Trabajo fin de grado
Ingeniería Aeroespacial

Exploración de un terreno usando drones periódicamente conectados

Autor: Antonio Mendoza Vázquez
Tutor: José Miguel Díaz Báñez

Dep. Matemática aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo fin de grado
Ingeniería aeroespacial

Exploración de un terreno usando drones periódicamente conectados

Autor:
Antonio Mendoza Vázquez

Tutor:
José Miguel Díaz Báñez
Profesor titular

Dep. Matemática aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo fin de grado: Exploración de un terreno usando drones periódicamente conectados

Autor: Antonio Mendoza Vázquez

Tutor: José Miguel Díaz Báñez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

Agradecimientos

A la gente que me ha apoyado, familia, amigos, al prof. Díaz-Bánez por haberme dado la oportunidad de hacer este interesante proyecto, y a Evaristo Caraballo por su permanente ayuda y disponibilidad.

Antonio Mendoza Vázquez

Sevilla, 2016

Resumen

En el ámbito aeronáutico, el uso de drones o vehículos aéreos no tripulados está cobrando cada vez más fuerza en diferentes labores, tales como el control de incendios forestales, control de la ganadería o rescate de personas. Este proyecto se centra en estudiar métodos eficientes de exploración y cubrimiento del terreno mediante UAVs restringidos a que cada cierto intervalo de tiempo deben reunirse en un punto de encuentro para compartir información, ver que dichos vehículos siguen operativos y comprobar que las tareas han sido realizadas de forma correcta. Las conexiones periódicas tienen interés debido a la existencia de obstáculos, de límites en el rango de comunicación de los vehículos, y como medida para el ahorro energético.

Para ello, se parte de un algoritmo propuesto, que intenta resolver de forma eficiente el problema de exploración del terreno mediante robots periódicamente conectados. Se estudian sus debilidades y aspectos en los que no es eficiente y se plantean posibles mejoras. Este trabajo podría tener utilidad no sólo en la exploración o cubrimiento de terrenos, sino también, en los nuevos proyectos de envíos de paquetes, control de incendios, trabajos del sector agrícola, etc…

Abstract

Nowadays, more and more drones or UAVs are used for many types of tasks. In this project we will focus on studying methods of exploration and terrain covering by UAVs, and how we can optimize these tasks, with the restriction that the vehicles must meet periodically with the aim of sharing information, checking they are still operating, checking the tasks have been done correctly, etc ... The idea of periodic connectivity is based on the existence of obstacles or limits at the range of communication between the drones.

For that reason, we start from a proposed algorithm. We have studied some environments where its use may not be efficient and we have proposed improvements. This project could be useful not only for exploring or covering many environments, also in new projects of package shipments or fire control.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 -INTRODUCCIÓN

Este proyecto tiene como objetivo el estudio y propuesta de posibles soluciones o heurísticos viables para tratar el problema de exploración y cubrimiento de un terreno con obstáculos mediante UAVs. Esta tarea puede ser útil para estimar el cambio climático, las condiciones de los océanos, buscar a una persona perdida en una montaña, etc. [2]. El uso de drones para cubrir terrenos posee las siguientes ventajas: No requieren pilotaje, nos permiten acceder a zonas donde el hombre es incapaz de entrar, nos dan una visión amplia (vista aérea) del terreno y pueden comunicarse entre ellos mediante métodos de radiofrecuencia con técnicas como modulación digital con una señal portadora.

El planteamiento del problema es el siguiente: Varios vehículos aéreos no tripulados o drones, parten de un punto inicial y tienen como objetivo el cubrimiento o exploración del terreno. Existen dos métodos fundamentales para la realización de dicha tarea. El primer método consiste en que los vehículos están en conexión permanente, método que posee las siguientes ventajas: La compartición continua de la información obtenida, la coordinación del siguiente paso del plan de vuelo, y la comprobación constante de que todos los UAVs siguen operativos. Y el segundo método consiste en que los robots se conectan periódicamente cada cierto intervalo de tiempo que llamaremos T_i [2]. Así, cada Ti unidades de tiempo, los drones deben situarse a una distancia relativamente cercana para compartir información. Las conexiones periódicas mejoran la eficiencia en el cubrimiento y exploración de los diferentes terrenos porque se disminuye el tiempo total de realización, y la distancia total recorrida por el equipo de drones. En el campo de los UAVs, el hecho de que las tareas se realicen eficientemente, conlleva ahorro en energía y por tanto alargamiento de vida útil del robot. Así, partiremos del algoritmo propuesto en [2], analizaremos sus debilidades y limitaciones, mostrando terrenos donde el algoritmo no de un buen rendimiento. Propondremos un algoritmo alternativo para la eficiencia en la realización de las tareas anteriormente mencionadas.

Por otro lado, debemos distinguir entre el problema de exploración y de cubrimiento. El problema de exploración consiste en ir en búsqueda de un objetivo concreto, ya sea una persona o un objeto. Idea que tiene grandes aplicaciones militares para labores de rescate, y búsqueda de ojetivos móviles enemigos [4]. Por otro lado, el problema de cubrimiento [3] consiste en visitar todo el área o volumen del terreno evitando obstáculos. Este problema tiene importancia en numerosas aplicaciones de la robótica, exploración de barcos o aviones hundidos en el mar, reconocimiento de terrenos, etc…. El algoritmo que se va a proponer en este proyecto está relacionado con el problema de cubrimiento [3], siendo el objetivo principal de este proyecto su resolución de la manera más eficiente posible.

Por otro lado, el lenguaje de implementación será MATLAB, por su rapidez computacional y eficiencia en el tratamiento vectorial, que nos va a permitir realizar las diferentes operaciones del algoritmo con agilidad.

En la figura 1.1 podemos observar el problema de exploración mediante drones periódicamente conectados. Los robots aéreos inician el intervalo de desconexión en una configuración donde están conectados, después, cada uno realiza sus tareas de forma

independiente, y tras Ti unidades de tiempo vuelven a estar conectados.

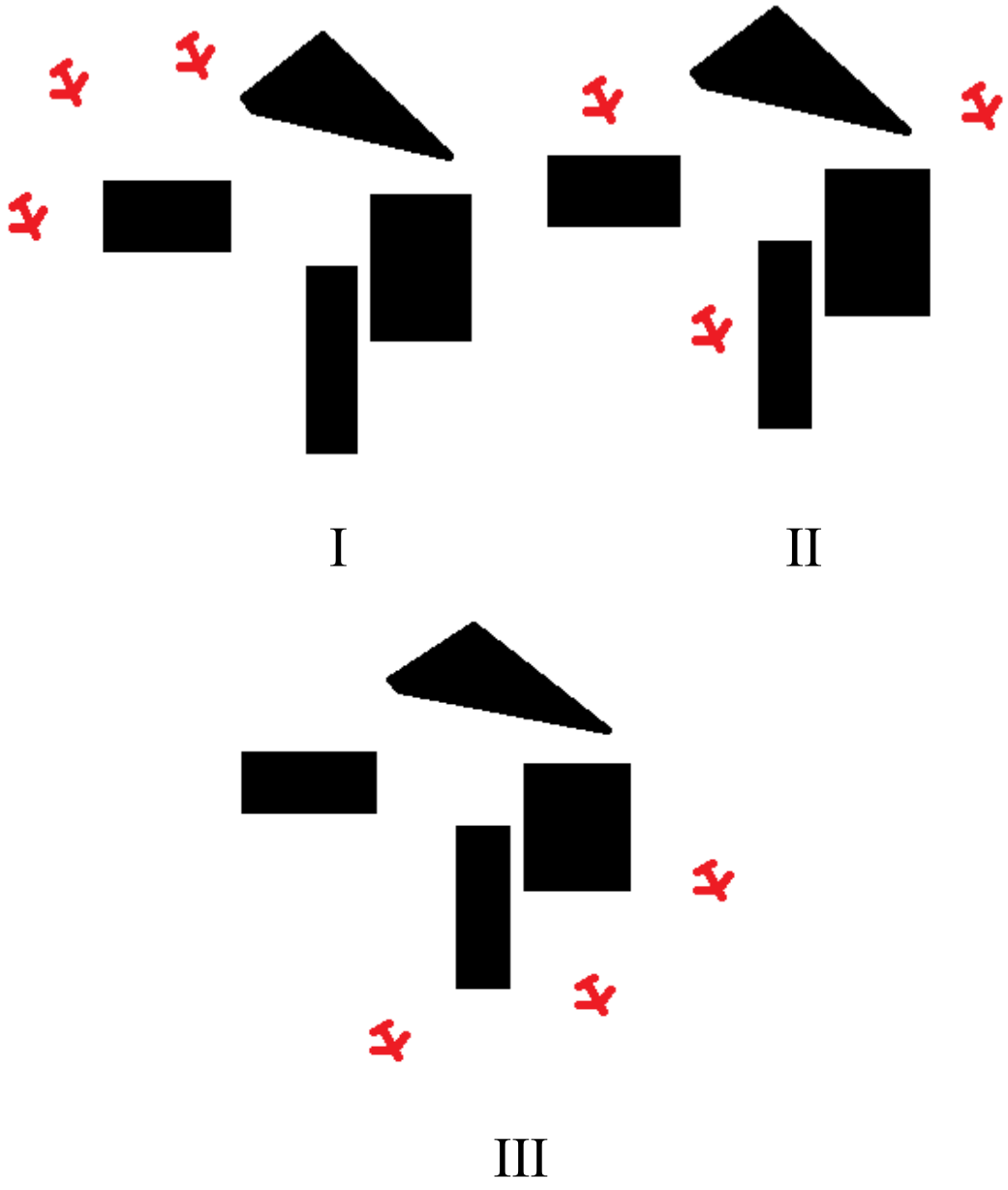


Figura 1.1. Inicialmente los robots parten de un punto inicial donde están conectados (I), en la situación (II), rompen la conexión para realizar sus tareas programadas durante un intervalo Ti. Finalmente vuelven a un punto donde están conectados de nuevo (III).

Como hemos mencionado anteriormente, este proyecto se va a centrar fundamentalmente en el problema de cubrimiento, por lo que el objetivo del algoritmo que vamos a desarrollar es visitar todo el terreno en el menor tiempo posible intentando hacerlo de manera eficiente y ordenada.

Además, es necesario comentar las posibles optimizaciones del problema, que serán fundamentalmente: Minimizar el número de unidades de área repetidas, y maximizar el número de celdas nuevas visitadas en cada paso del algoritmo. El concepto de celda viene dado por el método de discretización del terreno que vamos a usar en este proyecto, que

consistirá en dividir dicho terreno en diferentes casillas o celdas, diferenciando las celdas visitables y los obstáculos.

Finalmente, como trabajo relacionado a este proyecto, se ha partido como base de [2], artículo donde se ha planteado un algoritmo que propone un método para resolver el problema de exploración de un terreno con robots periódicamente conectados. Por otro lado, en [1], se extrapola el problema de conexiones periódicas que se plantea en [2] a problemas reales, como el de la navegación marítima. En [1] se proponen técnicas de navegación que consisten en ir buscando puntos de encuentros situados en el mar para tener controlada la ruta o el estado de un barco en ausencia de GPS o aparatos de radionavegación. Esta última idea será un punto clave en el desarrollo de nuestro algoritmo.

Otro aspecto importante a tener en cuenta en el ámbito de la navegación, es la situación de los diferentes puntos de encuentro, la cual no debe ser arbitraria. En [11] se plantea un sistema para que la posición relativa de los vehículos en dichos puntos de encuentro sea tal, que se maximice la conectividad de los vehículos, es decir, que éstos reciban mutuamente la máxima potencia posible de las señales emitidas.

Existen muchos artículos alternativos a [2] que plantean métodos para resolver problemas de cubrimiento o exploración. Por ejemplo, en [12], se estudia el problema de localizar un objetivo en un terreno determinado, ya sea una persona o un objeto. El algoritmo que se plantea en [12] tiene muchísimas aplicaciones en el ámbito militar. Sin embargo, nuestro algoritmo va a centrarse fundamentalmente en el problema de cubrimiento de un terreno determinado.

En [18] y [19], también se exponen estrategias para el cubrimiento de un terreno, sometido a restricciones en el rango de comunicación, con el objetivo de garantizar la compartición de la información entre todos los robots cada Ti unidades de tiempo.

2 -EL ALGORITMO DE COORDINACIÓN IMPLÍCITA

En este capítulo vamos a plantear el problema y explicar el método que exponen en [2], denominado *Implicit coordination algorithm* o MIPP-PC (multirobot informative path planning with periodic connectivity) . En este artículo, se estudia el cubrimiento y la exploración de un terreno mediante un equipo de drones conectados de forma periódica cada Ti unidades de tiempo. Cada dron en el intervalo de tiempo en el que permanecen desconectados, siguen un plan de vuelo o ruta determinada independiente de los demás.

2.1. Consideraciones iniciales.

Inicialmente, tenemos un terreno que queremos cubrir o explorar. Dicho terreno se va a discretizar en dos grafos. En primer lugar, el grafo del terreno denotado G= (N, E), donde N representa el conjunto de los nodos de dicho grafo que son las diferentes áreas conexas del terreno; y E representa el conjunto de las aristas, que simbolizan la vecindad entre dichas zonas coexas. Hay numerosos sistemas para discretizar terrenos, por ejemplo, en forma de mallado (grid) que es el método que vamos a usar en el algoritmo, o mediante triangulaciones. En segundo lugar, tenemos el grafo de conectividad, que lo expresamos mediante Gc= (N, Ec), donde Ec representa el conjunto de las aristas que determinan que las celdas del mallado del terreno son visibles entre sí [2].

En este problema, se considera que la distancia máxima a la que los drones son capaces de permanecer conectados (rango de visibilidad), es infinita, y por tanto, los aparatos electrónicos que intervienen en la conectividad de los diferentes robots aéreos, tienen una cobertura ilimitada. Así que dos UAVs son visibles entre sí, si hay una línea de visión entre ellos en la que no interfieran obstáculos. Véase la figura 2.1.

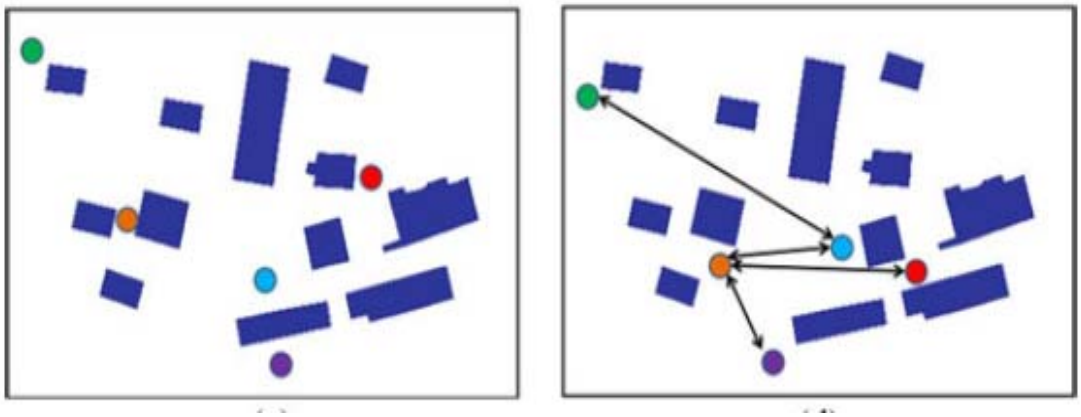


Figura 2.1. En la imagen de la izquierda, podemos observar a los robots en el intervalo de tiempo en el que están desconectados. Después de Ti unidades de tiempo, (imagen de la derecha), deben tener una línea directa de visión entre ellos. Figura tomada de [2].

Otra consideración inicial es el tiempo Ti, que es una constante dada en el problema, pero su valor es importante a la hora de analizar la calidad del cubrimiento del terreno. En el caso de la existencia de un obstáculo de gran tamaño en el terreno, puede ocurrir que el tiempo de desconexión no sea suficiente para rodear el obstáculo, y así, los robots tendrían que volver sobre el camino ya realizado para encontrar un punto de encuentro donde tengan visibilidad entre ellos. Por otro lado, se va a suponer para el desarrollo del problema, que más de un robot pueden ocupar la misma celda o espacio simultáneamente y que el riesgo de colisión es mínimo.

Finalmente, hay que considerar también el costo de cada arista del grafo, es decir el costo de una arista será el tiempo empleado en visitar nodos adyacentes de dicho grafo. En [2] se considera que cada paso tiene costo unidad de manera que si A_K es el camino planificado por el robot 'k' desde su posición inicial hasta el punto donde se conecta con los demás robots, C (A_K) $= T$. Donde C es una función que representa el coste total del camino establecido por dicho robot, y T es el tiempo de desconexión fijado. Así cada paso del dron, vale exactamente una unidad de tiempo, que corresponde al coste de una arista en el grafo.

2.2. Definición del problema.

Varios drones parten de una configuración inicial A (0), donde A representa la ruta de cada robot mediante la unión de los diferentes nodos que visita, de forma que, en la posición inicial están todos conectados. Así, cada Ti unidades de tiempo, todos los robots deben situarse en puntos en los que exista visibilidad entre ellos y puedan conectarse. Primeramente, se ha discretizado el terreno en un grafo G= (N, E), y cada vez que los robots realicen un paso, se actualiza dicho grafo teniendo en cuenta los nodos y aristas que se hayan visitado.

Una vez que parten de dicha configuración inicial, se inicia el intervalo de desconexión de cada uno de los vehículos. El objetivo de éstos en este intervalo es moverse por el terreno de forma independiente maximizando una función F que suponemos que es conocida y determinista. F puede ser de la siguiente forma:

- Una función que indica la probabilidad de encontrar un objetivo en cada punto del terreno, de forma que sería una función probabilística.
- Una función que indica la cantidad de información que los robots pueden obtener en cada punto del terreno.
- Una función que nos da la cantidad de área cubierta por cada robot en cada punto del terreno.

La propuesta de [2] es que los robots en su intervalo de desconexión no se mueven de forma arbitraria, sino que se mueven de forma coherente a una función objetivo que nos permite cubrir o explorar el terreno teniendo una finalidad definida.

Así, la formulación del problema, sabiendo que A_K es la ruta planificada por el robot 'k', y F es la función objetivo, sería la siguiente:

Problema: Dados K vehículos aéreos no tripulados, los grafos del terreno y de visibilidad G=(N,E), Gc=(N,Ec), el intervalo de tiempo de desconexión Ti y la función F. Encontrar la ruta de cada robot A_K que maximice dicha función Fcon la restricción de que se conecten cada Ti unidades de tiempo.

El algoritmo de [2] intenta planificar en cada tiempo de desconexión, una ruta eficiente, maximizando la función F que como se ha comentado anteriormente, es una función conocida y determinista asociada a cada punto del terreno.

Para ello, proponen un heurístico, denominado *Implicit coordination algorithm* cuyo objetivo es encontrar de forma eficiente, configuraciones donde los drones estén en conexión tras Ti unidades de tiempo, y la realización de una exploración exhaustiva del terreno en el intervalo de tiempo que permanecen desconectados. La figura 2.2 ilustra el intervalo de desconexión de los vehículos.

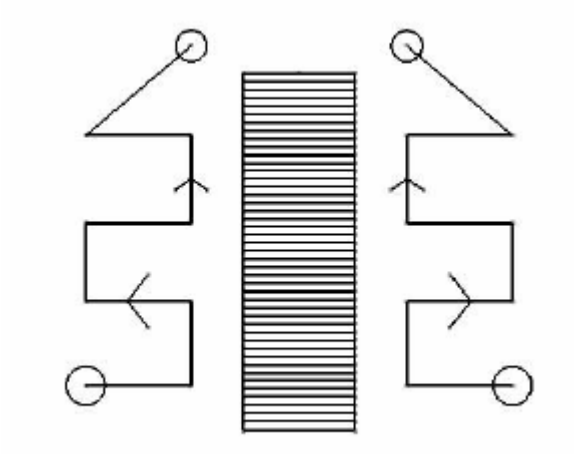


Figura 2.2. En esta figura se observa claramente cómo los robots exploran independientemente el terreno alrededor de un obstáculo hasta que finaliza el intervalo de desconexión Ti, instante en el que deben estar conectados (parte superior de la imagen).

2.2.1. Algoritmo de coordinación implícita (MIPP-PC).

El heurístico que [2] expone, consta de los siguientes pasos:

Input: Tiene como entradas el número de drones, los grafos del terreno y de conectividad, la función objetivo F, el tiempo de desconexión Ti, y la configuración inicial.

Paso 1: El primer paso del algoritmo es el siguiente: Se analiza para el primer vehículo posibles caminos y rutas completadas en Ti unidades de tiempo.

Paso 2: Una vez estudiadas diversas rutas completadas por dicho vehículo en Ti unidades de tiempo, se descartan aquellas rutas en las que en el instante Ti, el dron se sitúa en una posición en la que no tiene visibilidad con los demás.

Paso 3: De aquellas rutas posibles que no han sido descartadas en el paso 2, se elige aquella en la que el valor de la función F dada es mayor.

Paso 4: Se marcan los nodos del grafo como visitados, y se actualizan las diferentes variables.

Paso 5: Se inicia el ciclo con el siguiente vehículo.

De esta manera, el primer dron parte de la configuración conectada inicial, e inicia su recorrido. Empieza su ruta maximizando la función F, y en el instante Ti, debe situarse en un punto donde tenga una línea de visión con los otros robots que se encuentran estáticos en su posición anterior. Podemos observar una iteración del algoritmo en la figura 2.3.

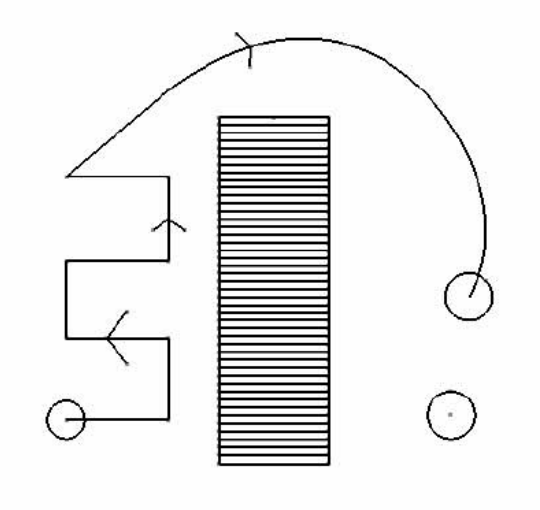


Figura 2.3. En esta figura se representa un ciclo del algoritmo expuesto en [2], en el que el primer robot sale, explora y cubre maximizando la función F en su intervalo de desconexión, y en el instante Ti vuelve para buscar una línea de visión con el segundo dron.

En este heurístico, se asume que se ha dado un tiempo Ti suficiente como para que todos los robots alcancen el estacionario [2], es decir, que tras Ti unidades de tiempo, el dron pueda situarse en un punto donde tenga visibilidad con los demás. Por otro lado, este algoritmo es un algoritmo off-line, mientras el primer dron explora o cubre el terreno de, busca una ruta que tras Ti unidades de tiempo, conecte con los demás vehículos que se encuentran estacionarios.

Cada iteración del algoritmo requiere tantas optimizaciones de la función F como número de drones tenemos disponibles para la exploración. Según el punto de vista computacional, el algoritmo es exponencial, y si todos los caminos posibles que pueden escoger los robots pueden alcanzar longitudes de Ti unidades de tiempo, el tiempo de computación es $O(K * b^{Ti})$ donde 'b' es un coeficiente propio del grafo, K es el número de drones y O es una función que nos proporciona el orden de magnitud.

2.2.2. Resultados de los diferentes experimentos.

Para probar la calidad y eficiencia del algoritmo anteriormente expuesto se realizaron experimentos con un par de robots aéros, los cuales planificaban su propio recorrido conectándose de forma periódica. Así se discretizó un terreno con obstáculos con un grafo usando diferentes métodos de discretización expuestos en [5]. La discretización del terreno podemos observarla en la figura 2.4

El objetivo de los robots es cubrir rápidamente el terreno o localizar un objetivo de forma que mantengan la conectividad periódica. Según este algoritmo, en un terreno con no

muchos obstáculos, aseguran que se realiza la cobertura completa del terreno en unos 3 minutos. Y exponen que, en un grupo de más robots, según los experimentos, realizar la tarea expuesta, sería cuestión de segundos.

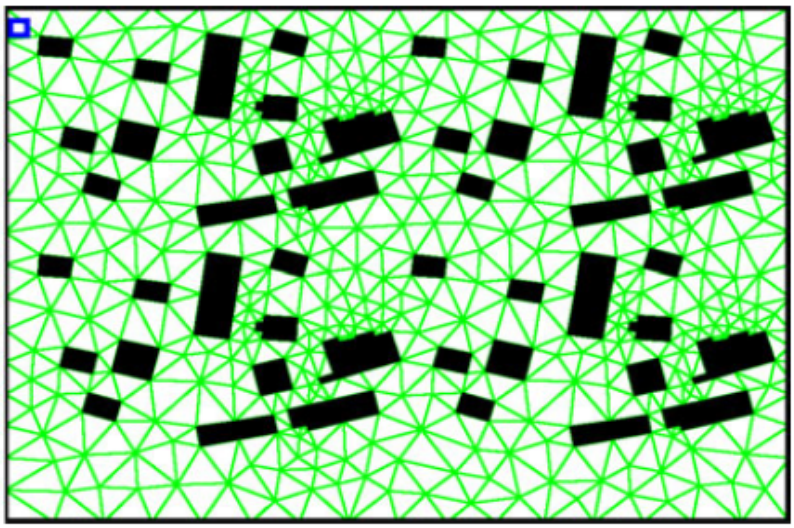


Figura 2.4. En esta imagen tomada de [2], podemos observar, un terreno cuyos obstáculos podemos ver en color negro, que se ha discretizado en un grafo mediante triangulación.

2.3. Inconvenientes y escenarios desfavorables del algoritmo

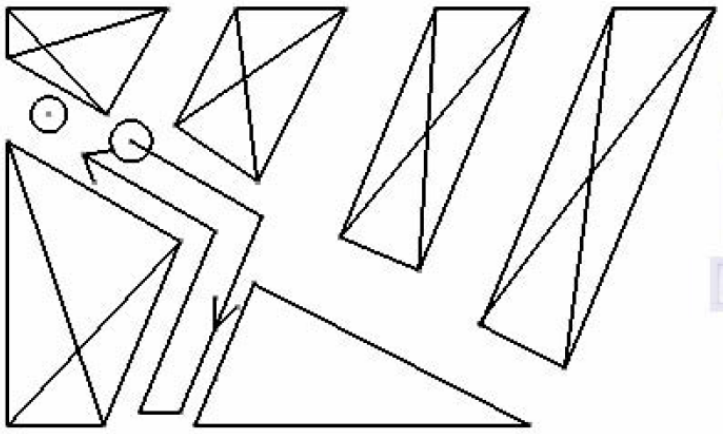
Una vez resumido el algoritmo que exponen en [2], procedemos a analizarlo y a ver aspectos críticos y escenarios donde es realmente ineficiente aplicarlo.

1. Los robots no se mueven a la vez, es decir, cuando el primero empieza su intervalo de desconexión, busca al segundo (en el caso que el número de vehículos no tripulados sea 2) el cual permanece estático a que el primer vehículo se conecte con él en un tiempo menor o igual que Ti unidades. Instante en el que el segundo empieza su intervalo de desconexión. En terrenos con obstáculos de gran tamaño, los vehículos apenas podrían avanzar, ya que el primero empezaría la ruta, pero posiblemente tendría que volver sobre áreas de terreno que ya había visitado para que en el instante Ti esté en conexión con el otro dron que permanece estacionario. Por lo tanto, se tardaría una gran cantidad de tiempo en cubrir el terreno de forma exhaustiva. Véase la figura 2.5.

   Alternativa: La idea que vamos a exponer en nuestro algoritmo es que los vehículos aéreos exploren o cubran el terreno de forma simultánea. De esta forma, realizaríamos la tarea programada en la mitad de tiempo, en el caso de que usáramos 2 drones y en 1/K unidades de tiempo en el caso que usáramos K drones.

   Esto supone un ahorro tanto en combustible, como de energía consumida. Así, aparte del coste del tiempo ahorraríamos cantidades significativas de recursos.

(I)



(II)

Figura 2.5. En esta ilustración, podemos observar cómo, debido a los numerosos obstáculos del terreno (I), el primer dron, parte cubriendo el terreno planificado, pero es obligado a volver sobre su camino recorrido (II), debido a la restricción que provocan éstos.

2. En la práctica, la cobertura de los drones es limitada, es decir, la distancia a la que estos vehículos pueden comunicarse e intercambiar información, a diferencia de lo que se expone en [2], no es infinita, tiene limitaciones debido a que los sistemas de aviónica de los UAVs no pueden llevar equipos que emitan y reciban señales muy potentes, por las limitaciónes de peso asociadas al diseño estructural del vehículo y de interferencia electromagnética. Por lo tanto, no es correcto suponer que dos robots están conectados si no hay ningún obstáculo que restrinja la línea de visibilidad entre ambos.

   Alternativa: Lo que vamos a proponer en nuestro algoritmo es lo siguiente: Para que dos robots puedan conectarse e intercambiar información, deben situarse en celdas contiguas del mallado del terreno o en la misma celda uno encima del otro. De esta forma, estamos restringiendo el alcance de las señales de comunicación emitida por los robots, y, por tanto, estamos limitando la elección de los puntos de encuentro. Véase la figura 2.6.

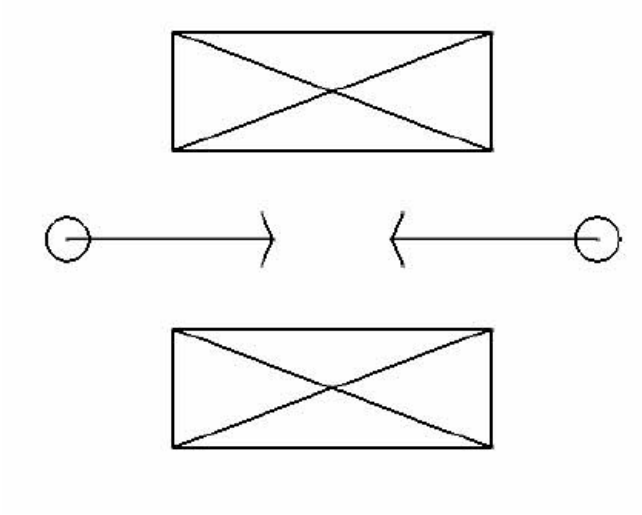


Figura 2.6. En esta ilustración, podemos observar cómo, debido a las limitaciones de cobertura entre dos robots, ambos tienen que acercarse, aunque haya una línea de visión directa entre los dos.

3. Finalmente, otro posible punto crítico de [2], está relacionado con la existencia de *islas,* que son pequeñas zonas conexas sin visitar que pueden dejar los robots mientras cubren las diferentes áreas del terreno. Estas pequeñas zonas pueden ser un inconveniente a la hora de diseñar un cubrimiento eficiente del terreno

   Alternativa: En el algoritmo que explicaremos posteriormente se va a proponer visitar las islas al final de todo el proceso resolviendo el problema del agente viajero [20]. Así, vamos a implementar una aproximación del algoritmo de Christofides [10] para visitarlas de forma ordenada y eficiente. Véase la figura 2.7.

A			
			5
1		4	
2		3	

Figura 2.7. En esta ilustración, podemos observar el orden en que debemos visitar las diferentes islas según el algoritmo de Christofides..Donde A es la posición del robot, las celdas verdes son celdas visitadas y las rojas son obstáculos.

3 -FORMULACIÓN DEL ALGORITMO

Este es el capítulo fundamental del proyecto. Una vez explicado el problema y analizado los puntos débiles del algoritmo propuesto en la literatura, así como ejemplos donde sería muy ineficiente aplicarlo, se va a plantear un algoritmo alternativo que tiene como objetivo principal el cubrimiento de terrenos de forma eficiente. En primer lugar, explicaremos por qué se ha elegido MATLAB como lenguaje de programación, se explicará el objetivo del algoritmo, las diferentes funciones empleadas, y se expondrán los resultados obtenidos.

3.1. ¿Por qué MATLAB?

Para implementar el algoritmo que hemos diseñado, se ha usado el programa MATLAB Rb2014 . Este programa usa un lenguaje de alto nivel orientado al desarrollo de cálculos técnicos. Integra cálculo, visualización y programación en un entorno interactivo de fácil manejo.

Se ha escogido éste como lenguaje de programación debido a su rapidez operativa, su claridad en el lenguaje de programación para tratar con vectores y matrices (ya que vamos a manejar grafos, que en la práctica supone operaciones con matrices), su gran eficiencia en tiempo de computación, y su interfaz gráfica de fácil manejo, que nos permite programar de forma ordenada y estructurada diferentes funciones [8].

3.2. Objetivo y restricciones del algoritmo.

Primeramente, vamos a partir de una discretización del terreno en forma de malla, en la que los vehículos pueden compartir información cuando están situados en celdas contiguas o en la misma, suponiendo que un dron puede situarse encima o debajo de otro. Un mallado es una descomposición del terreno que queremos explorar en una colección de celdas uniformes. Típicamente, las celdas son representadas como rectángulos, distinguiendo aquellas que representan los obstáculos y las que representan las áreas del terreno visitables. De esta forma, conseguimos una aproximación del terreno y sus obstáculos [3]. Véase figura 3.1.

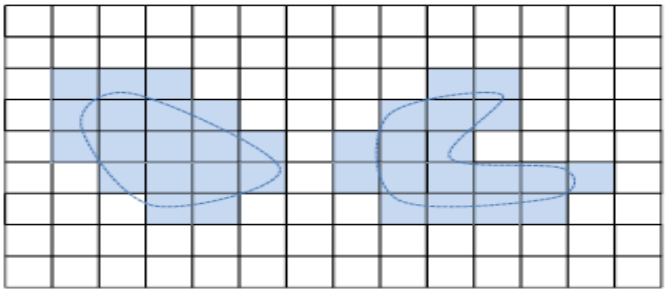


Figura 3.1. Esta imagen representa un mallado del terreno. Imagen tomada de [3].

Por simplicidad, consideramos el caso de dos robots, que inicialmente se sitúan en casillas contiguas del grid estando conectados. En nuestro algoritmo los drones se van a mover simultáneamente con el objetivo de cubrir todo el terreno en el menor tiempo posible, minimizando el número de áreas del terreno repetidas. Por otro lado, hay que considerar la restricción de tener que encontrarse periódicamente en celdas adyacentes (cobertura limitada), cada Ti unidades de tiempo.

Así, la formulación del problema que vamos a resolver es el siguiente:

Problema: Dados 2 vehículos aéreos no tripulados, el mallado del terreno, y el tiempo de desconexión Ti, cubrir todo el terreno, minimizando el nº de celdas repetidas (que conlleva una optimización del tiempo total de cubrimiento). Con las restricciones de encuentros periódicos cada Ti unidades de tiempo y considerando conectividad sólo en celdas adyacentes.

La metodología va a ser ir evaluando todos los pares de celdas vecinas en el terreno, de forma que, usando el algoritmo de Dijkstra [9] (véase figura 2.2.) los drones lleguen a éstas de forma simultánea en tiempo menor o igual que Ti unidades. La forma de evaluar los potenciales puntos de encuentro de los robots será clave en el método propuesto, y se explicará en el siguiente apartado.

Este método nos va a permitir relizar un recorrido eficiente en muchos tipos de terrenos, por ejemplo, en terrenos alargados con muchos obstáculos, donde el algoritmo de [2] sería muy ineficiente. Además, en terrenos con grandes obstáculos, este algoritmo nos permite rodearlos de forma rápida y eficiente, como explicaremos en el siguiente apartado del proyecto.

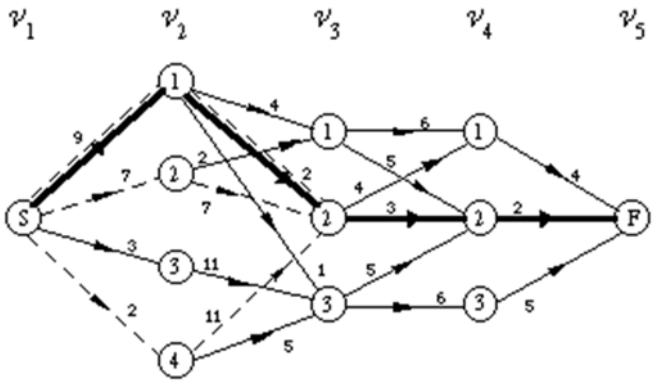


Figura 3.2. Esta figura es una representación de un recorrido directo mediante el método de Dijkstra.

3.3. Descripción del algoritmo y funciones.

Como ya hemos explicado, partimos de dos drones o vehículos aéreos no tripulados, en casillas contiguas del grid, es decir en puntos del terreno, en los que están conectados. Ambos saldrán simultáneamente de su posición de partida, con el objetivo de encontrar otro punto de encuentro, el cual se alcance en menos de Ti unidades de tiempo por vehículo, con el objetivo principal la mayor cantidad del terreno en el menor tiempo posible. Para ello, minimizaremos el número de unidades de áreas del terreno visitadas en

más de una ocasión.

Así, se analizan todas las celdas vecinas visitables tanto verticales como horizontales, y también, cada celda de forma independiente suponiendo que los drones pueden conectarse uno encima del otro. Posteriormente, se obtiene el camino mínimo de Dijkstra de cada robot a todos los pares de celdas que queremos evaluar y se estudia si ambos vehículos pueden llegar a ellas en un tiempo menor o igual que Ti unidades. Si la respuesta es afirmativa, se almacena ese punto de encuentro como potencialmente posible.

Una vez que se han obtenido todos los pares de celdas posibles donde los drones pueden conectarse en menos de Ti unidades de tiempo, el algoritmo elige un punto de encuentro de acuerdo a una función creada llamada *nota_arista.m* que posteriormente explicaremos y que establece una puntuación diferente a cada uno de acuerdo a diferentes parámetros tales como, el número de celdas nuevas que visita en su recorrido, el número de islas remanentes o el número de *pasillos*, es decir, zonas alargadas visitables donde cada celda tiene como máximo 2 vecinos.

El estudio de los diferentes puntos de encuentro potenciales, se realiza siguiendo un patrón zigzag en el terreno, como podemos observar en la figura 3.3. con las funciones *waypoint_horizontal.m, waypoint_vertical.m, y waypoint_mismopunt,* que desarrollaremos en la sección 3.3.2.

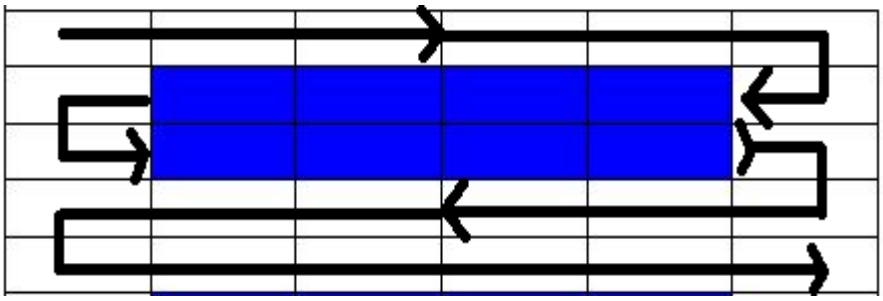


Figura 3.3. En esta imagen, podemos observar una representación de la forma en que nuestro algoritmo evalúa puntos de encuentro.

Un elemento adicional añadido al algoritmo es que en las diferentes islas que vayan quedando en el terreno, se calcule el recorrido de distancia mínima para visitarlas todas mediante el algoritmo del Christofides [10] que calcula, el camino mínimo en que visita todas, sin repetir ninguna. Además, en la función que posteriormente explicaremos *nota_arista.m* tendremos en cuenta la distancia total en recorrer todas las islas que quedarían por visitar en el mallado .

Así, el algoritmo consta de los siguientes pasos:

Input: Tiene como entradas el número de drones, el mallado del terreno, el tiempo de desconexión Ti, y la configuración inicial.

Paso 1: Se analizan todos los pares de celdas visitables del mallado, tanto horizontales como verticales y se calcula el camino mínimo de Dijkstra de los 2 vehículos simultáneamente a dichas celdas. (wp.m)

Paso 2: Se almacenan los puntos de encuentro como potencialmente seleccionable si cada robot emplea para llegar a éstas celdas según el camino de Dijkstra un tiempo menor o igual que Ti unidades. (Wp.m)

Paso 3: De todos esos puntos de encuentros posibles se escoge aquel que obtenga mayor puntuación en la función nota_arista, atendiendo a elementos como islas o pasillos. (nota_arista.m)

Paso 4: Se actualiza la posición de los robots y se reinicia el proceso. (wp.m)

Paso 5: Una vez visitado la mayor parte del terreno, se emplea el algoritmo de Christofides para visitar todas las pequeñas islas remanentes. (getlabeledminpath.m)

Podemos ver un diagrama de flujo del algoritmo en la figura 3.4

Una vez explicado los puntos fundamentales del algoritmo, procedemos a explicar las diferentes funciones y programas de MATLAB que intervienen.

3.3.1. *Función principal.*

Es el programa base del algoritmo, denominado en el código de MATLAB *main.m* el cual es encargado de llamar a las demás funciones que posteriormente explicaremos y operar con sus variables. Tiene como entradas el escenario, es decir el mallado del terreno, el tiempo de desconexión Ti y las celdas del grid donde se encuentran los vehículos no tripulados inicialmente.

Así, el programa recibe la matriz del grafo del terreno, que contiene un '0' en las casillas no visitadas y un '2' en los obstáculos. Esta matriz corresponde al grafo G= (N, E), y es la que emplearemos para realizar las diferentes operaciones y algoritmos.

Después de las diferentes inicializaciones, este programa inicia un ciclo que finaliza cuando se ha cubierto todo el terreno. Primeramente, se marcan las casillas iniciales como visitadas, en segundo lugar, se establecen las zonas que quedan sin visitar del terreno y el orden en que deben ser visitadas por los robots, resultado de la función que implementa el algoritmo de Christofides: *getLabeledMinPath.m.* Después, se llama a la función *wp.m* que es la que halla el punto de encuentro optimo, y el camino mínimo seguido según el algoritmo de Dijkstra. Finalmente, se actualiza el grafo, y se representa gráficamente el camino seguido, con la función *dibuja_terreno.m.*

Además, para estudiar la calidad de la aplicación del algoritmo sobre un terreno determinado, se estudia en este programa el número de celdas repetidas.

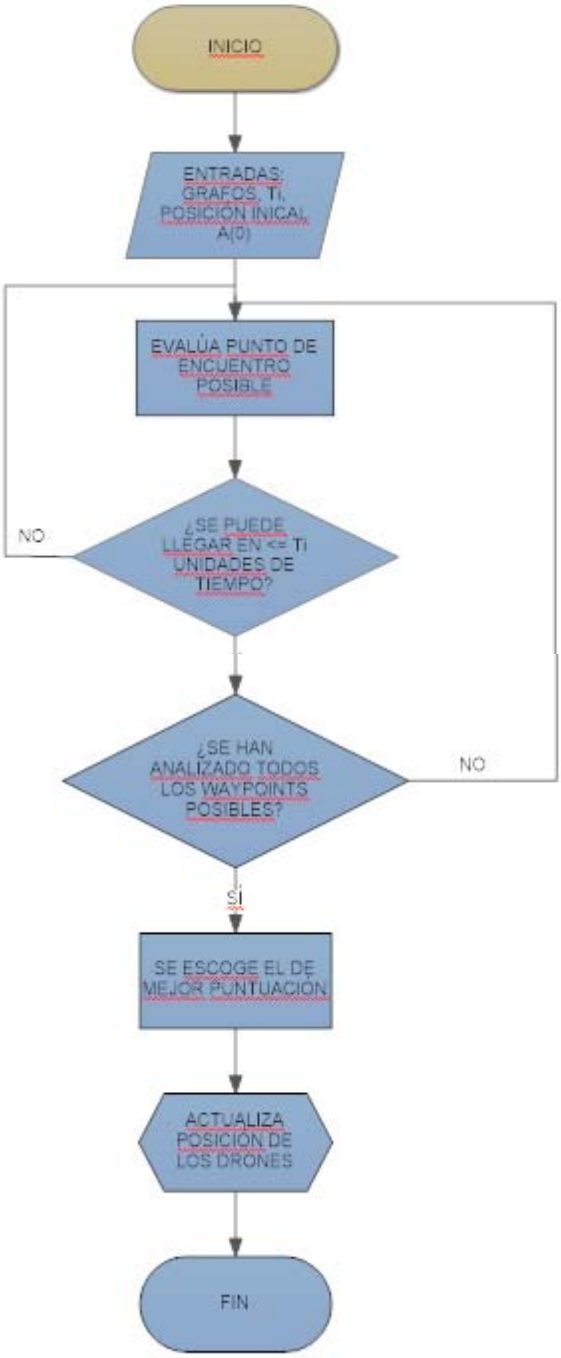


Figura 3.4. Esta figura representa el diagrama de flujo del algoritmo diseñado.

3.3.2. *Función que estudia de los diferentes puntos de encuentro.*

Estas funciones, denominadas *wp.m, wp_vertical.m y wp_mismopunto.m,* son llamadas por el programa anteriormente descrito *main.m* dentro del bucle. Este programa nos permite encontrar los puntos de encuentro cada Ti unidades de tiempo que mayores puntuaciones tengan dentro de la función *nota_arista.m.*

Esta función, recibe como entradas las celdas donde se encuentran los dos robots en el instante actual, la matriz actualizada del terreno, con un ‘1’ en las celdas que ya han sido visitadas, el orden de visita de las diferentes zonas conexas del terreno que por el algoritmo del Christofides es óptimo visitar, el límite de tiempo de desconexión, el vector de pesos, que tiene el tamaño del número de celdas de la matriz G,(inicialmente es un vector de unos y duplica su valor en el caso de que un robot haya visitado la celda de la matriz correspondiente al lugar que ocupa en dicho vector, y finalmente, la matriz L (Producto del programa que implementa el algoritmo de Christofides), que es una matriz de etiquetas donde cada zona está etiquetada con un número correspondiente al orden en el que ha de ser visitada. Véase la figura 3.5.

1	A	B	
1			2
1			2
1	1		

Figura 3.5. Esta figura es una representación de la matriz L de etiquetas en el terreno. Las celdas oscuras son de obstáculos, las claras de zonas visitadas, A, B son las posiciones actuales de los vehículos. Podemos observar las etiquetas en el terreno, refiriéndose 1 a la primera zona que ha de visitarse y 2 a la segunda, siguiendo el algoritmo aproximado del viajante.

Por otro lado, este programa, devuelve las rutas de los dos vehículos en cada intervalo de desconexión, una vez escogidos los puntos de encuentro de forma óptima, además de las diferentes matrices ya actualizadas, y la nueva posición de los dos drones.

Ahora, explicaremos los diferentes pasos de la función sin entrar en detalle de código, denominado wp.m en el apéndice A.

1. Empezamos a recorrer evaluar las celdas no visitadas del mallado del terreno (matriz G) en zigzag, de izquierda a derecha y de arriba hacia abajo.
2. Buscamos pares de celdas, tanto horizontales como verticales o coincidentes que hayan sido etiquetadas por la función *getLabeledMinPath.m* y su valor coincida con la primera zona a visitar que haya resultado esta función.
3. Establecemos con la función Dijkstra.m el camino mínimo de cada dron hacia estas celdas (el orden de los caminos no es arbitrario), duplicando el **costo** de las celdas que visita el primer robot para que el segundo evite las celdas del primero.
4. Se pregunta si el tiempo empleado por los dos robots para completar la ruta no se ha pasado del límite de tiempo establecido, y en caso afirmativo, se estudia la calidad de dicho punto de encuentro con la función *nota_arista.m* y se almacena como potencialmente posible.
5. De entre todos los puntos de encuentro posibles, tanto horizontal vertical o uno sobre el otro, se escoge aquel que mejor puntuación tenga, y se actualizan todas las variables del problema.

Ahora, vamos a analizar las diferentes funciones que emplea wp.m

3.3.3. Función que implementa el algoritmo de Dijkstra.

Esta función, denominada Dijkstra.m es una implementación en código MATLAB de lo que se conoce como el método del camino mínimo o método de Dijkstra. Es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices de un grafo con pesos en cada arista. [9]

Dado un grafo G=(N,E), a cuyas aristas se han asociado una serie de pesos, se define el camino de coste mínimo entre dos nodos del grafo,como el camino donde la suma de los pesos de las aristas que lo forman es la más baja entre las de todos los caminos posibles entre esos dos puntos. El algoritmo de Dijkstra es un algoritmo eficiente cuyo tiempo de computación está en el orden de n^2 donde n es el número de nodos del grafo, que sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos de dicho grafo. Fue diseñado por el holandés Edsger Wybe Dijkstra en 1959. El fundamento sobre el que se asienta este algoritmo es el principio de optimalidad: si el camino más corto entre los vértices u y v pasa por el vértice w, entonces la parte del camino que va de w a v debe ser el camino más corto entre todos los caminos que van de w a v. [13]. De esta manera, se van construyendo sucesivamente los caminos de coste mínimo desde un vértice inicial hasta cada uno de los vértices del grafo, y se utilizan los caminos conseguidos como parte de los nuevos caminos.

Así, este programa recibe como entradas los puntos iniciales y finales entre los que se quiere hallar el camino mínimo, además de la matriz que representa el terreno y el vector de pesos. Y como salidas tiene la ruta mínima seguida.

De esta forma, a la hora de rodear obstáculos de grandes longitudes saliendo ambos vehículos de forma simultánea, este algoritmo, usando Dijkstra, nos permite conseguir el camino más eficiente. Véase figura 3.6.

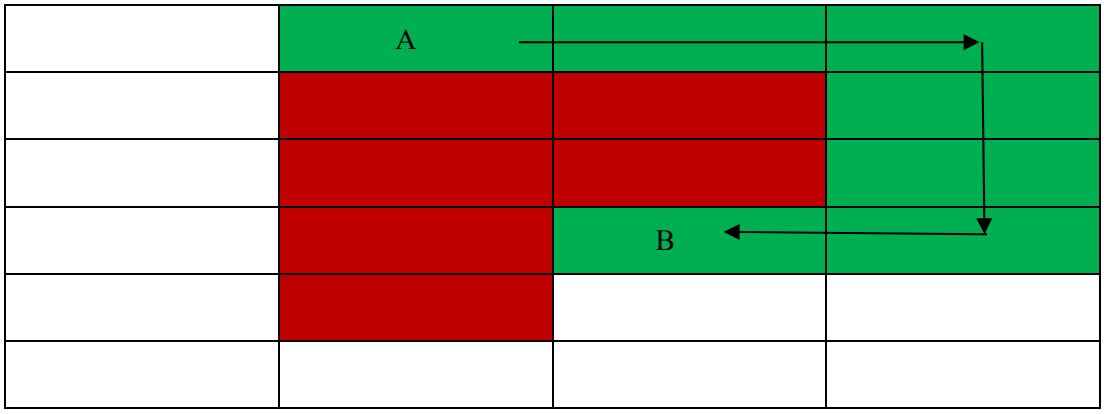


Figura 3.6. En esta figura se ve representado en verde el camino mínimo entre A y B mediante el método de Dijkstra.

Los pasos de este algoritmo son los siguientes:

1. Se inicializa un vector de distancias (que nos dice la distancia desde el origen hasta el nodo que estemos estudiando) con un valor infinito, ya que son desconocidas al principio, exceptuando la del nodo origen al nodo origen que es 0.

2. Marcamos la posición inicial de los drones como la actual.
3. Almacenamos todos los vecinos del nodo en el que estamos actualmente en una cola.
4. Calculamos la distancia del origen a cada vecino pasando por el actual.
5. Marcamos como completo el nodo que estamos estudiando actualmente.
6. Tomamos como próximo nodo actual el de menor valor en de la distancia hallada en 4 y volvemos al paso 3 mientras existan nodos no estudiados.

3.3.4. *Función que evalúa la calidad de cada punto de encuentro.*

Esta función denominada *nota_arista.m,* es la que nos permite cubrir el terreno de una manera eficiente. Es un heurístico cuyo objetivo es que, al ir buscando puntos de encuentro, se penalicen o beneficien diferentes situaciones estudiando la parte del terreno que hemos visitado y lo que nos queda por visitar. Así, se irá sumando o restando puntuación a dicho punto de encuentro de acuerdo a una fórmula que pondere el grado de beneficio o penalización de las diferentes situaciones del problema.

En primer lugar, vamos a estudiar las situaciones que son favorables y desfavorables a la calidad del punto de encuentro.

1. Se penalizará que, al establecer la ruta entre el punto origen y el punto de encuentro, se dejen pequeñas islas y zonas sin visitar en el terreno (véase figura 3.7.), de forma que, será beneficioso que exista una gran zona visitada y otra gran zona sin visitar. En el caso de que se vayan dejando pequeñas islas, al final de todo el proceso de cubrimiento, habría que visitarlas todas una a una y este hecho provocaría que los robots repitieran por zonas que ya habían sido visitadas.

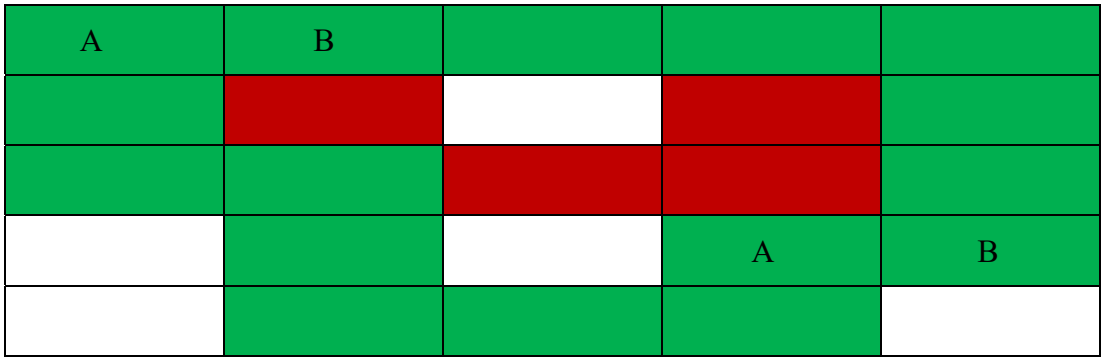


Figura 3.8. En esta figura se ve representado una situación negativa en la que la ruta de los dos vehículos deja pequeñas islas sin visitar.

2. En segundo lugar, será negativo el hecho de que la ruta establecida deje pasillos sin visitar, hecho que provocaría que los vehículos tuvieran que juntarse para recorrerlo, y así gastar combustible y tiempo inútilmente. Como se ha explicado anteriormente, un pasillo es una zona alargada de celdas no

visitadas las cuales tienen como máximo 2 vecinos. (Véase figura 3.8.)

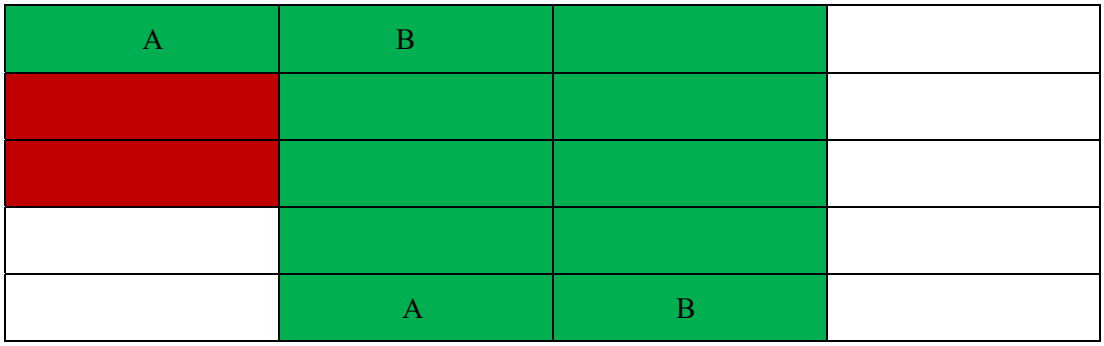


Figura 3.8. Esta figura representa una ruta realizada por los vehículos A y B que dejan un gran pasillo a la derecha del terreno, elemento que penaliza nuestro algoritmo. En verde celdas visitadas y en rojo obstáculos.

3. Se favorece que el número de celdas nuevas visitadas en cada iteración sea el mayor posible, es decir, cuantas más celdas que no habían sido exploradas por los vehículos, se visiten en un tiempo menor o igual que Ti unidades, mayor puntuación tendrá el punto de encuentro.
4. Finalmente, si la ruta seguida por los diferentes robots hacia un punto de encuentro deja varias islas sin visitar, se favorecerá aquel punto de encuentro cuyo recorrido de longitud mínima entre ellas, dado por la función que implementa el algoritmo de Christofides, tenga menor longitud. Para ello, la función *getlabeledminpath.m* nos dará como salida una variable, llamada 'coste' que será la longitud del recorrido mínimo entre todas las islas.

Una vez explicadas las diferentes situaciones que son favorables o desfavorables, la función que nos da la puntuación del punto de encuentro es la siguiente.

$$\boldsymbol{F = CV - e^{2*I-1} - e^{3*L-1} + e^{P-1}},$$

donde CV es el número de celdas nuevas visitadas en cada iteración del algoritmo, I es el número de islas remanentes, L es la longitud del recorrido entre todas las islas resultante de aplicar el algoritmo de Christofides y P hace referencia a la media de todos los vecinos de cada celda visitable, variable que penaliza la cantidad y longitud de los pasillos existentes en el terreno.

Así, hallando el número de zonas conexas del terreno mediante la función recurs.m, penalizamos que existan islas sin visitar en el recorrido, de manera que, a mayor número de islas, la penalización en la nota aumenta de forma exponencial. Por otro lado, a las funciones exponenciales se le resta 1 para que cuando no existan zonas el coste o el grado sea cero, ni penalice ni favorezca en la puntuación.

Por otro lado, el grado afecta al número de pasillos que quedan en el terreno, de forma que, cuantos más pasillos deje el recorrido hacia un punto de encuentro, peor será la calidad de éste.

Finalmente, la variable coste, representa, en el caso de que el recorrido tuviese que dejar islas sin visitar, la distancia del recorrido mínimo de entre todas, dada por la función que establece el algoritmo de Christofides, de esta forma, cuanto mayor es la distancia en dicho recorrido, peor será la calidad del punto de encuentro.

3.3.5. *Función que estudia las diferentes zonas conexas del grafo.*

Estas funciones denominadas recurs.m y meter.m, sirven para establecer una numeración a las diferentes zonas conexas del grafo, resultando una matriz de etiquetas (Figura 3.9.). Así, tiene como entradas la matriz del grafo actualizada, y devuelve 2 variables, una matriz etiquetada de las diferentes áreas conexas, donde cada número representa una zona conexa diferente, y otra variable que te indica el número de estas zonas menos 1.

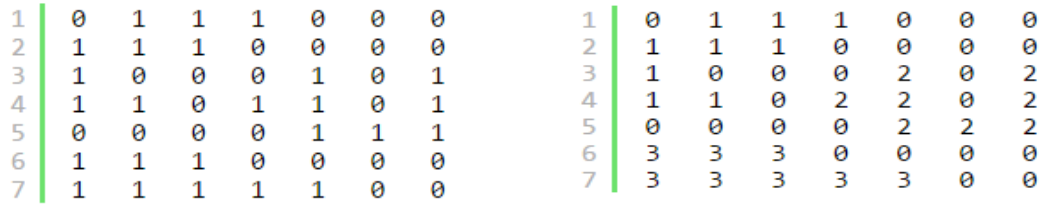


Figura 3.9. A la izquierda podemos ver la matriz del grafo, y a la derecha la matriz de etiquetas una vez usada esta función.

3.3.6. *Función que evalúa los pasillos del terreno.*

Esta función, nos permite evaluar los pasillos de un grafo. Tiene como objetivo estudiar el número de vecinos de todas las casillas que quedarían sin visitar en el mallado del terreno actualizado, y una vez que estudia el número de vecinos de todas estas celdas, se hace una media según la siguiente fórmula: $grado = \frac{nº\ vecinos\ de\ cada\ celda\ \text{sin}\ visitar}{nº\ de\ celdas\ \text{sin}\ visitar}$. De esta forma, si existen pasillos, el máximo número de vecinos es 2 mientras que si no existen, es 4. Así, cuanto mayor sea la media, mayor será la puntuación del punto de encuentro que estemos estudiando.

3.3.7. *Función que implementa el algoritmo de Christofides.*

Esta función, denominada getlabeledminpath, como hemos venido comentando, es la que implementa la aproximación del problema del agente viajero, es decir, tiene como entradas la matriz del terreno, y la posición de los vehículos en ese instante, y da como salida una matriz etiquetada L que identifica las islas no visitadas, el recorrido con distancia mínima entre todas ellas y el coste, es decir, la longitud necesaria para recorrer todas ellas.

En el Problema del Agente Viajero - TSP (Travelling Salesman Problem), el objetivo es encontrar un recorrido completo que conecte todos los nodos de una red, visitándolos tan solo una vez y volviendo al punto de partida, y que además minimice la distancia total de la ruta.

Este tipo de problemas tiene gran aplicación en el ámbito de la logística y distribución, así como en la programación de curvas de producción. [10].

Este problema es NP-duro, la complejidad del cálculo del problema del agente viajero ha despertado múltiples iniciativas por mejorar la eficiencia en el cálculo de rutas. El método más básico es el conocido con el nombre de fuerza bruta, que consiste en el cálculo de todos los posibles recorridos, lo cual se hace extremadamente ineficiente y casi que se imposibilita en redes de gran tamaño. También existen heurísticos que se han desarrollado

por la complejidad en el cálculo de soluciones óptimas en redes robustas, es por ello que existen métodos como el vecino más cercano, la inserción más barata y el doble sentido. Por último, se encuentran los algoritmos que proporcionan soluciones óptimas, como el método de branch and bound (ramificación y poda), que trabaja el problema como un algoritmo de asignación y lo resuelve por medio del método simplex. [10]

Nosotros vamos a usar una aproximación al TSP llamado el algoritmo de Christofides. El algoritmo de Christofides es un algoritmo aproximado que puede cometer un error de hasta el 50 %.

Los pasos del algoritmo son los siguientes:

1.-Se calcula el árbol generador mínimo T del grafo actualizado, que es un árbol compuesto por todos los vértices del grafo, siendo mínima la suma de sus aristas.

2.-Se obtiene el conjunto de vertices de grado impar V_o de dicho árbol.

3.-Se obtiene un apareamiento perfecto de mínimo peso M sobre el conjunto V_o.

4.-Se unen las aristas de M y T para crear el multigrafo H.

5.-Se obtiene, mediante el algoritmo de Fleury [20], un ciclo euleriano en H, (camino que pasa por cada arista una sola vez).

6.- Se obtiene un ciclo hamiltoniano (camino que visita todos los vértices una sola vez) a partir del ciclo euleriano anterior, descartando los nodos repetidos.

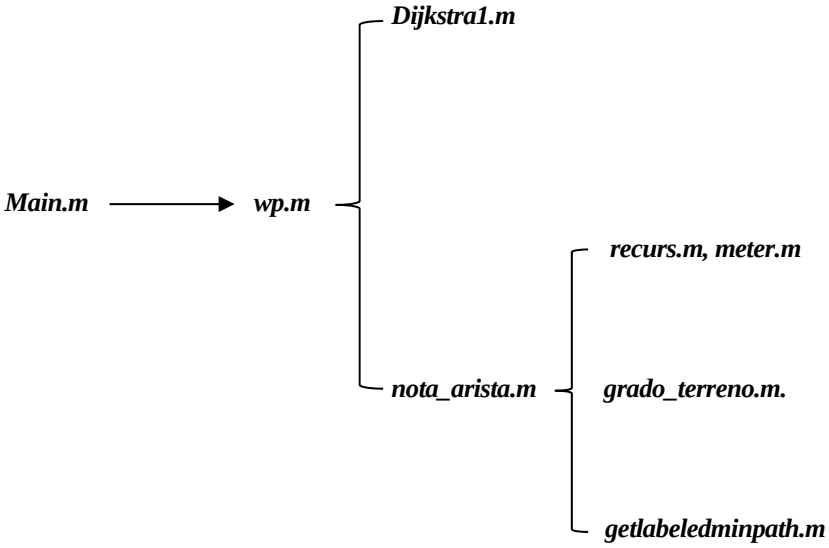


Figura 3.10. Esta figura es un esquema de las diferentes llamadas a las funciones.

4 -SIMULACIONES Y RESULTADOS EN DIFERENTES ESCENARIOS.

En este capítulo, vamos a aplicar el algoritmo expuesto en el capítulo anterior en diferentes escenarios para comprobar su eficiencia

4.1. Escenario 1.

Vamos a partir del siguiente terreno, donde las celdas rojas son obstáculos, y A,B son los vehículos situados en su posición inicial, las celdas visitadas serán representadas de color verde:

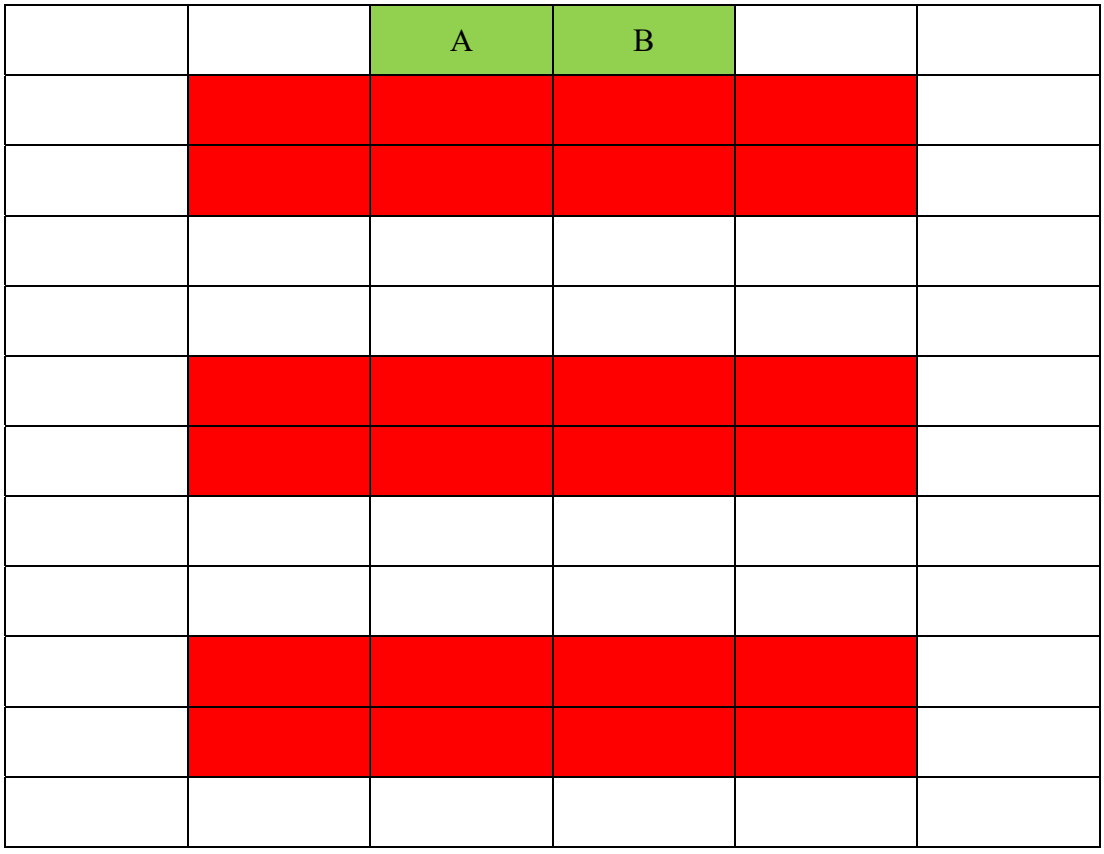


Figura 4.1. Representación del terreno, en rojo obstáculos y en verde casillas visitadas.

Tomaremos Ti=9 unidades de tiempo. Es fácil ver que en 3 iteraciones ha convergido nuestro algoritmo, además de obtener la solución óptima. Las diferentes iteraciones se ilustran en las figuras 4.2 4.3 y 4.4.

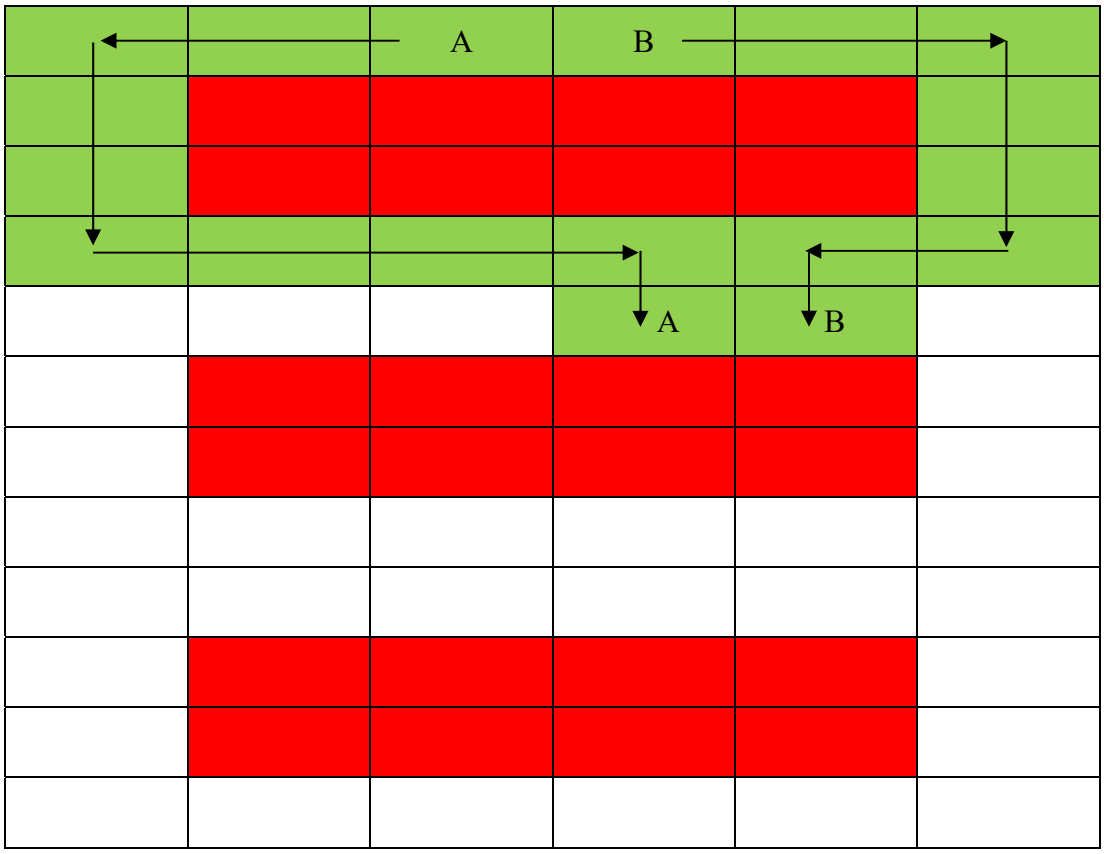


Figura 4.2. Representación del primer intervalo de desconexión Ti, En verde celdas visitadas y en rojo obstáculos

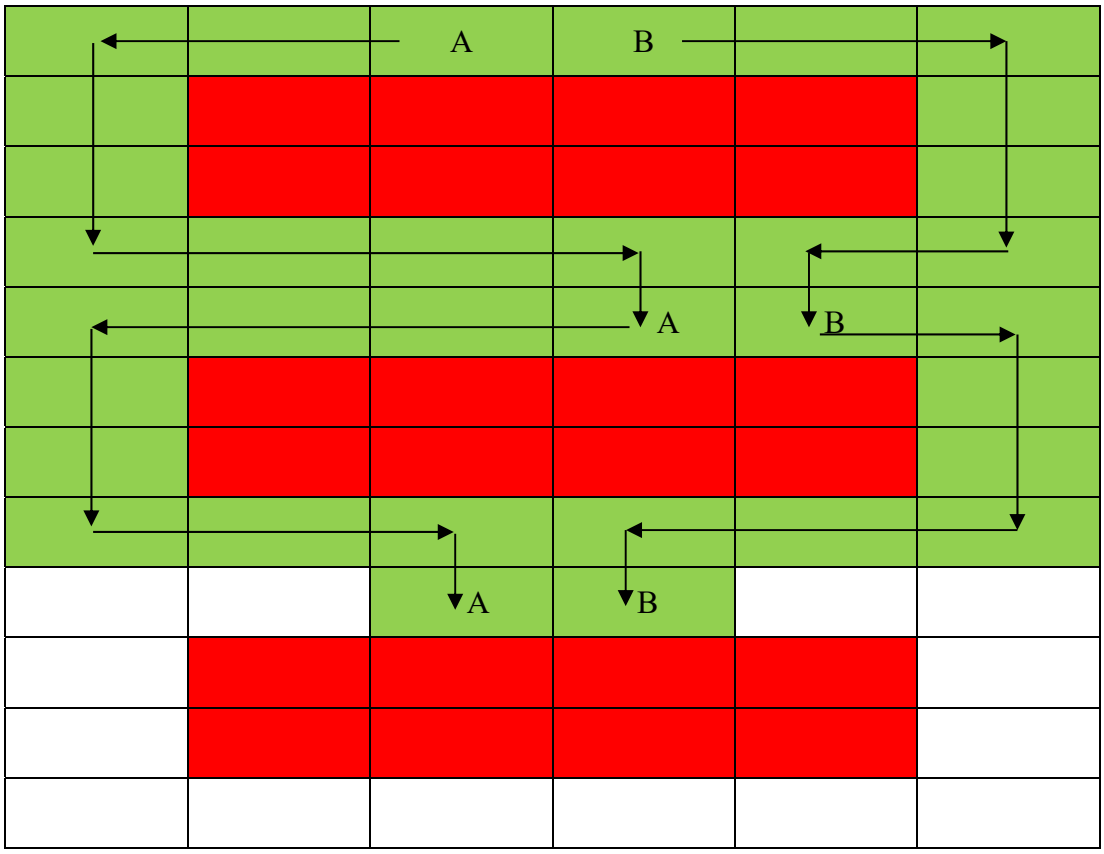


Figura 4.3. Representación del segundo intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

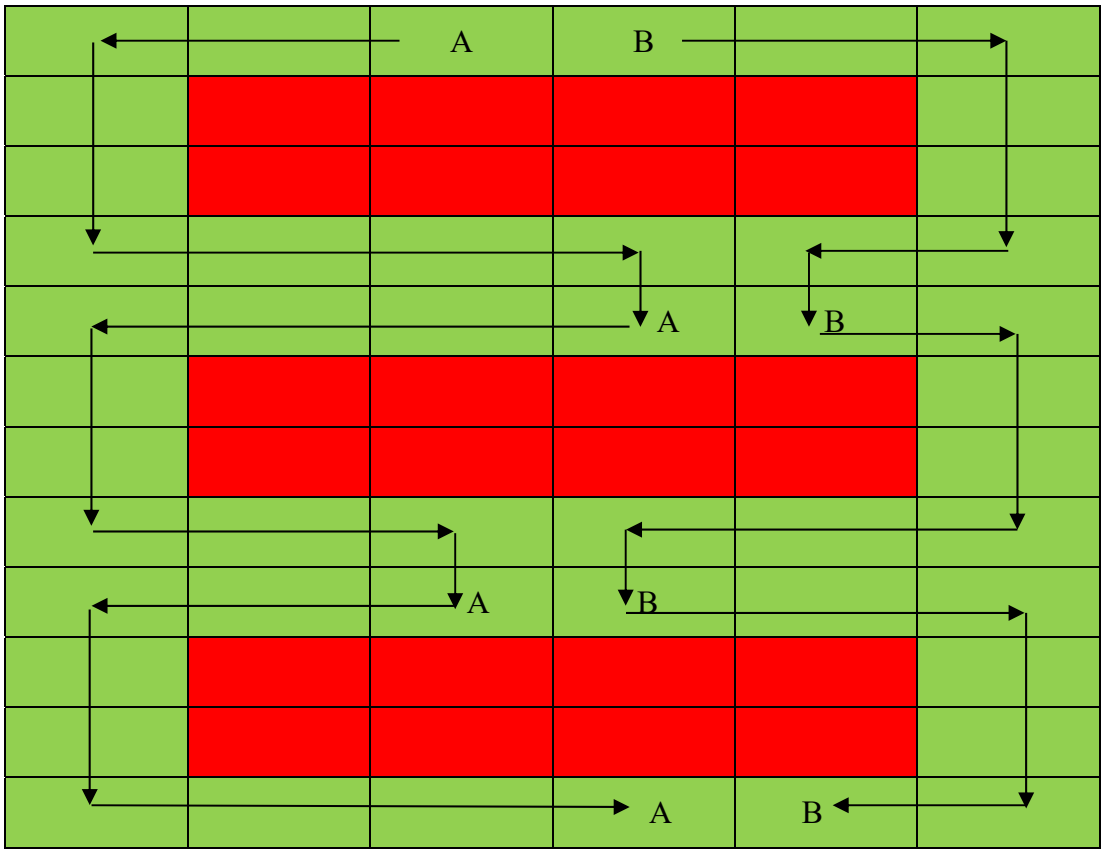


Figura 4.4. Representación del último intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

En este escenario, con grandes obstáculos y poco espacio de maniobra, se llega a la solución óptima en tres iteraciones, con ninguna celda repetida, de forma eficiente, sin dejar ninguna isla en ningún paso del algoritmo.

Tabla 4.1. Resultados de la simulación del escenario 1

Parámetro	***Valor***
Nº de robots	2
Ti	9
Nº islas dejadas en cada iteración	0
Nº pasillos dejados en cada iteración	0
Nº de celdas repetidas	0
Nº de iteraciones	3
Tiempo total de cubrimiento	27

4.2. Escenario 2.

El segundo escenario que vamos a estudiar es el siguiente:

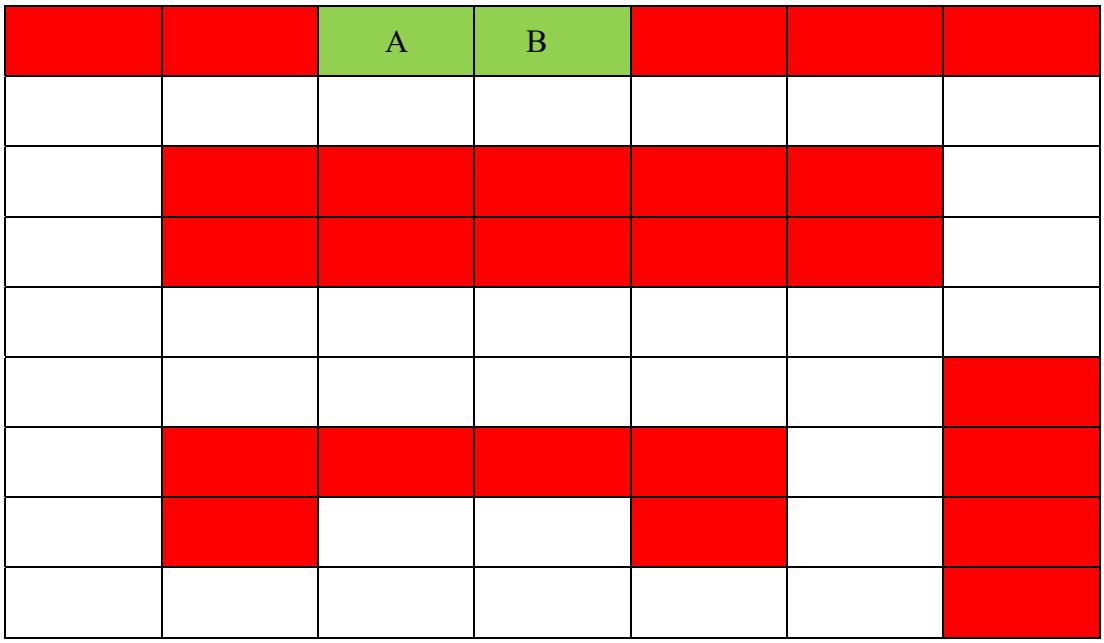


Figura 4.5. Representación del terreno, en rojo obstáculos y en verde casillas visitadas. En verde celdas visitadas y en rojo obstáculos

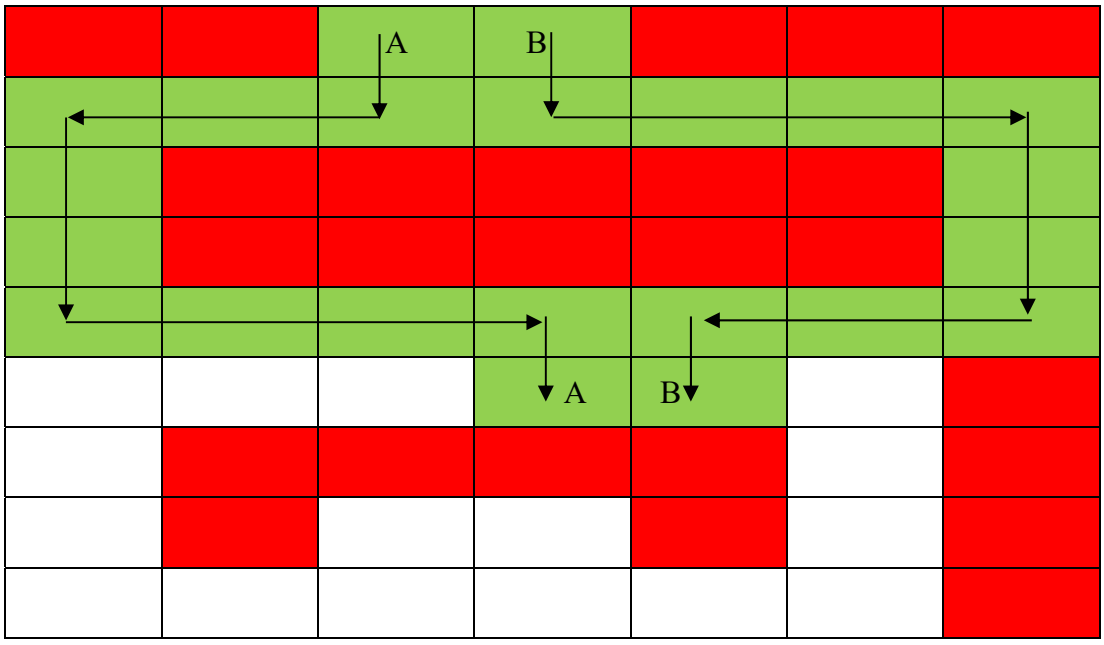


Figura 4.6. Representación del primer intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

Así, estableciendo Ti=9, se llega a la solución óptima, es decir, aquella en la que no se repiten celdas, en 2 iteraciones. Además, gracias a la función nota_arista, los robots no dejan pasillos ni islas.

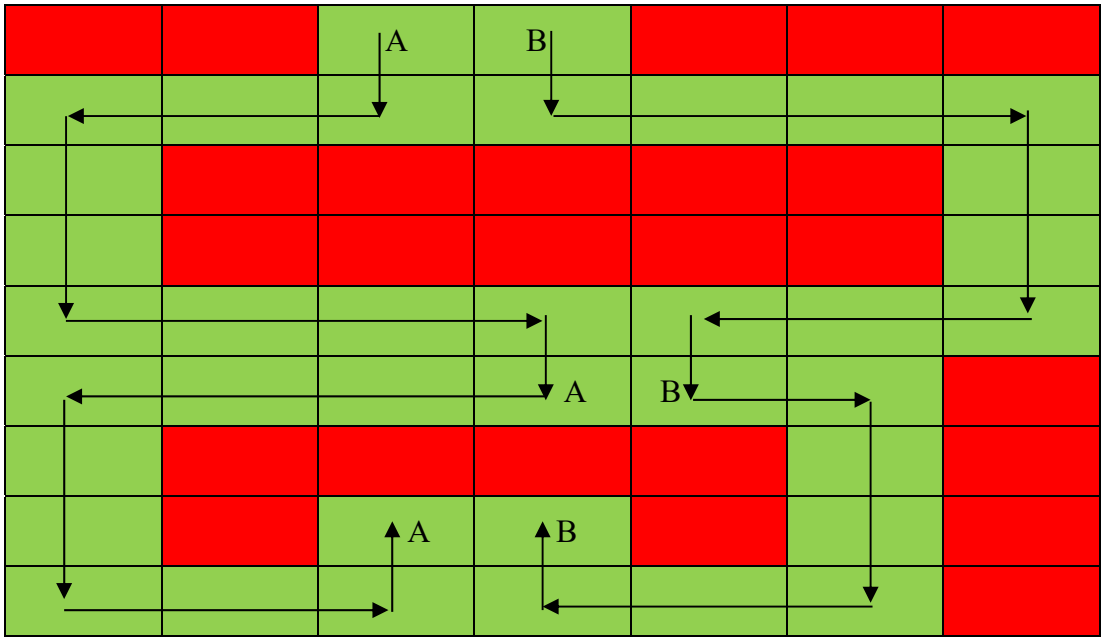


Figura 4.7. Representación del último intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

En este escenario habitual en centros urbanos con calles estrechas y pocos márgenes de maniobra, el algoritmo nos da la solución óptima, que es aquella solución en la que no se repite ninguna celda y por lo tanto, se minimiza el tiempo total de cubrimiento del terreno.

Tabla 4.2. Resultados de la simulación del escenario 2

Parámetro	***Valor***
Nº de robots	2
Ti	9
Nº islas dejadas en cada iteración paso	0
Nº pasillos dejados en cada iteración	0
Nº de celdas repetidas	0
Nº de iteraciones	2
Tiempo total de cubrimiento	18

4.3. Escenario 3.

Vamos a evaluar un escenario un poco más complejo.

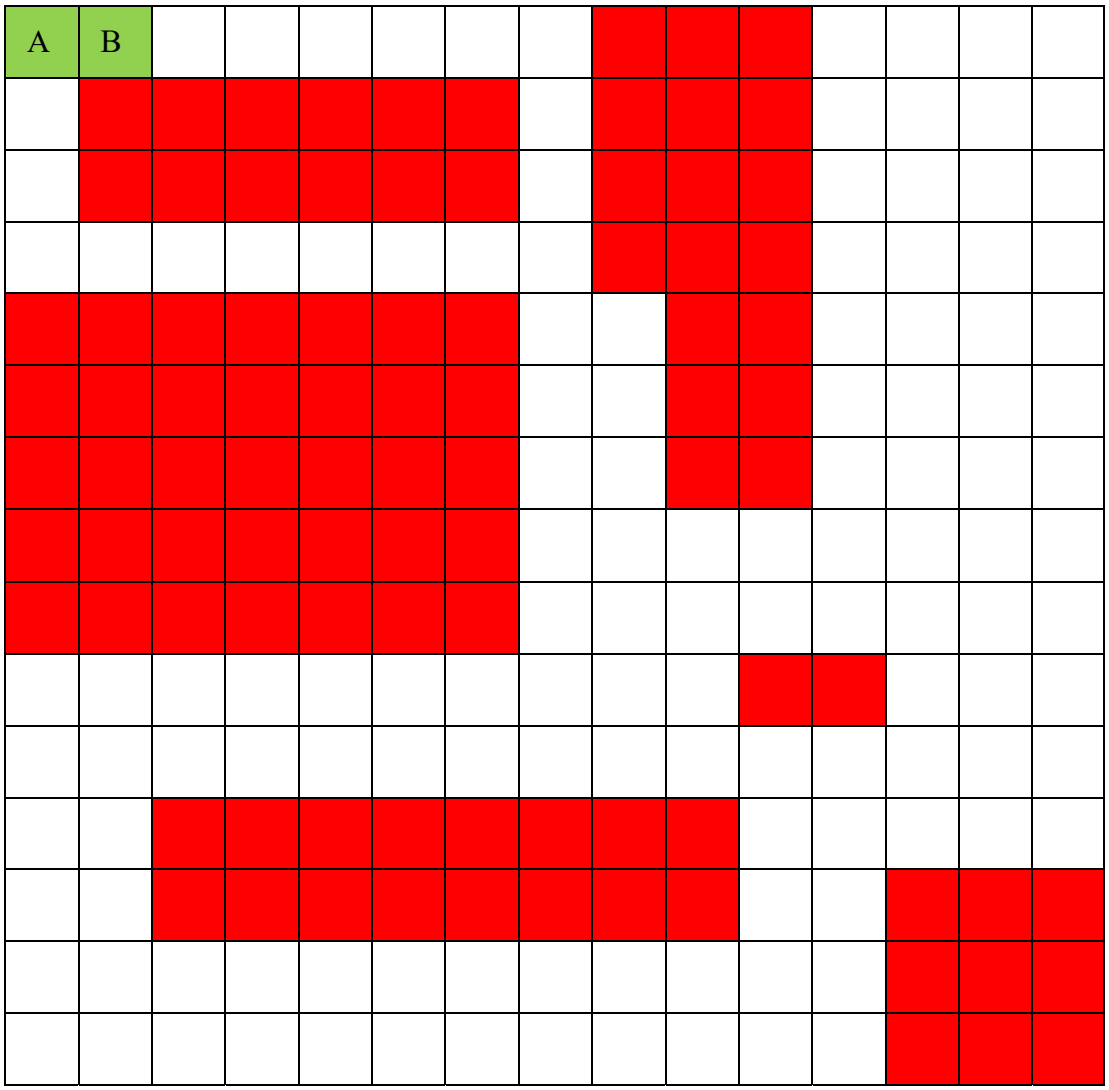


Figura 4.8. Representación del terreno, en rojo obstáculos y en verde casillas visitadas. y en rojo obstáculos

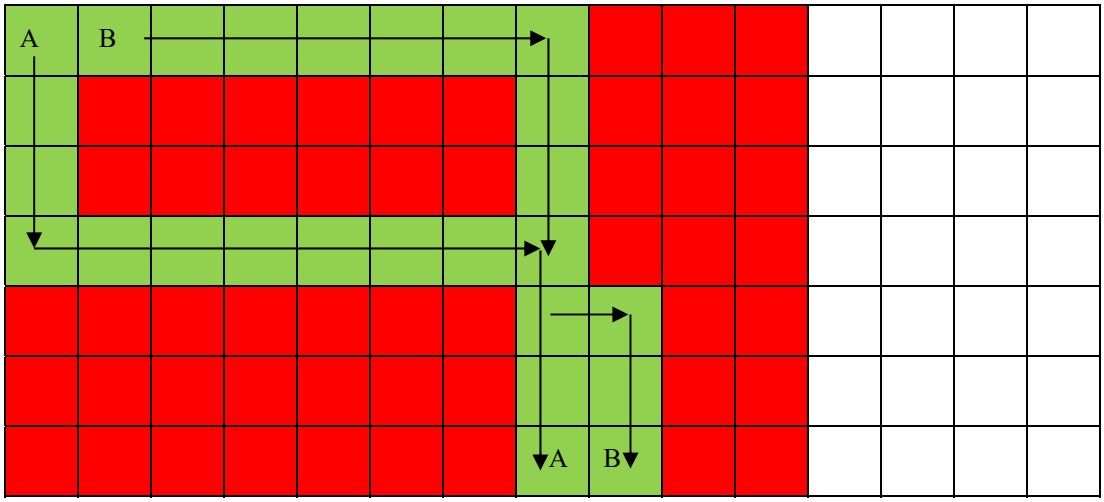


Figura 4.9. Representación de la primera iteración (parte superior del terreno).

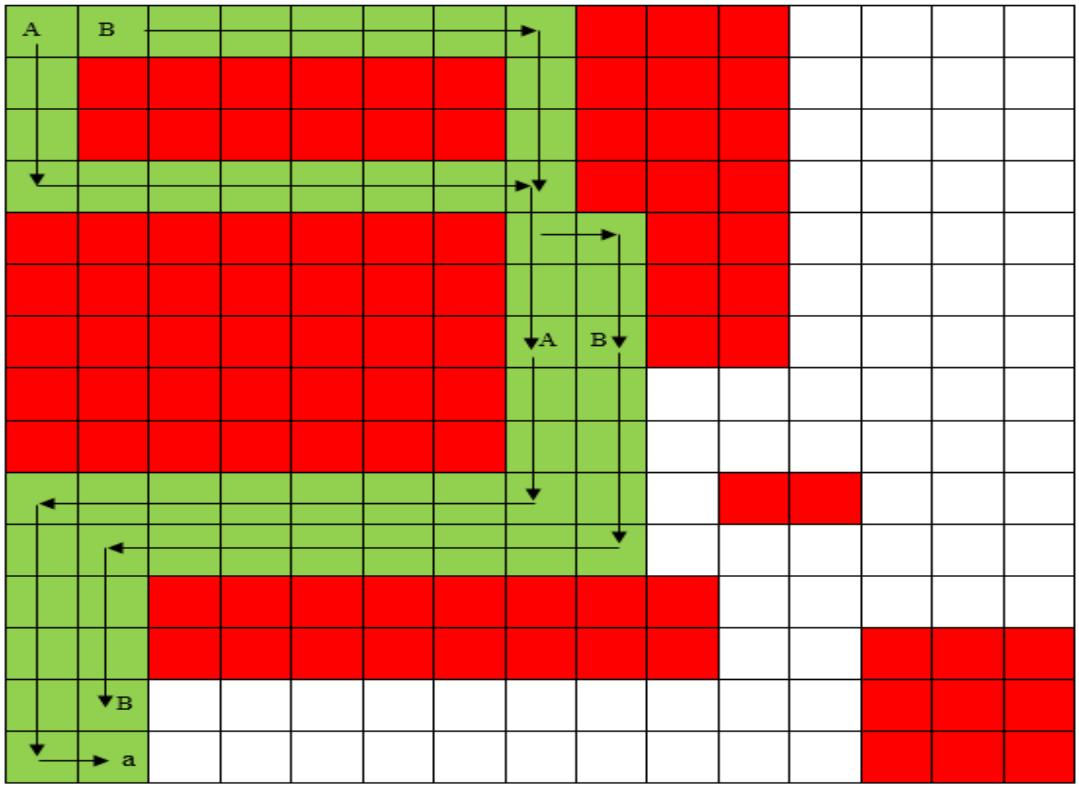


Figura 4.10. Representación del segundo intervalo de desconexión Ti

Podemos observar que gracias a la función que establece las puntuaciones de los puntos de encuentro, va dejando una gran zona sin visitar.

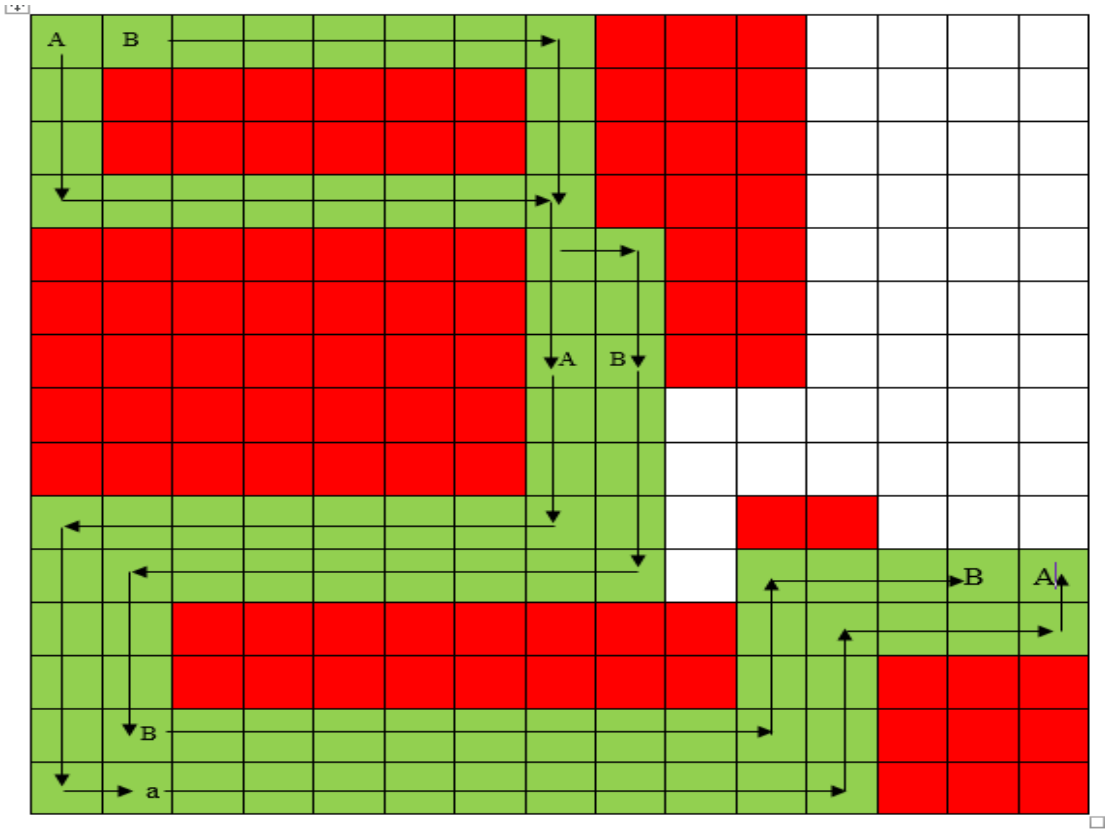


Figura 4.11. Representación del tercer intervalo de desconexión Ti

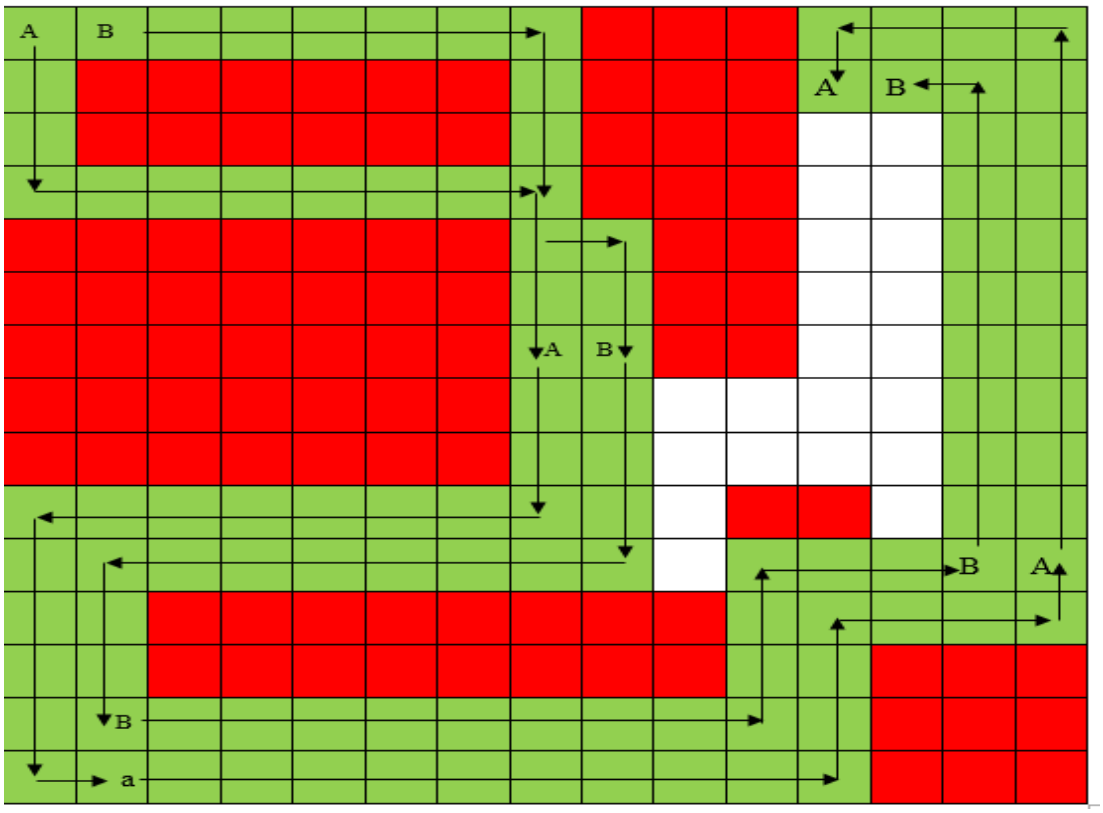


Figura 4.12. Representación del cuarto intervalo de desconexión Ti

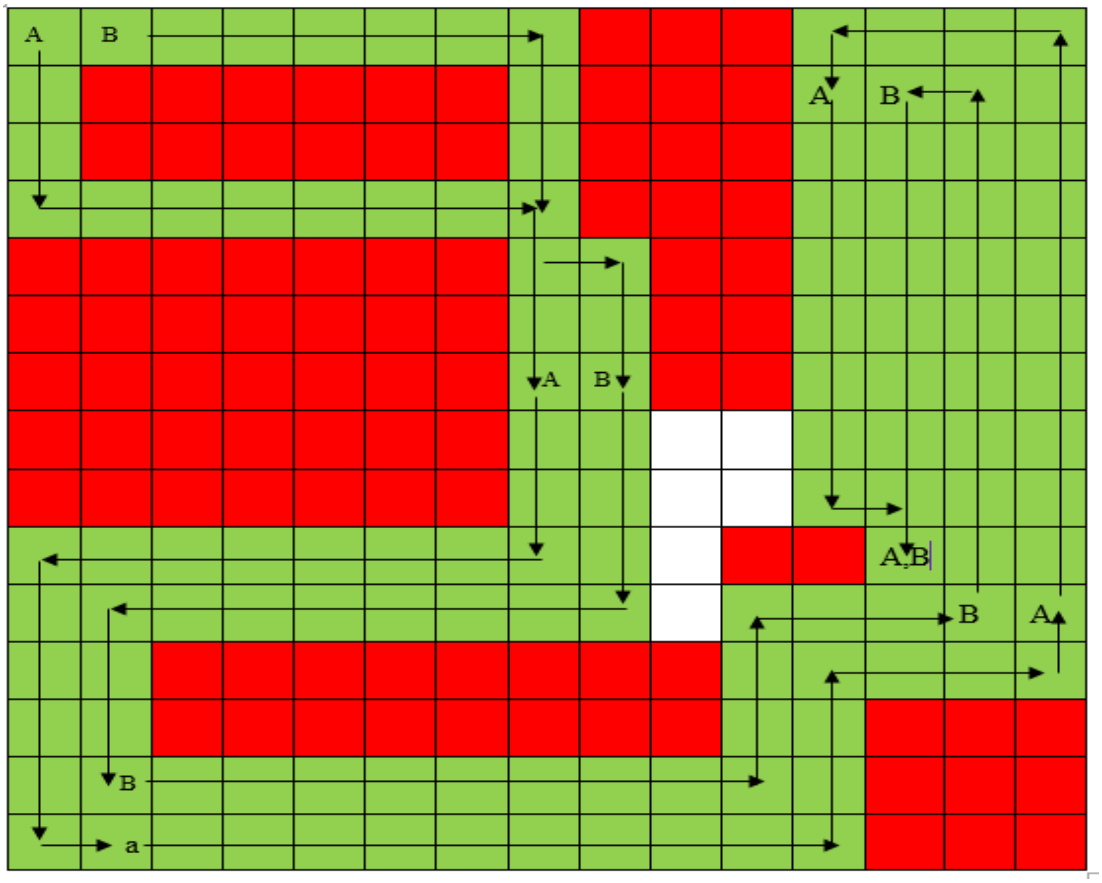


Figura 4.12. Representación del quinto intervalo de desconexión Ti

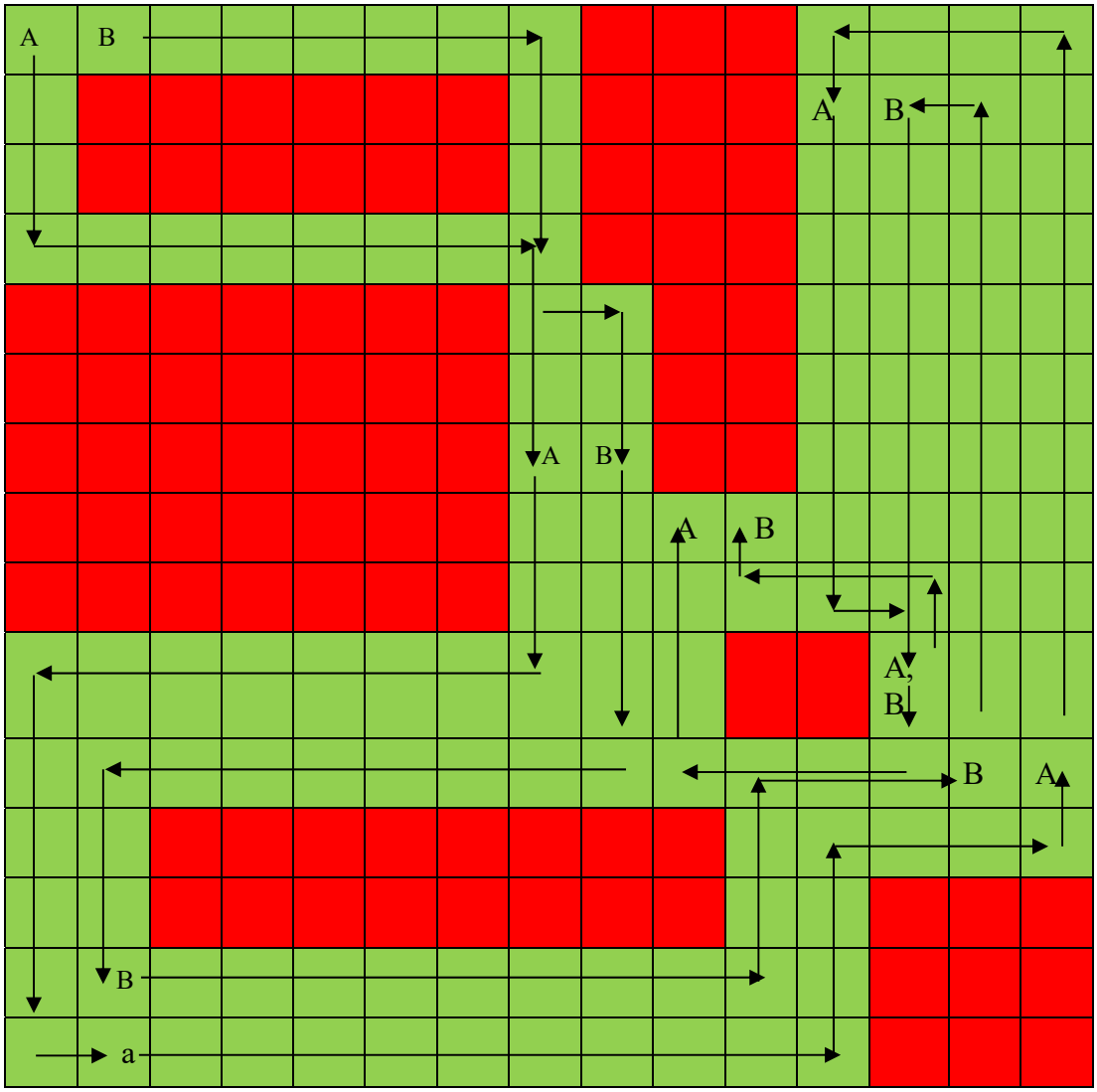


Figura 4.13. Representación del último intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

Tabla 4.3. Resultados de la simulación del escenario 3

Parámetro	***Valor***
Nº de robots	2
Ti	17
Nº islas dejadas en cada iteración	0
Nº pasillos dejados en cada iteración	0
Nº de celdas repetidas	4
Nº de iteraciones	6
Tiempo total de cubrimiento	102

Podemos observar que apenas repite áreas ya visitadas, únicamente repite 4 veces, y el algoritmo para este escenario converge en 6 iteraciones. Y viendo la complejidad y el tamaño del terreno, así como el límite de unidades de tiempo o pasos impuestos que eran 14, podemos concluir que también llega a la solución óptima o a una solución muy cercana a la óptima, y aunque en muchas ocasiones los vehículos van muy pegados, casi en conexión continua, a veces es conveniente optar por dicha solución antes que romper la conexión.

4.4. Escenario 4.

Por último, vamos a evaluar el siguiente escenario, el cual, con este algoritmo, convergerá en 1 sola iteración.

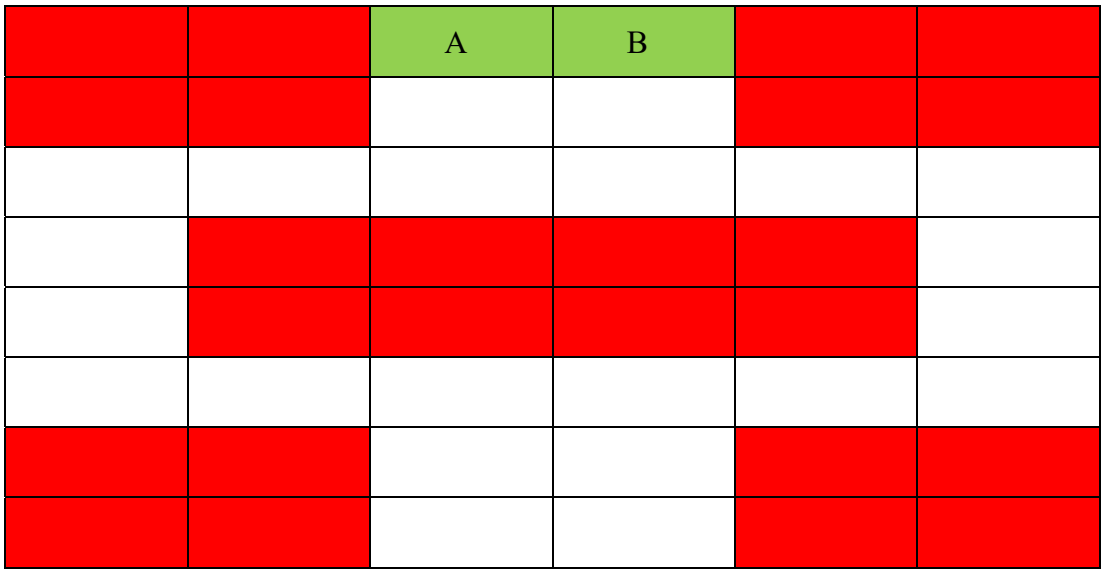


Figura 4.14. Representación del terreno, en rojo obstáculos y en verde casillas visitadas. En verde celdas visitadas y en rojo obstáculos

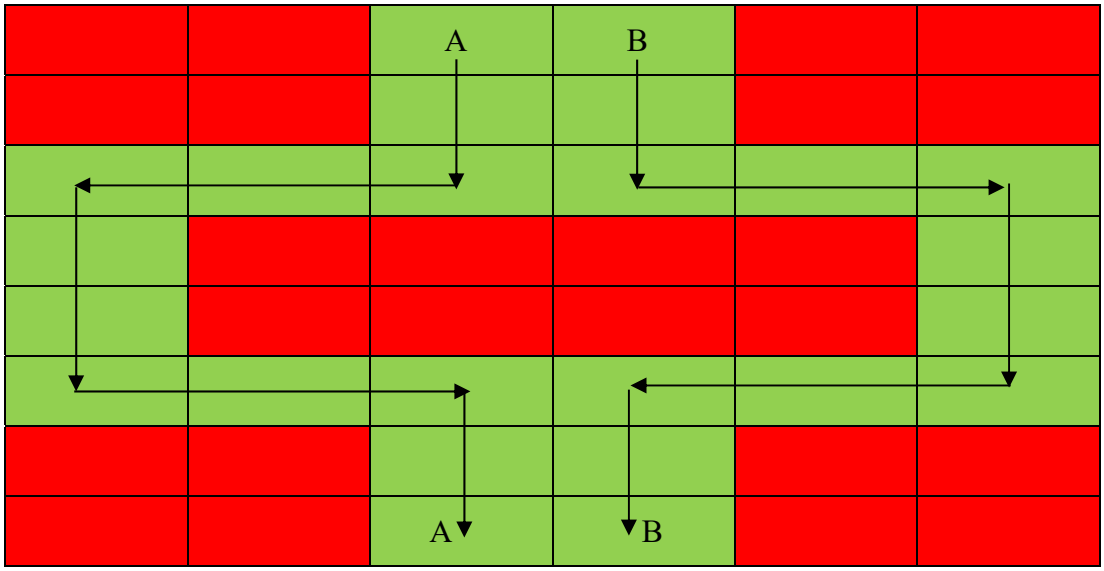


Figura 4.15. Representación del último intervalo de desconexión Ti En verde celdas visitadas y en rojo obstáculos

Tabla 4.4. Resultados de la simulación del escenario 4

Parámetro	***Valor***
Nº de robots	2
Ti	11
Nº islas dejadas en cada iteración	0
Nº pasillos dejados en cada iteración	0
Nº de celdas repetidas	0
Nº de iteraciones	1
Tiempo total de cubrimiento	11

5 -ESTUDIO COMPARATIVO.

En este capítulo, se va a realizar un estudio comparativo entre el algoritmo diseñado, y el MIPP-PC. Se analizarán sus principales diferencias y se pondrá como ejemplo un escenario con un osbtáculo grande y poco margen de maniobra para demostrar la mayor eficiencia del algoritmo diseñado.

Así, aunque ambos algoritmos tengan un objetivo común, y [2] sea la base del algoritmo diseñado; existen grandes diferencias. Las principales son las siguientes:

- En nuestro algoritmo, los dos vehículos parten hacia un punto de encuentro de forma simultánea, mientras que en [2] el primero parte buscando una ruta que tras Ti unidades de tiempo, conecte con el segundo vehículo que permanece estacionario. Así, el algoritmo diseñado ahorraría tiempo y nos permitiría avanzar en la exploración de un terreno sin entrar en bucle.
- En [2] se considera un rango de comunicación infinito, dos vehículos 'se ven' si hay una línea de visión directa entre ellos. En cambio, en el algoritmo diseñado, el rango de comunicación es limitado, acercándose más a la realidad. Los vehículos están conectados si se encuentran en celdas contiguas del grafo del terreno.
- A la hora de visitar todas las islas remanentes, el algoritmo que [2] expone, no soluciona la situación. Sin embargo, en nuestro planteamiento, se ha implementado una aproximación del TSP métrico, de manera que se recorran dichas islas de forma óptima y eficiente.

Así, lo veremos más claramente en el siguiente ejemplo:

Si establecemos un tiempo de desconexión de 6 unidades, la primera iteración resultante de aplicar el algoritmo de coordinación implícita la vemos en la figura 5.1.

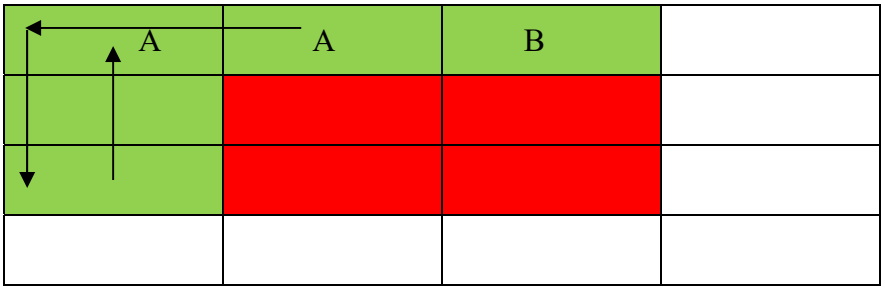


Figura 5.1. Representación del primer intervalo de desconexión Ti

Podemos observar que A comienza su intervalo de desconexión mientras B permanece estacionario en su posición inical. Así, en el tercer paso, A tiene que volver por la ruta ya recorrida para que, tras las 6 unidades de tiempo, tenga visibilidad con B.

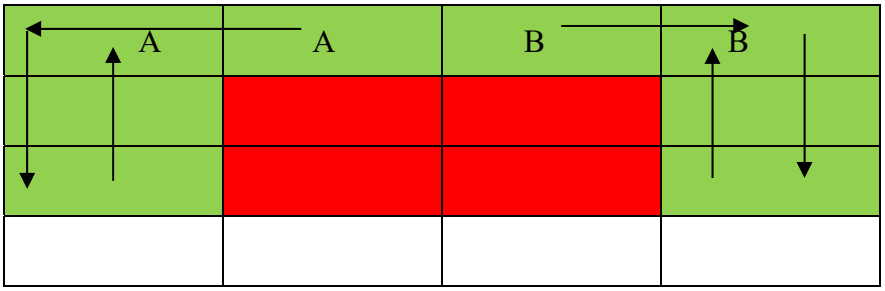


Figura 5.2. Representación del segundo intervalo de desconexión Ti

Figura 5.3. Representación del tercer intervalo de desconexión Ti

Podemos observar, que se ha necesitado 3 iteraciones para que el algoritmo converja, además se han repetido celdas ya visitadas en 6 ocasiones, lo que para el escenario que estábamos estudiando, significativamente pequeño, es realmente ineficiente.

En cambio, con nuestro algoritmo, que hemos explicado en el capítulo anterior se convergería a la solución óptima en sólo una iteración, sin repeticiones de la siguiente manera:

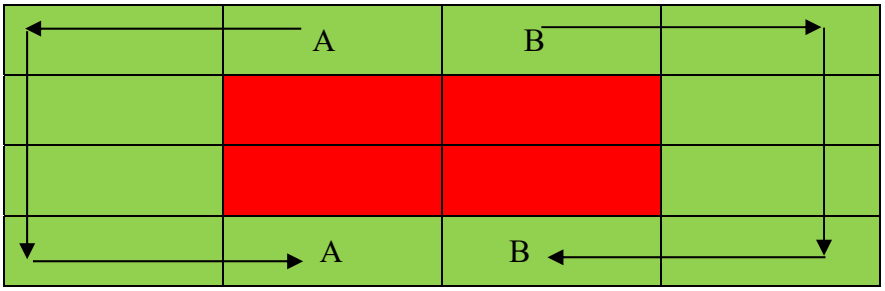


Figura 5.4. Representación del primer y último intervalo de desconexión Ti

Así, podríamos concluir que nuestro algoritmo en este tipo de terrenos con poco margen de maniobra y grandes obstáculos es mucho más eficiente que en el que han planteado en [2].

6 -CONCLUSIONES Y LÍNEAS FUTURAS.

En primer lugar, se ha partido del problema de exploración de terrenos mediante drones conectados periódicamente cada Ti unidades de tiempo. Se ha analizado un algoritmo que plantea una solución a este tipo de problemas [2], y se han estudiados sus puntos débiles y situaciones donde no es eficiente su aplicación. Finalmente, hemos planteado un algoritmo alternativo que intenta resolver dichos puntos débiles para mejorar la eficiencia en el cubrimiento de determinados terrenos.

Así, usando algoritmos como el de Dijkstra o el de Christofides, que además son eficientes computacionalmente, y partiendo de la idea de que los drones cubran el terreno de forma simultánea, se buscan puntos de encuentro que, considerando el camino de los vehículos hacia éstos, minimicen el número de islas y pasillos, y maximicen el número de celdas nuevas visitadas. Así, hemos llegado a un algoritmo que es eficiente en terrenos con muchos obstáculos y pocos márgenes de maniobras.

El algoritmo creado, se ha implementado en MATLAB, ya que este programa tiene una gran rapidez operativa, eficiencia computacional, y nos permite operar con las matrices y vectores de forma sencilla.

Finalmente, hemos comparado el MIPP-PC y el algoritmo que hemos propuesto, y se ha llegado a la conclusión de que, en ambientes y terrenos con poco margen de maniobra y grandes obstáculos, nuestro algoritmo es más eficiente porque en éste, los drones apenas repiten celdas ya visitadas y por tanto se minimiza el tiempo de cobertura del terreno.

Como aplicación, este algoritmo podría aplicarse a proyectos de repartición de paquetes mediante vehículos aéreos no tripulados. Después de terminar las diferentes tareas de repartición programadas (intervalo de desconexión), los drones se reunen para comprobar que todos los paquetes se han repartido de forma correcta, y que los vehículos no han sufrido ningún daño y se encuentran en óptimas condiciones para continuar sus tareas (conexiones periódicas). Así, con este método se agilizarían de manera considerable los repartos.

Otro uso que se podría dar a este algoritmo es en el control de la ganadería, en el que varios drones despegan para controlar la situación de los diferentes animales, con el objetivo de hacerlo de la manera más rápida posible y encontrarse una vez terminada dicha tarea, para comprobar que el ganado se encuentra en buen estado.

Como trabajo futuro, sería necesario extrapolar el caso de 2 vehículos no tripulados que hemos estudiado, al caso de 'n' vehículos. Así, un vehículo tendrá que tener en cuenta las rutas de los n-1 restantes y de esta forma, al utilizar más robots, el algoritmo cubriría todo el terreno de forma exhaustiva de una forma mucho más rápida.

En el siguiente enlace podemos ver el proyecto de AMAZON: http://www.rtve.es/alacarta/videos/la-tarde-en-24-horas/amazon-experimenta-drones-para-repartir-volando-paquetes/2188867/

Figura 6.1. Vehículo no tripulado diseñado por AMAZON para repartir paquetes.

APÉNDICE A: CÓDIGOS DE MATLAB.

main.m

```
escenario1
k=1;
format long
for i=1:size(G,1)
    for j=1:size(G,2)
        if G(i,j)~=2;
            M(i,j)=k;
            k=k+1;
        end
    end
end
V=ones(size(G,1),size(G,2));
for i=1:size(G,1)
    for j=1:size(G,2)
        if G(i,j)==2
            V(i,j)=0;
        end
    end
end
n_WP=size(G,1)*size(G,2);
peso=ones(n_WP,1);
n_robots=2;
rutag1=[];
rutag2=[];
zona_visitar=1;

dibuja_terreno(G)

for i=1:20
    L=G;
    pos=[nodo1(1) nodo1(2);nodo2(1) nodo2(2)];
    [etiq,label] = recurs(V);
    if label-1<=1
        zona_visitar=min(min(G));
    else
        [ L, Hl, cost, path ] = getLabeledMinPath(G, pos);
        t=evalua_vecinos(L,pos);
        if t==1
            zona_visitar=path(1);
        else
            zona_visitar=path(2);
        end
    end




    G(nodo1(1),nodo1(2))=1;G(nodo2(1),nodo2(2))=1;
    Vant=V;
```

```
    [D,D1,waypoints,H,V,nota,peso,zona_visitar,G]=wp(G,nodo1(1),…
    …nodo1(2),nodo2(1),nodo2(2),limite_pasos,H,M,zona_visitar,V,…
    …rutag1,rutag2,peso,Gd,L);
    D
    D1
    rutag1=[rutag1,D(end:-1:2)];
    rutag2=[rutag2,D1(end:-1:2)];
    [nodo1(1),nodo1(2)] = find(M==D(1));
    [nodo2(1),nodo2(2)] = find(M==D1(1));
    for m=length(D):-1:1
              [i,j]=find(M==D(m));
              V(i,j)=0;
              G(i,j)=1;
    end
    for n=length(D1):-1:1
              [u,v]=find(M==D1(n));
               V(u,v)=0;
               G(u,v)=1;
    end
    hold on,
    dibuja_camino(M,G,D,D1);shg
    pause
    if min(min(G))==1
        break
    end
end
```

dibuja_terreno.m

```
function dibuja_terreno(G)
for i=1:size(G,1)
    for j=1:size(G,2)
        x=[j-1 j j j-1];
        y=[-i-1 -i-1 -i -i];
        if G(i,j)==1
            patch(x,y,'w');
        else
            patch(x,y,'b');
        end
    end
end
```

dibuja_camino.m

```
function dibuja_camino(M,G,D,D1)
%el vector mayor es el a y el menor el b
a=[];
b=[];
x=[];y=[];x1=[];y1=[];
if length(D)>=length(D1)
    a=D;
    b=D1;
    h=length(D);
else
```

```
    a=D1;
    b=D;
    h=length(D1);
end

for k=h:-1:1
    [i,j]=find(M==a(k));
    x=[j-1 j j j-1];
    y=[-i-1 -i-1 -i -i];
        hold on, patch(x,y,'g')
    if k<=length(b)
        [i1,j1]=find(M==b(k));
        x1=[j1-1 j1 j1 j1-1];
        y1=[-i1-1 -i1-1 -i1 -i1];
        hold on, patch(x1,y1,'g');
    end
    pause(0.7)
end
```

recurs.m

```
function [etiq,label] = recurs(A)
[m,n] = size(A); %Número de filas y columnas

etiq = zeros(m,n);
label = 1;

for i=1:m
    for j =1:n
        if A(i,j) ~= 0
            [etiq,A] = meter(etiq,A,label,i,j);
            label = label+1;
        end
    end
end
```

meter.m

```
function [etiq,A] = meter(etiq,A,label,i,j)

A(i,j) = 0;
etiq(i,j) = label;

if i>1
    if A(i-1,j) ~= 0
        [etiq,A] = meter(etiq,A,label,i-1,j);
    end
```

```
    if j>1
        if A(i-1,j-1) ~= 0
            [etiq,A] = meter(etiq,A,label,i-1,j-1);
        end
    end
    if j<size(A,2)
        if A(i-1,j+1) ~= 0
            [etiq,A] = meter(etiq,A,label,i-1,j+1);
        end
    end
end
if j > 1
    if A(i,j-1) ~= 0
        [etiq,A] = meter(etiq,A,label,i,j-1);
    end
end
if j < size(A,2)
    if A(i,j+1) ~= 0
        [etiq,A] = meter(etiq,A,label,i,j+1);
    end
    if i<size(A,1)
        if A(i+1,j+1) ~= 0
            [etiq,A] = meter(etiq,A,label,i+1,j+1);
        end
    end
end
if i<size(A,1)
    if A(i+1,j) ~= 0
        [etiq,A] = meter(etiq,A,label,i+1,j);
    end
    if j>1
        if A(i+1,j-1) ~= 0
            [etiq,A] = meter(etiq,A,label,i+1,j-1);
        end
    end
end
```

getLabeledMinPath.m

```
function [ L, G, cost, path ] = getLabeledMinPath(M, pos)
% 2-approximation of TSP .
%
% [ L, G, cost, path ] = getLabeledMinPath(M, pos)
% M - map of the terrain 0-not visited, 1-visited cell and 2-
obstacle.
% pos - current positions of the robots.
% L - labeled of the terrain, each label identifies a not visited
island or
%       the edge(can be a single cell) of the robot
% G - adjacent matrix representing the distances between the islands
% cost - the cost of the estimated minimum path
% path - the estimated minimum path
    import java.util.LinkedList;
    q = LinkedList();
    M(pos(1,1), pos(1,2)) = 0;
    M(pos(2,1), pos(2,2)) = 0;
    L(1:size(M,1), 1:size(M,2)) = 0;
```

```
    mov = [1 0 -1 0; 0 1 0 -1];
    l = 0;
    for i = 1:size(M,1)
        for j = 1:size(M,2)
            if M(i,j) == 0 && L(i,j) == 0
                q.add([i j]);
                l = l + 1;
                while ~q.isEmpty()
                    cp = q.pop();
                    L(cp(1), cp(2)) = l;
                    for ii = 1:4
                        nr = cp(1)+mov(1,ii);
                        nc = cp(2)+mov(2,ii);
                        if nr>0 && nr<=size(M,1) && nc >0 &&
nc<=size(M,2)
                            if M(nr,nc) == 0 && L(nr, nc) == 0
                                q.add([nr nc]);
                            end
                        end
                    end
                end
            end
        end
    end
    G(1:l,1:l) = inf;
    m(l) = 0;
    for i = 1:size(M,1)
        for j = 1:size(M,2)
            if L(i,j) ~= 0 && m(L(i,j)) == 0
                nq = LinkedList();
                q.add([i j]);
                nq.add([i j]);
                D(1:size(M,1),1:size(M,2)) = inf;
                while ~q.isEmpty()
                    cp = q.pop();
                    D(cp(1), cp(2)) = 0;
                    for ii = 1:4
                        nr = cp(1)+mov(1,ii);
                        nc = cp(2)+mov(2,ii);
                        if nr>0 && nr<=size(M,1) && nc >0 &&
nc<=size(M,2)
                            if L(nr, nc) == L(cp(1), cp(2)) && D(nr,
nc) ~= 0
                                q.add([nr nc]);
                                nq.add([nr nc]);
                            end
                        end
                    end
                end
                while ~nq.isEmpty()
                    cp = nq.pop();
                    for ii = 1:4
                        nr = cp(1)+mov(1,ii);
                        nc = cp(2)+mov(2,ii);
                        if nr>0 && nr<=size(M,1) && nc >0 &&
nc<=size(M,2)
                            if M(nr, nc) ~=2 && D(nr, nc) > D(cp(1),
cp(2)) + 1
                                nq.add([nr nc]);
                                D(nr, nc) = D(cp(1), cp(2)) + 1;
```

```
                                if L(nr,nc) ~= 0 &&
G(L(i,j),L(nr,nc))>D(nr,nc)
                                    G(L(i,j), L(nr,nc)) = D(nr,nc);
                                end
                            end
                        end
                    end
                end
            end
        end
    end
    %disp(G);
    current = L(pos(1,1), pos(1,2));
    h(1:l-1) = 0;
    hback(1:l) = 0;
    c = 1;
    for i = 1:l-1
        if c == current
            c = c+1;
        end
        h(i) = c;
        hback(c) = i;
        c = c+1;
    end
    Gprime(1:l-1,1:l-1) = inf;
    w(1:l*(l-1)/2) = 0;
    x(1:l*(l-1)/2) = 0;
    y(1:l*(l-1)/2) = 0;
    c=1;
    for i = 1: l-1
        for j = i: l-1
            Gprime(i,j) = G(h(i), h(j));
            Gprime(j,i) = G(h(i), h(j));
            x(c) = i;
            y(c) = j;
            w(c) = G(h(i), h(j));
            c = c+1;
        end
    end
    DG = sparse(x,y,w);
    UG = tril(DG+DG');
    [~, pred] = graphminspantree(UG);
    disp(Gprime);
    disp(pred);
    linkedG(l-1) = LinkedList();
    for i = 1: l-1
        linkedG(i) = LinkedList();
    end
    root = 0;
    for i = 1:l-1
        if pred(i)==0
            root = i;
        else
            linkedG(pred(i)).add(i);
        end
    end
    travel(1:l-1) = 0;
    travel(1) = root;
    stack = LinkedList();
    stack.addFirst(root);
    c = 1;
```

```
    while ~stack.isEmpty
        iter = stack.element;
        if ~linkedG(iter).isEmpty
            child = linkedG(iter).pop();
            stack.addFirst(child);
            c = c + 1;
            travel(c) = child;
        else
            stack.pop();
        end
    end
    best = 0;
    best_dif = inf;
    dir = -1;
    for i = 1 : l-1
        for j = -1:2:1
            o = i + j;
            if o < 1
                o = length(travel);
            elseif o > length(travel)
                o = 1;
            end
            if G(current,h(travel(i)))-G(h(travel(i)),
h(travel(o)))<best_dif
                best_dif = G(current,h(travel(i)))-G(h(travel(i)),
h(travel(o)));
                best = i;
                dir = j;
            end
        end
    end

    cost = 0;
    path (1:l) = 0;
    path(1)=current;
    o = best;
    for i = 1:l-1
        path(i+1) = h(travel(o));
        cost = cost + G(path(i), path(i+1));
        o =  o + dir;
        if o < 1
            o = length(travel);
        elseif o > length(travel)
            o = 1;
        end
    end
end
```

evalua_vecinos.m

```
function t=evalua_vecinos(L,pos)
i=pos(1,1);
j=pos(1,2);
i1=pos(2,1);
j1=pos(2,2);
```

```
k=0;
k1=0;
for ii=-1:1
   for jj=-1:1
       if i+ii>0 && i+ii<=size(L,1) && j+jj>0 && j+jj<=size(L,2) &&
abs(ii)+abs(jj)<2 && L(i+ii,j+jj)~=0 && i+j+ii+jj~=i+j
         k=k+1;
       end
   end


for ii=-1:1
   for jj=-1:1
       if i1+ii>0 && i1+ii<=size(L,1) && j1+jj>0 && j1+jj<=size(L,2)
&& abs(ii)+abs(jj)<2 && L(i1+ii,j1+jj)~=0 && i1+j1+ii+jj~=i1+j1
         k1=k1+1;
       end
   end
end

if k>1 || k1>1
    t=1;
else
    t=0;
end
end
```

wp.m

```
function [dd,dd1,waypoints,H,V,nota,peso,zona_visitar,G]=wp(G,…
…nodox,nodoy,nodox1,nodoy1,limite_pasos,H,M,zona_visitar,V,rutag1,ru
…tag2,peso,Gd,L)
waypoints={};
nota=[0];
x=1;
dd=[];
dd1=[];
 [ddm,dd1m,waypointsm,Vm,notam,pesom,Gm]=wp_mismoputo(G,nodox,nodoy,
nodox1,nodoy1,limite_pasos,H,M,zona_visitar,V,rutag1,rutag2,peso,Gd,
L);
[ddv,dd1v,waypointsv,Vv,notav,pesov,Gv]=wp_vertical(G,nodox,nodoy,
nodox1,nodoy1,limite_pasos,H,M,zona_visitar,V,rutag1,rutag2,peso,Gd,
L);
for i=1:size(G,1)
    for j=1:size(G,2)-1
        peso1=peso;
        if L(i,j)==zona_visitar && L(i,j+1)==zona_visitar
            A=[nodox nodoy];
            A1=[nodox1,nodoy1];
            B=[i j];
            B1=[i,j+1];
```

```
            [D]=dijkstra1(A,B,Gd,peso1);
            for f=1:length(peso1)
              for t=1:length(D)
                   if f==D(t)
                   peso1(f)=peso1(f)+1;
                   end
              end
            end


            [D1]=dijkstra1(A1,B1,Gd,peso1);
            for b=1:length(peso1)
              for h=1:length(D1)
                   if b==D1(h)
                   peso1(b)=peso1(b)+1;
                   end
              end
            end


            pasos_a=length(D)-1;
            pasos_a1=length(D1)-1;
            g=[D,D1];
            f=[rutag1,rutag2];
            cont1=0;
            Vg=V;
            for w=1:length(g)
                [m,n]=find(M==g(w));
                cont1=cont1+Vg(m,n);
                Vg(m,n)=0;
            end
            casillas_nuevas=cont1;

            aux=max(nota);
            if pasos_a<=limite_pasos && pasos_a1<=limite_pasos
[nota(x),etiq]=nota_arista(pasos_a,pasos_a1,M,D,D1,V,rutag1,rutag2,
casillas_nuevas,G);
            else
                nota(x)=-10000;

            end
            x=x+1;
            if nota(x-1)>=aux
                waypoints={[i,j] [i,j+1]};
                dd=D;
                dd1=D1;
            end
        end
    end
end


for m=length(dd):-1:1
      [i,j]=find(M==dd(m));
       V(i,j)=0;
       G(i,j)=1;
end
for n=length(dd1):-1:1
      [u,v]=find(M==dd1(n));
       V(u,v)=0;
```

```
       G(u,v)=1;
end

for f=1:length(peso)
              for t=1:length(dd)
                   if f==dd(t)
                   peso(f)=peso(f)+1;
                   end
              end
end
for y=1:length(peso)
              for o=1:length(dd1)
                   if y==dd1(o)
                   peso(y)=peso(y)+1;
                   end
              end
end

if max(nota)==-10000
    dd=[];
    dd1=[];
end
notavec=[];
notap=0;
notap=max(nota);
notavec=[notav,notam,notap];
if max(notavec)==notav && max(nota)~=-10000 && notav~=-10000
    V=Vv;
    waypoints=waypointsv;
    dd=ddv;
    dd1=dd1v;
    peso=pesov;
    G=Gv;
end
if max(notavec)==notam && max(nota)~=-10000 && notav~=-10000 &&
notam~=-10000
    G=Gm;
    V=Vm;
    waypoints=waypointsm;
    dd=ddm;
    dd1=dd1m;
    peso=pesom;
end
end
```

Wp_vertical.m

```
Function [ddv,dd1v,waypointsv,Vv,notav,pesov,Gv]=wp_vertical(G,…
…nodox,nodoy,nodox1,nodoy1,limite_pasos,H,M,zona_visitar,V,rutag1,…
…rutag2,peso,Gd,L)
waypointsv={};
nota=[0];
pesov=peso;
x=1;
ddv=[];
dd1v=[];
Vv=V;
Gv=G;
for j=1:size(G,2)
    for i=1:(size(G,1)-1)
        peso1=peso;
        if L(i,j)==zona_visitar && L(i+1,j)==zona_visitar

            A=[nodox nodoy];
            A1=[nodox1,nodoy1];
            B=[i j];
            B1=[i+1,j];

            [D]=dijkstra1(A,B,Gd,peso1);
            for f=1:length(peso1)
              for t=1:length(D)
                   if f==D(t)
                   peso1(f)=3;
                   end
              end
            end
            [D1]=dijkstra1(A1,B1,Gd,peso1);
            for b=1:length(peso1)
              for h=1:length(D1)
                   if b==D1(h)
                   peso1(b)=3;
                   end
              end
            end
            pasos_a=length(D)-1;
            pasos_a1=length(D1)-1;
            g=[D,D1];
            f=[rutag1,rutag2];
             Vg=V;
             cont1=0;
            for w=1:length(g)
                [m,n]=find(M==g(w));
                cont1=cont1+Vg(m,n);
                Vg(m,n)=0;
            end
            casillas_nuevas=cont1;

            aux=max(nota);
            if pasos_a<=limite_pasos && pasos_a1<=limite_pasos


[nota(x),etiq]=nota_arista(pasos_a,pasos_a1,M,D,D1,V,rutag1,rutag2,
casillas_nuevas,G);
            else
```

```
                nota(x)=-10000;

            end
            x=x+1;
            if nota(x-1)>=aux
                waypointsv={[i,j] [i,j+1]};
                ddv=D;
                dd1v=D1;
            end
        end
    end
end
z=0;
z=isempty(ddv)+isempty(dd1v);
if z~=2
for m=length(ddv):-1:1
      [i,j]=find(M==ddv(m));
       Vv(i,j)=0;
       Gv(i,j)=1;
end
for n=length(dd1v):-1:1
      [u,v]=find(M==dd1v(n));
       Vv(u,v)=0;
       Gv(u,v)=1;
end


for f=1:length(pesov)
              for t=1:length(ddv)
                   if f==ddv(t)
                   pesov(f)=3;
                   end
              end
end
for y=1:length(pesov)
              for o=1:length(dd1v)
                   if y==dd1v(o)
                   pesov(y)=3;
                   end
              end
end
end

if max(nota)==-10000
    ddv=[];
    dd1v=[];
end

notav=max(nota);
end
```

wp_mismopunto.m

```
function
[ddm,dd1m,waypointsm,Vm,notam,pesom,Gm]=wp_mismoputo(G,nodox,…
nodoy,nodox1,nodoy1,limite_pasos,H,M,zona_visitar,V,rutag1,rutag2,pe
so,Gd,L)
waypointsm={};
nota=[0];
pesom=peso;
x=1;
ddm=[];
dd1m=[];
Vm=V;
Gm=G;
for j=1:size(G,2)
    for i=1:(size(G,1)-1)
        peso1=peso;
        if L(i,j)==zona_visitar

            A=[nodox nodoy];
            A1=[nodox1,nodoy1];
            B=[i j];
            B1=[i,j];



            [D]=dijkstra1(A,B,Gd,peso1);
            for f=1:length(peso1)
              for t=1:length(D)
                   if f==D(t)
                   peso1(f)=3;
                   end
              end
            end
            [D1]=dijkstra1(A1,B1,Gd,peso1);
            for b=1:length(peso1)
              for h=1:length(D1)
                   if b==D1(h)
                   peso1(b)=3;
                   end
              end
            end
            pasos_a=length(D)-1;
            pasos_a1=length(D1)-1;
            g=[D,D1];
            f=[rutag1,rutag2];
            Vg=V;
            cont1=0;
            for w=1:length(g)
                [m,n]=find(M==g(w));
                cont1=cont1+Vg(m,n);
                Vg(m,n)=0;
            end
            casillas_nuevas=cont1;

            aux=max(nota);
            if pasos_a<=limite_pasos && pasos_a1<=limite_pasos
[nota(x),zonas_convexas,etiq]=nota_arista(pasos_a,pasos_a1,M,D,D1,V,
rutag1,rutag2,casillas_nuevas,G);
```

```
            else
                nota(x)=-10000;

            end
            x=x+1;
            if nota(x-1)>=aux
                waypointsm={[i,j] [i,j]};
                ddm=D;
                dd1m=D1;
            end
        end
    end
end
z=0;
z=isempty(ddm)+isempty(dd1m);
if z~=2
for m=length(ddm):-1:1
      [i,j]=find(M==ddm(m));
       Vm(i,j)=0;
       Gm(i,j)=1;
end
for n=length(dd1m):-1:1
      [u,v]=find(M==dd1m(n));
       Vm(u,v)=0;
       Gm(u,v)=1;
end


for f=1:length(pesom)
              for t=1:length(ddm)
                   if f==ddm(t)
                   pesom(f)=3;
                   end
              end
end
for y=1:length(pesom)
              for o=1:length(dd1m)
                   if y==dd1m(o)
                   pesom(y)=3;
                   end
              end
end
end

if max(nota)==-10000
    ddm=[];
    dd1m=[];
end

notam=max(nota);
end
```

Dijkstra1.m

```
function [Ruta]=dijkstra1(A,B,G,peso)
Ruta=[];
k=1;
for i=1:size(G,1)
    for j=1:size(G,2)
        if G(i,j)~=0;
            N(i,j)=k;
            k=k+1;
        end
    end
end

n_WP      = size(G,1)*size(G,2);
visitado  = false(n_WP,1);     %vector visitado, iniciado todos a F
anterior  = zeros(n_WP,1);         %vector que nos dice el wp
anterior al que hemos llegado
distancia = Inf(n_WP,1); %vector que nos dice la distancia desde el
origen hasta ese wp
cola      = [];
Ini=N(A(1),A(2));
Fin=N(B(1),B(2));
distancia(Ini)=0;
%comenzamos
distancia(Ini)= 0;
cola          = Ini;
while~isempty(cola)
    %Comenzamos a trabajar con el nodo más prometedor (el de menor
    %distancia)
    [Y,I]       = min(distancia(cola));      %Y nos da el valor, I
nos da el índice
    u           = cola(I);                   %vector que esté en la
cola en la posición I
    visitado(u) = true;                      %marcamos el inicial
como visitado
    cola(I)     = [];
    if u == Fin
        break                                %si u es el último,
rompemos el bucle
    end
    %calculo de vecinos
    vecinos=busqueda_vecinos(N,G,u,visitado);     %encontramos los
WP a los que está conectado para añadirlo a la cola
    for i = vecinos
        aux = distancia(u) + peso(u); %distancia desde el vecino al
origen pasando por u
        if aux < distancia(i) && ~visitado(i)
            distancia(i) = aux;
            anterior(i) = u;
            if  any(cola == i)
            else                             %añade los elementos que
no estaban en la cola
                cola(end+1) = i;
            end
        end
    end
end
    pasos=distancia(u);
    n=1;
```

```
    while anterior(u)~=0;
        Ruta(n)=u;
        u=anterior(u);
        n=n+1;
    end
    Ruta(end+1)=Ini;
    H=Ruta;

End
```

búsqueda_vecinos.m

```
function vecinos=busqueda_vecinos(M,G,u,visitado)
vecinos=[];
[i,j] = find(M==u);
k=1;

for ii=-1:1
   for jj=-1:1
       if i+ii>0 && i+ii<=size(G,1) && j+jj>0 && j+jj<=size(G,2) &&
abs(ii)+abs(jj)<2 && M(i+ii,j+jj)~=0 && M(i+ii,j+jj)~=M(i,j) &&
visitado(M(i+ii,j+jj))~=1
           vecinos(k)=M(i+ii,j+jj);
           k=k+1;
       end
   end
end
end
```

REFERENCIAS

[1]. Roy, N., Burgard, W., Fox, D., & Thrun, S. (1999). Coastal navigation-mobile robot navigation with uncertainty in dynamic environments. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* (Vol. 1, pp. 35-40). IEEE.

[2]. Hollinger, G. A., & Singh, S. (2012). Multirobot coordination with periodic connectivity: Theory and experiments. *IEEE Transactions on Robotics*, *28*(4), 967-973.

[3]. Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, *61*(12), 1258-1276.

[4]. Hollinger, G., Singh, S., Djugash, J., & Kehagias, A. (2009). Efficient multi-robot search for a moving target. *The International Journal of Robotics Research*, *28*(2), 201-219. [5]. Peter Brass, Flavio Cabrera-Mora, "Multirobot Tree and Graph Exploration". Agosto 2011.

[6] Duncan, C. A., Kobourov, S. G., & Kumar, V. S. (2006). Optimal constrained graph exploration. *ACM Transactions on Algorithms (TALG)*, *2*(3), 380-402.

[7] Cesare, K., Skeele, R., Yoo, S. H., Zhang, Y., & Hollinger, G. (2015, May). Multi-UAV exploration with limited communication and battery. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2230-2235). IEEE.

[8] Moler, C. B. (2008). *Numerical Computing with MATLAB:* Revised Reprint. Siam.

[9]. Skiena, S. (1990). Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, 225-227.

[10]. Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, *20*(1), 255-282.

[11] Krause, A., Guestrin, C., Gupta, A., & Kleinberg, J. (2006, April). Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks* (pp. 2-10). ACM.

[12] Hollinger, G., Singh, S., Djugash, J., & Kehagias, A. (2009). Efficient multi-robot search for a moving target. *The International Journal of Robotics Research*, *28*(2), 201-219.

[13] Torrubia, G. S., & Terrazas, V. L. (2012). Algoritmo de Dijkstra. Un tutorial interactivo. *VII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2001).*

[14] Sujit, P. B., Sousa, J., & Pereira, F. L. (2009, May). UAV and AUVs coordination for ocean exploration. In *Oceans 2009-Europe* (pp. 1-7). IEEE.

[15] Michael, N., Zavlanos, M. M., Kumar, V., & Pappas, G. J. (2009). Maintaining connectivity in mobile robot networks. In *Experimental Robotics* (pp. 117-126).

[16] Hsieh, M. A., Cowley, A., Kumar, V., & Taylor, C. J. (2008). Maintaining network connectivity and performance in robot teams. *Journal of Field Robotics*, *25*(1-2), 111-131.

[17] Acevedo, J. J., Arrue, B. C., Maza, I., & Ollero, A. (2014, May). A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4735-4740). IEEE.

[18] Acevedo, J. J., Arrue, B. C., Díaz-Báñez, J. M., Ventura, I., Maza, I., & Ollero, A. (2013, May). Decentralized strategy to ensure information propagation in area monitoring missions with a team of UAVs under limited communications. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on* (pp. 565-574). IEEE.

[19] Acevedo, J. J., Arrue, B. C., Díaz-Báñez, J. M., Ventura, I., Maza, I., & Ollero, A. (2014). One-to-one coordination algorithm for decentralized area partition in surveillance missions with a team of aerial robots. *Journal of Intelligent & Robotic Systems*, *74*(1-2), 269-285.

[20] Bondy, J. A., & Murty, U. S. R. (1976). *Graph theory with applications* (Vol. 290). London.