por lo que no requiere ser instalado en el equipo). En este archivo se identificó que se puede modificar el ID de la cámara que se manda llamar al correr el programa. Cada cámara que se conecta a la computadora se encuentra ligada, en el sistema de administración de dispositivos, directamente con un ID para poder usarlas de forma independiente. Por lo que, al modificar este parámetro, se puede seleccionar entre las diferentes cámaras del sistema aquella que se desea utilizar en el programa.

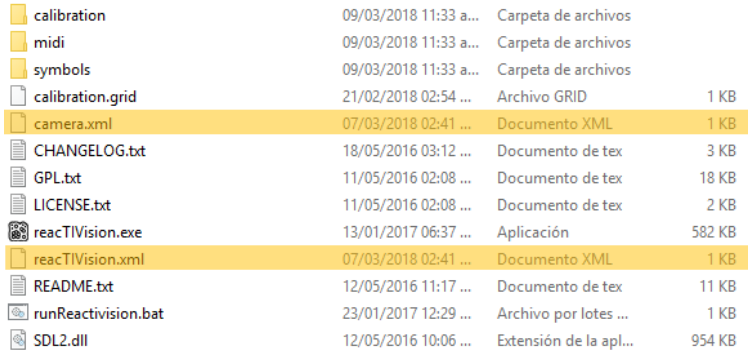


Figura 34. Archivos importantes para la configuración del target para ReacTIVision.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <portvideo>
  - <camera id="0">
      <capture compress="true" fps="max" height="480" width="640"/>
      <settings focus="min" gamma="min" sharpness="default" exposure="default"
        shutter="default" gain="199" contrast="default" brightness="default"/>
      <frame height="480" width="640" yoff="0" xoff="0"/>
    </camera>
</portvideo>
```

Figura 35. Contenido del archivo "camera.xml".

Para poder hacer la adquisición de datos desde ambas cámaras simultáneamente se duplicó la carpeta del programa de *ReacTIVision* y se realizaron las modificaciones para que uno se conectara con la cámara con el id=0 y el otro con la cámara con el id=1. Hasta este punto ya fue posible acceder a ambas cámaras, ahora, para poder identificar cuál es la fuente directa de la información que viene en los paquetes OSC es importante colocar una etiqueta a dicha información. *ReacTIVision* manda la información a partir del localhost de la computadora utilizando por default el puerto 3333 como medio para enviar datos con protocolo, este número de puerto es la información que se utilizará posteriormente durante el desempaquetamiento de datos y servirá como etiqueta para saber la fuente de los paquetes recibidos, de esta manera será posible identificar qué cámara es la que está enviando la información y se podrán unir ambas en el programa desarrollado en *C#*.

Para cambiar los parámetros que vienen por defecto en las configuraciones de *ReacTIVision*, con respecto al puerto que se ocupará para transmitir la información, se debe abrir el archivo "reacTIVision.xml" y modificar la variable *port* por el número del puerto que se desea utilizar. En este caso se seleccionó el puerto 3331 para una de las cámaras y se dejó el puerto que viene por defecto (3333) en la otra.

```
<reactivision>
        <!-- these are all the possible configuration tags and their d
        active tags need to be placed outside of this commented are
        size="12" sensitivity="100" /> <fiducial engine="amoeba"
        tile="10" threads="max" /> <calibration file="default.grid"
    <finger sensitivity="75" size="34"/>
    <image fullscreen="false" equalize="false" display="src"/>
    <threshold threads="max" tile="10" gradient="37"/>
    <calibration invert=" " file="default.grid"/>
    <tuio port="3333" host="127.0.0.1"/>
</reactivision>
```

Figura 36. Contenido del archivo "reacTIVision.xml".

Una vez que se han configurado independientemente las preferencias de *ReacTIVision* para cada cámara es posible ejecutar ambos programas simultáneamente y obtener información de cada una de ellas como se muestra en la figura 37.

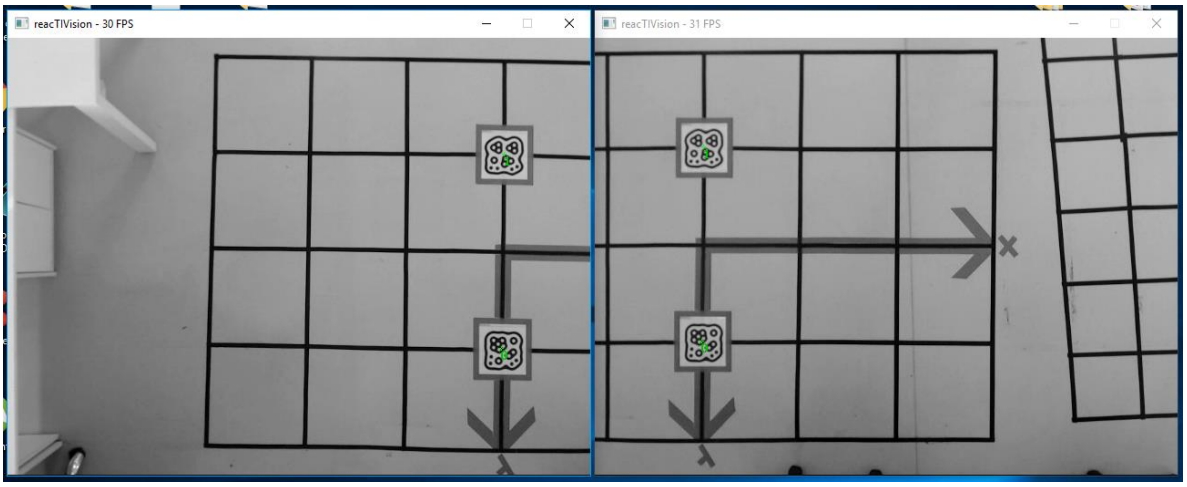


Figura 37. Dos ventanas abiertas al mismo tiempo recibiendo datos en paralelo por ReacTIVision.

De inicio el programa de *ReacTIVision* provee la información con respecto a las coordenadas relativas de las imágenes proporcionadas por las cámaras, esto es, las coordenadas son proporcionadas en pixeles comenzando desde la esquina superior izquierda con la coordenada (0,0) e incrementándose los valores de cada coordenada hacia la derecha y hacia abajo de la imagen respectivamente. Con esta configuración, tampoco se presentan valores negativos para las coordenadas. Pero para fines de este trabajo es necesario unir la información de manera que sea posible generar un sistema de coordenadas global utilizando la información de ambas cámaras, para lo que se utilizó el sistema de codificación *C#* para desempaquetar la información del formato OSC en protocolo TUIO y proceder a la unificación de coordenadas. Esto se realizó siguiendo el siguiente procedimiento conceptual:

1. Inicialización de las variables involucradas.

2. Inicialización de dos clientes TUIO en la función *main*. Uno debe escuchar al puerto 3333 y otro al puerto 3331; puertos que se configuraron en las preferencias de *ReacTIVision*. (Figura 38)

```
//GENERANDO 1 CLIENTE EN PORT 3333
if (client!=null) {
    client.addTuioListener(demo);
    client.connect();
    Console.WriteLine("listening to TUIO messages at port " + client.getPort());

} else Console.WriteLine("usage: java TuioDump [port]");

//GENERANDO 2 CLIENTE EN PORT 3331
if (client2 != null)
{
    client2.addTuioListener(demo);
    client2.connect();
    Console.WriteLine("listening to TUIO messages at port " + client2.getPort());

}
else Console.WriteLine("usage: java TuioDump [port]");
```

Figura 38. Código de inicialización de los clientes TUIO.

3. Cada uno de los clientes se quedan a la escucha para esperar la intervención de algún evento que tenga que ver un objeto TUIO entrante (*fiducial*).

4. Los eventos presentes en el cliente TUIO pueden ser los que se muestran a continuación, los cuales se encuentran referenciados a una correspondiente función dentro del programa que se indica entre paréntesis.

    a. *Adición de un nuevo fiducial*. (*addTuioObject*) Este caso se presenta cuando un nuevo elemento es añadido al entorno de trabajo y es detectado por las cámaras. La información que se manda por TUIO puede ser las coordenadas del elemento y las veces que ese elemento ha estado presente en el entorno desde que se inició el sistema *ReacTIVision*, además del número de *fiducial* asociado a la figura detectada. De esta manera se puede hacer un tratamiento individual de cada uno de los elementos dentro del entorno captado por las cámaras.

    b. *Actualización del elemento*. (*updateTuioObject*) Esta interrupción se presenta cuando algún elemento dentro del entorno cambia su posición o se mueve angularmente. Es posible saber cuál elemento se encuentra en movimiento y las nuevas coordenadas y posición angular.

    c. *Elemento borrado (delete)*. (*removeTuioObject*) Esto pasa cuando un elemento se pierde del entorno de trabajo y sólo es posible obtener las últimas coordenadas que se captaron justo antes de su salida del entorno de visualización.

5. De detectarse en cualquiera de los dos puertos (cámaras) los objetos 1 o 0, se procede a entrar en una rutina de calibración, ya que estos dos *fiducials* se utilizarán únicamente como referencia del sistema coordenado y para generar el sistema global de referencia con el que se unirán ambas cámaras. En dicha rutina de calibración se detecta la distancia que existe entre ambos *fiducials* en pixeles (unidad básica en *ReacTIVision*) para que posteriormente, con una medida estándar entre ellos, se pueda calcular una constante de proporcionalidad entre pixeles y unidades de distancia en el espacio de trabajo, relacionando esta constante con la cámara o puerto que hizo la detección. Esto mismo se realiza con las dos cámaras de modo que ambas se encuentren relacionadas con una constante de proporcionalidad específica.

6. De detectarse un *fiducial* distinto a los de referencia (0, 1) entonces se procede a leer sus coordenadas locales de las cámaras y referenciar dichas coordenadas al centro de la cámara, de modo que las coordenadas se encuentren referidas en pixeles a su centro y no a una esquina. Finalmente, se traducen las coordenadas del elemento de pixeles a unidades de distancia utilizando la constante de proporcionalidad correspondiente para el puerto o cámara que realizó la medición.

7. Como último paso, para pasar las coordenadas locales de cada cámara a coordenadas globales es necesario ubicar el punto de origen del sistema global de coordenadas. Este punto se seleccionó como el punto medio entre ambos *fiducials* de referencia (0 y 1), estableciéndose como eje coordenado y la línea que va de la posición del fiducial de referencia 1 a la posición del fiducial de referencia 0, y como eje coordenado x la línea

perpendicular que pasa por el origen. Como ambas cámaras se colocan de tal manera que pueden detectar los *fiducials* 0 y 1, ambas podrán calcular las coordenadas del centro de su imagen con respecto al origen global. Con dicha información es sencillo hacer una traslación del sistema de coordenadas locales de cada cámara al sistema global con la ecuación 30.

$$CoordGlobal = coordenda\ del\ centro\ de\ la\ cámara + coordenada\ local\ del\ fiducial \quad (30)$$

8. Adicionalmente, considerando la posibilidad de que alguna de las cámaras se encuentre rotada ligeramente con respecto de la otra, se incluyó una rotación de coordenadas que corrigiera esta situación. Si para la cámara en cuestión se detecta que la línea formada por los *fiducials* de referencia se encuentra rotada con respecto a la vertical de la imagen, todas las coordenadas de esta cámara se rotarán éste mismo ángulo antes de ser trasladadas, manteniendo así la misma referencia de direcciones de los ejes coordenados para las coordenadas globales y locales, y poder realizar así la transformación de coordenadas correctamente.

9. En este programa se incluyó también un ajuste dependiendo de la altura de los agentes. Debido a que los *fiducials* correspondientes a los agentes no se encuentran sobre el mismo plano de la superficie de trabajo (donde se ubican los *fiducials* de referencia), sino que se ubican sobre cada uno de los robots que intervienen en el proceso, estos estarán a una altura específica con respecto al piso. Por la forma de visión de las cámaras, la lectura de la ubicación de estos *fiducials* puede verse afectada, desplazándose su ubicación un poco hacia afuera. Para compensar esto, se realizó el ajuste a las coordenadas mediante el uso de trigonometría, considerando el comportamiento de visión de la cámara como una pirámide rectangular como se aprecia en la figura 41.

En la figura 40 se muestra esquemáticamente el campo de visión de las cámaras y la ubicación de los *fiducials* de referencia que se realizó para la unificación del sistema de coordenadas globales y para poder unir así la información de ambas.

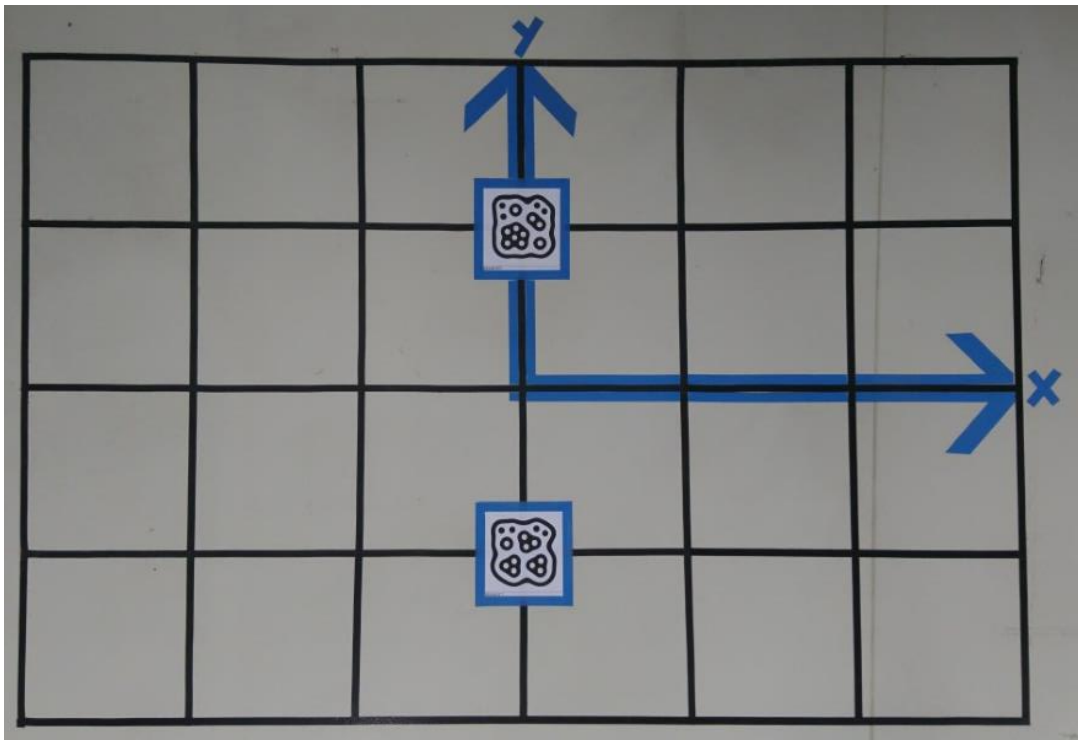


Figura 39. Espacio de trabajo real con fiducials de referencia. Cada cuadro mide 400 x 400 [mm].

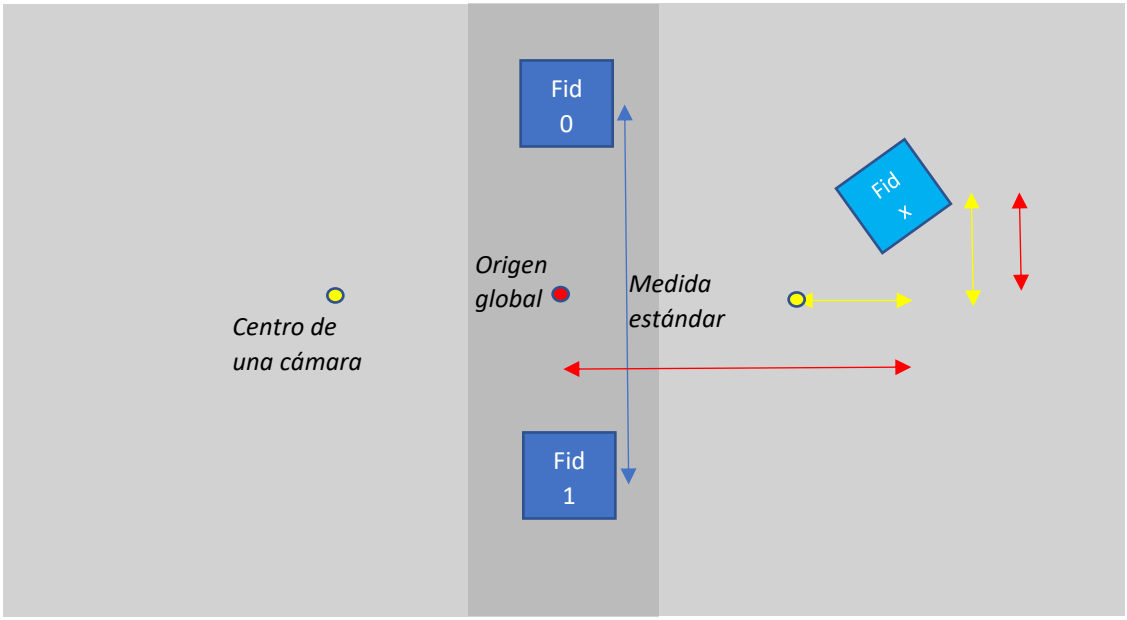


Figura 40. Descripción esquemática del sistema de calibración de coordenadas globales (Fiducial de estudio en color cian), (fiducials de referencia en azul 0 y 1). Las coordenadas locales se marcan en amarillo mientras las globales en rojo.

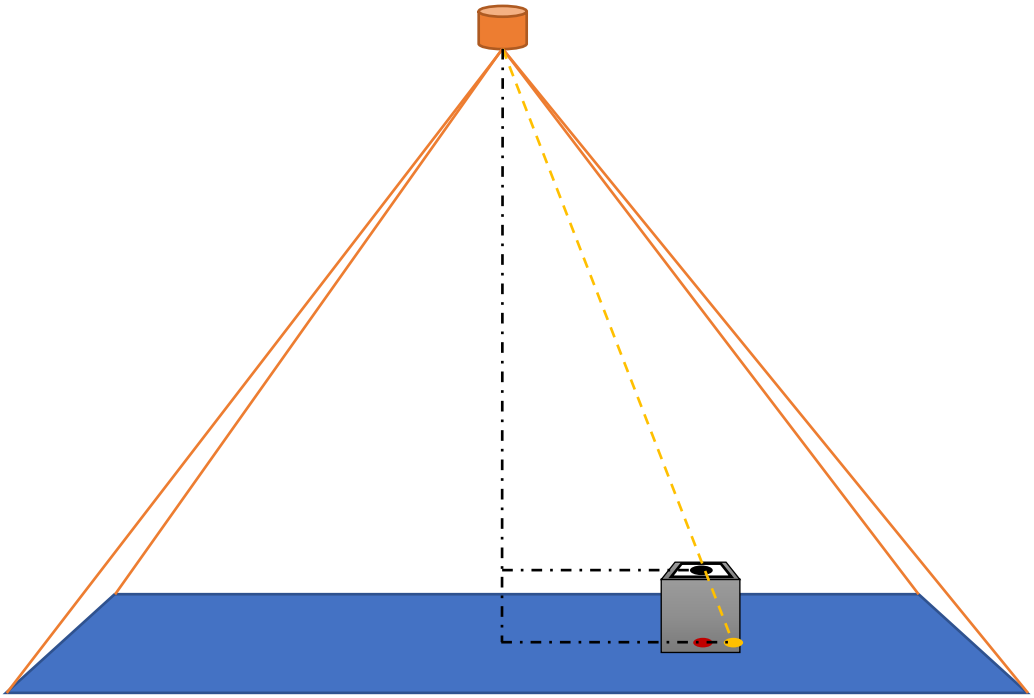


Figura 41. Representación de la visión de las cámaras para el ajuste en función de la altura del fiducial. El punto rojo representa la posición real del agente en el plano, mientras que el punto amarillo la posición vista por la cámara.

Hablando sobre las complicaciones referentes a la implementación de este sistema, es de destacar los conflictos que hubo por parte de los drivers de las cámaras. Al estar utilizando dos cámaras de iguales características en muchas ocasiones el *driver* o controlador de éstas captaba que una se había conectado pero la otra quedaba sin posibilidad de ajustes. Por lo tanto, sólo era posible modificar las características de brillo, nitidez y color de una de ellas, pero no de las dos. También por esta razón, en muchas ocasiones no había una buena conexión a la computadora por parte de las cámaras lo cual ocasionaba conflictos de imagen o que *ReacTIVision* fuera incapaz de resolver la información de alguna de ellas. Durante el desarrollo de las pruebas se encontró que al desinstalar los controladores desde el administrador de dispositivos era posible utilizar los controladores originales de versiones anteriores de la computadora para utilizar las cámaras sin conflictos de software y poder calibrar la calidad de imagen independientemente.

Otro problema importante al llegar a este punto fue el hecho de que el sistema de coordenadas se invertía al momento de hacer pruebas aparentemente sin ninguna razón. Dicho problema fue solucionado al hacer pruebas con las posiciones de los *fiducials* de referencia y observar que, si se encontraban desfasados en equis de manera que el *fiducial* 0 quedara más a la derecha, es cuando se ocasionaba dicho error (figura 42).

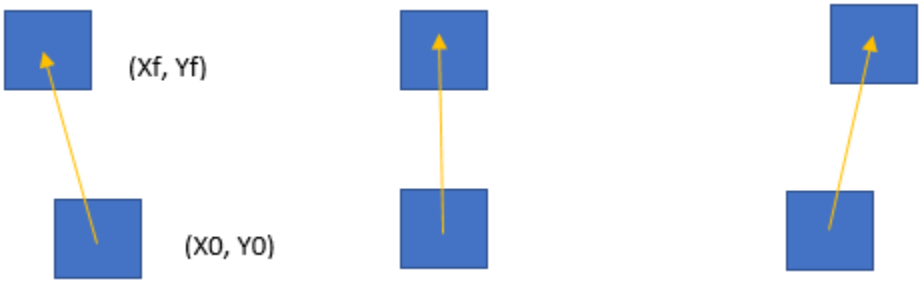


Figura 42. (izquierda) Marcaba adecuadamente las coordenadas. (Derecho) Invertía las coordenadas. (Centro) Punto crítico.

Se identificó que esto era ocasionado por la ecuación para encontrar la inclinación del vector mencionado anteriormente debido al uso de la función arco tangente, ya que al estar el *fiducial* que marca el fin del vector ligeramente desfasado a la derecha provocaba que el valor de X resultase mayor que el que marca el inicio del vector y por lo tanto, al restarse, resulta en un número negativo provocando errores de posicionamiento de la función arco tangente normal y un resultado de ángulo negativo. Esto provocaba una inversión del sistema coordenado al incluirse en las ecuaciones de rotación.

$$\emptyset = angtan\left(\frac{y_f - y_0}{x_f - x_0}\right) \qquad (31)$$

Siendo los subíndices f los finales del vector y los subíndices o las coordenadas de inicio del vector, y $\emptyset$ el ángulo de inclinación de los *fiducials*.

Para solucionar el problema se decidió ocupar la función arco tangente 2 que viene incluida en la clase *Math* de *C#* como atan2. Esta función elimina los problemas de ángulo negativo ya que sus argumentos son las coordenadas de fin del vector considerando que parte del origen y es capaz de detectar los ángulos en los cuatro cuadrantes coordenados; con ello fue posible detectar un ángulo adecuado para realizar la rotación del sistema coordenado sin errores de inversión.

Tras los ajustes realizados para el uso simultáneo de dos cámaras se logró una ampliación considerable del espacio de trabajo. Con el uso de una sola cámara con *ReacTIVision* se tenía un espacio de trabajo de 4.284 [m^2] (238 [cm] x 180[cm]), mientras que utilizando dos cámaras con el sistema desarrollado se logró generar un espacio de trabajo de 7.182 [m^2], esto implica un aumento del 66.39% en comparación con el uso de una sola cámara.

El código editado para la unión de ambas cámaras y la traducción a un sistema global de coordenadas implementado en C# se muestra en el Anexo 3 de este trabajo.

6.2.3 Transmisión de la información a *Simulink* y su procesamiento

Una vez solucionados los problemas referentes a la detección y tratamiento de los datos en C# para identificar y localizar a los *fiducials* dentro de un entorno global, fue necesario encontrar la manera de transmitir estos datos. Para dicho propósito se basó el trabajo en lo reportado por Hernández, I. [44], quien en su tesis de licenciatura propone el uso del protocolo UDP (*User Datagram Protocol*) como medio para enviar dicha información. Este protocolo se caracteriza por ser muy sencillo para

el envío de datagramas, ya que no requiere de una conexión previa y carece de control de errores, por lo que no realiza la retransmisión de paquetes perdidos, no realiza revisión del orden en que llegan los paquetes ni si hay duplicidad de estos. Esto implica que no se tiene garantía de la recepción de todos los paquetes y que cualquier corrección de errores u ordenamiento de paquetes deberá ser llevado a cabo posteriormente, sin embargo, resulta útil en aplicaciones de tiempo real que requieren un envío continuo de información donde resulta de mayor importancia la velocidad de envío de datos que la fiabilidad en la recepción de todos los paquetes de información. Al utilizar este protocolo es importante considerar que es posible que ciertos paquetes de datos no se reciban en orden o completos.

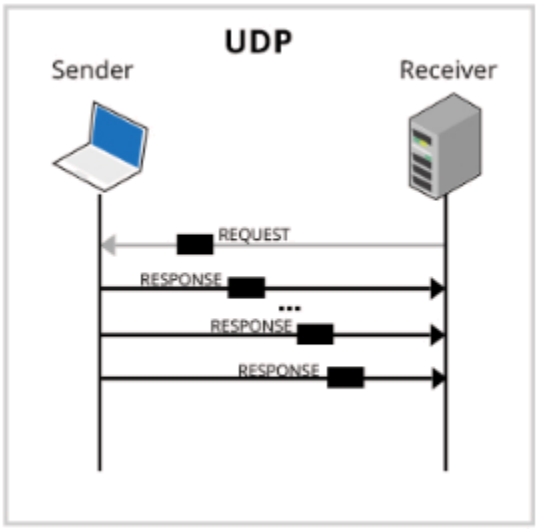


Figura 43. Diagrama simplificado para el procedimiento de envío de datos mediante UDP (Imagen de Oodles Technologies).

Por ello, es necesario implementar un sistema que ordene y acondicione los datos recibidos para poder ser leídos en orden y sea suficientemente robusto para evitar la pérdida de información, o en dado caso, considerar esta posibilidad. Hablando de cuestiones más específicas sobre la transmisión, es importante mencionar, que la trasmisión realizada mediante el protocolo de datagramas se hará con base en una cadena de caracteres, por lo cual, es necesario que el *Middle Worker* (en el caso del presente trabajo *C#*) traduzca toda la información a una cadena de caracteres de un tamaño estandarizado de modo que su decodificación resulte posible. Para ello se decidió estandarizar el envío de la información como se explica en la figura 44.

Figura 44. Estructura de la cadena de datos para la transmisión de la información.

Esta cadena, siguiendo la estructura mencionada, es necesario que tenga una longitud constante en todo momento de la transmisión, para lo cual se tuvieron las siguientes consideraciones:

1. Para aprovechar la resolución que da el sistema de visión (aproximadamente 3 mm y 3° para coordenadas lineales y posición angular respecto al eje positivo X respectivamente) se considera que los números componentes de la sección CoordX, CoordY y Angulo serán de 4 dígitos en milímetros (para tener la posibilidad de miles de milímetros y abarcar el espacio de trabajo en su totalidad) más un dato más de signo. Por lo anterior se considera que cada uno de estos componentes de la cadena deberá componerse de 5 caracteres.

2. Debido a la posibilidad de que la cadena, dadas las características del protocolo UDP, se adelante o atrase durante la transmisión con la consecuencia de haber malinterpretaciones en el procesamiento de datos, se consideró que es de suma importancia colocar al inicio un byte de información compuesto por un carácter estándar que se considere como un elemento de inicio de cadena para que se pueda ordenar y realizar un procesamiento adecuado. A esto, por lo tanto, se añade un carácter más

3. Posteriormente, debido a que el objetivo del programa desarrollado en el *Middle Worker* de C# es que sea capaz de captar y transmitir la información relevante a la posición espacial de los agentes involucrados en el espacio de trabajo, se colocó al final de la cadena de transmisión una etiqueta que sirve para identificar a cuál de los agentes dentro del espacio de trabajo se está refiriendo la información contenida en el datagrama.

4. Hay dos consideraciones muy importantes a tomar en cuenta, dentro del mismo espacio de trabajo (entendiendo como espacio de trabajo el área que abarcan las dos cámaras) no es posible colocar dos *fiducials* idénticos. En otras palabras, no es posible colocar dos agentes que tengan el mismo identificador, esto debido a que, de ser el caso, habría problemas con el algoritmo y podría haber confusión sobre la información contenida en el datagrama, concluyendo esto en errores de procesamiento. Lo segundo es que, considerando la información y las longitudes referidas anteriormente, la longitud total sería de 17 bytes o caracteres alfanuméricos en código ASCII que serán trasmitidos mediante UDP al sistema de procesamiento y análisis de *Simulink* desarrollado a continuación.

5. Debido a que la recepción de datos del lado de *Simulink* hace únicamente la lectura del número de bytes establecidos, sin detectar específicamente que esta lectura comience en el mismo carácter de inicio de la cadena transmitida, es posible que al momento de leer los datos se reciba la mitad de la cadena de un envío de información y la otra mitad del envío siguiente, resultando una mezcla de dos paquetes de información distintos. Para solucionar esto se decidió hacer el envío y recepción de dos cadenas completas iguales cada vez, para garantizar de esta manera que en cada lectura se tiene una cadena de información completa y correspondiente al mismo envío, independientemente de dónde inicie dicha lectura. Por lo tanto, el paquete de información enviado finalmente es una cadena de 34 bytes (dos cadenas de 17 caracteres cada una con la configuración mostrada en la figura 44).

La metodología implementada para la estandarización de la longitud de cadena a enviar y el proceso para el envío de información es el que se muestra a continuación (el código completo se puede encontrar en el anexo 3).

1. **Inicialización**

2. **Disparo del evento de adición de agente (*updateTuioObject*)**
    a. Si el *fiducial* detectado es alguno de referencia (0 y 1 para el algoritmo)
        i. Identificar distancia en pixeles.
        ii. Obtener la transformación de pixeles a milímetros con una distancia estándar.
        iii. Almacenar la información de la relación entre pixeles y milímetros, así como el ángulo de rotación de ejes para la transformación de coordenadas.
    b. En otro caso.
        i. Identificar el puerto (cámara) de donde viene el disparo.
        ii. Ejecutar las calibraciones de acuerdo con el puerto en cuestión:
            1. Calibración por altura de *fiducial*.
            2. Calibración para transformar coordenadas locales a coordenadas globales.
        iii. Asignar las variables de coordenadas en X, coordenadas en Y, ángulo del agente.

3. **Construir la cadena de datos.**
    a. Tomar cada una de las variables de coordenadas X, Y o ángulo.
    b. Observar cuál es el rango de valores en los que cae la variable:
        i. Entre 0 y 10, entre 10 y 100, entre 100 y 1000.
        ii. Entre 0 y -10, entre -10 y -100, entre -100 y -1000.
        iii. Dependiendo de cuál es el rango de valores en el que la variable cae, se añaden los ceros correspondientes para que, considerando el signo de la variable, el tamaño de ésta quede en 5 caracteres.
    c. Finalmente, se concatenan las variables en formato de cadena para generar la cadena final a enviar con la estructura expuesta en la figura 44.

4. **Apertura de la comunicación (función *Send*).**
    a. Declarar un objeto Socket.
    b. Declarar una dirección IP (local host o una computadora adherida a la misma red).
    c. Declarar un puerto no utilizado en la IP.
    d. Ajustar los parámetros del protocolo UDP en el Socket (funciones internas en *C#*).
    e. Tomar la cadena a mandar y proceder a enviarla a través del Socket ya configurado.

Una vez que se ha realizado con éxito la concatenación, codificación y envío de los datos mediante UDP, la manera en la que *Simulink* recibe dicha información es a través de un bloque configurable para la recepción de UDP´s. Dicho bloque se encuentra dentro de las opciones de *Real-Time Simulation*, por lo cual es necesaria la correcta instalación del *kernel* adecuado para dichas tareas. El *kernel* de *Simulink* que ayuda a la codificación del programa y su depuración para poder ser corrido en tiempo real es el *sldrtkernel*, el cual es instalado desde la ventana de comandos de Matlab mediante la instrucción: ***sldrtkernel -install.***

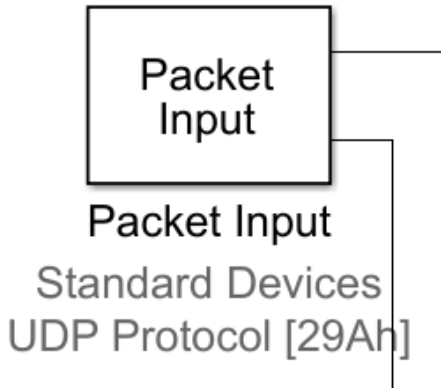


Figura 45. Bloque de recepción de datos por UDP.

El bloque *Packet* Input se utilizó para la recepción de los datagramas provenientes del *Middle Worker* (*C#*) y la configuración de este se realizó con base en los trabajos realizados con la misma metodología (UDP) dentro del *Mechatronics Research Group*. Uno de los parámetros más importantes para esta metodología es el tiempo de muestreo de dicho bloque, o sea, el tiempo que tardará en revisar si dentro del buffer de información recibida se encuentra algún datagrama.

$$T_{muestreoUDP} = 0.02\ [s]$$

Dicho tiempo de muestreo se ha visto experimentalmente que es el intervalo de tiempo entre cada lectura de datos desde el programa de *Simulink*. También este bloque permite la limitación de tick´s o bytes perdidos durante el traspaso de datos. Para fines de este trabajo, los tick´s perdidos no serán de mayor importancia debido a que, si se llega a perder alguno, el sistema simplemente continuará trabajando con el último dato recibido sin ningún inconveniente práctico. También, hablando sobre las configuraciones del bloque de recepción de datos, es importante configurar la cantidad de bytes (caracteres) a recibir y tener en cuenta que dichos números son la representación en ASCII de los números desplegados en el procesamiento de imágenes de C# y procesamiento de datos.

Por otro lado, para poder configurar los atributos de la conexión para la recepción de datos, es necesaria la instalación de una tarjeta de adquisición de datos para la conexión de *Simulink* al puerto

de salida donde se instaló el Socket en el paso anterior de C#. La configuración de dicha tarjeta se hace directamente en el cuadro de diálogo de configuraciones del Packet Input (figura 46) en el botón que refiere "*Board setup*". Al hacer clic en dicho botón se despliega otro cuadro de diálogo como se muestra en la figura 47 en el cual se configura la IP y el puerto que se establecieron previamente en la codificación de C#. En este caso, se está utilizando el localhost (la misma computadora como nodo) y el puerto 2019 para la transmisión de la información.

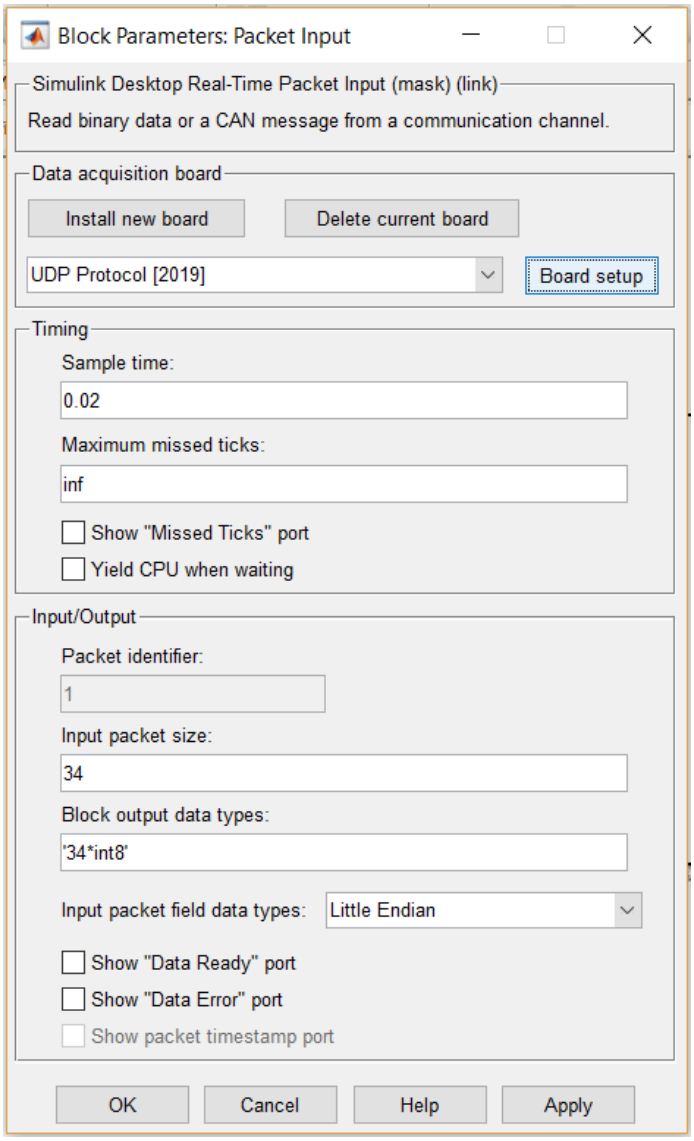


Figura 46. Configuraciones completas del Bloque de Packet Input de UDP.

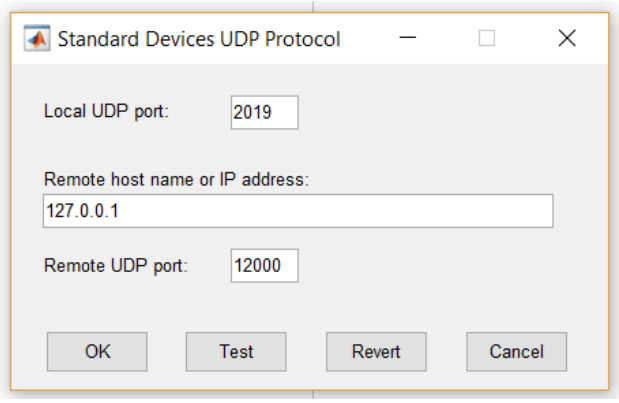


Figura 47. Cuadro de diálogo para la configuración de red para la recepción de paquetes de datos en Simulink.

También es posible, dentro del mismo cuadro de diálogo para la configuración de red, realizar una prueba de conectividad con el botón de prueba, mediante el cual, es posible confirmar que el *kernel* de tiempo real se encuentre instalado en Matlab, así como también verificar que el puerto y la IP son correctas y se encuentran disponibles para realizar la comunicación.

6.2.4 Decodificación de la cadena de información

Una vez recibida la cadena de caracteres es importante realizar un procesamiento más antes de obtener datos que se puedan usar para el procesamiento en *Simulink*, debido a que el protocolo de envío de datos mediante UDP, por su naturaleza, puede tender a que las cadenas se detecten desincronizadas y que los caracteres, por ende, no se lean en el orden adecuado. Para resolver este problema se colocó un carácter adicional 'M' que es útil para identificar cuando una cadena termina y empieza la siguiente.

El procedimiento se marca a continuación:

1. Se leen 34 bytes de información del Socket declarado. Dicha cadena de datos se guarda en una variable, la cual aún no se puede garantizar que esté en orden.
2. Se revisan cada uno de los elementos de la cadena hasta encontrar el carácter que refleja el inicio de una cadena. Una vez que se identifica este carácter, que en este caso se trata de la letra "M" o 77 en código ASCII, se guarda en una variable auxiliar el lugar dentro de la cadena en el que se encontró. Por ejemplo:
   De leerse la siguiente cadena de caracteres:
   '1' + '2' + '3' + '4' + 'M' + '0' + '0' + '1' + '2' + '3' + '-' + '0' + '1' + '2' + '3' + '+' + '0' +
   '1' + '2' + '3' + '4' + 'M' + '0' + '0' + '1' + '2' + '3' + '-' + '0' + '1' + '2' + '3' + '+' + '0'
   Entonces dicha variable auxiliar tomaría el valor de 5, dentro de los 34 lugares que conforman dicho ejemplo.
3. Esa variable auxiliar servirá como referencia debido a que ahora se realizará un nuevo arreglo de datos asignando el primer espacio al carácter inmediato siguiente al carácter de referencia de inicio de cadena, es decir, se irán asignando a una nueva cadena de caracteres los valores siguientes a la 'M' hasta terminar con la cadena de 17 bytes que contiene la

información completa de los datos de un agente. Después de este proceso la nueva variable se vería de la siguiente forma:

E = *'0' + '0' + '1' + '2' + '3' + '-' + '0' + '1' + '2' + '3' + '+' + '0' + '1' + '2' + '3' + '4' + 'M'*

Siguiendo con el ejemplo planteado.

4. Finalmente, se tiene el arreglo final ordenado con la configuración mencionada en la figura 44, con la única diferencia de que el carácter de referencia 'M' ahora se encuentra en el último sitio de la cadena. Se procede a separar la información de cada uno de los datos (x, y, ángulo) en diferentes variables.
5. También es importante mencionar que, debido a que todos los caracteres se transmiten en codificación ASCII, es necesario restar 48 unidades a cada uno de ellos para obtener así su valor numérico que pueda ser utilizado por el sistema, de igual manera para el caso de los números negativos se debe convertir aquellos con el símbolo '-' en valores negativos.

Una vez decodificada y ordenada la información de la cadena proveniente de la comunicación UDP es posible, mediante una función definida por el usuario dentro del entorno de *Simulink*, sacar el vector de coordenadas y también el dato sobre el *fiducial* al que corresponden dichas coordenadas, de manera que, posteriormente, utilizando una serie de bloques de *Switch* se pueda asignar el valor de las coordenadas al *fiducial* correspondiente.

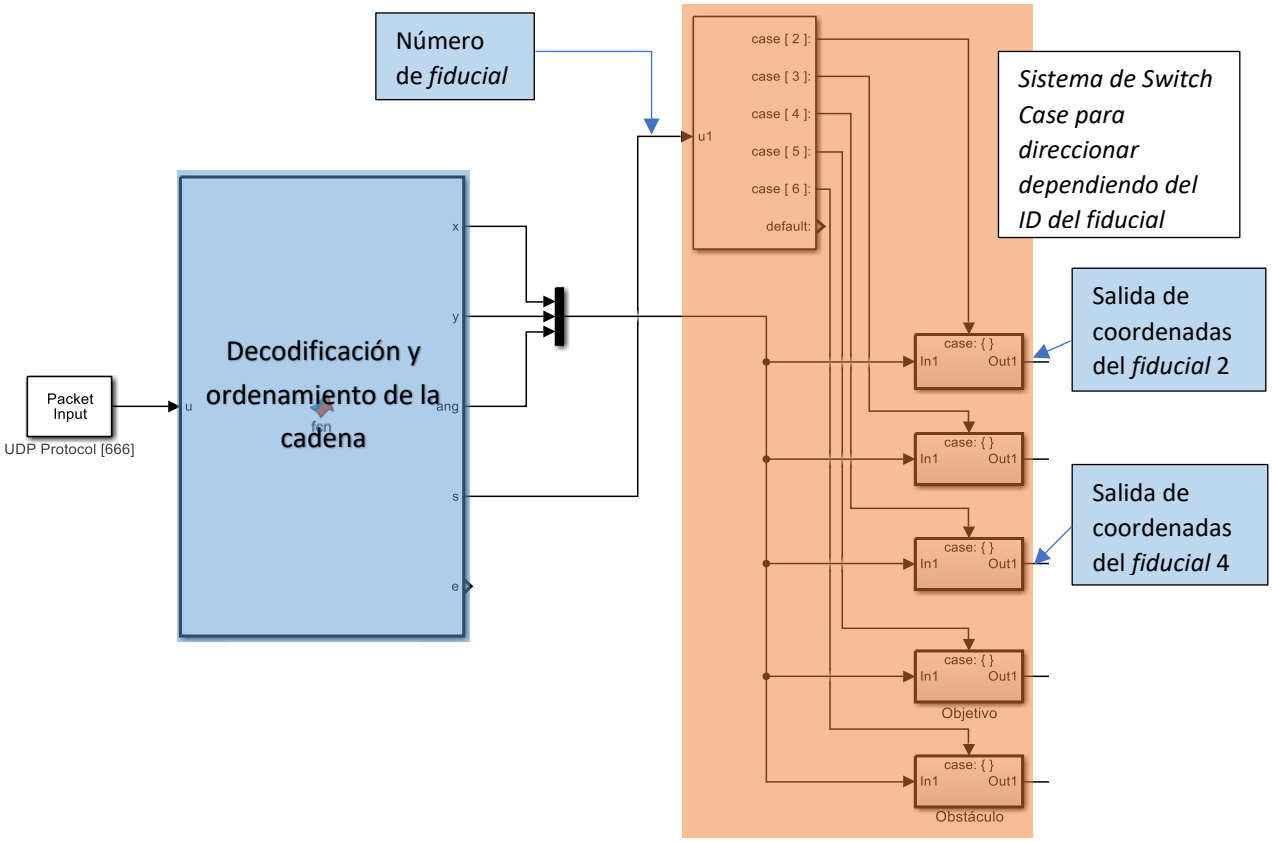


Figura 48. Sistema completo de recepción, ordenamiento, decodificación y distribución de la información contenida en paquetes UDP provenientes de C# a 17 bytes de longitud. Adaptable a cualquier fiducial y cantidad de agentes en el entorno de trabajo.

Los códigos referentes a cada uno de los pasos anteriormente citados se encuentran en anexos para referencias más específicas sobre cómo se desarrolló cada uno de ellos para el análisis conceptual realizado en los puntos anteriores.

Con el desarrollo de este sistema es posible escalar fácilmente las pruebas a realizar dado que ya no es necesario codificar desde la fuente para añadir más agentes al entorno de trabajo, sino que ahora, simplemente basta con agregar una casilla de *Case* al programa de *Simulink* para que el sistema sea capaz de detectar la inclusión de cualquier *fiducial* en el entorno de trabajo y canalizar la información, de manera que sus coordenadas puedan ser utilizadas para declarar la posición de un obstáculo, agente u otro ente que se requiera para un sinfín de posibilidades de experimentación.

6.2.5 Utilización y desventajas de la información ya decodificada

Una vez que se ha realizado el procedimiento de desempaquetamiento de la información, el bloque de ordenamiento y procesamiento de datos se encarga de dividir dicha información en tres factores clave para cada *fiducial* direccionado por el bloque de control de flujo de datos "*switch case*". Dicha información es:

1. Coordenada en X en el sistema de coordenadas global del *fiducial* leído.
2. Coordenada en Y en el sistema de coordenadas global del *fiducial* leído.
3. Ángulo respecto al eje positivo X del *fiducial* en cuestión.

Dicha información se puede obtener del bloque descrito anteriormente independientemente de cuáles sean los *fiducials* implementados en el espacio de trabajo, por lo que, aun cuando un *fiducial* no esté contemplado en los *fiducials* en uso por el sistema, si éste se encuentra dentro del espacio de trabajo su información será procesada de igual manera que el resto. Si bien esto no ocasionaría conflictos al sistema, ya que únicamente sería ignorada la información mientras no se indique que se trata de otro agente, se estaría llevando a cabo el desempaquetamiento de los datos, lo cual gastaría recursos innecesariamente y podrían perderse otros paquetes de información más relevantes durante este proceso. Por ello se evitará colocar *fiducials* adicionales dentro del área de visión.

Por otro lado, como se vio en las secciones 5.2 y 5.3, la información más trascendente para poder aplicar la metodología propuesta en dichos apartados es la localización y ángulo en el que se encuentra el agente. Así, por medio de la aplicación del comportamiento vectorial resultado del campo potencial propuesto y el sistema de *flocking* local por agente, se puede mandar la información suficiente al sistema robótico para indicar tanto la velocidad de avance necesaria del robot y la velocidad angular para alcanzar las condiciones indicadas a partir del sistema de procesamiento vectorial, haciendo uso de la cinemática inversa del robot.

Haciendo un recuento sobre la información contenida en los últimos capítulos de este trabajo, es útil recordar la figura 49, en la cual se explican los sistemas conceptuales para el diseño del banco de pruebas. En dicha figura se puede apreciar como en el recuadro gris transparente se encuentran los sistemas que se han resuelto a lo largo de este trabajo. Desde el sistema de visión utilizando los protocolos TUIO auxiliándose del sistema de visión *Open Source* de *ReacTIVision*, pasando por el procesamiento vectorial para obtener velocidad de avance y velocidad angular del agente calculadas

a partir del método planteado en el capítulo del modelo teórico (5.2 y 5.3) y, finalmente, analizando las necesidades de velocidades para obtener la velocidad de cada una de las llantas gracias a la cinemática inversa del robot móvil diferencial, abriendo la posibilidad para aplicar el método a cualquier configuración robótica simplemente descomponiendo el vector de velocidad en dos más que signifiquen velocidad en X e Y necesarias para el control de este tipo de robots.

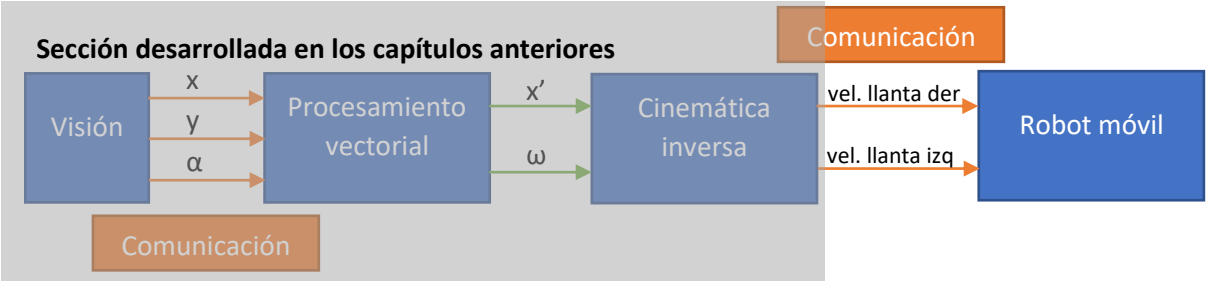


Figura 49. Ejemplificación en el avance del desarrollo del banco de pruebas.

6.2.6 Transmisión de la información al agente y materiales ocupados

Después de haber calculado las velocidades de cada una de las llantas de los agentes, considerando ya la cinemática particular de cada uno de ellos, se debe realizar una transmisión de dicha información para poder mover físicamente el sistema. Para la selección de la tecnología a implementar se realizaron las siguientes consideraciones:

1. Se debe contar con una tecnología de recepción y control de motores que se encuentre a bordo del robot móvil, dado que tener dicha tecnología afuera e implicaría una limitación a la movilidad del robot.
2. La transmisión de los datos debe ser inalámbrica para tener la capacidad de controlar al robot sin la necesidad de mantener una conexión física con el mismo.
3. La traducción de la información entre la llegada de la información hasta la motorización debe ser lo más simple posible y el código para mover los motores debe ser igualmente sencillo para que el robot móvil en estudio se mueva lo más inmediatamente posible una vez que llega un dato al agente.

Debido a estas consideraciones se decidió implementar, al igual que en el traslado de información entre *ReacTIVision* y *Matlab*, pasando por *C#*, el protocolo UDP para comunicarse con el robot. Dicho protocolo se asocia en *Simulink* con un bloque de salida de información (*Packet Output*), haciendo énfasis en que dicho bloque lo que hace es recibir dos datos, que por facilidad se convierten a enteros desde antes de mandarlos, y los transforma en un paquete para el protocolo UDP, lo cual ahorra el trabajo de procesar la información antes de mandarla, como era necesario hacerlo en el caso del *Middle Worker (C#).* En dicho bloque se debe configurar, siguiendo el mismo procedimiento que en el caso del bloque de entrada de datos, la IP y el puerto por el cual se mandará la información. Dichas configuraciones dependen, en esta ocasión, del dispositivo al cuál se quiere transmitir los datos de motorización para el robot móvil.

Como se ha hecho en diversas tesis dentro del *Mecathronics Research Group*, se decidió implementar el dispositivo *ESPino* por simpleza y debido a la versatilidad que tiene por estar dotado con un microcontrolador ESP8266 y con ello, tener la capacidad de comunicación I^2C. Las características técnicas de este componente son las mostradas en la figura 50.

Resumen

Microcontrolador	ESP8266 (32-bit RISC)
Comunicación	WiFi 802.11 (station, access point, P2P)
Voltaje de operación	3.3V
Voltaje de entrada	4.4-15V
Pines de I/O Digitales	9
Pines de entrada analógica	1 (10-bit ADC)
Corriente DC por Pin I/O	12 mA
Corriente DC máxima para el Pin 3.3V	800 mA
Memoria Flash (Programa)	4 MB
Instruction RAM	64 KB
Data RAM	96 KB
Boot ROM	64 KB
Velocidad de reloj	80 Mhz

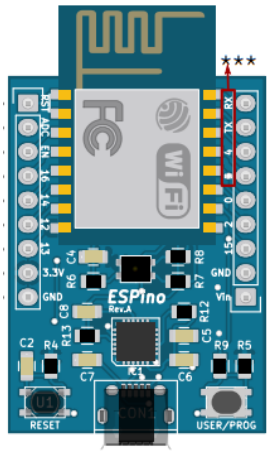


Figura 50. Características técnicas del sistema de recepción de datos de motorización del robot. (AG Electrónica "ESPino - Especificaciones")

Una de las principales características que satisface este dispositivo es una conexión inalámbrica estable en comparación con el Bluetooth, la cual carece de confiabilidad en la interacción entre el agente y la computadora, además de tener un alcance máximo de 30 metros en comparación con el alcance de hasta 300 metros con conexión *WiFi* estándar y, por último, un ancho de banda superior (24 Mbps del Bluetooth contra hasta 1 Gbps con comunicación *WiFi*). La velocidad de transmisión también es una característica importante para este proyecto debido a que la computadora estará leyendo y transmitiendo constantemente debido a la ejecución del sistema en tiempo real de *Simulink*.

Aunado a esta tecnología, también se rescata de varias tesis el controlador de motores MD25 el cual es un controlador de motores de doble puente H desarrollado para el control de dos motores de corriente directa EMG30. Dicho controlador goza de las siguientes características técnicas:

- Alimentación con voltaje único a 12 V.
- Regulador a 5 V para alimentar la circuitería externa.
- Suministro de hasta 3 A para cada motor.
- Comunicación I^2C para la recepción de las velocidades a los motores.
- Control de potencia y aceleración.

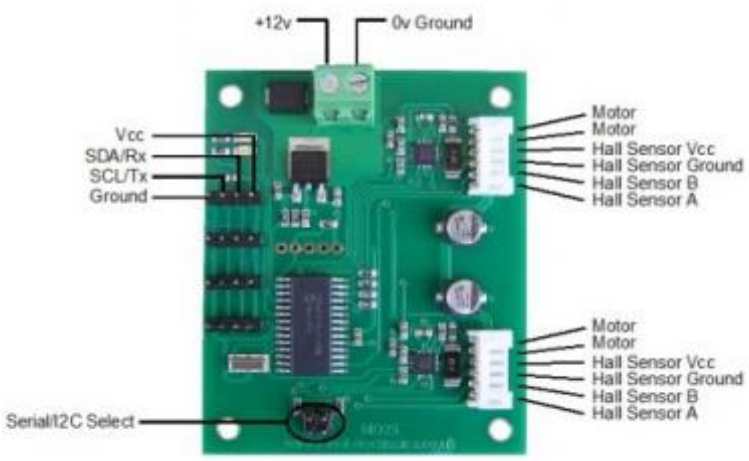


Figura 51. Pinout del circuito MD25 para controlar dos motores EMG30. [37]

Esta tarjeta tiene la ventaja de poder ser alimentada con una fuente externa para suministrar la corriente suficiente sin dañarse, utilizando los motores para la que fue diseñada (EMG30). El controlador puede escribir una velocidad a cada uno de los motores que controla de manera independiente en forma de un PWM en el puente H. Dicho PWM puede variar en un rango de 0 a 255 aunque, en este caso, el 0 no significa detener el motor, sino que, para detener el motor es necesario mandar un 128 (la mitad del intervalo), mientras que del intervalo de 0 a 127 va en una dirección y del intervalo de 129 a 255 cambia de dirección incrementando su velocidad. El autor Dorantes E. [37], en su tesis de maestría, menciona que los motores son capaces de alcanzar magnitudes de -20 rad/s hasta 20 rad/s. Con lo cual, realizando una relación lineal entre la variable de velocidad angular de las llantas y valor en PWM para el microcontrolador, el PWM adecuado para cada velocidad angular queda expresado de la siguiente manera [37]:

$$PWM(\omega) = 6.408 * \omega + 127.66 \qquad (32)$$

Adicionalmente, en el modo de funcionamiento implementado, este controlador de motores incluye un sistema de regulación automática de velocidad que es capaz de incrementar la potencia para lograr la velocidad establecida. Para conseguir esto hace uso de la lectura de *encoders* que sirven para registrar la velocidad del motor y poder realizar los ajustes correspondientes. Esta función permite controlar los motores más eficientemente en correspondencia con su velocidad.

También, al utilizar este controlador de motores es recomendable, según el fabricante, el uso de los motores para los que fue diseñado, los motores de corriente directa EMG30 con las siguientes características técnicas:

Voltaje nominal	**12 [V]**
Torque nominal	1.5 [Kg/cm]
Velocidad nominal	170 [rpm]
Corriente nominal	530 [mA]
Velocidad sin carga	216 [rpm]
Corriente sin carga	150 [mA]
Corriente a motor bloqueado	2.5 [A]
Potencia nominal	4.22[W]
Pulsos por vuelta del encoder	360
Velocidad máxima sin carga a 12[V]	200 rpm (20rad/s)

Figura 52. Características de los motores DC utilizados en el sistema robótico. (Robot Electronics)

Se necesitan, por lo tanto, dos motores de estas características instalados en el agente robótico para poder conformar el sistema del robot diferencial. Con lo cual, el sistema de manera nominal estaría consumiendo un total de aproximadamente 1 a 1.5 [A] dependiendo del estado de carga de los motores y la alimentación del módulo de recepción de datos por wifi, que no rebasará un incremento de 100 a 150 [mA]. Derivado de lo anterior se puede establecer una necesidad energética máxima y se propone el uso de una batería Steren recargable de ácido plomo con una carga máxima de 4 [A] y voltaje nominal de 12 [V], suficientes para la alimentación del controlador de motores y que éste a su vez alimente el circuito de control (tarjeta *ESPino*).

Figura 53. Batería utilizada para el desarrollo del sistema robótico.

Una vez aclaradas las diferentes tecnologías componentes del sistema robótico, la integración de dichas tecnologías se llevó acabo utilizando uno de los diseños ya existentes dentro del laboratorio del *Mecathronics Research Group* y utilizados en otras tesis como [37], [45]; en las cuales se prueban algunos conceptos sobre la configuración (2,0) del robot móvil y se obtuvieron resultados favorables con respecto al desempeño de dicho diseño integrador mostrado en la figura 54.

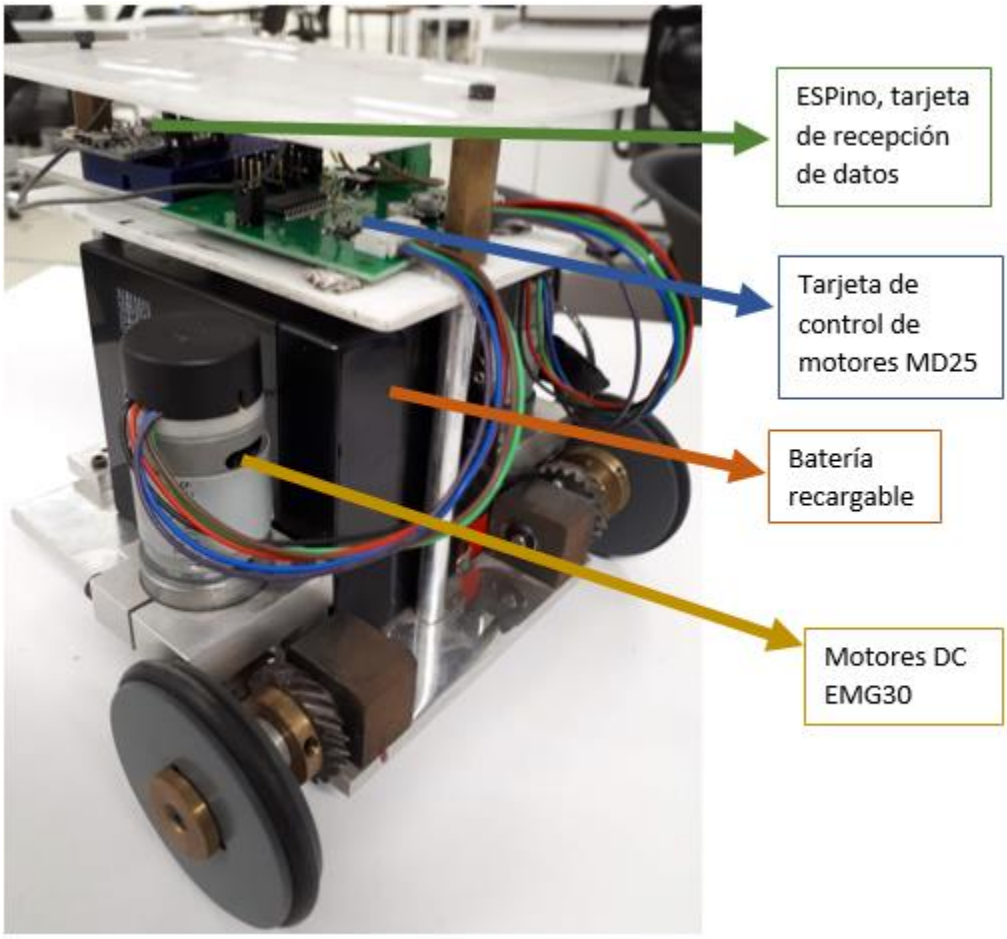


Figura 54. Dispositivo de pruebas con los subsistemas anteriores ya integrados. [37]

Utilizando dicho modelo integrador con la tecnología anteriormente descrita, se armaron tres agentes robóticos diferenciales con la misma técnica y utilizando el mismo diseño como se muestra en la figura 54.

Es importante mencionar que, en función de respetar la esencia de escalabilidad dentro del método de enjambre propuesto (*flocking*), cada uno de los agentes tiene las mismas características tanto mecánicas como electrónicas así como mismas velocidades de comunicación y ancho de banda, con lo cual, se garantiza que se encuentran todos en iguales condiciones y por lo tanto, se controla idénticamente diferenciándose exclusivamente por la posición relativa de cada uno de ellos dentro del ambiente inteligente.

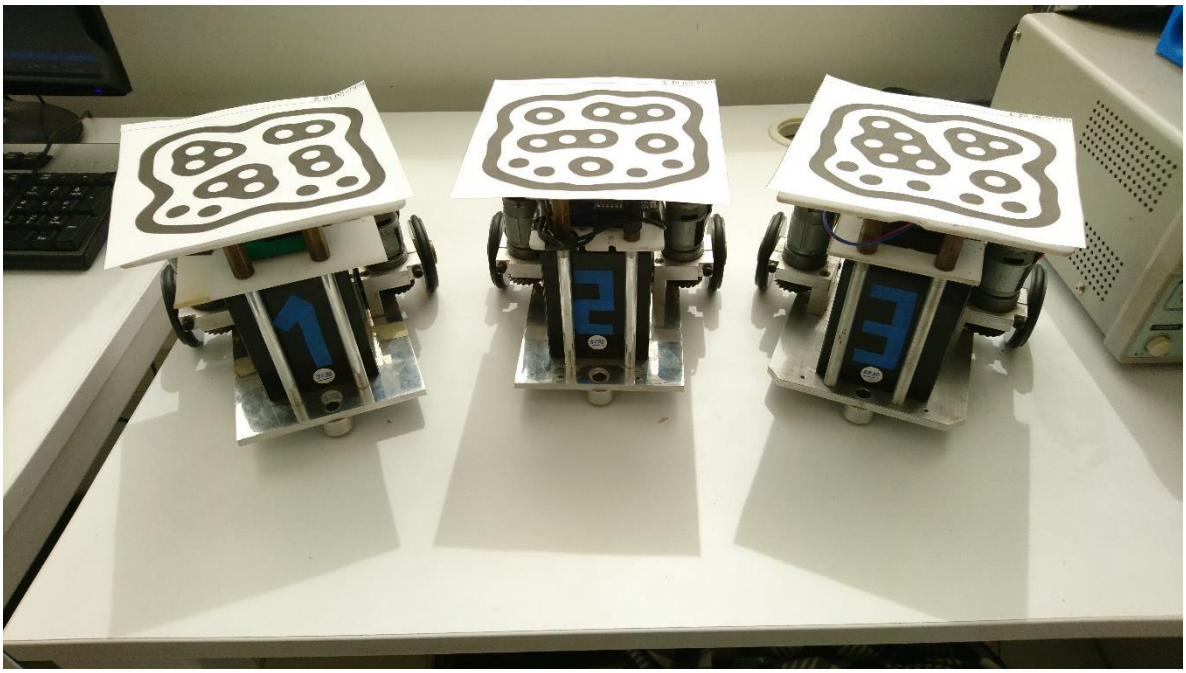


Figura 55. Agentes armados para pruebas de validación.

Por otro lado, para cumplir con la tarea de adquisición de imágenes, se utilizaron dos cámaras Logitech HD PRO C920 de alta definición a 1080p compatible con Windows 10 y Mac OS, conectadas a la computadora de adquisición de datos mediante un puerto USB 3.0 en función de maximizar la velocidad de transmisión. Utilizando un soporte universal con patas flexibles fue posible ubicar las cámaras a una distancia de 2.4 metros del piso.

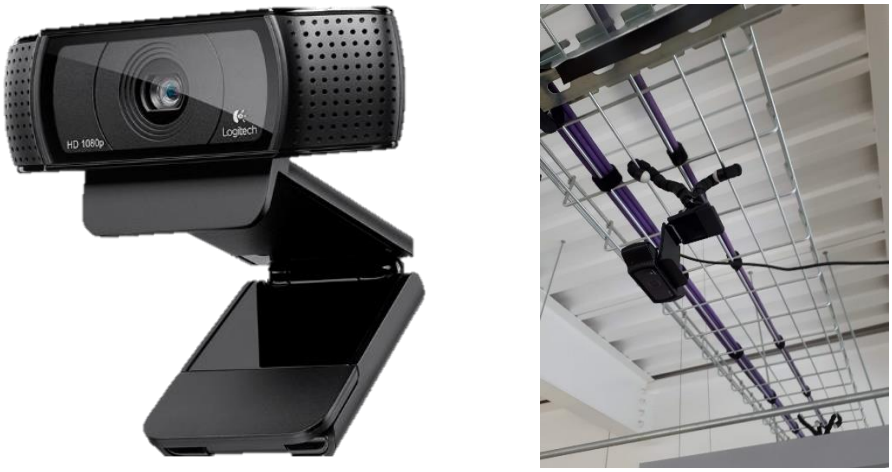


Figura 56. Cámara utilizada para la adquisición de imágenes colocada a una altura de 2.4 m sobre el nivel del piso.

A partir de esta cámara se realiza la adquisición de imágenes para pasarlas posteriormente, como se ha dicho antes, por los filtros y postprocesos en el software de *ReacTIVision* para captar la localización relativa de cada uno de los agentes con sus respectivos *fiducials*. Dichos agentes tienen colocados en la parte superior los *fiducials* 2, 3 y 4 que fueron impresos en hojas de papel Bond con un gramaje aproximado de 80 gr/cm^2. Con dimensiones de 14.5 x 14.5 cm, con las cuales se comprobó mediante pruebas anteriores que es un tamaño adecuado para que la cámara detecte

los movimientos del agente sin problemas de pérdida de visión. Se ha visto que, si el *fiducial* es muy pequeño con relación a la distancia a la cámara, se pueden tener problemas de visión al momento de moverse el agente en el entorno inteligente, lo cual, a su vez, puede hacer que las mediciones y cálculos de distancia no se realicen adecuadamente y traigan como consecuencias inestabilidades en el sistema durante el funcionamiento. Con lo anterior, se puede decir que existe una relación directamente proporcional entre la calidad de la visión y el cociente entre la distancia a la cámara y el tamaño del *fiducial*. Con lo cual es posible proponer un número que llamamos índice de tamaño, el cual se encuentra definido por:

$$IT = \frac{Longitud\ del\ fiducial}{Distancia\ a\ la\ cámara} \qquad (33)$$

Se puede determinar que, a mayor índice de tamaño, se tendrá una mayor cantidad de pixeles que representen al agente y, por lo tanto, una mejor capacidad de tomar y procesar imágenes; mientras que también, a menor distancia entre el agente y la cámara el índice de tamaño será mayor. El índice de tamaño que se utilizó durante las pruebas realizadas es el siguiente:

$$IT = \frac{0.145\ [m]}{2.4\ [m]} = 0.0604$$

Por otro lado, los *fiducials* que sirven como referencia para poder realizar la conversión entre pixeles a metros son de fundamental importancia para el desarrollo del experimento, debido a que si alguna de las cámaras dejara de observarlos entonces no habría referencia de tamaño. Por eso, en este caso, el índice de tamaño para ellos fue de:

$$IT_{fiduciales\ referencia} = 0.0833$$

Garantizando su visibilidad para las cámaras debido a que se trata de un índice 37.9% mayor en comparación al índice utilizado en los agentes robóticos.

Finalmente, toda la información y los cálculos para determinar la motorización de cada uno de los agentes en el ambiente inteligente, así como la declaración de variables se realizó en un sistema *Simulink de Matlab* 2017 versión R2018a corriendo sobre un sistema operativo Windows 10 en una computadora con las siguientes características:

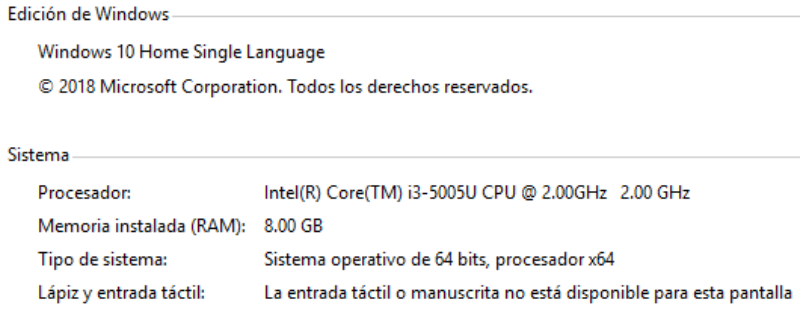


Figura 57. Características del sistema de cómputo utilizado para el procesamiento de datos.

6.2.7 Tratamiento de datos, conexión del sistema de recepción del agente

A lo largo de la sección anterior se explicaron las características de hardware que se utilizaron para la realización de los experimentos que se analizan con mayor profundidad en la sección 7, estos elementos en cada agente se encuentran interconectados entre sí, relacionándose tanto energéticamente como en comunicación siguiendo el esquema planteado en la figura 58.

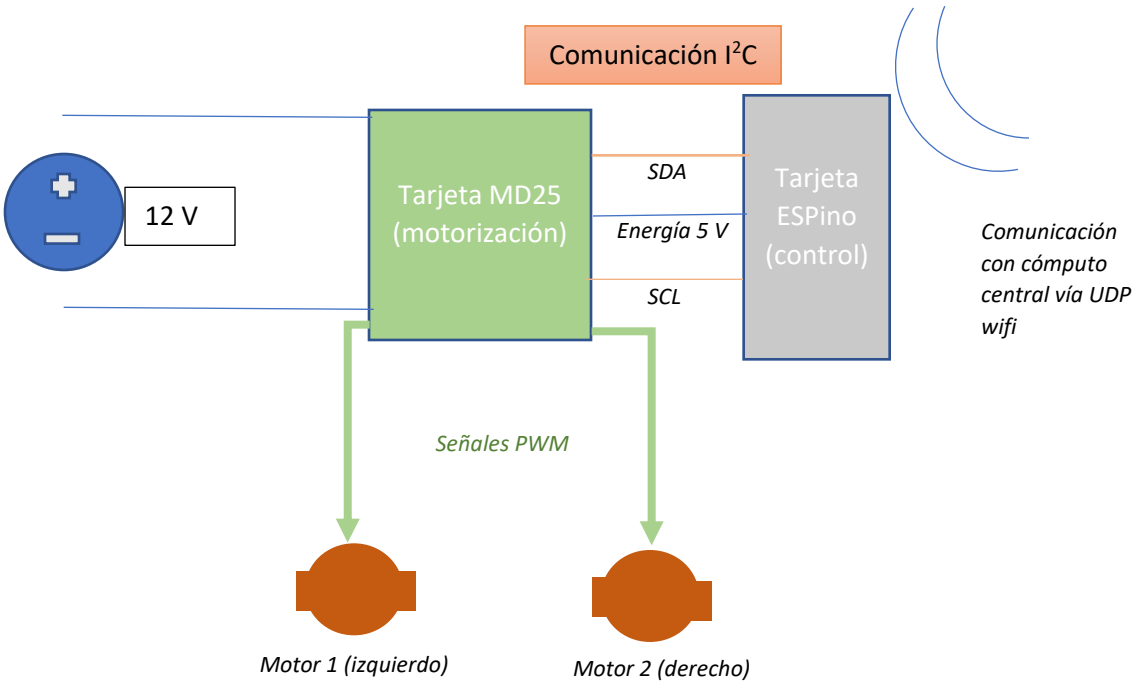


Figura 58. Esquema de conexiones y comunicación entre los elementos de hardware.

Cada agente sigue el mismo esquema de conexiones presentado en la figura anterior. Para cada uno de ellos se genera la información necesaria para su movimiento con base en el modelo propuesto y, como se planteó anteriormente, se transmite a partir de protocolo UDP con dirección a la tarjeta de control del agente *ESPino*. Dicha tarjeta se codifica en Arduino utilizando la librería wifi UDP para poder hacer uso de este protocolo durante el desempaquetamiento de información. Por otro lado, a diferencia de lo que ocurre para comunicar *C#* con Matlab *Simulink* por medio del mismo protocolo, la cantidad de datos a transmitir en este caso es mucho menor, dado que para la comunicación de C# con *Simulink* es necesario transmitir: número de *fiducial*, posición en X, posición en Y, ángulo del *fiducial* y carácter de término de cadena; mientras que en el caso de la transmisión de datos de la computadora central a la tarjeta de control *ESPino* del agente solamente es necesario transmitir dos valores enteros entre 0 y 255, uno para cada uno de los motores. Posteriormente, la tarjeta MD25 decodifica y traduce estos valores en señales PWM para regular la velocidad de los motores de cada uno de los agentes. Como en el caso para la comunicación entre *C#* y *Simulink*, donde se tuvo que codificar en ambos lados la información adecuada para mandar y recibir los datos, también en este caso es necesario configurar el Socket de salida que contiene la IP y el puerto a conectar para realizar la comunicación dentro del entorno de *Simulink* y, por otro lado, dentro de

la tarjeta *ESPino* también es necesario configurar dicho Socket para conectarse a la IP asignada y que sea capaz de recibir información a partir del puerto habilitado.

Una vez recibida la información mediante el protocolo UDP por medio de la conexión entre la computadora y la tarjeta, la tarjeta *ESPino* realiza la comunicación correspondiente I^2C con la tarjeta de motorización MD25, la cual utiliza las direcciones mostradas en la tabla 4 para la comunicación:

Tabla 4. Direcciones trascendentes para la comunicación del ESPino con la tarjeta de motorización MD25.

Dirección para I^2C	***Relevancia***
MD25ADRESS	Es el byte de información que distingue en qué dirección se encuentra conectada la tarjeta en la red I^2C. Sirve para hacer el *acknowledge* correspondiente para comenzar la comunicación
SPEED1	Es la dirección dentro del microcontrolador de la tarjeta MD25 que controla la velocidad de uno de los motores conectados a ella. Los rangos de valores posibles son de 0 a 255, como se ha visto anteriormente.
SPEED2	Goza de la misma función que la dirección anterior solo que para el otro motor conectado.

```
//**************************Código para I2C**
#define CMD                 (byte)0x00

#define MD25ADDRESS         0x58
#define SOFTWAREREG         0x0D
#define SPEED1              (byte)0x00
#define SPEED2              0x01
#define ENCODERONE          0x02
#define ENCODERTWO          0x06
#define VOLTREAD            0x0A
#define RESETENCODERS       0x20
//*********************************************
```

Figura 59. Direcciones utilizadas en el programa para configurar el ESPino.

Para realizar la configuración y programación del módulo *ESPino* se hizo uso del IDE de Arduino, debido a que se pueden instalar fácilmente las configuraciones para utilizar el ESP8266 en este entorno. Para realizar la instalación correspondiente se debe de agregar el link http://arduino.esp8266.com/stable/package_esp8266com_index.json en la sección de Archivo/Preferencias/Gestor de URL's Adicionales de Tarjetas. Posteriormente, en la sección de Herramientas/Placa/Gestor de Tarjetas, se debe seleccionar la opción de esp8266 y dar clic en instalar. Con este procedimiento queda instalado este tipo de tarjeta y podrá ser programada seleccionando el tipo de tarjeta como *Generic ESP8266 Module*.

La estructura general del programa implementado para la comunicación con los agentes es la siguiente:

1. Se generan las variables requeridas para la comunicación, incluyendo el nombre de la red, la contraseña de la red y el número de puerto.
2. Se inicia la comunicación Wifi para la recepción de datos provenientes del programa en *Simulink*, así como la comunicación I^2C entre el módulo *WiFi* ESP8266 y el controlador de los motores MD25.
3. Si hay paquetes de información recibidos se hace la lectura de la información de las velocidades de ambas llantas.
4. Se escriben los datos de las velocidades en las direcciones correspondientes del controlador MD25 para que sea éste quien controle el movimiento de los motores.

Este procedimiento se realiza constantemente durante la ejecución de la prueba para cada uno de los agentes utilizados, cada uno con un puerto (8080, 8880 y 8088) y dirección IP específicos para lograr la comunicación individual de los robots. El código completo utilizado se incluye en el Anexo 6 para su consulta.

Con esto queda completo el banco de pruebas y el sistema, por lo que se procede a la fase de experimentación e implementación del modelo ya en un espacio real y con agentes robóticos físicos, a fin de comprobar su funcionalidad y viabilidad de aplicación.

7. Pruebas y resultados

Una vez analizados los parámetros computacionales y físicos sobre la implementación del banco de pruebas es posible explicar algunos puntos importantes acerca de la naturaleza de los experimentos. Como se vio en la sección 4 en el estado del arte, algunas de las características fundamentales para poder hablar de cómputo distribuido o inteligencia de enjambre son localidad, escalabilidad, flexibilidad y robustez. En lo que corresponde a la localidad, y a la aplicación de un sistema descentralizado, puede pensarse que resulta contradictorio cumplir con esto mientras que, en la implementación del banco de pruebas, se está utilizando una única computadora que recolecta la información sobre las posiciones y distancias, y realiza los procesamientos requeridos para los cálculos de motorización de los robots. Sin embargo, el sistema está planteado de tal forma que se respete la localidad del algoritmo, es decir, que cada uno de los agentes solo puede hacer uso de los datos correspondientes a lo que alcanzaría a detectar por sí mismo. Esto se logra debido a que, dentro de la codificación del sistema de interacción vectorial, se toma en consideración la distancia de alcance del agente y, en el caso de que algún elemento se ubique a una mayor distancia, éste no tendrá efecto alguno sobre el primero. Lo mismo ocurre con el obstáculo, ya que, si un agente se encuentra más lejos que el radio de repulsión del obstáculo, entonces el agente no lo “ve” y el obstáculo no tendrá efecto sobre el comportamiento del agente.

Entonces, se puede decir que en este experimento se trata de una simulación de localidad en el ambiente de trabajo, dado que no se cuenta con cómputo distribuido como tal. Esto se debe a que, para poder dotar a cada agente por separado de la capacidad de toma de decisiones sobre circunstancias locales, se debe implementar un diseño de sensores capaces de actuar en el ambiente de trabajo con diferentes rangos de alcance para poder validar su funcionamiento, lo que depende tecnológicamente del alcance y calidad de los sensores. Además, también se debe dotar a los agentes de una unidad de procesamiento o microcontrolador que sea capaz de conectarse en tiempo real a la computadora.

El cómputo distribuido basado en el método propuesto permite hacer una aplicación simulada con las características que tendría un enjambre robótico real. Si bien, debido a las limitaciones tecnológicas y de recursos resulta complicado hacer el enjambre con todas estas características, el hecho de realizarlo de manera simulada ayuda para tener en cuenta estas posibles influencias sin invertir tantos recursos para elementos que podrían simularse con otros medios.

Teniendo en cuenta todo esto se realizaron diversas pruebas para evaluar el funcionamiento del algoritmo y determinar las características de éste ya aplicado en un sistema real. Se utilizaron los 3 robots diferenciales descritos en la sección 6.4 para implementar el modelo. El programa completo de *Simulink* implementado para el procesamiento en estas pruebas y el control de los agentes robóticos se encuentra en los anexos correspondientes.

En la siguiente tabla se puede encontrar un resumen de los parámetros y constantes utilizados para la implementación de las pruebas con el sistema robótico real.

Tabla 5. Parámetros y constantes para la configuración de las pruebas con agentes robóticos reales.

Configuración de ambiente			
Límite del espacio [mm]		**R obstáculo[mm]**	**R alcance(obs)[mm]**
X [-1200, 1200]	Y [-800, 800]	150	300
V máx. rep. obs[mm/s]:	800	**μ = 0.7**	
V máx. atr. Obj [mm/s]:	300		
Configuración de interacción entre agentes			
R atr [mm]	**R rep [mm]**	**R ali [mm]**	
1000	400	500	
V máx rep [mm/s]:	300	**V máx atr [mm/s]:**	400

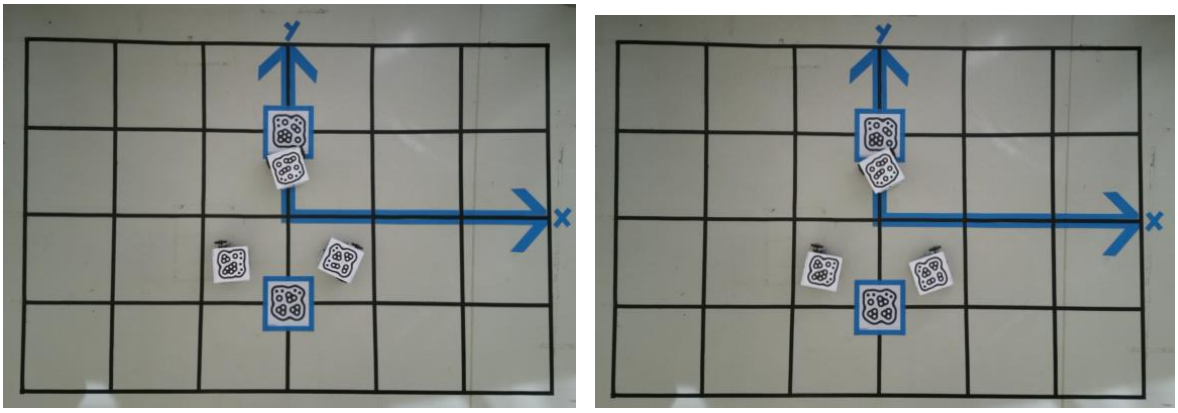
7.1 Pruebas de formaciones

Las primeras pruebas realizadas fueron con el objetivo de demostrar la capacidad de este modelo de mantener formaciones entre agentes, independientemente de la generación del desplazamiento hacia el objetivo. Para comprobar esto se colocaron los agentes dentro del espacio de trabajo y se observó su comportamiento al entrar en el área de alcance de interacción entre agentes.

7.1.1 Prueba de formación desde zona de atracción

a)

b)



c) d)

Figura 60. Prueba de formación entre 3 agentes desde zona de atracción.

Para la prueba mostrada en la figura 60 se colocaron los 3 agentes dentro del área de atracción del resto ubicándolos con distintas orientaciones y se observó la interacción generada entre ellos. Se observa que, inicialmente, debido a que los agentes se encuentran más separados que sus posiciones de equilibrio, estos se desplazan hacia el centro aproximándose entre sí. Posteriormente, una vez que se aproximan más allá de la frontera con la zona de orientación, los robots dejan de atraerse y se ven influenciados por una velocidad angular que tiende a orientarlos con respecto al resto del enjambre. Este efecto se observa más significativamente en el agente ubicado en la parte inferior derecha, ya que éste es el que se encuentra más desorientado comparado con los otros dos. Debido a la proporción que se le dio a esta influencia de la zona de orientación, el efecto generado no alcanza la magnitud necesaria para orientar perfectamente a todos los agentes en una misma dirección. Además, por el par requerido para vencer la inercia de los motores de los agentes, cuando la magnitud de la velocidad que se desea generar es muy baja, el agente simplemente no alcanzará a moverse hasta que se sobrepase una determinada magnitud.

A continuación, se presentan las gráficas correspondientes a las coordenadas y orientación de los agentes a lo largo de esta prueba.

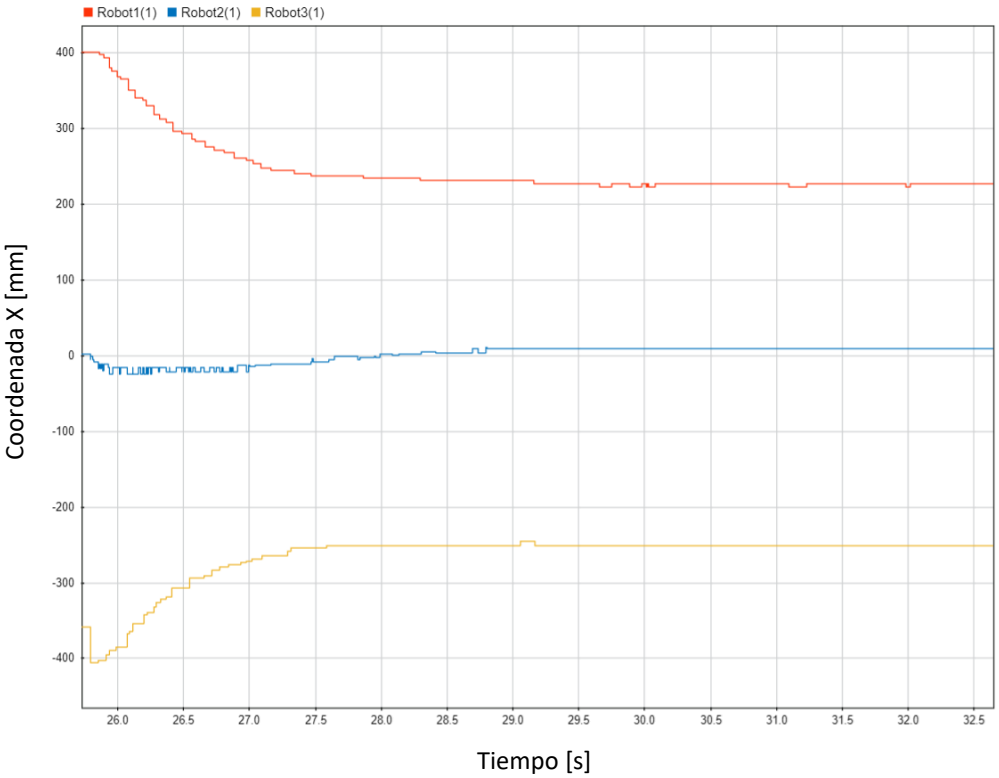


Figura 61. Coordenada 'x' de los agentes involucrados en la prueba de formación desde zona de atracción.

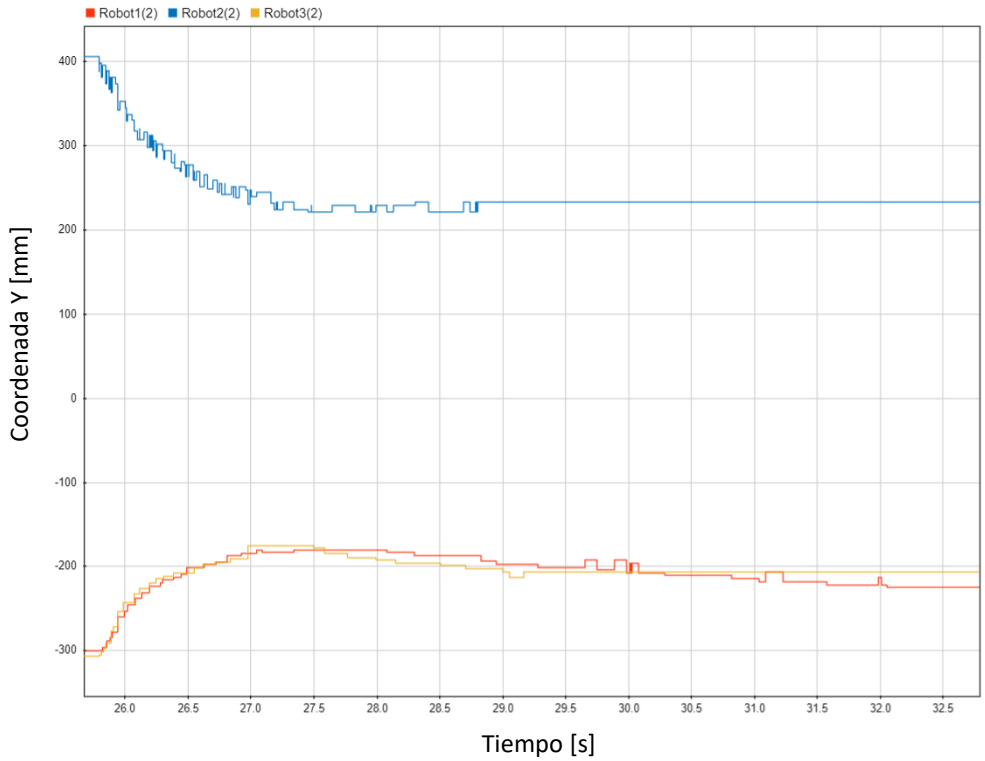


Figura 62. Coordenada 'y' de los agentes involucrados en la prueba de formación desde zona de atracción.

En la figura 61 y figura 62, se presentan los datos referidos a las coordenadas (x, y) de posición de los agentes respectivamente. Aquí se puede observar que los agentes se aproximan entre sí en ambos casos, por lo que la distancia entre ellos se reduce paulatinamente hasta llegar al punto de equilibrio. Transcurridos 3 segundos de funcionamiento (a partir del segundo 29), se aprecia que los agentes han salido ya de la zona de atracción, por lo que ya no se aproximan entre sí. Únicamente el agente 1 (identificado con color rojo) continúa presentando un ligero desplazamiento en el eje 'y'; esto resulta como consecuencia del efecto de orientación, ya que el agente no logra girar perfectamente alrededor del centro del *fiducial*, generando un corto desplazamiento. Si analizamos la orientación de los robots durante esta prueba (figura 63) se aprecia que la orientación del robot 1 efectivamente continúa cambiando tras este punto, aproximando su orientación cerca de 25° más con respecto al promedio de la orientación del resto de los robots.

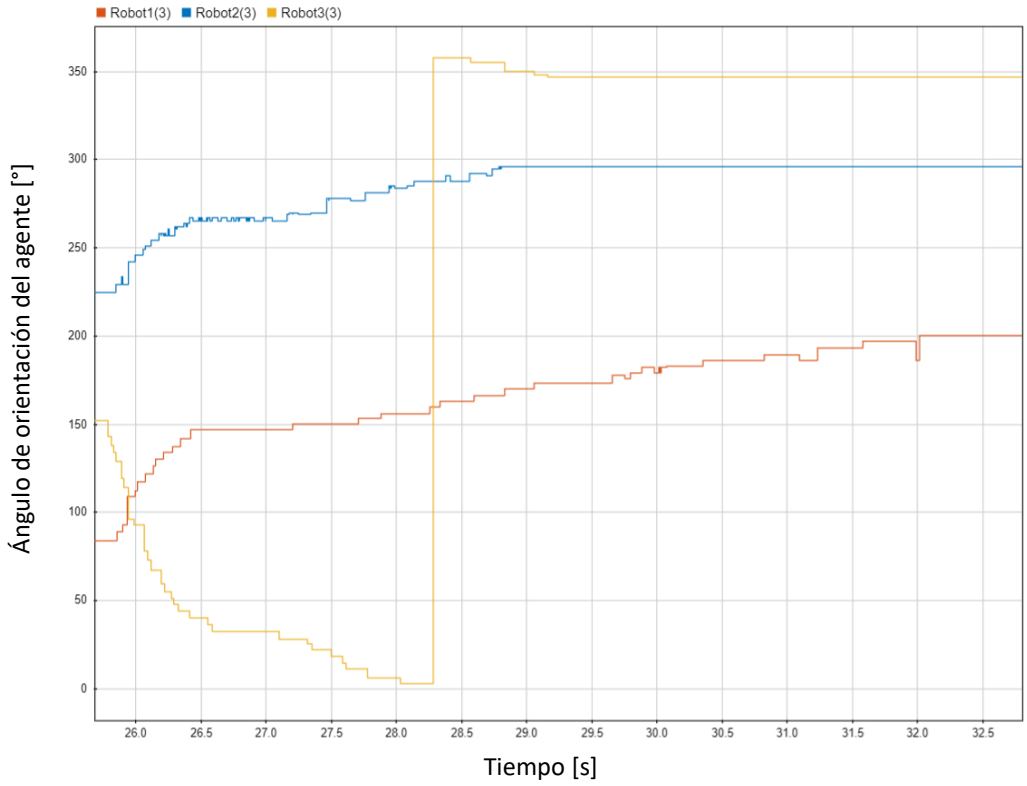


Figura 63. Orientación de los agentes involucrados en la prueba de formación desde zona de atracción.

Para la figura 63 cabe hacer la aclaración del salto que se presenta en el robot 3 (de color amarillo), esto se presenta debido a que la medición de la orientación de los robots se realizó mediante el ángulo entre la dirección del eje 'x' positivo y la dirección hacia donde apunta el agente en cuestión, utilizando un rango de 0° a 360° y excluyendo el último valor (360°) para tener una única representación para cada orientación. En consecuencia, si algún agente gira sobre estos valores frontera, se producirá esta discontinuidad en los valores registrados que se observa como un salto de 0° a 360°, o viceversa, en las gráficas de orientación.

A partir de esta información se puede calcular también la distancia promedio final entre agentes.

Tabla 6. Posiciones finales de los agentes en prueba de formación desde zona atractiva.

	X [mm]	Y[mm]
Agente 1 (rojo)	227	-225
Agente 2 (azul)	9	233
Agente 3 (amarillo)	-252	-206

Tabla 7. Distancias entre agentes en prueba de formación desde zona atractiva.

Distancia [mm]	Agente 1 (rojo)	Agente 2 (azul)	Agente 3 (amarillo)
Agente 1 (rojo)	0	507.24	479.38
Agente 2 (azul)	507.24	0	510.73
Agente 3 (amarillo)	479.38	510.73	0

Analizando esto se determinó que la distancia promedio final entre agentes para esta prueba fue de 499.12 [mm].

También es posible obtener la gráfica correspondiente a las trayectorias que siguieron los agentes desde la posición inicial hasta el final de la prueba. En las gráficas de trayectorias de los agentes presentados en este trabajo se tienen las siguientes consideraciones, se muestran las trayectorias de los agentes con diferentes colores (Agente 1 – rojo, Agente 2 – Azul, Agente 3 - verde). Se muestra la trayectoria del centroide de la formación con una línea punteada de color amarillo. Además, para cada una de las trayectorias mencionadas se muestra el punto de inicio con un asterisco en color negro, y el punto final con un asterisco en color magenta. Finalmente, la cuadrícula principal está configurada para corresponder con la cuadrícula de ambiente de trabajo real (cada 400 mm), y la cuadrícula secundaria está marcada cada 50 [mm].

La gráfica presentada en la figura 64 muestra las trayectorias seguidas por los agentes para esta primera prueba, así como la del centroide de la formación.

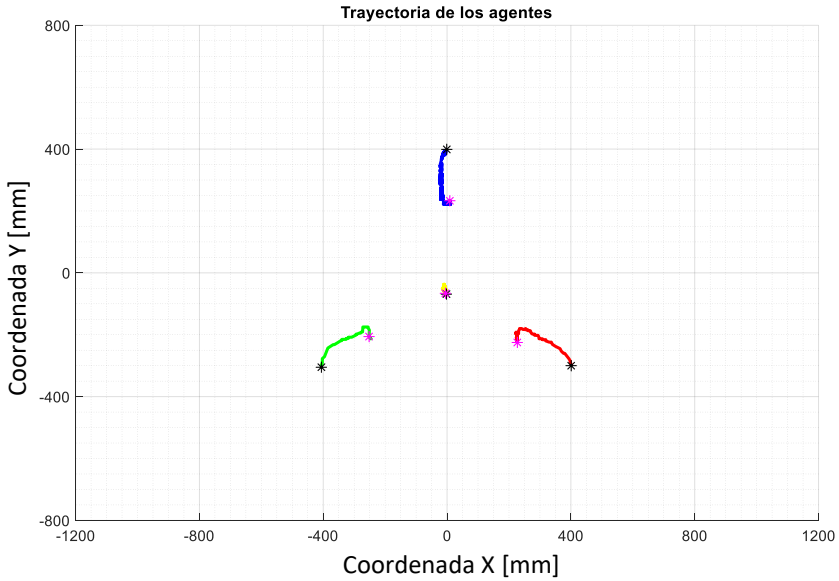


Figura 64. Trayectorias de agentes para prueba de formación desde zona de atracción.

Aquí se observa más claramente el movimiento de aproximación que llevaron a cabo los agentes, así como también se aprecia que el centroide de la formación se mantiene prácticamente en la misma posición de inicio.

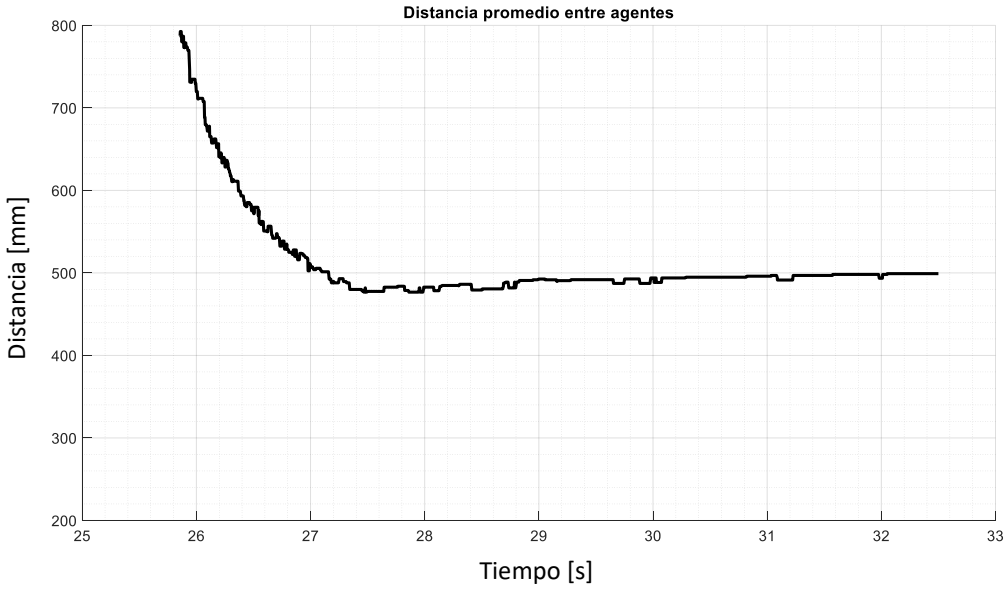


Figura 65. Distancia promedio entre agentes para prueba de formación desde zona de atracción.

Otra información importante por analizar para evaluar el comportamiento de enjambre es la distancia promedio entre agentes a lo largo de la prueba. En este caso se presenta en la figura 65 donde se observa cómo la distancia entre agentes disminuye hasta terminar en un valor aproximado a 500 [mm], que correspondería con las posiciones de equilibrio de los agentes en estas pruebas. Cabe resaltar, que esta distancia promedio entre agentes de equilibrio corresponde también con el límite exterior de la zona de orientación (radio de orientación), el cual se estableció precisamente

en 500 [mm] de radio alrededor del agente. Esto es de esperarse, ya que, al encontrarse los agentes en la zona de atracción entre sí, éstos se aproximarán hasta salir de dicha zona y entrar a la de orientación, por lo que tienden a quedar en equilibrio cerca de dicha frontera.

7.1.2 Prueba de formación desde zona de repulsión

Se realizó esta misma prueba en numerosas ocasiones cambiando las posiciones y orientaciones iniciales de los agentes, observándose que el objetivo del comportamiento de enjambre para mantener una formación se cumplía exitosamente. Entre las pruebas alternativas cabe resaltar la prueba en la que los agentes inician dentro del área de repulsión. En este caso el comportamiento fue el presentado en la figura 66.

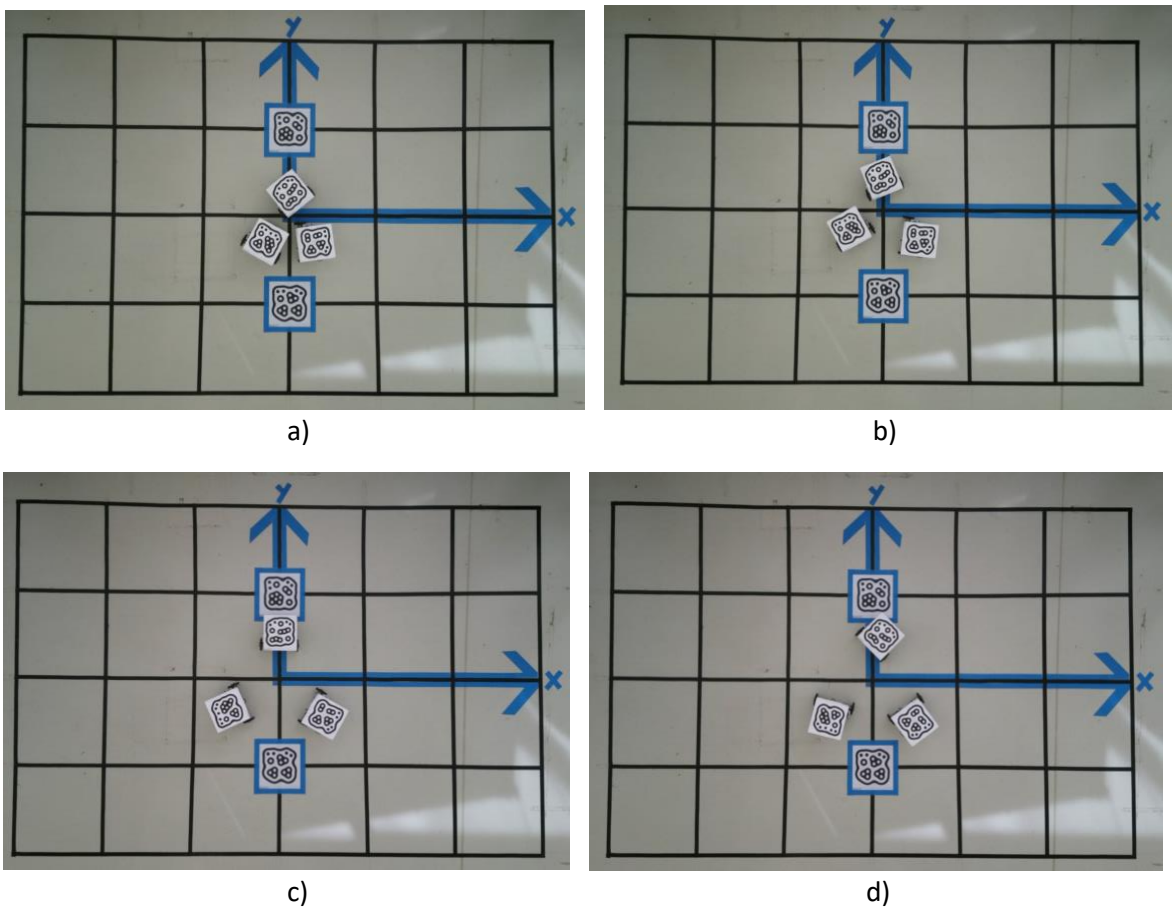


Figura 66. Prueba de formación entre 3 agentes desde zona de repulsión.

La principal diferencia de esta prueba con respecto a la anterior es que en ésta los agentes pasan de la zona de repulsión a la zona de orientación, es decir, llegan a la zona de orientación desde la frontera interior, mientras que en la prueba anterior llegan desde la frontera exterior. Esta diferencia se ve reflejada en las distancias finales entre agentes. Tras hacer los cálculos correspondientes se determinó que la distancia final promedio entre agentes para esta última prueba fue de 407.5 [mm], lo cual es considerablemente menor que el mismo resultado para la

prueba realizada partiendo desde la zona de atracción. Esto va de la mano con la definición de la zona de orientación, ya que los agentes tenderán a quedar localizados dentro de esta zona cerca de la frontera de la cual provienen. Esta propiedad puede observarse claramente al hacer la comparación entre estas distancias finales y las distancias de las fronteras de la zona de orientación del modelo implementado. La frontera interna de la zona de orientación se definió a una distancia de 400 [mm] del agente (correspondiente con los 407.5 [mm] de la última prueba), mientras que la frontera exterior quedó definida a 500 [mm] (correspondiente con los 499.1 [mm] de la primera prueba). La distancia promedio entre agentes para esta segunda prueba puede visualizarse en la figura 68.

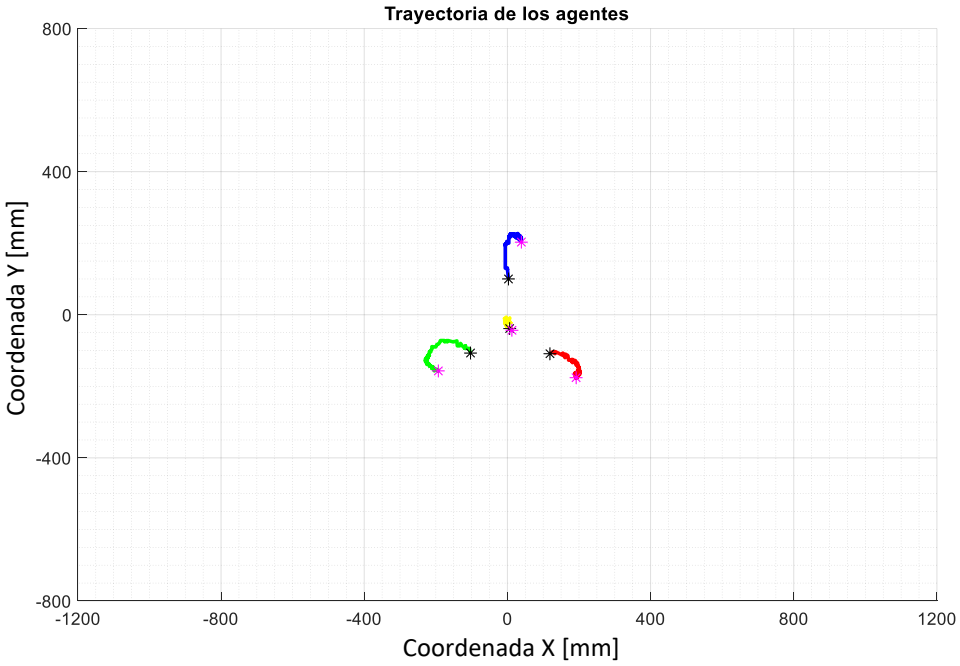


Figura 67. Trayectorias de agentes para prueba de formación desde zona de repulsión.

En esta prueba se puede observar un comportamiento opuesto al obtenido en la prueba anterior. Aquí los agentes en cuestión inician más cerca unos de otros, dentro de la zona de repulsión, por lo que el movimiento se hace para separarse y aumentar la distancia entre ellos. De igual manera se aprecia que el centroide de la formación se mantiene prácticamente en la misma posición a lo largo de la prueba, aun cuando los agentes se desplazan.

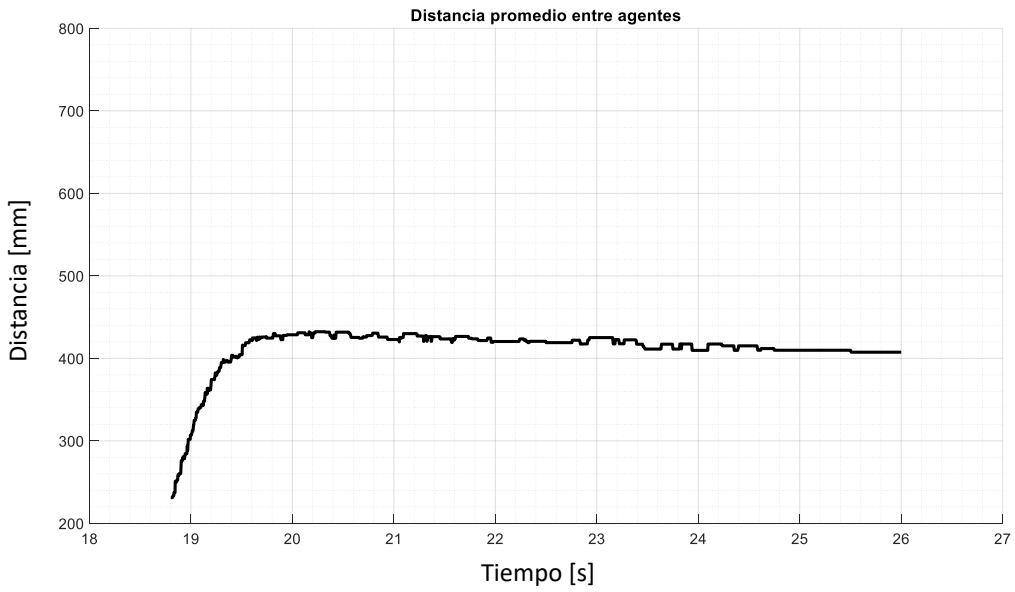


Figura 68. Distancia promedio entre agentes para prueba de formación desde zona de repulsión.

7.2 Pruebas de desplazamiento a un punto fijo

También se realizaron pruebas estableciendo un punto fijo en el espacio de trabajo como objetivo para el enjambre. En estas pruebas se busca que el enjambre sea capaz de desplazarse hasta el objetivo, sin que haya colisiones entre ellos ni con posibles obstáculos en el ambiente. Cabe resaltar que todas las pruebas desarrolladas con los agentes robóticos reales se hicieron con el mismo programa, no se modificaron parámetros, sino que se implementó el mismo modelo, cambiándose únicamente las posiciones iniciales de los agentes y del objetivo u obstáculo correspondiente. También se realizaron pruebas cambiando el número de agentes (con 1, 2 y 3 agentes), de igual manera sin modificar el modelo, llegando a la conclusión de que el sistema funciona correctamente y llega a cumplir su objetivo aun con estas variaciones en el sistema.

7.2.1 Prueba de desplazamiento sin obstáculo

En esta prueba se colocaron los 3 agentes en el espacio de trabajo, con el centroide de la formación ubicado cerca de las coordenadas (800,0), y se estableció el objetivo como el punto en las coordenadas (-800,0). De tal forma que los agentes se desplazan hacia el objetivo sin colisionar entre sí, tal como se muestra en la figura 69.

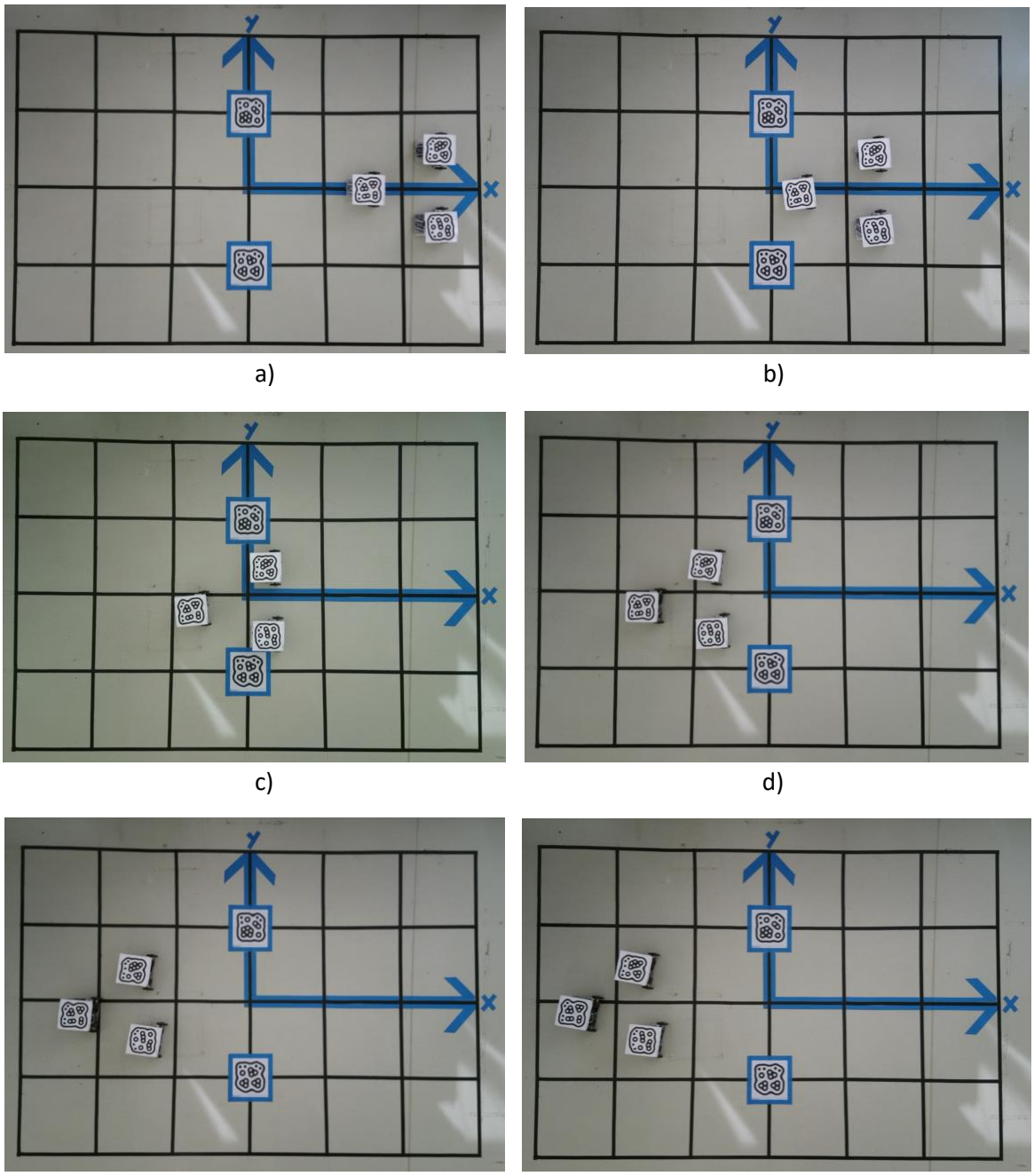


Figura 69. Prueba de desplazamiento de 3 agentes sin obstáculo.

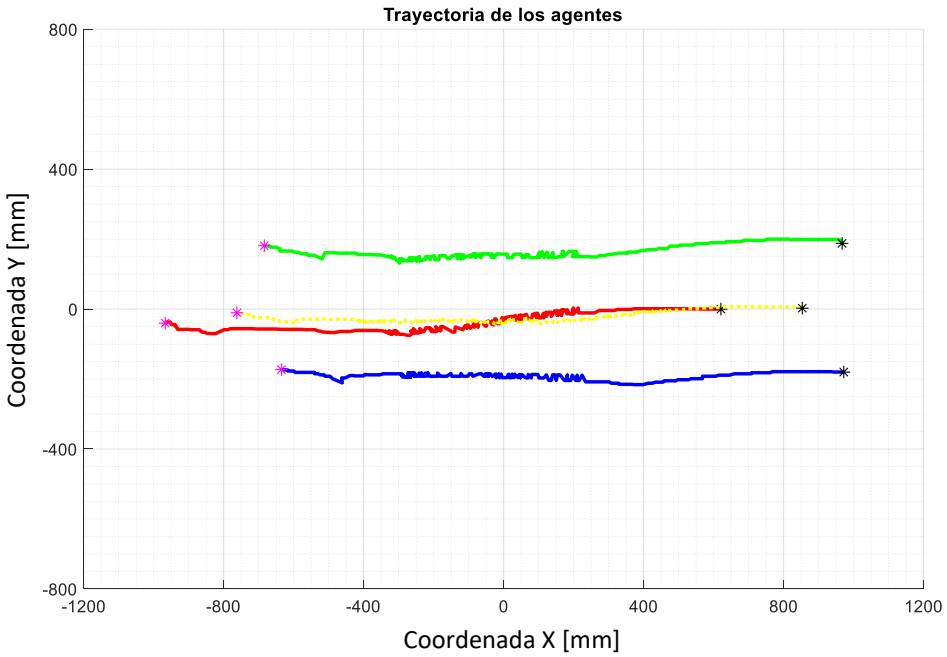


Figura 70. Trayectorias de agentes para prueba de desplazamiento sin obstáculo.

Al tratarse de una trayectoria en línea recta y en la dirección hacia donde apuntan los agentes, la estructura de la formación permanece prácticamente igual durante toda la prueba. Se observan las trayectorias como líneas paralelas entre sí y los agentes mantienen una separación con poca variación, como puede constatarse en la figura 71.

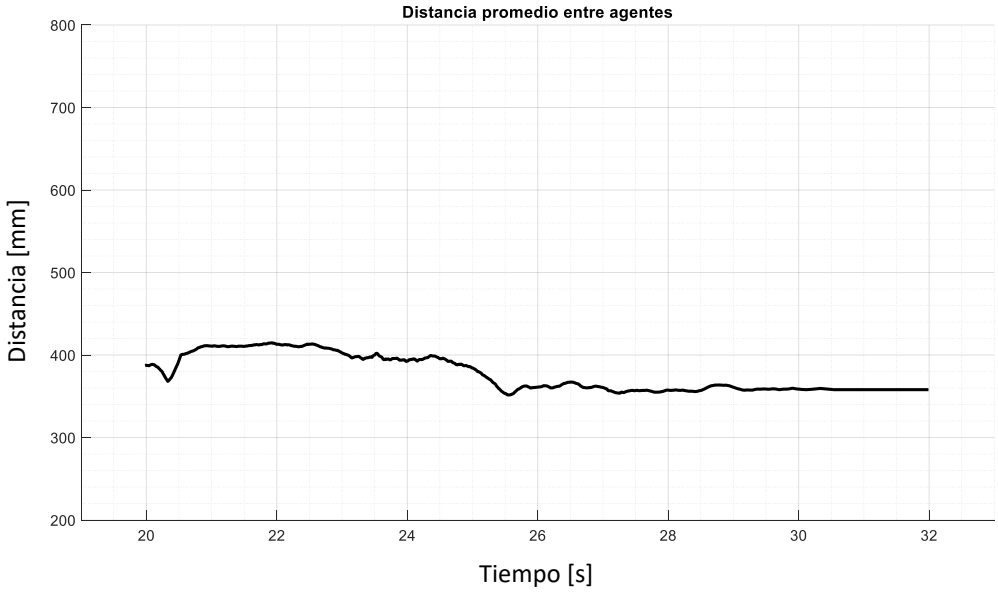


Figura 71. Distancia promedio entre agentes para prueba de desplazamiento sin obstáculo.

Adicionalmente, otro parámetro significativo a evaluar en este tipo de pruebas en las que se tiene un objetivo fijo es la distancia del enjambre al objetivo. Esto permite analizar el grado de precisión con el que se llegó al mismo y la rapidez con que se hizo esta aproximación. Para la posición del enjambre se consideró el centroide de la formación, por lo que, en la figura 72, se muestra la distancia entre dicho centroide y el objetivo, que en este caso fue el punto (-800,0).

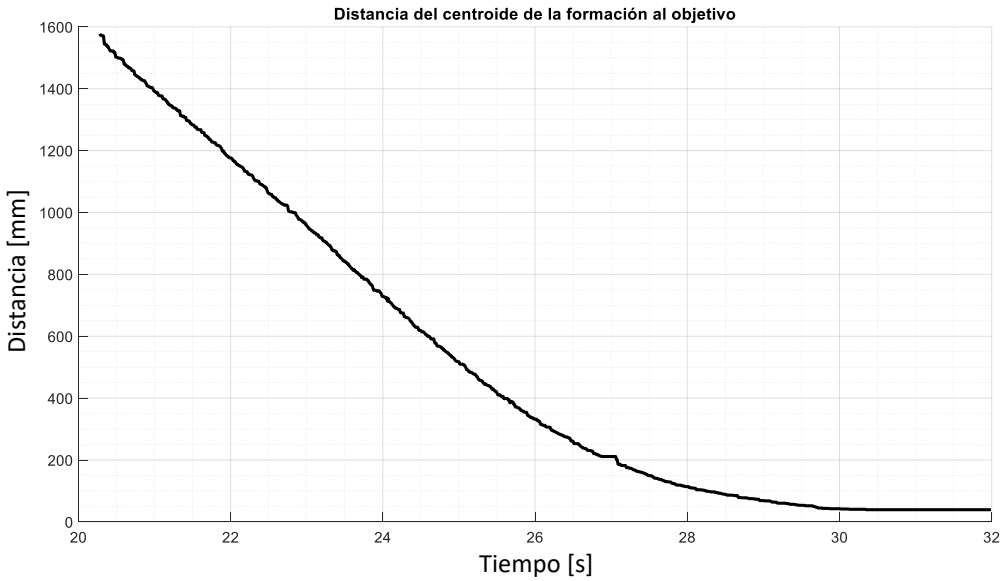


Figura 72. Distancia del centroide de la formación al objetivo en prueba de desplazamiento sin obstáculo.

Si bien el enjambre no llega a ubicarse exactamente en el punto objetivo, sí se aproxima considerablemente hacia él. En esta prueba se registró que la distancia final del centroide de la formación al objetivo fue de 39.5 [mm]. Esta diferencia de posición se debe principalmente a la distancia de protección establecida alrededor del objetivo y las limitantes físicas de desplazamiento que presentan los robots, en conjunto con el límite establecido para la velocidad mínima de desplazamiento de los agentes. Recordando que se definió una distancia de protección (d_p) en la sección 6.1.1, de tal forma que cuando un agente se ubica a una distancia menor que d_p del objetivo se considera que éste ya llegó con el fin de evitar inestabilidad en el sistema, para las pruebas se consideró d_p = 30 [mm]. Por otro lado, se observa que la rapidez de aproximación del objetivo disminuye entre más cerca se encuentre el enjambre de éste. Esto responde directamente a la aplicación del modelo, donde se implementó una velocidad de atracción directamente proporcional a la distancia entre el objetivo y el agente. Para el intervalo en el que los agentes se encuentran más distantes del objetivo la rapidez de aproximación es prácticamente constante, lo que se atribuye a la velocidad de atracción máxima establecida en el modelo.

7.2.2 Prueba de desplazamiento con obstáculo

En esta prueba se estableció como punto objetivo las coordenadas (-800, 0) y se colocó un obstáculo en (0,200). Se puede apreciar en la figura 73 que los agentes adaptan su formación para evadir el obstáculo, a la vez que se desplazan en conjunto y llegan hasta el objetivo. Como se puede ver en estas imágenes, la formación no es fija, es decir, no se mantiene la misma estructura y distancias en la formación durante todo el desplazamiento, sino que, al presentarse el obstáculo, cada agente reacciona de forma independiente.

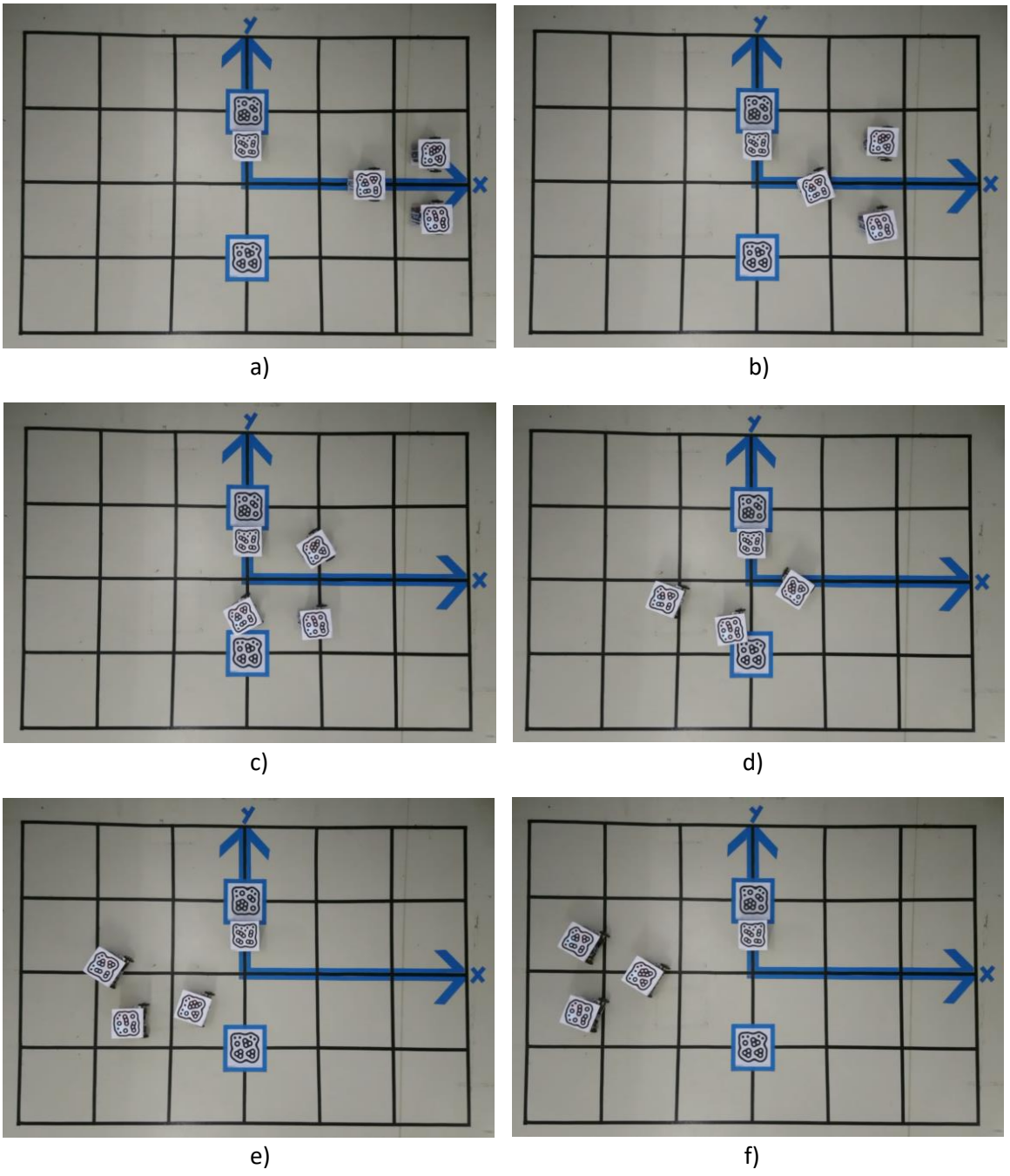


Figura 73. Prueba de desplazamiento de enjambre con 3 agentes y un obstáculo.

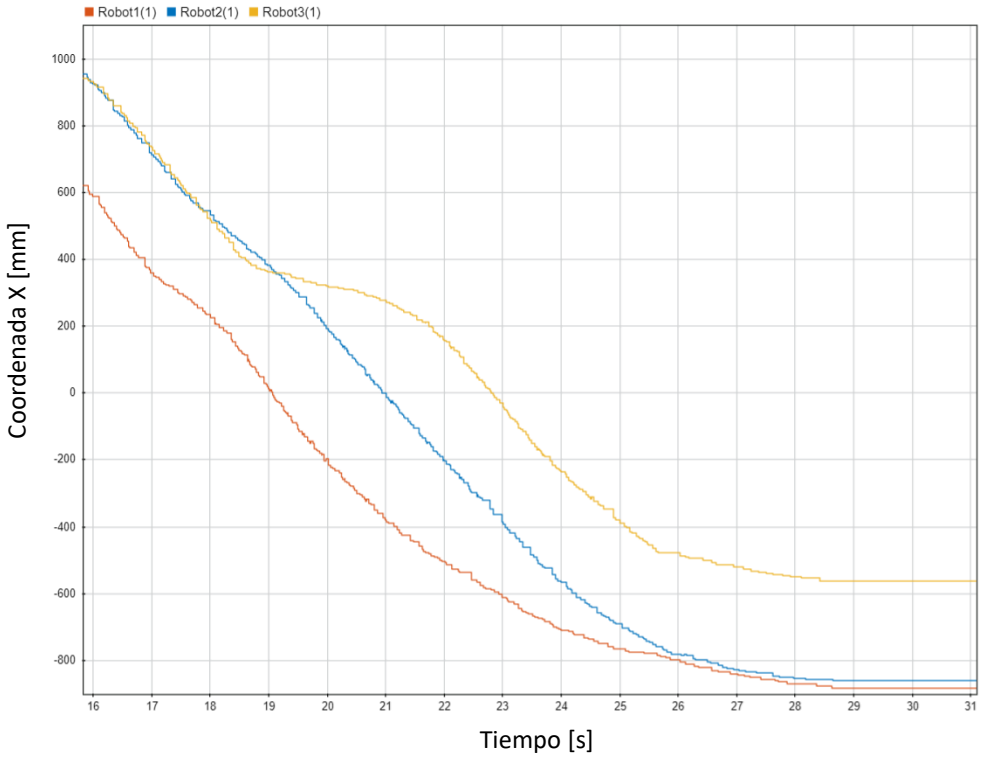


Figura 74. Coordenada 'x' de los agentes involucrados en la prueba de desplazamiento a objetivo fijo con un obstáculo.

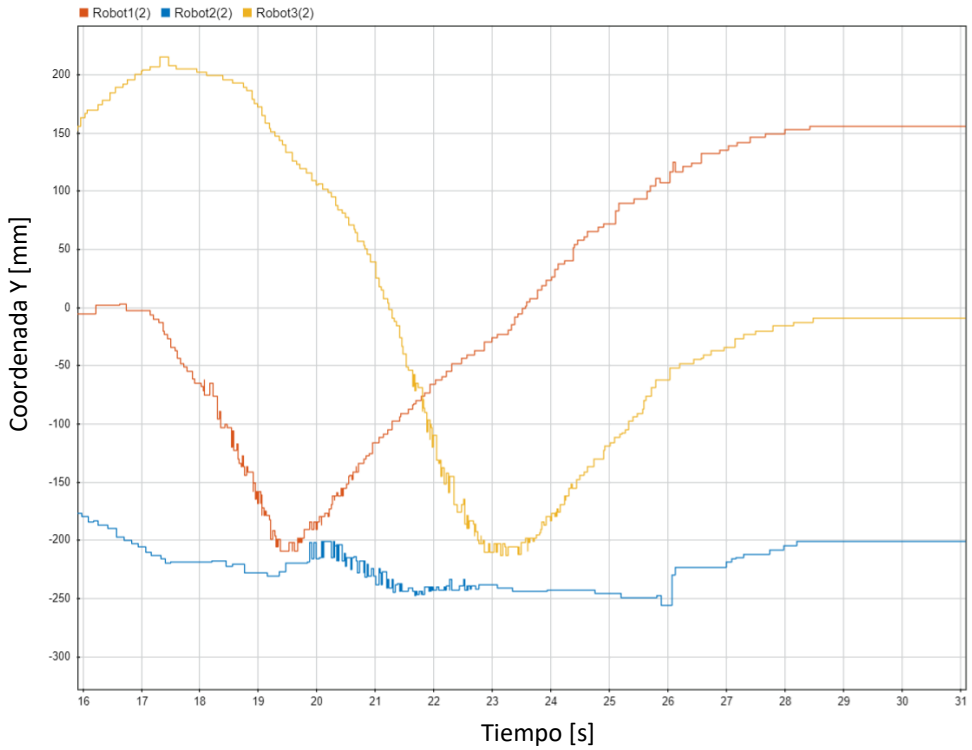


Figura 75. Coordenada 'y' de los agentes involucrados en la prueba de desplazamiento a objetivo fijo con un obstáculo.

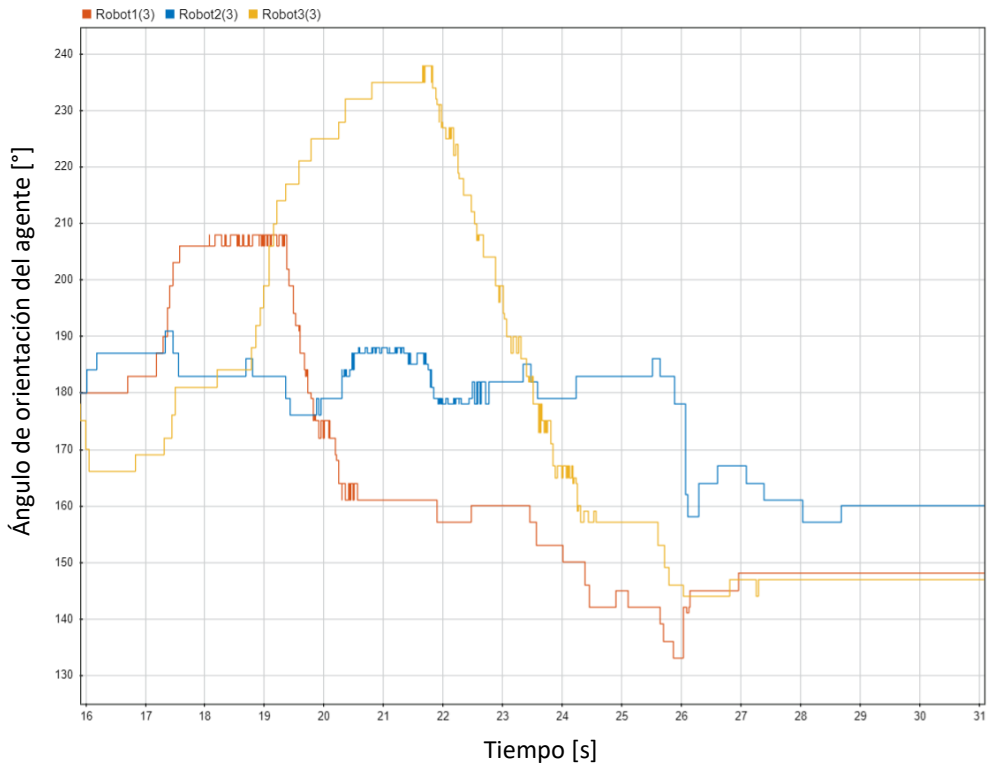


Figura 76. Orientación de los agentes involucrados en la prueba de desplazamiento a objetivo fijo con un obstáculo

En las figuras anteriores se puede apreciar el comportamiento de los agentes a lo largo de esta prueba, mostrándose sus coordenadas y la orientación de los robots. En la gráfica correspondiente a la coordenada 'x' se observa como los agentes se desplazan desde la derecha (coordenada x positiva mayor) hacia la izquierda (coordenada x negativa menor). Por otro lado, la gráfica correspondiente a la coordenada 'y' muestra que los agentes tuvieron que desviar su trayectoria hacia abajo para evadir el obstáculo, siendo más notoria la desviación del agente 3 cuyo desplazamiento se encontraba más directamente sobre el obstáculo.

Como se mencionó anteriormente, no es posible que todos los agentes se ubiquen sobre el objetivo, por lo que se tomará como referencia el centroide de la formación y será este punto el que se analice para determinar si el enjambre ha cumplido la meta de llegar al objetivo. Para esta prueba se tiene que las posiciones finales de los agentes involucrados terminaron con los siguientes valores.

Tabla 8. Posiciones finales de los agentes en prueba de desplazamiento con obstáculo.

	Coordenada x final [mm]	**Coordenada y final [mm]**	**Orientación final [°]**
Agente 1	-883	156	148
Agente 2	-862	-201	160
Agente 3	-565	-9	147
Promedio	-770	-18	151.6

Por lo tanto, la posición correspondiente al centroide del enjambre al término de la prueba corresponde a (-770,-18). El objetivo planteado fue el punto (-800, 0), por lo que la distancia entre la ubicación del centroide y el objetivo resulta ser de 34.98 [mm].

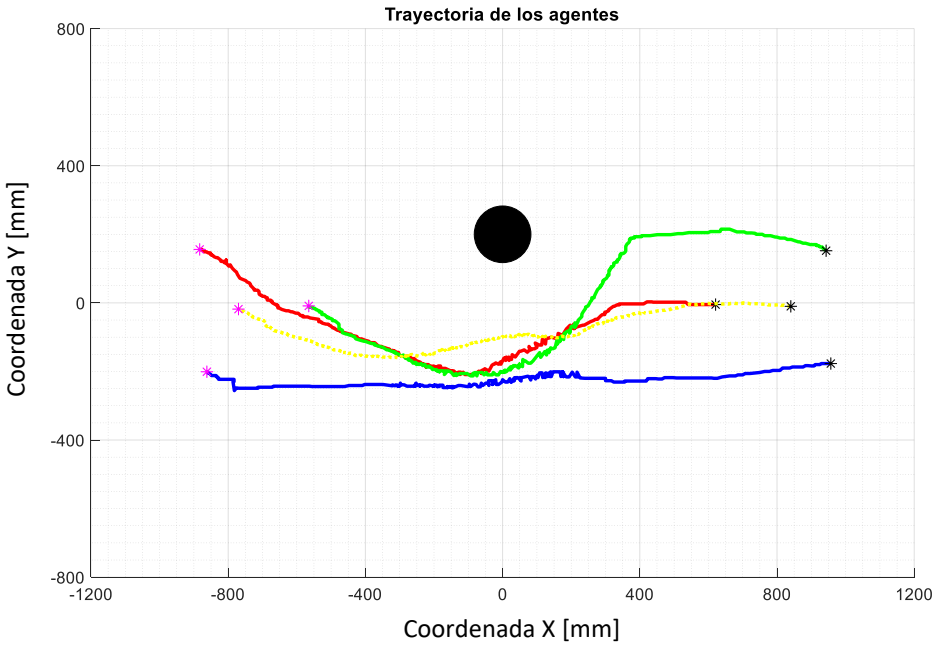


Figura 77. Trayectoria de agentes en prueba de desplazamiento con obstáculo.

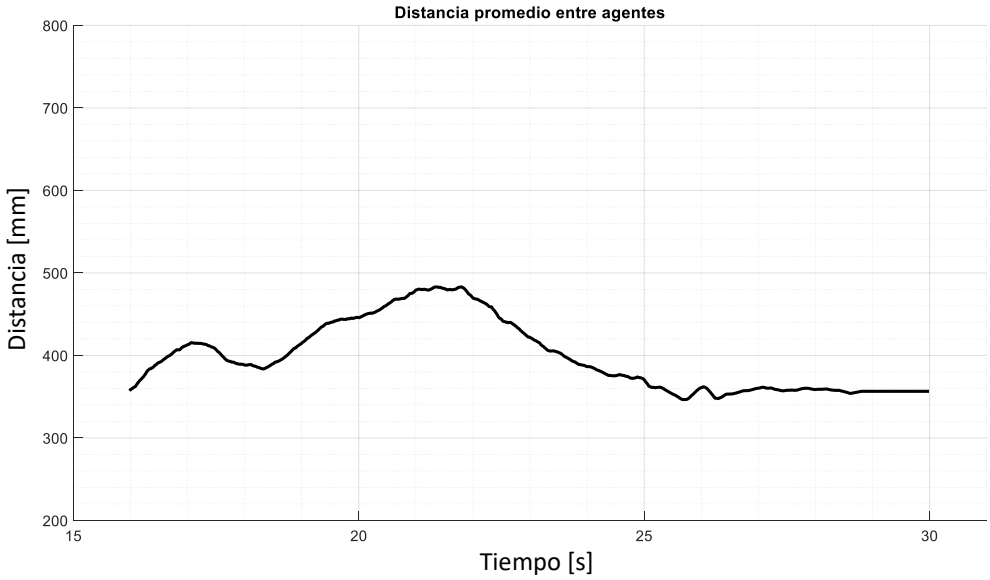


Figura 78. Distancia promedio entre agentes en prueba de desplazamiento con obstáculo.

En la figura 77 se muestran las trayectorias que siguieron los agentes para evadir el obstáculo. Resaltándose que la formación se adapta de acuerdo con las condiciones individuales de cada agente, no solo como una formación fija. Por ejemplo, analizando la trayectoria del agente presentado en color azul, se puede apreciar que dicha trayectoria se ve alterada en baja medida con respecto al desplazamiento sin obstáculo, mientras que la trayectoria del agente representado en color verde se ve mucho más afectada por la presencia del obstáculo. Esto provoca, a su vez, que la formación cambie de estructura durante la prueba, sin embargo, en este ejemplo se observa que, una vez que el obstáculo es evadido, los agentes retoman posiciones con forma similar a la formación definida por las posiciones de equilibrio del modelo.

Por otro lado, en la figura 78 se observa que la distancia entre agentes si sufrió alteraciones con respecto a la prueba sin obstáculos. La distancia promedio entre agentes se incrementó al momento de evadir el obstáculo debido a esta modificación en la formación en la que prácticamente se alinearon los agentes para rodearlo. Este comportamiento puede resultar significativo considerando que esta capacidad de adaptación de las formaciones puede llevar a que enjambres robóticos sean capaces de desplazarse en grupo por zonas y lugares por las que no se tendrían acceso si mantuvieran la misma formación todo el tiempo. Además, este ajuste y adaptación de la formación es dinámica, y depende directamente de las condiciones del entorno y del resto de los agentes que conforman el enjambre.

Adicionalmente, se muestra la distancia del centroide de la formación al objetivo en la figura 79. La distancia a la que queda finalmente el enjambre del objetivo es de 35 [mm], similar al resultado de la prueba sin obstáculo en donde el enjambre terminó a una distancia de 39.5 [mm], y la gráfica muestra de igual forma una aproximación continua con una menor rapidez para posiciones cercanas al objetivo.

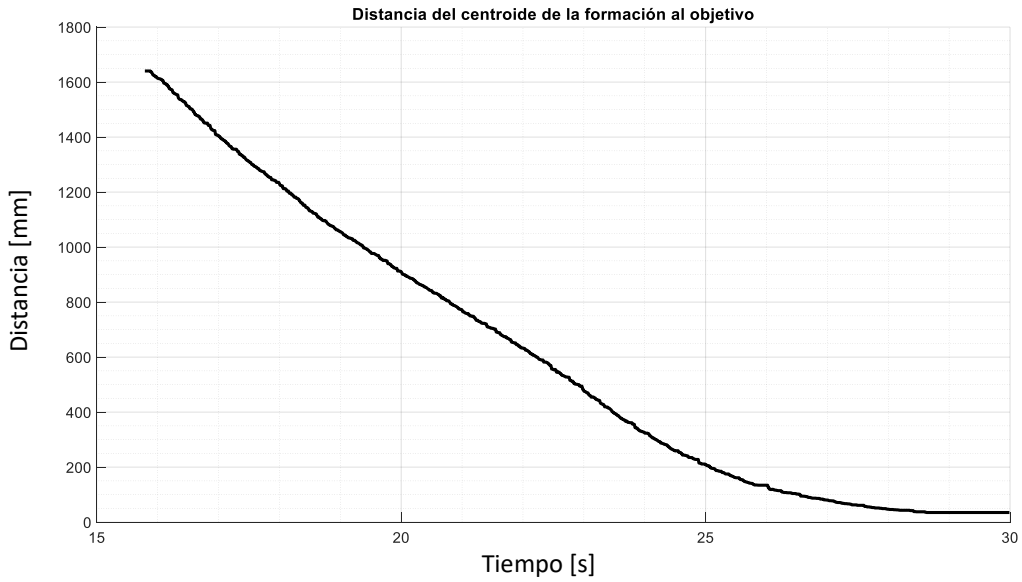


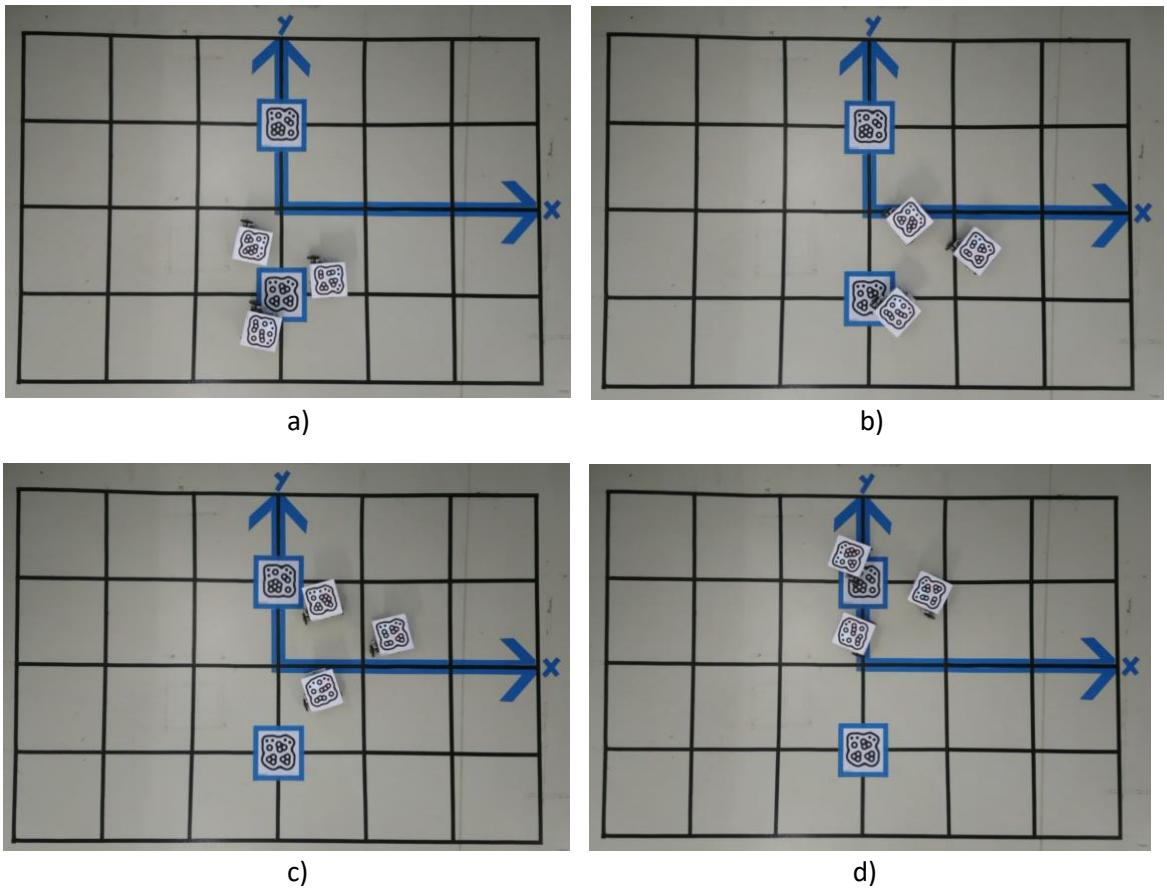
Figura 79. Distancia del centroide de la formación al objetivo en prueba de desplazamiento con obstáculo.

7.3 Pruebas de seguimiento de trayectorias

Otro tipo de comportamiento aplicable a los enjambres robóticos consiste en el seguimiento de trayectorias previamente definidas. Esto tiene aplicación por ejemplo para actividades de vigilancia o en algunos otros campos que requieran ciclos de movimiento repetitivos, por ejemplo, para limpieza, exploración o reconocimiento de alguna zona. Para demostrar la aplicación del modelo propuesto en este tipo de situaciones se realizaron también pruebas de seguimiento de trayectorias, en las que se configura el objetivo de tal forma que vaya trazando la trayectoria que los agentes deben seguir. Al igual que en el caso del desplazamiento a un punto fijo, se considerará el centroide de la formación como punto de referencia de la posición del enjambre en general.

7.3.1 Prueba de trayectoria circular sin obstáculo

Como primera trayectoria base para la realización de las pruebas se consideró una trayectoria circular con centro en el origen del sistema de referencia (0,0) y con un radio de 500 [mm]. Para lograr esto, en la simulación de *Simulink* se modificó el programa para poder incluir un objetivo que no fuera únicamente un punto fijo, sino que fuera cambiando de posición con base en el tiempo de simulación y fuera formando la trayectoria circular antes mencionada. El resultado es que el enjambre va siguiendo la trayectoria trazada por el objetivo como lo muestra la figura 80.



a) b)

c) d)

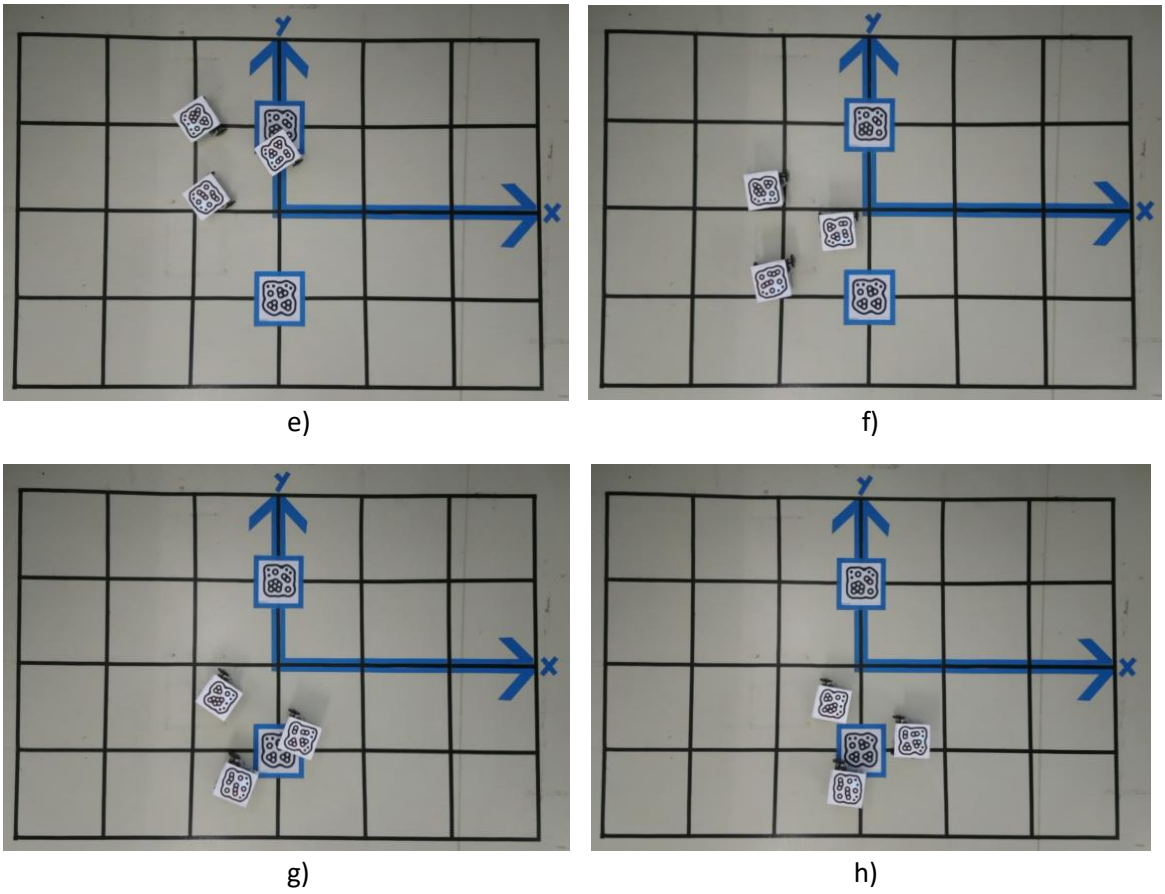


Figura 80. Prueba de seguimiento de trayectoria circular con 3 agentes sin obstáculo.

El enjambre mantiene su comportamiento de interacción entre agentes, por lo que no se presentan colisiones entre sí. Además, analizando el centroide de la formación se puede identificar claramente una trayectoria circular en torno al origen del sistema de referencia tal como se esperaba. Analizando más a detalle las trayectorias seguidas por los agentes (figura 81) se observa un comportamiento de trayectoria circular en cada uno de ellos, sin embargo, este no se realiza en torno al centro de la trayectoria establecida por el objetivo individualmente, más bien, cada agente realiza su movimiento de tal forma que es el centroide de la formación, y por tanto el enjambre completo, quien tiende a seguir la trayectoria establecida como objetivo. Recordando que los agentes están configurados para dirigirse al objetivo, se presenta el hecho de que la circunferencia trazada por el enjambre es de menor tamaño que la trayectoria trazada por el objetivo. Esto debido a que los agentes van siguiendo este objetivo en movimiento y no tienen programado como tal una trayectoria circular de desplazamiento, por lo que cortan un poco la trayectoria resultando en una circunferencia menor. Para solucionar esto se podría hacer el desplazamiento del objetivo más lento de tal forma que dé mayor tiempo a los agentes de desplazarse hasta él y completar la trayectoria con las dimensiones especificadas.

Por su parte, la distancia promedio entre agentes se muestra sin mucha variación, únicamente al principio, en lo que se alcanzan las posiciones de equilibrio en la formación, se presentan mayores variaciones y a partir de ahí se sigue constante. (figura 82)

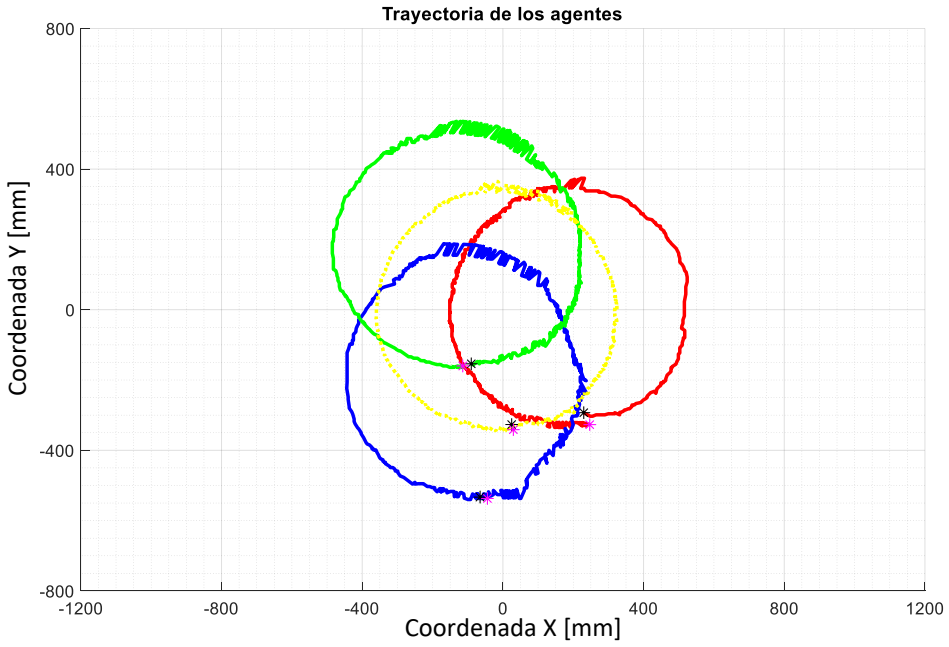


Figura 81. Trayectoria de agentes en prueba de seguimiento de trayectoria circular sin obstáculo.

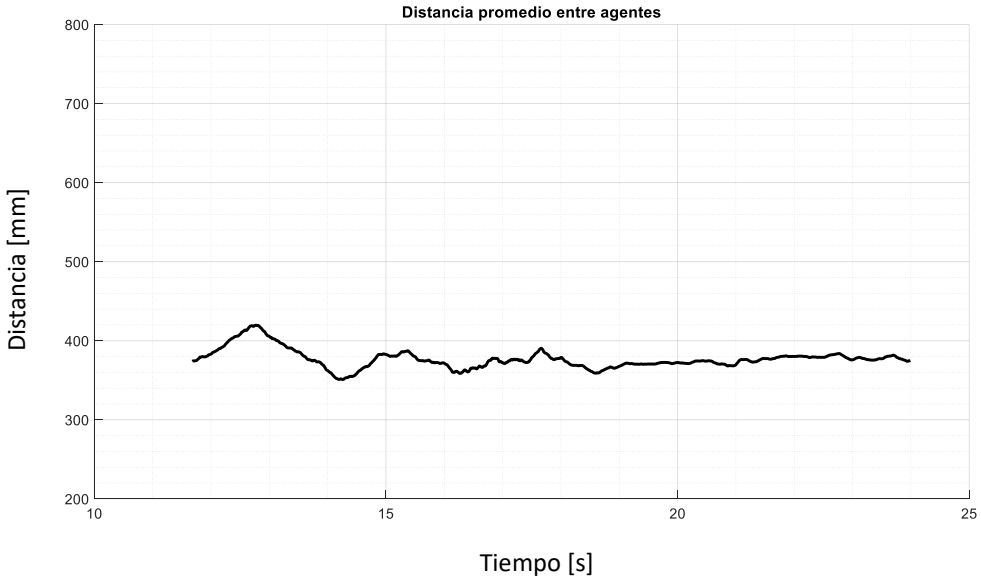
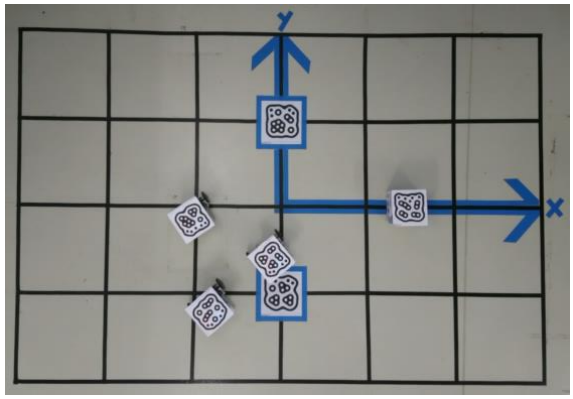


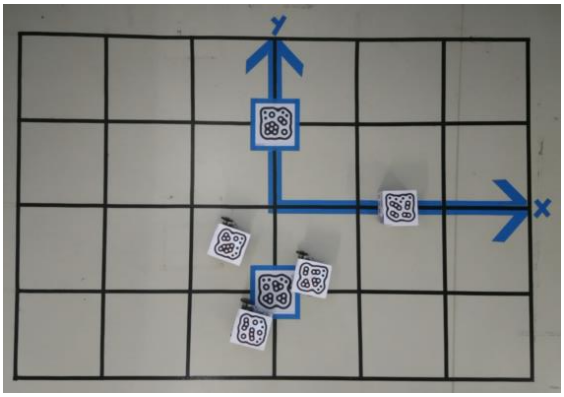
Figura 82. Distancia promedio entre agentes en prueba de seguimiento de trayectoria circular sin obstáculo.

7.3.2 Prueba de trayectoria circular con obstáculo

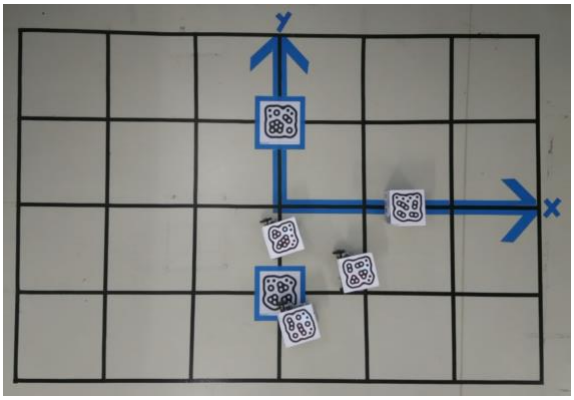
También en esta fase de pruebas con seguimiento de trayectorias se realizó el análisis del comportamiento del enjambre al presentarse un obstáculo en la trayectoria. Para esta prueba se incluyó un obstáculo en las coordenadas (600, 0) y se evaluó el comportamiento del enjambre. El resultado de esta prueba se muestra en la figura 83.



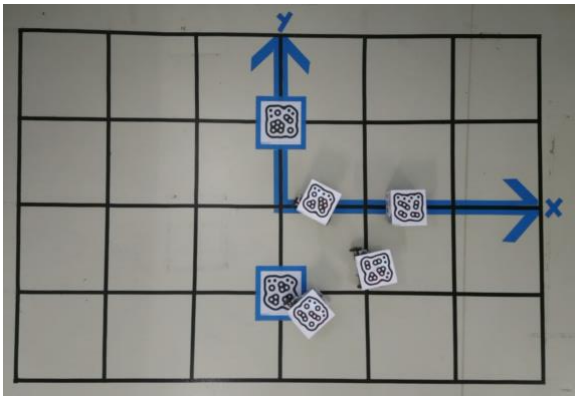
a)



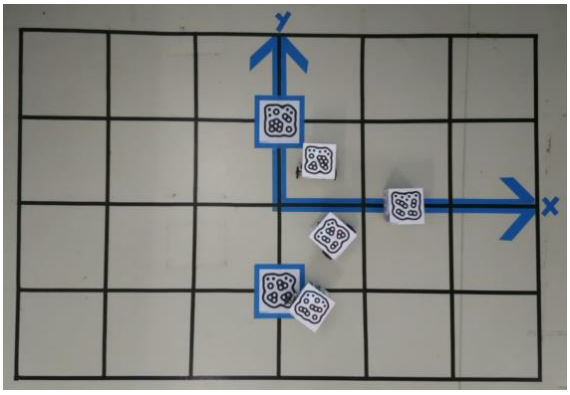
b)



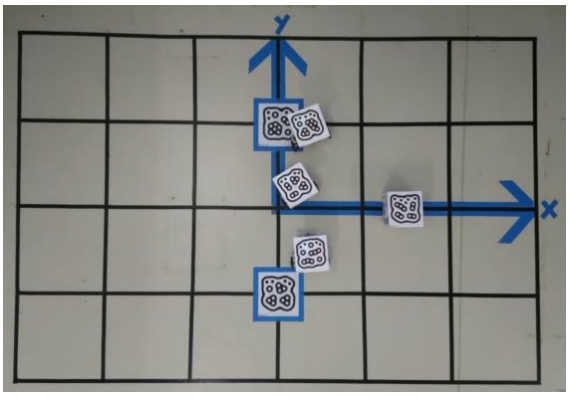
c)



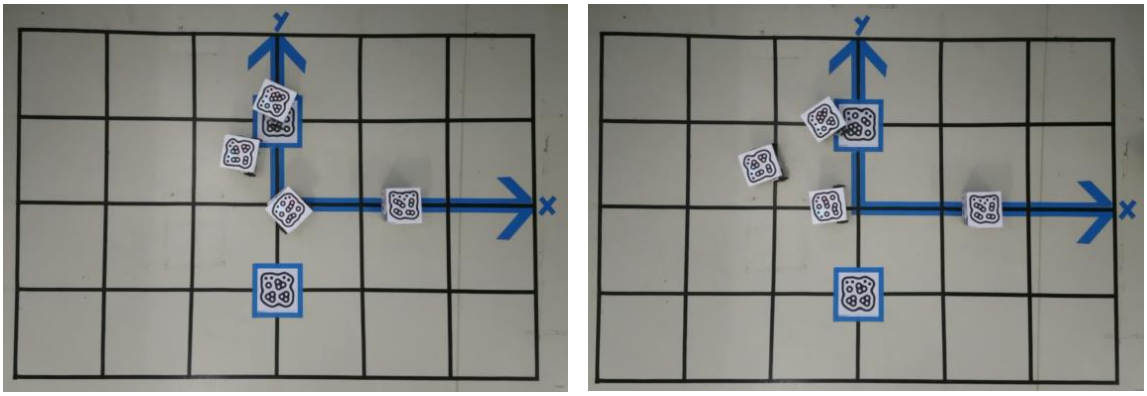
d)



e)



f)



g) h)

Figura 83. Prueba de seguimiento de trayectoria circular con 3 agentes con obstáculo

Como resultado de agregar este obstáculo se ven alteradas todas las trayectorias de los agentes (figura 84). Inicialmente se observa que la trayectoria correspondiente al centroide de la formación (en color amarillo) asemeja una circunferencia, pero se encuentra cortada de la parte hacia donde se localiza el obstáculo. Por su lado, cada una de las trayectorias individuales de los agentes modifica su curso a fin evitar alguna colisión con el obstáculo o con cualquier otro agente. Aquí se muestra, además, que la formación original se ve modificada tras la interacción con el obstáculo, ya que, por ejemplo, considerando las posiciones iniciales y finales de los agentes en la figura 84, se tiene que los agentes mostrados en color verde y rojo intercambian sus posiciones relativas en la formación. Esto respalda el concepto de no tener ninguna formación o posiciones relativas predefinidas, sino que el enjambre adapta su comportamiento a las interacciones con su entorno y entre los distintos agentes.

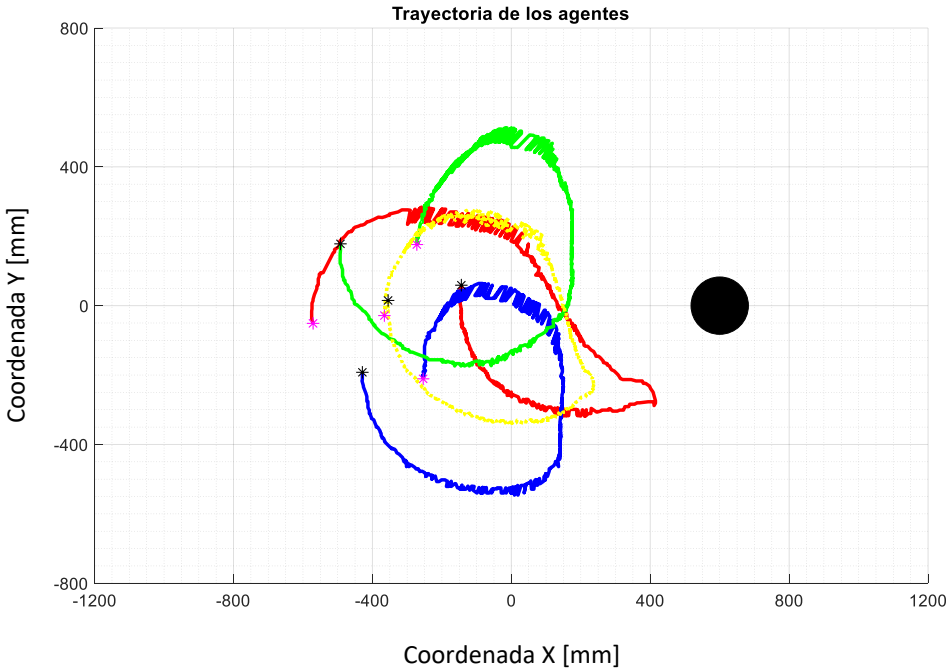


Figura 84. Trayectoria de agentes en prueba de trayectoria circular con obstáculo.

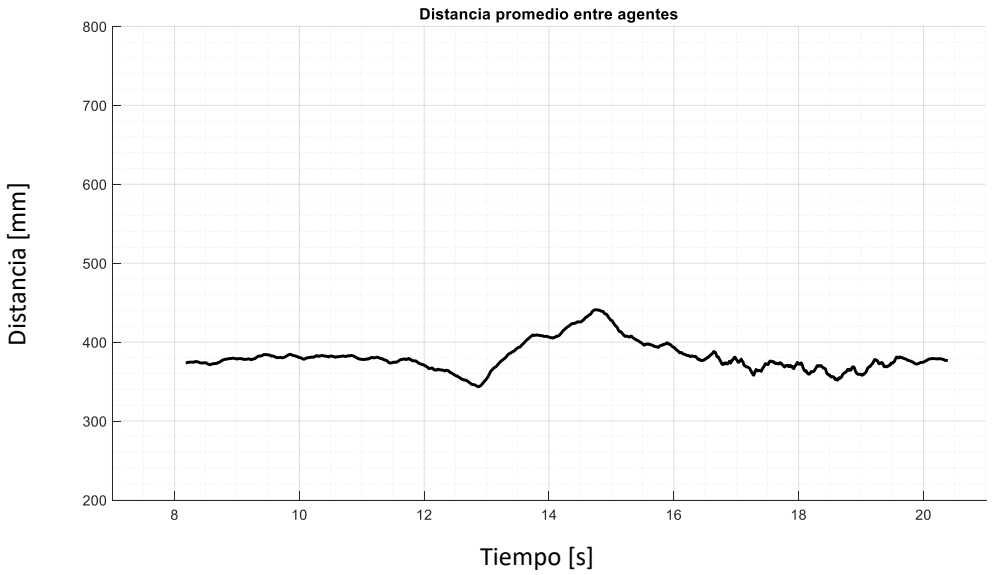


Figura 85. Distancia promedio entre agentes en prueba de trayectoria circular con obstáculo.

También se aprecia el efecto de la interacción con el obstáculo al analizar la distancia promedio entre agentes (figura 85). Esta distancia es similar tanto al principio como al final de la prueba, las variaciones se hacen presentes en el intervalo en el que interactúa el obstáculo con los agentes. Entre los segundos 12 y 16 de la prueba se observa primero una disminución y posteriormente un incremento, para finalmente, retomar la distancia entre agentes de la formación ordinaria.

7.4 Pruebas con evasión de obstáculo dinámico

Otro tipo de prueba realizado consiste en la interacción del enjambre con obstáculos dinámicos que se encuentran en movimiento. Aquí tenemos el caso de que se presente algún elemento móvil dentro del espacio de interacción de los agentes y sean ellos quienes deban desplazarse para evadir colisiones. Con el fin de representar un movimiento del obstáculo totalmente independiente se realizó su movimiento de forma manual como se muestra en la figura 86. Para esta prueba se consideró el objetivo sobre el origen del sistema de referencia, de tal forma que los agentes llegaran a una formación en torno a este punto, y, a continuación, se alteró el sistema desplazando el obstáculo a través del espacio de trabajo. Como respuesta a esta intrusión los agentes reaccionan desplazándose para evitar colisiones, pero posteriormente, en cuanto es posible, regresan a posicionarse en torno al objetivo que se mantiene en las coordenadas (0,0) durante toda la prueba.

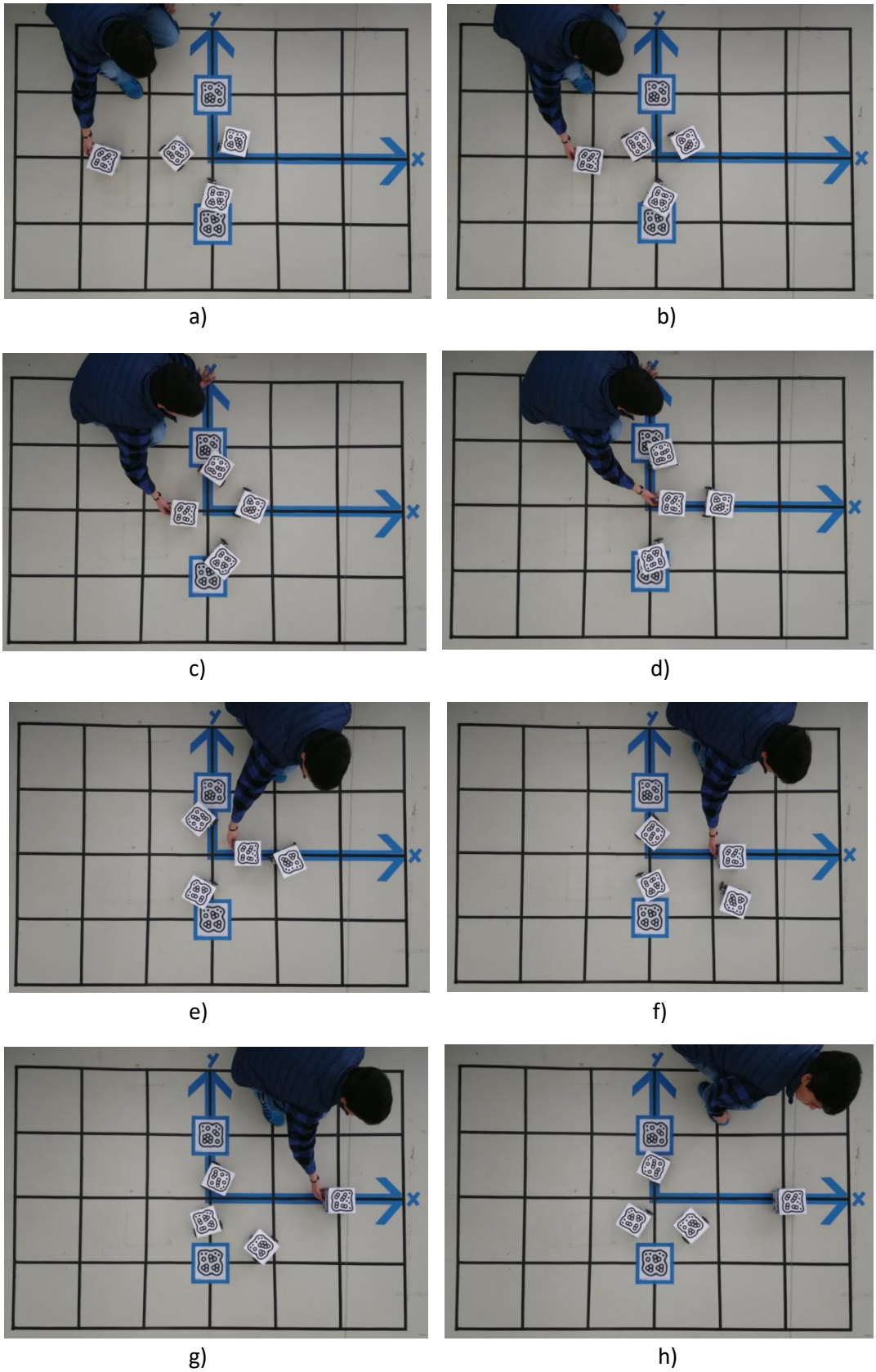


Figura 86. Prueba de formación de 3 agentes con objetivo en (0,0) y obstáculo dinámico.

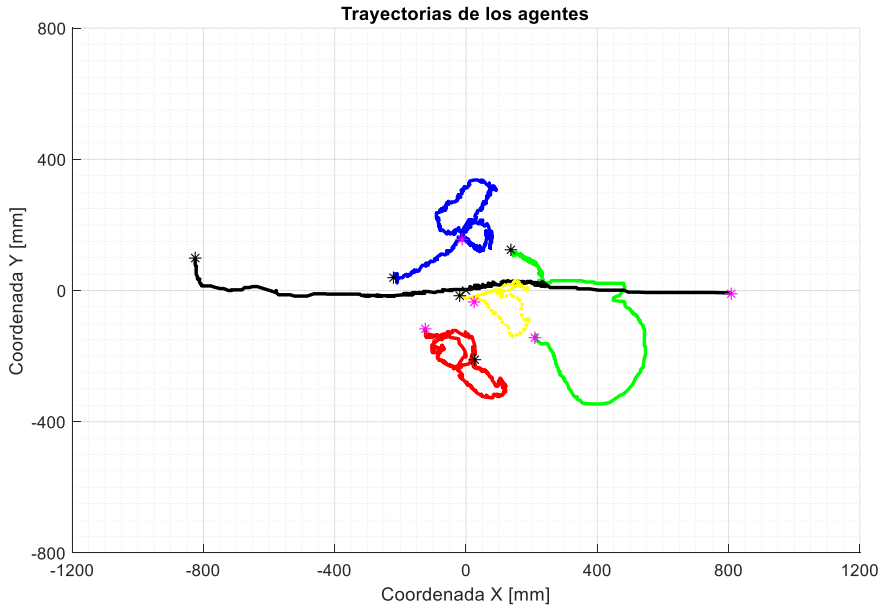


Figura 87. Trayectorias de agentes en prueba de formación con obstáculo manual.

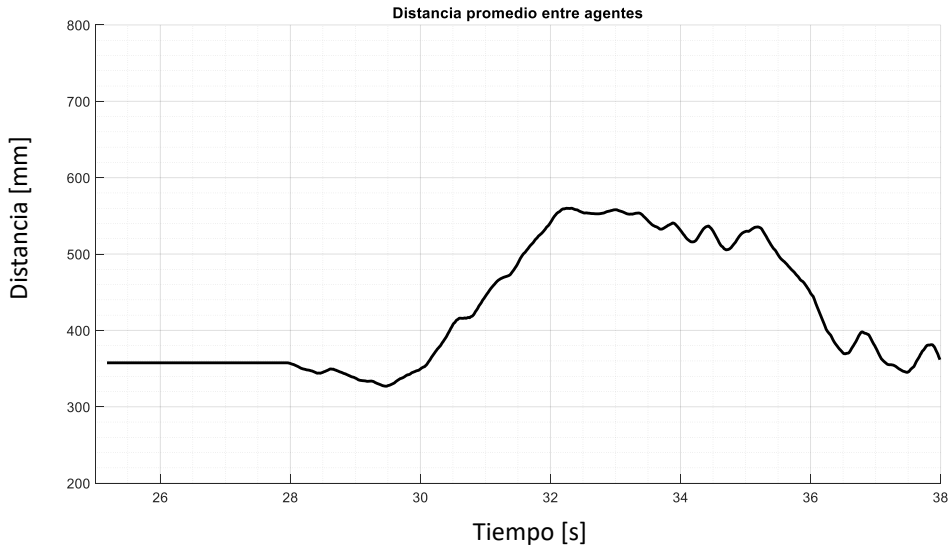


Figura 88. Distancia promedio entre agentes en prueba de formación con obstáculo manual.

En la gráfica correspondiente a las trayectorias se observa el comportamiento en el que los agentes sedesplazan para evadir al obstáculo en movimiento, regresando al objetivo una vez que el obstáculo se aleja. Por otro lado, en la gráfica de la distancia promedio entre agentes, se presenta un incremento en la separación entre ellos cambiando de 360 [mm] a 550[mm] de distancia en el momento en que se atraviesa el obstáculo en la formación, esto atiende al hecho de que los agentes se separan para evadir colisiones con el obstáculo dinámico y, posteriormente, vuleven a acercarse entre sí una vez que el obstáculo se ha alejado.

8. Discusión de resultados

8.1 Validación y funcionamiento del algoritmo planteado

Una vez realizadas las pruebas reales en el banco de pruebas, se procedió a realizar los mismos experimentos, bajo las mismas condiciones, en el espacio simulado computacionalmente. Para comparar dichos resultados no resultó factible la obtención de estimadores estadísticos directos dado que los tiempos de simulación y de ejecución diferían grandemente debido a condiciones de ajuste y periodo de inicio en las pruebas reales. A su vez, coordinar los tiempos de muestreo resultaba complejo debido a que era necesario coordinar la simulación en computadora y lo que sucedía en el banco de pruebas. Debido a lo anterior, se decidió utilizar tres gráficas para evaluar cualitativamente el comportamiento relativo entre lo real y lo teórico partiendo de tres variables fundamentales para el análisis del algoritmo:

1. *La trayectoria de los agentes.* Esta variable resulta importante debido a que la trayectoria de los agentes se ve directamente influida por el comportamiento planteado en el algoritmo propuesto. Si las condiciones de simulación son idénticas al caso real y se trata del mismo algoritmo, las trayectorias deben parecerse para poder validar el modelo en la teoría.
2. *Distancia promedio de los agentes.* Como se vio en las pruebas anteriores, el algoritmo y la modularidad en la programación de los agentes provocan que éstos siempre tiendan a encontrarse a cierta distancia determinada unos de otros, de tal manera que se pueda generar la formación sin colisiones entre los elementos involucrados. Debido a ello, para verificar el funcionamiento de las zonas de influencia y la correcta aplicación de los umbrales de frontera, se toma como medida de funcionamiento la distancia entre los agentes que, en teoría, debería mantenerse constante una vez alcanzada la distancia de equilibrio resultante de la interacción de fuerzas de atracción, orientación y repulsión en las respectivas zonas generadas por cada uno de los agentes involucrados.
3. *Distancia del centroide al objetivo.* Esta medida de funcionamiento del modelo se define como la distancia entre el promedio de las coordenadas de cada uno de los agentes en (X, Y) (denominada a lo largo del trabajo como centroide de la formación) y el objetivo. Esta medida se encarga de evaluar la velocidad de acercamiento del enjambre completo al objetivo planteado. Es importante mencionar que, tanto para el modelo real como para el simulado, esta medida de funcionalidad sólo aplica para el caso en el cual el objetivo se trata de un par coordenado no variable a lo largo del tiempo debido a que, en el caso de la evaluación de la formación con trayectoria, el objetivo estará siempre en movimiento durante el transcurso de la prueba.

Una vez mencionadas y explicadas las medidas de funcionalidad que se verán a continuación, se procede a presentar la comparación del modelo real (lado izquierdo) contra el modelo simulado computacionalmente (lado derecho) en las siguientes figuras con su respectivo análisis.

En la figura 89 se puede observar la trayectoria de los tres agentes involucrados en el experimento. A simple vista es posible afirmar que el parecido entre ellas es muy grande, a excepción que en el caso de la simulación computacional se tratan de líneas completamente rectas en dirección al objetivo, mientras que, en el caso de la prueba real, se pueden observar variaciones debido principalmente a lo caótico del algoritmo así como efectos dinámicos no considerados en la

simulación como es el caso de la fricción y la inercia de los agentes robóticos, que finalmente, termina en la ordenación de los agentes en torno al objetivo. Con esto, es posible afirmar que el algoritmo funciona correctamente de manera cualitativa para poder llevar a los agentes de un punto A hasta un punto B, utilizando como método de validación la simulación por computadora con una desviación mínima.

Es posible observar que los agentes, al momento de aproximarse al objetivo, tanto en la simulación como en el caso real tienden a acercarse entre sí, lo cual también hace hincapié en el efecto latente de la velocidad de atracción de los agentes hacia el objetivo, manteniendo una formación constante a lo largo de su trayectoria hasta llegar cerca del destino final, donde la formación tiende a cerrarse para concentrar el centroide de la formación en el objetivo planteado.

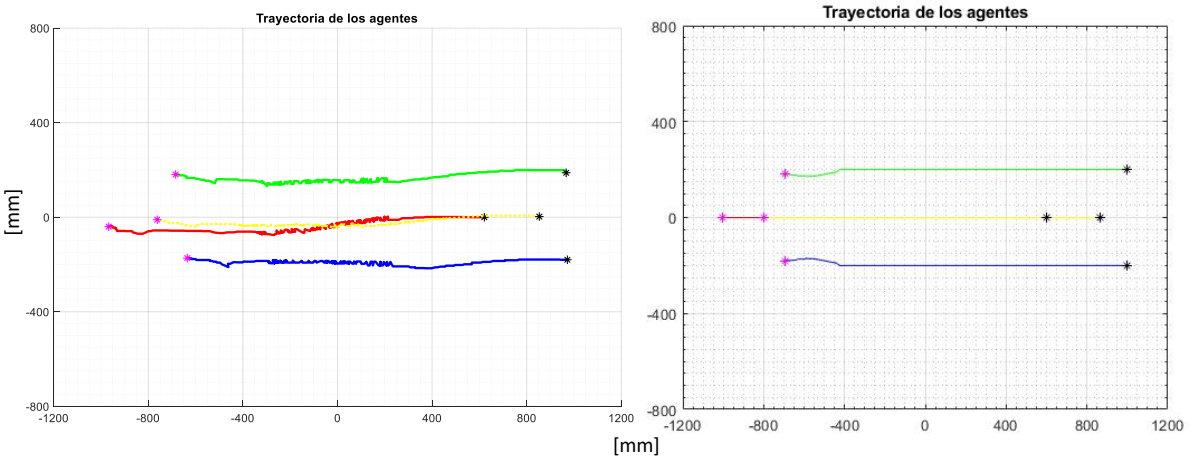


Figura 89. Comparación de prueba de punto fijo a punto fijo. Real contra simulación.

En el siguiente par de gráficas, que se pueden observar en la figura 90, es posible notar que existen dos partes de la gráfica diferenciadas por una curvatura descendente poco después de la mitad de la ejecución completa del experimento (segundo 25 aproximadamente en el caso real y segundo 4.5 en el caso de la simulación) dicha curvatura marca un descenso pronunciado en la distancia entre agentes que presumiblemente, derivado de la gráfica de la trayectoria de los agentes, se debe al momento en el cual comienzan a interactuar con el objetivo y entre sí, acercándose lo máximo permitido por las mismas condiciones de repulsión entre ellos y atracción al objetivo; marcando por tanto, dos distancias bien diferenciadas: la primera, que se mantiene a lo largo de la trayectoria de los agentes desde su punto de origen hasta poco antes de llegar al objetivo y la segunda, que marca la llegada al objetivo y el acomodo de los agentes en formación alrededor del mismo. Dichas distancias se observan en la siguiente tabla:

Tabla 9. Parecido entre la distancia promedio de agentes entre simulación y prueba real

	Simulación	***Prueba real***	***Error relativo de la prueba real***
Distancia entre agentes antes del objetivo	430 mm	420 mm	2.3 %
Distancia entre agentes al llegar al objetivo	360 mm	365 mm	1.3 %

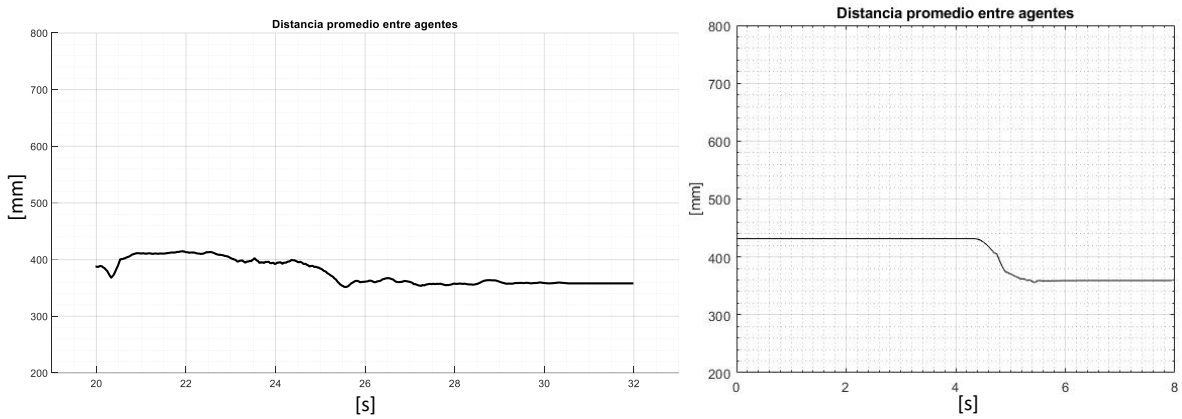


Figura 90. Comparación de la gráfica de distancia promedio entre agentes de la prueba física (izquierda) contra la simulación (derecha).

Como es posible observar en la tabla 9, la aproximación de la prueba real a la simulación teórica difiere un máximo del 2.3% en relación con la distancia entre agentes, lo cual significa, que el modelo presenta el comportamiento esperado aun en la implementación física en el banco de pruebas. Se comprueba también que se cumple satisfactoriamente la tarea de evitar las colisiones durante la ejecución de la trayectoria de los agentes a lo largo de campo de acción, por lo cual, se afirma que el modelo es efectivo bajo las condiciones de esta prueba.

Por otro lado, en la figura 91 se pueden observar dos tendencias muy parecidas entre sí, debido a que, como se menciona en la sección 7, se pueden encontrar dos comportamientos identificables: una sección rectilínea que indica una velocidad de acercamiento constante; y, por otro lado, al llegar al objetivo comienza a ralentizarse el avance. Dicha reducción en la velocidad de avance de la formación al objetivo, en el caso de la prueba física, se puede observar en el instante 26.5 [s] aproximadamente, mientras que en el caso de la simulación computacional se observa en el instante 5.2[s]. Esta diferencia en los tiempos se debe a que la prueba en la realidad no da inicio exactamente en el segundo 0 y se llevan a cabo los desplazamientos con distintas velocidades, por lo cual, se puede analizar cuál es la duración total del experimento. El tiempo de duración del experimento en la prueba real, desde el inicio del movimiento hasta alcanzar el estado de equilibrio en el objetivo, resultó ser de 12 segundos, mientras que para la simulación el tiempo fue de 8 segundos.

En general se observa que el experimento real tarda más en ejecutarse en comparación con la simulación computacional. La razón de este comportamiento es atribuida directamente a que en el caso de la simulación computacional no se consideran todos factores dinámicos involucrados en las pruebas y a que se trata de una situación idealizada. Por lo cual, a pesar de las diferencias en tiempos, es posible afirmar que el algoritmo funciona adecuadamente ya que de manera general los agentes se desplazan en dirección al objetivo en ambos casos y llegan en las dos situaciones a un estado estable en un tiempo finito, aunque no igual.

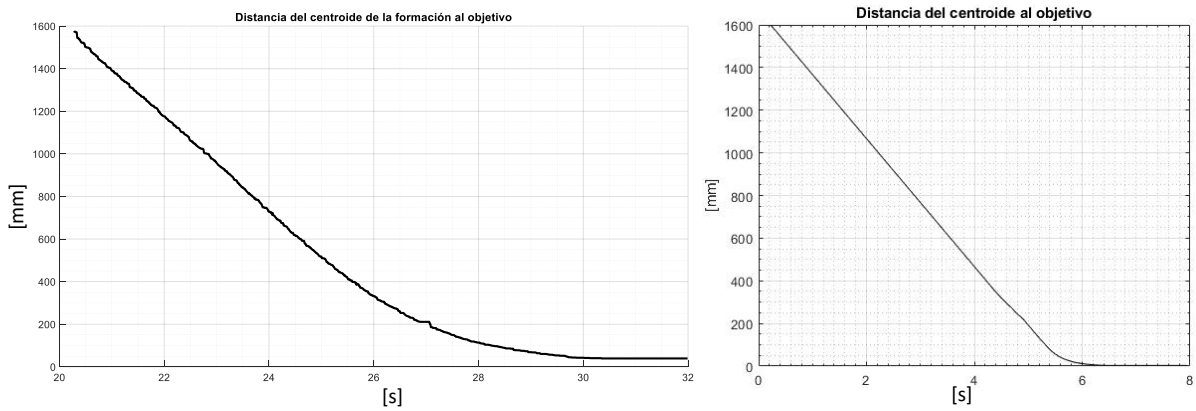


Figura 91. Distancia del centroide de la formación al objetivo. Prueba real (izquierda) y simulación computacional (derecha).

Otro caso que se ejemplificó en el simulador fue cuando se tiene un objetivo fijo con un obstáculo en el camino. Dicha trayectoria se puede observar en la figura 92, en donde la similitud entre la simulación y el experimento físico es evidente cualitativamente; haciendo un análisis cuantitativo se puede observar, en la figura 93, que en la prueba física se obtuvo un pico de distancia de hasta 490 [mm] mientras que en la simulación se obtuvo un máximo de 430 [mm] cuyo valor va en descenso desde el inicio de la prueba. A su vez, la gráfica correspondiente a la distancia entre agentes presenta magnitudes muy similares tanto en el inicio de la prueba como al final de ésta en comparación con la prueba similar realizada sin obstáculo. Sin embargo, al analizar a detalle el comportamiento resultan ser considerablemente diferentes durante la ejecución del experimento, principalmente al momento de evadir el obstáculo, en el cual, la distancia entre agentes se incrementa al adaptarse la formación. Al comparar las curvas que se presentan en la prueba real y la simulación se observa que son significativamente diferentes, sin embargo, ambas representan un comportamiento característico al momento de llegar al obstáculo para lograr evadirlo y llegar finalmente al objetivo establecido. Esto también hace evidente que, a pesar de utilizarse el mismo sistema y los mismos parámetros, dos pruebas no serán iguales debido a la naturaleza caótica del experimento. Dicha naturaleza provoca que cualquier variación, ya sea interna del sistema (como diferente torque en las llantas o factores variables de la dinámica del robot) o externa (como la resolución o la exposición de la cámara usada para detectar a los agentes en el espacio), impacte significativamente el comportamiento del sistema, aun a pesar de cumplirse para ambos casos los mismos objetivos de las pruebas, como es el caso de la prueba que se presenta en la figura 92 y figura 93.

Ahora bien, en la figura 94 es posible observar un comportamiento muy parecido al encontrado en la figura 91. Esto debido a que, como los agentes se encuentran programados de manera modular con parámetros idénticos de funcionamiento, éstos tenderán siempre a un comportamiento simétrico entre ellos, además, esto se refleja en la posición del centroide de la formación. Por lo tanto, es posible concluir que, sin importar las condiciones del entorno, la gráfica de aproximación del centroide de la formación al objetivo será similar entre los diferentes casos, siempre y cuando, se establezca un objetivo fijo. Esta situación no se presenta cuando se trata de un objetivo en movimiento, como en el caso de las trayectorias en las cuales la formación nunca alcanza como tal al objetivo ya que éste último siempre se encuentra en movimiento.

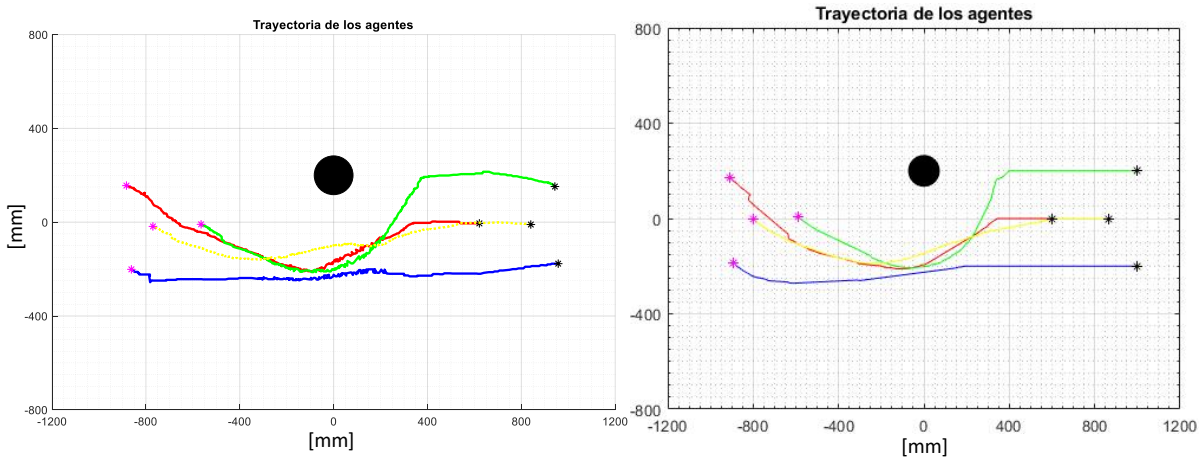


Figura 92. Trayectoria de los agentes con objetivo fijo con un obstáculo en el camino. Prueba física (izquierda) y simulación (derecha).

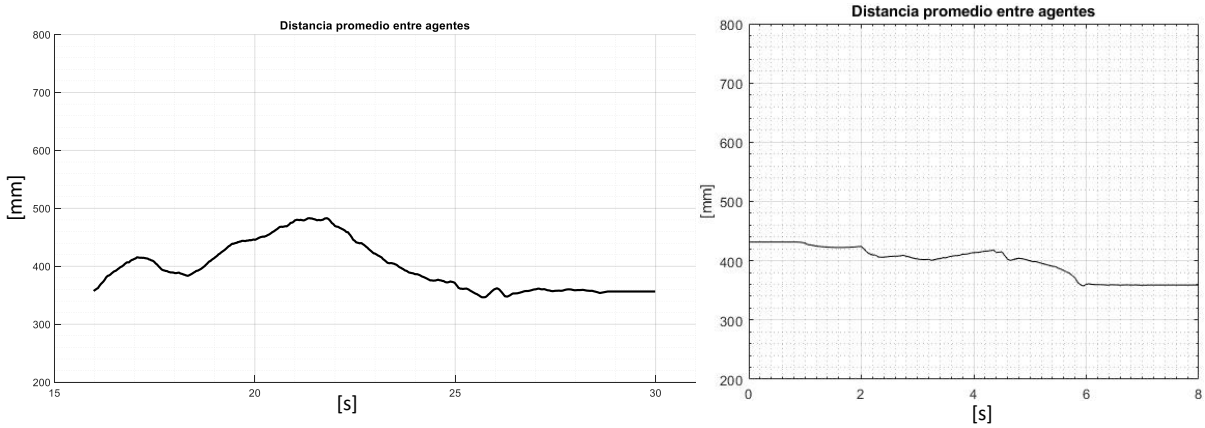


Figura 93. Distancia promedio entre los agentes durante la prueba. Prueba física (izquierda) y simulación (derecha).

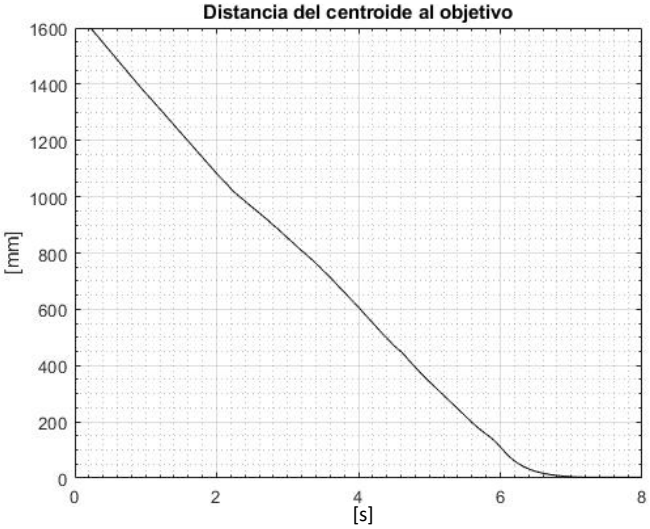


Figura 94. Distancia del centroide de la formación en la prueba de trayectoria recta con obstáculo en la simulación.

Otro aspecto importante por analizar es el experimento de la trayectoria circular, contra el mismo experimento planteado en la simulación computacional. En dicho experimento es posible observar cómo la trayectoria del centroide (trayectoria amarilla) describe una circunferencia tal como se estableció con la trayectoria del objetivo, además de tener las mismas dimensiones que en el caso de la simulación por computadora. Ya que se configuró una trayectoria circular en el objetivo, el resultado de esta prueba hace considerar y validar la posibilidad de realizar el seguimiento de trayectorias a partir del mismo modelo generado, no solamente de trayectorias circulares, sino de prácticamente cualquier trayectoria que pueda ser programada y que de suficiente tiempo a los robots de seguirla correctamente.

Ahora bien, refiriéndose al experimento con obstáculo, se nota que se trata de un comportamiento similar en términos generales, pero que presenta variaciones en el control de cada uno de los agentes robóticos. La circunferencia que se genera con el centroide deberá ser similar a la trayectoria indicada. Aunque, al analizarse con mayor detalle, se puede observar que en el caso de la prueba física los agentes en rojo y azul intercambian lugares para colocarse finalmente en una posición diferente a la inicial. Por otro lado, en el caso de la simulación por computadora se puede observar que no existe ese intercambio de lugares entre el agente rojo y azul, lo cual se debe a la naturaleza del algoritmo propuesto que, en contraposición con el arreglo de malla (algoritmo deliberativo), ahorra recursos y tiempo en la ejecución, al mismo tiempo que genera un algoritmo robusto y escalable en la cantidad de agentes involucrados.

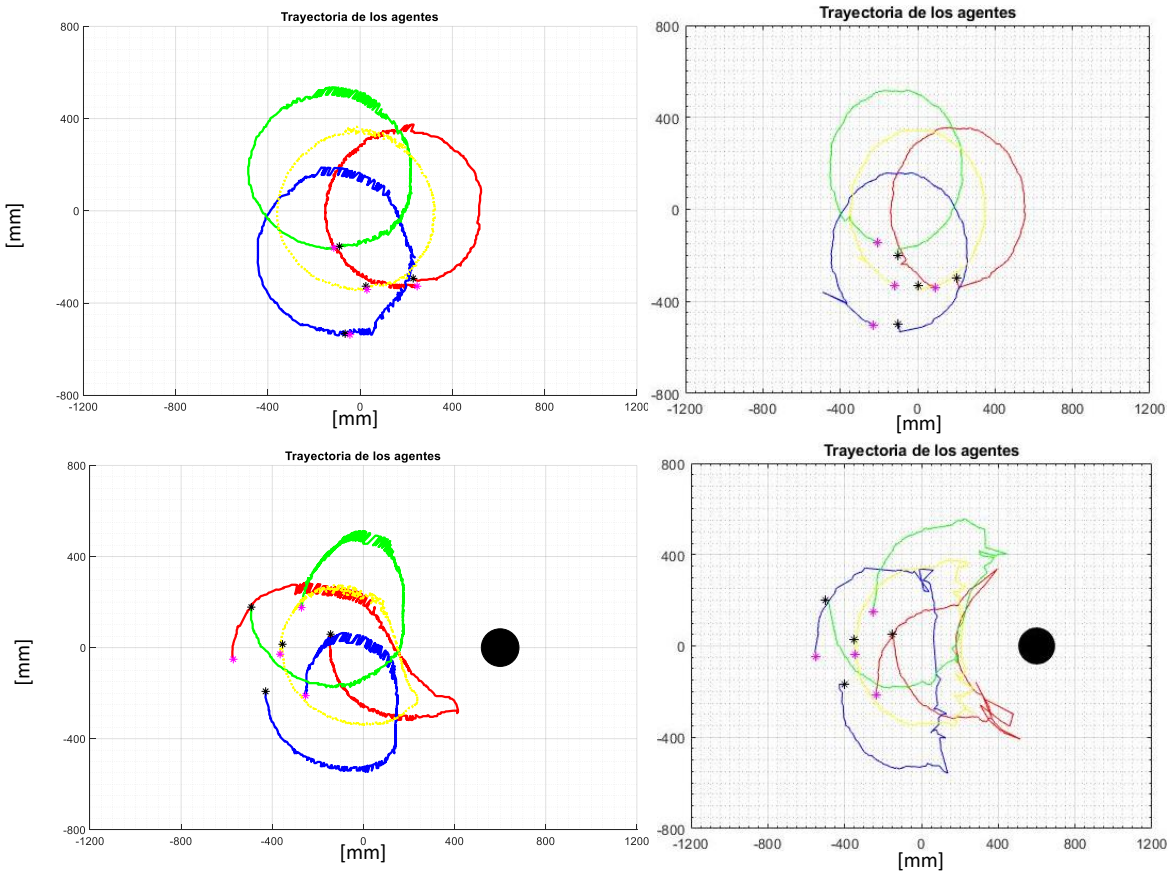


Figura 95. Trayectorias en la prueba real (izquierda) sin obstáculo (arriba) y con obstáculo (abajo), comparadas contra sus similares en la prueba computacional (derecha).

Con lo observado es posible validar tanto el modelo teórico en la práctica como la confiablidad del modelo en la simulación, ya que es posible predecir con alta precisión el comportamiento que tendrán los agentes a lo largo del tiempo utilizando el método propuesto.

En ninguno de los casos anteriores se observa colisión a pesar de lo cercanas que puedan estar las trayectorias de los agentes, por ejemplo, en el caso de la evasión de obstáculos de punto a punto fijos, se observa que los agentes pasan prácticamente por la misma región, pero gracias al comportamiento analizado en la sección 7 de atracción, repulsión y orientación los agentes son capaces de identificarse y organizarse para evadir eficientemente el obstáculo de forma reactiva sin colisionar, como lo muestra la gráfica de atracción de agentes (figura 93) que demuestra que siempre se encuentran a un mínimo de aproximadamente 350 [mm] entre sí.

8.2 Desempeño y cumplimiento del objetivo

Al analizar los resultados obtenidos de la formación de los agentes con respecto a sus trayectorias en la prueba de punto fijo a punto fijo, se midieron las distancias que mantiene el centroide de la formación con respecto al objetivo planteado en el experimento, tanto en el caso de la simulación por computadora como en el de la prueba física. Dichos resultados se muestran en la tabla 10.

Tabla 10. Distancia del centroide al objetivo.

Distancia del centroide al objetivo (real)	***Distancia del centroide al objetivo (simulación)***
39.5 mm	3.1 mm

Este valor de distancia se puede referenciar a la distancia promedio entre los agentes para así poder ponderar el valor de error que existe en cada una de las pruebas. Ahora bien, tomando en cuenta que en el caso de la prueba física se obtuvo un valor de distancia promedio entre los agentes de 365 [mm], la distancia del centroide al objetivo representa un 10.8% de este valor. Por otro lado, en el caso de la simulación por computadora, se obtuvo una distancia promedio entre agentes de 360 [mm], es decir que la distancia de 3.1 [mm] entre el centroide y el objetivo en dicha simulación representa un 0.86%. Con estos valores es sencillo concluir que el caso de la simulación por computadora tiende a resultar en un mejor desempeño del algoritmo, esto debido a que no se consideran, como se ha mencionado anteriormente, condiciones dinámicas de los agentes involucrados, así como la reacción que tiene al impulso los motores de corriente directa al ser accionados.

En el caso de las pruebas de trayectoria circular se encontró que la trayectoria marcada por el centroide de la formación, como se mencionó anteriormente, se trata de una circunferencia de menor tamaño a la que debería haberse trazado teóricamente dada la definición de esa trayectoria. Como se mencionó en la sección pasada, esto se debe a la velocidad con la que el objetivo cambia de posición, aunque también es posible evaluar cuantitativamente dicho error con la circunferencia generada contra la que teóricamente debió haberse generado para evaluar el desempeño del algoritmo. Considerando que el objetivo se programó para generar una trayectoria circular de radio

de 500 [mm], completando dicha trayectoria cada 12.6 [s], aproximadamente, se obtuvo una circunferencia generada por el centroide de la formación de 350 [mm]. Esto significa que existe un error del 30%. Si bien, se logra trazar una trayectoria circular con el enjambre robótico, en el caso de buscar una mayor exactitud del seguimiento de ésta es necesario aumentar el tiempo en el que se genera alrededor del círculo para dar tiempo a los agentes de posicionarse antes de volver a cambiar el objetivo de posición. Al mismo tiempo, es importante mencionar que, de tratarse de una trayectoria más amplia, los agentes deben aumentar la velocidad para poder generar en forma precisa.

8.3 Ventajas y resolución de problemas

Si bien los sistemas multiagente requieren un modelo de control más complejo que regule las interacciones entre los diferentes elementos que conforman el grupo, como un control de lógica difusa o un control adaptable, también ofrece diversas ventajas sobre los sistemas con un único agente. Por ejemplo, en el ámbito de la movilidad, el hecho de contar con un conjunto de robots más pequeños capaces de adaptar su formación al ambiente en el que se desarrollan permite tener una mayor accesibilidad en comparación con el uso de un único robot de mayores dimensiones. Aunado a ello, el manejo de múltiples agentes hace posible la distribución del trabajo, resultando en un mejor desempeño al momento de realizar tareas como la exploración o vigilancia, que requieren abarcar áreas extensas, o el desplazamiento de materiales o equipo de una zona a otra, especialmente para lugares de difícil acceso. Con el modelo propuesto se pretende encarar y dar solución al problema del desplazamiento en conjunto de un enjambre robótico y la metodología propuesta, como se ha desarrollado a lo largo del trabajo, es mediante el uso de inteligencia de enjambre con el algoritmo de *flocking* en combinación con el modelo de campos potenciales.

Tras analizar las pruebas realizadas, se concluye que el modelo propuesto resuelve correctamente la tarea de desplazar al enjambre robótico hasta el punto objetivo. A pesar de presentarse una pequeña diferencia entre el objetivo establecido y la posición final del enjambre, esta diferencia es mínima e incluso se observa que el punto objetivo se encuentra dentro del espacio abarcado por el enjambre. En las secciones anteriores se analizó esta diferencia y se observa también que en el caso de las simulaciones esta diferencia es aún menor que la registrada en las pruebas con los robots físicos, por lo que, mejorando la implementación del sistema de adquisición de datos y el sistema de motorización de los robots podría reducirse este error y desarrollarse una exactitud mayor. Además, para aplicaciones más concretas, esta función de desplazamiento del enjambre sería un paso intermedio de la realización de las tareas de los robots, ya que una vez que llegan al objetivo procederían a ejecutar la tarea correspondiente en dicho espacio. El modelo implementado hace posible una aplicación descentralizada e independiente del control de desplazamiento del enjambre, por lo que el sistema no requerirá de un robot o elemento adicional que se encargue de dirigir al resto del grupo, dando así a cada uno de los agentes la posibilidad de actuar tanto en forma individual como en grupo. Esto resulta fundamental para tareas que impliquen una separación temporal del enjambre en las que cada agente debe ser capaz de actuar correctamente sin importar si el líder se encuentra presente o no. Así mismo, se resuelve el problema de la escalabilidad del sistema ya que, por la naturaleza del comportamiento, es posible anexar o retirar agentes del grupo

sin que éste se vea afectado. Por las observaciones realizadas con respecto a la separación entre agentes es posible que al aumentar o reducir considerablemente el número de agentes en el grupo sea necesario un reajuste en los parámetros del sistema, debido a que al incrementarse el número de agentes estos se aproximan más entre sí como consecuencia del mayor número de interacciones entre ellos. Sin embargo, se comprobó que el modelo es válido para una cantidad variable de agentes. En el caso de que se pretenda implementar el sistema en grupos con cambios significativos en el número de elementos, podrían establecerse parámetros variables que cambien en función del número de agentes externos detectados por el agente local, así se tendría un sistema adaptable al número de agentes que conforman el grupo.

En cuanto a la evasión de obstáculos, resulta efectiva la aplicación de campos potenciales para regir el comportamiento de los robots. En las pruebas analizadas se muestra que los agentes son capaces de continuar su desplazamiento hasta el objetivo sin colisionar con los obstáculos. De esta forma se solucionan los problemas de colisiones simultáneamente, tanto por parte del obstáculo como con el resto de los agentes. Una consideración importante en esta parte es el hecho de que los obstáculos pueden presentar diferentes dimensiones y formas. En las pruebas realizadas con obstáculos se consideraron estos como elementos en posiciones bien definidas y con un radio fijo, esto principalmente por la facilidad de considerarlo así en conjunto con el sistema de visión implementado (que en realidad solamente detecta el centro del *fiducial* como posición), sin embargo, en ambientes reales se presenta también la posibilidad de que los obstáculos no tengan siempre el mismo tamaño y que puedan ser de una gran variedad de formas distintas. Al implementarse completamente la localidad del sistema, en la que cada agente tenga montado su propio sistema de sensores para adquisición de datos y su propia unidad de procesamiento, se haría posible la detección de cualquier tipo de obstáculo independientemente de su forma y tamaño. El concepto está basado en que el agente detectará aquellas direcciones hacia donde se encuentre el o los obstáculos y con base en ello y las distancias de cada uno sea capaz de trazar su propio campo potencial que le indique hacia donde desplazarse. De esta manera, el comportamiento estará basado en la detección de los bordes de los obstáculos que se encuentren al alcance del agente y no en el conocimiento completo del obstáculo, abriendo la posibilidad de que éste pueda ser de mayor tamaño o de forma distinta sin que se vea comprometido en funcionamiento del modelo. También esto hará que el agente vaya actuando de forma reactiva conforme va detectando la frontera del obstáculo aplicando la misma idea de campos potenciales y actualizándola dinámicamente conforme a su entorno.

Otra situación que se presenta con la evasión de obstáculos utilizando campos potenciales, que ha sido reportada numerosamente en trabajos realizados en el tema [8], [9], [46] y fue tratada en la sección 4.3 de este trabajo, es la presencia de mínimos locales. Si bien la aplicación del algoritmo de *flocking* no resuelve completamente el problema de los mínimos locales, sí puede llegar a resolver esta situación para casos específicos en los que el comportamiento de enjambre es suficiente para sacar a alguno de los agentes de un mínimo local. Un ejemplo concreto se presenta analizando las pruebas siguientes.

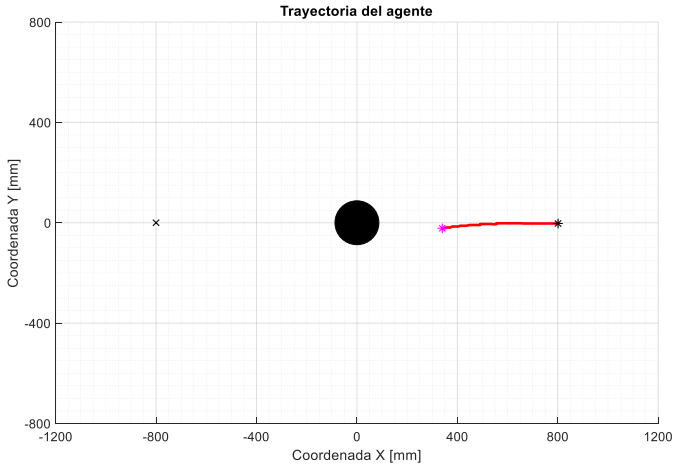


Figura 96. Fenómeno de mínimo local en prueba con un agente y un obstáculo.

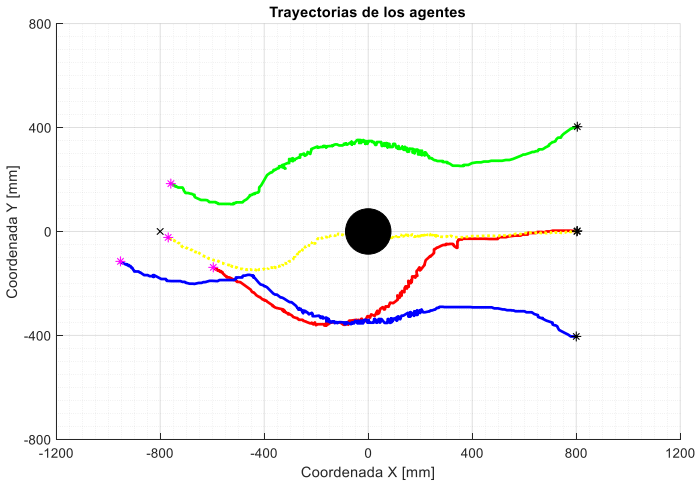


Figura 97. Escape de mínimo local con enjambre de 3 agentes alineados.

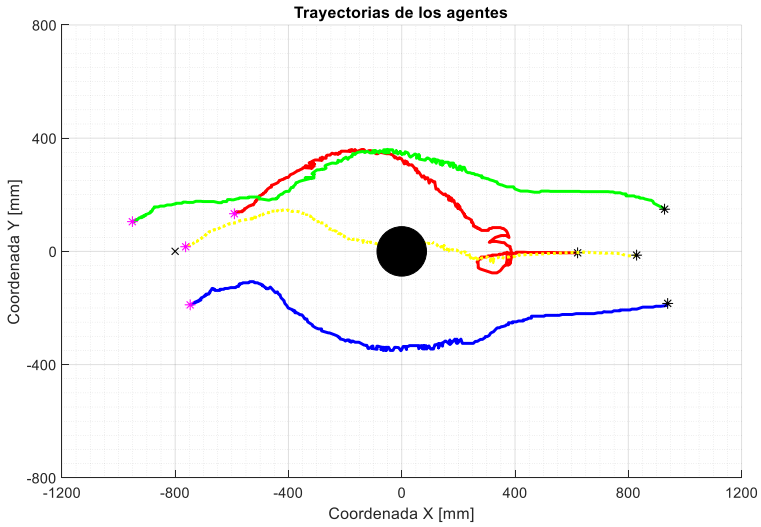


Figura 98. Escape de mínimo local con enjambre de 3 agentes en formación.

En la figura 96 se muestra una prueba en la que se presenta el problema de un mínimo local. Al estar alineadas la posición inicial del agente, el obstáculo y el objetivo, la velocidad atractiva del objetivo llega a compensarse con la velocidad repulsiva del obstáculo por lo que el agente se queda estancado antes del obstáculo sin llegar al punto objetivo. En la figura 97 y figura 98 se muestra como al introducir un mayor número de agentes en el sistema y aplicar el modelo de *flocking* el agente que había quedado estancado en la prueba individual consigue evadir el obstáculo y desplazarse hasta llegar al objetivo. Esto es consecuencia de las interacciones que se dan entre los agentes, las cuales logran desplazar al agente en cuestión y sacarlo del mínimo local para que pueda continuar con su desplazamiento. De esta forma es posible que el comportamiento en conjunto del enjambre resuelva condiciones de mínimos locales, en general. Cuando un agente se estanque en un mínimo local, las variaciones en las velocidades de interacción con los demás agentes podrán sacarlo de dicho punto. A pesar de ello, si un mínimo local ejerce el suficiente efecto para estancar al enjambre completo, es decir, a todos los agentes que lo conforman, éste no tendrá forma de salir del mínimo local. Incluso puede darse el caso en el que la mayoría de los agentes se queden estancados y lleguen a influenciar lo suficiente a otro agente de tal forma que éste tampoco sea capaz de continuar su desplazamiento hacia el objetivo, aun cuando su trayectoria no pasaba directamente por el mínimo local.

Una de las desventajas que se observaron en la implementación de esta estrategia de control de enjambres es que, en comparación con otros trabajos, por ejemplo en el caso de [33], el autor es capaz de modificar la topología de la formación con los cambios de ciertos parámetros en la ejecución del experimento, alterando así, la forma en la que interactúan los agentes en el entorno. A su vez, es importante mencionar que en este trabajo no existe un control de trayectorias o alguna forma de controlar a los agentes en su conjunto.

Otro de los trabajos en los que se pueden encontrar algunas similitudes con respecto a lo que en esta tesis se presenta es en [34], donde se propone un método de control de enjambres basándose en la biomimética para ordenar a los agentes en formaciones a lo largo de una dirección para representar artificialmente las migraciones de largas distancias. Pero, una ventaja practica notable con respecto a ese trabajo, es que, en el algoritmo propuesto aquí, se puede controlar la dirección y el objetivo puntual de la formación a lo largo del experimento, así como generar trayectorias definidas variando algunos parámetros que definen el algoritmo.

9. Conclusiones

Recordando el objetivo de la tesis, el cual consistía en la obtención de un nuevo algoritmo para controlar enjambres robóticos y generar formaciones adaptables para la evasión de obstáculos, se puede concluir que el objetivo se cumplió exitosamente, debido a que, como se puede mostrar en las secciones 7 y 8, el modelo propuesto fue validado tanto teórica como prácticamente con un banco de pruebas generado a partir del conocimiento previo para la preparación de un ambiente inteligente y la creación de un simulador para la validación del modelo computacional. La formación compuesta por tres agentes robóticos no sólo es capaz de llegar a un objetivo fijo, sino también, de trazar trayectorias definidas por el usuario de forma que el centroide de la formación trace dicha trayectoria, con posibilidades de disminuir el error entre la trayectoria deseada y la trazada por el centroide con algunas consideraciones como lo son la velocidad de los agentes y la velocidad en la generación de la trayectoria en función de las ecuaciones paramétricas que la describen.

También es importante mencionar que, derivado de las consideraciones realizadas para la obtención de este algoritmo, la cantidad de agentes está íntimamente relacionada con el desempeño del experimento, tanto en las pruebas de objetivo fijo como en aquellas de trayectoria dada, esto derivado a que en general, como se explica en la sección 5, dicha cantidad varía la distancia de equilibrio entre los agentes. Es interesante observar que, aunque dicha distancia varía y disminuye a medida que se aumenta la cantidad de agentes, la distancia promedio se mantiene prácticamente sin cambios. Esto resulta en la conclusión de que, a pesar de que se trata de un algoritmo que puede ser escalado a una cantidad n de agentes, también se deben hacer algunos ajustes en las constantes involucradas para evitar que la tendencia a la baja en la distancia mínima entre ellos llegase a provocar colisiones en los mismos o inestabilidad en la formación.

Derivado de lo anterior, también es importante mencionar como conclusión la generación de un método confiable para la validación y sintonización de las constantes de comportamiento del sistema propuesto: la simulación computacional. Ya que, como se pudo observar en los experimentos realizados, la simulación computacional se aproxima considerablemente al comportamiento generado en las pruebas reales, alcanzando hasta una diferencia de 1.3% entre las posiciones finales del experimento físico y el experimento virtual para las pruebas de desplazamiento a un punto fijo que se realizaron en este trabajo. Esto también abre la posibilidad de generar diferentes iteraciones, dependiendo del objetivo buscado, para así lograr una mayor exactitud en la ejecución del experimento y buscar la optimización de las constantes que controlan el comportamiento.

A su vez, al realizar esta simulación computacional es posible verificar que este modelo cumple con las especificaciones necesarias de velocidades y tiempos de estabilización antes de desarrollar estos experimentos físicamente, lo cual ahorra tiempo y recursos en la implementación. Además, es posible extender el modelo desarrollado a otros tipos de agentes y de condiciones de ambiente, ya que se encuentra separada y de forma modular la aplicación del modelo de control y la simulación cinemática del robot móvil, por lo que las adaptaciones y ajustes necesarios se pueden aplicar de forma directa ajustando las ecuaciones correspondientes a la cinemática del tipo de configuración robótica en cuestión.

10. Trabajo a futuro

Siguiendo la línea de investigación del presente trabajo, se abre la posibilidad de continuar desarrollando proyectos y avances tecnológicos en relación con la robótica de enjambre. Particularmente, se propone mejorar y completar la implementación de este sistema con el objetivo de que se le puedan dar aplicaciones concretas orientadas a la resolución de tareas específicas. Adicionalmente, existe una gran cantidad de algoritmos desarrollados en el ámbito de la inteligencia de enjambre con posibilidad de ser adaptados para su aplicación en otros campos como en de la ingeniería y la robótica.

Como se mencionó a lo largo del trabajo, una de las características que se pretende implementar en el modelo propuesto es la localidad y la aplicación de un sistema descentralizado. Debido a limitantes tecnológicas y de recursos en la investigación aquí desarrollada se llevaron a cabo las pruebas implementando esto de forma simulada, es decir, se tuvieron en consideración las limitantes que presentaría un sistema completamente descentralizado y con localidad en los agentes, pero en las pruebas físicas realizadas se utilizó un sistema de visión global para la adquisición de datos y una computadora central para el procesamiento de toda la información.

Para finalizar el desarrollo de este proyecto se propone diseñar e implementar un sistema que cumpla completamente con estas características de la robótica de enjambre. Para ello, sería necesario desarrollar un sistema de adquisición de datos particular para los agentes, de tal forma que cada uno de ellos, por sí mismo, sea capaz de detectar su entorno e identificar al resto de los agentes eficientemente sin requerir algún sistema externo como el sistema de visión utilizado en las pruebas realizadas como parte del proyecto. Esto implicaría, desde luego, un rediseño de los agentes implementados y una investigación más a fondo sobre los posibles sensores a utilizar, teniendo como requerimientos la capacidad de detectar agentes y obstáculos que se localicen a una determinada distancia del agente, poder identificar si se trata de otro agente del grupo o de un obstáculo en el espacio y conocer con cierto grado de precisión las direcciones y distancias a las que estos se encuentran.

Así mismo, la implementación de este nuevo tipo de adquisición de datos permitiría que los agentes robóticos actuaran con mayor libertad de desplazamiento sin estar limitados por el campo de visión de la cámara. Para ello también es necesario que cada robot haga el procesamiento internamente sin la necesidad de conectarse con una unidad de procesamiento central, por lo que posiblemente se requiera introducir una unidad de procesamiento distinta a cada agente. Finalmente, deberá desarrollarse algún método para indicar el objetivo global a cada agente. Esto puede depender de la tarea que se pretende realizar, por ejemplo, para el caso de el objetivo de desplazamiento podría incluirse algún sistema de posicionamiento global que sea quien determine la posición global del agente y del objetivo al que se pretende llegar.

Con estas modificaciones podría lograrse el funcionamiento del sistema de forma totalmente descentralizada y se aumentarían las prestaciones y capacidades del mismo.

Otra posibilidad de trabajo a futuro es la expansión del sistema desarrollado para su posible utilización bajo distintas condiciones. Una opción es la consideración de agentes distintos dentro del enjambre, por ejemplo, con diferentes características geométricas, diferentes capacidades de

sensado y de procesamiento, o incluso con diferentes configuraciones robóticas que puedan implicar una cinemática distinta. El modelo, como está planteado en este proyecto, es independiente de las características físicas de los agentes hasta la parte en la que se utiliza el modelado cinemático para traducir el comportamiento del modelo en parámetros de control directo del robot, por lo que sería posible la implementación del mismo modelo, únicamente adaptándolo para que funcione correctamente con otra configuración robótica, o incluso con una combinación de configuraciones robóticas diferentes en un mismo enjambre. De igual forma es posible una expansión del modelo para su implementación en ambientes en los cuales los agentes pueden desplazarse en tres dimensiones, y no únicamente sobre un plano. Esto resultaría particularmente útil para aplicaciones con agentes como unidades aéreas no tripuladas como es el caso de los drones.

También se propone analizar, adaptar e implementar el modelo ya con aplicaciones concretas para evaluar su desempeño en actividades como exploración, vigilancia y transporte. Además, puede implementarse sobre agentes diseñados específicamente para cada una de estas actividades a fin de desempeñar las tareas respectivas de forma satisfactoria.

Anexos

Anexo 1. Modelado cinemático de los robots

$$q' := \begin{pmatrix} x' \\ y' \\ \theta' \\ \varphi'_{fr} \\ \varphi'_{fl} \end{pmatrix} \qquad R(\theta) := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\omega_{1\_1} := \begin{pmatrix} 0 \\ 0 \\ \theta'_1 \end{pmatrix} \qquad v_{1\_1} := \begin{pmatrix} x'_1 \\ y'_1 \\ 0 \end{pmatrix}$$

$$R_{1\_wr} := R(0) \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R_{1\_wl} := R(0) \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P_{1\_wr} := \begin{pmatrix} -d \\ -b \\ 0 \end{pmatrix} \qquad P_{1\_wl} := \begin{pmatrix} -d \\ b \\ 0 \end{pmatrix}$$

La llanta derecha velocidades por propagación

$$\omega_{r\_r} := R_{1\_wr}^{-1} \cdot \omega_{1\_1} + \varphi'_{fr} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ \varphi'_{fr} \\ \theta'_1 \end{pmatrix}$$

Se ocupa la Y en el vector final porque en ese eje se hace el giro de la llanta

Normalmente usamos el ángulo o velocidad angular de la junta, aquí es la velocidad de la llanta

Se observa que la velocidad angular en los diferentes ejes en Y será la velocidad en la llanta derecha y en Z el giro de la plataforma

$$v_{r\_r} := R_{1\_wr}^{-1} \cdot \left(v_{1\_1} + \omega_{1\_1} \times P_{1\_wr} \right) \rightarrow \begin{pmatrix} x'_1 + b \cdot \theta'_1 \\ y'_1 - d \cdot \theta'_1 \\ 0 \end{pmatrix}$$

Velocidad lineal en X de la llanta derecha será la velocidad del agente más la velocidad de b (distancia sobre Y) por una velocdad angular de entrada a la plataforma (velocidad tangencial de salida)

$$v_{wr} := \begin{pmatrix} r \cdot \varphi'_{fr} \\ 0 \\ 0 \end{pmatrix}$$

Esta relaciona las velocidades obtenidas con la velocidad lienal de avance

$$v_{r\_r} = v_{wr}$$

$$v_{r\_r} - v_{wr} = 0$$

$$\begin{pmatrix} 1 & 0 & b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} - \begin{pmatrix} r \cdot \varphi'_{fr} \\ 0 \\ 0 \end{pmatrix} = 0$$

$$\begin{bmatrix} \frac{d}{dx'_1}\left(v_{r\_r_0}\right) & \frac{d}{dy'_1}\left(v_{r\_r_0}\right) & \frac{d}{d\theta'_1}\left(v_{r\_r_0}\right) \\ \frac{d}{dx'_1}\left(v_{r\_r_1}\right) & \frac{d}{dy'_1}\left(v_{r\_r_1}\right) & \frac{d}{d\theta'_1}\left(v_{r\_r_1}\right) \\ \frac{d}{dx'_1}\left(v_{r\_r_2}\right) & \frac{d}{dy'_1}\left(v_{r\_r_2}\right) & \frac{d}{d\theta'_1}\left(v_{r\_r_2}\right) \end{bmatrix} \rightarrow \begin{pmatrix} 1 & 0 & b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix}$$

Los unos indican velocidad interna del robot y los otros velocidades respecto al sistema inercial

$$\xi_1 := \begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} \qquad \xi := \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} \qquad \theta_1 := \theta$$

$$R(\theta)^T \rightarrow \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}$$

Se sustituye en la relación importante de antes la transición de lo local a lo inercial

$$\begin{pmatrix} 1 & 0 & b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} - \begin{pmatrix} r \cdot \varphi'_{fr} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} - \begin{pmatrix} r \cdot \varphi'_{fr} \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} x' \cdot \cos(\theta) + b \cdot \theta' + y' \cdot \sin(\theta) - r \cdot \varphi'_{fr} \\ y' \cdot \cos(\theta) - x' \cdot \sin(\theta) - d \cdot \theta' \\ 0 \end{pmatrix}$$

La llanta izquierda velocidades por propagación

$$\omega_{l\_l} := R_{l\_wl}^{-1} \cdot \omega_{l\_l} + \varphi'_{fl} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ \varphi'_{fl} \\ \theta'_1 \end{pmatrix}$$

Se ocupa la Y en el vector final porque en ese eje se hace el giro de la llanta

Normalmente usamos el ángulo o velocidad angular de la junta, aquí es la velocidad de la llanta

Se observa que la velocidad angular en los diferentes ejes en Y será la velocidad en la llanta derecha y en Z el giro de la plataforma

$$v_{l\_l} := R_{l\_wl}^{-1} \cdot \left(v_{l\_l} + \omega_{l\_l} \times P_{l\_wl} \right) \rightarrow \begin{pmatrix} x'_1 - b \cdot \theta'_1 \\ y'_1 - d \cdot \theta'_1 \\ 0 \end{pmatrix}$$

Velocidad lineal en X de la llanta derecha será la velocidad del agente más la velocidad de b (distancia sobre Y) por una velocdad angular de entrada a la plataforma (velocidad tangencial de salida)

Aquí fata la velocidad lineal de avance relacionada a la velocidad de la llanta

$$v_{wl} := \begin{pmatrix} r\cdot\varphi'_{fl} \\ 0 \\ 0 \end{pmatrix}$$

$$v_{r\_r} = v_{wr}$$

$$v_{r\_r} - v_{wr} = 0$$

$$\begin{bmatrix} \frac{d}{dx'_1}\left(v_{l\_l_0}\right) & \frac{d}{dy'_1}\left(v_{l\_l_0}\right) & \frac{d}{d\theta'_1}\left(v_{l\_l_0}\right) \\ \frac{d}{dx'_1}\left(v_{l\_l_1}\right) & \frac{d}{dy'_1}\left(v_{l\_l_1}\right) & \frac{d}{d\theta'_1}\left(v_{l\_l_1}\right) \\ \frac{d}{dx'_1}\left(v_{l\_l_2}\right) & \frac{d}{dy'_1}\left(v_{l\_l_2}\right) & \frac{d}{d\theta'_1}\left(v_{l\_l_2}\right) \end{bmatrix} \rightarrow \begin{pmatrix} 1 & 0 & -b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} - \begin{pmatrix} r\cdot\varphi'_{fl} \\ 0 \\ 0 \end{pmatrix} = 0$$

Los unos indican velocidad interna del robot y los otros velocidades respecto al sistema inercial

$$\xi_1 := \begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} \qquad \xi := \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} \qquad \theta_1 := \theta$$

$$\begin{pmatrix} x'_1 \\ y'_1 \\ \theta'_1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}$$

Se sustituye en la relación importante de antes la transición de lo local a lo inercial

$$\begin{pmatrix} 1 & 0 & -b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} - \begin{pmatrix} r\cdot\varphi'_{fl} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & -b \\ 0 & 1 & -d \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} - \begin{pmatrix} r\cdot\varphi'_{fl} \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} x'\cdot\cos(\theta) - b\cdot\theta' + y'\cdot\sin(\theta) - r\cdot\varphi'_{fl} \\ y'\cdot\cos(\theta) - x'\cdot\sin(\theta) - d\cdot\theta' \\ 0 \end{pmatrix}$$

Ecación de restricción y el modelo cinemático inverso

$$WR := \begin{pmatrix} x'\cdot\cos(\theta) + b\cdot\theta' + y'\cdot\sin(\theta) - r\cdot\varphi'_{fr} \\ y'\cdot\cos(\theta) - x'\cdot\sin(\theta) - d\cdot\theta' \\ 0 \end{pmatrix} \qquad WL := \begin{pmatrix} x'\cdot\cos(\theta) - b\cdot\theta' + y'\cdot\sin(\theta) - r\cdot\varphi'_{fl} \\ y'\cdot\cos(\theta) - x'\cdot\sin(\theta) - d\cdot\theta' \\ 0 \end{pmatrix}$$

$$A_T := \begin{pmatrix} \frac{d}{dx'}WR_1 & \frac{d}{dy'}WR_1 & \frac{d}{d\theta'}WR_1 & \frac{d}{d\varphi'_{fr}}WR_1 & \frac{d}{d\varphi'_{fl}}WR_1 \\ \frac{d}{dx'}WL_1 & \frac{d}{dy'}WL_1 & \frac{d}{d\theta'}WL_1 & \frac{d}{d\varphi'_{fr}}WL_1 & \frac{d}{d\varphi'_{fl}}WL_1 \\ \frac{d}{dx'}WR_0 & \frac{d}{dy'}WR_0 & \frac{d}{d\theta'}WR_0 & \frac{d}{d\varphi'_{fr}}WR_0 & \frac{d}{d\varphi'_{fl}}WR_0 \\ \frac{d}{dx'}WL_0 & \frac{d}{dy'}WL_0 & \frac{d}{d\theta'}WL_0 & \frac{d}{d\varphi'_{fr}}WL_0 & \frac{d}{d\varphi'_{fl}}WL_0 \end{pmatrix} \rightarrow \begin{pmatrix} -\sin(\theta) & \cos(\theta) & -d & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & -d & 0 & 0 \\ \cos(\theta) & \sin(\theta) & b & -r & 0 \\ \cos(\theta) & \sin(\theta) & -b & 0 & -r \end{pmatrix}$$

Si se hace cero es como si me metiera en el sistema interno del robot

$$q' \rightarrow \begin{pmatrix} x' \\ y' \\ \theta' \\ \varphi'_{fr} \\ \varphi'_{fl} \end{pmatrix}$$

$$A_1 := A_T \text{ substitute}, \theta = 0 \rightarrow \begin{pmatrix} 0 & 1 & -d & 0 & 0 \\ 0 & 1 & -d & 0 & 0 \\ 1 & 0 & b & -r & 0 \\ 1 & 0 & -b & 0 & -r \end{pmatrix} \qquad q'_1 := \begin{pmatrix} x'_1 \\ y'_1 \\ \theta' \\ \varphi'fl \\ \varphi'fr \end{pmatrix}$$

$$A_1\cdot q'_1 \rightarrow \begin{pmatrix} y'_1 - d\cdot\theta' \\ y'_1 - d\cdot\theta' \\ x'_1 + b\cdot\theta' - r\cdot\varphi'fl \\ x'_1 - b\cdot\theta' - r\cdot\varphi'fr \end{pmatrix}$$

Variables a controlar : dimension(A1)-rango(A1) ==== 5-3 = 2

El rango de una matriz es el número de ecaciones linealmente independientes

$$EC1 := y'_1 - d\cdot\theta'$$

$$EC2 := y'_1 - d\cdot\theta'$$

$$EC3 := x'_1 + b\cdot\theta' - r\cdot\varphi'fl$$

$$EC4 := x'_1 - b\cdot\theta' - r\cdot\varphi'fr$$

Espacio nulo de la matriz A (solución)

Given

$0 = EC1$

$0 = EC2$

$0 = EC3$

$0 = EC4$

$x'_1 = \eta 1$

$\theta' = \eta 2$

$$q'_1 \rightarrow \begin{pmatrix} x'_1 \\ y'_1 \\ \theta' \\ \varphi'fl \\ \varphi'fr \end{pmatrix}$$

$$sol := Find\left(x'_1, y'_1, \theta', \varphi'fl, \varphi'fr\right) \rightarrow \begin{pmatrix} \eta 1 \\ d\cdot\eta 2 \\ \eta 2 \\ \dfrac{\eta 1 + b\cdot\eta 2}{r} \\ \dfrac{\eta 1 - b\cdot\eta 2}{r} \end{pmatrix}$$

Con esto se puede colocar una expresión donde q = S(q)*[X´ θ´]T

$$FS_{qq}(\eta 1, \eta 2) := \begin{pmatrix} \eta 1 \\ d \cdot \eta 2 \\ \eta 2 \\ \frac{\eta 1 + b \cdot \eta 2}{r} \\ \frac{\eta 1 - b \cdot \eta 2}{r} \end{pmatrix}$$

$$S_p := \text{augment}\left(FS_{qq}(1,0), FS_{qq}(0,1)\right) \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & d \\ 0 & 1 \\ \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{pmatrix}$$

Este sistema es en realildad S1 porque está referido al sistema montado en el robot, pero se quiere encontrar la relación con el marco inercial.

$$R_q := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S_T := R_q \cdot S_p \rightarrow \begin{pmatrix} \cos(\theta) & -d \cdot \sin(\theta) \\ \sin(\theta) & d \cdot \cos(\theta) \\ 0 & 1 \\ \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{pmatrix}$$

$$u' := \begin{pmatrix} x' \\ \theta' \end{pmatrix}$$

$$q' = S_T \cdot u'$$

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \varphi'_{fr} \\ \varphi'_{fl} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -d\cdot\sin(\theta) \\ \sin(\theta) & d\cdot\cos(\theta) \\ 0 & 1 \\ \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & -\frac{b}{r} \end{pmatrix} \cdot \begin{pmatrix} x' \\ \theta' \end{pmatrix}$$

$$S_T\cdot u' \rightarrow \begin{pmatrix} x'\cdot\cos(\theta) - d\cdot\theta'\cdot\sin(\theta) \\ x'\cdot\sin(\theta) + d\cdot\theta'\cdot\cos(\theta) \\ \theta' \\ \frac{x'}{r} + \frac{b\cdot\theta'}{r} \\ \frac{x'}{r} - \frac{b\cdot\theta'}{r} \end{pmatrix}$$

Matriz de velocidad en X, Y, angular y en las llantas correspondientemente por cada renglón en función de velocidades de avance, angular y ángulo actual del robot

Anexo 2. Programa para simulación gráfica 2D en Matlab

```
%   Alejandro Ruiz Esparza Rodríguez
%   Moisés Sebastián González Chávez
%   UNAM - Facultad de Ingeniería
%   Mecatronic Research Group

%Configuración de imagen para simulación
clear
filename = 'Modelo.gif';            %Nombre del archivo para la animación
pasos = 500;                        %Número de pasos de la animación
x = -4:0.2:4;                       %Coordenadas x para la gráfica de campo
y = -2:0.2:2;                       %Coordenadas y para la gráfica de campo


%Variables
posicion=[];
velocidad=[];
fuerza=[];

F_atr_max=3;        %Fuerza máxima de atracción al objetivo
F_rep_max=5;        %Fuerza máxima de repulsión de los obstáculos
F_rep_a_max=1;      %Fuerza máxima de repulsión entre agentes
F_atr_a_max=0.5;    %Fuerza máxima de atracción entre agentes

num_Agentes=4;      %Número de agentes que conforman el enjambre
tam_ag=5;           %Tamaño de agentes (circulares) en términos de marcador
R_ag=1;             %Radio de acción del afecto de atracción de los agentes
r_ali=0.5;          %Radio de acción del efecto de orientación de los agentes
r_ag=0.3;           %Radio de acción del efecto de repulsión de los agentes

num_Obstaculos=2;   %Número de obstáculos en la simulación
R_obs=1;            %Radio de acción del efecto de repulsión de los obstáculos
tam_obs=[40,40];    %Tamaño de los obstáculos en términos de marcador
tam_1=110;          %Tamaño de la unidad (de gráfica) en términos de marcador

m_1=1; m_2=1;
g=[0;0];
h=0.01;

%Condiciones iniciales de cada uno de los agentes
posicion(:,1)=[-3;0.25];
velocidad(:,1)=[0;0];
fuerza(:,1)=[0;0];

posicion(:,2)=[-3;-1.5];
velocidad(:,2)=[0;0];
fuerza(:,2)=[0;0];

posicion(:,3)=[-3;1];
velocidad(:,3)=[0;0];
fuerza(:,3)=[0;0];

posicion(:,4)=[-3;-0.5];
velocidad(:,4)=[0;0];
fuerza(:,4)=[0;0];

objetivo=[3;0];                 %Objetivo del enjambre

obstaculo(:,1)=[0;0];           %Posición del obstáculo
obstaculo(:,2)=[1.5;-0.2];

% gráfica del campo
```

```
[X,Y] = meshgrid(x,y);  % Se establece un campo vectorial de las coordenadas

F_atr_x_1 = objetivo(1)-X;    % Fuerza de atracción proporcional a la distancia
F_atr_y_1 = objetivo(2)-Y;

F_atr_mag_1 = sqrt(F_atr_x_1.^2 + F_atr_y_1.^2);  % Magnitud

[sx,sy]=size(F_atr_x_1);

for i=1:1:sx        % Limita el valor máximo del campo de atración
    for j=1:1:sy
        if F_atr_mag_1(i,j)>F_atr_max
            F_atr_x_1(i,j) = F_atr_x_1(i,j)/F_atr_mag_1(i,j)*F_atr_max;
            F_atr_y_1(i,j) = F_atr_y_1(i,j)/F_atr_mag_1(i,j)*F_atr_max;
        end
    end
end

F_rep_x=zeros([size(X),num_Obstaculos]);
F_rep_y=zeros([size(X),num_Obstaculos]);
F_rep_mag=zeros([size(X),num_Obstaculos]);

for i=1:num_Obstaculos  % Fuerza de repulsión proporcional a la distancia
    F_rep_x(:,:,i) = X-obstaculo(1,i);
    F_rep_y(:,:,i) = Y-obstaculo(2,i);
    F_rep_mag(:,:,i) = sqrt(F_rep_x(:,:,i).^2 + F_rep_y(:,:,i).^2);

    F_rep_x(:,:,i) = F_rep_x(:,:,i)-
F_rep_x(:,:,i)./F_rep_mag(:,:,i).*((tam_obs(i)/2)/tam_1);
    F_rep_y(:,:,i) = F_rep_y(:,:,i)-
F_rep_y(:,:,i)./F_rep_mag(:,:,i).*((tam_obs(i)/2)/tam_1);

    F_rep_x(:,:,i) = F_rep_x(:,:,i).*(F_rep_mag(:,:,i)>((tam_obs(i)/2)/tam_1));
    F_rep_y(:,:,i) = F_rep_y(:,:,i).*(F_rep_mag(:,:,i)>((tam_obs(i)/2)/tam_1));
    F_rep_mag(:,:,i) = sqrt(F_rep_x(:,:,i).^2 + F_rep_y(:,:,i).^2);
end

[sx,sy]=size(X);
F_rep_x_total=zeros(sx,sy);
F_rep_y_total=zeros(sx,sy);

for k=1:1:num_Obstaculos
    for i=1:1:sx
        for j=1:1:sy
            if F_rep_mag(i,j,k)<R_obs && F_rep_mag(i,j,k)>0
                F_rep_x(i,j,k)=F_rep_x(i,j,k)/F_rep_mag(i,j,k)*(R_obs-
F_rep_mag(i,j,k))/R_obs*F_rep_max;
                F_rep_y(i,j,k)=F_rep_y(i,j,k)/F_rep_mag(i,j,k)*(R_obs-
F_rep_mag(i,j,k))/R_obs*F_rep_max;
            else
                F_rep_x(i,j,k) = 0;
                F_rep_y(i,j,k) = 0;
            end
        end
    end

    F_rep_x_total=F_rep_x_total+F_rep_x(:,:,k);
    F_rep_y_total=F_rep_y_total+F_rep_y(:,:,k);
end

F_pot_x = F_atr_x_1 + F_rep_x_total;        % Suma de fuerzas de atracción y
repulsión
F_pot_y = F_atr_y_1 + F_rep_y_total;

F_total=[1 1 1 1; 1 1 1 1];
```

```
% Simulación de desplazamiento
fig=figure('Position',get(0,'ScreenSize'));
for step=1:pasos
    % Se grafica: campo, agente, obstáculo y objetivo
    q = quiver(X,Y,F_pot_x,F_pot_y);
    q.AutoScaleFactor = 0.4;
    daspect([1 1 1]); hold on;

    for n=1:num_Obstaculos

plot(obstaculo(1,n),obstaculo(2,n),'or','MarkerSize',tam_obs(n),'MarkerFaceColor'
,'r');
    end
    plot(objetivo(1),objetivo(2),'x');
    for n=1:num_Agentes
        plot(posicion(1,n),posicion(2,n),'ok', 'MarkerSize',
10,'MarkerFaceColor','k');
    end
    hold off;

    title(['step : ' num2str(step)]);
    axis([min(x) max(x) min(y) max(y)]);
    set(gca,'FontSize',15)
    grid on;
    pause(0.01);
    %guardar valor anterior
    pos_ant=posicion;
    vel_ant=velocidad;

    F_atr_s = zeros(2,num_Agentes);
    F_atr_s_mag = zeros(1,num_Agentes);
    F_rep_s = zeros(2,num_Agentes);
    F_rep_s_mag = zeros(1,num_Agentes);

    %actualización de valores
    F_rep_s_total_mag=zeros(num_Agentes);
    for n=1:num_Agentes
        F_atr_s(:,n) = objetivo-posicion(:,n);
        F_atr_s_mag(n) = sqrt(F_atr_s(1,n)^2 + F_atr_s(2,n)^2);

        if F_atr_s_mag(n)>F_atr_max
            F_atr_s(1,n) = F_atr_s(1,n)/F_atr_s_mag(n)*F_atr_max;
            F_atr_s(2,n) = F_atr_s(2,n)/F_atr_s_mag(n)*F_atr_max;
        end

        F_rep_s_total=zeros(2,num_Agentes);
        for i=1:num_Obstaculos
            F_rep_s(:,n,i) = posicion(:,n)-obstaculo(:,i);
            F_rep_s_mag(n,i) = sqrt(F_rep_s(1,n,i)^2 + F_rep_s(2,n,i)^2);

            if F_rep_s_mag(n,i)<R_obs
                F_rep_s(1,n,i)=F_rep_s(1,n,i)/F_rep_s_mag(n,i)*(R_obs-
F_rep_s_mag(n,i))/R_obs*F_rep_max;
                F_rep_s(2,n,i)=F_rep_s(2,n,i)/F_rep_s_mag(n,i)*(R_obs-
F_rep_s_mag(n,i))/R_obs*F_rep_max;
            else
                F_rep_s(:,n,i) = [0;0];
            end

            F_rep_s_total=F_rep_s_total+F_rep_s(:,:,i);

        end
```

```
        F_rep_s_total_mag(n) = sqrt(F_rep_s_total(1,n)^2 + F_rep_s_total(2,n)^2);
%Cálculo de la magnitud de la fuerza de repulsión

    end

    %Fuerzas entre agentes
    dist=zeros(2,num_Agentes,num_Agentes);
    dist_mag=zeros(num_Agentes,num_Agentes);
    F_rep_a=zeros(2,num_Agentes,num_Agentes);
    F_total=F_atr_s+F_rep_s_total;

    %Efecto de fuerzas entre agentes
    for m=1:num_Agentes
        for n=1:num_Agentes
            dist(:,m,n)=posicion(:,m)-posicion(:,n);
            dist(:,n,m)=posicion(:,n)-posicion(:,m);

            dist_mag(m,n)=sqrt(dist(1,m,n)^2+dist(2,m,n)^2);
            dist_mag(n,m)=sqrt(dist(1,n,m)^2+dist(2,m,n)^2);

            if (dist_mag(m,n) < r_ag) && (dist_mag(m,n)~=0)
                F_rep_a(:,m,n)=dist(:,m,n)/dist_mag(m,n)*(r_ag-
dist_mag(m,n))/r_ag*F_rep_a_max;
            elseif (dist_mag(m,n)>=r_ali) && (dist_mag(m,n)<R_ag)
                F_rep_a(:,m,n) = -(dist(:,m,n)/dist_mag(m,n)*(dist_mag(m,n)-
r_ali))/(R_ag-r_ali)*F_atr_a_max;
            else
                F_rep_a(:,m,n) = 0;
            end

            if (dist_mag(n,m) < r_ag) && (dist_mag(n,m)~=0)
                F_rep_a(:,n,m)=dist(:,n,m)/dist_mag(n,m)*(r_ag-
dist_mag(n,m))/r_ag*F_rep_a_max;
            elseif (dist_mag(n,m)>=r_ali) && (dist_mag(n,m)<R_ag)
                F_rep_a(:,n,m) = -(dist(:,n,m)/dist_mag(n,m)*(dist_mag(n,m)-
r_ali))/(R_ag-r_ali)*F_atr_a_max;
            else
                F_rep_a(:,n,m) = 0;
            end

            F_total(:,m) = F_total(:,m) + F_rep_a(:,m,n);
            F_total(:,n) = F_total(:,n) + F_rep_a(:,n,m);
        end
    end

    vpm = velocidad;
    xpm = pos_ant + h*vpm;
    velocidad = F_total;
    posicion = pos_ant + h*vpm;

      %Guardar como gif
      frame = getframe(fig);
      im = frame2im(frame);
      [imind,cm] = rgb2ind(im,256);
      if step == 1
          imwrite(imind,cm,filename,'gif', 'Loopcount',inf,'DelayTime',0.01);
      else
          imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime',0.01);
      end

end
```

Anexo 3. Programa en C# para obtención de los datos de visión por medio de ReacTIVision.

```
/*
   Modificado y adaptado por:

    Moisés Sebastián González Chávez
    Alejandro Ruiz Esparza Rodríguez

    Trabajo de tesis: "Formaciones adaptativas para la evasión de obstáculos mediante flocking
aplicado a la robótica móvil"

    Universidad Nacional Autónoma de México
    Facultad de Ingeniería
    Mechatronics Research Group
 */

/*
        TUIO C# Example - part of the reacTIVision project
        http://reacTIVision.sourceforge.net/

        Copyright (c) 2005-2009 Martin Kaltenbrunner <martin@tuio.org>

        This program is free software; you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
        the Free Software Foundation; either version 2 of the License, or
        (at your option) any later version.

        This program is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with this program; if not, write to the Free Software
        Foundation, Intcur., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/


using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using TUIO;

public class TuioDump : TuioListener{
    /* Se coloca un fiducial fidRefX (en este caso el fiducial 0) sobre el eje y a una distancia n
en sentido positivo
        * del origen del sistema de referencia que se va a utilizar (en las coordenadas (0,+n)).
        * Se coloca otro fiducial fidRefY (en este caso el fiducial 1) sobre el eje y a una
distancia n en sentido negativo
        * del origen del sistema de referencia que se va a utilizar (en las coordenadas (0,-n)).
        *
        * La distancia entre ambos fiducials dRef = 2n (en mm) sirve como referencia de unidades de
longitud,
        * por lo que debe ser especificada en las variables definidas a continuación.
        *
        * Estos fiducials son utilizados para configurar el espacio de acuerdo con el sistema de
referencia establecido,
        * el origen del sistema queda locaizado en el punto medio de las posiciones de ambos
fiducials, quedando determinado
        * el eje y por los mismos fiducials y siendo el eje x perpendicuar a éste desde el origen.
        */

    ////////////////////// VARIABLES \\\\\\\\\\\\\\\\\\\\\\
```

```
    // Variables utilizadas para calibrar el entorno con el sistema de referencia establecido
    private readonly int fidRefX = 0;
    private readonly int fidRefY = 1;
    private readonly int dRef = 800;

    // Variables de referencia para la cámara 1
    private int xRefX1 = 0;
    private int yRefX1 = 0;
    private float angRefX1 = 0;
    private int xRefY1 = 0;
    private int yRefY1 = 0;
    private float angRefY1 = 0;
    private double pix2cm1 = 0;
    private double Cam1CentroX = 0;
    private double Cam1CentroY = 0;

    // Variables de referencia para la cámara 2
    private int xRefX2 = 0;
    private int yRefX2 = 0;
    private float angRefX2 = 0;
    private int xRefY2 = 0;
    private int yRefY2 = 0;
    private float angRefY2 = 0;
    private double pix2cm2 = 0;
    private double Cam2CentroX = 0;
    private double Cam2CentroY = 0;

    // Variables para ajuste de rotación de cámara
    private double coordX = 0;
    private double coordY = 0;
    private double angulo = 0;
    private double coordXr = 0;
    private double coordYr = 0;
    private double angPos = 0;

    // Variables para la calibración de la altura del fiducial
    private readonly int origenX = 640 / 2; // Posición del orígen de las cámaras
    private readonly int origenY = 480 / 2; // Posición del orígen en Y
    private double dist_Fid = 0;            // Distancia en pixeles del fiducial al orígen de la
cámara
    private double angFidCal = 0;           // Angulo del fiducial al orígen de la cámara
    private readonly double altTecho = 2400;// Altura del techo en milimetros
    private readonly double altFid = 155;   // Altura del fiducial en milímetros  Robots grandes:
120 pequeños: 155
    private double corr = 0;                // Variable asociada a la corrección del fiducial en
distanca en milímetros
    private double distReal = 0;            // Distancia real del fiducial al centro de la cámara
    private double coordLocalX = 0;         // Coordenada en X desde el centro de la cámara
    private double coordLocalY = 0;         // Coordenada en Y desde el centro de la cámara
    private double coordGlobalX = 0;        // Coordenadas despúes de la calibración por altura del
fiducial
    private double coordGlobalY = 0;
    private double coordGlobalXR = 0;       // Coordenadas despúes de la calibración por altura del
fiducial
    private double coordGlobalYR = 0;

    // Variables para el envío de datos por WIFI UDP
    private readonly string direccion = "127.0.0.1";
    private readonly int puerto = 666;


    ////////////////////// FUNCIONES \\\\\\\\\\\\\\\\\\\\\\

    // Función cuando ingresa un fiducial al campo de visión
    public void addTuioObject(TuioObject tobj, int port){
        if ((tobj.SymbolID == fidRefX || tobj.SymbolID == fidRefY) ){   // Fiducials de calibración
            switch (port){
                case 3331:  // Calibración de cámara 1
```

```
                    if (tobj.SymbolID == fidRefX){
                        xRefX1 = tobj.getScreenX(640);
                        yRefX1 = tobj.getScreenY(480);
                        angRefX1 = tobj.Angle;
                    }
                    else if (tobj.SymbolID == fidRefY){
                        xRefY1 = tobj.getScreenX(640);
                        yRefY1 = tobj.getScreenY(480);
                        angRefY1 = tobj.Angle;
                    }
                    pix2cm1 = dRef / Math.Sqrt(Math.Pow(xRefY1 - xRefX1, 2) + Math.Pow(yRefY1 -
yRefX1, 2));

                    coordX = -(320 - ((xRefY1 + xRefX1) / 2)) * pix2cm1;
                    coordY = -(240 - ((yRefY1 + yRefX1) / 2)) * pix2cm1;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY1 - xRefX1 != 0){
                        angPos = Math.Atan2(yRefY1 - yRefX1, xRefY1 - xRefX1);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    Cam1CentroX = coordXr;
                    Cam1CentroY = coordYr;

                    break;

                case 3333:  // Calibración de cámara 2
                    if (tobj.SymbolID == fidRefX){
                        xRefX2 = tobj.getScreenX(640);
                        yRefX2 = tobj.getScreenY(480);
                        angRefX2 = tobj.Angle;
                    }
                    else if (tobj.SymbolID == fidRefY){
                        xRefY2 = tobj.getScreenX(640);
                        yRefY2 = tobj.getScreenY(480);
                        angRefY2 = tobj.Angle;
                    }
                    pix2cm2 = dRef / Math.Sqrt(Math.Pow(xRefY2 - xRefX2, 2) + Math.Pow(yRefY2 -
yRefX2, 2));

                    coordX = -(320 - ((xRefY2 + xRefX2) / 2)) * pix2cm2;
                    coordY = -(240 - ((yRefY2 + yRefX2) / 2)) * pix2cm2;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY2 - xRefX2 != 0){
                        angPos = Math.Atan2(yRefY2 - yRefX2, xRefY2 - xRefX2);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    Cam2CentroX = coordXr;
                    Cam2CentroY = coordYr;

                    break;
            }
```

```
        }
        else{   // Fiducials no de caibración
            switch (port){
                case 3331:
                    coordX = -(tobj.getScreenX(640) - ((xRefY1 + xRefX1) / 2)) * pix2cm1;
                    coordY = -(tobj.getScreenY(480) - ((yRefY1 + yRefX1) / 2)) * pix2cm1;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY1 - xRefX1 != 0){
                        angPos = Math.Atan2(yRefY1 - yRefX1, xRefY1 - xRefX1);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    //Haciendo la corrección de altura del plano del fiducial
                    dist_Fid = (Math.Sqrt(Math.Pow((tobj.getScreenX(640) - origenX), 2) +
Math.Pow((tobj.getScreenY(480) - origenY), 2))) * pix2cm1;
                    corr = (altFid * dist_Fid) / altTecho; //Por triángulos semejantes se obtiene la
corrección
                    distReal = dist_Fid - corr;
                    angFidCal = Math.Atan2(tobj.getScreenY(480) - origenY, tobj.getScreenX(640) -
origenX);
                    coordLocalX = distReal * Math.Cos(angFidCal);
                    coordLocalY = distReal * Math.Sin(angFidCal);

                    coordGlobalX = Cam1CentroX + coordLocalX;
                    coordGlobalY = Cam1CentroY + coordLocalY;
                    coordGlobalXR = -Math.Round(coordGlobalX * Math.Cos(-(angPos - Math.PI / 2)) -
coordGlobalY * Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordGlobalYR = -Math.Round(coordGlobalX * Math.Sin(-(angPos - Math.PI / 2)) +
coordGlobalY * Math.Cos(-(angPos - Math.PI / 2)), 2);

                    angulo = (angulo - angPos * (180 / Math.PI) - 180);
                    if (angulo < 0){
                        angulo = angulo + 360;
                    }
                    break;

                case 3333:
                    coordX = -(tobj.getScreenX(640) - ((xRefY2 + xRefX2) / 2)) * pix2cm2;
                    coordY = -(tobj.getScreenY(480) - ((yRefY2 + yRefX2) / 2)) * pix2cm2;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY2 - xRefX2 != 0){
                        angPos = Math.Atan2(yRefY2 - yRefX2, xRefY2 - xRefX2);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    //Haciendo la corrección de altura del plano del fiducial
                    dist_Fid = Math.Sqrt(Math.Pow((tobj.getScreenX(640) - origenX), 2) +
Math.Pow((tobj.getScreenY(480) - origenY), 2)) * pix2cm2;
                    corr = (altFid * dist_Fid) / altTecho;
                    distReal = dist_Fid - corr;
                    angFidCal = Math.Atan2(tobj.getScreenY(480) - origenY, tobj.getScreenX(640) -
origenX);
                    coordLocalX = distReal * Math.Cos(angFidCal);
```

```
                    coordLocalY = distReal * Math.Sin(angFidCal);

                    coordGlobalX = Cam2CentroX + coordLocalX;
                    coordGlobalY = Cam2CentroY + coordLocalY;
                    coordGlobalXR = -Math.Round(coordGlobalX * Math.Cos(-(angPos - Math.PI / 2)) -
coordGlobalY * Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordGlobalYR = -Math.Round(coordGlobalX * Math.Sin(-(angPos - Math.PI / 2)) +
coordGlobalY * Math.Cos(-(angPos - Math.PI / 2)), 2);

                    angulo = (angulo - angPos * (180 / Math.PI) - 180);
                    if (angulo < 0){
                        angulo = angulo + 360;
                    }
                    break;
            }

            if (tobj.SymbolID == 5){
                EnvioDatos(coordXr, coordYr, angulo, tobj.SymbolID);
            }
            else{
                EnvioDatos(coordGlobalXR, coordGlobalYR, angulo, tobj.SymbolID);
            }
        }
        }

    // Función cuando se mueve un fiducial dentro del campo de visión
        public void updateTuioObject(TuioObject tobj, int port) {
        if (tobj.SymbolID != fidRefX && tobj.SymbolID != fidRefY){  // Fiducials no de referencia
            switch (port){
                case 3331:
                    coordX = -(tobj.getScreenX(640) - ((xRefY1+xRefX1)/2)) * pix2cm1;
                    coordY = -(tobj.getScreenY(480) - ((yRefY1+yRefX1)/2)) * pix2cm1;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY1 - xRefX1 != 0){
                        angPos = Math.Atan2(yRefY1 - yRefX1, xRefY1 - xRefX1);
                    }
                    else {
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    //Haciendo la corrección de altura del plano del fiducial
                    dist_Fid = (Math.Sqrt(Math.Pow((tobj.getScreenX(640) - origenX), 2) +
Math.Pow((tobj.getScreenY(480) - origenY), 2))) * pix2cm1;
                    corr = (altFid * dist_Fid) / altTecho; //Por triángulos semejantes se obtiene la
corrección
                    distReal = dist_Fid - corr;
                    angFidCal = Math.Atan2(tobj.getScreenY(480) - origenY, tobj.getScreenX(640) -
origenX);
                    coordLocalX = distReal * Math.Cos(angFidCal);
                    coordLocalY = distReal * Math.Sin(angFidCal);

                    coordGlobalX = Cam1CentroX + coordLocalX;
                    coordGlobalY = Cam1CentroY + coordLocalY;
                    coordGlobalXR = -Math.Round(coordGlobalX * Math.Cos(-(angPos - Math.PI / 2)) -
coordGlobalY * Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordGlobalYR = -Math.Round(coordGlobalX * Math.Sin(-(angPos - Math.PI / 2)) +
coordGlobalY * Math.Cos(-(angPos - Math.PI / 2)), 2);

                    angulo = (angulo - angPos * (180 / Math.PI) - 180);
                    if (angulo < 0){
                        angulo = angulo + 360;
                    }
                    break;
```

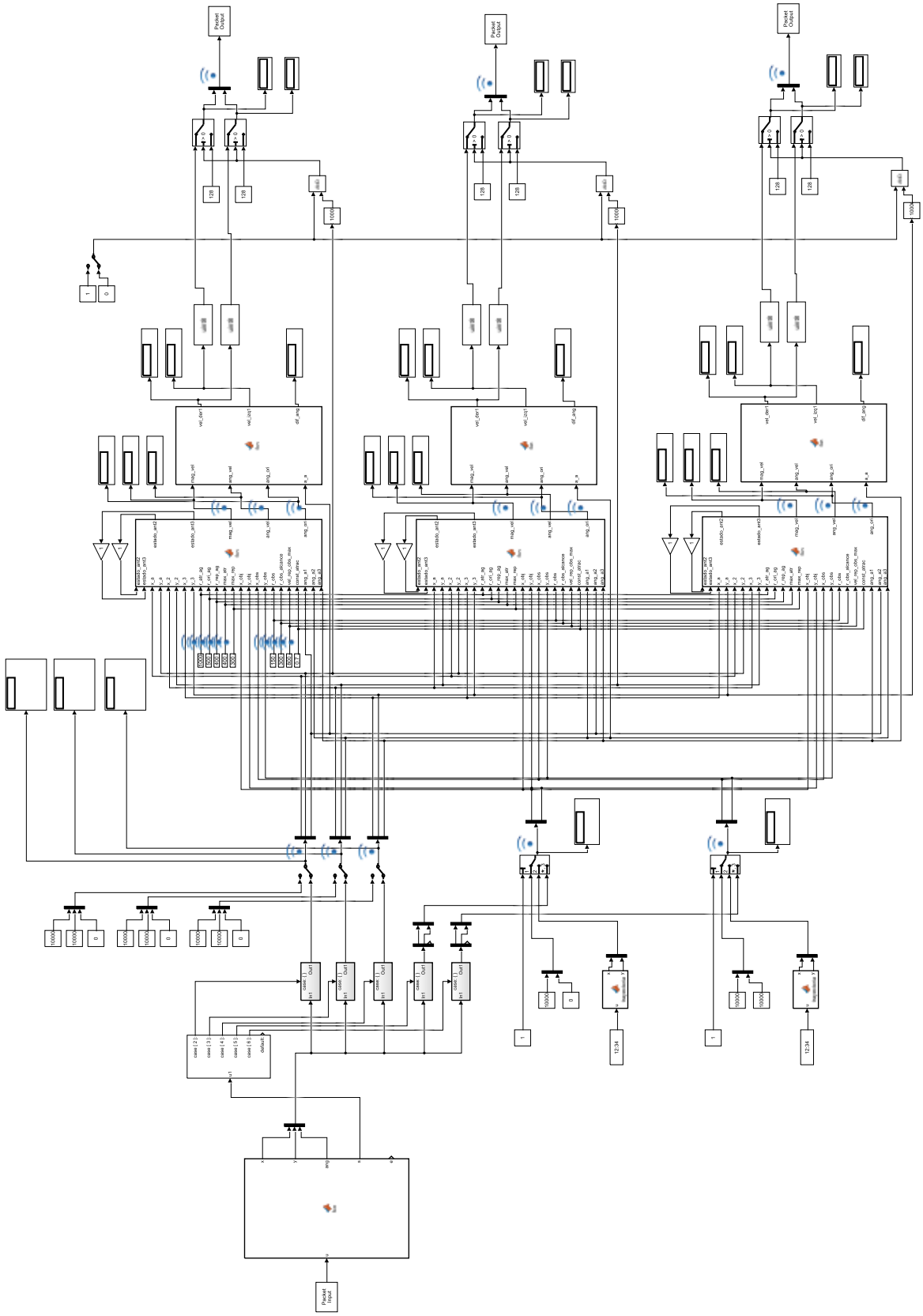
```
                case 3333:
                    coordX = -(tobj.getScreenX(640) - ((xRefY2+xRefX2)/2)) * pix2cm2;
                    coordY = -(tobj.getScreenY(480) - ((yRefY2+yRefX2)/2)) * pix2cm2;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY2 - xRefX2 != 0){
                        angPos = Math.Atan2(yRefY2 - yRefX2, xRefY2 - xRefX2);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);
                    angulo = (angulo - angPos*(180/Math.PI) - 180);

                    //Haciendo la corrección de altura del plano del fiducial
                    dist_Fid = Math.Sqrt(Math.Pow((tobj.getScreenX(640) - origenX), 2) +
Math.Pow((tobj.getScreenY(480) - origenY), 2)) * pix2cm2;
                    corr = (altFid * dist_Fid) / altTecho; //triángulos semejantes
                    distReal = dist_Fid - corr;
                    angFidCal = Math.Atan2(tobj.getScreenY(480) - origenY, tobj.getScreenX(640) -
origenX);
                    coordLocalX = distReal * Math.Cos(angFidCal);
                    coordLocalY = distReal * Math.Sin(angFidCal);

                    coordGlobalX = Cam2CentroX + coordLocalX;
                    coordGlobalY = Cam2CentroY + coordLocalY;
                    coordGlobalXR = -Math.Round(coordGlobalX * Math.Cos(-(angPos - Math.PI / 2)) -
coordGlobalY * Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordGlobalYR = -Math.Round(coordGlobalX * Math.Sin(-(angPos - Math.PI / 2)) +
coordGlobalY * Math.Cos(-(angPos - Math.PI / 2)), 2);

                    if (angulo < 0){
                        angulo = angulo + 360;
                    }
                    break;
            }

            if (tobj.SymbolID == 5){
                EnvioDatos(coordXr, coordYr, angulo, tobj.SymbolID);
            }
            else{
                EnvioDatos(coordGlobalXR, coordGlobalYR, angulo, tobj.SymbolID);
            }
        }

        else{   // Fiducials de referencia
            switch (port){
                case 3331:  // Calibración cámara 1
                    if (tobj.SymbolID == fidRefX){
                        xRefX1 = tobj.getScreenX(640);
                        yRefX1 = tobj.getScreenY(480);
                        angRefX1 = tobj.Angle;
                    }
                    else if (tobj.SymbolID == fidRefY){
                        xRefY1 = tobj.getScreenX(640);
                        yRefY1 = tobj.getScreenY(480);
                        angRefY1 = tobj.Angle;
                    }
                    pix2cm1 = dRef / Math.Sqrt(Math.Pow(xRefY1 - xRefX1, 2) + Math.Pow(yRefY1 -
yRefX1, 2));

                    coordX = -(320 - ((xRefY1 + xRefX1) / 2)) * pix2cm1;
                    coordY = -(240 - ((yRefY1 + yRefX1) / 2)) * pix2cm1;
                    angulo = Math.Round(tobj.AngleDegrees, 2);
```

```
                    if (xRefY1 - xRefX1 != 0){
                        angPos = Math.Atan2(yRefY1 - yRefX1, xRefY1 - xRefX1);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    Cam1CentroX = coordXr;
                    Cam1CentroY = coordYr;

                    break;

                case 3333:  // Calibración cámara 2
                    if (tobj.SymbolID == fidRefX){
                        xRefX2 = tobj.getScreenX(640);
                        yRefX2 = tobj.getScreenY(480);
                        angRefX2 = tobj.Angle;
                    }
                    else if (tobj.SymbolID == fidRefY){
                        xRefY2 = tobj.getScreenX(640);
                        yRefY2 = tobj.getScreenY(480);
                        angRefY2 = tobj.Angle;
                    }
                    pix2cm2 = dRef / Math.Sqrt(Math.Pow(xRefY2 - xRefX2, 2) + Math.Pow(yRefY2 -
yRefX2, 2));

                    coordX = -(320 - ((xRefY2 + xRefX2) / 2)) * pix2cm2;
                    coordY = -(240 - ((yRefY2 + yRefX2) / 2)) * pix2cm2;
                    angulo = Math.Round(tobj.AngleDegrees, 2);

                    if (xRefY2 - xRefX2 != 0){
                        angPos = Math.Atan2(yRefY2 - yRefX2, xRefY2 - xRefX2);
                    }
                    else{
                        angPos = Math.PI / 2;
                    }

                    coordXr = Math.Round(coordX * Math.Cos(-(angPos - Math.PI / 2)) - coordY *
Math.Sin(-(angPos - Math.PI / 2)), 2);
                    coordYr = Math.Round(coordX * Math.Sin(-(angPos - Math.PI / 2)) + coordY *
Math.Cos(-(angPos - Math.PI / 2)), 2);

                    Cam2CentroX = coordXr;
                    Cam2CentroY = coordYr;

                    break;
            }
        }
        }

    // Función cuando sale un fiducial del campo de visión
        public void removeTuioObject(TuioObject tobj) {
    }


    // Funciones Tuio adicionales no utilizadas
        public void addTuioCursor(TuioCursor tcur) {}
        public void updateTuioCursor(TuioCursor tcur) {}
        public void removeTuioCursor(TuioCursor tcur) {}
        public void addTuioBlob(TuioBlob tblb) {}
        public void updateTuioBlob(TuioBlob tblb) {}
        public void removeTuioBlob(TuioBlob tblb) {}
        public void refresh(TuioTime frameTime) {}
```

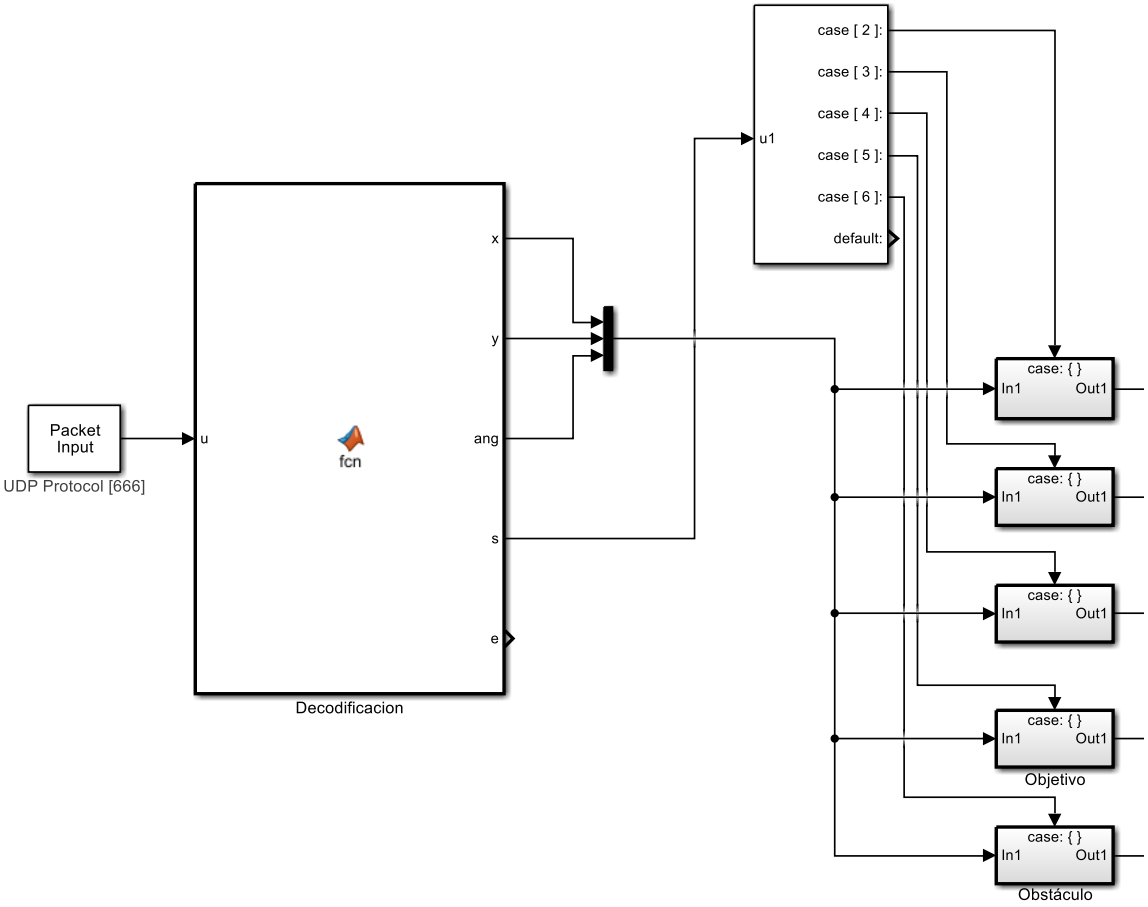
```
    // Función para el envío de datos
    public void EnvioDatos(double x, double y, double a, int id){
        string g = "M" + TresValores((int)x) + TresValores((int)y) + TresValores((int)a) + id + "M"
+ TresValores((int)x) + TresValores((int)y) + TresValores((int)a) + id;

        Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
        IPAddress ipaddress = IPAddress.Parse(direccion);
        IPEndPoint endpoint = new IPEndPoint(ipaddress, puerto);
        byte[] buffer = Encoding.ASCII.GetBytes(g);

        socket.SendTo(buffer, endpoint);

        Console.WriteLine(id + "  " + x + "  " + y + "  " + a);
        Console.WriteLine(g);
    }

    // Función para convertir un número en 5 caracteres (incluyendo el signo - para números
negativos)
    public string TresValores(int numero){
        string devuelve;
        // Positivos
        if (numero < 100 && numero >= 10){
            devuelve = "000" + numero;
        }
        else if (numero < 10 && numero >= 0){
            devuelve = "0000" + numero;
        }
        else if (numero >= 100 && numero < 1000){
            devuelve = "00" + numero;
        }
        else if (numero >= 1000){
            devuelve = "0" + numero;
        }
        // Negativos
        else if (numero < 0 && numero > -10){
            devuelve = "-000" + Math.Abs(numero);
        }
        else if (numero <= -10 && numero > -100){
            devuelve = "-00" + Math.Abs(numero);
        }
        else if (numero <= -100 && numero > -1000){
            devuelve = "-0" + Math.Abs(numero);
        }
        else if (numero <= -1000){
            devuelve = "-" + Math.Abs(numero);
        }
        else{
            devuelve = "Error";
        }

        return devuelve;
    }

    // Función Main ejecuta el programa
    public static void Main(String[] argv) {
                TuioDump demo = new TuioDump();
                TuioClient client = null;
        TuioClient client2 = null;

                switch (argv.Length) {
                        case 1:
                                int port = 0;
                                port = int.Parse(argv[0],null);
                                if(port>0) client = new TuioClient(port);
                client2 = new TuioClient(3331);
                                break;
                        case 0:
                                client = new TuioClient();
```

```
                client2 = new TuioClient(3331);
                                break;
                }
        //GENERANDO 1 CLIENTE EN PORT 3333
                if (client!=null) {
                        client.addTuioListener(demo);
                        client.connect();
                        Console.WriteLine("listening to TUIO messages at port " + client.getPort());
                }
        else Console.WriteLine("usage: java TuioDump [port]");

        //GENERANDO 2 CLIENTE EN PORT 3331
        if (client2 != null){
            client2.addTuioListener(demo);
            client2.connect();
            Console.WriteLine("listening to TUIO messages at port " + client2.getPort());
        }
        else Console.WriteLine("usage: java TuioDump [port]");
        }
}
```

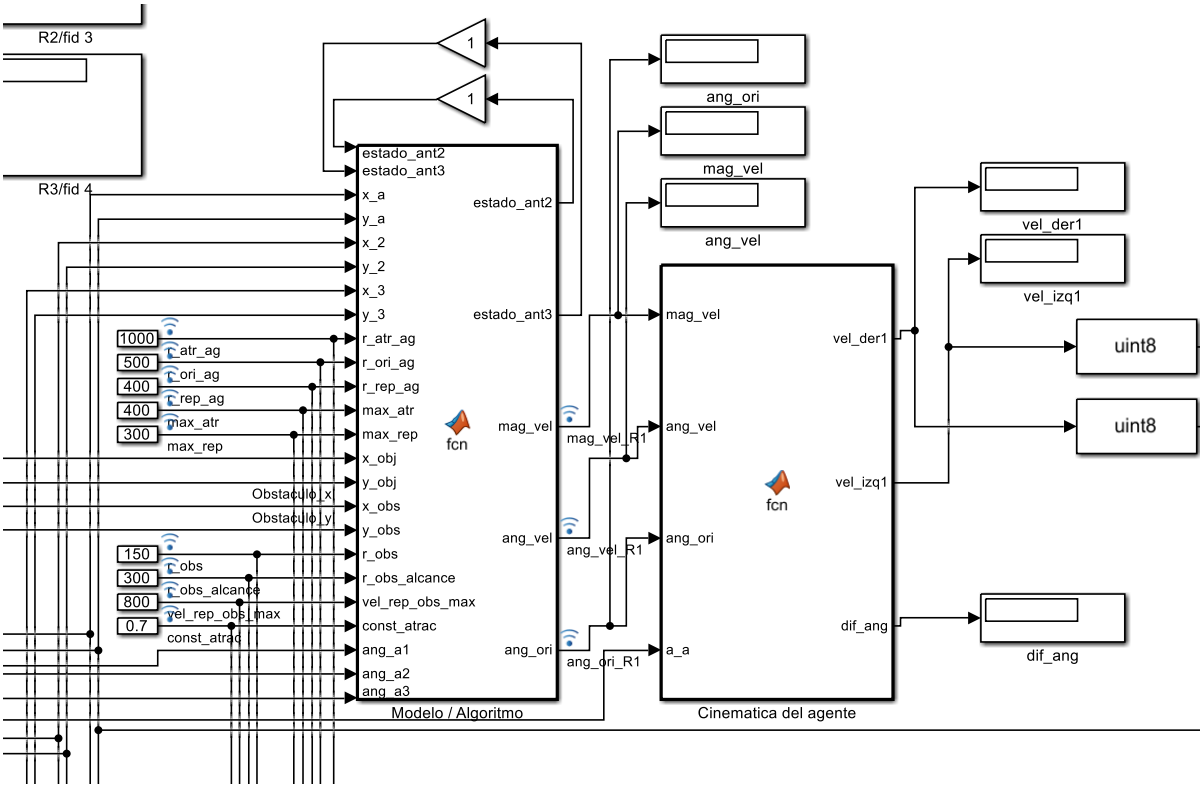
Anexo 4. Diagrama de Simulink del sistema desarrollado



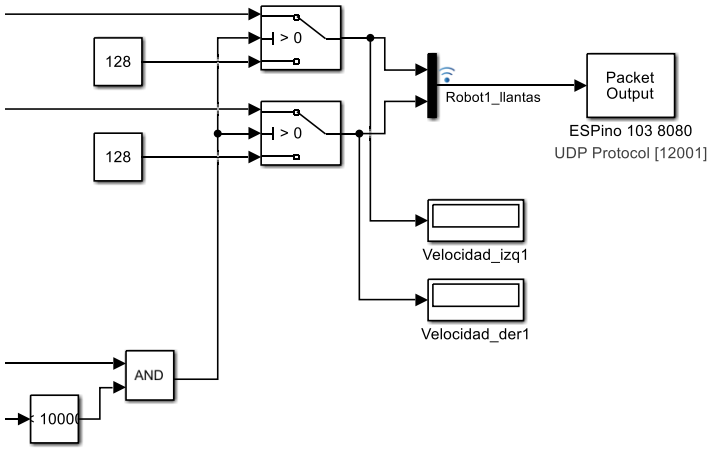
Sección de recepción y decodificación de la información:



Sección de implementación del modelo de campos potenciales, modelo de flocking, y cinemática del robot:



Sección de transmisión de información al robot:



Anexo 5. Código de Matlab para la implementación del sistema.

Bloque decodificación:

```
function [x, y, ang, s, e] = fcn(u)
    aux=0;
    e=zeros(17);

    for i=1:1:17
        if u(i)==77
            aux=i;
        else
        end
    end
    r=17;
    for j=1:1:r
        e(j)=u(aux+j);
    end

    s = e(16)-48;

    x=(e(2)-48)*1000+(e(3)-48)*100+(e(4)-48)*10+(e(5)-48);
    y=(e(7)-48)*1000+(e(8)-48)*100+(e(9)-48)*10+(e(10)-48);
    ang=(e(13)-48)*100 + (e(14)-48)*10 + (e(15)-48);

    if(e(1)==45)
           x = -x;
    end
    if(e(6)==45)
           y = -y;
    end
end
```

Bloque modelo / algoritmo:

```
function [estado_ant2,estado_ant3,mag_vel, ang_vel,ang_ori]=
fcn(estado_ant2,estado_ant3,x_a,y_a,x_2,y_2,x_3,y_3,r_atr_ag,r_ori_ag,r_rep_ag,ma
x_atr,max_rep,x_obj,y_obj,x_obs,y_obs,r_obs,r_obs_alcance,vel_rep_obs_max,const_a
trac,ang_a1,ang_a2,ang_a3)

      if estado_ant2~=0 && estado_ant2~=1 && estado_ant2~=2 && estado_ant2~=3
          estado_ant2 = 0;
      end
      if estado_ant3~=0 && estado_ant3~=1 && estado_ant3~=2 && estado_ant3~=3
          estado_ant3 = 0;
      end

      umbral = 40;

      %% VELOCIDAD DE ATRACCION AL OBJETIVO
      if x_obj>=10000 || y_obj>=10000
          d_obj=0;
      else
          d_obj=sqrt((x_obj-x_a)^2+(y_obj-y_a)^2);  %Calcula la distancia del
agente al objetivo
      end
```

```
      vel_obj = const_atrac * d_obj;            %Velocidad hacia el objetivo es
proporcional a la distancia al mismo
      if vel_obj > 300                          %% Establece velocidad máxima al
objetivo
          vel_obj = 300;
      end
      ang_obj = atan2(y_obj-y_a, x_obj-x_a);    %Cálculo del ángulo con respecto
al eje x del agente al objetivo
      comp_x_atr_obj = vel_obj*cos(ang_obj);    %Cálculo de la componente x de la
atracción al objetivo
      comp_y_atr_obj = vel_obj*sin(ang_obj);    %Cálculo de la componente y de la
atracción al objetivo
      %% VELOCIDAD DE REPULSIÓN DEL OBSTÁCULO
      d_obs=sqrt((x_obs-x_a)^2+(y_obs-y_a)^2);  %Calcula la distancia del agente
al centro del obstáculo
      d_obs=d_obs-r_obs;                        %Resta el radio del obstáculo
para tener la distancia al borde del obstáculo
      ang_obs = atan2(-y_obs+y_a,-x_obs+x_a);   %Cálculo del ángulo del agente al
obstáculo (Porqué - a las coordenas del obstáculo???)

      %% ACCION DE LOS OTROS AGENTES

      % Efecto del agente externo 2
      d_2=sqrt((x_a-x_2)^2+(y_a-y_2)^2);                          %Cálculo de
distancia entre los agentes
      [mag_2,estado_nvo2] =
atraccion(d_2,r_atr_ag,r_ori_ag,r_rep_ag,max_atr,max_rep,estado_ant2,umbral);
%Ejecuta la función para calcular el efecto entre agentes
      ang_2 = angulo(x_a,y_a,x_2,y_2,estado_nvo2);        %Ejecuta la función
para calcular el ángulo entre los agentes
      comp_x_2 = mag_2*cos(ang_2);                                %Cálculo de
componente x del efecto del agente externo 2
      comp_y_2 = mag_2*sin(ang_2);                                %Cálculo de
componente y del efecto del agente externo 2

      % Efecto del agente externo 3
      d_3=sqrt((x_a-x_3)^2+(y_a-y_3)^2);
      [mag_3,estado_nvo3] =
atraccion(d_3,r_atr_ag,r_ori_ag,r_rep_ag,max_atr,max_rep,estado_ant3,umbral);
      ang_3 = angulo(x_a,y_a,x_3,y_3,estado_nvo3);
      comp_x_3 = mag_3*cos(ang_3);
      comp_y_3 = mag_3*sin(ang_3);

      ang_ori =
orientacion(d_2,d_3,r_ori_ag,r_rep_ag,ang_a1,ang_a2,ang_a3,estado_nvo2,estado_nvo
3,umbral);

      %%
      if ((estado_nvo2 == 3 || estado_nvo2 == 1)&&mag_2<10)
          estado_nvo2 = 2;
      end
      if ((estado_nvo3 == 3 || estado_nvo3 == 1)&&mag_3<10)
          estado_nvo3 = 2;
      end

      estado_ant2 = estado_nvo2;
      estado_ant3 = estado_nvo3;

      %% TERMINA CÁLCULO DE REPULSIÓN DE OBSTÁCULO Y SE SUMAN LOS EFECTOS
      if d_obs<r_obs_alcance && d_obs>0
          vel_rep_obs = ((r_obs_alcance-d_obs)/r_obs_alcance)*vel_rep_obs_max;
          comp_x_vel_rep_obs = vel_rep_obs * cos(ang_obs);
```

```
          comp_y_vel_rep_obs = vel_rep_obs * sin(ang_obs);

          ang_vel = atan2(comp_y_vel_rep_obs + comp_y_atr_obj + comp_y_2 +
comp_y_3, comp_x_vel_rep_obs + comp_x_atr_obj + comp_x_2 + comp_x_3);
          ang_vel=ang_vel*(180/pi);
          if ang_vel<0
            ang_vel=ang_vel+360;
          end
          mag_vel = sqrt((comp_y_vel_rep_obs + comp_y_atr_obj + comp_y_2 +
comp_y_3)^2+(comp_x_vel_rep_obs + comp_x_atr_obj + comp_x_2 + comp_x_3)^2);
      else
          ang_vel = atan2(comp_y_atr_obj + comp_y_2 + comp_y_3, comp_x_atr_obj +
comp_x_2 + comp_x_3);
          ang_vel=ang_vel*(180/pi);
          if ang_vel<0
            ang_vel=ang_vel+360;
          end
          mag_vel = sqrt((comp_y_atr_obj + comp_y_2 + comp_y_3)^2+(comp_x_atr_obj
+ comp_x_2 + comp_x_3)^2);
      end
end
```

```
function ang = angulo(x_a,y_a,x_f,y_f,estado_nvo)
      if estado_nvo ~= 0
          if estado_nvo == 3                              % Zona de repulsión
             ang = atan2(y_a-y_f,x_a-x_f);                % Cacula ángulo del
agente externo al agente actual
          else
              if estado_nvo == 2                          % Zona de orientación
                  ang = 0;
              else                                        % Zona de atracción
                 ang = atan2(y_f-y_a,x_f-x_a);            % Calcula ángulo del
agente actual al agente externo
              end
          end
      else                                                % Fuera de la zona de
acción entre agentes
          ang = atan2(y_f-y_a,x_f-x_a);                   % Calcula ángulo del
agente actual al agente externo
      end
end
```

```
function [mag,estado_nvo] =
atraccion(d_a,r_atr_ag,r_ori_ag,r_rep_ag,max_atr,max_rep,estado_ant,umbral)
%Calcula el efecto de interacción entre agentes.
    r_robot = 150; %Radio considerado para el robot, para la máxima repusión
cuando estén prácticamente en contacto, no sobre el mismo punto

      if (d_a < r_atr_ag-umbral && d_a>0 && estado_ant<1) || (d_a <
r_atr_ag+umbral && d_a>0 && estado_ant>=1)       % Si se encuentra dentro del
campo de acción de otro agente
```

```
          if (d_a < r_rep_ag-umbral && estado_ant<3) || (d_a < r_rep_ag+umbral &&
estado_ant>=3)            % Condición dentro del radio de repulsión

              mag = ((r_rep_ag+umbral-d_a)/(r_rep_ag+umbral-r_robot))*max_rep;
% Repulsión inversamente proporcional a la distancia entre agentes (Entre más
cerca se encuentren mayor es la repulsión)
              estado_nvo = 3;

          else
              if (d_a < r_ori_ag-umbral && estado_ant<2) || (d_a <
r_ori_ag+umbral && estado_ant>=2)    % Condición en la zona de orientación
                  mag = 0;
                  estado_nvo = 2;
              else                                        % Condición en la zona de
atracción

                  mag = ((d_a-(r_ori_ag-umbral))/((r_atr_ag+umbral)-(r_ori_ag-
umbral)))*max_atr;         % Atracción proporcional a la distancia entre agentes
mag = (d_a/r_atr_ag)*max_atr;
                  estado_nvo = 1;

              end
          end
      else
          mag = 0;
          estado_nvo = 0;
      end
end
```

```
function mag_ori =
orientacion(d_2,d_3,r_ori_ag,r_rep_ag,ang_a1,ang_a2,ang_a3,estado_nvo2,estado_nvo
3,umbral)
%Calcula el efecto de interacción entre agentes.
      if estado_nvo2 == 2   %d_2 > r_rep_ag && d_2 < r_ori_ag    % Condición en
la zona de orientación
          if estado_nvo3 == 2   %d_3 > r_rep_ag && d_3 < r_ori_ag

              ang_prom = (ang_a2+ang_a3)/2;
              if abs(ang_prom-ang_a2)>90 || abs(ang_prom-ang_a3)>90
                  ang_prom = ang_prom + 180;
              end
              if ang_prom > 360
                  ang_prom = ang_prom-360;
              end

              mag_ori = ang_prom-ang_a1;

              if mag_ori > 180
                mag_ori = mag_ori - 360;
              else
                if mag_ori < -180
                    mag_ori = mag_ori + 360;
                end
              end

              k = ((r_ori_ag-r_rep_ag)/2 - abs((d_2+d_3)/2-
(r_rep_ag+r_ori_ag)/2))/(r_ori_ag-r_rep_ag)/2;
              mag_ori = mag_ori*k;
```

```
            else
                mag_ori = ang_a2-ang_a1;

                if mag_ori > 180
                  mag_ori = mag_ori - 360;
                else
                  if mag_ori < -180
                      mag_ori = mag_ori + 360;
                  end
                end

                k = ((r_ori_ag-r_rep_ag)/2 - abs(d_2-
(r_rep_ag+r_ori_ag)/2))/(r_ori_ag-r_rep_ag)/2;
                mag_ori = mag_ori*k;
            end
        else
            if estado_nvo3 == 2   %d_3 > r_rep_ag && d_3 < r_ori_ag
                mag_ori = ang_a3-ang_a1;

                if mag_ori > 180
                  mag_ori = mag_ori - 360;
                else
                  if mag_ori < -180
                      mag_ori = mag_ori + 360;
                  end
                end

                k = ((r_ori_ag-r_rep_ag)/2+umbral - abs(d_3-
(r_rep_ag+r_ori_ag)/2))/((r_ori_ag-r_rep_ag)/2+umbral);
                mag_ori = mag_ori*k;
            else
                  mag_ori = 0;
            end
        end

end
```

Bloque cinemática del agente:

```
function [vel_der1,vel_izq1,dif_ang]= fcn(mag_vel,ang_vel,ang_ori,a_a)

      r = 60;               % Radio de las llantas [mm]   Robots grandes: 98
pequeños: 60
      rp = 30; %60              % Radio de protección del robot (se considera que
ya llegó al objetivo) [mm]
      ap = 15; %20               % Ángulo de protección [°]
      b = 85;              % Distancia del eje de desplazamiento del robot al
centro de las llantas [mm]   Robots grandes: 140 pequeño:90
      d = 60;               % Distancia del eje de las llantas al punto de
control en la dirección de desplazamiento [mm]    Robots grandes: 20 pequeño: 60


      % Haciendo que la velocidad de desplazamiento en x (v) sea
      % proporcional a la distancia entre el atacante y el objetivo que se
      % pretende llegar.

      vel = 4.5;  % Constante de proporción de la velocidad
```

```
        velMax = 100;
        d_ao = mag_vel;        % Distancia del atacante al objetivo

        if (d_ao < rp)
            v = 0;
        else
            v = vel * d_ao;     % Velocidad de desplazamiento en x
        end

        % Haciendo proporcional la velocidad angular (w) con la diferencia de
        % ángulo entre la orientación del robot y la orientación hacia donde
        % se encuentra el objetivo.

        vang = 0.3; % Constante de proporción de velocidad angular
        dif_ang = ang_vel - a_a;

        if (dif_ang > 180)
            dif_ang = dif_ang - 360;
        end

        if (dif_ang < -180)
            dif_ang = dif_ang + 360;
        end

        if (abs(dif_ang) < ap)
            w = 0;
        else
            w = vang * dif_ang; % Velocidad angular
        end

        if (d_ao < rp)
            w = 0;
        end

        if abs(ang_ori)>5
            w = w + 0.05*ang_ori;
        end

        % Cálculo de las velocidades de giro de las llantas

        vLlantaDer = -((v + b*w)/r); % Velocidad de giro de la llanta derecha
Signo - por ajuste de los motores
        vLlantaIzq = (v - b*w)/r; % Velocidad de giro de la llanta izquierda

        % Límite de la velocidad máxima para la detección de la cámara.

        if (vLlantaDer > velMax)
          vLlantaDer = velMax;
        elseif (vLlantaDer < -velMax)
          vLlantaDer = -velMax;
        end

        if (vLlantaIzq > velMax)
          vLlantaIzq = velMax;
        elseif (vLlantaIzq < -velMax)
          vLlantaIzq = -velMax;
        end

%       ********************* Ajuste para tarjetas md25
**********************
        vel_der1 = 128 + vLlantaDer;
        vel_izq1 = 128 + vLlantaIzq;
end
```

Anexo 6. Programa para comunicación con los robots móviles para el ESP8266

```
#include <ESP8266WiFi.h>  //libreria wifi espino
#include <WiFiUDP.h>      //libreria UDP
#include <Wire.h>         //Libreria I2C

//*************************Código para conexión WIFI y UDP*************************
// wifi connection variables
const char* ssid = "RoboticaMovil2";
const char* password = "123456789";
boolean wifiConnected = false;
byte a, b;

// UDP variables
unsigned int localPort = 8080;
WiFiUDP UDP;
boolean udpConnected = false;
char packetBuffer[255]; //buffer to hold incoming packet,
char ReplyBuffer[] = "acknowledged"; // a string to send back
//*********************************************************************************

//*************************Código para I2C*****************************************
#define CMD                 (byte)0x00                      // Values of 0 eing sent using write
                                                            // This is a but with arduino 1
#define MD25ADDRESS         0x58                           // Address of the MD25(ATACANTE)
#define SOFTWAREREG         0x0D                            // Byte to read the software version
#define SPEED1              (byte)0x00                      // Byte to send speed to first motor
#define SPEED2              0x01                            // Byte to send speed to second motor
#define ENCODERONE          0x02                            // Byte to read motor encoder 1
#define ENCODERTWO          0x06                            // Byte to read motor encoder 2
#define VOLTREAD            0x0A                            // Byte to read battery volts
#define RESETENCODERS       0x20                            // Byte to Reset encoders
//*********************************************************************************

void setup() {
  // Initialise Serial connection
  Serial.begin(115200);
  Wire.begin();
  // Initialise wifi connection
  wifiConnected = connectWifi();
  // only proceed if wifi connection successful
  if (wifiConnected)
    udpConnected = connectUDP();
}

void loop() {
  // check if the WiFi and UDP connections were successful
  if (wifiConnected) {
    if (udpConnected) {

      // if there's data available, read a packet
      int packetSize = UDP.parsePacket();
      if (packetSize)
      {
        int len = UDP.read(packetBuffer,255);
        if(len>0)
      {
        packetBuffer[len]=0;
```

```
        a = packetBuffer[0];
        b = packetBuffer[1];

        Serial.print("a: ");
        Serial.print(a);
        Serial.print("   b: ");
        Serial.println(b);
      }

      }

      //delay(10);
    }
   }

    Wire.beginTransmission(MD25ADDRESS);                    // Drive motor 2 at speed value stored in x
    Wire.write(SPEED2);
    Wire.write(a);
    Wire.endTransmission();

   Wire.beginTransmission(MD25ADDRESS);                    // Drive motor 2 at speed value stored in x
    Wire.write(SPEED1);
    Wire.write(b);
    Wire.endTransmission();
}

// connect to UDP - returns true if successful or false if not
boolean connectUDP() {
  boolean state = false;

  Serial.println("");
  Serial.println("Connecting to UDP");

  if (UDP.begin(localPort) == 1) {
    Serial.println("Connection successful");
    state = true;
  }
  else {
    Serial.println("Connection failed");
  }

  return state;

}
// connect to wifi - returns true if successful or false if not
boolean connectWifi() {
  boolean state = true;

  WiFi.begin(ssid, password);
  return state;
}
```

Referencias y bibliografía

[1] J. Yan, X. Guan, X. Luo, and F. Tan, “Formation and obstacle avoidance control for multiagent systems,” *J. Control Theory Appl.*, vol. 9, no. 2, pp. 141–147, 2011.

[2] G. Wu and X. Wang, “Obstacle avoidance algorithm for a finite-time formation control,” *2014 IEEE Work. Adv. Res. Technol. Ind. Appl.*, pp. 1084–1089, 2014.

[3] Z. Duan and X. Gu, “Animal group behavioral model with Evasion Mechanism,” *Proc. Int. Jt. Conf. Neural Networks*, pp. 1167–1172, 2014.

[4] J. Li, W. Zhang, H. Su, Y. Yang, and H. Zhou, “Coordinated obstacle avoidance with reduced interaction,” *Neurocomputing*, vol. 139, pp. 233–245, 2014.

[5] Q. Yuan, J. Zhan, and X. Li, “Outdoor flocking of quadcopter drones with decentralized model predictive control,” *ISA Trans.*, vol. 71, pp. 84–92, 2017.

[6] R. Toyota and T. Namerikawa, “Formation Control of Multi-Agent System Considering Obstacle Avoidance,” pp. 446–451, 2017.

[7] N. S. Morozova, “Formation Control and Obstacle Avoidance for Multi-Agent Systems with Dynamic Topology,” 4894.

[8] L. Zhou and W. Li, “Adaptive Artificial Potential Field Approach for Obstacle Avoidance Path Planning,” *2014 Seventh Int. Symp. Comput. Intell. Des.*, no. 1, pp. 429–432, 2014.

[9] R. Osorio-Comparan, I. Lopez-Juarez, A. Reyes-Acosta, M. Pena-Cabrera, M. Bustamante, and G. Lefranc, “Mobile robot navigation using potential fields and LMA,” *Proc. 2016 IEEE Int. Conf. Autom.*, pp. 1–7, 2016.

[10] M. M. J. A. Butr, M. Exico, and C. P, “M ´ etodos Usados en la Soluci ´ on del Problema de Evasi ´ on de Obst ´ aculos en RMR,” 2009.

[11] R. Lopez-Padilla, V. Ayala-Ramirez, and R. E. Sanchez-Yanez, “Some experiments on reactive obstacle avoidance for a mobile robot,” *Proc. - 18th Int. Conf. Electron. Commun. Comput. CONIELECOMP 2008*, pp. 193–196, 2008.

[12] M. Víctor and J. González, “Evadiendo Obstáculos con Robots Móviles,” *Rev. Digit. Univ.*, vol. 6, no. 1, pp. 1–9, 2005.

[13] R. Arkin, *Behavior Based Robotics*. 2000.

[14] D. J. T. Sumpter, “The principles of collective animal behaviour,” *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 361, no. 1465, pp. 5–22, 2006.

[15] R. Bouffanais, *Design and Control of Swarm Dynamics*. 2016.

[16] S. Kumra, R. Saxena, and S. Mehta, “An Extensive Review on Swarm Robotics,” pp. 140–145, 2009.

[17] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, “A taxonomy for swarm robots,” *Proc. 1993 IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS ’93)*, vol. 1, no. January 2014, pp. 441–447, 1993.

[18] L. Bayindir and E. SAHIN, “A review of studies in swarm robotics,” *Turk. J. Elec. Engin*, vol. 15, no. 2, pp. 115–148, 2007.

[19] S. Camazine, J. L. Deneubour, N. R. Franks, J. Sneyd, G. Theraulauz, and E. Bonbeau, “Self-Organisation in Biological Systems,” *Princet. Univ. Press*, 2001.

[20] E. S. O. Soysal, “Probabilistic Aggregation Strategies in Swarm Robotic Systems,” *Proc. IEEE Swarm Intell. Symp.*, 2005.

[21] M. D. S. Nouyan, “Chain Formation in a Swarm of Robots,” Belgium, 2004.

[22] M. D. R. Grob, M. Bonani, F. Mondada, “Autonomous Self-assembly in a Swarmbot,” *Proc. Third Int. Symp. Auton. Minirobots Res. Edutainment*, pp. 314–322, 2006.

[23] T. Balch, “Hierarchic Social Entropy: An Information Theoretic Measure of Robot Group Diversity,” *Auton. Robots*, vol. 8, pp. 209–238, 2000.

[24] J. D. V. Trianni, T. Labella, R. Grob, E. S¸ahin, M. Dorigo, “Modeling Pattern Formation in a Swarm of Self-Assembling Robots,” Belgium, 2002.

[25] E. S. E. Bah¸ceci, “Evolving Aggregation Behaviors for Swarm Robotic Systems: A Systematic Case Study,” in *Proc. of the IEEE Swarm Intelligence Symposium*, 2005.

[26] a. T. Hayes and P. Dormiani-Tabatabaei, “Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots,” *Proc. 2002 IEEE Int. Conf. Robot. Autom. (Cat. No.02CH37292)*, vol. 4, no. May, pp. 1–7, 2002.

[27] P. K. P. Tangamchit, J. Dolan, “The necessity of average rewards in cooperative multirobot learning,” in *In IEEE International Conference on Robotics and Automation*, 2002, pp. 1296–1301.

[28] G. S. A. Howard, M. Mataric, “Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem,” *DARS 02*, 2002.

[29] M. Luisa and S. Tortosa, “Agentes y enjambres artificiales : modelado y comportamientos para sistemas de enjambre robóticos Agentes y enjambres artificiales : modelado y comportamientos para sistemas de enjambre rob ´.”

[30] 0.(Artificial Intciligence Laboratory Khatib and S. University), “REAL-TIME OBSTACLE AVOIDANCE FOR MANIPULATORS AND MOBILE ROBOTS,” pp. 500–505, 1985.

[31] D. H. K. D. H. Kim, H.-W. L. H.-W. Lee, S. Shin, and T. Suzuki, “Local Path Planning Based on New Repulsive Potential Functions with Angle Distributions,” *Third Int. Conf. Inf. Technol. Appl.*, vol. 2, pp. 0–5, 2005.

[32] H. Rezaee and F. Abdollahi, “Mobile robots cooperative control and obstacle avoidance using potential field,” *2011 IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, pp. 61–66, 2011.

[33] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, “Distributed, physics-based control of swarms of vehicles,” *Auton. Robots*, vol. 17, no. 2–3, pp. 137–162, 2004.

[34] A. Kumar, S. Sharma, R. Tiwari, and S. Majumdar, “Area exploration by flocking of multi robot,” *Procedia Eng.*, vol. 41, no. Iris, pp. 377–382, 2012.

[35] F. Gökçe and E. Şahin, “The pros and cons of flocking in the long-range ‘migration’ of

mobile robot swarms," *Theor. Comput. Sci.*, vol. 411, no. 21, pp. 2140–2154, 2010.

[36] T. Balch and M. Hybinette, "Social Scalable Multi-Robot Formations," no. April, pp. 73–80, 2000.

[37] Dorantes Emmanuel, "Robot Móvil Omnidireccional Redundante: Modelado, Control e Implementación," Universidad Nacional Autónoma de México, 2015.

[38] V. J. G. Villela, R. Parkin, M. L. Parra, J. M. D. González, and M. J. G. Liho, "A wheeled mobile robot with obstacle avoidance capability," vol. 1, no. 5, pp. 159–166, 2004.

[39] S. Méndez, "Experimentación en tiempo real sobre deformación reactiva de trayectorias para robótica móvil," Universidad Nacional Autónoma de México.

[40] C. Reynolds, "Flocks: herds and schools: A distributed behavioral model," *Comput. Graph. (ACM).*, vol. 21(4), pp. 25–34, 1987.

[41] P. J. Gálvez Valadez, "Desarrollo De Banco De Pruebas Para El Estudio De Robots Moviles." p. 74, 2015.

[42] M. Kaltenbrunner and R. Bencina, "reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction," *Proc. 1st Int. Conf. Tangible Embed. Interact.*, pp. 69–74, 2007.

[43] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "TUIO: A Protocol for Table-Top Tangible User Interfaces," vol. 2, pp. 1263–1267, 2004.

[44] H. Irene, "Manipulador móvil omnidireccional , su coordinación de movimientos en ambientes inteligentes," 2016.

[45] J. A. Romero Esquinca, "Estabilización experimental de un robot móvil diferencial alrededor de trayectorias tipo C2," Universidad Nacional Autónoma de México, 2013.

[46] G. C. S. Cruz and P. M. M. Encarnação, "Obstacle Avoidance for Unmanned Aerial Vehicles," *J. Intell. Robot. Syst.*, vol. 65, no. 1–4, pp. 203–217, 2012.