Universidad Zaragoza
1542

Trabajo Fin de Grado

# Configuración Distribuida de Formas para Enjambres de Robots. Simulación y Experimentación en el Robotarium.

# *Distributed Shape Formation for Robotic Swarms. Simulation and Experimentation in the Robotarium.*


Autora

Raquel Moreno Royo

Director

Enrique Teruel Doñate

Grado en Ingeniería Electrónica y Automática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza


Noviembre de 2022

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

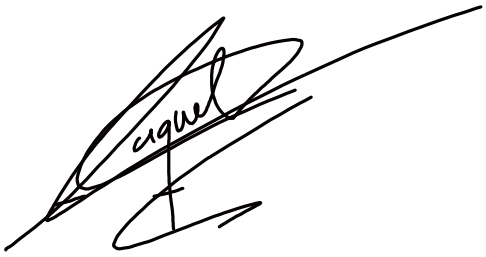
D./Dª. Raquel Moreno Royo,

con nº de DNI 73133226B en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Configuración Distribuida de Formas para Enjambres de Robots. Simulación y Experimentación en el Robotarium.

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de noviembre de 2022



Fdo: Raquel Moreno Royo

# AGRADECIMIENTOS

A mi familia y amigos de siempre, por estar presentes, acompañarme, inspirarme y brindarme todo su apoyo a lo largo de toda mi experiencia universitaria.

A mi director, Enrique Teruel Doñate, y a todos mis compañeros y amigos a los que he tenido la suerte de conocer durante este Grado y con los que he tenido la fortuna de rodearme en proyectos, tanto académicos como personales.

A todos, mi más sincero agradecimiento.

# RESUMEN

El objetivo de este proyecto es la implementación y validación de algoritmos de configuración distribuida de sistemas multi-robot en una forma dada, tanto en simulación, como sobre los robots físicos proporcionados por la Universidad Georgia Tech a través de su Robotarium. Se trata de desarrollar y poner a prueba algoritmos concretos relacionados con las líneas de investigación del director del Trabajo de Fin de Grado, así como de evaluar críticamente el alcance y limitaciones del Robotarium como plataforma de simulación y experimentación, proponiendo métodos y buenas prácticas para trabajar con el mismo.

La implementación se ha realizado mediante librerías de *Python* desarrolladas a propósito del Robotarium, para simulación y experimentación en entorno real. Los casos de estudio que se han abordado y que se van a comentar prolíficamente en sus secciones pertinentes, son sistemas multi-robot, formados por numerosos robots móviles con control autónomo (local), por lo que se requiere que la programación sea orientada a objetos, modular y jerárquica. Téngase en cuenta, que la gestión local y descentralizada propia de la robótica de *swarm*, se opone diametralmente a la filosofía global y centralista del Robotarium, por lo que ha sido necesario emplear una capa intermedia de código para manejar la información global del Robotarium, de forma que los algoritmos distribuidos implementados puedan ser validados en el mismo. Tratándose de control de robots en movimiento, se ha requerido aplicar métodos propios del control automático.

El proyecto está articulado según sigue: en el Capítulo 1 se ofrece una introducción a la robótica de *swarm*, exponiendo sus bondades, además de proyectos de interés. En el Capítulo 2 se introduce el Robotarium, exponiendo las características de su *hardware* y *software*. En el Capítulo 3 se detallan las particularidades de la arquitectura que se ha implementado. En los Capítulos 4 y 5 se expone el artículo de referencia empleado, así como la implementación del algoritmo, sus problemáticas y la alternativa propuesta. Por último, en el Capítulo 6 se ofrecen los resultados obtenidos y recomendaciones para el uso del entorno real del Robotarium. Las conclusiones y líneas futuras de trabajo se han señalado en el Capítulo 7.

# Índice

# Capítulo 1

# Robótica de *swarm*

## 1.1. Introducción

De acuerdo con [1], la robótica de *swarm* es el estudio de grupos constituidos por un gran número de robots, relativamente simples físicamente, que se pueden diseñar de tal forma que se obtenga un comportamiento colectivo deseado de las interacciones locales entre los robots, así como entre los robots y el entorno en el que operan.

Se enumeran a continuación los aspectos más característicos de un sistema de robótica de *swarm* según [2]:

- Los robots son autónomos.
- Los robots se sitúan en el entorno y pueden actuar para modificarlo.
- Las capacidades para comunicar y detectar por medio de sensores son locales.
- Los robots no tienen acceso a control centralizado o noción global del *swarm*.
- Los robots cooperan entre ellos para abordar una tarea asignada.

Los comportamientos colectivos de los *swarm* de robots están inspirados en la conducta de los enjambres presentes en la naturaleza, y en cómo los individuos que los componen interactúan entre sí. Conforme a [3], se aportan algunos ejemplos característicos: los insectos sociales, entre los que se hallan las colonias de hormigas y abejas como casos más representativos, los bancos de peces (ver Figura 1.1b), las bandadas de pájaros (ver Figura 1.1a) e incluso colonias de bacterias, que poseen intrincados mecanismos de comunicación entre colonias acorde con [4].

La ventaja primordial que ofrecen los sistemas de *swarm*, ya sea en la naturaleza o en el campo de la robótica, radica en que se benefician de la actuación de los individuos del enjambre como colectivo. En general, los agentes simples que conforman el *swarm*

no serían capaces de solventar tareas de cierto grado de complejidad por sí mismos, pero como se puede constatar en los proyectos comentados en la Sección 1.3, logran concluir con éxito tareas de lo más sofisticadas cuando actúan en conjunto, valiéndose de un conjunto de normas sencillas.

Los enjambres de seres vivos que habitan en la naturaleza poseen características inherentes de gran interés para la aproximación desde el punto de vista de la robótica. Destaca especialmente la sincronización de su operación, puesto que, pese a que carecen de coordinación centralizada en la realización de tareas sincronizadas, los enjambres funcionan de forma robusta, flexible y altamente adaptable, como se va a comentar seguidamente [1]:

- **Robustez**, en el sentido de que el *swarm* sea capaz de continuar funcionando, a pesar de ocurrir fallos en los robots individuales o perturbaciones en el entorno. Los factores que más contribuyen a que estos sistemas sean robustos son: la redundancia de los mismos, esto es, cuando un individuo se pierde es fácilmente reemplazable, la descentralización de la coordinación o ausencia de un líder, por lo que la destrucción de una parte del sistema no tiene por qué afectar al resto y, por último, la simplicidad de los individuos que conforman el *swarm*, haciéndolos menos propensos a fallar y más fáciles de reemplazar.

- **Flexibilidad**, entendida como la habilidad para lidiar con un amplio espectro de entornos y tareas a realizar. El objetivo es que el sistema posea la facultad de aportar y ejecutar soluciones moduladas dependiendo de las condiciones de operación, además de ser capaz de afrontar adecuadamente los cambios que puedan ocurrir en el entorno. Gracias al auge de las técnicas de *machine learning*, se abre un campo de posibilidades de uso en la robótica de *swarm*, puesto que podría facilitar la tarea de elegir la solución más conveniente en función de las condiciones a las que se se enfrente el *swarm*.

- **Adaptabilidad**, se trata la capacidad de que el *swarm* funcione correctamente en un amplio rango de tamaños, por ejemplo, en el caso que nos ocupa, (en el que se va a abordar la formación de figuras) se entendería como la competencia de disponerse en distintas formas de mayor o menor tamaño.

Una vez contextualizado el concepto y los antecedentes de la robótica de *swarm*, cabe mencionar que existen otros tipos de robótica multi-agente. En base a lo expuesto anteriormente, se deducen las ventajas más significativas que ofrece la robótica de *swarm* frente a estos últimos, motivo por el que se emplea para la configuración

(a) Pájaros *flocking*.

(b) Banco de peces.

Figura 1.1: Swarms en la naturaleza.

distribuida de formas, objeto de este Trabajo de Fin de Grado. Entre las mencionadas ventajas, se quiere destacar el coste mínimo del hardware. Como se ha comentado, los robots empleados son simples, ofreciendo percepción local por medio de sensores de rango bastante limitado y comunicación local de similares características. También se ve favorecido por la homogeneidad de los robots, minimizando la especialización y división de roles dentro del *swarm*.

## 1.2. Ámbitos de aplicación

Las tareas que típicamente se busca afrontar empleando *swarms* de robots son aquellas que se sitúan en entornos inaccesibles o potencialmente peligrosos, como puede ser el desminado o limpieza de un campo de minas. Esto se debe a que, gracias a la robustez inherente de los *swarm* y a la descentralización que los caracteriza, se puede permitir perder algunos robots para completar la tarea, sin afectar necesariamente a la actuación de todo el *swarm*. Se detallarán algunos proyectos de interés en la Sección 1.3. Los ámbitos de aplicación se pueden agrupar en dos categorías diferenciadas que se van a comentar acto seguido:

1. La primera de ellas se enfoca en problemas que involucran patrones, como pueden ser tareas de agregación, cartografía, despliegue de agentes distribuidos o la cobertura de una región, entre otros. En esta categoría se incluye la configuración distribuida de formas, objeto de este Trabajo de Fin de Grado. Un caso particular de aplicación de los algoritmos que se han desarrollado, podría ser su empleo para

monitorizar la calidad del agua de un lago. En caso de que hubiese un vertido de alguna sustancia química perniciosa, el *swarm* podría movilizarse a la zona del vertido para poder solventar el problema.

2. La segunda categoría se focaliza en problemática relacionada con entidades en el entorno, como puede ser la búsqueda de objetivos, en aplicaciones de seguridad o vigilancia, así como en rescate en zonas peligrosas o inaccesibles para el ser humano. Un caso ejemplo, sería la localización de vetas de mineral en entornos inexplorados.

De todos modos, téngase en cuenta que la robótica de *swarm* puede tener que abordar problemáticas más sofisticadas, de forma que ambas categorías descritas anteriormente se vean involucradas al tiempo en tareas como transporte cooperativo, exploración de un planeta y navegación en entornos muy extensos entre otras.

## 1.3. Proyectos de interés

A continuación se incluyen algunos proyectos en los que se ha empleado la robótica de *swarm* y en los que se materializan los ámbitos de aplicación descritos en la Sección 1.2. Los algoritmos desarrollados en este Trabajo de Fin de Grado pueden emplearse en proyectos análogos a los que se describen a continuación, aportando herramientas para optimizar el despliegue y la operación de los *swarm* involucrados.

### 1.3.1. Seaswarm

El MIT Senseable City Lab desarrolló una flota de robots *low-cost* de absorción de petróleo denominados *Seaswarm* [5], con el objetivo de ser capaces de sumergirse en el océano y eliminar petróleo vertido. Los robots que conforman el *swarm* encargado de dicha tarea, están constituidos por un nanomaterial que permite la absorción de petróleo de hasta 20 veces el peso de un robot individual. Se trata de una propuesta económica para preservar el medio oceánico.

### 1.3.2. Proyectos militares

La robótica de *swarm* puede emplearse también para aplicaciones militares, según se propone en [6]. Se trata de un sistema multi-robot auto configurable para búsqueda y rescate en entornos vastos y capaz de lidiar con fallos ocasionales gracias a su inherente flexibilidad.

### 1.3.3. iRobot swarm project

Con el fin de tener una perspectiva de la escala de los proyectos de robótica de *swarm*, se ha querido incluir el iRobot swarm project, desarrollado por el MIT [1] para lograr la cooperación de más de 100 robots. El objeto del iRobot swarm project no es otro que desarrollar algoritmos distribuidos para *swarms* de gran escala (del orden de cientos de robots) y entornos complejos reales, poniendo a prueba su tolerancia a errores de varios individuos del *swarm*. En el proyecto se introduce un sistema de monitorización para todo el *swarm*, así como una estación de recarga automática para los robots.

### 1.3.4. I-swarm project

En última instancia, cabe mencionar el I-swarm project [7] que aúna micro-robótica, sistemas distribuidos y adaptativos, y sistemas de *swarm* biológicos auto adaptables. Su pequeño tamaño permite trabajar con *swarms* de más de 100 robots, que cuentan con sensores de percepción local, y que facilita el trabajo con ellos para aplicaciones médicas (para explorar órganos internos, por ejemplo) a un relativo bajo coste.

## 1.4. Plataformas de simulación

Típicamente, el trabajo en aplicaciones de robótica de *swarm* requiere el empleo de una gran cantidad de robots, de modo que su coste puede llegar a ser bastante prohibitivo para instituciones dedicadas a la investigación. Para que no sea necesario hacer grandes inversiones en *hardware*, las plataformas de simulación se erigen como una solución intermedia, proporcionando una aproximación de la realidad lo suficientemente buena. En general, los entornos de simulación son más sencillos de emplear, además de ser considerablemente más baratos y rápidos. Se incluyen a continuación un par de las plataformas de simulación más empleadas, de acuerdo con la literatura consultada.

### 1.4.1. Webots

Webots [8] es una plataforma de uso ampliamente extendido (se cuentan hasta 1018 instituciones universitarias entre sus usuarios) que se utiliza para modelar, programar y simular robots móviles. Admite el diseño de configuraciones complejas de robots, contando con una gran selección de sensores y actuadores disponibles. Permite configurar los objetos presentes en el entorno, además de proporcionar un controlador para testear los robots reales. Gracias a su uso generalizado, hay disponible documentación de calidad al respecto de su manejo, y hay APIs disponibles para C,

C++, Python, Java, MATLAB o ROS. Antes de remitir el código se hace una revisión de su idoneidad para comprobar la calidad del mismo.

### 1.4.2. CoppeliaSim (V-REP)

CoppeliaSim [9] (anteriormente conocido como V-REP) es un simulador de robots 3D que se puede emplear como recurso abierto, y que permite crear sistemas completos de robots, incluyendo su simulación e interacción con *hardware*. CoppeliaSim, al igual que su predecesor V-REP, se basa en una arquitectura de control distribuida, articulada en *scripts* de control que se pueden adjuntar directamente a los objetos presentes en el escenario de simulación, permitiendo así que sea un entorno versátil y perfecto para trabajar con aplicaciones de multi-robots, logrando una aproximación de la realidad bastante buena. Posee herramientas y módulos muy útiles entre los que se puede destacar: simulación de sensores de proximidad y de cámaras, modelos cinemáticos directo e inverso, *path planning*, cálculo de distancia mínima, gestión de gráficos y un largo etcétera.

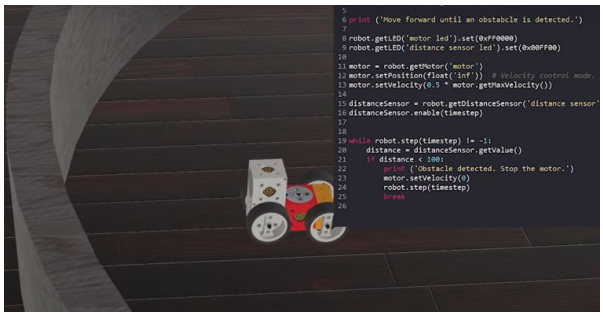



Figura 1.2: Interfaz de Webots.

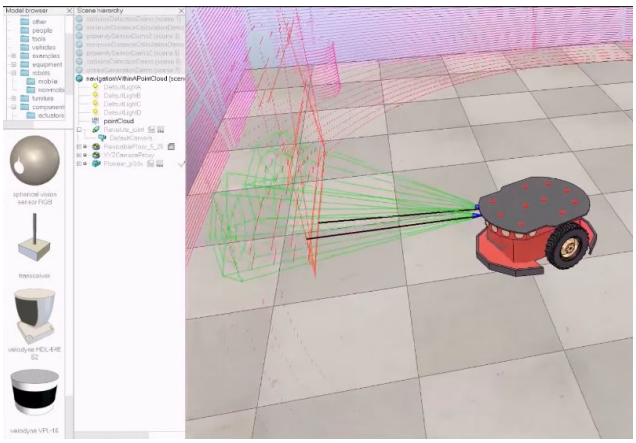


Figura 1.3: Interfaz de CoppeliaSim/V-REP.

# Capítulo 2

# Introducción a The Robotarium (*by GeorgiaTech*)

## 2.1. Contexto

Los *testbeds* (bancos de pruebas) existentes en el campo de sistemas multi-robots empleados para la investigación tienden a ser costosos, excesivamente complejos y requieren una elevada inversión temporal para desarrollar, operar y mantener el equipo. En [10] se puede adquirir una visión general del contexto mencionado. En el marco de esta realidad, el Robotarium proporciona a sus usuarios un acceso remoto a las instalaciones de multi-robots con tecnología *state-of-the-art*, garantizando la seguridad del equipo ya desde la etapa de diseño, según se explicará en más detalle en la sección 2.2.3.

En el desarrollo de este Trabajo de Fin de Grado, se ha escogido el empleo del Robotarium puesto que, el resto de *testbeds* existentes presentan problemáticas en lo referente al despliegue del *swarm* de robots, puesto que es complicado simular las vicisitudes que puede implicar la realización coordinada de tareas con sistemas multi-robots, y cabe resaltar que no todos consideran las medidas de seguridad en el despliegue, dejándolas a expensas del usuario. Algunos ejemplos de otros *testbeds* de multi-robots son Mobile Emulab [11] y HoTDeC [12].

## 2.2. The Robotarium

Tal y como se ha mencionado, el factor diferenciador del Robotarium [13] al respecto de otros *testbeds* disponibles, es su atención puesta en promover un acceso seguro a la investigación de sistemas multi-robot.

Las medidas de seguridad que incorpora el Robotarium [13] incluyen evitar colisiones peligrosas y su cuantificación a través de una puntuación de seguridad (*safety*

*score*) que determina si el código remitido por el usuario se puede ejecutar sin poner en consideración medidas de seguridad adicionales. En síntesis, el Robotarium permite al usuario acceder remotamente al *hardware*, independientemente de sus intenciones, evitando todo tipo de daños y erigiéndose como una opción segura y flexible para trabajar en sistemas de multi-robots, que acerca esta rama de la robótica a todo tipo de usuarios.

Sus características más reseñables son las que siguen:

- Se trata de un *testbed* robusto, que proporciona operación segura y a largo plazo de grandes grupos de robots.
- Garantiza la mínima intervención humana, reduciendo la necesidad de mantenimiento no automático.
- El mantenimiento automático recae en un seguimiento robusto de la posición de los robots, recarga automática de la batería y la ejecución del movimiento sin colisiones para evitar daños.

### 2.2.1. *Hardware*

Los robots empleados se denominan GRITSBots, y cuentan con un sistema de seguimiento de posición, *hardware* de comunicación inalámbrica y sistema de recarga automático. La placa base de los GRITSBots emplea el ESP8266, chip que admite comunicación WiFi que opera a 160 MHz y permite que el robot pueda llevar a cabo comunicación inalámbrica, estimación de la posición, el control del robot a bajo nivel, además de su comportamiento de alto nivel. Según se comentaba, el ESP8266 permite la comunicación inalámbrica, por lo que se puede actualizar el código en los robots en cuestión de segundos. En lo que respecta al seguimiento de posición, se lleva a cabo empleando una *webcam* en uso combinado con ArUco (librería de OpenCV para aplicaciones de realidad aumentada). El mecanismo empleado para la recarga automática de las baterías de los robots, hace uso del estándar de carga *Qi wireless*, cada robot tiene una bobina receptora anexionada, y el espacio físico en el que se realiza el despliegue (*Robotarium Arena*) cuenta con transmisores en su superficie.

### 2.2.2. *Software*

El Robotarium cuenta con infraestructura para simulación y virtualización (también empleada para verificación de código basada en simulación), componentes de interacción (API) y el servidor de coordinación. Como ya se ha mencionado repetidamente, el código remitido por los usuarios debe superar una verificación basada

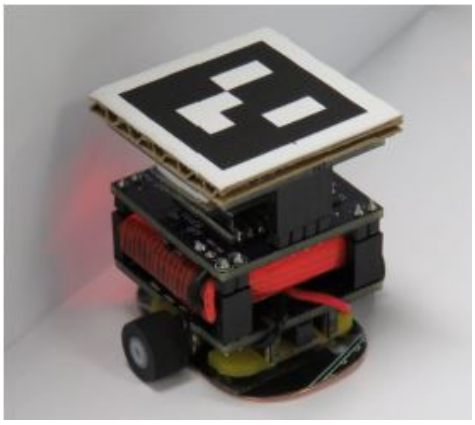


Figura 2.1: GRITSBot cargándose en una estación de Robotarium

en simulación, que determina si la ejecución del mencionado código está exenta de errores y colisiones antes de su ejecución en el entorno real. Respecto a la coordinación entre todos los robots desplegados, véase que el código a propósito del control se ejecuta en el servidor y realiza el control remoto de los robots, proporcionando los comandos de control de velocidad a los robots. Ha de ponerse en consideración, no obstante, que los robots se gestionan de forma centralizada en el Robotarium, lo que puede propiciar errores en los experimentos realizados en caso de que falle alguno de los robots que componen el *swarm*, desembocando en cuellos de botella al respecto de la comunicación y la computación del código.

### 2.2.3. Medidas de seguridad

Ya se ha referido repetidamente la política del Robotarium [13] de asignar puntuaciones de seguridad a los experimentos, cuyo cometido es determinar su idoneidad para ser desplegados con robots reales en la *Robotarium Arena*. Esta puntuación de seguridad (*safety score*) se basa en simulaciones empleando la descomposición de Monte Carlo para computar los valores esperados de daño y seguridad del experimento en varias simulaciones (típicamente 50). En suma, la puntuación de seguridad constituye una medida de la frecuencia y severidad de las colisiones en simulación, cuyo objetivo es predecir lo más fielmente posible el comportamiento de las colisiones cuando se prueba el código en el *hardware* del Robotarium [13].

### 2.2.4. Brecha simulación-entorno real

Por último, cabe comentar la inevitable brecha que existe entre el entorno de simulación y el *hardware* empleado para la ejecución de un mismo experimento. Puede parecer obvio, pero es importante que los simuladores proporcionen una aproximación

lo suficientemente razonable del comportamiento de los robots reales, puesto que el código que se prueba en el entorno de simulación es exactamente el mismo que se emplea posteriormente en el entorno real. Esta brecha entre simulación y realidad se puede caracterizar matemáticamente por medio de regresión lineal sobre los datos existentes, que recaen en el modelo dinámico de los GRITSBots (*unicycle*). Viendo el desarrollo matemático descrito con detalle en [13], es posible calcular el error medio entre las trayectorias simulada y real.

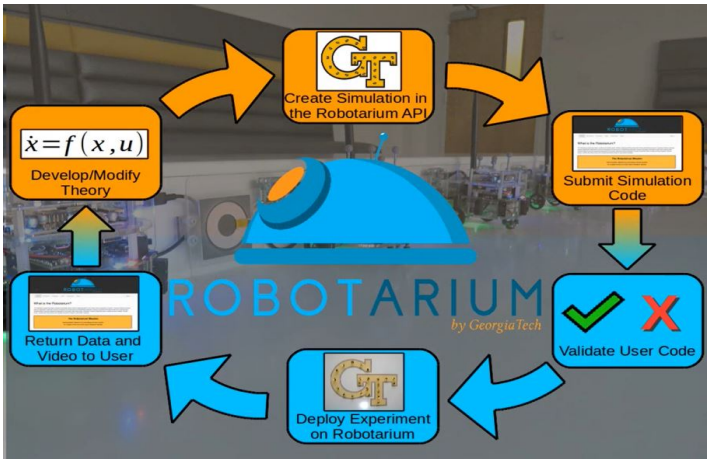



Figura 2.2: Gestión de experimentos en The Robotarium

# Capítulo 3

# Arquitectura de la implementación

## 3.1. Nivel intermedio

Téngase en consideración que el Robotarium está articulado de forma centralizada, esto es, la posición y orientación de los robots se recogen en una matriz global de dimensiones 3xN, siendo N el número de robots del *swarm*. Así pues, se tiene la información de todo el *swarm*, permitiendo el cálculo de la velocidad lineal y angular que se transmitirá a los actuadores de los robots para su movimiento. Dicha información se almacena en una matriz (también global) de dimensiones 2xN, y se obtiene empleando el control automático.

Puesto que en este Trabajo de Fin de Grado se ha propuesto apostar por un enfoque de robótica de *swarm* (plenamente distribuida), la filosofía centralista del Robotarium resulta del todo incoveniente, en vista de que en todo momento se conoce cuántos robots hay en el *swarm*, así como su localización en el espacio. A propósito de organizar el algoritmo con un enfoque más descentralizado, característico de la robótica de *swarm*, (cuyas bondades se han explicado en el Capítulo 1) se propone lo siguiente.

Cada uno de los robots se va a gestionar como un ente individual, esto es, a priori conoce únicamente su propia información. Proporcionando un ejemplo concreto y a propósito del algoritmo de referencia [14], cada robot es consciente de si se trata de un robot interno o de contorno, pero no conoce dicha información del resto de robots que componen el *swarm*.

Indefectiblemente, ello conduce a la necesidad de interacción entre los robots del *swarm*. De acuerdo con la perspectiva de interacción local propia de la robótica de *swarm*, se han implementado algunas funciones de percepción local simuladas (en vista de que los robots del Robotarium no disponen de dichas funcionalidades).

En esta línea, cabe destacar el sensor *range and bearing*, que proporciona las lecturas de los robots 'visibles' desde un robot concreto (distancia y orientación). La visibilidad de los robots se ha configurado estipulando un radio de percepción limitado.

Por otro lado, cada uno de los robots tiene un alma (o varias). Su *raison d'être* no es otra que proporcionar una analogía, de nuevo, a la operación temporal de los algoritmos propios de la robótica de *swarm*. La actualización de las almas se realiza de forma periódica (con un periodo de muestreo o *step* definido, T). De este modo, ello se asemeja al tiempo de computación de la información de los sensores de percepción (de tipo *range and bearing*), empleada para calcular la velocidad con la que ha de moverse el robot en cuestión.

En suma, con el fin de manejar la interfaz del Robotarium acorde con la filosofía descentralizada y local típica de la robótica de *swarm*, se ha articulado el código en clases según sigue:

- **Robot:** se trata de un robot en movimiento en un espacio Robotarium. Contiene información básica sobre sí mismo, por ejemplo, si se trata del robot que establece el origen del sistema relativo de coordenadas (*seed robot*) o si es alguno de los dos robots (B o C) que se van a emplear para configurar los ejes $X$ e $Y$ del citado sistema de coordenadas (véase el algoritmo en el Capítulo 5).

  Puesto que el robot estará en movimiento, es necesario considerar los parámetros de bajo nivel que se calculan en la API de Robotarium. Entre ellos, un parámetro indispensable: la posición actual de cada uno de los individuos. Este parámetro permite conocer cuánto le queda a dicho robot en movimiento para alcanzar su lugar dentro de la formación final (en caso de ser de contorno), así como hallar la velocidad a la que debe moverse el robot para alcanzar la posición final en el tiempo deseado (respetando los umbrales de velocidad lineal y angular estipulados por el Robotarium).

- **Shape:** se trata de la forma que se quiere obtener con el *swarm*, en este caso un polígono de $4n$ lados. Adicionalmente, aporta apoyo visual, muy útil para dirimir cuán precisa es la formación final en la etapa de verificación.

- **Space:** consiste en un espacio (ya sea real o simulado) poblado por un número dado de robots en movimiento (N). Dicho espacio se ha configurado de tal forma que los robots, salvo que se especifique lo contrario, estén dispuestos de manera

arbitraria antes de comenzar con la formación deseada (típicamente, agrupados dentro de la figura definida en *shape*). Obsérvese que la clase *Space* contiene todas las funciones de percepción que se comentaban brevemente al introducir esta sección. A saber, el sensor *range and bearing*, cuyo cometido es devolver las lecturas de los robots vecinos a uno concreto, *nearby*, que proporciona un listado de los mencionados robots vecinos, o *nearest*, para informar de cuál es robot más cercano.

En *Space* también se incluye la función *step*, a propósito de la actualización de la dinámica de todos los robots de forma global (empleando la función análoga del Robotarium) y de las almas de cada uno de los robots. Se ha implementado además, el control del algoritmo distribuido de Lloyd con las funciones de control propias del Robotarium, empleando la información contenida en el alma de cada robot asociada a *Voronoid*.

A expensas de la posibilidad de errores o comportamientos dispares entre simulación y *hardware*, propiciados por la particularidad de la gestión de tiempo, se han realizado dos experimentos incluidos en la Sección 3.3 de este Capítulo.

- **Knowledge:** se trata de una definición básica de lo que conoce el alma de un robot. Admite especialización según el algoritmo con que se esté trabajando, por lo que en primera instancia únicamente se ha añadido una variable *estado* a modificar o informar según convenga.

- **Soul:** es el alma del robot, y su cometido es controlar y manipular el comportamiento de cada uno de los robots que pueblan un espacio. Téngase en cuenta que un robot individual puede tener varias almas, y que de forma general son llamadas por el bucle principal de forma periódica (típicamente transcurrido el tiempo de muestreo T) para actualizar el comportamiento del robot en cuestión.

- **Voronoid:** especialización de *Soul*. Involucra la computación de las regiones de Voronoi de cada uno de los robots según el algoritmo descrito en la Sección 5.2 y el movimiento de los robots al centroide de su celda de Voronoi correspondiente.

- **APF:** especialización de *Soul*. Comporta el cálculo de las fuerzas virtuales que generan el campo de potencial artificial, y su consiguiente traducción en las velocidades lineal y angular con las que se pretende que se mueva el robot. Se describe su funcionamiento en detalle en la Sección 5.1.

En última instancia, cabe destacar el archivo *main* (ejecutable), en el que primero se procede a inicializar el espacio Robotarium, se añaden los robots deseados (con sus

respectivas almas) y, por último, se configura el bucle principal que no se detiene hasta que haya concluido la formación (*shape*) deseada por el usuario.

## 3.2. Gestión de la sincronización de los robots

Los robots pueden estar sincronizados o no, en este caso se propone emplear un modelo asíncrono, en el que las acciones de los robots son independientes de las del resto del *swarm*. En el momento que un robot completa el procesado de los datos procedentes de sus sensores, puede ser que, por ejemplo, varios robots ya se estén moviendo hacia su posición objetivo.

La secuencia de computación de cada uno de los robots que componen el *swarm* es un ciclo la esbozada en la Sección 3.1, constituida por cuatro pasos según [15]:

1. Espera (*Wait*): añadido a expensas del modelo asíncrono, puesto que los robots operan en ciclos independientes de duración variable.

2. Mirar alrededor (*Look*): el robot registra su alrededor, identificando al resto de robots vecinos.

3. Procesado de la información (*Compute*): ejecución del algoritmo con el fin de obtener el punto de destino de cada robot. El algoritmo es el mismo para todos los robots del *swarm*.

4. Movimiento (*Move*): el robot se dirige hacia el punto objetivo.

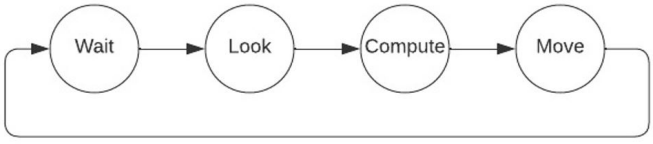



Figura 3.1: Secuencia de computación.

## 3.3. Validación de la gestión

Antes de comenzar a exponer los algoritmos, se han realizado los experimentos 1 y 2 con el objetivo de dilucidar si la gestión de la sincronización de los robots y de los periodos de muestreo como se ha descrito en la Sección 3.1 de este Capítulo interfiere con la operación temporal del Robotarium (que funciona con *steps* de 33 ms).

### 3.3.1. Experimento 1

Acceso al vídeo completo del experimento desarrollado en el *hardware* del Robotarium: **vídeo experimento 1**.

En este primer experimento se ha probado la sincronización expuesta en la Sección 3.1. El objetivo es analizar si la gestión temporal del Robotarium y la que se ha implementado con el fin de darle un enfoque más descentralizado, son compatibles a nivel de control.

Para ello, se han asignado las mismas coordenadas iniciales (3.1) y puntos destino (3.2) a este y al Experimento 3.3.2 para comparar los tiempos de ejecución, en simulación y en los robots. Esto permite determinar si hay alguna disonancia entre las dos formas de gestión y si la ejecución en los robots reales del Robotarium se asemeja a los resultados obtenidos en simulación.

$$start = \begin{bmatrix} 0{,}6 & 0{,}9 & -0{,}9 & -0{,}9 & 0{,}6 & 0 \\ -0{,}3 & -0{,}9 & 0{,}3 & 0 & 0{,}6 & -0{,}3 \\ 0{,}478 & 2{,}706 & 0{,}764 & -0{,}654 & 2{,}797 & -3{,}09 \end{bmatrix} \tag{3.1}$$

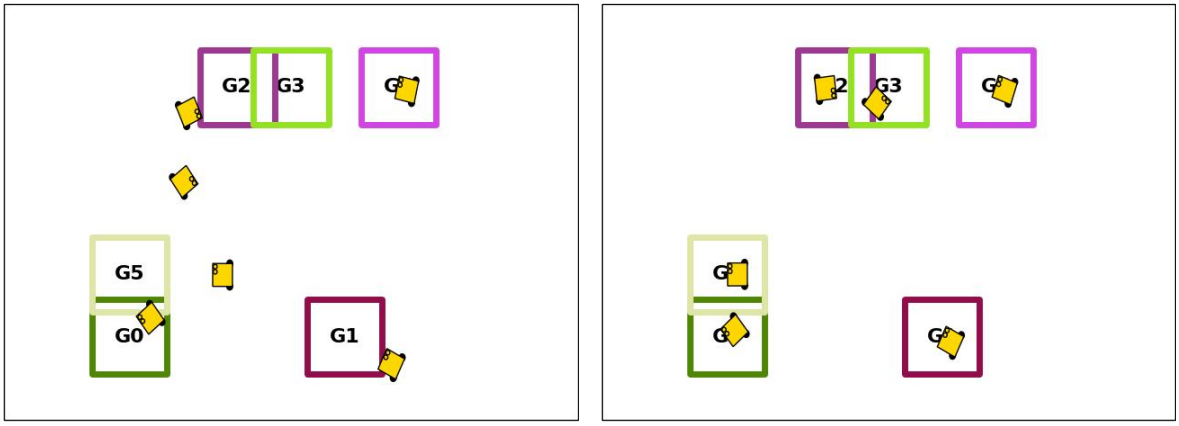
$$goals = \begin{bmatrix} 0{,}9 & 0{,}3 & 0{,}3 & 0 & 0{,}6 & 0{,}9 \\ 0{,}6 & -0{,}6 & 0{,}6 & 0{,}6 & 0{,}6 & -0{,}3 \\ 1{,}235 & 1{,}681 & -0{,}836 & 0{,}031 & 0{,}439 & 0{,}853 \end{bmatrix} \tag{3.2}$$

Dichos resultados se incluyen en la Figura 3.2 para el caso de simulación, y en la 3.3 para el experimento con el *hardware* del Robotarium; a priori no se observa ninguna disonancia aparente entre ambos. Los registros del tiempo que ha tomado realizar el experimento figuran en la Tabla 3.1.

### 3.3.2. Experimento 2

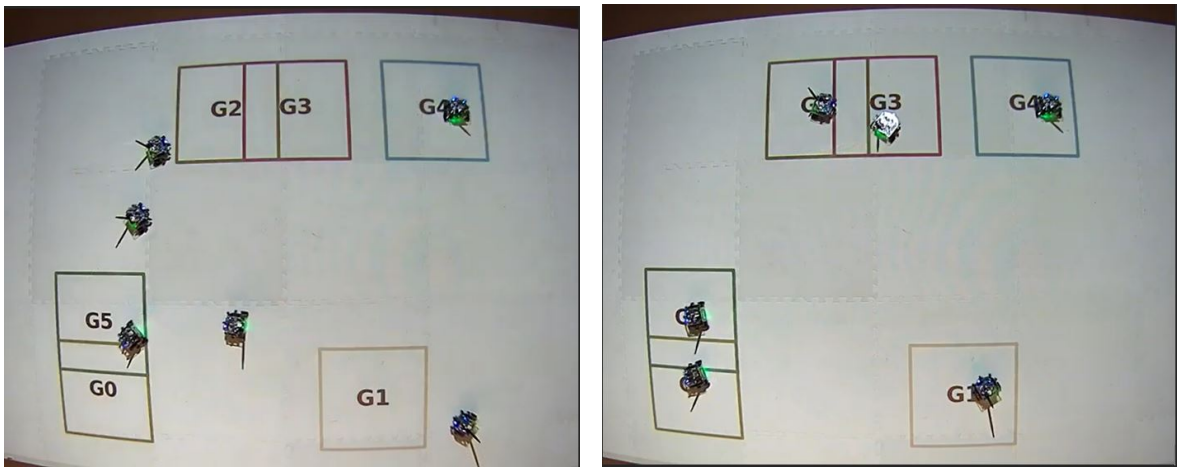
Acceso al vídeo completo del experimento desarrollado en el *hardware* del Robotarium: **vídeo experimento 2**.

Como se comentaba al introducir el experimento 3.3.1, se ha tomado un archivo aportado en la API del Robotarium (*si go to point with plotting*) y se han especificado las mismas condiciones iniciales y finales que el el caso 3.3.1, indicadas en las matrices 3.1 y 3.2. Los resultados obtenidos en simulación y experimentación con el *hardware* del Robotarium, se recogen en las Figuras 3.4 y 3.5, respectivamente. Según se ve en las citadas imágenes, el resultado es idéntico al observado en las Figuras 3.2 y 3.3, y el tiempo que tarda en realizarse el experimento es el mismo que el del anterior, (8 segundos aproximadamente, ver Tabla 3.1) por lo que se deduce que la gestión implementada es del todo compatible con la del Robotarium.




(a) En movimiento hacia el destino. (b) En destino.

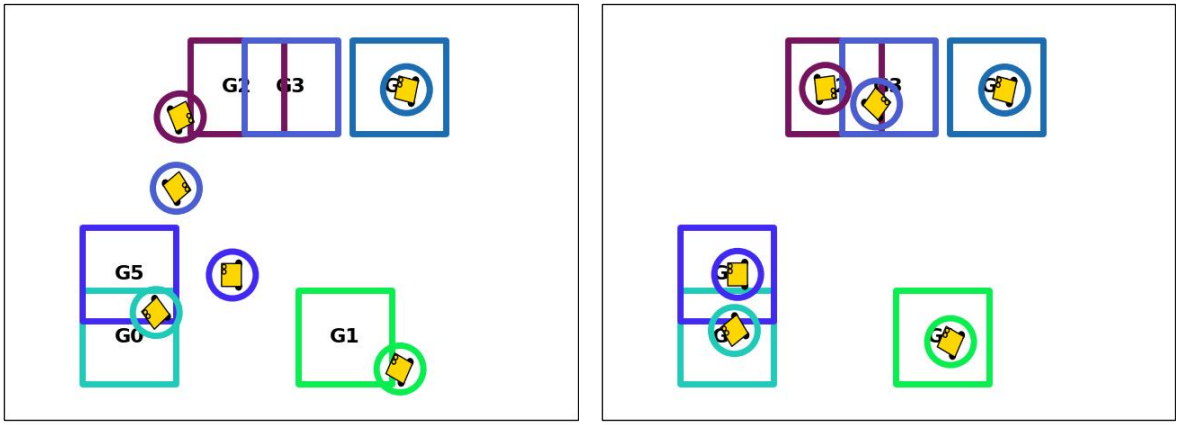
Figura 3.2: Experimento 1 (simulación): con gestión temporal descrita en la Sección 3.1




(a) En movimiento hacia el destino. (b) En destino.

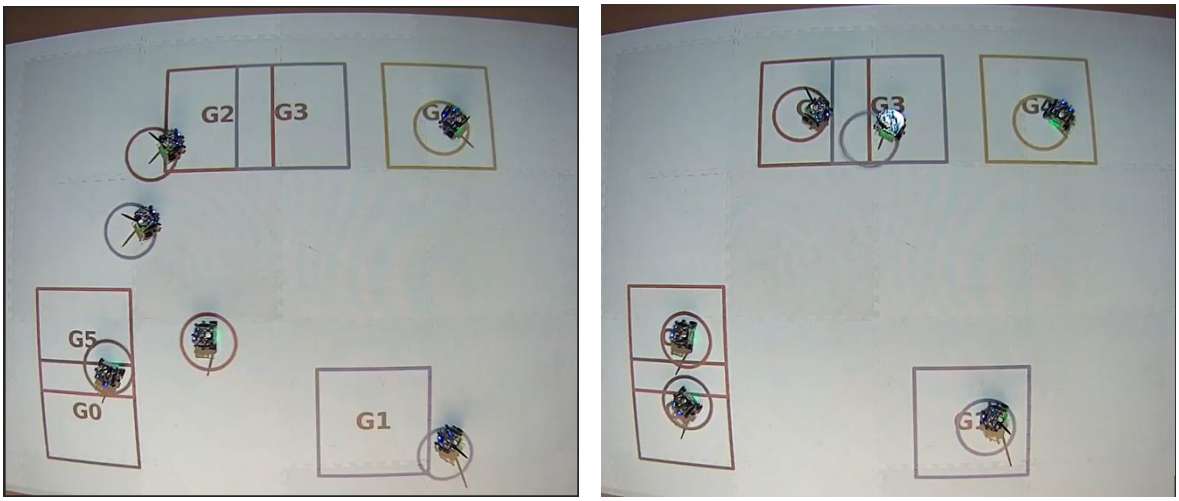
Figura 3.3: Experimento 1: con gestión temporal descrita en la Sección 3.1

Una vez dilucidada la idoneidad de la nueva arquitectura de la implementación, se puede proceder a la validación de la implementación de los algoritmos que se van a introducir y cuyos resultados se incluyen en el Capítulo 6.




(a) En movimiento hacia el destino. (b) En destino.

Figura 3.4: Experimento 2 (simulación): *si go to point with plotting alike*.




(a) En movimiento hacia el destino. (b) En destino.

Figura 3.5: Experimento 2: *si go to point with plotting alike*.

| Tiempos de ejecución | | |
|---|---|---|
| Experimento | Nº de robots | Tiempo de ejecución (s) |
| 1 | 6 | 8 |
| 2 | 6 | 8 |

Tabla 3.1: Resultados obtenidos en simulación y despliegue con Robotarium.

# Capítulo 4

# Artículo de referencia: *A parallel shape formation method for swarm robotics*

Se va a proceder a describir el artículo que se ha tomado como referencia para la implementación del algoritmo desarrollado. También se detalla la problemática asociada al método que expone, así como las mejoras propuestas a realizar.

## 4.1. Propuesta

En Yang et al. [14] se propone emplear un método de formación paralela para robótica de *swarm*, con el fin de cumplir con tareas de diversa índole en entornos no estructurados, que se va a proceder a implementar y verificar empleando la infraestructura del Robotarium. El objetivo que se pretende lograr es que la configuración del sistema se pueda ajustar de manera autónoma y flexible, en función de una forma 2D especificada por el usuario. A tenor de ello, la formación tendrá lugar en dos subtareas a ejecutarse en paralelo: la formación de los robots de la periferia del enjambre y la cobertura del espacio restante por parte de los robots de interior. Mientras tiene lugar el movimiento, los robots actualizan su posición local en tiempo real hasta que se haya concluido el proceso de formación.

Los sistemas multi-robots basados en control centralizado enfrentan típicamente baja tolerancia ante fallos (si tiene lugar un error en uno de los robots, ello puede desencadenar en un fallo general del sistema), alto coste computacional y adaptabilidad limitada cuando se trabaja con problemas complejos y a gran escala. Para solucionar este problema, la robótica de *swarm* se presenta como una buena alternativa según se ha discutido detalladamente en el Capítulo 1. Inspirada en comportamientos de seres gregarios en la naturaleza, la robótica de *swarm* permite disponer a los robots

del *swarm* (cuyo rango de percepción es limitado) de manera eficiente y uniforme en el área destinada a la formación, a través de la comunicación entre los robots individuales y sin necesidad de control centralizado.

En este contexto, se va a proceder a analizar el método de formación paralela propuesto por Yang et al. [14], en el que el *swarm* cambia su configuración geométrica dependiendo de la tarea (forma 2D) asignada, de forma autónoma y flexible. Este método va a permitir a los robots individuales completar la configuración de forma flexible, autónoma y robusta.

## 4.2. Problemática observada en tareas de formación

En la literatura dedicada al estudio de robótica de *swarm* se aprecia que, de forma generalizada, sigue habiendo un problema de eficiencia en las tareas de formación. Ello se debe a que todavía no hay maneras efectivas y precisas de llevar a cabo la formación versátil para figuras arbitrarias. Se adjuntan a continuación algunos ejemplos de artículos dedicados al estudio de problemas similares y las vicisitudes que enfrentan:

### 4.2.1. Morfología de las figuras

No es posible realizar determinadas formas, como ocurre en el caso de [16], que propone un algoritmo basado en un robot que actúa como conductor o líder del *swarm*, y el resto de robots forman una cadena de vecinos o *amigos*, de forma que cada robot está referenciado a un *amigo*. Se forma pues, una cadena en la que cada robot posee un ID (identificador) a transferir al resto del *swarm*. La naturaleza del algoritmo no permite formaciones como J, U o ˜, esto es, figuras con 'cabos sueltos'.

Ocurre una situación similar en el caso de [17], en el que las figuras que se pueden generar por medio del algoritmo planteado se ven limitadas por las matemáticas subyacentes a dicho algoritmo.

### 4.2.2. Tiempo de formación

Se invierte demasiado tiempo en la formación porque la distribución inicial de los robots está muy apartada de la formación que se desea alcanzar, según se ha concluido en [18]. La propuesta inicial en este artículo consiste en formar una configuración 2D valiéndose de 1024 Kilobots, operando de forma autónoma y empleando un algoritmo de auto-ensamblaje para realizar la formación.

### 4.2.3. Cobertura del espacio interno

Solo se completa el despliegue de los robots de la periferia del *swarm* y no se concluye con éxito la cobertura uniforme del espacio interno en el caso de [19]. La propuesta realizada para trabajar con el sistema de robótica de *swarm* es una *gene regulatory network* (GRN, en español red de regulación génica), inspirada en organismos biológicos para la organización de los robots en distintas figuras. Los sistemas biológicos, como se ha comentado en la introducción 1.1, pueden generar comportamientos robustos y formas complejas valiéndose de interacciones locales entre los agentes en presencia de incertidumbre. El proceso de formación se trata de optimizar empleando un algoritmo genético.

### 4.2.4. Brecha simulación-hardware

No se toma en cuenta la implementación en el hardware, como ocurre en [20], planteando un controlador de forma basado en regiones para un *swarm* de robots. Con este método de control, el movimiento de los robots como conjunto tiene lugar en una región determinada, manteniendo una distancia de seguridad entre ellos.

## 4.3. Problemática observada en Yang et al. y líneas de mejora del algoritmo

Los problemas que se han advertido en la verificación del algoritmo de Yang et al. [14] están relacionados con el despliegue del *swarm* mediante el método del APF (*Artificial Potential Field method*), más concretamente, en el proceso de despliegue de los límites de la forma (realizado por los robots de periferia del *swarm*). Según se ha podido constatar, el error en la posición de los mencionados robots aumenta conforme lo hace el tamaño de la figura a formar. Ello se debe a que los robots se localizan en el *swarm* a través de navegación por estima (*dead-reckoning*), implicando que los errores de posición de los robots son acumulativos.

Nótese que en ocasiones hay algunos huecos más grandes entre algunos robots del contorno, como ocurre en el último fotograma observado en la Figura 4.1. Cabe comentar que no tiene efectos serios significativos sobre el resultado final.

Como alternativa al método del APF para el despliegue del *swarm*, se propone emplear un algoritmo de Lloyd de implementación distribuida cuyo último objetivo es cubrir el espacio ocupado por la figura con regiones de Voronoi (o *centroidal Voronoi*

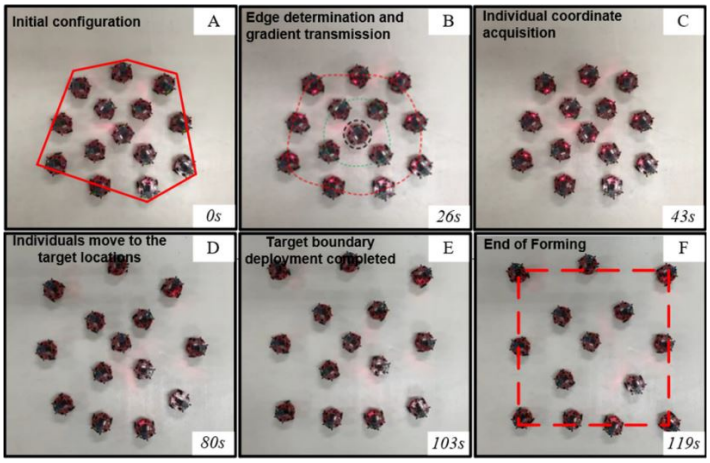



Figura 4.1: Progreso de la formación de un rectángulo procedente de [14].

*tessellations*). La posición destino de los robots será el centroide de dichas celdas de Voronoi, cuya información se actualiza en cada iteración del algoritmo.

Este tipo de algoritmo, inspirado en Cortés et al. [21] y discutido posteriormente en más detalle, presenta sus ventajas e inconvenientes frente al método del *Artificial Potential Field*, por lo que resultaría de gran interés comparar los resultados de la formación implementando ambos, con el fin de dirimir cuál resulta más apropiado para la tarea. Los resultados y discusión sobre la idoneidad de los dos métodos se incluyen en el Capítulo 6.

# Capítulo 5

# Descripción del algoritmo

## 5.1. Algoritmo propuesto

A continuación, se detalla el algoritmo de formación sugerido en Yang et al. [14], que, según se ha introducido en la Sección 4.1, está basado en un mecanismo de formación paralela. Así pues, se va a proceder a implementar dicho algoritmo con el fin de visualizar su funcionamiento en simulación y en la realidad a través de la infraestructura del Robotarium (ver resultados en el Capítulo 6).

El funcionamiento del algoritmo propuesto es el que sigue:

1. Obtención de la configuración para la forma deseada.
2. Diferenciación de tareas de acuerdo con la posición inicial (estática) de cada individuo. Para ello se debe determinar si el robot es de periferia o interno.
3. Se selecciona un robot localizado aproximadamente en el centro del *swarm* (denominado robot semilla o *seed robot*) por medio de un mecanismo de generación y transmisión de gradiente, con el fin de establecer un sistema de coordenadas relativo.
4. Se elige un robot (B) para formar el eje X, y otro robot (C) para el eje Y del sistema de coordenadas relativo.
5. La posición de cada uno de los individuos se determina por medio de posicionamiento trilateral (*trilateral positioning*) y navegación por estima (*dead reckoning*).
6. Por último, se emplea el método del campo de potencial artificial para disponer los robots de periferia en la forma deseada y realizar la cobertura de la zona interior de manera uniforme.

Seguidamente se incluye el Algoritmo 1, detallando lo expuesto anteriormente y evidenciando el proceso de formación paralela característico que se ha comentado, en función de la diferenciación de tareas.

**Algorithm 1** Proceso de formación paralela (en función de la distribución de tareas)

posiciones iniciales (robots agrupados)
**paso 1:** diferenciación de tareas:
**if** $\theta \geq 120^{o}$ **then**
  *edge_per* = 1 ▷ es robot de contorno
  **paso 2:** despliegue del contorno de la figura
  generación de la señal de gradiente
  transmisión del gradiente al resto del *swarm*
  *trilateral positioning*: cálculo de coordenadas relativas, esperar información del entorno
  **while** edge robot *not* en *target positions* **do**
    cálculo de la dirección del movimiento (fuerza resultante)
    robot en movimiento
    actualización de las propias coordenadas actuales
  **end while**
  *stop*
**else**
  *edge_per* = 0 ▷ es robot interno
  **paso 2:** cobertura del espacio interno de la figura
  **if** gradiente recibido=i **then**
    envío de *feedback* a los vecinos
    transmisión de gradiente= $i + 1$ a los vecinos
    **if** *feedbak* no recibido **and** *feedbak* de centro no recibido **then**
      envío de *feedbak* de centro
      establecimiento del sistema de coordenadas relativo
    **end if**
  **end if**
  **while** fuerzas no equilibradas **do**
    cálculo de la dirección del movimiento (fuerza resultante)
    robot en movimiento
  **end while**
  *stop*
**end if**

### 5.1.1. Disposición inicial y descripción de la forma

Inicialmente los robots permanecen estáticos y agrupados en el interior del polígono de 4n lados que describe la formación a alcanzar.

Según el artículo de referencia [14], la figura a describir (en Yang et al. [14] se prueban formaciones de cuadrado, triángulo y círculo) por los robots de contorno se computa por medio de funciones analíticas con el fin de calcular las posiciones objetivo

de los robots de contorno. Nótese que únicamente estos robots tendrán una posición objetivo, puesto que los robots internos se limitarán a moverse por el espacio libre dejado por los robots de contorno hasta que se equilibren las fuerzas repulsivas de su campo de potencial artificial.

Para los experimentos realizados y expuestos en el Capítulo 6, se ha optado por emplear un polígono regular de 4 lados inscrito en una elipse.

### 5.1.2. Diferenciación de tareas

Una vez resuelto esto, es necesario hacer la diferenciación de tareas que van a tener lugar paralelamente durante la formación: la periferia del *swarm* y la cobertura del espacio interior. Esto se determina en función de la posición inicial del robot: un robot en cuyo rango de vecindad $\theta$ mayor a $120^{\text{o}}$ no tiene ningún vecino se considera robot de periferia y, en caso contrario, robot interno. Para detectar la presencia de robots vecinos se emplean los sensores de percepción de alcance local (*range and bearing*).

### 5.1.3. Adquisición de coordenadas iniciales

El *swarm* funciona de modo completamente descentralizado, por lo que cada robot únicamente cuenta con la información local proporcionada por su sensor de percepción (por ejemplo, si tiene algún vecino y en caso afirmativo, a qué distancia y orientación está). A tenor de ello, es necesario establecer un sistema de coordenadas relativo, siguiendo a grandes rasgos el siguiente procedimiento:

1. Definición del origen del sistema (por medio de robot semilla A).
2. Definición del eje X del sistema (con A y eligiendo un segundo robot B).
3. Definición del eje Y del sistema (con A y eligiendo un tercer robot C).
4. Una vez establecido el sistema, se calculan las coordenadas relativas del resto de robots del swarm valiéndose de *trilateral positioning.*

Para completar la primera tarea, se emplea un método de propagación de gradiente que tiende a seleccionar un robot cerca del centro del *swarm* (A), elegido como origen del sistema de coordenadas relativo. Este método consiste en propagar el gradiente partiendo de los robots de la periferia, y transferirlo capa a capa hasta que quede un único robot sin propagar, que será A.

Una vez definido el origen del sistema de coordenadas, se procede a establecer el sistema con los ejes coordinados correspondientes. Estos se determinan mediante dos robots vecinos aleatorios (B y C).

Primero se escoge el robot B, que junto con A definirá el eje X del sistema. Sus coordenadas relativas son *($d_{AB}$, 0)*. Seguidamente se hace lo propio con el robot C (define el eje Y con A), que determina sus coordenadas a través del siguiente sistema:

$$x_C^2 + y_C^2 = d_{AC}^2 \tag{5.1}$$

$$(x_C - d_{AB})^2 + y_C^2 = d_{BC}^2 \tag{5.2}$$

Obsérvese que $d_{AC}$ es la distancia entre el robot semilla A y el C, que $d_{BC}$ es la distancia entre los robots B y C y que *($x_C$, $y_C$)* son las coordenadas relativas al nuevo sistema que se pretende hallar. En última instancia, cabe recalcar que el sistema proporciona dos soluciones, de las cuales se toma la solución con un valor positivo de $y_C$, que se traduce en la dirección positiva del eje Y del sistema de coordenadas.

Con ello, se concluye el establecimiento del citado sistema y se procede al cálculo de las posiciones del resto de robots del *swarm*, mediante posicionamiento trilateral.

Dada la situación de posicionar un robot cualquiera del *swarm*, es necesario que esté rodeado de tres vecinos que sí conozcan sus coordenadas en el sistema definido, siendo estas *($x_a$, $y_a$)*, *($x_b$, $y_b$)* y *($x_c$, $y_c$)*. Valiéndose del sensor de percepción local el robot obtiene las distancias respectivas a sus tres vecinos, a saber, $d_a$, $d_b$ y $d_c$.

Se tiene pues la siguiente relación:

$$(x - x_a)^2 + (y - y_a)^2 = d_a^2 \tag{5.3}$$

$$(x - x_b)^2 + (y - y_b)^2 = d_b^2 \tag{5.4}$$

$$(x - x_c)^2 + (y - y_c)^2 = d_c^2 \tag{5.5}$$

Así, las coordenadas de un robot cualquiera del swarm se calcularán según sigue:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2(x_a - x_c) & 2(y_a - y_c) \\ 2(x_b - x_c) & 2(y_b - y_c) \end{bmatrix}^{-1} * \begin{bmatrix} x_a^2 - x_c^2 + y_a^2 - y_c^2 + d_c^2 - d_a^2 \\ x_b^2 - x_c^2 + y_b^2 - y_c^2 + d_c^2 - d_b^2 \end{bmatrix} \tag{5.6}$$

Siendo los tres puntos *($x_a$, $y_a$)*, *($x_b$, $y_b$)* y *($x_c$, $y_c$)* no colineales.

## 5.1.4. Movimiento del *swarm*

### 5.1.4.1. Introducción al método del campo de potencial artificial

Se propone el método del *Artificial Potential Field* como técnica de navegación para distribuir a los robots, método bastante popular y cuyo fundamento es que el robot se mueva, tratando de alcanzar el máximo gradiente en un campo de potencial continuo

en el espacio. Dependiendo del control implementado, las técnicas se pueden subdividir en dos grupos:

- Campos de potencial para *path planning*, como planeador de alto nivel con previo conocimiento de la función del potencial.
- Campos de potencial para *reactive navigation*, cuyo objetivo es computar una respuesta en función de los alrededores del robot. En este caso, el algoritmo propuesto en Yang et al. [14] y que se va a proceder a implementar pertenece a esta segunda categoría.

En líneas generales, el funcionamiento esperado consiste en que el *Artificial potencial field* atraiga al robot a la meta y lo aleje de cualquier obstáculo, en este caso, el resto de robots del *swarm*.

Así, el cómputo total del campo de potencial artificial comporta la suma de dos términos: el potencial atractivo y el repulsivo como se indica en 5.7:

$$U_t(\mathbf{x}) = U_a(\mathbf{x}) + U_r(\mathbf{x}) \tag{5.7}$$

De manera gráfica, en la Figura 5.1 se ha representado un ejemplo de la distribución de fuerzas característica de este método. Por otro lado, en la Figura 5.2 obtenida de [22], se esquematiza la distribución global de fuerzas, que, recoge y ejemplifica de forma visual lo expuesto anteriormente.

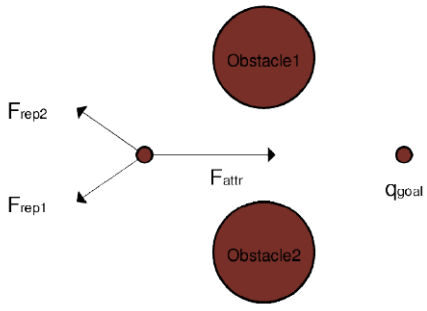



Figura 5.1: Distribución de fuerzas con dos obstáculos y una meta utilizando APF

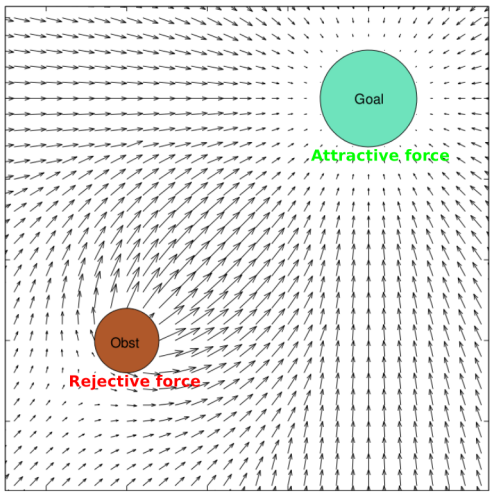



Figura 5.2: Esquema del funcionamiento de APF

Ha de ponerse en consideración que el potencial atractivo debe ser diseñado de forma que todos los puntos tengan mayor valor que el punto objetivo, mientras que el potencial repulsivo será mayor en zonas cercanas a obstáculos. De igual manera,

la fuerza resultante originada por el campo de potencial se subdivide también en las mencionadas componentes. Dicha fuerza se calcula por medio del descenso de gradiente, materializado como el gradiente negativo que se indica en 5.8 y guiará al robot hacia la meta.

$$F_t(\mathbf{x}) = -\nabla U_t(\mathbf{x}) = -\nabla U_a(\mathbf{x}) - \nabla U_r(\mathbf{x}) = F_a(\mathbf{x}) + F_r(\mathbf{x}) \tag{5.8}$$

Partiendo de la fuerza resultante, (ver 5.8) se calculará consecuentemente la velocidad del robot.

#### 5.1.4.2. Control del movimiento del *swarm* basado en el método del campo de potencial artificial

Como se ha comentado en la Sección anterior, este método consiste fundamentalmente en calcular la fuerza resultante de cada robot, teniendo en cuenta la existencia de obstáculos, que generan una fuerza repulsiva, así como la existencia de una posición de meta (para los robots de contorno), que genera una fuerza atractiva. De esta manera, la fuerza resultante (el sumatorio de ambas fuerzas), se emplea para determinar la dirección y velocidad a la que debe moverse cada uno de los robots.

Siguiendo el algoritmo propuesto en Yang et al. [14], el objetivo es que los robots en la periferia, que constituyen los límites del *swarm*, encuentren el camino más corto para adoptar la forma deseada, evitando en el proceso colisiones con el resto de robots (mediante fuerzas de repulsión), mientras se mantiene el rango de comunicación con el *swarm*. Cabe destacar, que la cobertura colaborativa de los robots internos ocurre en paralelo con la disposición de los robots de periferia.

De forma similar al despliegue de los límites del *swarm*, se realiza la cobertura de la zona interna por medio del método de potencial artificial, en el que los robots vecinos emplearán fuerzas repulsivas para prevenir colisiones entre los robots del *swarm*. En el momento que todos los robots internos tengan sus fuerzas equilibradas, se podrá afirmar que la formación del *swarm* se ha completado con éxito. Para evitar el choque entre los robots de la periferia y los internos, que están situados inmediatamente al lado de ellos, se genera una fuerza repulsiva por parte de los robots de periferia, de forma que los robots internos no se muevan más allá de la zona interna.

#### 5.1.4.3. Implementación

En primer lugar, se diseña el campo de potencial para extrapolar el comportamiento deseado de cada uno de los robots y acto seguido se procede a traducir esta información al formato del *input* que recibirán los robots del *swarm*, esto es, en las velocidades lineal y angular que regirán el movimiento de dicho robot hacia la meta.

Se va a proceder a describir la implementación propuesta en Yang et al. [14] según sigue. Para proceder al diseño del campo de potencial atractivo, se va a emplear una función proporcional a la distancia de la meta, $\rho_{goal}$, siendo el cuadrado de la mencionada distancia euclídea, como se puede apreciar en 5.9.

$$U_a(\mathbf{x}) = \frac{1}{2}k_a\rho_{goal}^2(\mathbf{x}) = \frac{1}{2}k_a||\mathbf{x} - \mathbf{x}_{goal}||^2 \tag{5.9}$$

Del mismo modo, el cálculo de la fuerza atractiva se lleva a cabo por medio del gradiente según 5.10.

$$F_a(\mathbf{x}) = -\nabla U_a(x) = -k_a(\mathbf{x} - \mathbf{x}_{goal}) \tag{5.10}$$

En el caso del potencial repulsivo, se emplea la función 5.11 para evitar que el robot colisione con algún obstáculo.

$$U_r(\mathbf{x}) = \begin{cases} \frac{1}{2}k_r\left(\frac{1}{\rho_{obs}(\mathbf{x})} - \frac{1}{\rho_o}\right)^2 & if \quad \rho_{obs}(\mathbf{x}) \leq \rho_0 \\ \\ 0 & if \quad \rho_{obs}(\mathbf{x}) > \rho_0 \end{cases} \tag{5.11}$$

Siendo $\rho_0$ la distancia a partir de la cual el robot debe percibir los obstáculos y $\rho_{obs}$ la distancia a un obstáculo. De la misma forma, se calcula la fuerza repulsiva para todos los puntos del rango de percepción del robot, denominado $\Theta$ y siendo $\theta$ la componente angular del movimiento proporcionada por los sensores de percepción.

$$F_r(x) = -\int_{\Theta} \nabla_x U_r \, \partial\theta \tag{5.12}$$

Discretizando 5.12, se procederá a implementar:

$$F_r(x) = k_r \sum_i f(\theta_i) \text{ with } i = 0, 1, 2, ... \tag{5.13}$$

donde,

$$f(\theta_i) = \begin{cases} \left(\frac{1}{\rho^3_{obs}(\theta_i)} - \frac{1}{\rho_o\, \rho^2_{obs}(\theta_i)}\right)(\mathbf{x} - \mathbf{x}_{goal}) \quad if \\ \\ \quad \rho_{obs}(\theta_i) \leq \rho_0 \\ \\ 0\ otherwise. \end{cases} \tag{5.14}$$

#### 5.1.4.4. Problemática inherente al APF y mejoras propuestas

Téngase en cuenta, no obstante, que el comportamiento de los campos de potencial diseñados dista de ser ideal. El primer problema inherente a esta técnica es la existencia de mínimos locales, en los que el robot puede quedarse atrapado por ser nulo el gradiente. Además, la fuerza calculada puede resultar en una reacción desmesurada a los obstáculos, provocando la desviación demasiado pronunciada del camino 'ideal' hasta la meta. Ello puede devenir en accidentes con otros obstáculos. En general, suele resultar un trabajo delicado ajustar las fuerzas para que la trayectoria hacia la posición objetivo sea suave y ocurra sin complicaciones.

Según se discute ampliamente en [23] y [24], el comportamiento errático propiciado por los problemas comentados puede mejorarse en cierto grado, (que no solucionarse por completo) tratando de afinar los parámetros de las expresiones que se han indicado anteriormente, a saber, $\rho_0$, $k_a$ y $k_r$. Una posible solución para la sobrerreacción ante la presencia de obstáculos es añadir un término lineal ($f_2$) a la fuerza de repulsión 5.13 (véase 5.15), cuyo objetivo es que la modificación de la trayectoria ideal sea más suave al toparse con un obstáculo.

$$F_r(x) = k_r \sum_i f_1(\theta_i) + \sin^2(\alpha) f_2(\theta_i) \text{ with } i = 0, 1, 2, ... \tag{5.15}$$

siendo $f_2(\theta_i)$,

$$f_2(\theta_i) = \begin{cases} (\mathbf{x} - \mathbf{x}_{goal}) \qquad if \quad \rho_{obs}(\theta_i) \leq \rho_0 \\ \\ 0\ otherwise. \end{cases} \tag{5.16}$$

Obsérvese que, siguiendo la lógica, cuando un obstáculo está por detrás de la dirección en la que se está moviendo el robot, la probabilidad de impacto es baja, mientras que cuando lo está por delante, el riesgo de colisión se torna alto, ergo la fuerza repulsiva lo será también. Dicha percepción se puede materializar matemáticamente en una implementación 'dinámica' de $\rho_0$ (ver 5.17), adaptada a la posición angular del obstáculo más cercano (denominada $\phi$), de acuerdo con lo definido en [25].

$$\rho_0(\phi) = \rho_m \frac{1 + \mathrm{e}^{\cos(\phi)}}{1 + \mathrm{e}} \tag{5.17}$$

Nótese que $\rho_m$ es una constante que limita el máximo valor posible de $\rho_0$. Tal y como se ha aludido anteriormente, $\rho_0$ tomará su máximo valor cuando el robot esté orientado hacia el obstáculo más cercano.

Al respecto del problema del mínimo local, cabe mencionar que aparece cuando el punto objetivo está cerca de un obstáculo. Dado el caso, la fuerza atractiva está definida en dirección a la meta y la fuerza repulsiva originada por el obstáculo descrita en la dirección contraria. Se propone pues, reducir en consecuencia el valor de la constante repulsiva, $k_r$, cuando el robot se vaya acercando a la meta. De este modo, el robot puede ignorar los obstáculos lo suficiente como para minimizar la fuerza repulsiva, visto que la meta estaría más cerca que el obstáculo en este escenario.

#### 5.1.4.5. Obtención del input de los robots: cálculo de las velocidades lineal y angular

Primera opción (más básica, con problemas): hacer la velocidad lineal y angular proporcionales al módulo y la componente angular de la fuerza resultante.

$$v = k_v \cdot |F_t|;\ k_v = \frac{v_{max}}{|F_t|_{max}} \tag{5.18}$$

$$\omega = k_w \cdot F_\phi;\ k_w = \frac{\omega_{max}}{F_{\phi,max}} \tag{5.19}$$

Problemas como resultado: en caso de que el obstáculo esté justo delante del robot, puesto que la fuerza resultante tiene un ángulo mayor a $90^{\text{o}}$, de forma que las velocidades lineal y angular toman un valor excesivamente alto.

Para solventar esto, se valora que la velocidad lineal sea proporcional a la componente x de la fuerza resultante, de forma que su valor sea nulo cuando dicha componente sea negativa.

$$v = k_v \cdot \max(0, F_{t,x});\ k_v = \frac{v_{max}}{F_{t,x,max}} \tag{5.20}$$

## 5.2. Alternativa al APF para el despliegue: *centroidal Voronoi tessellations*

Como alternativa al método del APF (*Artificial Potential Field*) para el despliegue del *swarm*, en vista de que su empleo dista de ser ideal y puede acarrear algunos problemas detallados pertinentemente en la Sección 5.1.4.4, se propone emplear un

método alternativo: la cobertura del espacio ocupado por la figura que se desea obtener por medio de *centroidal Voronoi tessellations*. A grandes rasgos, el espacio de la formación se distribuye en tantas regiones de Voronoi como robots se tienen en el *swarm*, y la posición objetivo de cada uno de ellos coincide con el centroide de dichas regiones de Voronoi.

El algoritmo que se ha puesto en práctica es un algoritmo de Lloyd con implementación distribuida, inspirado en la propuesta de Cortés et al. [21], que presenta sus ventajas e inconvenientes frente al método del *Artificial Potential Field*.

Se comenta seguidamente la motivación que ha suscitado el empleo de este algoritmo, así como las particularidades de su implementación.

### 5.2.1. Ventajas respecto al APF

La primera de ellas es que no es necesaria la diferenciación de tareas, por lo que se adhiere óptimamente a la filosofía de la robótica de *swarm* de máxima homogeneidad entre los roles de los robots que conforman el *swarm*, favoreciendo así la descentralización de su operación.

Así, se sugiere emplear únicamente la cobertura con celdas de Voronoi de la región a rellenar. Como consideración adicional, se añadirá una condición al algoritmo de Voronoi detallado en 5.2.3 de modo que no se estipulará una región Q estática de cobertura. En lugar de ello, se ofrece la posibilidad de que, en el momento en que el centroide de la celda de Voronoi asociada a alguno de los robots se salga del contorno de la figura deseada, dicho robot se detenga. De este modo, el único requisito a cumplir es que en la situación inicial los robots se hallen dentro del contorno de la formación a alcanzar, y se logrará que dicho contorno quede bien definido en la figura final.

La siguiente ventaja significativa que ofrece este algoritmo es su estabilidad, puesto que una vez se estipula la *target position* de un robot, que es el centroide de su celda de Voronoi correspondiente, se elimina la posibilidad de colisionar con otros robots en el movimiento. Puede ocurrir que el algoritmo de Voronoi sea más lento en converger que el descrito para el APF, pero de forma genérica las primeras iteraciones suelen ser bastante rápidas.

### 5.2.2. Fundamentos de los diagramas de Voronoi

Los diagramas de Voronoi (también conocidos como teselaciones del plano o *centroidal Voronoi tessellations*) son particiones del plano de uso fundamental en geometría computacional, que aportan información de utilidad sobre la proximidad entre puntos (o robots como en el caso que nos ocupa). Véase la Figura 5.3 para hacerse una idea general del aspecto típico que presentan los diagramas de Voronoi.

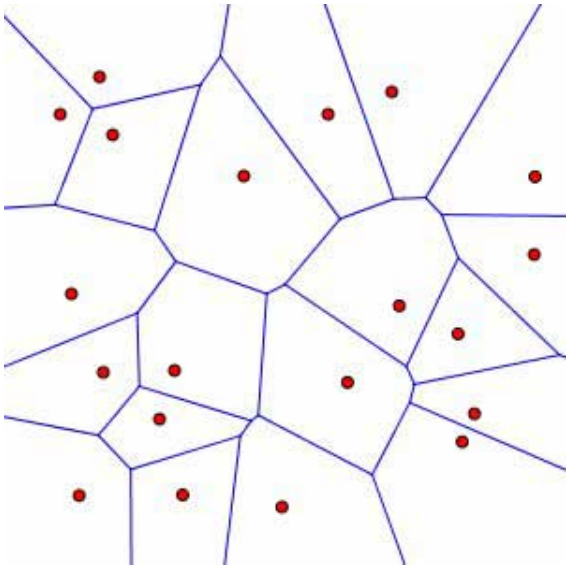


Figura 5.3: Diagrama de Voronoi, imagen tomada de la *American Mathematical Society*.

Acorde con Cortés et al. [21], la partición óptima de Q está formada por las regiones de Voronoi ($V(P) = \{V_1, ..., V_n\}$) generadas por los puntos $\{p_1, ..., p_n\}$, de tal forma que:

$$V_i = \{q \in Q| \quad ||q - p_i|| \leq ||q - p_j|| \quad \forall j \neq i\} \tag{5.21}$$

Los puntos $\{p_1, ..., p_n\}$ se conocen como generadores. En el caso de este Trabajo de Fin de Grado, los generadores representan la posición de los robots que conforman el *swarm*.

Las líneas que componen el diagrama son las bisectrices entre los puntos $\{p_1, ..., p_n\}$, generando semiplanos para hallar el diagrama. Dichas líneas se denominan fronteras, y por definición, los puntos que las componen son equidistantes a los dos puntos generadores vecinos. Así, cuando dos regiones de Voronoi ($V_i$ y $V_j$) son adyacentes $p_i$ es considerado vecino de $p_j$ y viceversa.

De acuerdo con la notación de la expresión 5.21, Q representa la región del espacio a cubrir con el diagrama de Voronoi. Conforme a Cortés et al. [21], dado que Q es un poliedro convexo en un espacio euclídeo finito, el contorno de cada región de Voronoi

($V_i$) es un polígono convexo.

### 5.2.3. Computación del diagrama de Voronoi: implementación distribuida

La forma de proceder se ha incluido en los algoritmos 2 y 3 inspirados en la propuesta distribuida de Cortés et al. [21].

En esta sección de describe la implementación del algoritmo de Lloyd, en su versión distribuida. De este modo, el algoritmo se ejecuta en un grupo de agentes (robots en este caso) que llevan a cabo la misma secuencia de instrucciones y comparten información (de posición y orientación) de una manera predeterminada. Por otro lado, el hecho de que se ejecute el algoritmo de cobertura de forma distribuida, también implica que se proporciona un mecanismo de *feedback* en tanto que se permite la adaptación de la red a cambios en el número de nodos debido a fallos, llegadas o partidas de los agentes. Además, otra característica de máxima significancia de los algoritmos distribuidos para grupos o redes de agentes móviles, es que pueden ser considerados como reglas de interacción locales, del todo conveniente para el caso que nos ocupa.

Se incluye un ejemplo del funcionamiento del algoritmo 2, cuya representación gráfica se puede observar en la Figura 5.4. Las características de los parámetros son las que siguen: un robot (estático) situado en el origen de coordenadas *(0, 0)* con dos vecinos (también estáticos), posicionados en *(2,1)* y *(-2.4, -2)*. En este caso ejemplo se ha empleado un radio de percepción de 2 para el robot.

Aludiendo a la Figura 5.4, los datos de interés se recogen en la intersección entre el radio de percepción del robot y los polígonos computados para sus vecinos según 2. Como se puede percibir de forma intuitiva, se trata de las bisectrices que generan los semiplanos de los que se compone el diagrama de Voronoi, denominadas fronteras y cuyos puntos son equidistantes al robot y a sus respectivos vecinos.

Nótese que en este ejemplo (sin valor experimental, pero añadido para clarificar la operación del algoritmo 2) se han añadido dos vecinos por simplicidad, pero típicamente los robots del *swarm* suelen estar completamente rodeados por vecinos, dando lugar a las celdas de Voronoi representadas en el Capítulo 6.

Ha de tenerse en cuenta que los robots estarán en movimiento hacia su meta (el centroide de su respectiva celda de Voronoi), de modo que el algoritmo se irá actualizando periódicamente hasta que los robots hayan concluido su movimiento y

**Algorithm 2** Computación de la celda propia de Voronoi (Soul del robot)

**Require:** (i) coordenadas del robot, (ii) lista de los vecinos del robot, (iii) radio de percepción del robot

**For each vecino** $n$ **de cada robot:**

1: establecimiento del radio de percepción del robot (W)
2: computación de un polígono con puntos equidistantes de n (q)
3: cálculo de la bisectriz entre n y el robot (W-q)
4: **return** puntos del exterior del polígono W-q

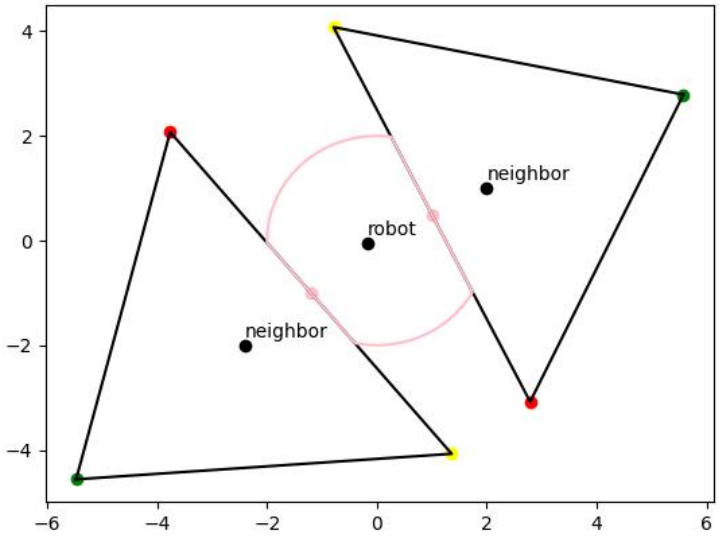



Figura 5.4: Computación de las fronteras de la región de Voronoi de un robot con dos vecinos.

la región de la figura esté cubierta por el swarm de manera uniforme.

A continuación, se muestra el algoritmo de Voronoi propuesto [3] que toma como referencia el algoritmo detallado en el artículo [21].

**Algorithm 3** Versión distribuida del algoritmo de LLoyd

**Require:** (i) computación de la celda propia de Voronoi, (ii) computación del centroide

1: **for** each individuo $i$ **do**
2: determinar la celda de Voronoi $V_i$ del robot
3: determinar el centroide $Cv_i$
4: establecimiento de $u_i$ (dinámica del robot)

### 5.2.4. Otros algoritmos

En la literatura dedicada a la geometría computacional y a los diagramas de Voronoi, se presentan algunos algoritmos distintos al que finalmente se ha

implementado. No se ha empleado ninguno de ellos puesto que no se pueden emplear de forma distribuida como era necesario en este caso, sino que son necesariamente centralizados. Típicamente se emplean en otro tipo de aplicaciones (como gráficos y asignación de recursos). Con el fin de aportar una noción general de los más utilizados y sus características más relevantes.

- **Divide y vencerás (*divide-and-conquer)*:** tal y como se especifica en [26], este tipo de algoritmos se fundamenta en la idea de dividir progresivamente un problema complejo, hasta obtener simples subproblemas que puedan ser resueltos independientemente. Este método se caracteriza principalmente por tener una alta eficiencia y un coste computacional razonable, a costa de una gran complejidad a nivel de desarrollo.

- **Intersección de semiplanos:** este método, cuya definición y base matemática se describen en la sección 4.2 del libro [27], se basa en la idea de la construcción de regiones de Voronoi mediante la intersección de $n - 1$ semiplanos (siendo $n$ el número total de semiplanos). La principal desventaja es el alto coste computacional, dado que depende de manera cuadrática del número del número de semiplanos computados.

- **Algoritmo incremental:** a pesar de su elevado coste computacional y baja eficiencia, los algoritmos incrementales son ampliamente utilizados en el estado del arte. Su popularidad se basa en la simplicidad a nivel de desarrollo para construir el diagrama de Voronoi. La base matemática de este tipo de métodos se puede encontrar en los capítulos 4.3 y 6.2 del libro [27]. En el caso de la utilización de este método para dibujar el diagrama de Voronoi, como el que se describe en [28], el método utilizado consiste en añadir puntos mientras se modifica el cierre convexo (dependiendo de si el punto siguiente se encuentra o no dentro de la región).

- **Algoritmo de Fortune:** este algoritmo para construir el diagrama de Voronoi es, con diferencia, el más repetido en la literatura consultada. Tal y como se detalla en [29], fue desarrollado por Steve Fortune en 1985, proporcionando una mayor eficiencia que los clásicos algoritmos incrementales (descritos anteriormente). Este nuevo método, supuso un gran impulso para el estado del arte, dado que logró aunar las ventajas de los algoritmos incrementales con las de *Divide y vencerás*: un desarrollo simple con un coste computacional razonable.

# Capítulo 6

# Verificación en simulación y en entorno real

Se han puesto a prueba los algoritmos descritos en el Capítulo 5, adicionando las particularidades de la implementación para adoptar una filosofía descentralizada al operar con el Robotarium vista en el Capítulo 3, y se exponen seguidamente los resultados obtenidos de forma simulada y experimental.

Para acceder a la lista de reproducción con los vídeos de todos los experimentos, seleccione en el siguiente vínculo: **vídeos**.

## 6.1. Particularidades del *hardware* del Robotarium a considerar

Téngase en cuenta que, según se ha comentado en el Capítulo 2, en el Robotarium se implementan barreras que evitan las colisiones entre los robots físicos al ejecutar los experimentos enviados. Los datos concretos proporcionados por el Robotarium son los siguientes: el diámetro de los robots es de alrededor de 11 cm de diámetro, no obstante, se recomienda que los robots estén separados unos 15 cm. Ello conlleva que, aunque los robots no vayan a colisionar debido a las características del algoritmo implementado, si están muy cerca, como ocurre en algunos robots observados en la mitad inferior del rombo de la simulación en la Figura 6.1, al ejecutar el algoritmo en la plataforma física del Robotarium no permanecerán quietos como debieran. Es por eso que, pese a alcanzar el centroide de su respectiva celda de Voronoi, en el experimento con robots reales, estos se siguen moviendo para permanecer fuera del rango de las barreras de los vecinos. Este efecto, como es de esperar, se manifiesta tanto en los experimentos desarrollados con el algoritmo de Voronoi, como en los que se emplea el APF. También ha de tenerse cuidado con las posiciones iniciales asignadas por esto mismo. Si se asignan manualmente, puede darse el caso de que

nunca llegue a ejecutarse el algoritmo porque los robots estén demasiado cerca y las posiciones de partida nunca puedan llegar a alcanzarse. Así pues, se recomienda emplear en la medida de lo posible la función del Robotarium creada a propósito de generar coordenadas aleatorias con la suficiente separación como para no tener este problema (ver la Sección A.2.3.3 del Anexo A para más detalles).

Esta situación respecto al *hardware* también afecta a la condición de terminación de los algoritmos. En simulación se ha estipulado de tal forma que cuando los robots se detienen, lo que implica que ya han alcanzado su posición objetivo, termine la simulación. Teniendo en mente lo explicado sobre las barreras de los individuos que componen el *swarm*, los robots no llegan a detenerse cuando se hace el despliegue físico, por lo que se ha optado por modificar la condición de modo que el experimento se concluya tras un número determinado de iteraciones. Nuevamente, esta circunstancia se aplica tanto al algoritmo de Voronoi como al del APF.

Por último, cabe discutir el número de robots elegidos para la validación de los algoritmos. Los experimentos se han realizado con *swarms* de 16 robots, en vista de que 20 es el número máximo de robots que puede desplegar el Robotarium simultáneamente. Debido a las circunstancias de las barreras de los robots comentadas, no pueden estar excesivamente juntos en la disposición de la figura, por lo que se ha determinado que 16 robots ofrecen un buen equilibrio entre añadir el máximo número de robots posibles y que no haya disonancias muy significativas entre los resultados en simulación y en la realidad.

## 6.2. Resultados obtenidos con la cobertura de Voronoi

Se procede a comentar la validación realizada para la implementación del algoritmo de Lloyd en su versión distribuida, explicado en la Sección 5.2.3.

### 6.2.1. Experimento 3

En primer lugar, se ha implementado el algoritmo para cubrir uniformemente la región (Q) de la forma elegida, en este caso un rombo centrado en el origen. Se ha ejecutado con un *swarm* de 16 robots, y se ha obtenido el resultado simulado que se presenta en la Figura 6.1. Según se expone en la imagen, la región Q queda perfectamente cubierta por las celdas de Voronoi de cada uno de los robots del *swarm* y estos están posicionados en el centroide de sus respectivas celdas.

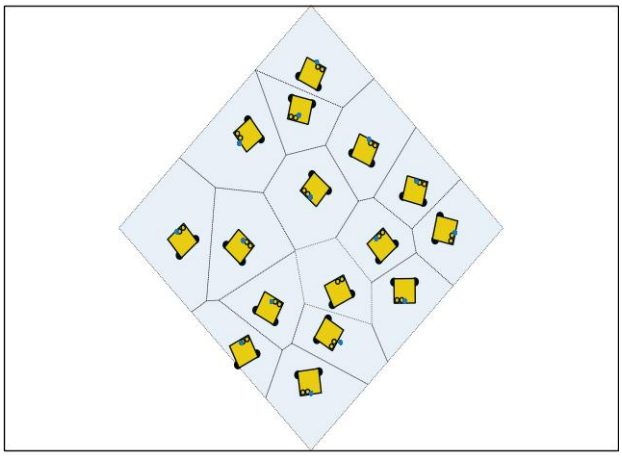


Figura 6.1: Experimento 3: computación de la cobertura con celdas de Voronoi, región Q definida.

### 6.2.2. Experimento 4

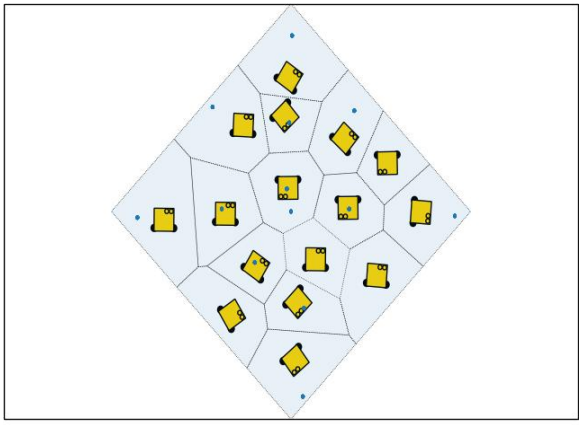
Obsérvese que en el Experimento 6.2.1 la distribución del *swarm* no conforma el contorno del polígono deseado sino que lo rellena de forma uniforme, por lo que se ha hecho una pequeña (pero significativa) modificación del algoritmo para lograr dicho contorno deseado.

Así pues, en lugar de definir la región Q (el polígono a formar) en el algoritmo como se ha hecho en el Experimento 6.2.1, no se especifica ninguna región Q en el algoritmo. En su lugar, se deja que los robots se muevan hacia el centroide de su celda de Voronoi libremente. En el momento en que el centroide se dispone a salir de la región de la forma, se detiene el avance del centroide y este permanece estático en el borde de la forma hasta que el robot lo alcanza, según se puede vislumbrar en el segundo y tercer fotogramas de la Figura 6.2.

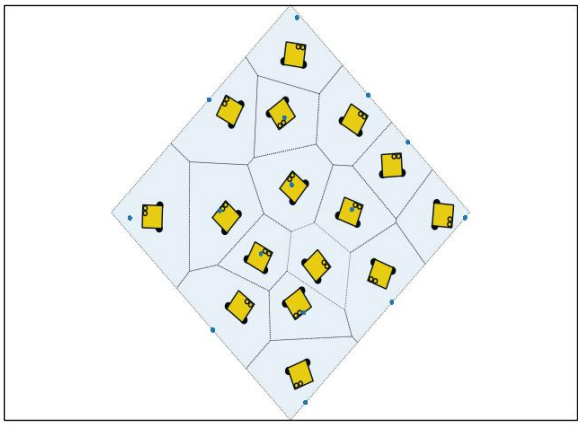
En principio, este método parece funcionar de manera bastante satisfactoria. No lleva excesivo tiempo que los robots alcancen su centroide respectivo (de hecho es más rápido que el APF, según la Tabla 6.1) y que se estabilicen las regiones de Voronoi, y además se garantiza que no haya colisiones entre los robots.

| Tiempos de ejecución | | |
|---|---|---|
| Experimento | Nº de robots | Tiempo de despliegue (s) |
| 3 | 16 | 10 |
| 4 | 16 | 22 |
| 5 | 16 | 34 |

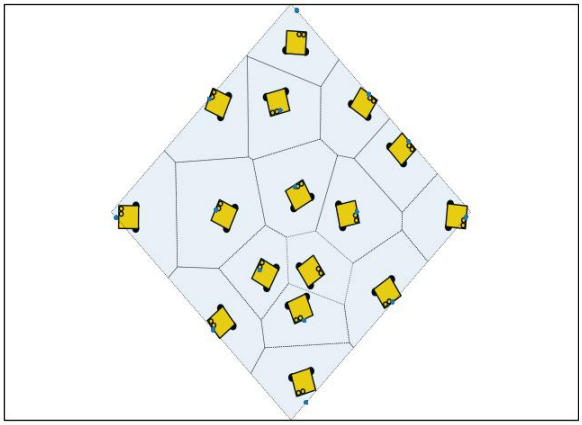
Tabla 6.1: Resultados obtenidos en simulación y despliegue con Robotarium.



(a) Posiciones iniciales.



(b) En movimiento.



(c) En destino.

Figura 6.2: Experimento 4: cobertura de teselaciones de Voronoi en un *swarm* de 16 robots, región Q no definida.

## 6.3. Resultados obtenidos con el método APF (*Artificial potential field*)

Como era de esperar a raíz de las particularidades del algoritmo implementado (descritas prolíficamente en la Sección 5.1.4.3), ha sido necesario realizar un ajuste de las constantes involucradas en el cálculo de las fuerzas atractivas y repulsivas como se comenta a continuación.

### 6.3.1. Experimento 5

Como se ha introducido brevemente, la parte más crítica de la implementación de este algoritmo ha sido el ajuste experimental de los parámetros típicos de las fuerzas y las constantes de velocidad listadas a continucación: $\rho_0$ (rango de percepción de los obstáculos, el resto de robots del swarm), $k_a$ (constante atractiva de la fuerza correspondiente), $k_r$ (constante repulsiva de la fuerza asociada), $k_v$ (constante asociada a la velocidad lineal) y $k_{om}$ (constante ligada a la velocidad angular).

Ha sido necesario realizar diversas pruebas para tratar de encontrar el equilibrio óptimo entre todos estos parámetros involucrados de forma experimental.

Primero, y más importante, ha sido necesario ajustar la constante de las fuerzas repulsivas ($k_r$) generadas por los robots vecinos considerados como obstáculos. No puede ser 'demasiado baja', puesto que, dado el caso, se generan colisiones con los robots vecinos y no se evitan aquellos que se van encontrando conforme el robot se mueve hacia el contorno de la figura. Tampoco puede ser 'demasiado alta', puesto que podría producirse una sobrerreacción ante cualquier robot vecino encontrado, relativamente porque se han limitado las velocidades lineal y angular a los valores máximos definidos por el Robotarium (0.2 m/s y 3.6 rad/s, respectivamente). Los valores significativos que se han probado son los que siguen:

- $k_r$=1. Demasiado bajo, los robots colisionan entre ellos.
- $k_r$=10. También demasiado bajo, ofrece una ligera mejora respecto al anterior, pero no especialmente notable.
- $k_r$=50. Es el que se ha elegido finalmente.
- $k_r$=90. No ofrece una mejora significativa respecto al anterior.

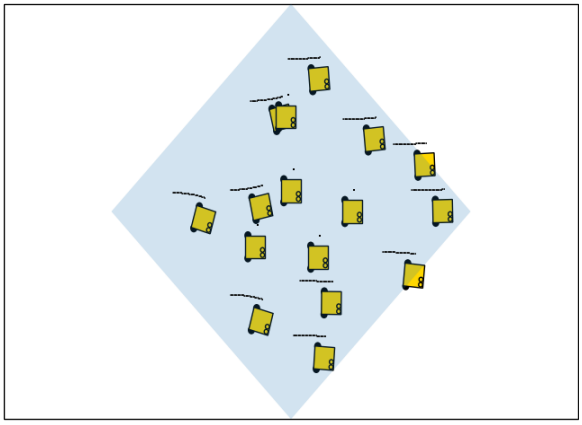
También es crítico ajustar las constantes de velocidad lineal y angular, $k_v$ y $k_{om}$ respectivamente. No sería admisible por ejemplo, que $k_v$ fuera excesivamente alta,

puesto que, pese a que la velocidad lineal está limitada a 20 cm/s, los robots avanzan en línea recta llevándose a otros por delante en el proceso y quedando apiñados en el contorno. Así, tampoco puede definirse una $k_{om}$ muy grande. El efecto es el que se podría esperar, y es que los robots empiezan a describir trayectorias circulares, colisionando con otros robots en el proceso, a pesar de que como el caso de la velocidad lineal, se ha restringido al máximo admisible del Robotarium, 3.6 rad/s. Tras varias pruebas de validación en simulación (véanse las Figuras 6.3 y 6.4), se concluyó que los valores óptimos para evitar los casos extremos descritos son $k_v$ =0.002 y $k_{om}$=0.5.

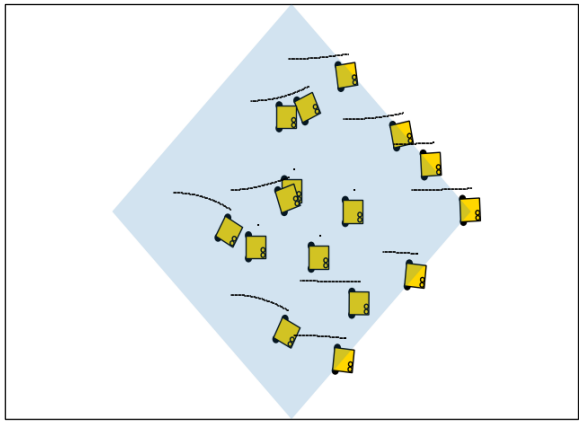
También ha sido necesario ajustar $\rho_0$ a 0.01 (consultar Figura 6.5 para el caso de $\rho_0$=1), representando la influencia de la distancia al obstáculo desde el robot. Nótese que si, como en este caso hay muchos robots agrupados y el rango de percepción es alto, un robot puede considerar (erróneamente) obstáculos a robots que realmente no están dentro del rango de vecindad, obteniendo un comportamiento errático en la trayectoria que propicie alguna colisión indeseada.

Obsérvese que en el experimento reflejado en la Figura 6.6, únicamente se mueven los robots de contorno, de acuerdo con las trayectorias representadas en color negro. Este ha sido el mejor resultado que se ha obtenido, sin tener lugar ninguna colisión. En los casos que se ha implementado el movimiento de los robots internos, las fuerzas de repulsión son demasiado fuertes, de modo que se mueven hacia la periferia, colisionando en algunos casos con los los robots de contorno. Se ha intentado reducir la constante repulsiva para evitar este efecto, pero ello generaba aún más colisiones entre los propios robots de periferia y los internos, al atenuar el efecto de repulsión a los otros robots.

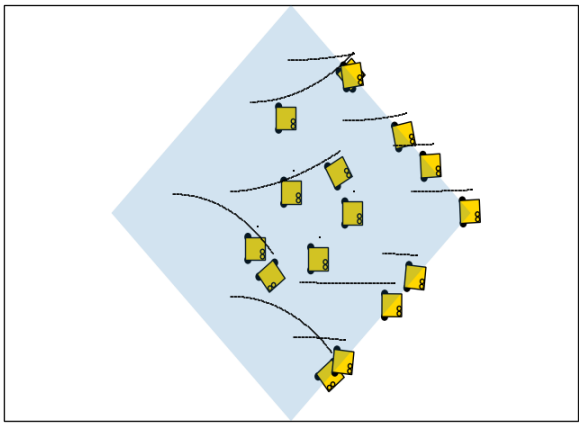
Según se ha podido constatar, ajustar todos los parámetros experimentalmente de forma que se alcance un equilibrio aceptable entre todos, resulta complicado e incluso puede ocurrir alguna colisión inesperada pese a funcionar bien a priori para un caso concreto.



(a) Posiciones iniciales.

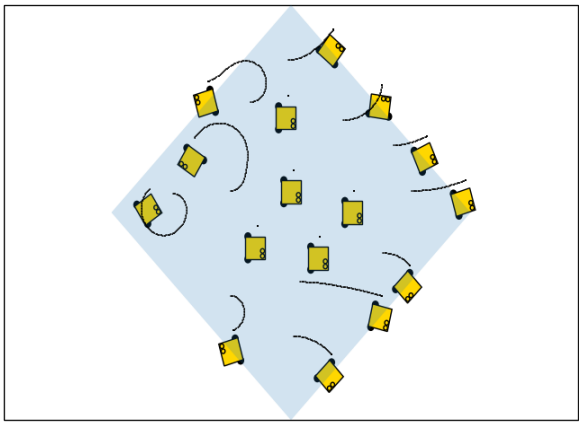


(b) En movimiento.

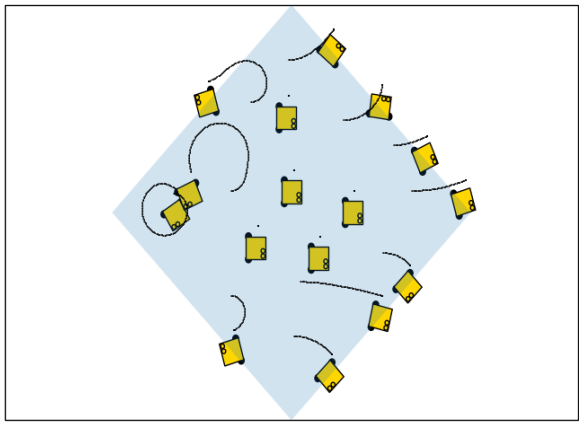


(c) En destino.

Figura 6.3: Pruebas con $k_{om}$=0.1



(a) Posiciones iniciales.



(b) En movimiento.

Figura 6.4: Pruebas con $k_{om}$=1

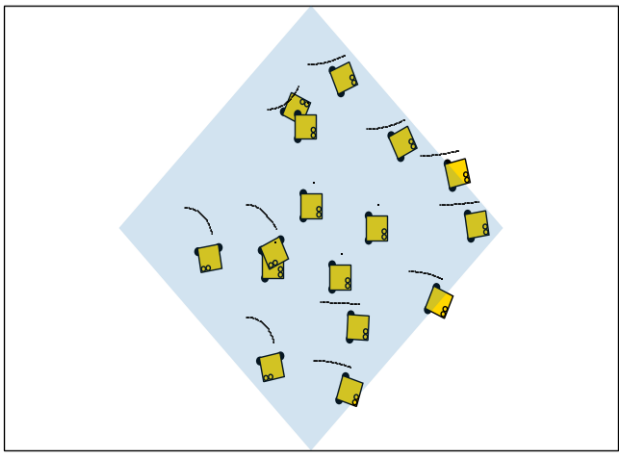
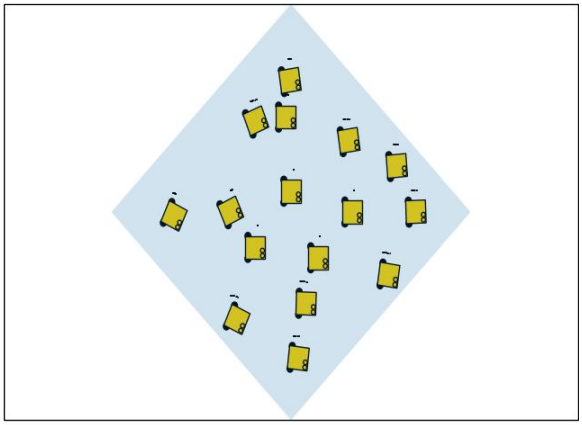
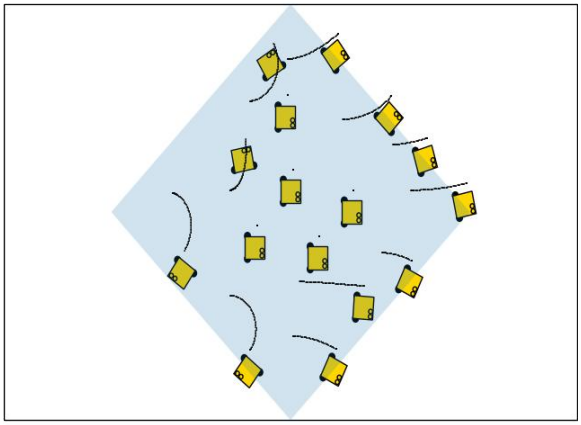


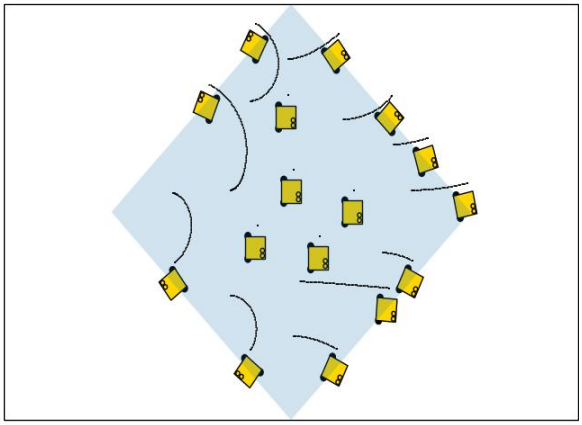
Figura 6.5: Pruebas con $\rho_0$=1.



(a) Posiciones iniciales.



(b) En movimiento.



(c) En destino.

Figura 6.6: Experimento 5: resultados con el algoritmo del APF para un *swarm* de 16 robots.

# Capítulo 7

# Conclusiones

En primera instancia, se ha constatado la idoneidad del Robotarium para trabajar con sistemas distribuidos, (pese a su articulación plenamente centralista y global) mediante la nueva gestión propuesta; se ha hecho uso de sensores simulados para crear la ilusión de percepción local y descentralizada deseada.

Según se ha detallado en el Capítulo 6, es necesario tener en cuenta, por un lado, la brecha simulación-realidad a la hora de desarrollar experimentos en el *hardware*, por otro, las limitaciones físicas de los robots reales, y por último, las particularidades de las medidas de seguridad anti colisiones del Robotarium. Poniendo en práctica todas las recomendaciones realizadas, el Robotarium se erige como una alternativa de mucho interés para trabajar con robótica de *swarm*, dado que se pueden emplear sus instalaciones para desplegar todos los experimentos que se desee a ningún coste económico. La comunicación con los desarrolladores del Robotarium es muy fluida, en caso de obtener resultados anómalos en los experimentos, siempre ha sido posible consultarles.

Al respecto de los algoritmos que se han implementado y validado, según se puede apreciar en los resultados expuestos en el Capítulo 6, se ha concluido que es claramente preferible emplear la implementación distribuida del algoritmo de Lloyd, cubriendo la región del espacio definida por la forma con teselaciones de Voronoi. Se ha mostrado que el tiempo de ejecución es menor que el empleado por el algoritmo del campo de potencial artificial, (APF) y además se ha comprobado que es mucho más estable que este último. Ello se refiere a que la implementación distribuida del algoritmo de Lloyd garantiza la ausencia de colisiones entre los robots, realizando una cobertura del espacio interno y del contorno de la figura muy aceptable. El algoritmo del APF se comporta más errática e inestablemente, y no se puede garantizar con seguridad la ausencia total de colisiones, pese a haber realizado un ajuste experimental de los

parámetros críticos, implicados en el cálculo de las fuerzas de atracción y repulsión que generan el campo de potencial. De tal forma, se puede concluir que este método, pese a ser ampliamente utilizado en aplicaciones de *path planning* y *reactive navigation* en el campo de robots autónomos, (donde exhibe un comportamiento satisfactorio) no es el método idóneo en las aplicaciones expuestas en este Trabajo de Fin de Grado.

Por último, se establecen las líneas futuras de trabajo, en relación con las conclusiones obtenidas. En primer lugar y más significativo, se posibilita la implementación y validación de otros algoritmos distribuidos en el Robotarium, (ya sea de formación, o de otras disciplinas de la robótica de *swarm*) gracias a la arquitectura que se ha implementado.

Además, se propone la opción de poner en práctica un algoritmo híbrido de los dos que se han implementado. Puesto que el despliegue de los robots de periferia con el método del campo de potencial artifical no ocasiona colisiones, se sugiere que éste se haga mediante el APF. La cobertura del espacio interior dejado por los robots de contorno, (conflictiva con el APF) se ejecutaría valiéndose del algoritmo distribuido de Lloyd, cubriendo la región con celdas de Voronoi para los robots internos.

# Capítulo 8

# Bibliografía


[1] Sahin E. Swarm robotics: from sources of inspiration to domains of application. *Swarm robotics, lecture notes in computer science, vol. 3342. Springer*, pages 10–20, 2005.

[2] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7:1–41, 03 2013.

[3] Ying Tan and Zhong yang Zheng. Research advance in swarm robotics. *Defence Technology*, 9(1):18–39, 2013.

[4] B Bassler. How bacteria talk to each other: regulation of gene expression by quorum sensing. *Current Opinion in Microbiology*, 2(6):582–587, 1999.

[5] Homepage of Seaswarm project. `http://senseable.mit.edu/seaswarm/index.html`, 22 de Junio de 2022.

[6] Gambardella Mondada Floreano Nolfi et al Pettinaro, Kwee. Swarm robotics: A different approach to service robotics. *Proceedings of the 33rd international symposium on robotics*, pages 71–6, 2002.

[7] Bender N Estana R Thiel M Wörn H Seyfried J, Szymanski M. The iswarm project: intelligent small world autonomous robots for micromanipulation. *Swarm robotics, lecture notes in computer science, vol. 3342. Springer*, pages 70–83, 2005.

[8] Olivier Michel. Webotstm: Professional mobile robot simulation. *CoRR*, abs/cs/0412052, 2004.

[9] Homepage of CoppeliaSim project. `https://coppeliarobotics.com/indexl`, 22 de Junio de 2022.

[10] Adrián Jiménez-González, Jose Ramiro Martinez-de Dios, and Anibal Ollero. Testbeds for ubiquitous robotics: A survey. *Robotics and Autonomous Systems*, 61(12):1487–1501, 2013.

[11] David Johnson, Tim Stack, Russ Fish, Daniel Flickinger, Leigh Stoller, Robert Ricci, and Jay Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. pages 1–12, 04 2006.

[12] A. Stubbs, V. Vladimerou, A.T. Fulford, D. King, J. Strick, and G.E. Dullerud. Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control. *IEEE Control Systems Magazine*, 26(3):56–69, 2006.

[13] Sean Wilson, Paul Glotfelter, Li Wang, Siddharth Mayya, Gennaro Notomista, Mark Mote, and Magnus Egerstedt. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 40(1):26–44, 2020.

[14] Xin Duan Gaopan Shen Hong-an Yang, Yuhua Li and Shaohua Zhang. A parallel shape formation method for swarm robotics. *Robotics and Autonomous Systems*, 2022.

[15] Asaf Efrima and David Peleg. Distributed algorithms for partitioning a swarm of autonomous mobile robots. *Theoretical Computer Science*, 410(14):1355–1368, 2009. Structural Information and Communication Complexity (SIROCCO 2007).

[16] Jakob Fredslund and Maja Mataric. A general algorithm for robot formations using local sensing: And minimal communication. *IEEE Transactions on Robotics*, 18:837–846, 10 2002.

[17] Yaochu Jin, Hongliang Guo, and Yan Meng. Robustness analysis and failure recovery of a bio-inspired self-organizing multi-robot system. *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 154–164, 2009.

[18] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[19] Yan Meng and Hongliang Guo. A bio-inspired developmental approach to swarm robots self-organization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3512–3517, 2012.

[20] Chien Chern Cheah, Saing Paul Hou, and Jean Jacques E. Slotine. Region-based shape control for a swarm of robots. *Automatica*, 45(10):2406–2411, 2009.

[21] Jorge Cortés, Sonia Martínez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks: variations on a theme. 01 2003.

[22] Zhongliang Zhao and Torsten Braun. Topology control and mobility strategy for uav ad-hoc networks: A survey. 2012.

[23] Maher Khatib and Raja Chatila. An extended potential field approach for mobile robot sensor-based motions. In *Proc. International Conference on Intelligent Autonomous Systems (IAS'4)*, 1995.

[24] D. Green, J. Sasiadek, and G. Vukovich. Path tracking, obstacle avoidance and position estimation by an autonomous, wheeled planetary rover. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1300–1305. IEEE, 1994.

[25] Haddad Haddad, Maher Khatib, Simon Lacroix, and Raja Chatila. Reactive navigation in outdoor environments using potential fields. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 2, pages 1232–1237. IEEE, 1998.

[26] Douglas R. Smith. The design of divide and conquer algorithms. *Science of Computer Programming*, 5:37–58, 1985.

[27] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[28] FRANZ AURENHAMMER and OTFRIED SCHWARZKOPF. A simple on-line randomized incremental algorithm for computing higher order voronoi diagrams. *International Journal of Computational Geometry & Applications*, 02(04):363–381, 1992.

[29] Reetu Gupta, Rajiv Gandhi, and P Indira. Fortune's method: An efficient method for voronoi diagram construction. 2013.

[30] Moumita Mondal, Sruti Gan Chaudhuri, Ayan Dutta, Krishnendu Mukhopadhyaya, and Punyasha Chatterjee. Uniform circle formation by oblivious swarm robots. 12 2020.

[31] Desislava Dimitrova, Marc Brogle, Torsten Braun, Geert Heijenk, and Nirvana Meratnia. Joint ercim emobility and mobisense workshop. *IEEE Journal on Selected Areas in Communications-JSAC*, pages 1651–1656, 2012.

# Lista de Figuras

# Lista de Tablas

# Anexos

# Anexos A

# The Robotarium

## A.1. Instalación de la API de la Universidad de Georgia Tech

1. Descarga de la API de Robotarium para Python.

2. Instalación de librerías de Python necesarias para emplear el simulador: NumPy, matplotlib y CVXOPT.

3. Para ejecutar uno de los scripts ejemplo, abrir la terminal e introducir lo siguiente:

```
python "path_to_simulator"/rps/examples/plotting/
barrier_certificates_with_plotting.py
```

## A.2. Librerías de interés

Al respecto del manejo general de la información relevante sobre los robots en el Robotarium, como puede ser su posición o velocidad, cabe comentar que se recoge en matrices que incluyen la información global de todos los robots. Con el fin de acceder y/o modificar esta información de forma individual, se ha incluido una descripción detallada del código a nivel intermedio desarrollado con dicho propósito en 3.1.

En la carpeta nombrada rps que figura en la API, (ver A.1) se encuentran disponibles los scripts de Python que contienen el código de bajo nivel que articula las funcionalidades del Robotarium. Se incluyen algunas nociones generales de las que se han considerado más relevantes durante el curso de desarrollo del presente proyecto.

### A.2.1. robotarium_abc.py

Se trata de la interfaz de la clase Robotarium, cuya función es gestionar la representación gráfica de las entidades que van a figurar en el entorno simulado del

Robotarium, además de asegurar que el comportamiento del simulador y los robots se corresponden adecuadamente. Define algunas constantes y parámetros relevantes que se destacan a continuación:

- El número de robots presentes en el entorno, se permite añadir 50 como máximo en entorno de simulación, no obstante, el número de GRITSBots disponible para el despliegue del hardware es más limitado.
- Las dimensiones del entorno, de anchura 3.2 y longitud 2.
- Las dimensiones de los robots: diámetro de los robots (0.11), radio de las ruedas (0.016) y la longitud de la base de los robots (0.105).
- Las velocidades, lineal (0.2) y angular (calculada a partir del radio de las ruedas, la velocidad lineal máxima y el diámetro de los robots) máximas.
- El tiempo de un step de simulación, de duración 0.033.

### A.2.2. robotarium.py

Proporciona rutinas a propósito de la interfaz del Robotarium vista en A.2.1.

- La función *get_poses()* devuelve un 3xN numpy array con la posición y orientación de los robots en el entorno del Robotarium.
- *step()* función que se emplea para actualizar la dinámica de los robots, en relación con su posición actualizada transcurrido un step cuya duración se ha indicado en A.2.1, así como con la representación gráfica del cambio de posición.

### A.2.3. utilities

En la carpeta utilities figuran scripts con herramientas para gestionar el movimiento de los robots en el entorno del Robotarium, relacionadas con la seguridad (evitar colisiones), asignar el punto de partida del movimiento de los robots y un largo etcétera. Acto seguido se comentan brevemente los aspectos más significativos:

#### A.2.3.1. Barrier certificates

Scripts a propósito de evitar las colisiones entre robots en simulación. Ver A.3.2 para más detalles sobre los resultados que proporcionan las funciones proporcionadas en los scripts *barrier_certificates.py* y *barrier_cerificates2.py*.

#### A.2.3.2. Controllers

Funciones a propósito de realizar el control de posición de los robots en movimiento, por ejemplo, para llevar un *single integrator* a una posición deseada por medio de control proporcional. Ver A.3.3 para más detalles sobre las pruebas que se han hecho con los controladores disponibles en el script *controllers.py*.

#### A.2.3.3. Miscellanea

En el script *misc.py* figuran algunas funciones de gran utilidad que se han utilizado repetidamente en el presente Trabajo de Fin de Grado:

- Generar posiciones iniciales aleatorias para todos los robots en el entorno del Robotarium: *generate_initial_conditions*
- Comprobar cuándo los robots están lo suficientemente cerca (por defecto con un error de posición de 0.05 y un error de rotación de 0.2) de la posición y orientación destino: *at_poses*

#### A.2.3.4. Transformations

En lo que concierne al mapeo de *single integrator* a dinámica *unicycle* con restricciones de la magnitud de la velocidad angular, con capacidad para que los robots puedan moverse hacia atrás y viceversa. Funcionalidades disponibles en *transformations.py*.

## A.3. Experimentos básicos en The Robotarium

Con el objetivo de familiarizarse con la API del Robotarium y con sus funcionalidades, se han realizado algunos experimentos previos en relación con el objetivo final del presente proyecto, enumerados y detallados seguidamente.

### A.3.1. Experimento 1: realizar una trayectoria

Se propone realizar la trayectoria de un cuadrado empleando un único robot. A continuación se exponen los pasos seguidos para cumplir dicho cometido. Obsérvese que se ha empleado como plantilla el script *si_go_to_point.py* y se comentan los cambios más significativos realizados:

1. Añadir el número de robots deseados al espacio del Robotarium, (1 en este caso) así como su posición inicial:

```
# Instantiate Robotarium object
N = 1 # number of robots
initial_conditions = np.array(np.mat('1;0.8;0'))
# when needed random initial locations, uncomment the
# following line instead of using the previous one
# initial_conditions=generate_initial_conditions(N)
r = robotarium.Robotarium(number_of_robots=N,
show_figure=True, initial_conditions=initial_conditions,
sim_in_real_time=False)
```

2. Definir los cuatro vértices del cuadrado. Estos constituirán los puntos meta de cada una de las trayectorias lineales que conforman los 4 lados del cuadrado.

```
goal_points_1=np.array(np.mat('1;-0.2;0')) #1st vertex
goal_points_2=np.array(np.mat('0;-0.2;0')) #2nd vertex
goal_points_3=np.array(np.mat('0;0.8;0')) #3rd vertex
goal_points_4=np.array(np.mat('1;0.8;0')) #4th vertex
```

3. Añadir un bucle *while* por cada uno de los vértices del cuadrado a propósito del movimiento del robot hacia la meta, con una condición para que el robot se detenga una vez completada la forma deseada.

```
while (np.size(at_pose(np.vstack((x_si,x[2,:])),
goal_points_1, rotation_error=100)) != N):
```

### A.3.2. Experimento 2: manejo de colisiones

Con el fin de estudiar el manejo de los obstáculos y la forma de evitarlos propuesta en el Robotarium, se ha implementado la misma trayectoria que la expuesta en A.3.1, incluyendo en este caso un obstáculo, otro robot, en medio de la misma para poder observar la respuesta obtenida.

Resultados:

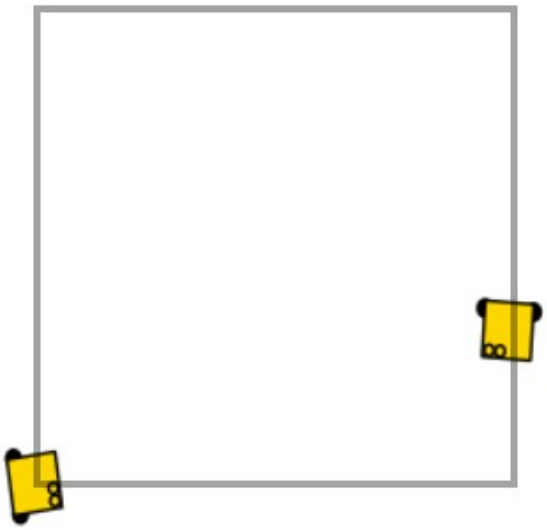


Figura A.1: Recorrido antes del encuentro de los dos robots.

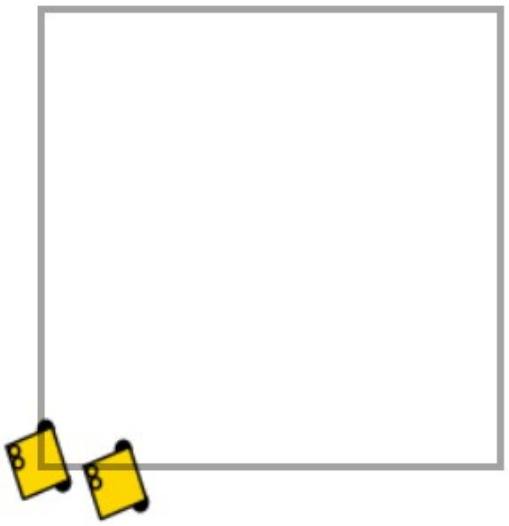


Figura A.2: Resultado final de la simulación, colisión evitada con éxito.

Nótese que se ha empleado una de las funciones proporcionadas en el script barrier_certificates.py para llevar a cabo el control de posición mediante *single integrator control*, como medida de seguridad para evitar la colisión según se ve en la Figura A.2, estableciendo entre otras cosas, un radio de seguridad que determina cuán cerca pueden estar los dos robots, (en este caso 0.07) además de un límite de ganancia, denominado *barrier_gain* cuya función consiste en limitar lo rápido que se admite que los dos robots se acerquen entre sí (toma valor de 100 en este ejemplo).

### A.3.3. Experimento 3: control de velocidad

Se ha empleado como referencia el ejemplo *uni_go_to_point.py*. Se incluye a continuación un pequeño sumario de los distintos controladores de posición que proporciona Robotarium, y que se pueden consultar en el archivo controllers.py, disponible en la carpeta utilities (ver A.2.3.2):

1. *Single integrator position controller*: crea un controlador de posición para *single integrators*, dirigiéndolo al punto deseado empleando control proporcional.
2. *CLF unicycle position controller*: crea un modelo *unicycle* para controlar la posición y orientación del agente.
3. *CLF unicycle pose controller*: devuelve un controlador que conduce a un agente modelado como *unycicle* a una posición y orientación deseadas. El control se basa en una función Lyapunov.

4. *Hybrid unicycle pose controller*: conduce al robot en línea recta a la posición deseada y luego rota.

Al respecto del cálculo de la entrada del control, cabe comentar que se trata de un 3xN numpy array de estados *unicycle* [x; y; theta], que describe la posición actual del robot, siendo *positions* otro array que describe las posiciones deseadas para el/ los robots, esto es, *[x_goal; y_goal]*. La resta de ambos dos valores proporciona el error de posición, según se ve en el código empleado por The Robotarium adjunto seguidamente.

```
pos_error = positions - states[:2][:]
rot_error = np.arctan2(pos_error[1][:], pos_error[0][:])
dist = np.linalg.norm(pos_error, axis=0)

dxu[0][:]=linear_velocity_gain*dist*np.cos(rot_error-
states[2][:])
dxu[1][:]=angular_velocity_gain*dist*np.sin(rot_error-
states[2][:])
```

El array *dxu* constituye los dos comandos de velocidad, lineal y angular respectivamente, que posteriormente se van a aplicar al robot.

Probando con un *clf unycicle position controller* y distintos valores de K se han obtenido los siguientes resultados:

1. linear_velocity_gain=3, angular_velocity_gain=5 → inestable
2. linear_velocity_gain=1.5, angular_velocity_gain=5 → estable
3. linear_velocity_gain=1.5, angular_velocity_gain=1.5→ inestable
4. linear_velocity_gain=0.8, angular_velocity_gain=8 → estable, trayectoria menos abombada y más directa.

Conforme aumenta la inestabilidad del controlador, la trayectoria se abomba progresivamente e incluso no se llega a completar puesto que el robot comienza a girar sobre sí mismo. Se han mantenido pues los valores indicados por defecto en el código del Robotarium, que figuran el el último caso de los probados.

### A.3.4. Experimento 4: añadir formas impresas en el entorno

Puesto que el objetivo de este Trabajo de Fin de Grado es llevar a cabo la configuración distribuida de formas empleando la robótica de swarm, se ha considerado oportuno añadir una funcionalidad que represente gráficamente en el entorno del Robotarium la forma en cuestión que se pretende alcanzar.

Para este experimento, y en vista de las formas que se han implementado en este Trabajo de Fin de Grado, se ha decidido representar un círculo, pero téngase en cuenta que se admiten otras formas como se comentará a continuación.

De forma inicial, se define el círculo en el espacio de tal forma que, en caso de que no se especifiquen las coordenadas iniciales del mismo, se sitúe aleatoriamente en el espacio.

Nótese que la forma del obstáculo admite modificaciones por medio del parámetro marker, actualmente definido como marker= 'o' (círculo).

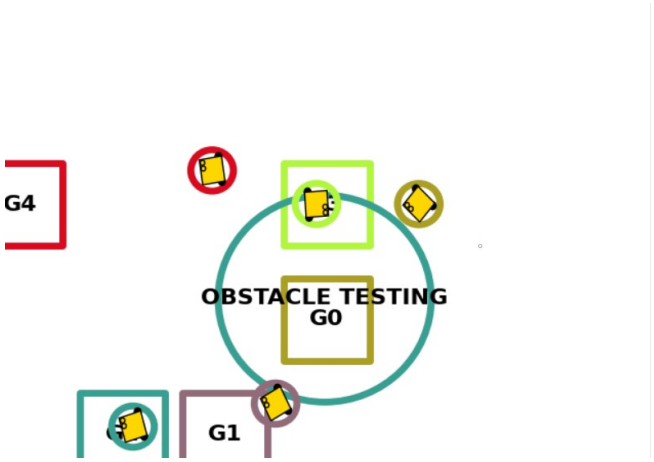



Figura A.3: Forma de círculo impresa en el entorno del Robotarium con leyenda *Obstacle testing*.