ANEXOS

ANEXO I – Estado del arte

1. Introducción

Desde sus inicios, la robótica y la automática han venido desarrollando equipos dotados de la mayor autonomía posible que han permitido liberar a las personas de tareas físicamente costosas, peligrosas y repetitivas e incluso liberarles en procesos de toma de decisiones.

En un primer momento, su campo de aplicación estaba limitado a la industria, en donde su principal objetivo era obtener una producción eficiente y eficaz a la vez que ofrecía una mayor calidad de vida en el empleo. Con el paso de los años su aplicación se ha extendido a otros sectores logrando la liberación de funciones del ser humano en su propio hogar, con aspiradoras autónomas o sistemas de conducción asistida, mejorando la calidad de vida de las personas.

Actualmente, la robótica avanza hacia el desarrollo de los denominados sistemas multi-robot, sistemas formados por múltiples robots móviles autónomos que tienen un objetivo común. Estos sistemas presentan ventajas respecto a los sistemas mono-robot, las más destacables son:

- Incremento de la capacidad de trabajo: la unión de varios robots permite llevar a cabo tareas que con uno solo serían muy difíciles de ejecutar.
- Rapidez: permiten explotar el paralelismo, reduciendo el tiempo de ejecución.
- Robustez: se incrementa la tolerancia del sistema ante posibles fallos individuales.
- Flexibilidad: los sistemas multi-robot pueden reconfigurarse y adaptarse muy fácilmente a diferentes tipos de entornos y tareas. Además, se incrementan las posibles maneras de combinar las habilidades de cada uno de ellos para llevar a cabo una misma tarea.
- Reducción de costes: tener múltiples robots simples suele resultar más económico que tener un único robot complejo

Las ventajas de los sistemas multi-robot únicamente se pondrán de manifiesto si se implementan mecanismos de cooperación apropiados. Para ello se han de resolver, entre otros, tres problemas básicos:

- Planificación: consiste en dividir la tarea inicial en subtareas que puedan ser ejecutadas por un único robot o un subconjunto de ellos, decidiendo también en qué orden se van a llevar a cabo estas.
- Asignación de tareas: se decide qué robot o subconjunto de ellos ha de ejecutar cada una de las tareas.
- Coordinación del movimiento: cada robot ha de decidir qué movimientos realizar para llevar la tarea que tiene asignada de la mejor manera posible sin entorpecer la ejecución del resto de robots.

La utilización de equipos de robots que cooperan para la realización de tareas ha adquirido un importante auge en la investigación y se ha incrementado el abanico de posibles aplicaciones. Cuando se trata de trabajar en un espacio amplio para realizar una misión, la utilización de múltiples robots coordinados permite aumentar la eficacia y eficiencia de la operación.

Una posible aplicación que se presenta en este proyecto es la de coordinar un sistema multi-robot para realizar actividades de búsqueda y neutralización de amenazas en entornos conocidos con obstáculos.

2. Pursuit-evasion problems

Tradicionalmente, los problemas de búsqueda y captura han sido tratados desde dos perspectivas muy distintas. Una busca diseñar estrategias que aseguren la captura en un tiempo finito frente a adversarios mucho más potentes que los perseguidores, lo que genera soluciones demasiado conservadoras para su aplicación práctica. La otra hace un enfoque más probabilístico, asumiendo, por ejemplo, conocimiento sobre el comportamiento de los evasores o su localización.

En la robótica, los juegos de búsqueda y captura son usados para estudiar problemas de planificación del movimiento que pueden tener aplicación inmediata, por ejemplo, en técnicas de rescate, tareas de vigilancia u operaciones militares. Al margen de la robótica, su estudio también genera interés en campos de lo más diversos, como el modelado del comportamiento animal o la seguridad informática. Como consecuencia, las investigaciones llevadas a cabo han estudiado versiones muy diversas de estos juegos. La Imagen 39 muestra una taxonomía parcial del juego de persecución-evasión.

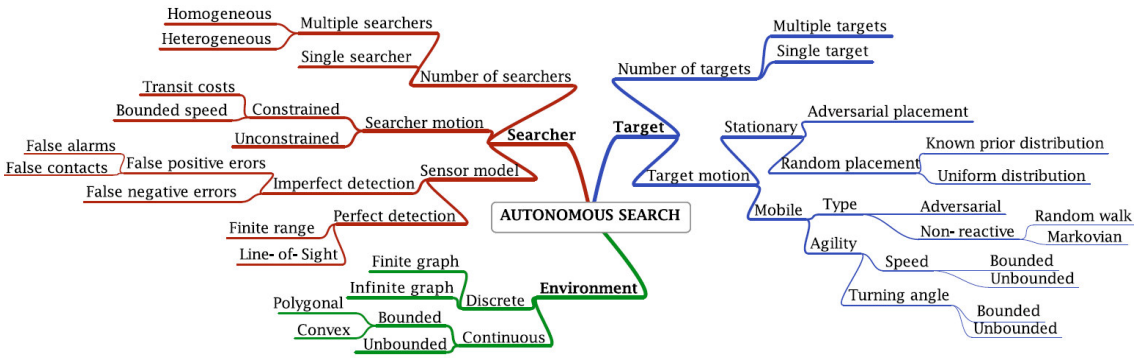


Imagen 39 - Espacio de parámetros para modelos de búsqueda autónoma.

En la literatura de la robótica, la resolución de estos problemas se puede plantear desde dos enfoques distintos, bien mediante una descripción con ecuaciones diferenciales o bien usando técnicas combinacionales. El enfoque combinacional consiste en hacer una representación del entorno, ya sea geométrica o mediante un grafo, y usar esta representación para la resolución del problema. Dado que este último permite tratamientos más sencillos con un menor coste computacional, es el empleado en este proyecto.

Para el tratamiento combinacional, el primer paso es obtener una representación del entorno, la cual puede ser una representación geométrica o una representación en forma de grafo, y en función de ello el tratamiento del problema será distinto. No obstante, como es el caso de este proyecto, uno puede comenzar con una representación geométrica para posteriormente elaborar a partir de ella un grafo desde el cual resolver el problema en cuestión.

En [3] se plantean diferentes juegos de búsqueda y captura en función de la representación llevada a cabo del entorno y, para cada uno de ellos, se hace una descripción de las soluciones aportadas en diferentes artículos de investigación. También se discute la implementación de estas técnicas en robots móviles, así como las nuevas líneas de investigación en este campo.

3. Modelado del entorno para la generación de trayectorias

La planificación de trayectorias seguras y eficientes en robots móviles autónomos juega un papel fundamental a la hora de hacerlos interactuar con su alrededor. La representación del entorno a partir de la información facilitada por sensores de abordo o sistemas auxiliares es fundamental en cualquier algoritmo de planificación de caminos.

Una clasificación de estos algoritmos, presentada en [4] y [5], reside en si estos se basan en el uso de cuadrículas (uniformes o no uniformes), la descomposición exacta de celdas, roadmaps probabilísticos, grafos de visibilidad o diagramas de Voronoi, pudiendo englobar los dos últimos en los métodos de roadmaps.

Los métodos roadmaps convierten el entorno multidimensional en una red de curvas unidimensionales, llamadas roadmaps, que se encuentran en una zona libre de obstáculos. Las técnicas más utilizadas son los grafos de visibilidad (Imagen 40) y los diagramas de Voronoi (Imagen 41).

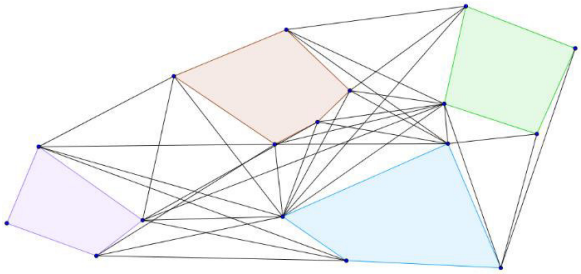


Imagen 40 - Grafo de visibilidad.

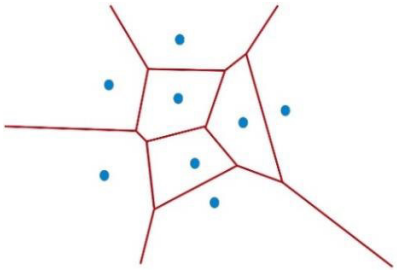


Imagen 41 - Diagrama de Voronoi.

Los grafos de visibilidad son aquellos en los que cada nodo representa un vértice de los polígonos y cada arista una conexión entre dos nodos visibles entre sí. Una vez generado este grafo, un algoritmo de búsqueda generará los vértices que conforman la secuencia a seguir para conectar un punto inicial con un punto final de forma óptima. Con este método el camino resultante suele pasar muy cerca de los obstáculos, ya que los puntos de paso del camino serán vértices de los obstáculos. En contraposición, los diagramas de Voronoi, empleados en este proyecto, generan un grafo en el que el camino resultante se encuentra lo más alejado posible de los obstáculos.

4. Algoritmos de búsqueda

Los algoritmos de búsqueda están diseñados para localizar un elemento con ciertas propiedades dentro de una estructura de datos. Cuando el sistema agente (en este caso, el robot) posee algún tipo de información del medio, se utilizan técnicas de búsquedas informadas; sin embargo, si carece de conocimiento alguno, se deberán emplear algoritmos de búsqueda no informadas que, por lo general, son más ineficientes en tiempo y memoria que los primeros.

En cuanto a los algoritmos de búsqueda informada, destacan los siguientes: Dijkstra, Bellman-Ford, A*, D*, Deep-Firs Search (búsqueda en profundidad) y Breadth-Firs Search (búsqueda en amplitud).

En este proyecto se utiliza el algoritmo de Dijkstra para calcular el camino de mínima distancia entre dos nodos de un grafo. Este algoritmo cuenta con una gran eficiencia para grafos de puntos reducidos.

5. Asignación de tareas

El problema de asignación consiste en asignar ciertos recursos para la realización de determinadas tareas al menor coste, suponiendo que cada recurso se destina a una sola tarea, y que cada tarea es ejecutada por uno solo de los recursos.

En los problemas de búsqueda y captura, los agentes perseguidores son considerados como recursos mientras que los agentes evasores son considerados como tareas. Dado que en este tipo de problemas una tarea puede quedar desatendida temporalmente o tener asignada más de un recurso, no se trata de un problema de asignación sino de un problema de transporte.

El problema de transporte es un problema de programación lineal en el cual se debe minimizar el coste de abastecimiento a una serie de puntos demanda (agentes evasores) a partir de un grupo de puntos de oferta (agentes perseguidores), teniendo en cuenta los distintos precios de envío (distancias) de cada punto de oferta a cada punto de demanda. Por tanto, el problema de asignación es un caso particular del problema de transporte, en el que la oferta en cada origen y la demanda en cada destino son ambas de valor unitario.

Mientras que para los problemas de asignación existen numerosos algoritmos resolutivos, siendo el más extendido el algoritmo húngaro, las soluciones a los problemas de transporte se sustentan tradicionalmente en un conjunto de métodos para resolver problemas de programación lineal conocidos como el método Simplex, siendo por tanto el empleado en este proyecto para abordar la fase de asignación de objetivos en las tareas de atrapamiento con múltiples evasores.

ANEXO II – Herramientas aplicadas

El presente anexo introduce las herramientas empleadas en este proyecto, entiéndase formalismos matemáticos y algoritmos, para abordar el modelado de entornos complejos y las tareas de atrapamiento coordinado.

A partir de un mapa binario del entorno como único dato de entrada, se desea obtener una representación del entorno en forma de grafos: uno que constituya un mapa de trayectorias libres de colisiones, y otro que divida el entorno en regiones convexas.

- Para la elaboración de trayectorias existen diversas técnicas, siendo las más destacadas el uso de cuadrículas, los grafos de visibilidad y el diagrama de Voronoi. De entre estas, se ha escogido la última porque maximiza la distancia entre trayectorias y obstáculos, además es una herramienta con la que ya se ha trabajado en la fase anterior del proyecto.
- Para la generación de regiones convexas, la técnica empleada debido a su sencillez computacional y su eficacia frente a geometrías muy dispares ha sido la triangulación de Delaunay.

En cuanto a la captura, esta se planifica sobre el grafo regiones mediante el algoritmo de Dijkstra, un algoritmo de búsqueda informada en grafos que calcula el camino de mínima distancia entre dos nodos de un grafo. Este algoritmo cuenta con una gran eficiencia para grafos de puntos reducidos.

Adicionalmente, en capturas con múltiples evasores, es necesario incluir una fase de asignación de objetivos. Se trata de un problema de programación lineal conocido como problema de transporte, y la solución se obtiene mediante un conjunto de métodos conocidos como método Simplex.

1. Diagramas de Voronoi

Estos diagramas. que toman su nombre del matemático ruso Georgy Voronoi, aunque también son conocidos como Polígonos de Thiessen o Teselación de Dirichlet, hacen una descomposición de un espacio métrico en regiones, asociada a la presencia de objetos, de tal forma que, en dicha descomposición, a cada objeto se le asigna una región del espacio métrico más cercano a él que a ningún otro objeto [7].

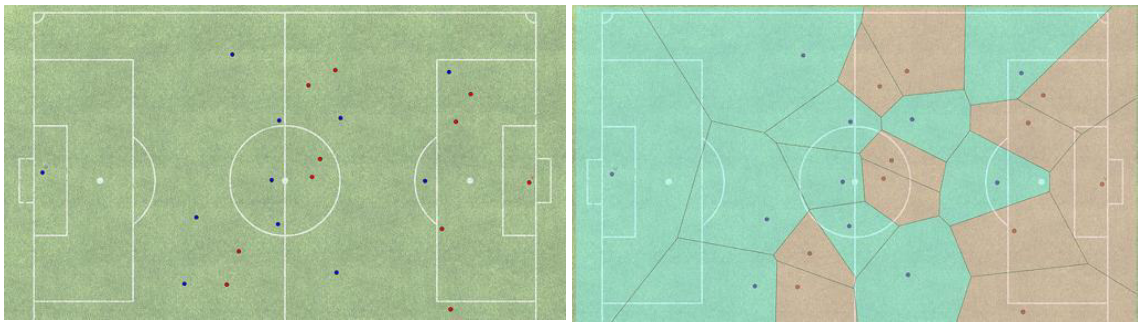


Imagen 42 - Diagrama de Voronoi aplicado a 22 jugadores sobre un campo de fútbol.

Los diagramas de Voronoi encuentran infinidad de aplicaciones más allá de la robótica, como por ejemplo en áreas tales como gráficos por computadora, medicina, epidemiología, geofísica, meteorología, sistemas de información geográfica o sistemas de navegación. Incluso en el fútbol se le puede encontrar aplicación para visualizar la ventaja posicional de un equipo sobre otro (Imagen 42).

Los diagramas de Voronoi son uno de los métodos de interpolación más simples basados en la distancia euclidiana y es uno de los métodos más utilizados para obtener grafos de proximidad. De los distintos algoritmos que existen para la construcción del diagrama de Voronoi, el más extendido debido a su sencilla implementación es el algoritmo de Fortune, basado en una de las técnicas clave dentro de la geometría computacional denominada barrido de la recta.

En el presente proyecto, para la obtención del diagrama se hace uso de la función v*oronoi* que tiene Matlab implementada, la cual implementa el algoritmo de Fortune. Los datos generados son procesados posteriormente para obtener una representación adecuada del entorno.

2. Grafo euclídeo no dirigido y matriz de adyacencia

En matemáticas y ciencias de la computación, un grafo es un conjunto de vértices o nodos unidos por enlaces llamados aristas o arcos [6]. Los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de distribución de mercancías puede representarse mediante un grafo, en el cual los nodos representan los puntos de recogida y/o llegada de mercancía, y las aristas reflejan la distancia o el coste asociado de ir de un nodo a otro (Imagen 43).

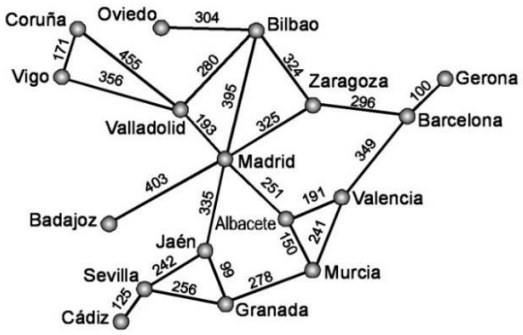


Imagen 43 - Grafo en el que los nodos representan ciudades y las aristas la distancia en km entre nodos.

Concretamente, un grafo euclídeo es un grafo cuyos vértices representan puntos del plano, y a sus aristas se le asignan pesos iguales a la distancia euclídea entre sus dos vértices.

Los grafos pueden ser clasificados de muchas formas, una de ellas atendiendo a la reciprocidad o no de las conexiones entre dos nodos, pudiendo distinguir entre grafos dirigidos y no dirigidos (Imagen 44).

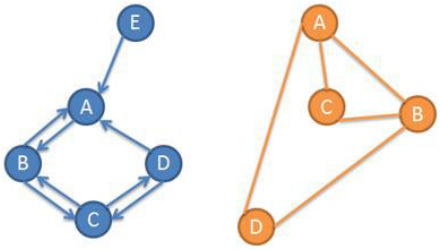
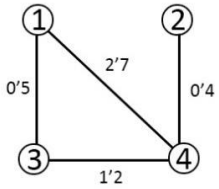


Imagen 44 - Izqda.: Grafo dirigido. Dcha.: Grafo no dirigido.

En el presente proyecto, los vértices del conjunto de rectas que definen las trayectorias libres de colisiones representan los nodos del grafo, mientras que las aristas reflejan la unión entre dos nodos que están conectados por una recta, con un peso asignado igual a la distancia que los separa. Por tanto, el grafo obtenido es un grafo euclídeo no dirigido.

Este grafo está representado por una matriz de adyacencia (Imagen 45): matriz cuadrada de dimensión n, siendo n el número de nodos, en el que los elementos constituyen los pesos asociados a las aristas, de modo que el elemento (i,j) indica el coste asociado a avanzar del nodo i al nodo j. Un 0 indica que los nodos no están conectados.



$$G = \begin{pmatrix} 0 & 0 & 0.5 & 2.7 \\ 0 & 0 & 0 & 0.4 \\ 0.5 & 0 & 0 & 1.2 \\ 2.7 & 0.4 & 1.2 & 0 \end{pmatrix}$$

Imagen 45 - Matriz de adyacencia asociada a un grafo no dirigido.

3. Triangulación de Delaunay

La triangulación de Delaunay genera en espacios bidimensionales, para un conjunto dado de vértices, una red de triángulos conexa y convexa que cumple la condición de Delaunay: la circunferencia circunscrita de cada triángulo de la red no debe contener ningún vértice de otro triángulo [8]. La Imagen 46 muestra la triangulación de Delaunay para un conjunto de 10 pùntos.

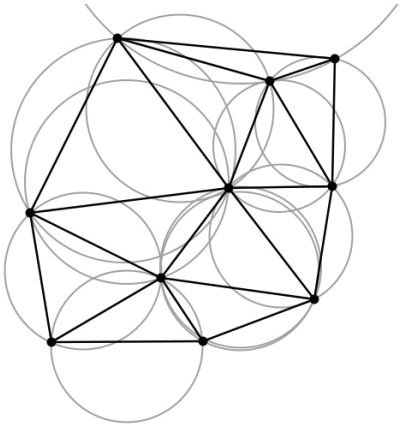


Imagen 46 - Triangulación de Delaunay de 10 puntos.

De los distintos algoritmos que existen para calcular la triangulación de Delaunay, los que permiten su construcción directa son los algoritmos de barrido. De hecho, una técnica común es emplear el algoritmo de Fortune para elaborar el diagrama de Voronoi, pues a partir de este puede extraerse fácilmente la triangulación de Delaunay.

En el presente proyecto, la triangulación se obtiene haciendo uso de la función *delaunay* que tiene Matlab implementada. Los datos generados son procesados posteriormente para obtener una representación adecuada del entorno.

4. Algoritmo de Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto desde un vértice origen hacia el resto de los vértices de un grafo con pesos en cada arista [9]. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo describió por primera vez en 1959.

Una de sus aplicaciones más importantes reside en el campo de la telemática. Gracias a él, es posible resolver grafos con muchos nodos, encontrando las rutas más cortas entre un origen y todos los destinos en una red.

Este algoritmo cuenta con una extensa documentación y funciones que lo implementan en los distintos lenguajes de programación. En el presente proyecto se usa una función en Matlab que recibe como argumentos de entrada la matriz de adyacencia, el nodo origen y el nodo destino, y devuelve la secuencia de nodos que constituye el camino más corto, así como la distancia recorrida entre el nodo origen y el nodo destino.

5. Programación lineal: Método Simplex

La programación lineal es el campo de la optimización matemática dedicado a maximizar o minimizar una función lineal, denominada función objetivo, de tal forma que las variables de dicha función satisfagan una serie de restricciones expresadas mediante un sistema de ecuaciones o inecuaciones también lineales.

$$\min_x f(x) : \begin{cases} A_{eq} \cdot x = b_{eq} \\ A \cdot x \leq b \end{cases}$$

Donde:

- $x = vector\ de\ N\ variables, con\ x_i \geq 0.$
- $f(x) = \sum_{i=1}^{N} c_i \cdot x_i$, con $c_i = coeficientes.$
- $A, A_{eq} = matrices.$
- $b, b_{eq} = vectores.$

El método tradicionalmente usado para resolver problemas de programación lineal es el Método Simplex [10]. Fue desarrollado por el matemático norteamericano George Dantzig en 1947, y procede de forma iterativa examinando vértices adyacentes del poliedro de soluciones, siendo, por tanto, un algoritmo de pivote.

En este proyecto, este algoritmo es empleado para resolver el problema de asignación de objetivos en capturas con múltiples evasores. La herramienta Matlab integra una función para este fin que resuelve problemas de programación entera, denominada `intlinprog`, que internamente implementa el método Simplex. El problema queda definido por los parámetros:

`x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub)` → $\min_x f^T x$ subject to $\begin{cases} x(\text{intcon}) \text{ are integers} \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub. \end{cases}$

ANEXO III – Diagrama de Voronoi en entornos cerrados convexos

La función *voronoi* de Matlab recibe como argumentos de entrada dos vectores fila, los cuales almacenan, respectivamente, las coordenadas X e Y de los puntos para los que se desea obtener el diagrama. Como argumento de salida devuelve dos matrices que almacenan, en columnas, los vértices finitos de las rectas Voronoi que componen el diagrama (Ver Imagen 47).

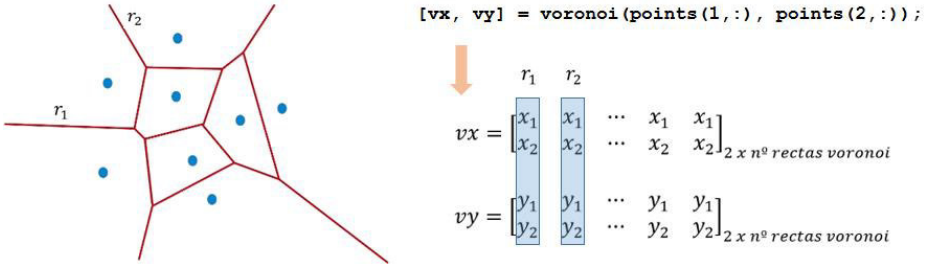


Imagen 47 - Argumentos de salida de la función 'voronoi' que incorpora Matlab R2015.

La función creada en este proyecto para obtener Voronoi en entornos cerrados convexos recibe como argumentos de entrada dos matrices, una que contiene las coordenadas de los puntos para los que se desea trazar el diagrama, y otra que contiene las coordenadas de los puntos que definen el contorno cerrado. La función devuelve tres argumentos de salida (Ver Imagen 48):

- Una matriz con las coordenadas de los puntos para los que finalmente se calcula la partición de Voronoi, que son el subconjunto de los puntos de entrada que están contenidos en el contorno especificado.
- Una matriz de adyacencia que indica, para cada punto, qué otros puntos comparten frontera Voronoi con él.
- Una matriz multidimensional columna en la que cada fila constituye una matriz que almacena las coordenadas de los vértices de las rectas Voronoi que delimitan a un punto, así como con qué vecino están compartiendo cada frontera.

```
function [points_in, vecinos, regiones] = voronoi_boundary_2D(points, bound)
```

$$points\_in = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix}_{2\ x\ nº\ puntos\_in}$$

$$vecinos = \begin{bmatrix} 0 & \square & \square & \square & \square \\ \square & 0 & \square & \square & \square \\ \square & \square & 0 & \square & \square \\ \square & \square & \square & 0 & \square \\ \square & \square & \square & \square & 0 \end{bmatrix}_{nº\ puntos\_in\ x\ nº\ puntos\_in}$$

$$regiones = \begin{Bmatrix} [\ ]_{5\ x\ nº\ rec\_vor\_reg\_1} \\ [\ ]_{5\ x\ nº\ rec\_vor\_reg\_2} \\ [\ ]_{5\ x\ nº\ rec\_vor\_reg\_3} \\ [\ ]_{5\ x\ nº\ rec\_vor\_reg\_4} \\ [\ ]_{5\ x\ nº\ rec\_vor\_reg\_5} \end{Bmatrix} \quad regiones\{1\} = \begin{bmatrix} x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 \\ y_1 & y_1 & y_1 \\ y_2 & y_2 & y_2 \\ v_{r1} & v_{r2} & v_{r3} \end{bmatrix}_{5\ x\ nº\ rec\_vor\_reg\_1}$$

Imagen 48 - Argumentos de salida de la función 'voronoi_boundary_2D'.

1. Función voronoi_boundary_2D

Inicialmente, la función procesa los argumentos de entrada para asegurar, por un lado, que el contorno sea convexo y, por otro lado, que los puntos introducidos estén contenidos en dicho entorno convexo. A continuación computa, con la función *voronoi* de Matlab, el diagrama Voronoi asociado a dichos puntos, a partir del cual se elaborará el diagrama Voronoi delimitado.

El algoritmo desarrollado para delimitar el diagrama Voornoi es bastante complejo, y se ha desarrollado en varias fases:

1. Reconocimiento de vértices compartidos. Fase necesaria para el posterior análisis de intersecciones.
2. Análisis de intersecciones. Si una recta intersecta con el contorno, se actualizan sus vértices en consecuencia para que quede delimitada por el contorno, además de crear dos nuevas rectas desde el punto intersección a los extremos de la recta contorno con la que delimita. La lógica de detección de intersecciones se ha recogido en una función llamada *intersect_2D*.
3. Creación de rectas contorno. Se amplia el diagrama Voronoi con las parejas de vértices que constituyen las distintas rectas del contorno convexo.
4. Reconocimiento de regiones y vecinos. Para cada par de vértices, se comprueba la distancia que hay a cada punto. Que las distancias coincidan refleja la existencia de una recta compartida entre los puntos cuyas distancias coinciden, mientras que si no coinciden y son mínimas para un punto se trata de una recta que pertenece únicamente a dicho punto. Rectas que no cumplen ninguno de los casos descritos son eliminadas.
5. Cierre de regiones abiertas. Fase necesaria cuando dos o más rectas intersectan con una misma recta contorno.

Las Imagen 49 muestra el orden lógico recién descrito para distintas tipologías de contornos:

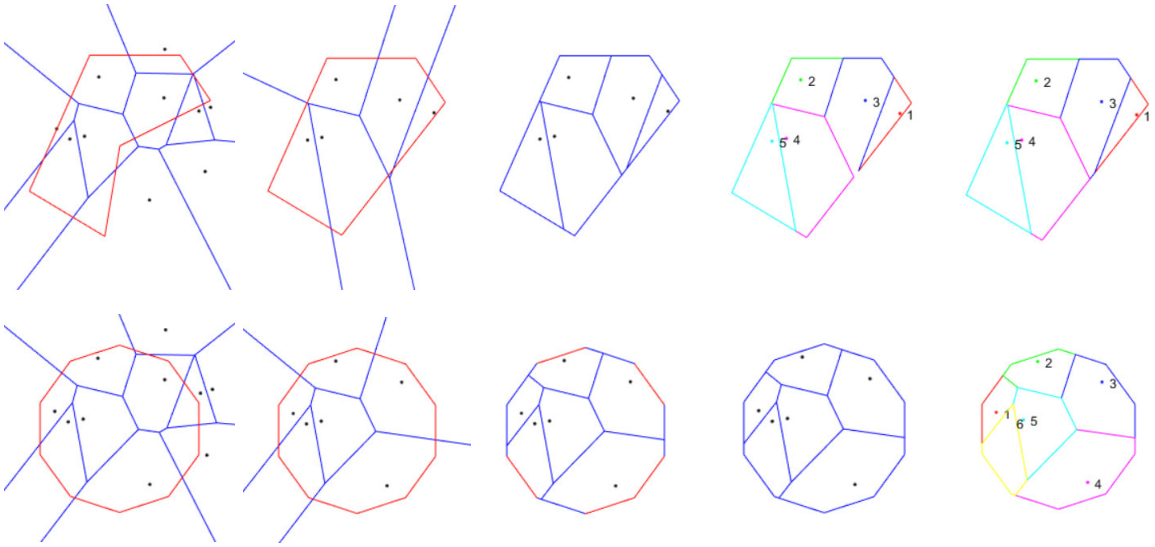


Imagen 49 – Secuencia lógica de obtención del diagrama Voronoi delimitado para distintos entornos cerrados.

2. Función intersect_2D

Esta función se encarga de calcular, si existe, el punto de intersección de una recta Voronoi con el contorno que la delimita. Es importante observar que todas las rectas que constituyen el diagrama de Voronoi comparten, como mínimo, uno de sus dos vértices con otra recta. Este hecho se ha tenido en cuenta a la hora de desarrollar el algoritmo.

```
function [p_corte, j_bound] = intersect_2D(x_1, x_2, y_1, y_2, boundary, compartido)
```

Argumentos de entrada:

- Coordenadas de un vértice compartido.
- Coordenadas del segundo vértice.
- Matriz columna que contiene las coordenadas de los puntos que definen las rectas que componen el contorno cerrado convexo.
- Booleano que indica si el segundo vértice es o no un vértice compartido.

Argumentos de salida:

- Matriz columna con las coordenadas de los puntos de intersección detectados.
- Vector fila que indica con qué recta ha ocurrido la intersección para cada punto de intersección detectado. Si no existe intersección se devuelve un escalar con el valor 0.

ANEXO IV – Elaboración del grafo de trayectorias

1. Procesado del mapa binario

La Imagen 50 muestra el procesado realizado para distintos mapas binarios de entrada.

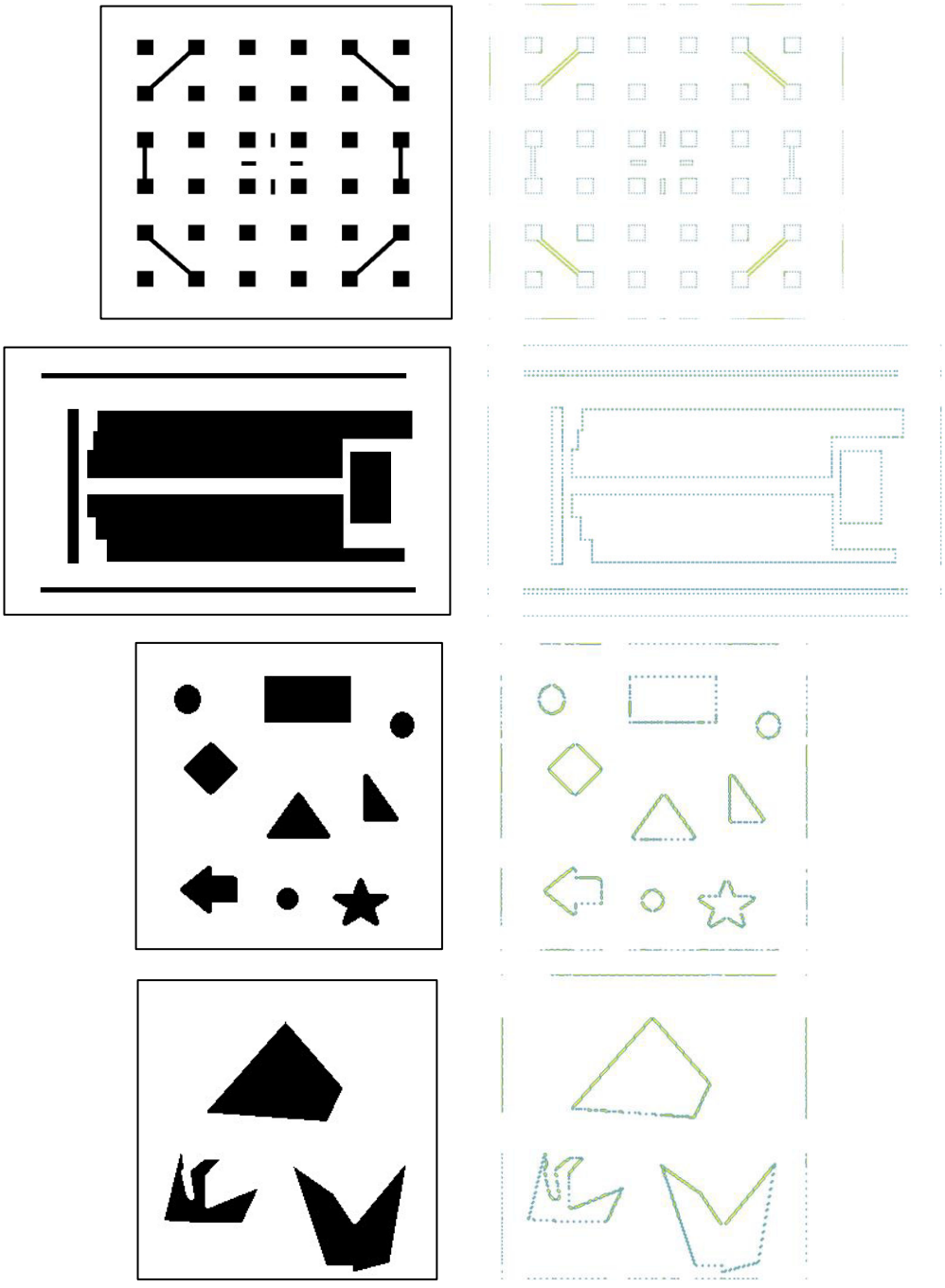


Imagen 50 – Ejemplos de procesado del mapa binario de entrada.

2. Eliminación de rectas no válidas

Se entiende como recta no válida toda recta resultante del diagrama de Voronoi que se encuentra a una distancia menor que la distancia de seguridad establecida de una zona no transitable. Esta distancia de seguridad viene dada en pixeles, de modo que su equivalencia en unidades métricas depende de la resolución espacial de la matriz binaria de entrada.

Esta primera fase de post-procesado se computa comprobando, para cada una de las rectas que componen el diagrama de Voronoi inicial, si las coordenadas, en pixeles, de los vértices que constituyen una recta se corresponden a una zona no transitable, o bien si estas coordenadas se encuentran de una zona no transitable a una distancia menor que la definida como seguridad. Por tanto, esta comprobación se realiza sobre la matriz que constituye el mapa binario de entrada.

El efecto que tiene la distancia de seguridad sobre el grafo trayectorias puede verse en la Imagen 51.

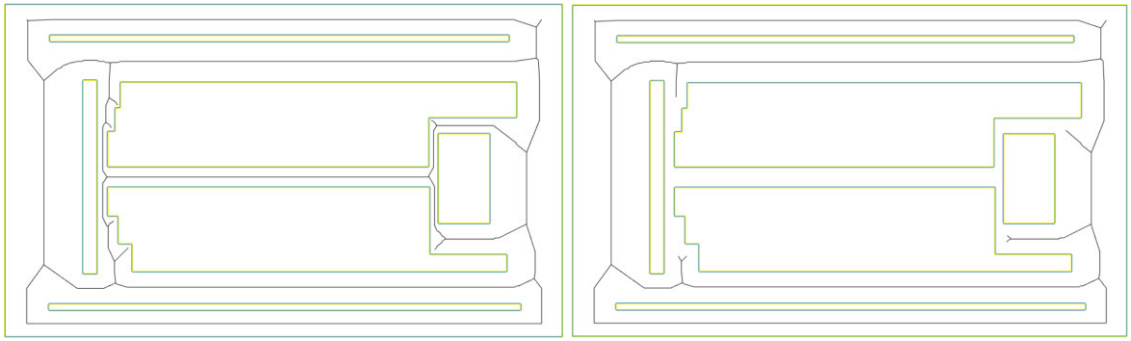


Imagen 51 - Efecto de la distancia de seguridad sobre las trayectorias generadas.

3. Filtrado de rectas inconexas

En entornos que presentan geometrías que no son marcadamente verticales u horizontales, el diagrama que se obtiene presenta un número notable de trayectorias que no aportan información útil al mapa de trayectorias, tal y como puede verse en la Imagen 52. Este tipo de rectas pueden identificarse detectando vértices inconexos, es decir, rectas que no conectan con ninguna otra recta por uno de sus extremos.

Por tanto, esta segunda fase de post-procesado puede realizarse eliminando todas las rectas que tengan vértices inconexos, obteniendo en este caso el mapa de trayectorias de la Imagen 53. No obstante, esta solución no es válida en otro tipo de entornos como el de la Imagen 54, pues en este caso sólo un pequeño porcentaje de rectas inconexas son las que no aportan información útil.

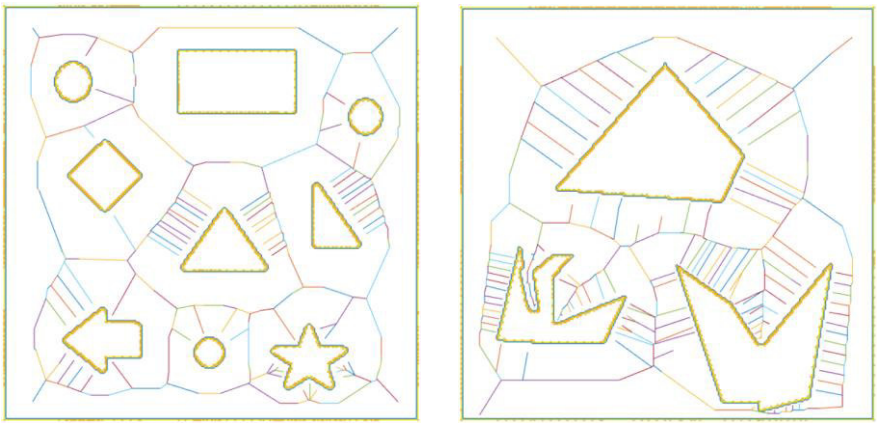


Imagen 52 - Post-procesado final del diagrama de Voronoi en entornos irregulares.

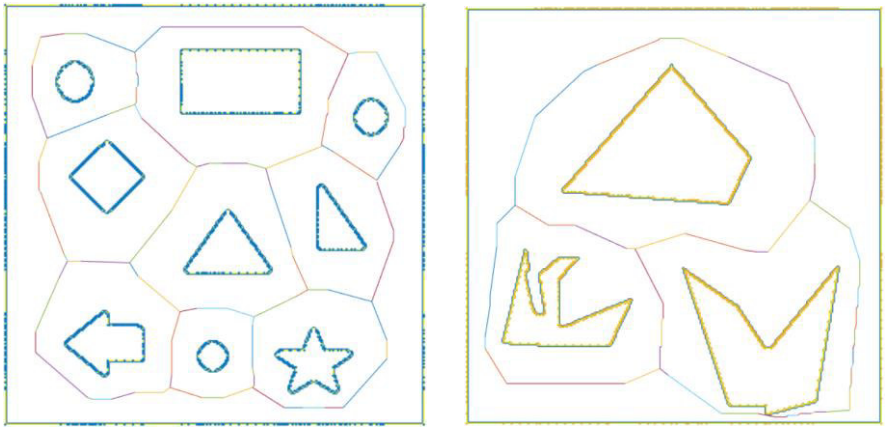


Imagen 53 - Post-procesado final del diagrama de Voronoi en entornos irregulares sin rectas inconexas.

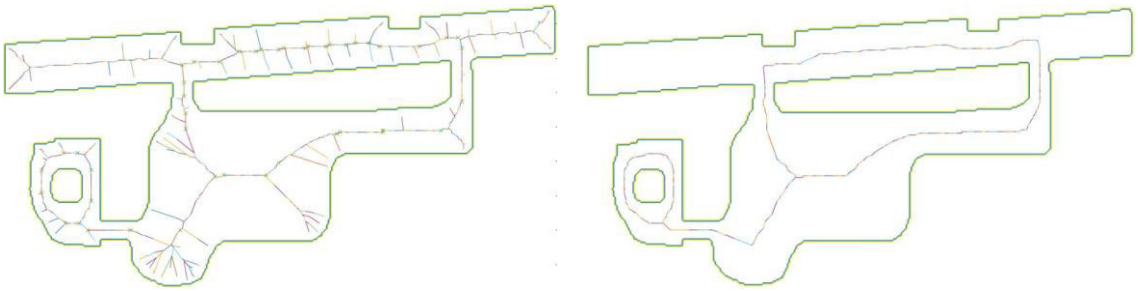


Imagen 54 - Efecto de eliminar todas las rectas inconexas en un entorno con caminos abiertos.

Por este motivo, es necesario definir un parámetro de filtrado ajustable según el tipo de entorno. Este parámetro limita el número máximo de rectas inconexas de una misma rama que se pueden eliminar de forma consecutiva. De este modo se consigue una solución que es válida tanto para entornos de caminos cerrados, con un parámetro de filtrado elevado, como para entornos de caminos abiertos, con un parámetro de filtrado menor.

La Imagen 55 muestra, para el entorno de la Imagen 54, la solución obtenida bajo la lógica de filtrado ajustable. Las mejoras que introduce esta solución respecto a la eliminación total son notorias, y lo son más aún en otro tipo de entornos como el de la Imagen 8, donde la eliminación total de rectas inconexas genera que el grafo de trayectorias deje gran parte del entorno sin cubrir.

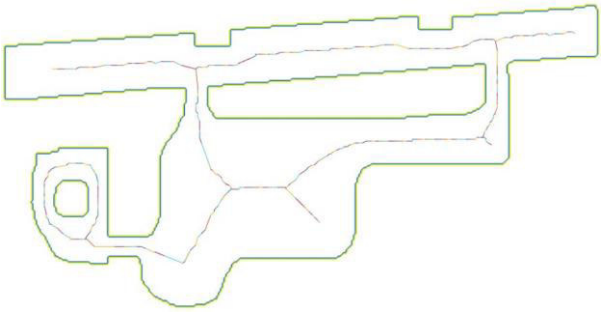


Imagen 55 - Filtrado ajustable de rectas inconexas en un entorno con caminos abiertos.

4. Fusionado de rectas

Gran parte de los algoritmos empleados en la planificación de la captura o la asignación de objetivos tienen un coste computacional que proporcional al tamaño del grafo de trayectorias. Por este motivo, y dado que se desea que el algoritmo de captura pueda ejecutarse en tiempo real, es importante reducir, en la medida de lo posible, el número de nodos del grafo sin que ello conlleve una pérdida de información. Esto se consigue mediante el fusionado de rectas.

Este proceso de fusionado se realiza en dos fases. La primera consiste en el fusionado de rectas horizontales y verticales, mientras que la segunda es más compleja y tiene en cuenta diversos factores:

- Si la diferencia de ángulo entre una recta y la recta resultante de fusionar esta con su contigua es menor que un valor de comparación determinado, ambas rectas se fusionan en una sola.
- Sólo está permitida la fusión si la longitud de la recta resultante está por debajo de un valor definido. Por defecto, este valor será igual al campo de visión de los robots.
- Rectas cuya longitud sea menor que un valor definido, podrán fusionarse aunque no cumplan la condición de ángulo.
- Las rectas verticales u horizontales sólo podrán fusionarse si tienen una longitud menor que un determinado valor.
- La fusión sólo ocurrirá si los vértices eliminados únicamente pertenecen a las dos rectas objeto de estudio, de lo contrario se producirán desconexiones en el grafo.

Esta última fase del post-procesado consigue resultados satisfactorios para todo tipo de entornos para una misma configuración de parámetros. La Imagen 56 muestra el post-procesado final para un entorno marcadamente cuadrado, y otro marcadamente circular.

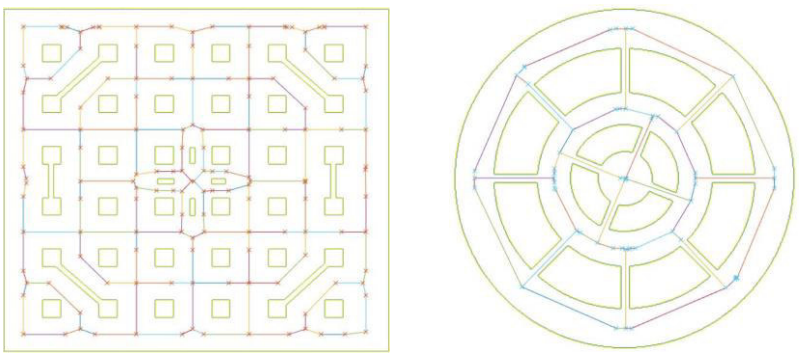


Imagen 56 - Post-procesado final del diagrama de Voronoi en un entorno cuadrado (izqda.) y en uno circular (dcha.).

ANEXO V – Elaboración del grafo de regiones

1. Resultados

Los resultados obtenidos en la elaboración del grafo de regiones convexas se ven afectados por la geometría del entorno. Entornos de geometría regular con formas mayormente rectangulares, como los mostrados en la Imagen 57, son los que ofrecen mejores resultados, consiguiendo una división ordenada y uniforme del espacio.

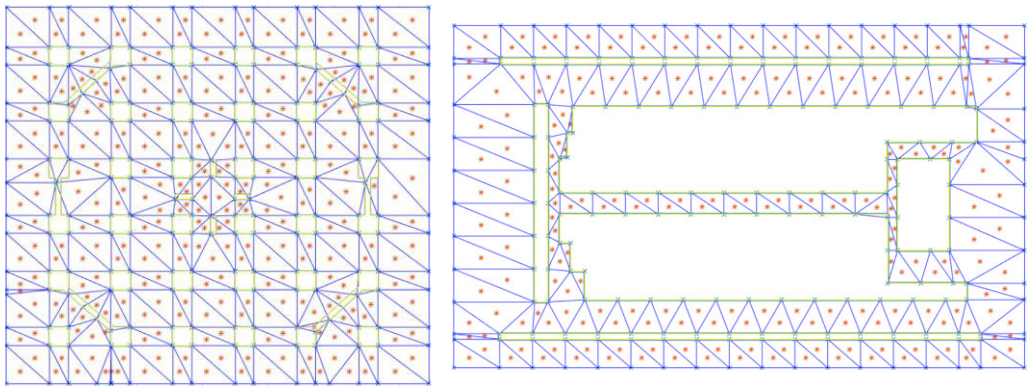


Imagen 57 – Grafo de regiones convexas en entornos de geometría mayormente rectangular.

En entornos marcadamente circulares también se obtiene una división ordenada, aunque menos uniforme, del espacio. En el caso del entorno mostrado en la Imagen 58, debido a la diferencia de anchura entre los anillos circulares y los pasillos intermedios, es necesario realizar el particionado con un tamaño de región pequeño, obteniéndose un grafo de número de nodos elevado, lo que puede traducirse en algunos casos en una mayor carga computacional.

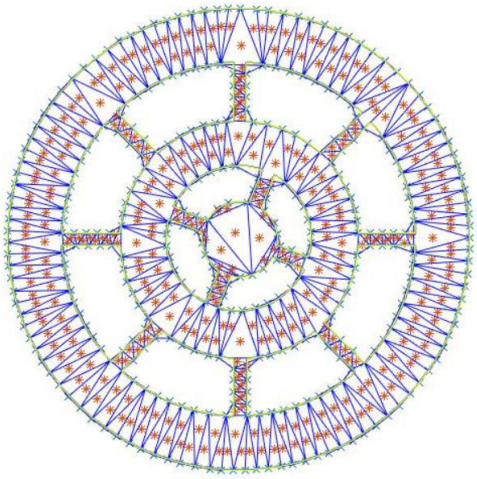


Imagen 58 – Grafo de regiones convexas en un entorno marcadamente circular.

Sin embargo, conforme la geometría del el entorno se hace más irregular, el particionado que se obtiene se vuelve menos ordenado y uniforme, tal y como se muestra en la Imagen 59. No obstante, este hecho no perjudica al desarrollo posterior de la captura.

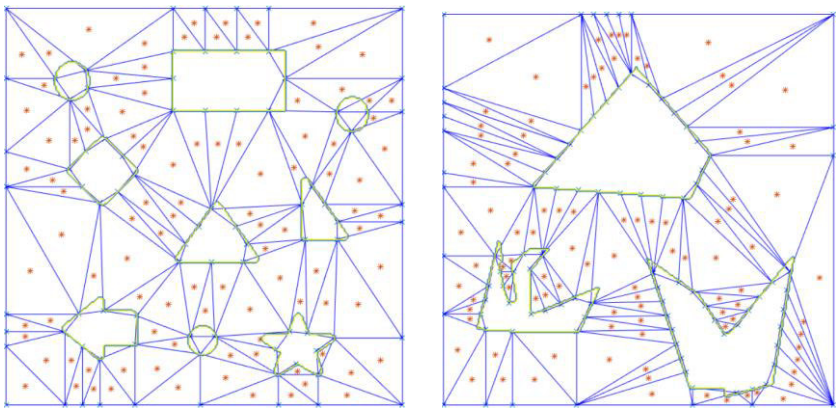


Imagen 59 - Grafo de regiones convexas en entornos de geometría irregular.

ANEXO VI – Función voronoi_2P

Esta función genera, internamente, el diagrama de Voronoi correspondiente a dos puntos que se encuentran en un entorno cerrado convexo, y devuelve, como único dato de salida, la recta Voronoi compartida por ambos puntos.

```
function [frontera] = voronoi_2P(purs, evad, boundary, HV)
```

El diagrama asociado a dos puntos delimitados por un polígono convexo se puede obtener de manera sencilla trazando una recta que pase por el punto medio de la recta que une ambos puntos y que además sea perpendicular a esta. Por tanto, los pasos lógicos que sigue el algoritmo son los siguientes:

1. Obtención del punto medio y de la pendiente de la recta que une ambos puntos.
2. Obtención de la ecuación de la recta que es perpendicular a la anterior que intersecta con su punto medio.
3. Cálculo de los puntos de intersección de ésta última recta con el contorno del polígono cerrado convexo que delimita los puntos.

Los argumentos de entrada que recibe la función son, respectivamente:

- Coordenadas del agente perseguidor.
- Coordenadas del agente evasor.
- Matriz columna que contiene los puntos que definen las rectas que componen el contorno cerrado convexo.
- Escalar que representa un valor infinito a efectos prácticos del problema.