Universidad Zaragoza
1542

Trabajo Fin de Grado

Control de Cobertura para Redes de Sensores Móviles
Simulación y Experimentación en el Robotarium
Coverage Control for Mobile Sensing Networks
Simulation and Experimentation in the Robotarium

Autor/es

Adolfo Enrique Ber San Agustín

Director/es

Enrique Teruel Doñate

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2023

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Adolfo Enrique Ber San Agustín ,

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 Estudios de la titulación de

Grado en Ingeniería Electrónica y Automática (Título del Trabajo)

Control de Cobertura para Redes de Sensores Móviles. Simulaciónn y Experimentación en el Robotarium.

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

Zaragoza, 1 de Junio de 2023

Fdo: Adolfo Enrique Ber San Agustín

AGRADECIMIENTOS

Agradezco a:

mi tutor Enrique, por toda su paciencia y comprensión con mis idas y venidas en el desarrollo de este trabajo, y su inestimable ayuda y motivación cuando me surgían dudas.

mis padres Rosa y Benjamín, por exigirme sin juzgarme, por ser el apoyo más grande que he podido tener en todos los aspectos de mi vida, incluido el académico y por estar pendientes todo el rato.

mi hermana Marta, por ser un referente para mí aún siendo completamente distintos, por aportar orden y estabilidad a mi caos, y por todo el afecto y cariño que me da.

a todos mis amigos que he hecho durante los años de carrera, sin ellos hubiese sido imposible continuar en muchos momentos. En especial a Dani, por ser un generador de buenos recuerdos y un compañero excepcional.

al resto de mis amigos, por tranquilizarme y quererme en mis peores momentos, por no permitirme dejarlo todo y demostrarme que podía con lo que fuese.

a todas aquellas personas que me han escuchado y ayudado en cualquier momento. Gracias.

Control de Cobertura para Redes de Sensores Móviles. Simulación y Experimentación en el Robotarium.

RESUMEN

Existen multitud de situaciones donde es necesario explorar y monitorizar una región, pero hay situaciones específicas en las que el acceso para las personas es imposible, muy costoso, potencialmente peligroso o inefectivo. Como solución a este problema surge el desarrollo e investigación de los sistemas multi-robot. Con esta premisa nace la motivación de este trabajo, comprobar el funcionamiento en simulación y en robots físicos de un algoritmo de despliegue de sistemas multi-robot. Para ello se han utilizado los robots proporcionados por el Georgia Institute of Technology a través de uno de sus proyectos, The Robotarium. Además de validar el comportamiento de un algoritmo basado en teselaciones de Voronoi, se van a evaluar las limitaciones del Robotarium y su idoneidad como plataforma de experimentación.

Lo primero a realizar era la familiarización con la plataforma y su simulador, a través de una serie de experimentos sencillos con el propósito de entender las funcionalidades que se proporcionan y el funcionamiento y características de los robots. A la vez se debía investigar los algoritmos basados en teselaciones centrales de Voronoi y sus propiedades para la definición del algoritmo a desarrollar. Una vez realizada la investigación y con la ayuda del director se definió el algoritmo a desarrollar. El desarrollo del algoritmo debía definir un mundo en el que fuese posible visualizar los robots y sus celdas correspondientes en todo momento, en el que existiesen obstáculos y cuyo fin fuese ocupar todo el espacio disponible pudiendo darle forma a ese espacio a ocupar. Una vez creado había que ajustarlo experimentalmente con el sistema físico para que su comportamiento fuese correcto. Para concluir se debían diseñar diversos experimentos donde viésemos el potencial del algoritmo creado y obtuviésemos una comparación entre los resultados obtenidos entre la simulación y el entorno real. El paso definitivo, realizado de forma paralela al resto, era la redacción de la memoria explicativa del trabajo.

Coverage Control for Mobile Sensing Networks. Simulation and Experimentation in the Robotarium.

ABSTRACT

There are many situations where it is necessary to explore and monitor a region, but there are specific situations where access for people is impossible, too costly, potentially dangerous or ineffective. impossible, too costly, potentially dangerous or ineffective. As a possible solution to this problem has arisen the development and research of multi-robot systems. With this premise, the motivation of this work was born and its main objective is to test the operation, in simulation and on physical robots, of an algorithm for the coverage of multi-robot systems. To this end, the robots provided by the Georgia Institute of Technology through one of its projects, The Robotarium, have been used. As well as validating the behaviour of an algorithm based on Voronoi tessellations, the limitations of the Robotarium and its suitability as a platform for experimentation will be evaluated.

The first thing to do was to become familiar with the platform and its simulator, for which a series of simple experiments were to be carried out with the aim of understanding the functionalities provided and the operation and characteristics of the robots. At the same time, the algorithms based on central Voronoi tessellations and their properties had to be investigated in order to define the algorithm to be developed. Once the research had been carried out and with the help of the director, the algorithm to be developed was defined. The development of the algorithm had to define a world in which it was possible to visualise the robots and their corresponding cells at all times, in which there were obstacles and whose purpose was to occupy all the available space, being able to shape the space to be occupied. Once it had been created, it had to be experimentally adjusted with the physical system so that its behaviour would be correct. Finally, we had to design several experiments where we could see the potential of the created algorithm and obtain a comparison between the results obtained between the simulation and the real environment. The final step, which done in parallel to the rest, was the writing of the explanatory report of the work.

Índice

Capítulo 1

Introducción

1.1. Motivación

En un mundo donde la robótica cada vez está más presente en diversidad de campos, ya no sólo se limita su uso a un entorno industrial, sino que existen aplicaciones fuera de este entorno muy variadas, tanto con interacción humana como con el medio natural. Los robots destinados a estas aplicaciones se engloban en la robótica de servicio, dividida en servicio personal, profesional o humanoides/androides. La diversidad de tareas que este campo alberga supone que existan ocasiones donde sean necesarios varios robots trabajando en conjunto con el mismo objetivo, como los enjambres robóticos, robots individuales formando un solo conjunto con un objetivo único, de una manera similar a la que nos podemos encontrar en la naturaleza. La robótica de enjambres tiene multitud de aplicaciones, desde exploración de terrenos de difícil acceso, sistemas de vigilancia, rescates en terrenos de imposible o peligroso acceso, como pueden ser rescates en montaña o en zonas afectadas por catástrofes naturales, inspección de elementos o en limpieza. Aún con el potencial de ser robustos, escalables y flexibles, hasta ahora, los enjambres robóticos nunca han sido utilizados para aplicarse en el mundo real y de momento se mantienen en la investigación académica. En el actual estado de desarrollo en este campo, la investigación se focaliza en obtener los comportamientos colectivos deseados y entender sus propiedades [1]. Para evitar problemas que surgen en aplicaciones reales, investigadores usualmente se limitan a experimentos de aplicaciones simplificadas.

El desarrollo y estudio de estos sistemas robóticos adquiere mayor relevancia en el ámbito académico. Pero existe un inconveniente intrínseco y evidente, son necesarios varios robots, lo que no siempre es posible por un coste demasiado elevado de adquisición, por no tener espacios/recursos suficientes para realizar esos experimentos o por no poder admitir el mantenimiento pues requiere una dedicación alta para unos

niveles de utilización muy bajos. Esta dificultad propia e inevitable en el estudio de estos sistemas origina que la mayoría se realicen únicamente a través de simulación sin pruebas en un sistema físico real. Este es el motivo que originó este trabajo, explorar la herramienta del Robotarium como posible solución a este problema. El Robotarium es una plataforma propiedad del Instituto de Tecnología de Georgia, que proporciona un enjambre de robots autónomos accesible desde cualquier parte.

1.2. Objetivos

El principal objetivo de este proyecto es desarrollar un algoritmo de despliegue basado en teselaciones de Voronoi y comprobar su comportamiento, tanto en simulación como sobre los robots físicos proporcionados por la Universidad de Georgia Tech. De él se desprenden dos objetivos: en primer lugar, validar el funcionamiento del algoritmo desarrollado; en segundo lugar, hacer un análisis crítico del Robotarium, explicando sus beneficios y sus limitaciones, así como proponer recomendaciones y métodos de trabajo para un óptimo funcionamiento de éste.

Para lograr el primer objetivo debía entenderse los conceptos básicos de la técnica "Centroidal Voronoi Tesselations", leyendo diversos artículos de investigación donde se describen las ideas iniciales y las posteriores modificaciones y algoritmos que se han ido estudiando y desarrollando hasta las investigaciones más actuales, incluyendo las realizadas por mi director. Posteriormente debía definirse la versión del algoritmo a implementar sobre el Robotarium, adaptando las funciones de bajo nivel, para que el funcionamiento en el mundo físico fuese correcto y por tanto los experimentos válidos.

Respecto a la consecución del segundo objetivo, se debía iniciar con la familiarización con los métodos, los controladores y las especificaciones físicas propios del Robotarium, llevada a cabo mediante el desarrollo e implementación de diferentes casos prácticos sencillos. Posteriormente se debía hacer la adaptación del algoritmo y analizar la comparativa entre los resultados obtenidos en simulación y en el experimento físico. Finalmente realizar un análisis sobre el futuro uso de esta herramienta, así como las recomendaciones pertinentes para un óptimo uso de la misma.

1.3. Estructura de la memoria

El primer capítulo se destina a explicar el contexto en el que se da este trabajo, la motivación a realizarlo, los objetivos marcados y la memoria. A continuación, se encuentra la explicación del Robotarium y sus necesidades específicas, junto con los

primeros experimentos realizados para entender mejor el funcionamiento de éste. El siguiente capítulo explica la teoría tras el algoritmo de despliegue creado, explicando las diferentes casuísticas de las teselaciones centrales de Voronoi. El capítulo 4 explica el algoritmo propuesto, su programación a través de las clases y funciones más relevantes y la puesta a punto de éste con los ajustes que se debieron hacer. Tras este capítulo se presentan los resultados obtenidos y se analiza la idoneidad del algoritmo y del uso del Robotarium en futuras ocasiones. Para concluir, expreso mi opinión del uso futuro de esta herramienta y doy recomendaciones para que los posibles futuros usos sean óptimos.

Capítulo 2

Robotarium: explicación y familiarización

El Robotarium es un banco de pruebas con un enjambre robótico accesible de forma remota desde todo el mundo, desarrollado por el Georgia Institute of Technology [2]. Nació con la finalidad de democratizar el acceso a hardware propio de robótica, por ello su uso para investigación y el ámbito educativo es gratuito. Para empezar, vamos a hacer una breve presentación de las principales características del Robotarium que se compone de 4 elementos físicos: la pista o espacio disponible, el proyector, la cámara y los robots. La pista, es el espacio físico, en él se realizan los experimentos y se pueden mover los robots, mide 3.2m x 2m. El proyector posibilita representar imágenes sobre la pista para poder ver los puntos donde se dirigen los robots, darle una representación distinta a los obstáculos o proyectar las barreras asignadas a los robots. Existe una única cámara fija colocada sobre la pista, la cual posibilita una vista de pájaro sobre todo el escenario. Estos tres componentes nos permiten visualizar de forma muy parecida los experimentos a cómo se ven en el entorno de simulación. Por último, los robots, son los responsables de los posibles cambios entre la simulación y el experimento real en un sistema físico. Permite la utilización de un número variable de robots entre 1 y 20 y tienen unas dimensiones de 11 cm de ancho, 10 cm de largo y 7 cm de alto, sin incluir antena y sistema de rastreo, pero como nuestros experimentos se restringen a un plano 2D, la altura no es relevante. Es importante también conocer las limitaciones que estos robots tienen para moverse. Las velocidades máximas son 20cm/s de forma lineal y 3.6 rad/s en movimiento de rotación. Se debe prestar especial atención a las situaciones donde se combinan ambas velocidades, puesto que la velocidad de giro de las ruedas está limitada a 12.5 rad/s.

Para contextualizar un poco más el funcionamiento de los robots, vamos a explicar brevemente qué circuitos los conforman. La versión más actualizada está controlada por dos microcontroladores, un Atmega168/328 (8MHz, 16/32 KB flash,

2 KB RAM) encargado del control de los motores y por lo tanto en la placa correspondiente. Y un ESP8266 (80/160 MHz, 80kB DRAM (Data RAM), 35kB IRAM (Instruction RAM)) encargado del control general. Ambas placas se pueden programar con Arduino IDE. Para la programación el robot debe ser desmontado. La placa del motor necesita comunicación SPI, por lo que necesita un programa especializado. Sin embargo, la principal tiene un gestor de arranque de serie por lo tanto sólo necesita un convertidor USB-to-TTL para hacerlo de forma directa. La placa principal es la encargada del control de alto nivel del comportamiento del robot, de las comunicaciones y el suministro de energía. Los principales componentes que la forman son: ESP8266 microcontroller, donde también se encuentra el hardware del WiFi y ejecuta el software específico; MCP73831 LiPo chip cargador de batería; regulador de voltaje AP2112K-3.3V, capaz de suministrar 600 mA; un convertidor boost MCP1640, suministra la energía a los motores; sensor de voltaje y corriente INA219 I2C; por último, un chip de autenticación ATECC108. La placa del robot contiene todos los componentes relacionados con el movimiento. Como apunte, contiene sensores infrarrojos orientados hacia el suelo para poder realizar aplicaciones de seguimiento de líneas. Está compuesta por: un microcontrolador Atmega168/328, dos drivers LB1836M para los motores, dos sensores infrarrojos QRE1113, un circuito de protección para los pines de entrada del cargador, y dos LEDs.

Además de todo el soporte físico el Robotarium suministra el código de un simulador con las características del sistema real, con el objetivo de ejecutar el experimento en él de forma previa a enviar el código para su ejecución en el sistema físico. Además este simulador te da *feedback* sobre si es un experimento con altas probabilidades de ser ejecutado correctamente o por el contrario tiene problemas, aunque tampoco te da total seguridad. Añadido al simulador se pueden descargar varios ejemplos de posibles experimentos que sirven como introducción a las funcionalidades propias del Robotarium. Las librerías y el simulador están disponibles Matlab y Python. El lenguaje original es Matlab, tiempo después decidieron crear la estructura con un lenguaje que fuese libre, para facilitar el acceso a más personas. De los lenguajes universalmente accesibles, dudaron entre Python y ROS, decantándose por el primero al ser un lenguaje menos específico, más conocido y flexible. A raíz de esto el primer paso era familiarizarse con estas librerías y con el simulador proporcionado por el Robotarium, con la realización de sencillos experimentos iniciales. El objetivo de estos primeros experimentos era entender los parámetros que se podían cambiar usando las librerías proporcionadas, empezando desde recorridos con un solo robot, ir aumentando el número de robots y finalmente añadir complementos visuales para facilitar el entendimiento del experimento en cuestión.

2.1. Primeros pasos

Lo primero era realizar movimientos sencillos en el simulador con los controladores de posicionamiento propios del Robotarium. Son cuatro controladores, el primero un control proporcional de la posición para un sistema integrador, simplemente multiplica por una ganancia el error en la posición y normaliza el valor para que sea menor de la velocidad máxima. El siguiente controlador es otro proporcional pero para robots con tracción diferencial, controlando velocidad lineal y angular. El tercero también es un controlador para uniciclos, esta vez controlando posición y orientación. El último es también controlador de orientación y posición pero en esta ocasión separan ambos movimientos, primero se hará el movimiento en línea recta hasta la posición deseada con un proporcional y después rotará hasta la orientación deseada. Estos controladores sólo serán utilizados en los experimentos iniciales para la familiarización, no en el cuerpo del trabajo donde especificaremos cuál debe ser la velocidad lineal y angular de cada robot, y recogeremos donde están cuando lo necesitemos.

Por último, subí algún fichero a la plataforma del Robotarium para entender los pasos necesarios para la entrega de un experimento en la página web. Un proceso muy sencillo donde primero accedes a tu cuenta, vas a tu dashboard personal y allí tendrás todos los experimentos realizados y los experimentos pendientes de realizar. Encima de estos últimos hay un recuadro verde donde pone: *+ Submit New Experiment*, clicas en él y te llevará a una nueva página donde debes poner los datos de tu experimento, nombre, duración estimada, descripción del experimento, número de robots, y código a ejecutar, todos ellos obligatorios.

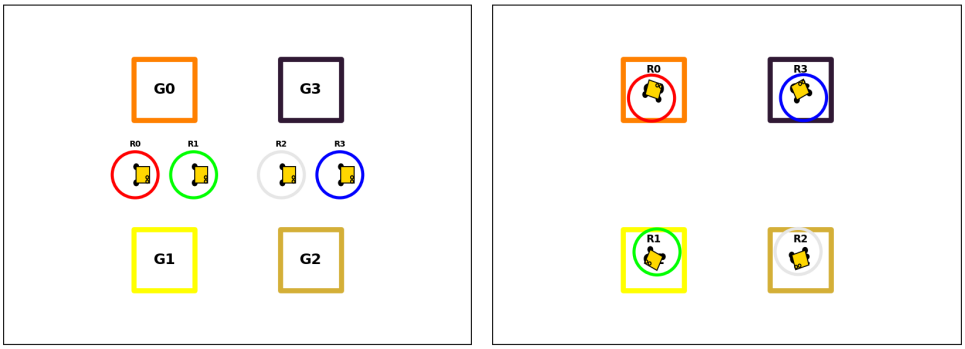
Enlace a la lista de reproducción que contiene todos los vídeos iniciales: `https://www.youtube.com/playlist?list=PLpfhyB7Xtf4aqLbie9QiqbBTEk90z_E6N`

2.2. Primer experimento

El Robotarium exige un método de prevención de las colisiones entre robots. Da la libertad de usar uno propio, pero el experimento no se realizará si en el simulador se producen choques entre robots. Como última prevención de las colisiones física se ejecutan los métodos propios del Robotarium para evitar esa colisión, con el fin de que los robots no puedan sufrir daños. Para ver su funcionamiento y entenderlos bien diseñé el primer experimento.

El primer experimento consistía en 4 robots moviéndose a la vez, cada uno de ellos haciendo un recorrido en cuadrado de tal forma que entre cada posición se encontraban de frente dos robots que iban en sentido inverso. Antes de comenzar el movimiento en

cuadrado, deben posicionarse desde la línea central (2.1a) en su respectiva esquina inicial (2.1b). Aclaración, el robot en la esquina superior izquierda irá a la esquina izquierda inferior, esquina derecha inferior, esquina superior derecha y vuelta a la posición inicial. Y en su recorrido se encuentra primero con el robot 2, luego con el robot 4, otra vez con el 2 y por último con el 4. Para cada uno de los robots utilizamos un controlador del tipo integrador único, para individualizar el control de cada uno. Para este experimento, utilizamos los métodos propios del Robotarium para dibujar información sobre el escenario. Esto aporta una visualización más fácil de las posiciones a las que deben ir y también de las barreras creadas alrededor de los robots. Además, añadimos otra funcionalidad del Robotarium que es el guardado de datos en un fichero .txt en el cual escribimos las velocidades y las distancias hasta el objetivo, para poder comparar la simulación con el experimento real.



(a) Posición inicial primer experimento (b) Primera posición del cuadrado

Figura 2.1: Movimiento inicial en el primer experimento

Para finalizar voy a explicar la función principal de prevención de colisiones, *create_single_integrator_barrier_certificate*. Los parámetros que necesita en su definición son:

- *barrier_gain*: valor que controla la rapidez con la que se pueden aproximar los robots. Un valor más pequeño significa que se aproximarán más lento.
- *safety_radius*: distancia mínima entre robots
- *magnitude_limit*: velocidad lineal máxima

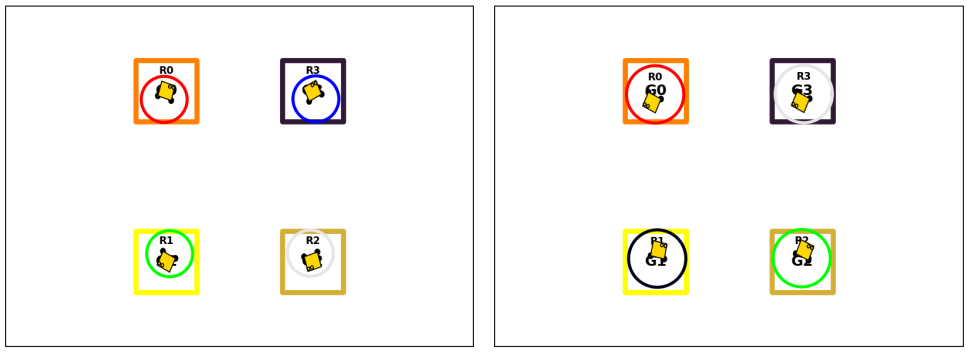
Al llamar a la función debemos pasarle como parámetros la velocidad actual de los robots y su localización. Esta función principal resuelve una ecuación cuadrática encontrando los valores que la minimizan mediante el comando *qp* de la librería *CVXOPT*. La ecuación a resolver es:

$$min_x = \frac{1}{2}x^T Px + q^T x \text{ subject to } Gx <= h, Ax = b \tag{2.1}$$

En nuestro código P es una matriz dispersa de doses, q es la matriz de velocidades convertida en fila y multiplicada por -2, G es la matriz distancia, siendo cada valor la distancia entre posición actual de un robot y la posición de uno de sus vecinos. Y h una matriz donde cada valor viene determinado por: $barrier\_gain * (distancia\_en\_x^2 + distancia\_en\_y^2 - safety\_radius^2)^3$. Tras la construcción de la ecuación a resolver, se utiliza el comando qp y el resultado será una nueva matriz de velocidades a aplicar. Las nuevas velocidades minimizan el acercamiento entre robots cercanos.

2.3. Segundo experimento

El objetivo de este segundo experimento era mejorar el comportamiento observado en el anterior, donde se producían muchos errores por no controlar la orientación y se sobrepasaban los límites de velocidad en varias ocasiones. Para ello implementé los controladores de posición más orientación, los cuales aparte de suponer una mejora en la suavidad de los movimientos, permiten ajustar más parámetros para reducir las acciones a implementar. Además, se realizaron mejoras en la propia implementación, ahora sólo utilizamos un controlador, cuando en el anterior utilizábamos cuatro, uno para cada robot, lo que facilita el código. Este cambio se debe a una mejor comprensión de los argumentos utilizados por las funciones encargadas del cálculo de las velocidades. En la imagen inferior (2.2), podemos ver la diferencia de posicionamiento después del movimiento inicial de colocación.



(a) Posición primer experimento (b) Posición segundo experimento

Figura 2.2: Comparación de posición tras movimiento entre 1º y 2º experimento

2.4. Tercer experimento

Este experimento consistía en explorar otras posibles funcionalidades con el código dado junto al simulador, en situaciones más parecidas a las que se iban a implementar en los experimentos de nuestro trabajo. Por eso, en este experimento queremos conocer

y modificar el comportamiento sólo con aquellos robots que tengamos cerca, los vecinos del robot actual, para ello utilizamos la función *topological_neighbors (L, i)* donde L representa una matriz Laplaciana de todo el gráfico completo e i representa el robot sobre el que se realiza la operación. Tras utilizar esta función obtenemos un array con los vecinos del robot actual, es decir, aquellos que están conectados en la matriz. El código restante es muy sencillo, primero multiplicamos la ganancia por la distancia euclídea entre distancia actual y deseada, entre el robot y un vecino. Segundo multiplicamos ese valor por la distancia real entre vecinos. Y la velocidad final es la suma de todos esos valores. Después comprobamos si hay dos robots a la distancia deseada, y si es así ese robot cumple su requisito. Y, por último, comprobamos si todos los robots cumplen este objetivo lo que significa que la formación ha sido obtenida y el experimento finalizado. Al terminar, cada robot acaba en una punta simbólica de una estrella, visible en la figura 2.3. Por lo tanto, este experimento nos sirve para entender los métodos que tiene el Robotarium para detectar robots vecinos, aspecto crucial en el desarrollo del algoritmo de CVT. Aunque realmente no se vaya a utilizar esa función se entiende cómo sería un funcionamiento en el que no tienen la información de todo lo que hay en el espacio disponible.

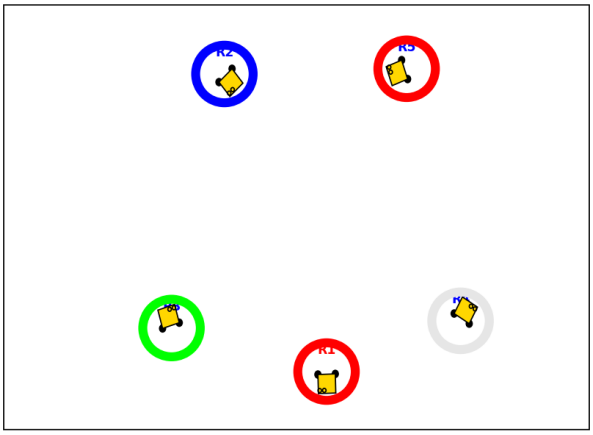


Figura 2.3: Posición final experimento 3 e inicial experimento 4

2.5. Cuarto experimento

El objetivo de este último experimento era probar otra vez entender el comportamiento de los robots cuando sólo reciben información de sus vecinos y no de todo el espacio. Con esta finalidad realicé un ejemplo de algoritmo de consenso, donde todos los robots deben ir al mismo punto y moverse de forma igualitaria. Iniciamos con la función *topological neighbors(L, i)*, pero ahora L no es el gráfico Laplaciano del espacio completo, sino de un grafo circular. Una vez obtenemos los vecinos de nuestro

robot, calculamos las distancias con cada uno de ellos, en coordenadas x e y, sumamos todas y obtenemos la velocidad en ambas direcciones a la que tiene que ir nuestro robot. Pasamos la velocidad por la función para evitar colisiones, traducimos de *single integrator* a robot uniciclo y aplicamos esas velocidades.

Capítulo 3

Algoritmo de despliegue basado en CVT

Uno de los puntos cruciales de cómo distribuir un enjambre robótico en un espacio, es la partición de ese mismo espacio en regiones. Evidentemente debe haber tantas regiones cómo robots compongan el sistema. Hay multitud de formas de dividir el espacio de forma matemática: partición binaria del espacio, consiste en dividir recursivamente un espacio euclídeo en dos con hiperplanos; partición de polígonos, divide un polígono en polígonos básicos, cómo triángulos o cuadrados, que no se superponen y su unión da cómo resultado el polígono inicial; o el teselado, cubre el espacio con una o más formas geométricas sin superposición ni huecos, aquí se encuentran los diagramas de Voronoi y la triangulación de Delanuy.

La forma matemática escogida para realizar esta partición han sido los diagramas de Voronoi. Los diagramas de Voronoi dividen el espacio en regiones con forma poliédrica, denominadas celdas o teselas de Voronoi, a raíz de unos puntos denominados generadores. Este diagrama tiene multitud de propiedades positivas, como son: los dos generadores más cercanos van a estar en celdas adyacentes, lo que implica que la búsqueda del vecino más próximo se restringe a celdas adyacentes; puede generalizarse a un espacio de n dimensiones; pueden utilizarse distancias distintas a la euclídea; en nuestro caso utilizamos la distancia euclídea y esto implica que sea convexo, es decir, la línea que une dos puntos dentro de una celda está siempre dentro de la celda; y por último, existe estabilidad geométrica, un pequeño cambio en la ubicación de los centroides calculados supone un cambio pequeño en las teselas calculadas.

Esta formulación matemática, no es nueva para nadie, incluso si nunca la ha oído en su vida habrá visto multitud de ejemplos en la naturaleza, como puede ser la piel de una jirafa, la forma que adquieren las pompas de jabón al estar junto a otras, o las regiones de dominio de un animal en un ecosistema. Tiene aplicaciones en infinidad de campos,

desde meteorología, ciencias naturales, robótica, geometría, inteligencia artificial o en planificación urbanística.

Antes de definirlo formalmente, voy a hacer una breve explicación de la idea principal de este método, puesto que la idea básica es intuitiva y fácil de entender, antes de perdernos en las matemáticas. Imaginemos el caso más sencillo posible, un plano con dos puntos y queremos obtener la división de ese espacio en dos regiones en las cuales los puntos de las mismas estén más cerca de un punto que del otro. Para obtener el resultado deseado sólo haría falta trazar la línea de unión entre los dos puntos, en esta línea encontrar el punto medio, equidistante a ambos puntos, y desde ese punto medio, trazar una línea perpendicular a la primera línea. Esta línea nos ha divido el espacio en dos regiones, cada una de ellas nos indica el punto que se encuentra más cerca de un tercer punto aleatorio. Y la línea divisoria es equidistante a los dos puntos.

En el caso de estudio de este trabajo, seguimos en un mundo plano, es decir, de dos dimensiones, pero donde nuestro conjunto de puntos (cada punto representa un robot) es variable pero mayor que dos y la región a cubrir puede no ser todo el plano. Aun así la idea sería la misma, calcular de forma iterativa para cada punto la línea que defina los puntos equidistantes de ese punto con todo el resto del conjunto, así para todos los robots en el sistema. Con esta idea básica se calcularían demasiadas líneas inútiles, pues no todas las regiones se ven afectadas por todos los puntos, sino sólo por aquellos que están a una distancia cercana, estos son los vecinos del robot x.

Para definir formalmente los diagramas de Voronoi necesitamos un subconjunto $\Omega \subseteq \mathbb{R}^{\mathbb{N}}$, el conjunto $V_{i=1}^{k}$ es una teselación de Voronoi de Ω, si $V_i \cap V_j = \emptyset$ para todo $i \neq j$ y $U_{i=1}^{k}\overline{V_i} = \overline{\Omega}$. Siendo $||$ la norma Euclídea en $\mathbb{R}^N$. Dado un grupo de puntos $z_{ii} = 1, ..., k$ perteneciente a $\overline{\Omega}$, la región V_i correspondiente al punto z_i está definida por: (3.1)

$$V_i = |x - z_i| < |x - z_j| \text{ para } j = 1, ..., k, j \neq i \tag{3.1}$$

esta fórmula, define que una celda de Voronoi es aquella zona donde la distancia entre un punto y el generador i es menor que la distancia entre ese punto y cualquiera de los restantes generadores. Por lo tanto los puntos $z_{ii} = 1^k$ son los generadores, el subconjunto $V_{ii} = 1^k$ es la celda o teselación de Voronoi respectiva al punto z_i. Las regiones resultantes son poliédricas.

En el particular caso que nos ocupa en este trabajo no queremos cualquier solución para nuestras regiones calculadas, queremos la beneficiosa situación conocida como Teselación de Voronoi Centrada (Central Voronoi Tessellation, CVT, en inglés).
Conocida una región $V \subseteq \mathbb{R}^N$, y una función de densidad ρ, definida para V, podemos

definir varias cantidades que necesitaremos para el cálculo de nuestros algoritmos. Estas cantidades son, la masa, el centro de masas o centroide y el momento de inercia, definidas por: 3.2

$$M_V = \int_V \rho(q)dq, z_i^* = \frac{1}{M_V}\int_V q\rho(q)dq, J_{V,p} = \int_V ||q-p||^2\rho(q) \tag{3.2}$$

Para obtener la configuración deseada queremos la situación donde : $z_i = z_i^*, i = 1, ..., k$. Los puntos z_i, nuestros generadores para las teselaciones son a su vez los centros de masas de esas teselaciones. La solución a este problema generalmente no es única. Por ejemplo, en el caso con $N = 2, \Omega \subseteq \mathbb{R}^2$ sea un cuadrado, y $\rho = 1$. Podemos obtener las soluciones mostradas en la Figura 3.1, pero otras podemos obtenerlas mediante rotación. Otro ejemplo es las tres teselaciones normales de $\mathbb{R}^2$ en cuadrados, triángulos y hexágonos. ([3])

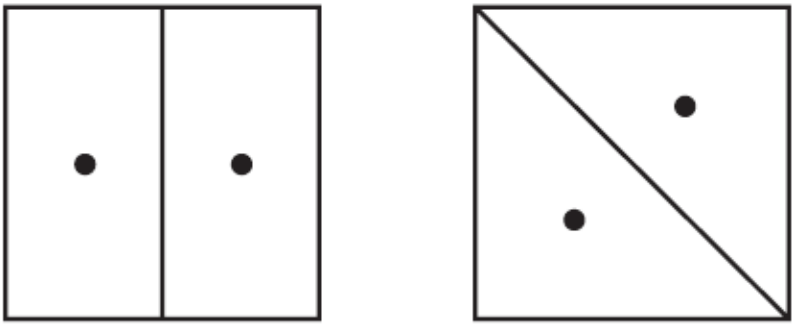


Figura 3.1: Dos CVT de un cuadrado, y sus centroides.

Cálculos con polígonos con densidad uniforme En la ecuaciones 3.2, hemos definido todo en base a la función de densidad $\rho(q)$, pero no hemos definido ninguna propiedad de la misma. En nuestros caso, es una función de densidad uniforme, lo que nos permite simplificar las ecuaciones considerablemente. Asumiendo que la región Voronoi V_i es un polígono convexo con N_i vértices $(x_0, y_0), ..., (x_{N_i-1}, y_{N_i-1}$. Es conveniente definir $(x_{N_i}, y_{N_i}) = (x_0, y_0)$ y asumir que $\rho(q) = 1$. Evaluando las ecuaciones 3.2, podemos obtener:

$$M_{V_i} = \frac{1}{2}\sum_{k=0}^{N_i-1}(x_k y_{k+1} - x_{k+1}y_k) \tag{3.3}$$

$$z_{i,x}^* = \frac{1}{6M_{V_i}}\sum_{k=0}^{N_i-1}(x_k + x_{k+1})(x_k y_{k+1} - x_{k+1}y_k) \tag{3.4}$$

$$z^*_{i,y} = \frac{1}{6M_{V_i}} \sum_{k=0}^{N_i-1} (y_k + y_{k+1})(x_k y_{k+1} - x_{k+1} y_k) \tag{3.5}$$

Los vértices de la celda pueden expresarse como una función de sus vecinos. Los vértices del polígono i dentro de Q son los circuncentros de los triángulos formados por p_i y otros dos vecinos adyacentes cualquiera.

Para no alejarnos del mundo real, vamos a presentar brevemente tres ejemplos de donde se puede aplicar CVT. ([3]) En procesado de imagen, podemos reducir la información, por ejemplo en una imagen con muchos píxeles donde cada píxel tiene asignado un color, una posible combinación de los tres colores básicos. Si tenemos 10^6 píxeles con un color asociado, si este color se describe con un número de 24 bits, si lo podemos describir con un número de 8 bits, se reduce la cantidad de datos un tercio sin cambiar la resolución.
El segundo ejemplo, es la representación óptima de datos observados. Una de las primeras aplicaciones fue la estimación de la precipitación total en un año e una región geográfica. Formulando la precipitación de una zona como: $p(x) = m(x) + \epsilon(x)$, con m(x) una función continua y ϵ una función aleatoria con valor esperado cero, minimizar el error estimado se reduce a encontrar el diagrama de Voronoi centrado.
El tercer ejemplo, es uno de los más prácticos, la asignación óptima de recursos distribuidos en una ciudad o una zona de esa ciudad, cómo podrían ser hospitales, farmacias o cubos de basura, de forma que estén lo más cerca posible de todos los usuarios. El coste individual es una función de la distancia entre la casa y el recurso a distribuir, el coste total es la distancia media, la localización óptima es la que minimiza el coste total. El coste total se define como:

$$\varepsilon(x_i, i = 1, ..., k) = \sum_{i=1}^{k} \int v_i f(|x - x_i|)\phi(x)dx \tag{3.6}$$

Es fácil ver que la solución óptima va a ser un CVT, utilizando la densidad de población cómo $\phi(x)$ y $f(z) = z^2$.

3.1. Algoritmo distribuido

La aplicación de las Central Voronoi Tessellation, que nos ocupa es el despliegue de redes sensoriales de robots autónomos móviles. Los problemas de optimización espacial

han sido muy estudiados por multitud de disciplinas, pero normalmente asumen una computación centralizada, la cual en el caso de las redes multi–robóticas móviles no es aplicable. Esto se debe a que estos sistemas están basados en una comunicación y computación distribuida. Por lo tanto necesitamos algoritmos que tengan en cuenta las particularidades de estas redes. Deben ser adaptables, capaces de ajustarse a cambios en el entorno, en la tarea o en la topología de la red (pueden aparecer o desaparecer agentes). Distribuidos, el comportamiento de cada vehículo depende exclusivamente de la localización de sus vecinos. Asíncronos, capaces de utilizarse en un sistema donde los componentes evolucionen con velocidades y capacidades de cálculo y comunicación distintas. Por último tienen que garantizar que al alcanzar las configuraciones CVT, los centroides sean puntos críticos de la solución óptima de la cobertura de los sensores.

$$H(P,W) = \sum_{i=1}^{n} \int_{W_i} f(||q - p_i||)\phi(q)dq \tag{3.7}$$

En esta definición asumimos que el sensor i, es responsable de tomar las medidas de su región W_i. Resaltar que la función H deber ser optimizada tanto para la localización de los sensores cómo para la asignación de las celdas W. Si a esta función aplicamos la definición de los diagramas de Voronoi, tenemos $min_{i\in(1,\ldots,n)} f(||q - p_i||)$ todo $q \in V_j$, con lo que obtenemos:

$$H_V(P) = E_{(Q,\phi)}\left[min_{i\in(1,\ldots,n)} f(||q - p_i||)\right] \tag{3.8}$$

la función a optimizar se puede considerar como un valor esperado en una operación de mínimo. A destacar uno puede demostrar que:

$$\frac{\partial H_V}{\partial p_i} = \int_{V_i} \frac{\partial}{\partial p_i} f(||q - p_i||)\phi(q)dq, \tag{3.9}$$

La derivada parcial de H_V respecto al sensor ith depende únicamente de su propia posición y la posición de sus vecinos.

Siendo $V(u) = V - u$. Para cada punto v en $V(u)$, vamos a declarar el punto medio del segmento uv, como v', mostrado en la figura 3.2 . Definimos la región adjunta como (3.10)

$$E(u) = \cap_{v\in V(u)} H_V, \tag{3.10}$$

La región de Voronoi y la adjunta se muestran en la figura 3.2. Son similares en forma, con los vértices de la celda situados a la mitad de los vértices respecto a u. Claramente,

da igual buscar cualquiera de las dos regiones. De aquí en adelante hablaremos siempre del cálculo de la región adjunta.

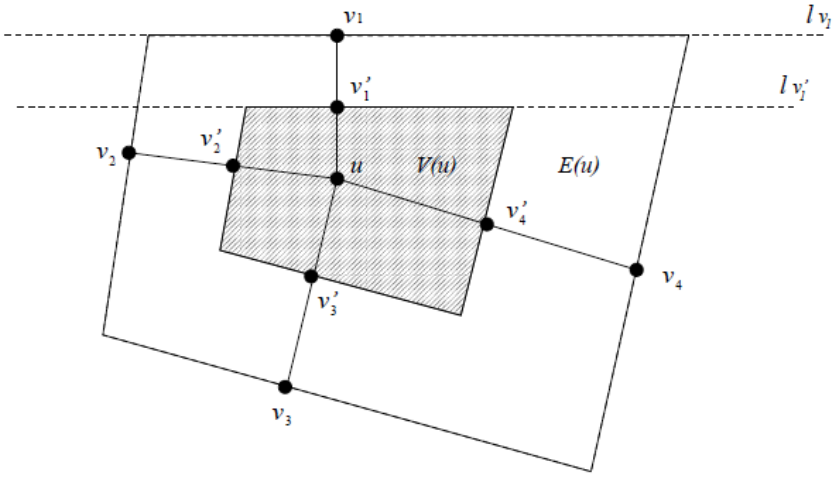


Figura 3.2: Región de Voronoi y región adjunta del nodo u [4]

Un punto que define un segmento de una celda de Voronoi del punto i, es un vecino de i. Llamamos al número de vecinos de i como $N(i)$. Al ser un algoritmo distribuido cada unidad tiene que calcular su propia teselación de forma autónoma. Debemos tener en cuenta que existe un gran coste temporal de comunicación asociado a recibir y mandar información a los nodos. Teniendo n nodos en un área cerrada S, ordenados de manera ascendente por la distancia desde el nodo u, el algoritmo propuesto funciona de forma que cada nodo mira a sus vecinos, definidos por un radio propuesto, e ir aumentando este vecindario hasta que tenga completo conocimiento de su teselación de Voronoi.

Para calcular esta región es tan simple cómo calcular la intersección de S con el semiplano definido por el nodo más cercano, así tantas veces cómo nodos tengamos. Cada iteración que realizamos sólo utiliza cálculos geométricos básicos, y esto implica que el coste computacional sea reducido. Pero aún así ya hemos comentado que no todos los nodos se ven afectados por todos sino que sólo necesitan la información de unos pocos de su alrededor. Si no necesitamos todos, para que calcularlos, pero que criterio utilizamos para detener el proceso iterativo. Muy simple, este criterio va a ser la máxima distancia posible al nodo u dentro de la región en cada iteración.

Si encontramos el círculo $D(u, r_i)$ donde los únicos nodos que estén dentro sean los nodos ya explorados, habremos encontrado nuestra región adjunta y nuestra celda de Voronoi. Se puede comprobar que esto es así, en la figura 3.3 podemos observar un círculo centrado en un vértice v, con radio $r_i/2$. La definición de las teselaciones de Voronoi este círculo sólo puede tener dos nodos en su borde y ninguno de los

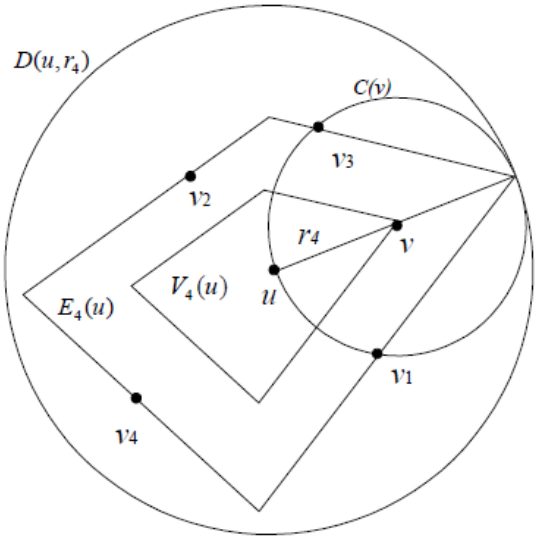


Figura 3.3: Representación del criterio utilizado para detener el proceso iterativo. [4]

ya explorados en su interior. Al no haber nodos dentro de $D(u, r_i)$ excepto los anteriormente explorados, y que $C(v)$ pertenece a $D(u, r_i)$, $C(v)$ está vacío. Si esto pasa con todos los vértices v en $V_i(u)$, $V_i(u) = V(u)$. En esta iteración hemos encontrado la celda, debemos parar el proceso iterativo.

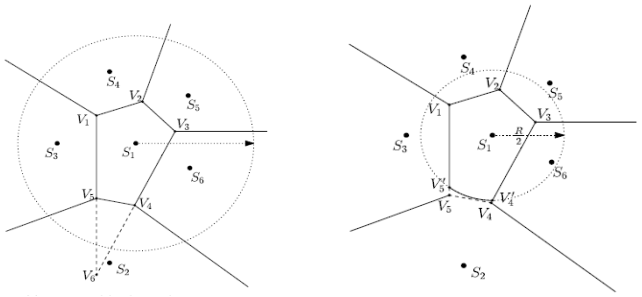
El círculo $D(u, r_i)$ puede contener más nodos, pero estos no se pueden encontrar dentro de la región adjunta. Teóricamente si los nodos están muy espaciados, los vecinos de un nodo u son 6, podemos por lo tanto concluir que en un sistema de comunicación entre robots móviles con comunicación entre ellos, es mucho más eficiente y práctico calcular las celdas de Voronoi utilizando un algoritmo distribuido y no centralizado.

Nuestra disposición es dinámica, después del primer cálculo el sistema en conjunto va a tener que realizar un movimiento, o pueden darse cambios, puede incluirse un nuevo nodo o dejar de funcionar uno de los existentes, por lo tanto la red debe ajustarse a la nueva realidad. Este movimiento dentro del sistema puede ser visto como dos eventos: se desactiva el nodo de la posición inicial y se activa un nuevo nodo en la posición final. Primero, cómo afecta el nuevo nodo, w. Es evidente que este nodo sólo afectará a los nodos cuyo $D(u)$ contenga w, para ello w comunica su posición a los robots de su entorno, estos comprueban si van a ser afectados o no. Si son afectados, la región $E(u) = E(u) \cap H_w$, si el resultado no es conjunto vacío, se añade w a la lista de vecinos de u $N(u)$ y se eliminan aquellos que ya no definen el borde. Para borrar un nodo, sólo afectará a los nodos u que sean sus vecinos, y puede introducir nuevos vecinos. Si w es el vecino i y E_i está definida por $v_1, ..., v_{i-1}$ redefinimos $i = i + 1$ y aplicamos el

algoritmo anteriormente explicado hasta que encuentre la nueva $E(u)$ y se detenga.

3.2. Celdas Voronoi con comunicación limitada

Ahora planteémonos una situación en la que sea imposible para un nodo recibir información de todos sus vecinos, posiblemente porque uno de ellos se encuentre fuera de su rango sensorial. La consecuencia derivada es que probablemente no sea viable encontrar la solución correcta, lo que suponga mayores distancias recorridas e imposibilitando que converja la solución. Esta situación la podemos visualizar en la figura 3.4a. Para solucionar este inconveniente definimos un nuevo rango de comunicación a la mitad del rango real. Ahora para calcular la celda de Voronoi utilizaremos el área superpuesta entre la celda calculada erróneamente y la circunferencia de radio R/2. En nuestro ejemplo quedaría cómo la figura 3.4b.



(a) Problema por rango de comunicación

(b) Celda de Voronoi obtenida con R/2

Figura 3.4: Problema debido al limitado rango de comunicación y la aplicación del cálculo con R/2 [5]

Calculamos así la nueva celda, para asegurar que una teselación no contenga puntos que se encuentren en otra teselación, y esto lo podemos asegurar porque si dos sensores no se pueden comunicar las circunferencias definidas por la mitad del rango no pueden superponerse entre ellas, por lo tanto no comparten puntos. Así un robot sin vecinos no se moverá, pero aquellos a los que les falte información se moverán hacía un despliegue definido por la circunferencia R/2 y sus vecinos visibles. Conforme los robots se vayan moviendo y reubicando irán aprendiendo de nuevos vecinos y se conseguirá el despliegue global.

Capítulo 4

Desarrollo del trabajo

4.1. Definición del algoritmo

Las principales características del algoritmo a implementar ya han sido fundamentadas y explicadas en el anterior capítulo. Está basado en un algoritmo de teselaciones centrales de Voronoi, con un sistema distribuido y en el que los robots tienen un rango sensorial limitado. Además, la celda calculada con estas características va a disminuir su tamaño con una distancia de seguridad. Está distancia simplemente acorta la celda al robot, añadiendo una capa más de seguridad frente a posibles colisiones con otros robots u obstáculos. [6] Por ello necesitaremos crear un entorno donde se ejecute todo, en el que definir nuestros robots y sus capacidades, los obstáculos, ya sean o no móviles, el control de las velocidades, como guardar las estadísticas y los parámetros más importantes, así como una función que calcule la distribución espacial de todos los robots. La función *voronoi* encargada de los cálculos utilizará las librerías *Point2D* y *shapely.geometry*, cuya implementación la explicaré en profundidad más adelante. En este mundo es donde se va a realizar la ejecución de todo en cada iteración de 33 ms, el flujo de una iteración queda reflejado en el pseudocódigo 1.

4.2. Programación

Los ficheros principales de este trabajo son tres, *Utilities*, *World* y *main/experiment*, aunque cada uno de ellos tiene diferentes versiones debido a su posterior uso, simulación o implementación física. Enlace a una carpeta de Drive con todo el código importante, `https://drive.google.com/drive/folders/1mr8A-L7A3G7LqsFEvXWBEpLuu5kEf6YQ?usp=share_link`

4.2.1. Archivo Utilities

En este archivo encontramos algunas funciones útiles para el resto del código y que se usan de forma recurrente. Existe una función para crear obstáculos de forma aleatoria, las funciones específicas para crear las gráficas de las simulaciones, la definición de la clase $Point2D$ y la función responsable del cálculo de cada teselación de Voronoi. La definición de Point2D como una clase propia dentro de mi código se debe a la imposibilidad de realizar un *import* en el experimento real, porque en el Robotarium no está disponible.

Vamos a detenernos a explicar la función *voronoi*, responsable del cálculo de Voronoi. El proceso iterativo, para cada robot (me) y cada vecino de él($neigh$) se hace lo mismo, queda reflejado en la figura 4.1. Primero se define la posición del robot, cuadrado denominado me y alrededor suyo la celda creada con la función $Polygon(Point(me).buffer(r))$, donde r es el rango sensorial. Después se calcula la distancia (v) con el vecino y se define el punto medio, pc en el dibujo. A continuación, se define un nuevo punto central (pc'), restando la distancia de seguridad a pc. Tras esto, se redefine el módulo de v igualándolo al radio sensorial. Ahora se define el triángulo, donde Q0 se encuentra a dos veces la distancia v del vecino, y Q1 y Q2 a 2*v del pc' en la perpendicular a v. Definido el triángulo sólo queda restarlo a la celda original, resultando la celda sombreada a rayas.

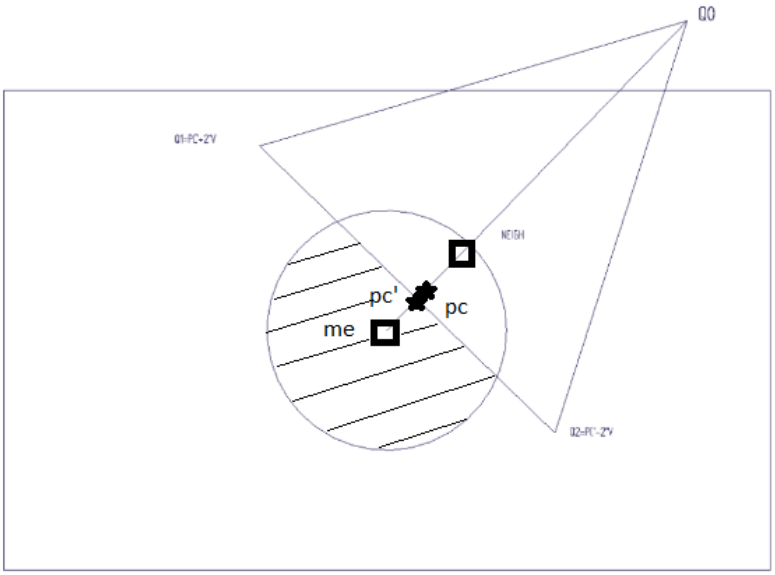


Figura 4.1: Representación del cálculo de la teselación de Voronoi

4.2.2. Archivo World

Es el archivo principal, en el definimos todo: el entorno, los obstáculos, los robots, el control del movimiento y el guardado de parámetros característicos; utilizando algunas funciones y pensando siempre en la posterior aplicación en el Robotarium. Vamos a ir clase por clase explicando su motivo y sus características más relevantes.

Clase Space

Representa el espacio del Robotarium, es la primera en ejecutarse y es la encargada de ejecutar cada iteración mediante la función *step*. Tiene muchas variables de entrada: *name*; *numRob*, número de robots a crear; R, radio sensorial; fix_obs obstáculos fijos; *initial_conditions* una matriz de 3xnumRob representando las localizaciones iniciales de los robots; $point, width, height$ definen un rectángulo donde se van a posicionar los robots al iniciar de forma aleatoria, *point* es la esquina superior derecha; $safe$, distancia mínima entre objetos al inicio; $safety\_radius$, distancia de seguridad a recortar en las teselaciones; dt, incremento temporal de cada iteración.
Principales funciones definidas en *Space*

1. *nearby*: Devuelve una lista con todos los objetos que están cerca de otro y sean de los tipos correctos. Necesita como parámetros el objeto que la llama, tipos a detectar, un valor que defina lo que es cerca y cuantos grados puede ver un robot.

2. *update_dist*: Actualiza la matriz de $len(bodies)$x$len(bodies)$, donde se guardan las distancias entre todos los objetos presentes.

3. *create_ini_cond*: Crea las localizaciones iniciales, utilizando la función del Robotarium *generate_initial_conditions* y comprobando que esas posiciones no colisionen con nada. Sus parámetros son el tipo de obstáculos, distancia entre localizaciones, rectángulo donde crearlas, y distancia de seguridad.

4. *MinimunDistance*: Devuelve la distancia mínima del cuerpo que sea a cualquier objeto.

5. *to_cover*: Define el valor del área a cubrir por los robots. Necesita la distancia de seguridad de las celdas para reducir el espacio. Básicamente es el área de ese rectángulo reducido menos las áreas de los obstáculos. Es una de las condiciones de parada del experimento conjunto a la desviación estándar de las áreas de las celdas de Voronoi.

6. *step*: Iteración principal, pseudocódigo en algoritmo 1.

Algorithm 1 Una iteración del algoritmo

```
for body in list bodies do
  if body is Robot or MovObs then
    body.update()
    if body is Robot then
      Añade body.vel to vel
      sum_area + = rob.area
    end if
  end if
end for
actualizar visualización y distancias
if es la primera iteración then
  step() en el Robotarium          {Necesitamos avanzar una iteración porque en la
  inicialización ya se ha utilizado la función get_poses()}
  Actualizar las posiciones
else
  Actualizar las posiciones
end if
Comprobar si vel_R calculada por el Robotarium es próxima a nuestra vel
for vel distinta do
  if hay otro robot a menos de 0.2 metros then
    colisiones +=1
    vel = vel_R
    break
  end if
end for
for robot do
  if hay algún obstáculo a menos de 0.12 metros then
    colisiones +=1
    break
  end if
end for
Actualizar índices cada 3 iteraciones          {Actualiza cada décima de segundo}
Transmitir velocidades
step() en el Robotarium
Iteraciones +=1
if condiciones de parada ciertas then
  if sólo ha pasado una iteración then
    cuenta +=1
    if cuenta > 90 then
      Finalizar experimento
    end if
  else
    cuenta = 1
  end if
end if
```

Clase Robot

Representa los robots físicos en la simulación. Para crear un robot debemos definir *space*, espacio donde se encuentra; *index*; *pos*, coordenadas iniciales; *r_encl*, radio de la región adjunta centrada en *pos*; th, v, w, orientación y velocidades iniciales. Para acceder a los objetos que existen en *Space* hay un array definido en esta clase donde se encuentran todos, pero esto supone un problema porque en los métodos propios del Robotarium sólo existen los robots, los objetos reales, el resto son imágenes. Esto hace necesario que cada robot creado tenga un índice para indicar su posición en la lista global y otro índice que indique que robot es, estos índices son *index* e *index_rob* respectivamente. Cabe puntualizar que en la lista de objetos siempre van a estar escritos siguiendo este orden, [objetos fijos, robots, objetos móviles]. Evidentemente cada robot va a tener dos variables para las velocidades. Además, cada robot debe tener un alma o varias, es decir, aquello que controla su comportamiento respondiendo al entorno. Son dos, una encargada de las teselaciones y su actualización y otra controlando el movimiento del robot. También deben tener las variables responsables de la visualización, $self.plot$ y $self.label$.

Aparte de la inicialización, existen otras dos funciones: *cmd_vel* y *update*. La primera sirve para indicar el valor que deben tomar las velocidades en la siguiente iteración. La función *update* actualiza el valor de la posición y la orientación con los valores devueltos por el Robotarium y llama a la actualización de las almas.

Clase Obstacle

Define qué consideramos un obstáculo en nuestro entorno. Sólo tiene inicialización, define obstáculos fijos y constituye la base para los obstáculos móviles. Siempre debemos pasarle una lista de vértices para definir el polígono que va a ser nuestro obstáculo, si se quiere iniciar de forma aleatoria se debe utilizar el resultado de la función *randomInit* del archivo *Utilities*. La posición del obstáculo queda definida por el centroide del polígono. Todo obstáculo debe tener un nombre que lo diferencie, y un color para la representación. Dentro de sus variables propias está $self.areaObs$, el área que ocupa el obstáculo más la distancia de seguridad a recortar las teselaciones.

Clase MovObs(Obstacle)

Los parámetros de la inicialización exclusivos son, vel, w, th y *mov_iter*, el último representa el número de iteraciones hasta que el movimiento del objeto se vuelve aleatorio. También se pueden no definir los vértices, se creará un círculo aleatoriamente. Existe una variable propia $self.p$, la probabilidad de cambiar de orientación y velocidad,

y un límite a la velocidad lineal. Al ser un obstáculo móvil necesita una función para actualizar la localización y el control de la velocidad. Cómo ya hemos dicho se pueden producir cambios aleatorios en la dirección, de 45 o -45 grados y en la magnitud puede decrecer, crecer o mantenerse. Para que estos cambios se produzcan deben haber pasado las iteraciones definidas por mov_iter.

Se implementa un método para evitar obstáculos, donde comprueba si sus vecinos no están muy cerca y si lo están cambia su dirección $-\pi/2$ y la normaliza entre $\pi, -\pi$. Después se calcula el incremento en el eje x y en el eje y: $dy = self.vel * dt * np.sin(self.th + dth/2)$, con el coseno si es dx. Si la nueva posición deseada se encuentra cerca de salirse de los bordes del espacio, retrocede y cambia de dirección evitando moverse fuera del espacio. Para finalizar se actualiza la posición, los vértices y el tiempo.

Clase Soul

Esta clase es el esqueleto de las clases que definen la inteligencia del robot. Simplemente define las variables propias de tiempo, periodo, las necesarias para la visualización y añade a la lista de almas del robot una nueva. Su actualización permite la actualización de sus clases inferiores, devolviendo un booleano cierto cuando el tiempo del robot es mayor que el propio más el periodo.

Clase Move(Soul)

Es la responsable del control de los movimientos de un robot. Para inicializarla es necesario definir: un robot, un periodo, una constante del proporcional, el tipo de objeto a detectar como obstáculos, una distancia a la que considera el robot que ha llegado a su destino, el rango de visión, una distancia de seguridad, y un string que indica que hacer cuando se ha llegado al destino, $'keepgoing'$ o $'stop'$. En todos los experimentos sólo existirá la opción de mantener el movimiento, para poder reaccionar antes a los cambios que puedan producirse en la teselación, por eso no hay una configuración final del sistema sino que siempre va cambiando y adaptándose a esos cambios.
La función *cmd_set* modifica el destino (*goal_point*), el tiempo en el que tiene que llegar al destino (*when*), la constante del proporcional (Kp), distancia considerada cerca (*close*) y qué hacer cuando se está cerca (*nw*).

En la actualización se calculan las nuevas velocidades siempre que la clase *Soul* así lo permita. De ser así se comprueba si hay un destino, si no lo hay se mantendrá el movimiento donde la magnitud de la velocidad lineal estará comprendida entre 0.1 y

0.025 m/s. Si hay destino, pero está muy cerca, es cómo si no hubiese y si lo hay se define la distancia al mismo como un $Point2D$, lo que nos permite tener módulo y orientación. A continuación, si hay obstáculo debemos evitarlo, para ello declararemos la distancia al mismo como un vector y su magnitud la modificaremos por $(self.safety - obs_dist.r)/obs_pos.r$, y esta distancia la sumaremos a la distancia al destino. Hecho todo esto calcularemos lo mal orientado que está el robot $misorientation = pipi(distance.a - rob.th)$, luego la velocidad lineal $v = distance.r/max((self.when - self.time+self.T), ss.dt)*max(0, np.cos(misorientation)**3)$, simplemente el módulo de la distancia partido del tiempo en el que tenemos que llegar multiplicado por el coseno del error en el ángulo para saber dirección. El coseno se eleva al cubo para atenuar el valor de la velocidad resultante. La velocidad angular es un control proporcional donde multiplicamos el error en el ángulo por la constante proporcional. Y lo último es comunicarle al robot las velocidades a las que debe ir utilizando la función propia de la clase Robot *rob.cmd_vel (v, w)*.

Clase Voronoid(Soul)

Encargada del cálculo del algoritmo basado en CVT. En su inicialización inicia el alma de movimiento del robot. En su actualización se crea una lista de los vecinos con la función *nearby* propia de *Space*, y para cada vecino guardamos su punto reflejo, que para los obstáculos es dos veces la diferencia al punto más cercano conservando la dirección. Utilizamos este punto porque al calcular la celda se reduce a la mitad por lo tanto si cogemos el punto más cercano la celda estaría muy lejos del obstáculo no consiguiendo un buen comportamiento. Con esta lista aplicamos la función *voronoi* para cada vecino y después se restan los bordes del espacio al polígono resultante de *voronoi*. Para calcular las celdas completas, hacemos lo mismo pero la distancia de seguridad es 0. Para obtener el punto al que se debe dirigir el robot, sólo hay que aplicar el argumento *.centroid* a nuestro polígono y transmitírselo al alma de movimiento mediante $self.Move.cmd\_set(self.destination, self.T)$

Clase Performance

Se encarga de guardar los índices de interés de los experimentos, los cuáles son las iteraciones hechas, las colisiones, la distancia mínima entre cualquier robot y algo, la media de las áreas de las teselaciones, y la desviación estándar del área de las teselaciones. La actualización consiste en adjuntar a los arrays propios creados los valores correspondientes. La media y la desviación estándar se calculan con la librería statistics. Existe otra función que guarda estos índices en archivos externos.

4.3. Puesta a punto

Al principio se observó que los robots reales tenían un movimiento muy brusco, había muchos cambios de dirección y era muy fácil que dejasen de estar en las áreas que se considera has alcanzado tu destino. Por lo tanto se debía ajustar mejor los parámetros que controlan el movimiento. Lo primero fue ajustar la ganancia del controlador proporcional, que al principio se encontraba en 10 y tras varias simulaciones se terminó por ver que debía atenuar la velocidad angular para que las ruedas no saturasen, al final se dejó un valor constante de 0.8. También hubo que ajustar lo que se consideraba una distancia lo suficientemente cercana al objetivo, pues al nunca pararse y mantener el movimiento constantemente dejan de estar en esa posición deseada. Por lo tanto debe ser muy pequeña esa distancia, al final se estableció dividir por 4 el radio adjunto del robot, traducido a centímetros, unos 3. Además al principio se permitía a los robots moverse hasta la velocidad lineal máxima permitida, pero cómo no se pueden combinar velocidades lineales y angulares cercanas a las máximas, debido a la limitación en el giro de las ruedas, se limitó a 0.15m/s la velocidad máxima lineal. Se planteó también aumentar el periodo de control, pues esta velocidad se calcula como distancia partido por el tiempo en el que llegar, pero no se obtuvieron buenos resultados en la limitación de los actuadores hasta que el periodo era muy grande. Por último, la inicialización primero de los obstáculos fijos se debe al posterior posicionamiento inicial de los robots. Como para iniciar el Robotarium se exige iniciar las posiciones de los robots, ya sea pasándolo como comando o aleatoriamente con la función *create_ini_cond*, no es posible tener en cuenta los obstáculos después. Por lo tanto debes definir los obstáculos fijos y pasar sus vértices como argumento en la inicialización del espacio, que también sigue una lógica en el mundo real, pues los obstáculos fijos siempre van a estar en la misma posición y colocados antes de iniciar.

¿Por qué existen diferentes archivos para la simulación y el Robotarium? Las diferencias en el archivo mundo son mínimas: básicamente en las simulaciones no existe condición de parada, además existe la posibilidad de guardar la simulación en vídeo y el tratamiento de la información es diferente entre ambos. La principal diferencia es dónde y cómo se guardan los índices de rendimiento. En el archivo correspondiente a las simulaciones, se utiliza la librería *pandas* lo que posibilita crear Excel donde cada simulación hecha del mismo experimento se guarda en una hoja diferente de ese Excel. Sin embargo, en el destinado al Robotarium se guardan los datos en un archivo de texto, mediante *np.savetxt*(). Los archivos realmente diferentes son aquellos que definen el experimento a realizar, debido a que las simulaciones deben ejecutarse 10 veces seguidas con posiciones iniciales aleatorias para obstáculos fijos y robots. Mientras

que los experimentos reales son sólo una única ejecución donde todo tiene que estar definido y fijo para poder comprobar las diferencias entre la simulación y la realidad.

Capítulo 5

Resultados y análisis

5.1. Resultados

Dentro de este apartado se ha analizado el funcionamiento del algoritmo en sí con simulaciones en el ordenador y su proyección física en el Robotarium. Enlace a la lista de reproducción, *Results Scenarios*, que contiene los vídeos de los experimentos simulados y ejecutados en el entorno real: `https://www.youtube.com/playlist?list=PLpfhyB7Xtf4ZPbdUdJtjEqtpUl2Y25DNP`

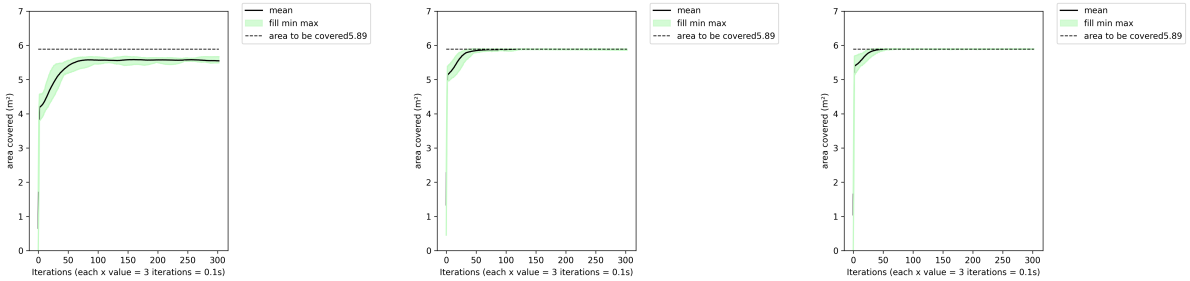
5.1.1. Simulaciones

El objetivo de estas simulaciones era comprobar cómo afectan diversos parámetros al funcionamiento del algoritmo de despliegue creado. Para ello se plantearon tres escenarios: en el primero todo el espacio está libre y simplemente deben ocuparlo guardando la distancia de seguridad con los límites del espacio; el segundo, existen dos obstáculos fijos con forma de círculo y área constante, cuyas posiciones son aleatorias pero siempre en un rango del espacio; por último, una simulación mucho más cercana a una situación real donde los robots se encuentran dentro de una habitación deben ocupar un pasillo y existen obstáculos móviles con movimiento aleatorio. Para los dos primeros escenarios el procedimiento realizado ha sido ir cambiando los parámetros más relevantes y hacer diez iteraciones de la simulación donde se iniciaban aleatoriamente todas las posiciones iniciales. Primero se modifica un parámetro fijando el resto y después se fija ese parámetro al mejor y se cambia otro y así sucesivamente.

Sin obstáculos

Los parámetros que se modifican son: el número de robots, el radio sensorial, la distancia de seguridad para reducir las celdas y la distribución de las posiciones aleatorias de los robots. El primer parámetro modificado es el número de robots porque *a priori* pensé que sería el que más afectaría, los valores son 10, 15 o 20 y el resto están

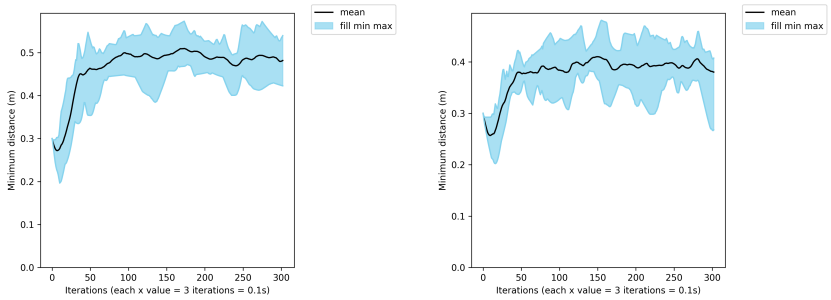
fijados: R=1, safety_radius=0.05 y todo el espacio disponible. El primer indicador de si el algoritmo funciona es comprobar que se cubre todo el espacio disponible. Como podemos ver en la figura 5.1 con 10 robots no se llega a cubrir todo el espacio.



(a) Nº de robots = 10 (b) Nº de robots = 15 (c) Nº de robots = 20

Figura 5.1: Comparación entre la ocupación del espacio con diferente número de robots

Lo siguiente es comprobar cómo de cerca han estado los robots con el resto de objetos, se pueden ver las distancias mínimas en la imagen 5.2. En ninguna configuración se producen colisiones y cómo era lógico en la que hay 15 robots se estabilizan a un valor mayor aunque siempre existe una zona entre la distancia más pequeña y la mayor entre experimentos. Para finalizar hay que fijarse en el tamaño de las áreas, al igual que con la distancia máxima la configuración con más robots tiende a un valor de la media de las áreas menor, pero en ambas sin banda de valores. Y en la desviación estándar tiende a un valor más pequeño, pero con un aspecto muy parecido a la gráfica con 15 robots. Consecuentemente decidí que la mejor opción era utilizar 15 robots pues la distancia mínima es mayor cuando el experimento se estabiliza y para que utilizar 5 robots más si ya se cubren todas las necesidades.



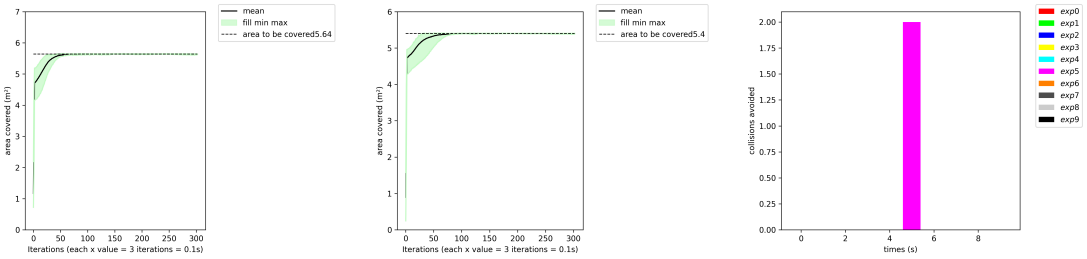
(a) N = 15 (b) N = 20

Figura 5.2: Mínima distancia de un robot a algo con N = 15, 20

El siguiente parámetro que se modificó fue el radio sensorial, probamos con la mitad del radio anterior, es decir 0.5m, para 15 y 20 robots y en ambos casos el radio es tan pequeño que imposibilita ver todo el espacio, el máximo valor sumando todas las áreas de los círculos de las celdas son 3.93 m^2 respecto a los 5.89 m^2 a cubrir. También se

probó con el doble de radio para 10 robots el funcionamiento es correcto incluso tarda menos en cumplir las condiciones de parada que la mejor opción con 15 robots, pero se descartó por ser demasiado sencillas las condiciones iniciales, pues un radio sensorial de 2 metros supone ver más de la mitad del espacio disponible para cada robot.

El siguiente parámetro para modificar era la distancia de seguridad en las celdas de Voronoi, quedan fijos el número de robots a 15 y el radio sensorial a 1. Se probaron con distancias mayores, pues hasta ahora se había utilizado la mitad de las dimensiones de los robots, 5cm. Se probó con 7.5cm y con 10cm y no se aprecia una mejora del rendimiento, si acaso se empeora porque se tarda más en cubrir el área total, figuras 5.3a y 5.3b, y se llegan a producir dos colisiones con 7.5cm 5.3c. Estas colisiones se deben a unas posiciones iniciales cercanas y enfrentadas, pero no al funcionamiento del algoritmo una vez se pasa esos instantes iniciales.



(a) Área vista $safety = 7{,}5cm$ (b) Área vista $safety = 10cm$ (c) Colisiones $safety = 7{,}5cm$

Figura 5.3: Resultados de los experimentos con distancias de seguridad mayores

Por último cambiamos el tamaño y la posición de las posiciones iniciales del robot. Hasta ahora los robots podían iniciar el experimento en cualquier punto del espacio disponible guardando siempre una distancia con sus vecinos. En la primera configuración limitamos el tamaño disponible a un rectángulo de 2 metros de ancho y 0.9 metros de alto. En la figura 5.4 cada columna representa una configuración: izquierda, el rectángulo está en el centro; en medio, el rectángulo se encuentra en una izquierda del espacio; derecha todo el espacio disponible. He de señalar que para las dos configuraciones con espacio reducido, el valor de $safety\_radius$ es 7.5cm mientras que en el otro es 5cm. En la primera fila de la figura podemos observar la media de las áreas de las teselaciones, y observamos cómo en las dos primeras el valor inicial es menor y cómo tardan más en llegar a su valor estable. Este valor es más pequeño por la diferencia en la distancia de seguridad. La siguiente fila corresponde con la mínima distancia entre un robot y algo, aquí es sorprendente ver cómo la distancia mínima también se estabiliza en un valor menor que para la configuración de la derecha, aunque tiene más ruido, la zona en la que se encuentran los valores es mayor. Este hecho sorprendente no lo es tanto, porque se relaciona con la distancia de seguridad pues con los límites del espacio

es este valor el que se guarda, por lo tanto el espacio global se reduce también haciendo que los robots al final estén más cerca.

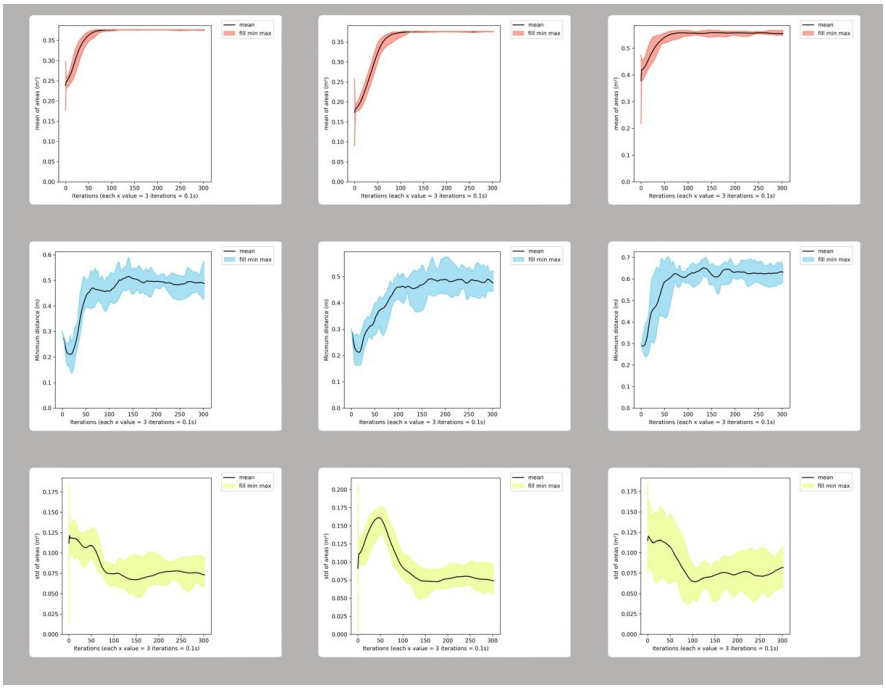


Figura 5.4: Resultados para las diferentes configuraciones para las posiciones iniciales.

Y por último tenemos la desviación estándar donde vemos cómo columna izquierda y columna derecha se parecen mucho, aunque la primera tiene menos zona. Sin embargo, la gráfica interesante es la central donde vemos un pico muy definido al inicio de la simulación, aunque en el resto de las gráficas no se aprecia esa diferencia de áreas. Esto se debe a que al comenzar en una esquina del espacio, el peso de explorar recaerá en los robots que se encuentren en los bordes interiores, los cuáles tendrán una dirección donde nada corte con su celda y por lo tanto tengan mucha área 5.5, sin embargo el resto tendrán áreas muy pequeñas, por lo que en media se compensan y cómo realmente los robots están próximos las distancias entre ellos pueden ser pequeñas.

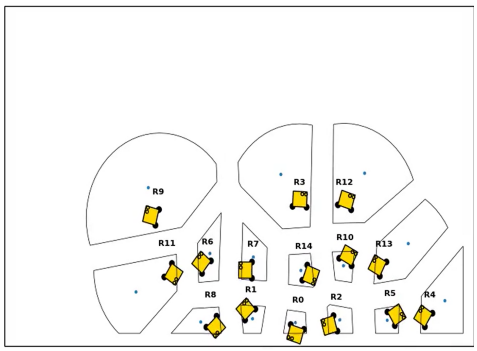
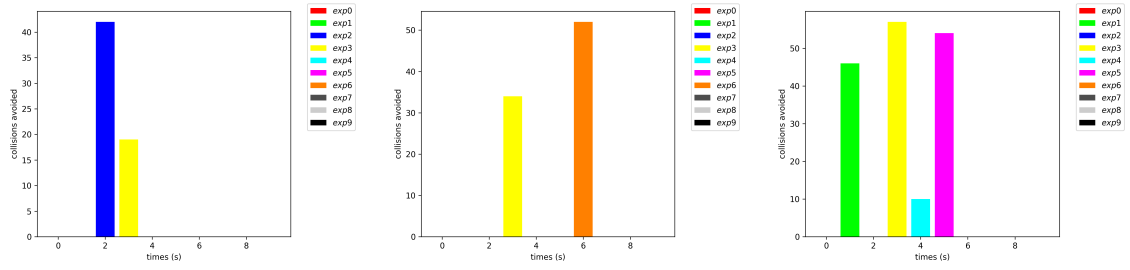


Figura 5.5: Instante inicial con la configuración en esquina

Con obstáculos fijos

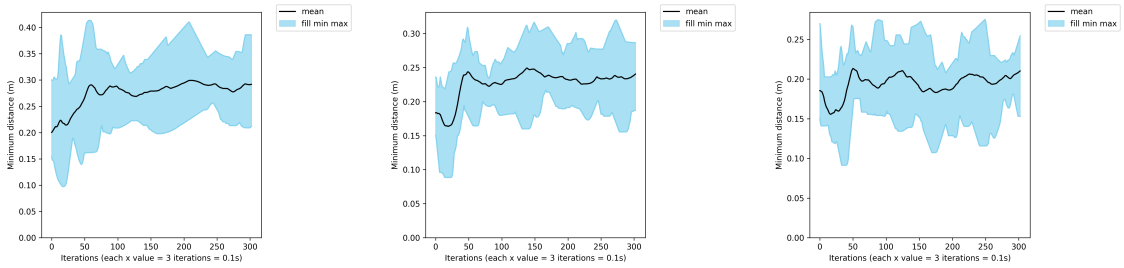
El procedimiento es igual que para el escenario sin obstáculos, los parámetros a modificar son los mismos a excepción de la distribución espacial que ahora siempre va a ser todo el espacio disponible. El primer parámetro que se comprueba su incidencia es el número de robots, ahora que el espacio disponible es menor y hay más probabilidades de que se pase cerca de un objeto. Lo primero a destacar es que fijando $R = 1$ y $safety = 0{,}05$, con 10 robots vuelve a no llegar a ocupar todo el espacio posible. Sin embargo ahora las tres posibilidades causan colisiones, cómo podemos ver en la figura 5.6. Una colisión puede ser que entre la protección frente a los choques propia del Robotarium o que un robot esté a menos de 12 cm de otro objeto, en ningún caso corresponde a colisiones verdaderas donde dos objetos estén superpuestos. Para saber a cuál de las dos opciones necesitamos mirar las gráficas de distancia mínima 5.7. Observamos cómo en las tres en alguna simulación se produce un posicionamiento inicial que hace que en los primeros instantes se de alguna iteración donde dos robots se encuentran a menos de 12 cm, pero la mayoría se deben a esa segunda protección, siendo mucho más frecuente en la tercera opción donde la media se encuentra por debajo de esa distancia de 20 cm. Con 15 robots es cierto que las distancias más pequeñas en el rango son menores de 20 cm, pero observamos que la media se encuentra por encima, lo que se corresponde con que haya colisiones sólo en dos simulaciones de las 10. En la primera se ve que tanto la media como los valores mínimos tienden a estar por encima de 20 cm, lo que se ve también en las colisiones donde las barras son más pequeñas. Esto se puede deber a unas posiciones iniciales muy desfavorables o que la posición de los obstáculos impida acceder a una parte del espacio o estén muy cerca y sea imposible pasar entre ambos. Es evidente que la mejor opción es utilizar solo una quincena, pues menos no cubren toda el área y más provocan colisiones más frecuentemente.



(a) Colisiones con $N = 10$ (b) Colisiones con $N = 15$ (c) Colisiones con $N = 20$

Figura 5.6: Situaciones de peligro de colisión con diferente número de robots

El siguiente experimento era probar con 20 robots pero un radio sensorial dividido entre 2 y funciona horrible, no llega a ocupar todo el suelo y además provoca que los robots siempre estén muy cerca de los otros, lo que ocasiona muchas más colisiones



(a) Distancia mínima $N = 10$ (b) Distancia mínima $N = 15$ (c) Distancia mínima $N = 20$

Figura 5.7: Distancia mínima entre un robot y algo para 10, 15 y 20 robots

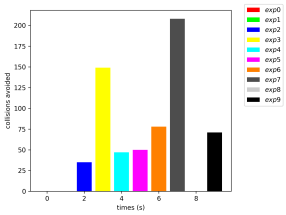


Figura 5.8: Colisiones producidas con 20 robots y un radio sensorial de 0.5m

como podemos ver en la imagen 5.8. Después se probó con el doble de radio sensorial, para 15 robots se vio cómo se producían colisiones en más iteraciones, por lo que descartó esta opción. Sin embargo para $N = 10$, el aumento del radio si que hace que sea una opción a tener en cuenta. Por ello vamos a comparar en la imagen 5.9 las dos situaciones, 10 robots con R=2 o 15 robots con R=1.

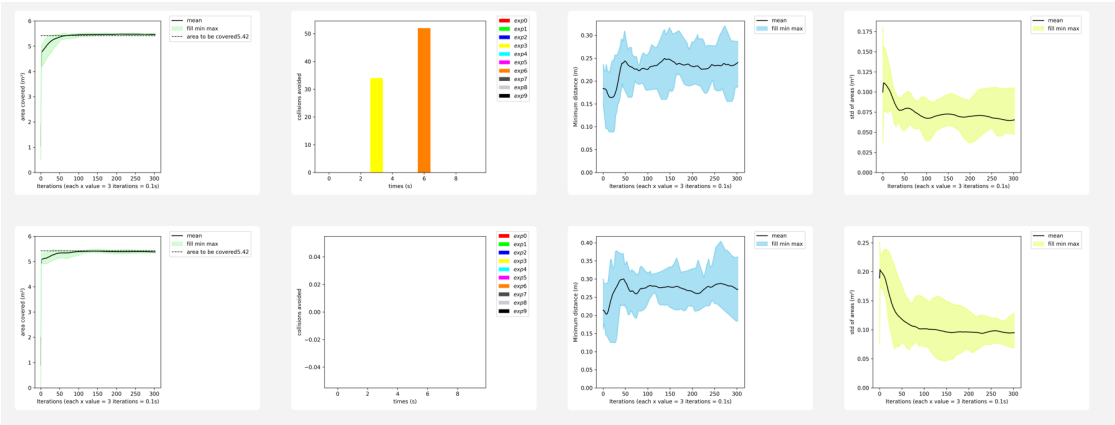


Figura 5.9: Comparación entre N=15 y R=1 (arriba) o N=10 y R=2 (abajo)

Se observa como con menos robots se evitan las colisiones y la distancia mínima es mayor, pero de forma muy sutil se puede no cubrir todo el área disponible y la diferencia entre las áreas es mayor, al tener una desviación estándar mayor. Pero como el área mínima que se cubre es muy próxima a la disponible y la diferencia en la desviación tampoco es muy grande, la mejor configuración sería con 10 robots y más rango, pues evita totalmente las colisiones. Sin embargo antes que ya hemos comentado que es

una situación muy poco real y muy ventajosa pues observan gran casi la totalidad del espacio a cubrir. De esta condición de irrealidad surge la pregunta, ¿hay alguna manera de evitar las colisiones utilizando 15 robots y un radio de 1 metro? La respuesta a esta pregunta se encuentra en el parámetro que quedaba por modificar, la distancia de seguridad de las teselaciones. Y es que la función de este valor es precisamente evitar las colisiones, por lo tanto aumentándolo evitaremos acercarnos entre objetos. Se probó con 7.5cm y 10 cm. Ambas configuraciones consiguen evitar las colisiones, y el resto de parámetros no tienen mucha diferencia con los ya vistos para 5cm. Por lo tanto la mejor configuración posible de las comprobadas es: N=15, R=1m, safety=7.5cm y la distribución inicial por todo el espacio.

Simulación final

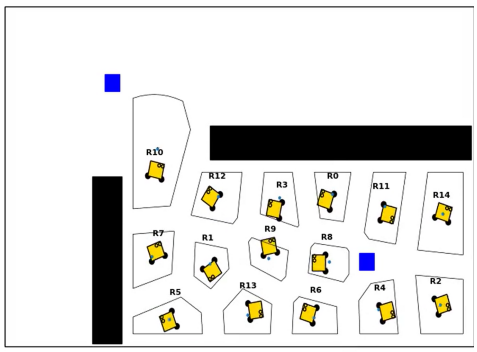


Figura 5.10: Posición inicial para el experimento final

El experimento final empieza con la disposición que vemos en 5.10. En ella los robots comienzan en la habitación definida por dos paredes, rectángulos negros, y los límites del espacio. Y deben salir al pasillo, resto del espacio libre, evitando los obstáculos móviles, cuadrados azules, que simulan el movimiento aleatorio de lo que podría ser una persona, aunque es un movimiento bastante más aleatorio. Los parámetros utilizados son los de la mejor configuración obtenida en simulaciones, es decir, N=15, R=1m y safety=7.5cm. Los resultados son que se logra el objetivo deseado de salir al pasillo y ocupar todo el espacio disponible, pero se producen muchas colisiones, o mejor definido, situaciones de peligro de colisión cómo podemos observar en 5.11. Quiero hacer hincapié en que las colisiones no detectan si el borde de dos robots se toca o si se toca con un obstáculo, sino las distancias entre las dos posiciones de los robots o con el punto más cercano del obstáculo. Lo cual genera problemas, porque las posiciones del Robotarium no se definen en el centro del robot, sino en el centro del eje de las dos ruedas, por eso dos robots enfrentados y a menos de 18 centímetros realmente chocarán pero nosotros no lo contaremos cómo tal, sin embargo a 12 centímetros pero en direcciones opuestas

no están cerca de chocar sus extremos pero si se contará cómo situación peligrosa y por tanto como impacto. Además está definición de la posición de los robots, hace que la distancia de seguridad de 7.5cm que se guarda con los obstáculos no sea suficiente pues se impacta con la pared si están a unos 10cm y el robot va frontalmente contra el obstáculo. Por lo que se probó a utilizar una distancia de 10 cm para aumentar la seguridad y evitar todos los choques, pero esta implementación causaba que en todas las simulaciones se produjesen errores donde la celda de Voronoi pasaba a ser negativa, lo que implicaba un error que paraba la ejecución.

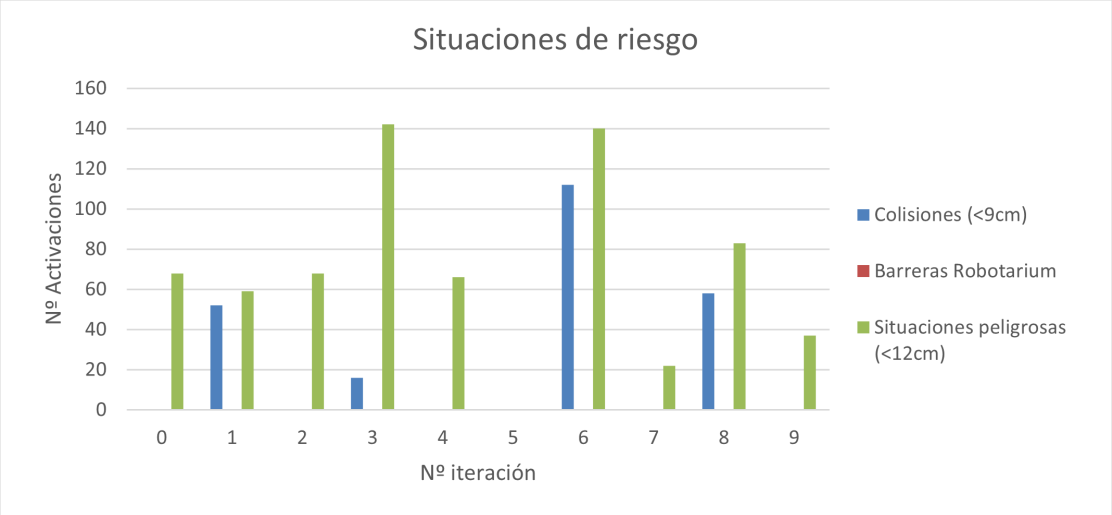
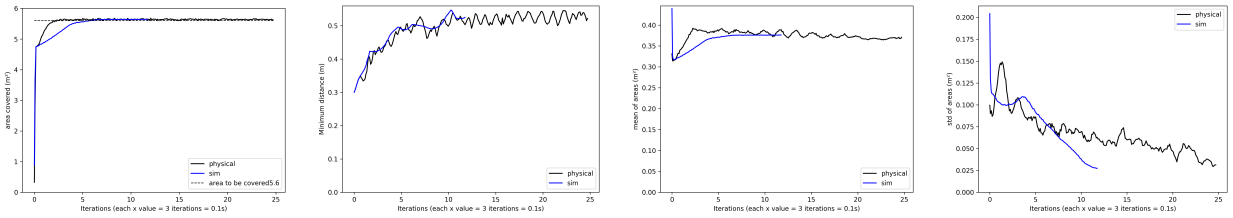


Figura 5.11: Colisiones provocadas en el experimento final

5.1.2. Comparación con experimento real

Sin obstáculos

La única diferencia con las simulaciones es que ahora todo está fijado, las posiciones iniciales y los tiempos de control de cada robot, para que no existiesen diferencias entre la simulación y el experimento real debidas a esa aleatoriedad. Vídeos 1 y 2 de la lista de reproducción del capítulo.



(a) Área cubierta m^2 (b) Distancia mínima m (c) Media de áreas m^2 (d) DS de áreas m^2

Figura 5.12: Comparación entre la simulación y la realidad para el escenario sin obstáculos

En la figura 5.12 podemos observar cómo la principal diferencia es el tiempo que tarda en realizarse el experimento, en la simulación tan sólo eran 12 segundos mientras en la aplicación física son 25. Esto es debido a la desviación estándar donde se observa que se producen muchos más picos y se tarda más en llegar y mantener el valor deseado, por debajo de 0.04, no cumpliendo la condición de finalización. El resto del comportamiento se observa cómo es bastante similar, aunque se despliega por toda la superficie antes y en la media tiene una progresión mucho más rápida y posteriormente va disminuyendo hacia el valor alcanzado en la simulación, esto nos indica que al principio existen áreas grandes y después se van reduciendo a ser todas un valor semejante, cómo también indica la desviación estándar. Los picos de sierra en la distancia mínima reflejan ese movimiento que tienen los robots una vez se alcanza el despliegue, cuando ya alcanzan la posición deseada tienen que volver a cambiar de orientación porque en la siguiente iteración ya se han salido de la región cercana.

Con obstáculos fijos

La situación inicial del experimento queda reflejada en la imagen 5.13. Queda claro que la parte derecha es la más libre y, por lo tanto, serán los robots más cercanos a ella los que tengan más movimiento y más área al principio. Cómo pasaba anteriormente el experimento real tarda mucho más que la simulación en este caso siendo mucho más relevante pues el real nunca llega a cumplir la condición de parada referente a la homogeneidad de las áreas. Volvemos a ver lo ya comentado en el escenario anterior, con

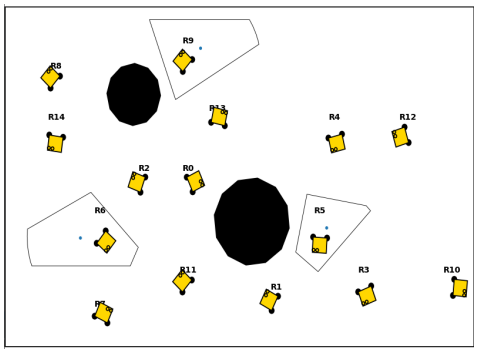
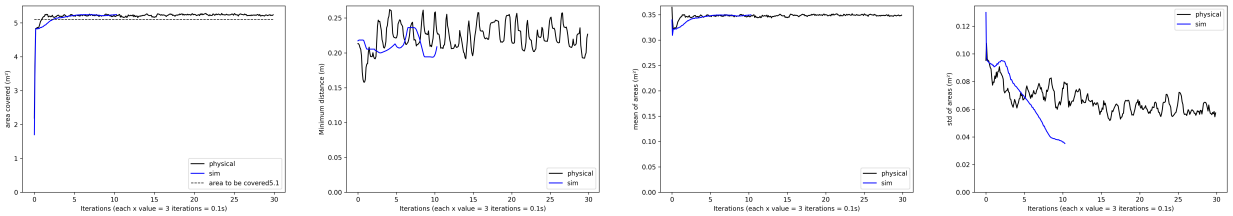


Figura 5.13: Disposición inicial del experimento con obstáculos fijos.

la ligera modificación que ahora los picos en la mínima distancia son más pronunciados, sucede lo mismo que antes sólo que ahora están siempre más cerca y por tanto ese movimiento al superar la zona considerada destino, es mayor en comparación con el tamaño de las celdas. Los vídeos correspondientes son el 3 (*Fixed Obstacles Simulation*) y el 4 (*Fixed Obstacles Physical Experiment*).

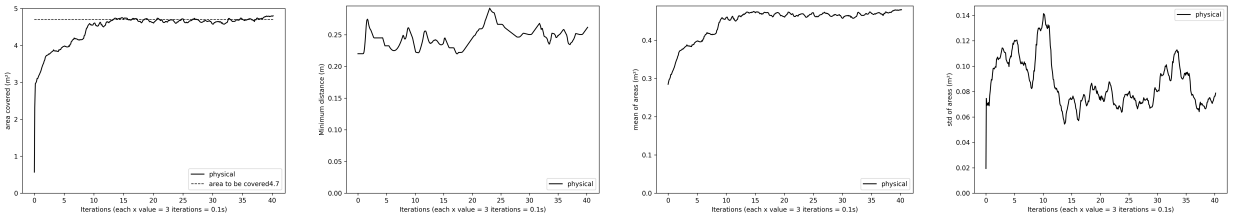


(a) Área cubierta m^2 (b) Distancia mínima m (c) Media de áreas m^2 (d) DS de áreas m^2

Figura 5.14: Comparación entre simulación y realidad para el escenario con obstáculos

Simulación final

Este experimento cambia entre el ordenador y el experimento físico, debido a un problema que comentaré en la sección de análisis de este capítulo. Las diferencias son el número de robots y las posiciones iniciales, el experimento real pasa a tener 10 robots y dos de ellos estarán en los finales de los pasillos. Este escenario es beneficioso porque no se acercan tanto los obstáculos móviles o tienen más formas por donde escapar de ellos los robots, por lo tanto no se producen colisiones y la distancia entre objetos es bastante estable y con seguridad pues no baja de 20 centímetros y la media serían unos 24. Este experimento nos demuestra cómo en una situación más parecida a la realidad, el algoritmo sigue cumpliendo su objetivo principal, desplegarse por todo el espacio evitando el resto de objetos. Los vídeos correspondientes son: el quinto; el sexto, donde podemos observar el error al utilizar 15 robots; y el séptimo, donde refleja la configuración final.



(a) Área cubierta m^2 (b) Distancia mínima m (c) Media de áreas m^2 (d) DS de áreas m^2

Figura 5.15: Resultados del experimento físico final

5.2. Análisis

El comportamiento del algoritmo se comprueba y ajusta en las simulaciones. En el escenario vacío los parámetros más cruciales para conseguir un correcto funcionamiento son el número de robots y el radio sensorial, lógico si pensamos que son los que definen el área que seremos capaces de cubrir, en una situación donde los robots sean capaces de comunicarse por casi todo el espacio, necesitaremos muy pocos robots para cubrir toda el área. Sin embargo, teniendo mucho más limitada la comunicación necesitaremos más robots, llegando incluso a no ser capaces de cubrir toda el área. Aún así el parámetro más influyente en el comportamiento es la posición inicial, debido a que ella define cuanto espacio queda por cubrir, no es lo mismo empezar en una esquina todos juntos, que distribuidos por todo el espacio donde los movimientos a realizar serán menos y más pequeños. El escenario donde existen obstáculos fijos confirma lo visto en el anterior, aunque ahora la posición inicial tiene menos margen al tener que estar los robots a una distancia segura de esos obstáculos. Pero, introduce la necesidad del tercer parámetro, la distancia de seguridad para las teselaciones, es decir cuánto margen dejamos entre celda y celda libre para evitar colisiones. Este parámetro es crucial en la realidad pues una colisión entre dos robots físicos supondría la pérdida de ambos y posiblemente un funcionamiento erróneo del resto. Por último, en el escenario realista vemos las limitaciones del algoritmo al ponerlo frente a una situación mucho más exigente. Vemos cómo el número de situaciones peligrosas aumenta e incluso existen colisiones en varios de los experimentos, aunque nunca entre robots y siempre entre robot y obstáculo. Estas situaciones únicamente se producen por la limitación en el tamaño y por el tamaño en sí del Robot. Al tener que limitar el espacio al existente en la realidad, estamos limitando el espacio a un poco más de 6 m^2, por lo tanto, las teselaciones nunca serán superiores a 0.4 m^2 para 15 robots. Esta limitación en las áreas afecta principalmente a la distancia que podemos aplicar para evitar colisiones pues si ponemos 12cm, que significaría que cabe un robot entero en ese espacio, reducimos a más de la mitad el tamaño de la teselación y por tanto el movimiento de los robots.

Adicionalmente existe el problema de que los robots para esa área serían muy grandes, implicando que cualquier movimiento conllevaría salir de la zona definida cercana al destino. Para finalizar, hay que resaltar la importancia del movimiento continuo de los robots, aportando dinamismo al algoritmo así cómo adaptabilidad y capacidad de reacción ante cambios en el entorno.

En los experimentos reales vemos muy clara la problemática del tamaño reducido, una vez se consigue la configuración más o menos estable y por lo tanto ya sólo queda realizar movimientos pequeños, los robots físicos se pasan enseguida de la zona destino y por lo tanto tienen que girarse sobre sí mismos porque ahora el destino está detrás suyo, afectados también por la inercia que en el simulador no existe y en la realidad hace que los cambios entre velocidades y orientaciones sean más lentos. Sin embargo, esto supone que una vez se haya llegado a esa configuración deseada se produzcan menos cambios, pues los movimientos se limitan a entrar y salir de la zona deseada, mientras en las simulaciones se podía dar que hubiese cambios muy lentos por una orientación parecida de varios robots, que favorece el continuar con el movimiento en línea recta.

Por último, hay que comentar la problemática intrínseca al uso del Robotarium cómo banco de pruebas y la responsable de tener que cambiar el experimento final por uno con menos robots. Así como en la simulación es posible establecer las condiciones iniciales de forma libre, siempre que estén alejados los robots entre ellos, en la realidad no. Todos los experimentos empiezan con los móviles fuera de los límites del espacio. Al empezar, éstos tienen que ir a unas posiciones aleatorias definidas por el método *generate_inital_conditions*, pero distintas de las condiciones iniciales definidas por el usuario. Es evidente que va a ser necesario moverse desde esas posiciones aleatorias a las posiciones iniciales establecidas por el usuario. Hasta aquí nada parece un problema, pero ese movimiento se realiza por los controladores propios del Robotarium, con los valores predefinidos y es imposible para el usuario saber previamente a la devolución de la ejecución si este movimiento ha sido satisfactorio o no. Por esto aunque era posible la creación de las posiciones iniciales en la simulación para 15 robots dentro de la habitación, era imposible que el sistema físico las alcanzase porque se activaban las barreras de colisión muy pronto y esto suponía un movimiento por fuera de los límites, especialmente del inferior, y esto implica que se pare el experimento porque el Robotarium considera esta posición como un error prioritario y finaliza la ejecución. En consecuencia nuestro experimento ni siquiera llegaba a empezar a ejecutar el algoritmo de este TFG, y la ejecución finalizaba. Con el añadido que la notificación recibida por correo era que el experimento había sido un éxito, cuando ni siquiera se había empezado a ejecutar.

Capítulo 6

Conclusiones

Podemos concluir que el algoritmo de despliegue basado en teselaciones de Voronoi, con las modificaciones hechas, es una buena y robusta opción para las aplicaciones de cobertura de un espacio por un enjambre robótico. Llegamos a esta conclusión porque si en las aplicaciones explicadas funciona, con unas condiciones desfavorables: robots no holónomos, con restricciones de velocidad importantes y muy grandes respecto a la distancia que guardan entre sí, en aplicaciones realistas donde el espacio se mucho mayor y por lo tanto con unas distancias más lejanas funcionará de forma óptima.

Respecto a la utilización del Robotarium y siempre hablando desde mi experiencia considero que es una herramienta útil, pero limitada y con algunas funciones mejorables. Me parece una herramienta beneficiosa si se utiliza de una forma adecuada y teniendo en cuenta qué necesita para que el funcionamiento sea correcto. Posibilita probar tu código en un sistema real, lo que puede ser muy útil para comprobar el funcionamiento de controladores, de sistemas pequeños o de pequeñas porciones de experimentos mayores. Además de no tener que esperar un tiempo excesivo entre que se entrega un experimento y se devuelve su ejecución, unos dos días laborables en fines de semana no funciona. Considero que la experiencia sería mucho más beneficiosa si se mejorasen dos aspectos fundamentales. El primero, la posibilidad de controlar o visualizar en el simulador ese movimiento inicial entre las posiciones aleatorias y las definidas por el usuario. Por ejemplo, una solución muy sencilla sería que los robots siempre comenzasen en una misma posición y existiese un método para que de uno en uno fuesen yendo a la posición en orden por criterios cómo máxima distancia o más cruces en el movimiento con otros robots. El otro aspecto por mejorar es la retroalimentación del experimento por correo, que la única comprobación del funcionamiento del experimento sea si se ha ejecutado en menos tiempo que el que se estableció en la entrega, no aporta información. Sería mejor un mensaje que sólo informase de que ya están tus resultados o una forma de comparación entre la posición final, algún parámetro o una comparación con un vídeo obtenidos en la simulación.

Como conclusión algunas recomendaciones para utilizar el Robotarium. Comprueba que tu ley de control de movimiento no cree muchos movimientos que combinen velocidades angulares con velocidades lineales, saturarán los controladores y no conseguirás una buena ejecución. Siempre ten en cuenta que si un robot se sale de los límites se terminará la ejecución, puede ser útil reducir el espacio efectivo de tus experimentos. Si quieres hacer una pregunta específica defínela muy bien, tardan en responder y no siempre se ajustan a lo que has preguntado. Siempre pon tiempo de más en los experimentos reales, aunque el simulador especifique que tu experimento va a tardar un tiempo determinado, es orientativo puede funcionar bien y aun así tardar más. Ten en cuenta que si se cumple el funcionamiento se finalizará el experimento da igual cuanto tiempo lleve en ejecución, pero si ajustas mucho el tiempo y por algún motivo aún no ha llegado se cortará y tendrás que volver a entregar el experimento y esto te retrasará. La última recomendación es que compruebes cómo afectará los métodos de protección de barrera en tu código, para ser consciente de que puede haber cambios en el movimiento sin tener tú el control.

Capítulo 7

Bibliografía

[1] M. Brambilla, E. Ferrante, and M. Birattari. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intell*, 7:1–41, 2013.

[2] 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.

[3] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM REVIEW*, 41(4):637–676, 1999.

[4] Min Cao and Christoforos Hadjicostis. Distributed algorithms for voronoi diagrams and applications in ad-hoc networks. 2003.

[5] Yuan Song, Bing Wang, Zhijie Shi, Krishna R. Pattipati, and Shalabh Gupta. Distributed algorithms for energy-efficient even self-deployment in mobile sensor networks. *IEEE Transactions on Mobile Computing*, 13(5):1035–1047, 2014.

[6] Enrique Teruel, Rosario Aragues, and Gonzalo López-Nicolás. A distributed robot swarm control for dynamic region coverage. *Robotics and Autonomous Systems*, 119:51–63, 2019.

[7] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[8] Enrique Teruel, Rosario Aragues, and Gonzalo López-Nicolás. A practical method to cover evenly a dynamic region with a swarm. *IEEE Robotics and Automation Letters*, 6(2):1359–1366, 2021.

[9] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. pages 1699–1706, 2017.

Lista de Figuras