DISEÑO DE UN ALGORITMO DE TRAZADO DE RUTAS PARA EL DESPLAZAMIENTO DE UN ROBOT EN EL PLANO

IGNACIO ARMANDO GUTIÉRREZ GÓMEZ

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA
BOGOTÁ DC
2017

DISEÑO DE UN ALGORITMO DE TRAZADO DE RUTAS PARA EL DESPLAZAMIENTO DE UN ROBOT EN EL PLANO

IGNACIO ARMANDO GUTIÉRREZ GÓMEZ

Trabajo de grado para optar por el título de Ingeniero Electrónico

Director
Brayan Jair Sáenz Cabezas
Ingeniero Electrónico

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA
BOGOTÁ DC
2017

Nota de aceptación

_____

_____

_____

_____

_____

Firma

Nombre:

Presidente del jurado

_____

Firma

Nombre:

Jurado

_____

Firma

Nombre:

Jurado

Bogotá DC, 8 de junio de 2017

A Esperanza G. T., Cristina G. T., Francisco G. G. y demás familiares que aportaron esfuerzos de alguna manera.

AGRADECIMIENTOS

Infinitas gracias a Óscar Penagos, Brayan Sáenz, Iván Darío Ladino y demás personas de la institución que soportaron y aportaron al desarrollo de este trabajo.

ÍNDICE

Pág.

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE TABLAS

Pág.

RESUMEN

En el presente documento se describe un método de búsqueda de ruta eficiente y evasiva de obstáculos, para el desplazamiento de un robot. Mediante la transformación de un entorno a su forma discreta; con lo cual se identificarán los objetos catalogados como obstáculos, y a partir de ello, se determinan espacios de tránsito por medio de descomposición del espacio vía *expansión en teselaciones con rombos*, hasta obtener puntos coordenados, que servirán como guía al robot para desplazarse. Dependiendo de un trazo de ruta entre un punto de partida y uno de llegada, se desplegara la búsqueda de ruta mas conveniente en distancia.

La respuesta inicial del algoritmo es presentar una ruta sub-optima(la distancias mas cortas entre punto de partida y de llegada) con el valor seleccionado de heurística inicialmente. En casos que el trayecto o recorrido sea muy intrincado, el algoritmo aumentara el valor de heurística y de promedio de distancia de trayectos, para lograr trazar la ruta sin cruzar por obstáculos.

El algoritmo se desempeña correctamente en entornos con un obstáculo de gran tamaño, así como de todos los trayectos dentro del espacio de configuración(áreas fuera de obstáculos).

INTRODUCCIÓN

Los algoritmos enfocados al desplazamiento en un plano, son desarrollados con el objetivo de obtener la mejor ruta y la mas confiable. Esto lo vemos por ejemplo en el desplazamiento de vehículos, robots encargados en la búsqueda, o incluso y sin ir muy lejos, en las nuevas implementaciones de vehículos con autonomía de conducción.

Con este algoritmo de búsqueda, desarrollado a partir del procesamiento de un entorno discreto, se logra satisfacer los criterios de desplazamiento eficaz, eficiente y sin colisiones para un robot, en un entorno en dos dimensiones. Ubicando aquellos objetos catalogados como obstáculos, y a partir de ellos, los espacios disponibles y fiables para el tránsito. Una vez localizados estos espacios, se procede a desplegar un punto de partida y uno de llegada para verificar el funcionamiento del algoritmo que determina el plan de ruta conveniente.

JUSTIFICACIÓN

La idoneidad de los trayectos permitirán asegurar la llegada al punto objetivo, sin importar si se encuentran uno o varios obstáculos, claro esta, siempre que halla un espacio suficiente para el desplazamiento del robot. Las trayectorias de desplazamiento se desplegaran en linea recta en la mayoría del recorrido, evitando ondular la ruta en todo el recorrido. Estas características pueden ser de utilidad en sistemas de desplazamiento de vehículos autónomos, en el sector agro-industrial, por ejemplo en la recolección de productos del cultivo, en la repartición de abono y suplementos a las siembras de gran extensión Ya que son entornos temporalmente estáticos y en los cuales se puede delimitar los espacios para desplazamiento, determinando el color resaltante de la siembra como obstáculos. Otra ventaja de aplicarlo en este escenario es que no se corren riegos de colisión con otros vehículos en movimiento con personas en una ciudad.

OBJETIVOS

OBJETIVO GENERAL

Desarrollar un algoritmo que determine un plan de ruta eficaz, de menor distancia y sin riesgo de colisión para el desplazamiento de un robot.

OBJETIVOS ESPECÍFICOS

- ▷ Representar de manera adecuada el espacio de configuración(área fuera de obstáculos).
- ▷ Limitar el espacio de búsqueda con propósitos de eficiencia computacional.
- ▷ Trazar la ruta mas corta dentro del espacio de búsqueda.

1. MARCO REFERENCIAL

1.1. ESTADO ACTUAL

El estado del arte del algoritmo de búsqueda que se quiere proponer, tiene como origen el planteamiento del profesor Subhrajit Bhattacharya en su tesis de grado como Doctor en (PhD) en Ingeniería Mecánica, con el titulo de: *"TOPOLOGICAL AND GEOMETRIC TECHNIQUES IN GRAPH SEARCH-BASED ROBOT PLANNING"*[2], presentado en la facultad de ingeniería mecánica de La Universidad de Pennsylvania en el año 2012. La tesis doctoral esta escrita en idioma inglés y se tomaran algunos textos concernientes al desarrollo de este proyecto de grado.

En virtud de la importancia del documento para este proyecto, y por respeto a la integridad de los derechos intelectuales del autor, se hará el mayor esfuerzo por hacer la interpretación adecuada de las ideas y conceptos de la tesis al idioma castellano, al igual que con otros textos en idioma ingles referenciados al final.

1.2. TESELACIONES, POLÍGONOS Y DIAGRAMAS

El componente teórico de teselaciones adquiere importancia como método para delimitar las rutas, ya que se expanden en los sitios del entorno donde se puede tener una posibilidad de trazado de ruta. Según su expansión, se obtendrá un punto equidistante aproximado entre los objetos considerados obstáculos, y de los obstáculos con los limites del entorno, cuando se determine su centro de masa. La técnica de las teselaciones se relaciona con los diagrama de Voronoi y su funcionalidad para el desarrollo del algoritmo. A continuación la teoría expuesta por [3] que explica su construcción

1.2.1. Construyendo el diagrama el Voronoi

En cuanto a la idea de partición de una superficie, existen varios criterios de percepción. En primer lugar se encuentran los poligonos Thiessen, diagramas de Voronoi entre otros, pero en realidad, quien los utilizó en un tratado sobre formas cuadráticas fue Voronoi(1908). También se les llama Regiones de Dirichlet o células de Wigner-Seitz. Dan Hoey ha sugerido que la descripción más acertada e imparcial es el término "polígonos proximales"[3].

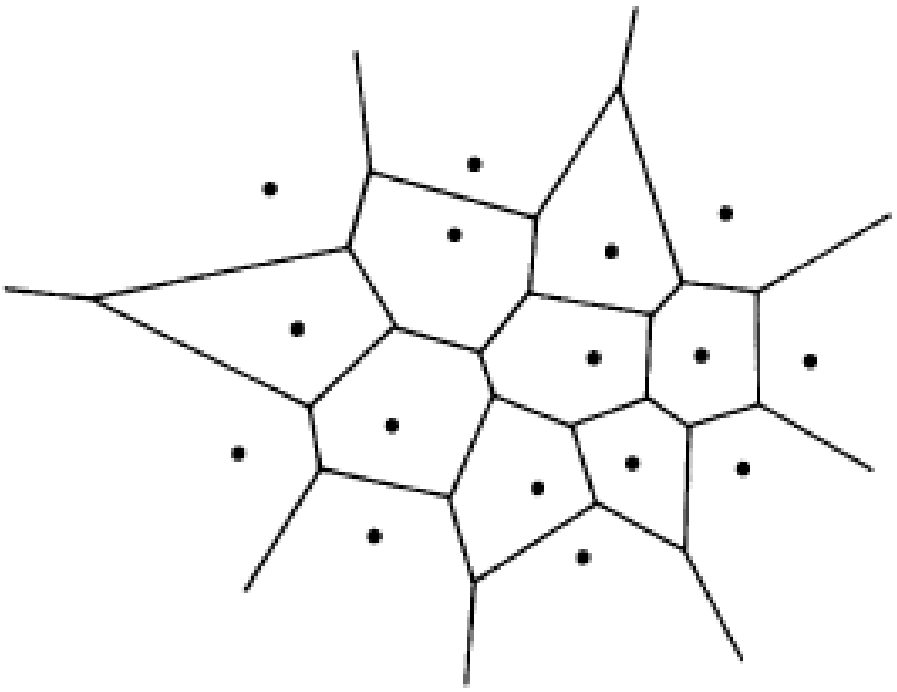


Figura 1.1: Diagrama de Voronoi[3]

La construcción de los diagramas de Voronoi es un fin en sí mismo en una serie de campos. En arqueología, los polígonos de Voronoi se utilizan para mapear la difusión del uso de herramienta, en ecología, cultura antigua (influencia de comercio ribal), estudios de recursos de consumo en una vecindad, entre otros.[3]

Para construir el diagrama de Voronoi $Vor(S)$ de un conjunto de puntos S (de aquí en adelante referido formalmente como problema DIAGRAMA VORONOI), nos referiremos a producir una descripción del diagrama como un gráfico plano incrustado en el plano, consistente en los siguientes elementos:[3]

1. Las coordenadas de los vértices de Voronoi.
2. El conjunto de bordes (cada uno formado por un par de vértices de Voronoi), y los dos bordes que son sus sucesores en el sentido contrario a las agujas del reloj, en cada punto extremo (doblemente conectados-borde-lista). Esto implica que provee los ciclo de borde o expansión, en cada vértice,en el sentido anti horario, y el ciclo de borde o expansión, en sentido horario alrededor cada cara.[3]

Los párrafos anteriores describen la construcción convencional del *Diagrama de Voronoi*, de la misma forma que se hace, en el algoritmo Tesela_ romb_ IG_ definitivo_ 3 desarrollado en [**1**], el cual servirá para el algoritmo de búsqueda de ruta planteado. en la Figura 1.1 podemos ver varios elemento que constituyen el Diagrama de Voronoi:

- Los vértices de Voronoi: puntos negros.
- Los polígonos de Voronoi: formas geométricas convexas.
- Los bordes de los polígonos de Voronoi:las lineas negras.

En el algoritmo de búsqueda planteado, se consideran los vértices de Voronoi como los puntos distribuidos en el espacio de configuración, en el entorno discreto. Están distribuidos de manera equidistante entre obstáculos y entre obstáculos y limites del contorno.
Los polígonos de Voronoi se pueden comparar a las formas generadas por las expansiones de las teselaciones*, cuyos orígenes se encuentran en los vértices planteados anteriormente. La finalidad de estas expansiones es retirar el vértice lo mas posible de los limites de los obstáculos y de los limites del contorno, una vez sea calculado el centro de masa de estas teselaciones o polígonos de Voronoi. Estos centros de masa se conoceran como centroides o como puntos de referencia definitivos para el trazado de ruta en cada caso. con ello se evitan las colisiones con los obstaculo, ya que le da mas tolerancia a los espacios cuando la ruta necesita rodear el obstaculo. La Figura 1.2 muestra un ejemplo de expasión de telesciones.
Los bordes de los polígono de Voronoi son comparables a los limites creados por las teselaciones, con los cuales es posible calcular los centros de masa de las teselaciones expandidas.

*teselaciones y teselado hacen referencia a una regularidad o patrón de figuras que recubren o pavimentan completamente una superficie plana que cumple con dos requisitos: Que no queden espacios. Que no se superpongan las figuras.

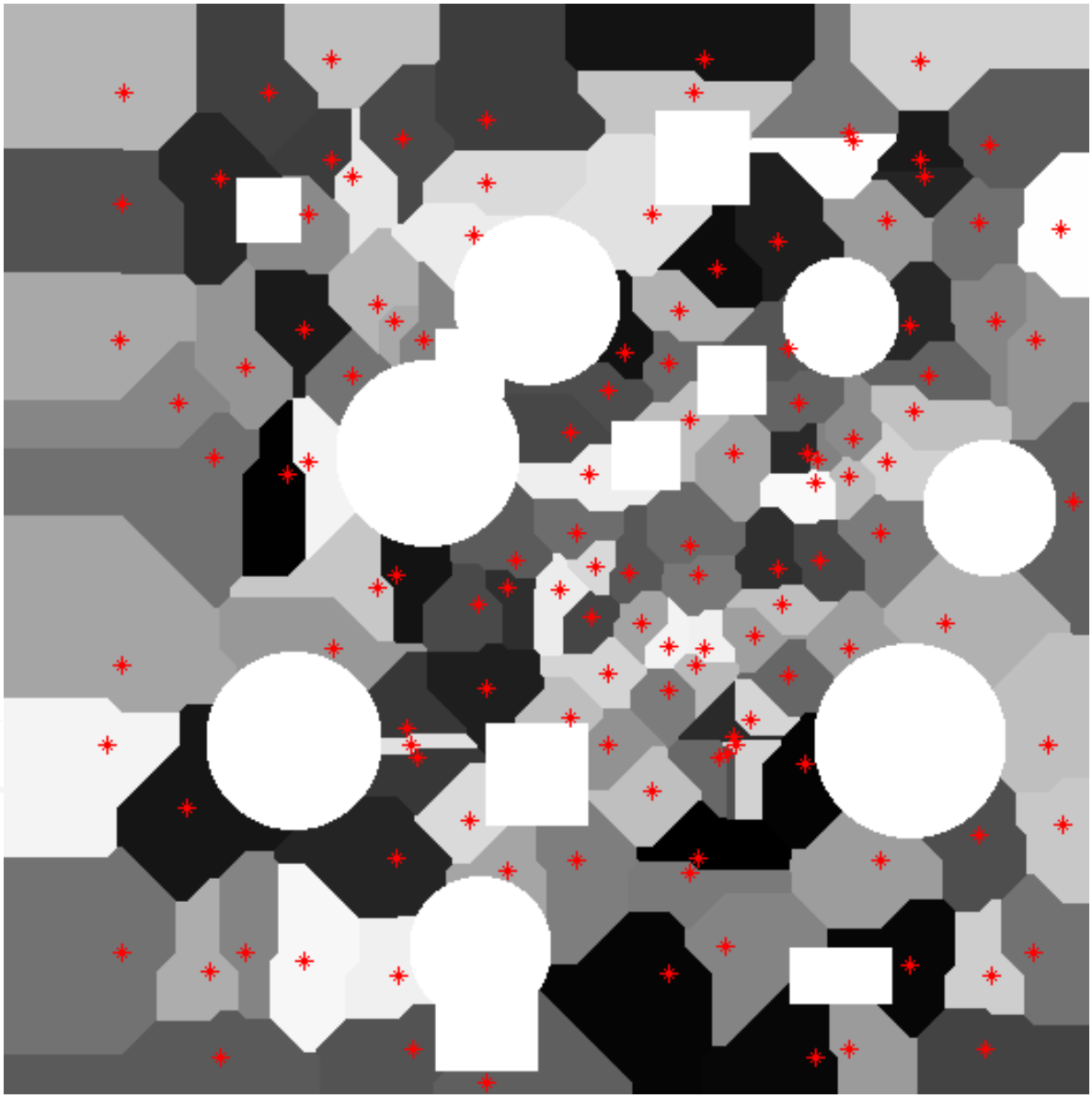


Figura 1.2: ejemplo de un entorno teselado con los vertices de origen en color rojo y sus limites.fuente autor

1.3. ALGORITMO DE DIJKSTRA

El algoritmo de Dijkstra [4] es uno de los algoritmos fundamentales que debe ser referenciado si se quiere desarrollar un algoritmo de búsqueda de una ruta optima o de menor distancia de recorrido. Aunque su principal aplicación se da en teoría de grafos o en teoría de construcción de arboles por distancia. Lo que podría dar a entender que funciona en entornos direccionales o limitados y enfocados en una dirección. El algoritmo funciona utilizando un conjunto de nodos (vértices) dispersos en el entorno, y a partir de un nodo de inicio p_i y un nodo de llegada o fin, busca una ruta por medio de los nodos que lo desplace hasta el objetivo. Esto lo hace determinando la distancia mas corta de desplazamiento entre los puntos vecino o cercanos al punto de inicio p_i. Una vez hallado el nodo vecino, se moviliza hasta ese nodo y elimina el nodo anterior; y nuevamente calcula las distancias con los nodos vecinos al nuevo punto de inicio, hasta lograr llegar al punto objetivo. A continuación una parte del texto de su escrito y el pseudocódigo tomado de [2][**] que podría representar la implementación en un algoritmo:

Consideramos puntos (nodos), algunos o todos los pares de los cuales están conectados por una rama; Se da la longitud de cada rama. Nos limitamos al caso en el que al menos un camino existe entre dos nodos cualesquiera. Ahora consideramos dos problemas. Problema 1. Construir el árbol de longitud total mínima entre los n nodos. (Un árbol es un gráfico con un solo camino entre cada dos nodos.) En el curso de la construcción que presentamos aquí, las ramas se subdividen en tres conjuntos:[4]

1. las ramas definitivamente asignadas al árbol en construcción (que Forman un sub-árbol).
2. las ramas de las que se añadirá la siguiente rama al conjunto 1, serán seleccionadas.
3. Las ramas restantes (rechazadas o no consideradas).

Los nodos se subdividen en dos conjuntos:

A. los nodos conectados por las ramas del conjunto 1.
B. los nodos restantes (una y sólo una rama del conjunto 2 conducirá a cada uno de estos nodos).

[**] En ciencias de la computación, y análisis numérico, el pseudocódigo (o falso lenguaje) es una descripción de alto nivel compacta e informal del principio operativo de un programa informático u otro algoritmo.

Comenzamos la construcción eligiendo un nodo arbitrario como el único miembro del conjunto A, y colocando todas las ramas que terminan en este nodo en el conjunto 2. Para empezar, el conjunto 1 está vacío. A partir de entonces realizaremos los dos pasos siguientes repetidamente.

- [Paso 1.]La rama más corta del conjunto 2 se elimina de este conjunto y se añade al conjunto 1. Como resultado, se transfiere un nodo del conjunto B al conjunto 2.
- [Paso 2.]Considere las ramas que conducen desde el nodo que acaba de ser transferido al conjunto A, a los nodos que todavía están en el conjunto B. Si la rama considerada es más larga que la rama correspondiente en el conjunto 2, se rechaza; si este es más corto, sustituye el conjunto correspondiente 2, y éste es rechazado.

Luego regresamos al Paso 1 y repetimos el proceso hasta que los conjuntos 2 y B estén vacíos. Las ramas del conjunto 1 forman el árbol requerido.[4]

Algorithm 2.3.1

g = **Dijkstras** (G, p)
Inputs: a. Graph G
b. Start node $p \in \mathcal{V}(G)$
Outputs: a. The shortest distance map $g : \mathcal{V}(G) \to \mathbb{R}^+$

```
1  Initiate g: Set g(v) := ∞, for all v ∈ V(G)    // Minimum distance
2  Set g(p) = 0
3  Set Q := V(G)    // Set of un-expanded nodes
4  while (Q ≠ ∅)
5      q := argmin_{q'∈Q} g(q')    // Vertex to expand. Q is maintained by a heap data-structure.
6      if (g(q) == ∞)
7          break
8      Q = Q − q    // Remove q from Q
9      for each ({w ∈ N_G(q)})    // For each neighbor of q
10         Set g' := g(q) + C_G([q, w])
11         if (g' < g(w))
12             Set g(w) = g'
13 return g
```

where, $\mathcal{N}_G(u) = \{w' \in \mathcal{V}(G) \mid [u, w'] \in \mathcal{E}(G)\}$, the set of neighbors of u.

Figura 2.1: pseudocódigo del algoritmo de Dijkstra. $V(G) = Q = conjuntodenodos$, $\epsilon(G) = alosbordesdelosfrentesdeonda$ y $C_G = funcioncostoomenordistancia$ tomado de [2]

Ahora veamos gráficamente como funciona *el algoritmo de Dijkstra*:

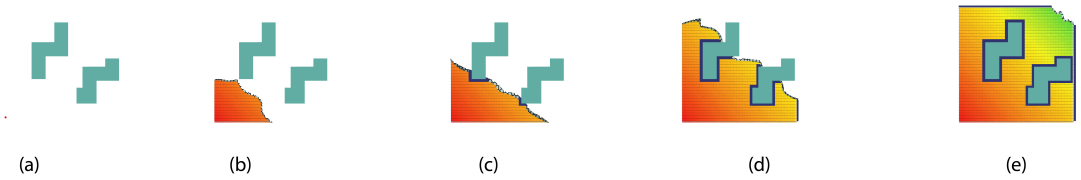


Figura 2.2: Ilustración del proceso del algoritmo de Dijkstra. Nodo de inicio en rojo.tomado de [2]

[(a)] $iter = 0$, [(b)] $iter = 0.25$, [(c)] $iter = 0.5$, [(d)] $iter = 0.75$, [(e)] $iter = 1$,

Haciendo una analogía entre el algoritmo de búsqueda de ruta planteado y el de Dijkstra, se puede encontrar el parentesco con la ubicación de aquellos nodos o vértices o puntos cercanos al punto de inicio o de partida de la ruta, ya que es muy importante determinar aquel punto que acorta distancia para llegar al objetivo, y de esta manera mantener un avance o desplazamiento en pro de la eficacia del algoritmo.

1.4. ALGORITMO A^*

Al igual que el algoritmo de Dijkstra[4], **el algoritmo** A^*[5] es fundamental en los antecedentes de algoritmos de búsqueda de rutas, ya que es una version con aplicación al desplazamiento de objetos (vehículos, robot, comunicaciones entre otros), adicional a que es aplicado en tareas de navegación de robot por su metodología de búsqueda orientada a enfocar esfuerzo a partir de la ubicación del objetivo o punto meta. Quizás una de sus características, usada en muchos algoritmos, es *la función heurística* que tiene el fin de calcular en todo momento la distancia mínima o costo de cada trayecto. Esta heurística depende de dos parámetros que son un par de puntos,vértices o nodos en los cuales la función determinara el costo total del recorrido. Estos nodos son el punto de partida y de llegada. A continuación el texto que sustenta esta explicación. Posterior a ello, la imagen del pseudocódigo del algoritmo y una imagen que ilustra el funcionamiento.

El algoritmo de Dijkstra[4] sigue siendo hasta el día de hoy uno de los mejores enfoques para solucionar de manera óptima el problema de trayecto más corto simple donde todos los arcos tienen longitudes no negativas.

En otra contribución temprana, Hart et al. [40](1968) desarrollaron una estrategia de búsque-

da, llamada A^*, para resolver caminos de costo mínimo. El enfoque A^* difiere de otros métodos ya que incorpora una estimación del costo de la 'trayectoria completa'. Para ciertas clases de funciones de estimación, A^* encontrará el camino óptimo.

En los últimos 50 años ha habido un interés continuo y sostenido en el desarrollo de algoritmos más rápidos para resolver el problema de trayecto más corto, ya que este problema se aplica en una amplia variedad de áreas incluyendo el enrutamiento de llamadas telefónicas y redes de comunicación y enrutamiento de vehículos en las redes de carreteras. El objetivo principal de este trabajo es proporcionar una comparación directa de A^* con lo que se considera ser la mejor implementación de algoritmos de trayecto más cortos para identificar el enfoque más eficiente cuando se utilizan datos procedentes del *Sistema de Información Geográfica (GIS)* . Durante la última década, se han realizado varios intentos notables de comparar algoritmos de trayecto más cortos (Cherkassky et al.,[34] 1996, Zhan y Noon [35] 1998). En particular, la falta de trabajo anterior es una comparación directa de A^* y eficaces implementaciones de Dijkstra[4]. Esto es algo comprensible, dado que A^* requiere una función para estimar el costo de terminación de un camino. Debido a esto, A^* no es adecuado para muchas aplicaciones de trayecto más cortas; Sin embargo, es potencialmente ideal para una aplicación que utiliza información procedente de GIS. Una de las primeras pruebas de A^* se debe a Golden y Ball [36] (1978) en redes euclidianas (es decir, una red definida en el espacio euclidiano r-dimensional con el coste de desplazamiento de cada arco proporcional a la distancia en línea recta entre los nodos que conecta) Donde se encontró que A^* no expandió más de aproximadamente el 10 % de los nodos de lo que sería ampliado por el algoritmo de Dijkstra[4]. Esta posibilidad también fue reconocida por Shekhar et al. [37](1993, véase también Shekhar y Fetterer [38]1996), donde realizaron pruebas preliminares de A^* aplicadas a la red de calles de Minneapolis y la compararon con el algoritmo de Dijkstra[4], así como con un método de búsqueda de "anchura inicial".

Su conclusión general no estaba clara cuando sugirieron que cada algoritmo parece tener alguna ventaja sobre los demás, dependiendo del problema que se resuelve y la longitud de la trayectoria resultante. Ertl (1998) rechazó esencialmente A^* como una alternativa viable cuando afirmó que A^*, en el mejor de los casos, es modestamente más rápido que el algoritmo de Dijkstra[4]. Por lo tanto, prácticamente todas las investigaciones previas han ignorado la existencia de A^*, han probado A^* en un problema especializado o han llegado a la conclusión de que tal vez sea tan bueno o modestamente mejor que el algoritmo de Dijkstra[4]. En una revisión reciente, Fu et al. (2006) afirmó que A^* es competitivo con los enfoques basados en Dijkstra[4], pero luego enfatizó las posibles ventajas de usar A^* como una heurística en lugar de como un algoritmo.[5]

Es importante introducir primero alguna notación general sobre una red. Siguiendo a Ga-

llo y Pallottino[33] (1986), utilizamos el símbolo $G(N, A; l)$ para representar una red dirigida con $n = |N|$ Nodos,$m = |A|$ arcos y una función de longitud $l : A- > R$ donde $R \geqslant 0$ (es decir, l es un conjunto de longitudes de arco de valores reales mayores o iguales a cero). Denotamos la longitud de un arco específico (i, j) por $l(i, j)$. Para el nodo i, definimos el conjunto de arco $FS(i) = \{(i, j)|arco(i, j) \in A\}$ como *su estrella directa*, y $SUC(i) = \{j|arc(i, j) \in A\}$ como el conjunto de los sucesores de i. La estrella directa es una estructura de datos de uso común que es una lista de los arcos que están generalizados en el nodo i y que están dirigidos a otros nodos. Nos referimos a la fuente de un camino más corto o la raíz de un árbol de ruta más corta **(STP (protocolo del árbol de expansión))** como nodo s. En el contexto del problema de trayecto más corto uno a uno, nos referimos al destino como nodo t.

Gallo y Pallottino[33] (1986) afirmaron que casi todos los algoritmos SPT de interés práctico pueden derivarse de un solo prototipo de método. Este prototipo puede estructurarse de la siguiente manera:

- [Paso 1:] Comience con cualquier árbol dirigido T enraizado en s y para cada $v \in N$, sea $d(v)$ la longitud de la trayectoria de s a v en T.
- [Paso 2:]Sea $(i, j) \in A$ un arco para el que $d(i) + l(i, j) < d(j)$, entonces ajuste el vector d fijando $d(j) = d(i) + l(i, j)$, y actualizar el árbol T reemplazando el arco actual incidente en el nodo j por el nuevo arco (i, j).
- [Paso 3:]Repita el paso 2 hasta $d(i) + l(i, j) \geqslant d(j)$ para cada $(i, j) \in A$.

La idea básica detrás de este prototipo es que uno primero comienza con un árbol T arraigado en el origen s y luego iterativamente actualiza este árbol reemplazando los arcos hasta que todos los arcos del árbol cumplan las condiciones de optimismo de Bellman (Bellman [39]1958), es decir, Para cada arco (i, j), la longitud de la trayectoria desde el origen s al nodo i más la longitud del arco (i, j) no es menor que la longitud de la trayectoria de s a j.[5]

1.4.0.1. La heurística A^* El algoritmo A^* fue presentado originalmente por Hart et al. [40](1968). Fue diseñado para resolver el problema de trayecto más corto entre un origen y un destino. Supongamos que además de la red de nodos y arcos, tenemos información en la que podemos estimar la distancia desde cualquier nodo hasta el destino. Formalmente, $h(i)=$ a la estimación de completar la ruta desde el nodo i al nodo de destino t. Un candidato para $h(*)$ es la medida de distancia euclidiana. La medida de distancia euclidiana es siempre un límite inferior de lo que tomaría en la distancia para completar un camino de i al nodo t en un plano cartesiano (para una región grande, sería la distancia del arco del gran círculo).[5]

El algoritmo A^* para resolver el problema de trayecto más corto uno a uno se puede describir como sigue:[5]

- [Paso 1:]Comience por establecer $d(i) = +\infty$ para cada $i \in N$; siguiente conjunto $g(s) = 0$ y $d(s) = g(s) + h(s)$,Finalmente, $S = \{s\}$;
- [Paso 2:]Selección de nodos: identificar el nodo $v \in S$ donde $d(v) \leq d(i)$ para todos $i \in S$; conjunto $S = S - \{v\}$.
- [Paso 3:]Criterios de detención: si $v = t$, detener como el camino más corto al destino t se ha encontrado; De lo contrario, vaya al paso 4.
- [Paso 4:]Expansión: para cada nodo w donde $arc(v, w) \in A$, si $g(w) > g(v) + l(v, w)$ entonces actualizar $g(w) = g(v) + l(v, W)$, fije $d(w) = g(w) + h(w)$ y cuando $w \notin S$ deje $S = S + \{w\}$; Después de que se hayan realizado todas las actualizaciones, vaya al paso 2.

Al igual que el algoritmo de Dijkstra[4], el algoritmo A^* mantiene un conjunto S de Nodos candidato (nodos que aún no han sido seleccionados como el siguiente nodo más cercano al origen) y aplica un método best-first para seleccionar de esta lista el siguiente nodo para expandir / escanear.

Cuando se ve desde una perspectiva miope, A^* es exactamente la misma que la de Dijkstra[4]; Sin embargo, A^* también incluye un componente orientado hacia el futuro, que es una estimación de la longitud para completar la ruta hacia el destino desde un nodo específico. Cuando el nodo i es colocado en la lista de candidatos S, el algoritmo A^* establece una etiqueta de distancia temporal con una función que consta de dos partes: $d(i) = g(i) + h(i)$ donde $g(i)$ es la longitud estimada del camino más corto desde el origen s al nodo i, y $h(i)$ es el longitud estimada de la trayectoria más corta desde i hasta el destino t. Si $h(i) = 0$ para todos nodos i, entonces A^* es esencialmente el mismo que SPA de Dijkstra[4]. Uno puede pensar en los costes de terminación estimados, $h(*)$, como tipo de penalización, en el que los nodos destino se penalizan menos que los nodos que están más lejos del destino.

El componente $h(*)$ de la etiqueta del nodo ayuda a dirigir el espacio de búsqueda hacia el destino, mientras que el algoritmo de Dijkstra[4] tiende a expandir el espacio de búsqueda uniformemente en todas las direcciones. Aunque ambos procedimientos identifican y etiquetan los nodos en distancia, Dijkstra[4] evalúa los nodos en orden de distancia creciente desde el origen y A^* en orden de la distancia creciente desde el origen más la estimación de la distancia de terminación al destino.

Algorithm 2.3.3

g = **A_star** (G, p, r, h)
Inputs: a. Graph G
b. Start node $p \in \mathcal{V}(G)$
c. Goal node $r \in \mathcal{V}(G)$
d. An admissible heuristic function $h : \mathcal{V}(G) \times \mathcal{V}(G) \to \mathbb{R}^+$
Outputs: a. A path connecting start vertex to goal vertex, $P = [p{=}\rho_1,\ \rho_2,\ \rho_3,\ \cdots,\ \rho_n{=}r]$

```
1  Initiate g: Set g(v) := ∞, for all v ∈ V(G)    // Minimum distance
2  Set g(p) = 0
3  Initiate f: Set f(v) := ∞, for all v ∈ V(G)    // f-values
4  Set f(p) = h(p)
5  Set Q := ∅    // Closed set (set of expanded nodes)
6  Set R := {p}    // Open set (candidate nodes for expansion)
7  while (R ≠ ∅  &&  r ∉ R)
8        q := argmin_{q'∈R}  f(q')    // Vertex to expand. R is maintained by a heap data-structure.
9        if (q == r)    // Goal vertex reached.
10             return  Reconstruct_Path (G, g, r)
11       R = R − q    // Remove q from open set.
12       Q = Q ∪ q    // Add q to closed set.
13       for each ({w |w ∈ N_G(q) and w ∉ Q})  // For each neighbor of q that's not in closed set
14            R = R ∪ w    // Add w to open set.
15            Set g' := g(q) + C_G([q, w])
16            if (g' < g(w))
17                 Set g(w) = g'
18                 Set f(w) = g' + h(w, r)
19 return [ ]    // Goal vertex is not reachable.
```

Figura 3.1: Representación en pseudocódigo del algoritmo de A^*. Tomado de[2]

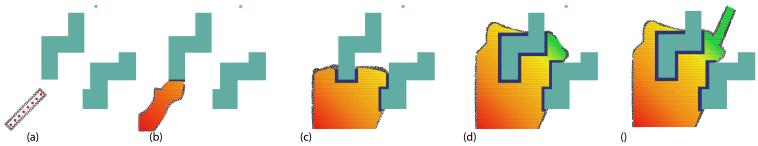


Figura 3.2: Ilustración del proceso del algoritmo A^*. El conjunto abierto esta marcado por los círculos azules vacíos. Tomado de [2]

[(a)] $iter = 0$, [(b)] $iter = 0.25$, [(c)] $iter = 0.5$, [(d)] $iter = 0.75$, [(e)] $iter = 1$.

Como vemos en los textos tomados de [5] tiene muchas aplicaciones y percepciones de su eficiencia. En el algoritmo de búsqueda planteado, muchos de sus procesos son muy parecidos a los del algoritmo A^*, ya que cuenta con una constante para reducir ciclos de procesamiento, llamada **Parágrafo: Heurística aplicada**. También cuentan con el direccionamiento de la ruta por medio de la recta entre los puntos de partida y de llegada; y por supuesto cuenta con la función que determina las distancias mas corta en los trayectos que lo acerquen al punto destino u objetivo sin colisionar por un obstáculo, según el valor de heurística.

2. MARCO METODOLÓGICO

2.1. DESCRIPCIÓN DEL PROCEDIMIENTO

2.1.1. Requerimientos

Para el correcto funcionamiento del algoritmo se requiere como "*ingrediente*" un entorno discreto, por lo cual se trabajará con imágenes binarizadas, en las cuales, ya a nivel de código el color negro se representará con el valor cero (0) y el color blanco con el el valor doscientos cincuenta y cinco (255). Dicho entorno es una *abstracción* de un entorno real, es decir, los obstáculos se modelan vía figuras geométricas como círculos y cuadrados y la unión de las mismas.

Adicionalmente, se seleccionó el formato `PNG`* como el formato de las imágenes de entrada a procesar por el algoritmo.

2.1.2. Descripción de la forma del entorno

Mediante los algoritmos de creación del entorno, referenciados en el proyecto de grado ***[1]***, se ubicaran de manera aleatoria, unas formas geométricas de tipo cuadrados, círculos y rectángulos, y las uniones entre ellas, que harán las veces de obstáculos. Los obstáculos se visualizaran en color blanco para poder identificar que es espacio de desplazamiento y espacio ocupado por obstáculo.

Ubicando de forma ordenada, en todo el espacio disponible de la imagen, unos puntos con coordenadas que ocuparán posiciones de manera equidistante entre los siguientes elementos: entre límites de obstáculos y los límites obstáculos con el contorno. Donde se discriminarán aquellos puntos que NO se ubican en obstáculos, los cuales serán los puntos efectivos para el siguiente proceso.

*Formato basado en algoritmos de compresión cuyas siglas significan: gráfico de red portátil

Una vez desplegados estos puntos, se producirá una expansión en cada uno de ellos; esta expansión se conocerá como ***una teselación en forma de rombo*** hasta llegar o encontrarse con otras teselaciones, o con obstáculos, o con los límites de la imagen de entorno.

Luego de ello, se ubicará el centro de masa en las teselaciones expandidas mediante coordenadas, para así tener una opción de punto de paso de la ruta mas apropiada.

Paso seguido, se desplegarán dos puntos que representarán respectivamente el punto de origen y el punto de meta. Es a partir de aquí, que se despliega el algoritmo de búsqueda, que determinará la ruta de menor coste o más eficiente. Se desplegará una recta entre los puntos de partida y de llegada. A partir en esa recta se seleccionarán los puntos o centroides adyacentes(laterales) a esa recta en una distancia menor o igual que la heurística. Esos puntos serán por los cuales se trace la ruta dependiendo de las trayectorias entre esos puntos. A partir de esas trayectorias, se determinará un camino posibles y validando sobre espacio libre o de configuración que no cruce ningún obstáculo. Esto se llevara a cabo mediante la medición de distancias parciales (en cada punto próximo de menor distancia con trayectoria valida) que presente la menor distancia con respecto al punto de llegada o meta, para así desplazarse de punto en punto con las mismas característica, hasta llegar a la meta.

2.1.2.1. Heurística aplicada

Dentro del algoritmo de búsqueda se ha introducido un valor constante con una utilidad muy importante. Esta utilidad es la de agilizar la etapa de procesamiento de búsqueda de ruta, ya que esta constante restringe el número de puntos disponibles para el desplazamiento. Dando así menor cantidad de puntos para la comparación.

Esta constante representa una distancia; una distancia euclidiana que referencia aquellos puntos que se encuentren por debajo de ese valor, en filas y columnas, harán parte de la colección de puntos por los cuales puede pasar la ruta seleccionada.

Esta característica adicional a la de puntos cercanos a una recta entre los puntos de partida y de llegada, orientan o enfocan la búsqueda de ruta, ademas que reducen los tiempo de procesamiento si la ruta es corta.

La descripción matemática de la acción podría declararse de la siguiente manera:

Sea l_n una colección de distancias euclidianas de todos los puntos disponibles en el entorno. Sea (x, y) las coordenadas de ese punto al cual hace referencia esa distancia parcial.

$A = \{l_n(x, y) : \|l_n(x, y)\| < Heuritica\}$

La imagen (Figura 4.1) representa esta característica.

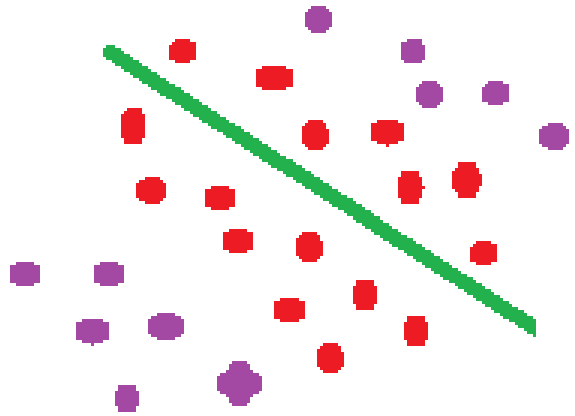
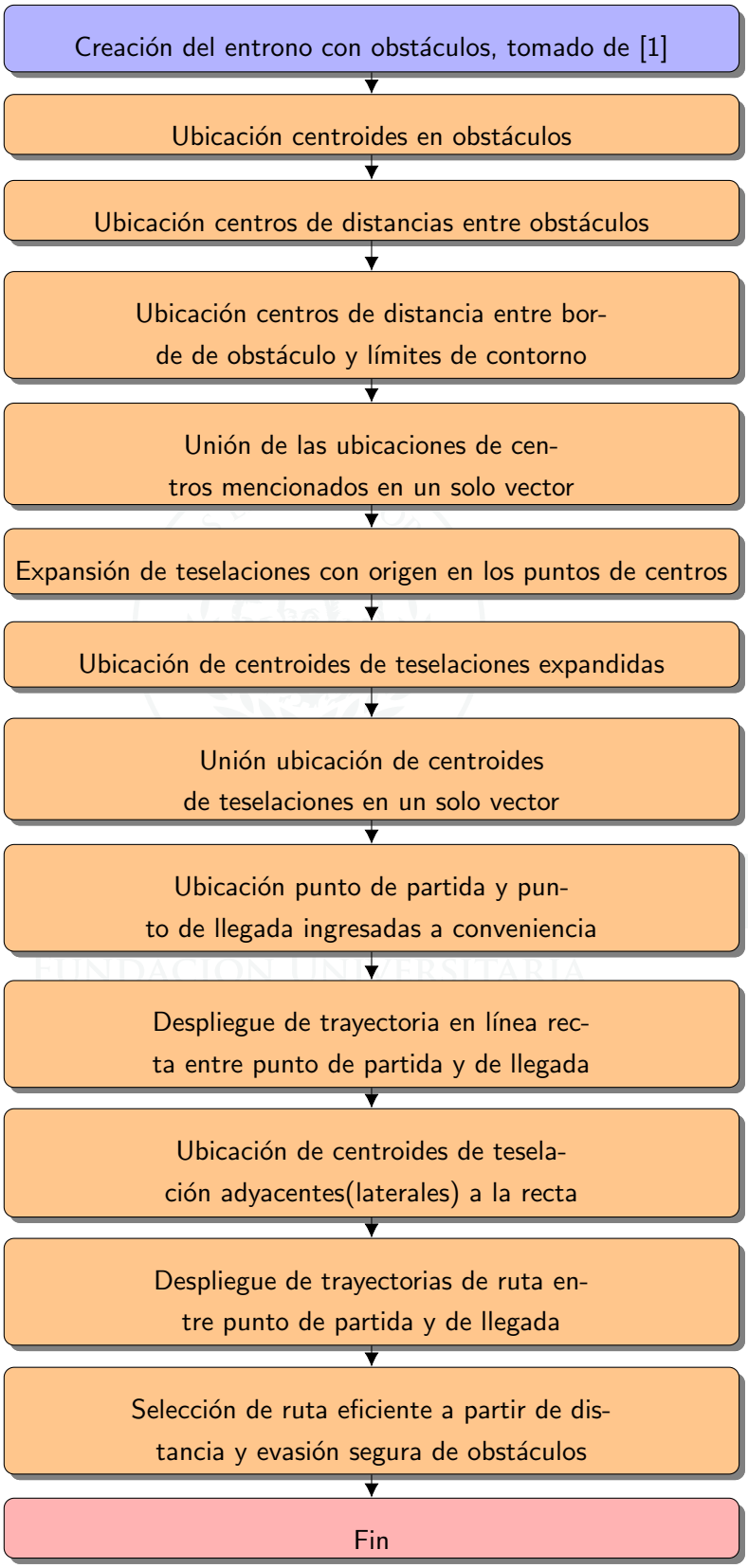
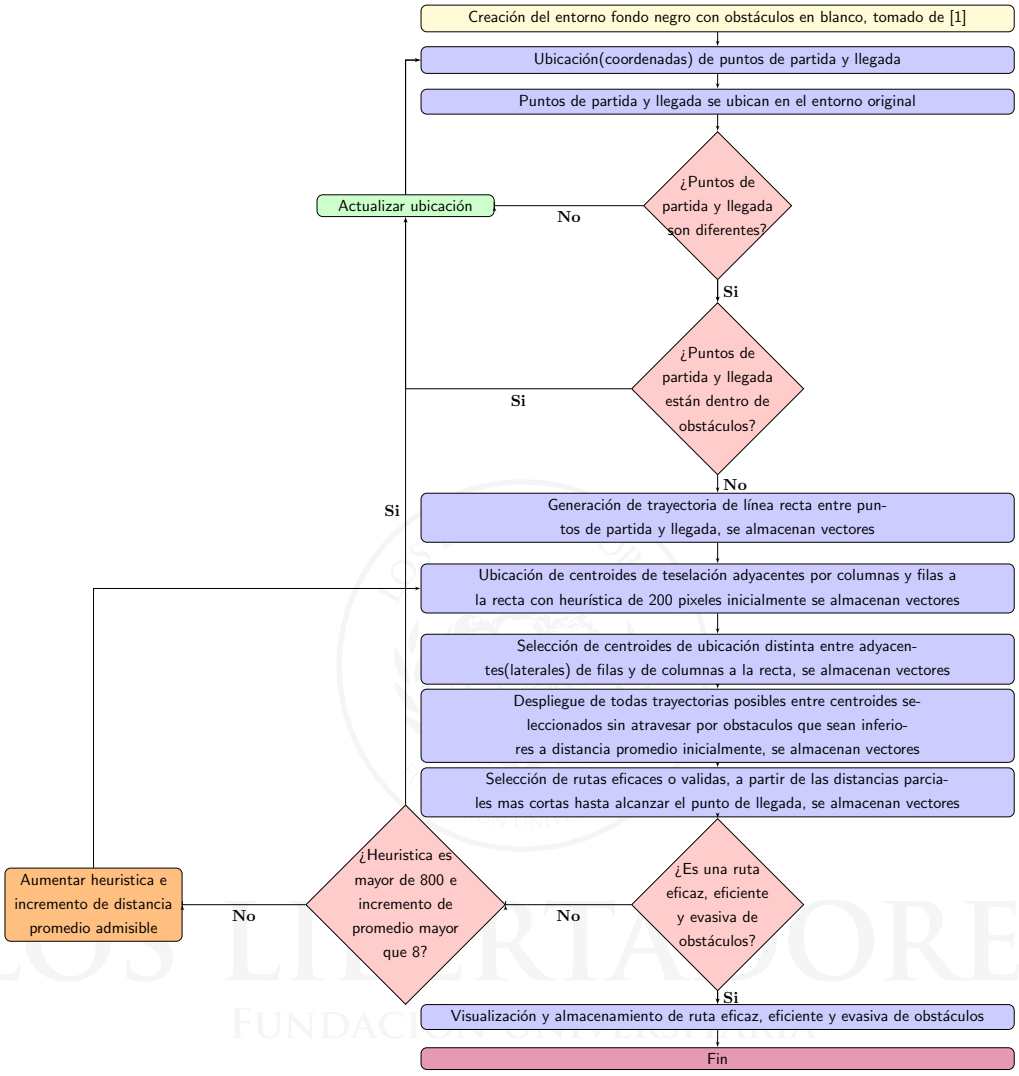


Figura 4.1: Demostracion de Heuristica en el algoritmo, puntos rojos:distancia menores que heurítica. puntos morados:distancias mayores que heurítica. Fuente autor

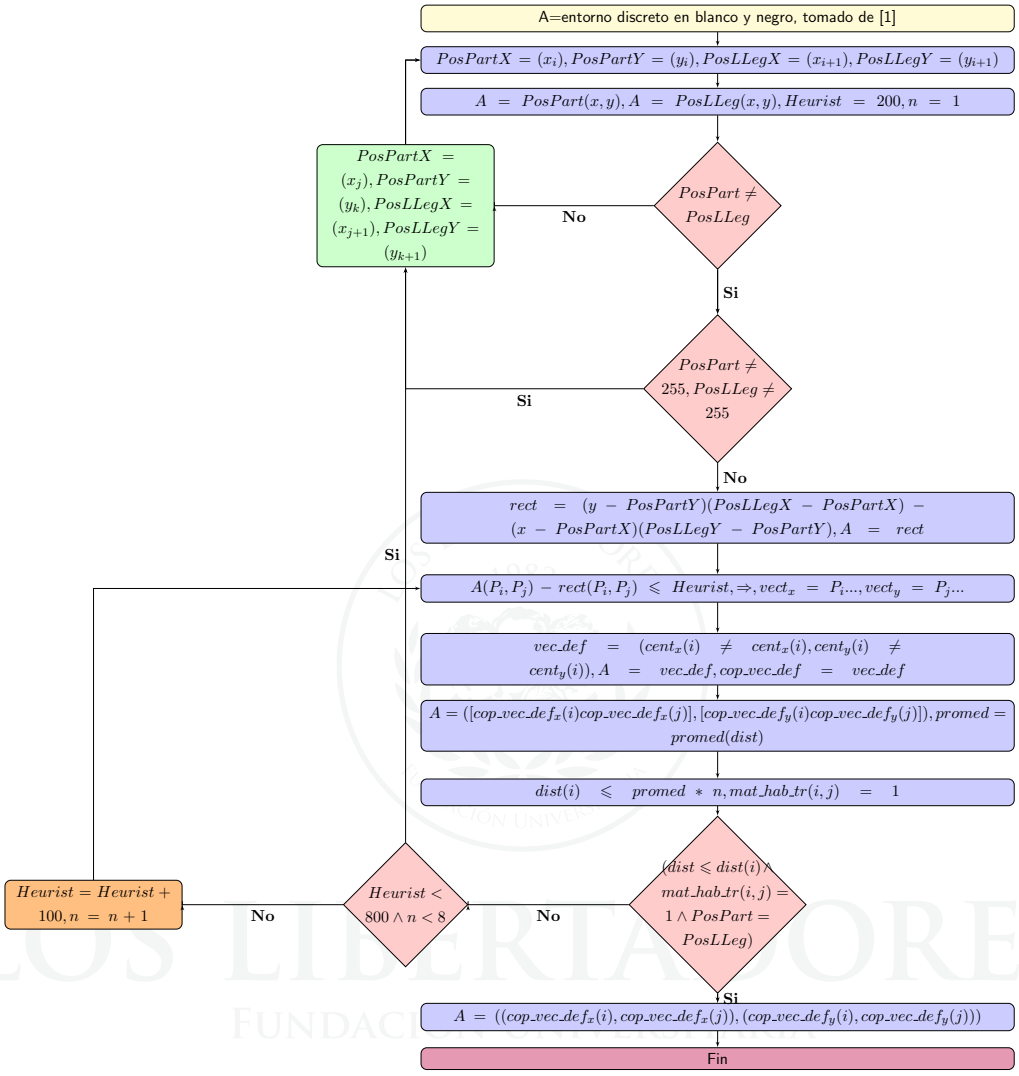
2.1.3. *Diagrama de bloques de simulación de algoritmo de búsqueda*



2.1.4. Diagrama de flujo de algoritmo de búsqueda



2.1.5. *Diagrama de flujo de algoritmo de búsqueda*



3. SIMULACIÓN DEL ALGORITMO

3.0.1. Simulación

A continuación se demostrara el funcionamiento del algoritmo por secciones, cada una desarrollada en `Matlab 2013b`. Cada una de estas secciones esta acompañada de la descripción del proceso, el pseudocódigo del algoritmo implementado, y una imagen del proceso realizado sobre el entorno.

La simulación del algoritmo requiere de 15 secciones para obtener los resultados necesarios, a continuación el listado de secciones que orienta el proceso y su descripción:

1. Creación del entrono fondo negro con obstáculos en blanco.
2. Ubicación de puntos de partida y llegada.
3. Puntos de partida y llegada se ubican en el entorno original.
4. ¿Puntos de partida y llegada son diferentes?.
5. Actualizar ubicación.
6. ¿Puntos de partida y llegada están dentro de obstáculos?.
7. Actualizar ubicación.
8. Generación de trayectoria de línea recta entre puntos de partida y llegada, se almacenan vectores.
9. Ubicación de centroides de teselación adyacentes(laterales) por columnas y filas a la recta con heurística de 200 pixeles en distancia inicialmente y distancia promedio incrementada en 1 inicialmente, se almacenan vectores.
10. Selección de centroides de ubicación distinta entre adyacentes de filas y de columnas a la recta, se almacenan vectores.
11. Despliegue de todas trayectorias posibles entre centroides seleccionados sin atravesar por obstáculos, se almacenan vectores.
12. Selección de rutas eficaces o validas,a partir de distancias cortas parciales se almacenan vectores.
13. ¿ Es una ruta eficiente y evasiva de obstáculos?.
14. Actualizar ruta.

15. Visualización y almacenamiento de ruta eficiente y evasiva de obstáculos.

3.0.1.1. Sección 1. Creación del entrono fondo negro con obstáculos en blanco:

La primera sección consiste en la creación del entorno. A partir del modelo presentado en el proyectos de grado ***[1]***. Se desarrollo el entorno como se describe, en la sección "RESULTADOS", primera parte: *Creación de un entorno aleatorio.*

En esta parte del algoritmo se definieron las dimensiones del entorno y a partir de geometría Euclidiana, se generó de manera aleatoria la cantidad de obstáculos, el tamaño de cada uno, la posición en el entorno y la forma. En la Figura 6.1 se puede observar un entorno generado por el algoritmo, donde las áreas blancas equivalen a los obstáculos y el área de color negro al espacio de configuración.
Cabe aclarar que este proceso contempla también, todo el desarrollo de ubicación de puntos de desplazamiento los cuales serán los orígenes de expansión de las teselaciones, del mismo modo la ubicación de los centroides de masa de las teselaciones expandidas, que de forma definitiva serán los puntos validos para la ruta; todo almacenado en vectores para el próximo algoritmo. En el Algoritmo 1 se explica una parte del proceso utilizado y los resultados en Figura 5.1.

Algorithm 1 Creación del entorno

Require: **ENTRADA** Q = Número de obstáculos
Require: **ENTRADA** P(x,y) = Posición del obstáculo
Require: **ENTRADA** T(1,Q) = Tamaño del obstáculo
Require: **ENTRADA** O = Contador actual de obstáculos
Require: **SALIDA** I = Imagen de salida

1: **while** $O \neq Q$ **do**
2: Posición = P
3: Definir forma aleatoria = F
4: **for** $i \leftarrow 1$ to $T(1, Q)$ **do**
5: Definir Contorno según F
6: $Contorno = 255$
7: **end for**
8: O = O + 1;
9: **end while**
10: **return true**

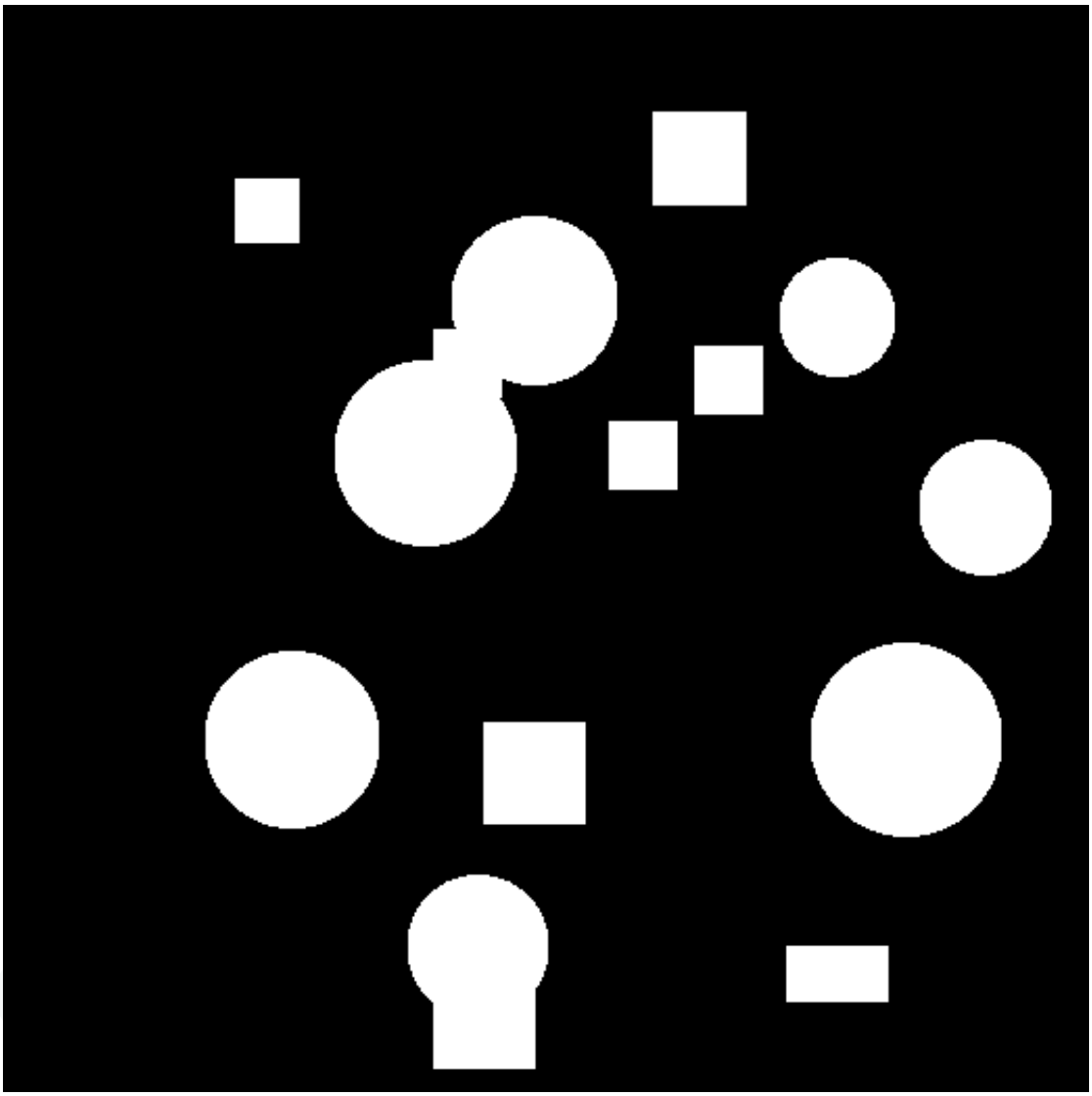


Figura 5.1: Entorno discreto aleatorio en tamaño, formas y ubicacion, creado por la Sección 1.

3.0.1.2. Sección 2. Ubicación(coordenadas) de puntos de partida y llegada: Esta sección hace parte de primer algoritmo ($Director_busqueda_IG_3$) donde se debe ingresar las coordenadas seleccionadas que determinen el punto de partida y el de llegada; para los cuales se necesita determinar una ruta. A continuación el algoritmo 2 demostrará el proceso y los resultados en Figura 5.2.

Algorithm 2 Ubicación(coordenadas) de puntos de partida y llegada

Require: **ENTRADA** I = Imagen de entrada
Require: **ENTRADA** Ux = Vector de centroides de teselaciones.
Require: **ENTRADA** Uy = Vector de centroides de teselaciones.
Require: Gx = Copia de vector Ux.
Require: Gy = Copia de vector Uy.
Require: iniciox = variable para almacenar las coordenadas del punto de partida.
Require: inicioy = variable para almacenar las coordenadas del punto de partida.
Require: finx = variable para almacenar las coordenadas del punto de llegada.
Require: finy = variable para almacenar las coordenadas del punto de llegada.

{**comentario:**Se ingresan las coordenadas de los punto de partida y llegada ($Director_busqueda_IG_3$):}

1: PosPartX=...
2: PosPartY=...
3: PosLLegX=...
4: PosLLegY=...
5: **return true**

3.0.1.3. Sección 3. Puntos de partida y llegada se ubican en el entorno original:

En esta sección se pretende visualizar la ubicación de los puntos de forma gráfica(dependiendo del entorno de programación que se utilice o el formato que se programe uint8 (tonos de 0-255) o double (decimales menores que 1). Se recomienda el uso de formato uint8 para esta aplicación)en el entorno original para saber que sección de centroides de teselaciones se necesitarán. A continuación el algoritmo que representa la sección y los resultados en Figura 5.2.

Algorithm 3 Puntos de partida y llegada se ubican en el entorno original

Require: **ENTRADA** I = Imagen de entrada
Require: **ENTRADA** Ux = Vector de centroides de teselaciones.
Require: **ENTRADA** Uy = Vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**

1: I(iniciox, inicioy) = '*'
2: I(finx, finy) = '+'
3: —printf—(I)
4: **return true**

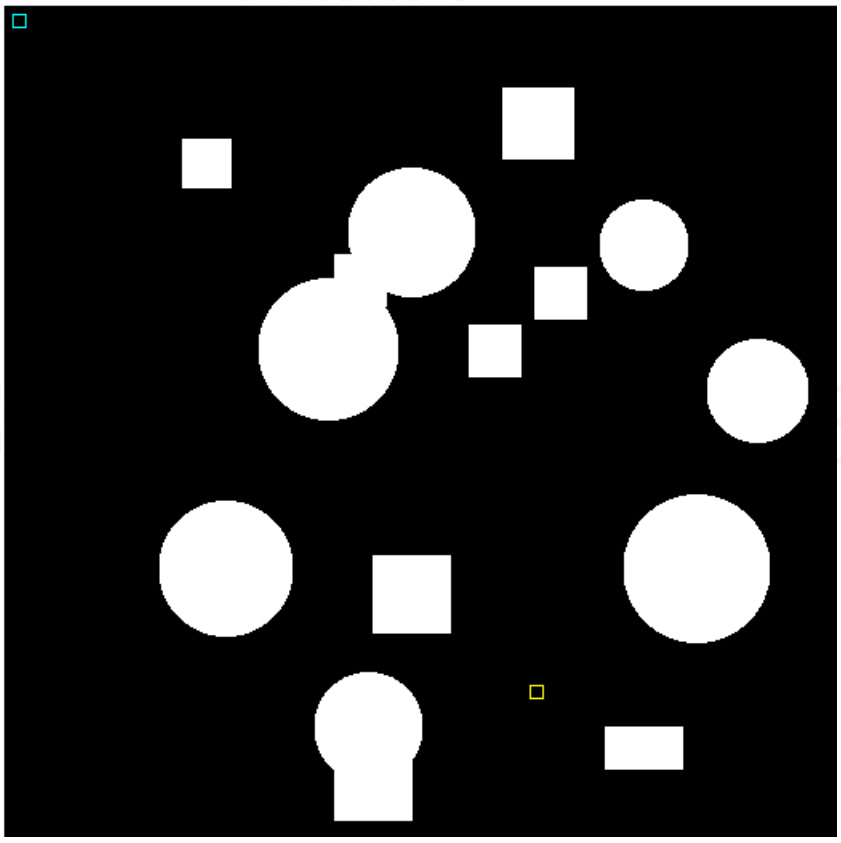


Figura 5.2: Entorno con puntos de partida y llegada. Sección 2 y 3.

3.0.1.4. Sección 4. y 5. ¿ Puntos de partida y llegada son diferentes? y Actualizar ubicación:

En esta sección de emplea una condicional para averiguar si las coordenadas de partida o

las de llegada, apuntan al mismo punto. En tal caso, no existe recorrido que realizar y no tienen sentido desarrollar el algoritmo. Si es así, se debe actualizar la ubicación de nuevas coordenadas en el entorno, de tal modo, que no direccionen las mismas coordenadas. Hasta tanto nos sean unas coordenadas validas no se sigue con el proceso. A continuación el algoritmo representativo de esta situación y los resultados en Figura 5.3.

Algorithm 4 ¿ Puntos de partida y llegada son diferentes? y Actualizar ubicación

Require: ENTRADA I = Imagen de entrada
Require: ENTRADA Ux = Vector de centroides de teselaciones.
Require: ENTRADA Uy = Vector de centroides de teselaciones.
Require: iniciox
Require: inicioy
Require: finx
Require: finy

1: I(iniciox, inicioy) = '*'
2: I(finx, finy) = '+'
3: —printf—(I)
4: **if** $inicioy \neq finy \wedge iniciox \neq finx$ **then**
5: **comentario:** sección de programación orientada al trazado entre los puntos de partida y llegada
{**comentario:**Lazo de retorno a la sección 2.}
6: **else**
7: Actualizar ubicación
8: **goto:** Ubicación de puntos de partida y llegada
9: **end if**
10: **return true**

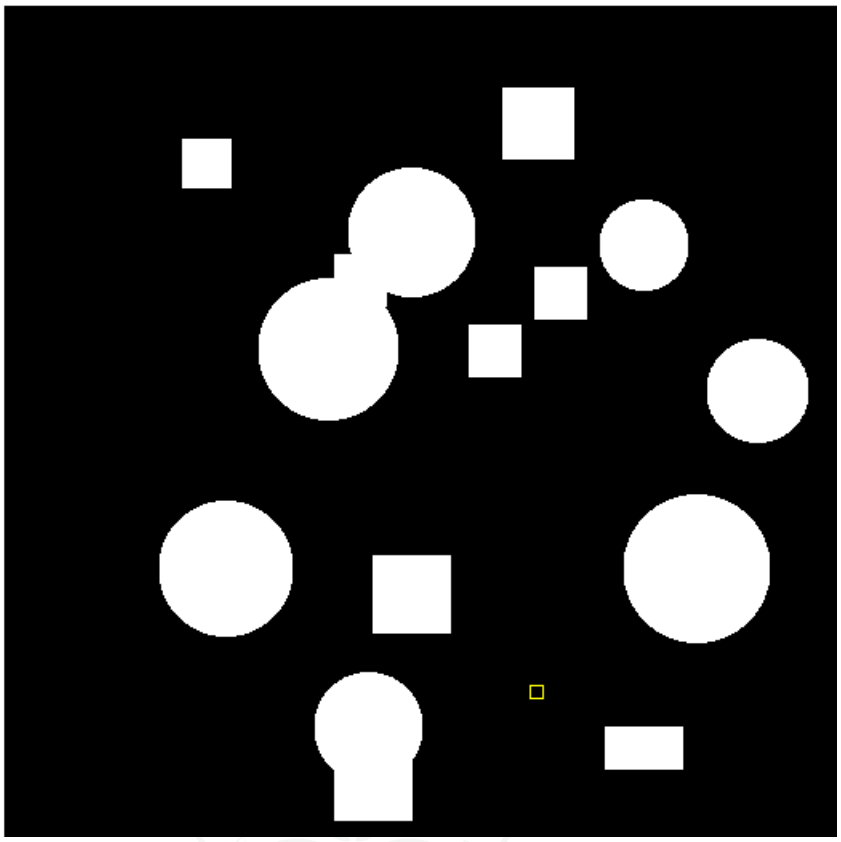


Figura 5.3: Entorno con puntos de partida y llegada en la misma coordenada(cuadro de color cian y amarillo en el mismo punto). Sección 4. y 5.

3.0.1.5. Sección 6. y 7. ¿ Puntos de partida y llegada están dentro de obstáculos? y Actualizar ubicación: En esta sección de emplea una condicional para averiguar si las coordenadas de partida o las de llegada, en este caso apuntan dentro del área de un obstáculo. Teniendo en cuenta las recomendaciones de la sección 3. (uso de formato unit8 negro=0, blanco=255). Si es así, se debe actualizar la ubicación de nuevas coordenadas en el entorno, de tal modo, que no direccionen hacia un obstáculo. hasta tanto nos sean unas coordenadas validas no se sigue con el proceso. A continuación el algoritmo representativo de esta situación y los resultados en Figura 5.4.

Algorithm 5 ¿ Puntos de partida y llegada están dentro de obstáculos? y Actualizar ubicación

Require: ENTRADA I = Imagen de entrada
Require: ENTRADA Ux = Vector de centroides de teselaciones.
Require: ENTRADA Uy = Vector de centroides de teselaciones.
Require: iniciox
Require: inicioy
Require: finx
Require: finy

1: I(iniciox, inicioy) = '*'
2: I(finx, finy) = '+'
3: —printf—(I)
4: **if** $I(iniciox, inicioy) \neq 255 \wedge I(finx, finy) \neq 255$ **then**
5: **comentario:** sección de programación orientada al trazado entre los puntos de partida y llegada
{**comentario:**Lazo de retorno a la sección 2.}
6: **else**
7: Actualizar ubicación
8: **goto:** Ubicación de puntos de partida y llegada
9: **end if**
10: **return true**

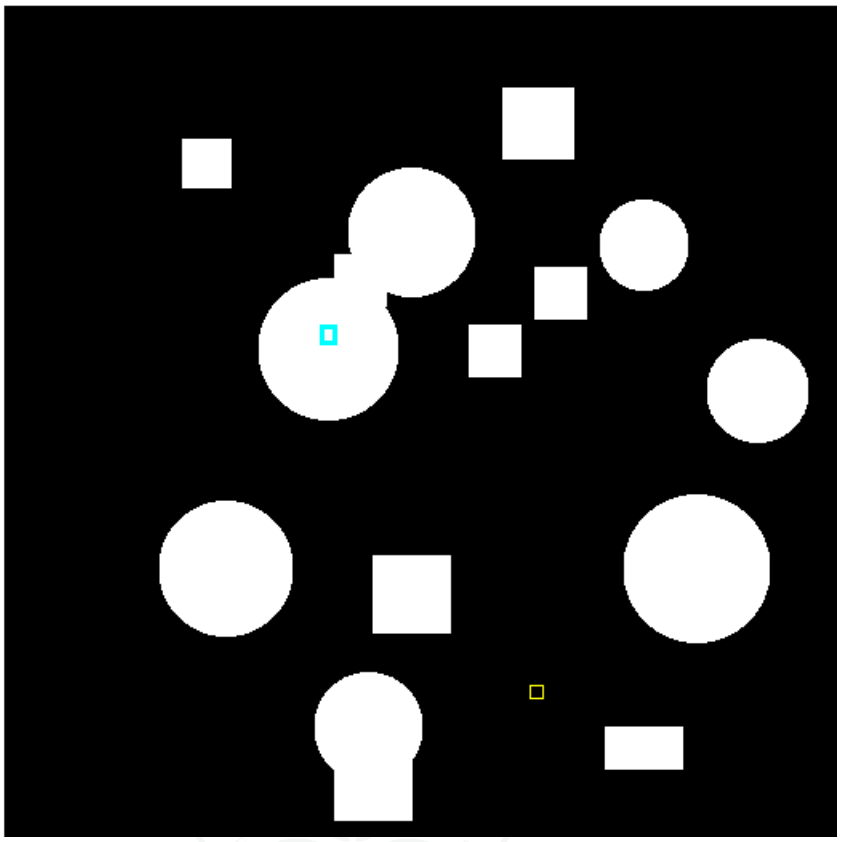


Figura 5.4: Entorno con puntos de partida o llegada dentro de obstaculo. Sección 6. y 7.

3.0.1.6. Sección 8. Generación de trayectoria de línea recta entre puntos de partida y llegada, se almacenan vectores: En esta sección se desarrolla el trazado de recta entre punto de partida y punto de llegada. Marcando cada pixel o punto con un color definido en toda la trayectoria. Luego se evidencia el trazado en la imagen del entorno original. A continuación el algoritmo que expone la explicación y los resultados en Figura 5.5.

Algorithm 6 Generación de trayectoria de línea recta entre puntos de partida y llegada, se almacenan vectores

Require: **ENTRADA** I = Imagen de entrada
Require: **ENTRADA** var = variable de almacenamiento.
Require: **ENTRADA** movx = variable de almacenamiento temporal de coordenadas.
Require: **ENTRADA** movy = variable de almacenamiento temporal de coordenadas.
Require: **contador** = variable de almacenamiento temporal.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **vtx** Vector de almacenamiento.
Require: **vty** Vector de almacenamiento.
Require: **SALIDA** S = Imagen de salida
{**comentario:CONTINUA EN EL SIGUIENTE ALGORITMO**}
1: **return true**

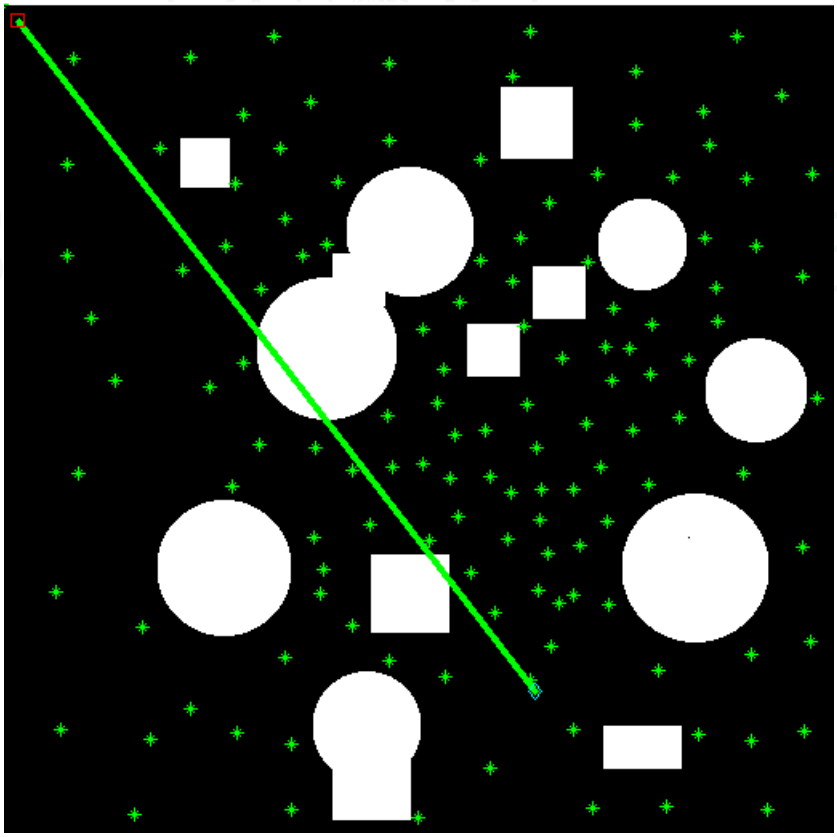


Figura 5.5: Entorno con puntos de partida y llegada y la trayectoria recta. Sección 8.

Algorithm 7 Generación de trayectoria de línea recta entre puntos de partida y llegada, se almacenan vectores

1: **if** $I(iniciox, inicioy) \neq 255 \wedge I(finx, finy) \neq 255$ **then**
2: $Distancia = sqrt(|\ (iniciox - finx)\ |^2 + |\ (inicioy - finy)\ |^2)$
{**comentario:**averiguar el tipo de pendiente de esa recta}
3: **if** Pediente tiene orientación positiva **then**
4: $var = 1$
5: $pend = redondeo(finy - inicioy)/(finx - iniciox)$
6: **else**
7: $var = 2$
8: $pend = redondeo(finx - iniciox)/(finy - inicioy)$
9: **end if**
10: I(iniciox, inicioy) = '*'
11: I(finx, finy) = '+'
12: S=I
13: —printf—(S)
14: contador=1
15: **while** $iniciox = finx \wedge inincioy = finy$ **do**
16: **if** $var = 1$ **then**
17: $movx = movx \pm 1$
18: $movy = ((contador) * (pend)) \pm inicioy$
19: **end if**
20: **if** $var = 2$ **then**
21: $movx = ((contador) * (pend)) \pm iniciox.$
22: $movy = movy \pm 1$
23: **end if**
24: I(movx, movy) = '.'
25: S=I
26: —printf—(S) {**comentario:**trazado de trayectoria almacenado en vectores}
27: vtx(contador)=movx
28: vty(contador)=movy
29: contador=contador+1
30: **end while**
31: **end if**
32: **return true**

3.0.1.7. Sección 9. Ubicación de centroides de teselación adyacentes(laterales) por columnas y filas a la recta con heurística de 200 pixeles (inicialmente) en distancia, se almacenan vectores: La característica de esta sección consiste en realizar, un barrido por los puntos o centroides de teselación; y averiguar cuales se encuentran cercanos a la recta descrita en la sección anterior. Empezando por las columnas y luego por la filas, hasta encontrar todos los puntos o centroides con distancias inferiores a 200 pixeles a la recta en todo su recorrido, por cada punto. Cabe anotar, que la medida de 200 pixeles (inicialmente), se debe a la dispersión de estos puntos en una forma conveniente, para obtener rutas posibles para el desplazamiento. Esta medida puede cambiar según criterio del programador del algoritmo y de la necesidad de llegar al punto de destino. Este valor de heurística se incrementa cuando la ruta no logra llegar a su fin; debido a esto, debe aumentar el margen para agregar mas puntos que sirvan para el desplazamiento. A continuación el algoritmo que expone la explicación y los resultados en Figura 5.6.

Algorithm 8 Ubicación de centroides de teselación adyacentes(laterales) por columnas y filas a la recta con heurística de 200 pixeles máximo en distancia, se almacenan vectores

Require: **ENTRADA** I = Imagen de entrada
Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.
Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **v1** Vector de unos (1) al tamaño necesario.
Require: **contador** = variable de almacenamiento temporal.
Require: **vtx** Vector de almacenamiento de trayectoria de la recta.
Require: **vty** Vector de almacenamiento de trayectoria de la recta.
Require: **SALIDA** S = Imagen de salida

1: contador=1
2: condición= verdadera
3: **while** $condicion = verdadera$ **do**
4: $vectX =\mid ((Gx - vtx(contador) * v1(1, tamano(Gx)))) \mid$
5: $vectY =\mid ((Gy - vty(contador) * v1(1, tamano(Gy)))) \mid$
6: vauxX= Ubicación de coordenadas de valor mínimo o igual a cero en Gx.
7: vauxY= Ubicación de coordenadas de valor mínimo o igual a cero en Gy.
8: vectx1= Ubicación con coordenadas de valor mínimo a la recta en vector Gx en columnas.
9: vecty1= Ubicación con coordenadas de valor mínimo a la recta en vector Gy en columnas.
10: vectx2= Ubicación con coordenadas de valor mínimo a la recta en vector Gx en filas.
11: vecty2= Ubicación con coordenadas de valor mínimo a la recta en vector Gy en filas.
12: contador=contador+1
13: **if** $contador \geq tamano(vtx) \vee contador \geq tamano(vty)$ **then**
14: condición=falsa
15: **end if**
16: **end while**
17: vectx1= vector con coordenadas de centroides cercanos a la recta en columnas.
18: vecty1= vector con coordenadas de centroides cercanos a la recta en columnas.
19: vectx2= vector con coordenadas de centroides cercanos a la recta en filas.
20: vecty2= vector con coordenadas de centroides cercanos a la recta en filas.
21: **return true**

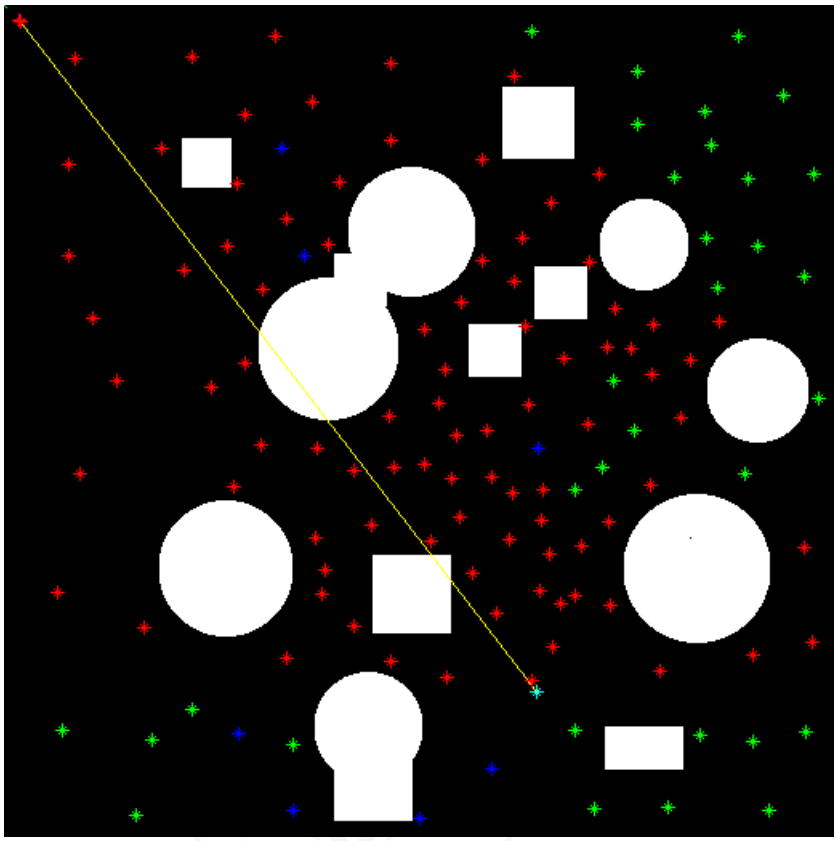


Figura 5.6: Ubicación de centroides de teselación adyacentes(laterales) por columnas y filas a la recta con heurística de 200 pixeles máximo en distancia, se almacenan vectores. Sección 9.

3.0.1.8. Sección 10. Selección de centroides de ubicación distinta entre adyacentes(laterales) de filas y de columnas a la recta, se almacenan vectores: En esta sección se filtran la recolección de los puntos o centroides de columnas y de filas que NO compartan coordenadas entre ellas(columnas y filas). De esta manera obtener los punto mas distantes entre si pero cercanos a la recta. A continuación el algoritmo que expone la explicación y los resultados en Figura 5.7.

Algorithm 9 Selección de centroides de ubicación distinta entre adyacentes(laterales) de filas y de columnas a la recta, se almacenan vectores

Require: **ENTRADA** I = Imagen de entrada
Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.
Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **contador** = variable de almacenamiento temporal.
Require: **vtx** Vector de almacenamiento de trayectoria de la recta.
Require: **vty** Vector de almacenamiento de trayectoria de la recta.
Require: **vtxdef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **vtydef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **SALIDA** S = Imagen de salida

1: contador=1
2: **for** $i \leftarrow 1$ to $tamano(vectx1)$ **do**
3: **for** $j \leftarrow 1$ to $tamano(vectx2)$ **do**
4: **if** $vectx1(i) \neq vectx2(j) \wedge vecty1(i) \neq vecty2(j)$ **then**
5: vtxdef(contador)=vectx2(j)
6: vtydef(contador)=vecty2(j) {**comentario:** visualización y almacenamiento de puntos filtrados cercanos a la recta}
7: I(vtxdef(contador), vtydef(contador)) = '*'
8: S=I
9: —printf—(S)
10: contador=contador+1
11: **end if**
12: **end for**
13: **end for**
14: **return true**

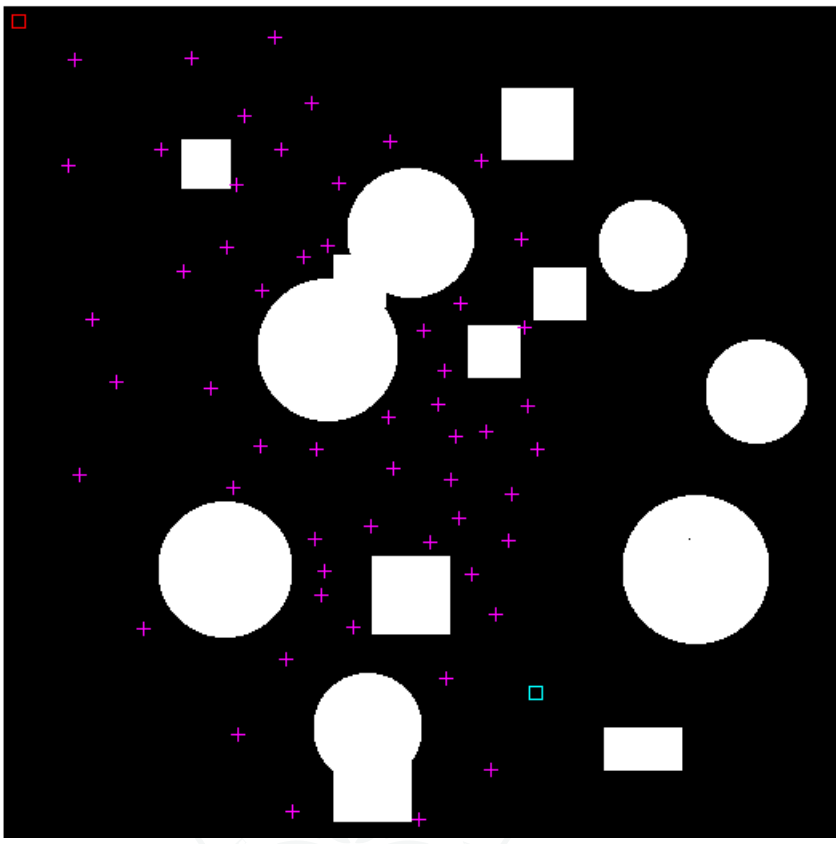


Figura 5.7: Selección de centroides de ubicación distinta entre adyacentes(laterales) de filas y de columnas a la recta, se almacenan vectores. Sección 10.

3.0.1.9. Sección 11. Despliegue de todas trayectorias posibles entre centroides seleccionados sin atravesar por obstáculos que sean inferiores a distancia promedio inicialmente, se almacenan vectores: En esta sección se procesa y se analiza las posibles rutas que se puedan emplear, para desplazarse desde el punto de partida y hacia el punto de llegada. La ruta puede resultar sencilla (a unos pocos centroides), como puede resultar bastante intrincada entre obstáculos. Para averiguarlo, se despliegan en linea recta desde el punto de partida, trayectorias que pasen por los centroides cercanos, hasta arribar al punto de llegada. Si se presenta el caso de que no llegue al punto de destino, se incrementa la cantidad de puntos disponibles. A continuación el algoritmo que expone la explicación y los resultados en Figura 5.8.

Algorithm 10 Despliegue de todas trayectorias posibles entre centroides seleccionados sin atravesar por obstáculos que sean inferiores a distancia promedio inicialmente, se almacenan vectores

Require: **ENTRADA** I = Imagen de entrada
Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.
Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **vtx** Vector de almacenamiento de trayectoria de la recta.
Require: **vty** Vector de almacenamiento de trayectoria de la recta.
Require: **vtxdef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **vtydef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: *movX* variable de almacenamiento punto a punto.
Require: *movY* variable de almacenamiento punto a punto.
Require: *promDist* variable de promedio extendido para filtrado de distancias largas.
Require: *matHabTr* matriz de unos y ceros que valida el trayectos entre punto que no atraviesa obstáculos.
Require: **SALIDA** S = Imagen de salida
{**comentario:CONTINUA EN EL SIGUIENTE ALGORITMO**}
1: **return true**

Algorithm 11 Despliegue de todas trayectorias posibles entre centroides seleccionados sin atravesar por obstáculos, se almacenan vectores

1: **for** $i \leftarrow 1$ to $tamano(vtxdef)$ **do**
2: **for** $j \leftarrow 1$ to $tamano(vtxdef)$ **do**
3: **if** $i < j \wedge distTrazad < promDist \wedge iniciox \neq finX \wedge inicioy \neq finY \wedge I(iniciox, inicioy) \neq 255 \wedge I(finX, finY) \neq 255 \wedge I(movX, movY) = 255$ **then**
4: {**comentario:** ($promDist$) se almacena una distancia promedio entre trayectos con el fin de enfocar la búsqueda y reducir tiempos de procesamiento}
5: {**comentario:** ($iniciox \neq finX \wedge inicioy \neq finY$...)es el condicional principal para realizar el trazado y con el cual se valida la búsqueda de los puntos para la ruta}
6: {**comentario:**($I(movX, movY) = 255$) esta condicional clasifica si el trazado atraviesa un obstáculo, 1 atraviesa, 0 no atraviesa}
7: $distTtrazad = sqrt(abs((vtxdef(i) - vtxdef(j)))^2 + abs((vtydef(i) - vtydef(j)))^2)$ {**comentario:** distancia euclidiana}
8: matHabTr(i, j)=0
9: **else**
10: matHabTr(i, j)=1
11: Sx=[vtxdef(i) vtxdef(j)]
12: Sy=[vtxdef(i) vtxdef(j)]
13: I(vtxdef, vtydef) = '*'
14: S=I
15: S=Sx,Sy
16: —printf—(S) {**comentario:** visualización y almacenamiento de trayectorias filtradas entre puntos de partida y de llegada}
17: **end if**
18: **end for**
19: **end for**
20: **return true**

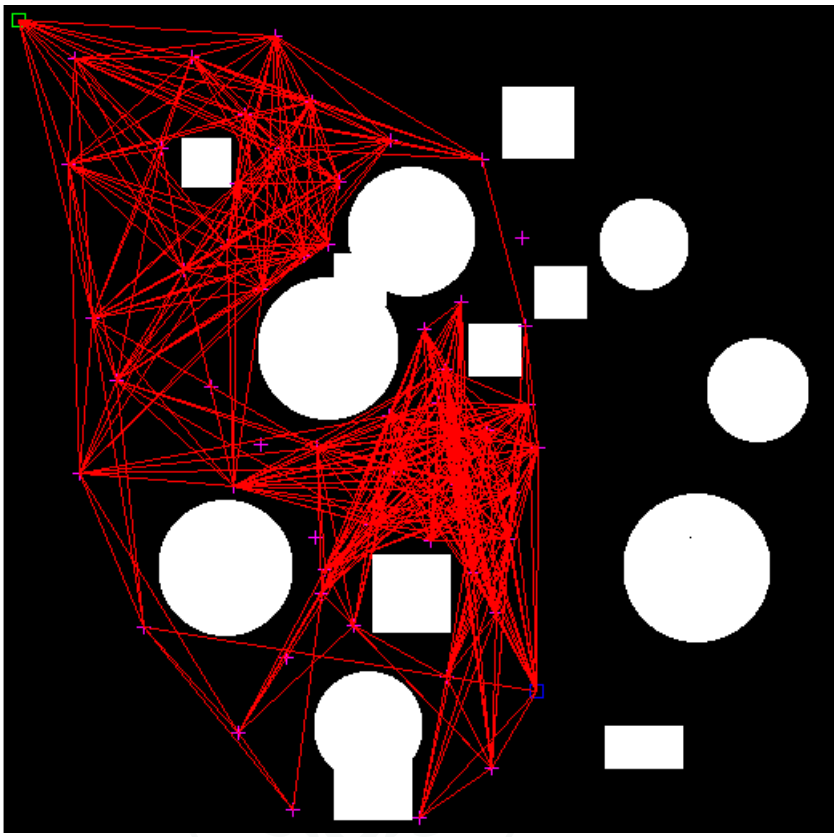


Figura 5.8: Despliegue de todas trayectorias posibles entre centroides seleccionados sin atravesar por obstáculos que sean inferiores a distancia promedio inicialmente, se almacenan vectores. ej: heurística=300, 1*promedio. Sección 11.

3.0.1.10. Sección 12. Selección de rutas eficaces o validas, a partir de las distancias parciales mas cortas hasta alcanzar el punto de llegada, se almacenan vectores: En esta sección se procesa y se analiza las posibles rutas eficaces, es decir, las rutas que llegaron a su destino a través de los punto adyacentes(laterales) por medio de trayectos hacia puntos cercanos a al destino, esto quiere decir, que evalúa la distancia del punto cercano con trayectoria valida, que mas lo acerque al punto final, luego de ello, avanza hacia ese punto y vuelve a preguntar, cual punto cercano con trayectoria lo acerca al punto final. A continuación el algoritmo que expone la explicación y los resultados en Figura 5.9.

Algorithm 12 Selección de rutas eficaces o validas, a partir de las distancias parciales mas cortas hasta alcanzar el punto de llegada, se almacenan vectores

Require: **ENTRADA** I = Imagen de entrada
Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.
Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **contador** = variable de almacenamiento temporal.
Require: **vtxdef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **vtydef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **SALIDA** S = Imagen de salida.
Require: mem_i=1 variable de almacenamiento de indices de la matriz de trayectoria validas.
Require: $mem2_i$=1 variable de memoria almacenamiento de indices de la matriz de trayectoria validas.
Require: distanciast=1000 distancia asignada para comparación.
Require: $visu_dist_x$=0 variable de almacenamiento de ruta eficiente.
Require: $visu_dist_y$=0 variable de almacenamiento de ruta eficiente.
Require: $memoria_dist$=1000 memoria de distancia asignada para comparación.
Require: $destino_x$ variables auxiliares
Require: $destino_y$ variables auxiliares
Require: $memoria_x$ variable auxiliar
Require: $memoria_y$ variable auxiliar
Require: $actualX$ variable auxiliar
Require: $actualY$ variable auxiliar
{**comentario:CONTINUA EN EL SIGUIENTE ALGORITMO**}
1: **return true**

Algorithm 13 Selección de rutas eficaces o validas, a partir de las distancias parciales mas cortas hasta alcanzar el punto de llegada, se almacenan vectores

1: mem_i=1
2: $mem2_i$=1
3: distanciast=1000
4: $visu_distx$=0
5: $visu_disty$=0
6: $memoria_dist$=1000
7: **while** $actualX \neq finX \vee actualY \neq finY$ **do**
8: **for** $i \leftarrow 1$ to $tamano(vtxdef)$ **do**
9: **if** $mat_hab_tr(mem_i, i) == 1$ **then**
10: $destino_x$ = vtxdef(i)
11: $destino_y$ = vtydef(i)
12: $distanciast = sqrt((destino_x - fin_x)^2 + (destino_y - fin_y)^2)$
13: **if** distanciast ¡ $memoria_dist$ **then**
14: $memoria_dist$ = distanciast
15: $memoria_x = destino_x$
16: $memoria_y = destino_y$
17: $mem2_i$=i
18: **end if**
19: **end if**
20: **end for**
21: matHabTr(mem_i,$mem2_i$) = 0
22: mem_i=$mem2_i$;
23: $visu_distx$=[$visu_distx$ [actualX $memoria_x$]]
24: $visu_disty$=[$visu_disty$ [actualY $memoria_y$]]
25: I(vtxdef, vtydef) = '*'
26: S=I
27: S=$visu_distx$ $visu_disty$
28: —printf—(S)
29: actualX = $memoria_x$;
30: actualy = $memoria_y$;
31: **end while**
32: **return true**

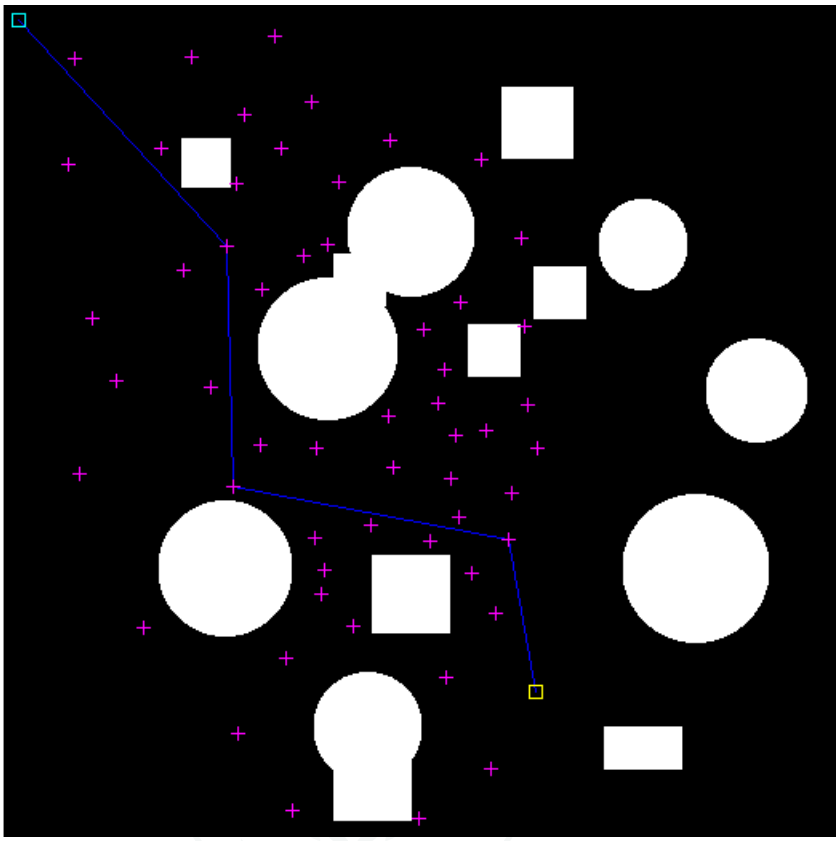


Figura 5.9: Selección de rutas eficaces o validas, a partir de las distancias parciales mas cortas hasta alcanzar el punto de llegada, se almacenan vectores. ej:heurística=300pixeles, 1*promedio. Sección 12.

3.0.1.11. Sección 13. y 14. ¿ Es una ruta eficaz, eficiente y evasiva de obstáculos? o Actualizar ruta: En esta sección se evalúa la eficiencia y el desplazamiento seguro del robot(que no colisione con obstáculos), con las rutas escogidas anteriormente. Debido a que en algunas ocasiones los puntos de partida y llegada se encuentran en lugares de difícil accesos, se necesita cambiar el valor de la heurística y el porcentaje del promedio por unos mas alto en valor. Cuando no se obtiene respuesta en un tiempo por conteo, se actualiza el valor de la heurística y un incremento del promedio, al igual que se retorna al algoritmo de selección de puntos adyacentes(laterales) a la recta, para así, expandir las zonas de desplazamiento, y lograr que la ruta llegue a su destino, el punto de llegada. Cabe mencionar que esta parte de algoritmo se ubica dentro del while de algoritmo anterior bajo las mismas condiciones. A continuación el algoritmo que expone la explicación.

Algorithm 14 ¿Es una ruta eficaz, eficiente y evasiva de obstáculos? o Actualizar ruta

Require: **ENTRADA** I = Imagen de entrada

Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.

Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.

Require: **iniciox**

Require: **inicioy**

Require: **finx**

Require: **finy**

Require: **contador** = variable de almacenamiento temporal.

Require: **vtx** Vector de almacenamiento de trayectoria de la recta.

Require: **vty** Vector de almacenamiento de trayectoria de la recta.

Require: **vtxdef** Vector de almacenamiento de centroides cercanos a la recta definitivos.

Require: **vtydef** Vector de almacenamiento de centroides cercanos a la recta definitivos.

Require: **SALIDA** S = Imagen de salida

Require: $visu_dist_x$=0 variable de almacenamiento de ruta eficiente.

Require: $visu_dist_y$=0 variable de almacenamiento de ruta eficiente.

Require: $Heurist$=0 variable de almacenamiento de nueva Heurística .

Require: $promDist$=0 variable de almacenamiento de incremento de promedio.

1: **if** $contador > 1000 \wedge actualX \neq finX \wedge actualY \neq finY$ **then**
2: Heurist=Heurist+100
3: promDist=promDist+1 {**comentario:** se retorna al algoritmo de selección de punto adyacentes a la recta con nuevo valor de heurítica e incremento de promedio}
4: ALGORITMO 10(PosPartX,PosPartY,PosLLegX,PosLLegY,Heurist,promDist)
5: contador=0
6: —printf—(CALCULANDO NUEVA TRAYECTORIA)
7: —printf—(variando Heuristica y promDist)
8: **end if**
9: {**comentario:** sección cuando no es posible con los cambio de heurística e incremento de promedio que la ruta llegue a su destino}
10:
11: **if** $Heurist \geqslant 800 \vee promDist \geqslant 8$ **then**
12: —printf—(error(PROGRAMA TERMINADO NO ENCOTRO RUTA))
13: **end if**
14:
15: **return** **true**

3.0.1.12. Sección 15. Visualización y almacenamiento de ruta eficaz, eficiente y evasiva de obstáculos: Como sección final se visualiza el plan de ruta eficaz, eficiente y evasiva de colisiones con obstáculos, que satisface los criterios del algoritmo de búsqueda planteado y los objetivos propuestos. A continuación el algoritmo que expone el resultado final y los resultados en Figura 5.10..

Algorithm 15 Visualización y almacenamiento de ruta eficiente y segura

Require: **ENTRADA** I = Imagen de entrada
Require: Gx = Copia de vector Ux, vector de centroides de teselaciones.
Require: Gy = Copia de vector Uy, vector de centroides de teselaciones.
Require: **iniciox**
Require: **inicioy**
Require: **finx**
Require: **finy**
Require: **contador** = variable de almacenamiento temporal.
Require: **vtx** Vector de almacenamiento de trayectoria de la recta.
Require: **vty** Vector de almacenamiento de trayectoria de la recta.
Require: **vtxdef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **vtydef** Vector de almacenamiento de centroides cercanos a la recta definitivos.
Require: **SALIDA** S = Imagen de salida

```
1: I(vtxdef, vtydef) = '*'
2: S=I
3: —printf—(S)
4: S=[visu_dist_x visu_dist_y]
5: —printf—(S)
6: —printf—(FIN)
7: return true
```

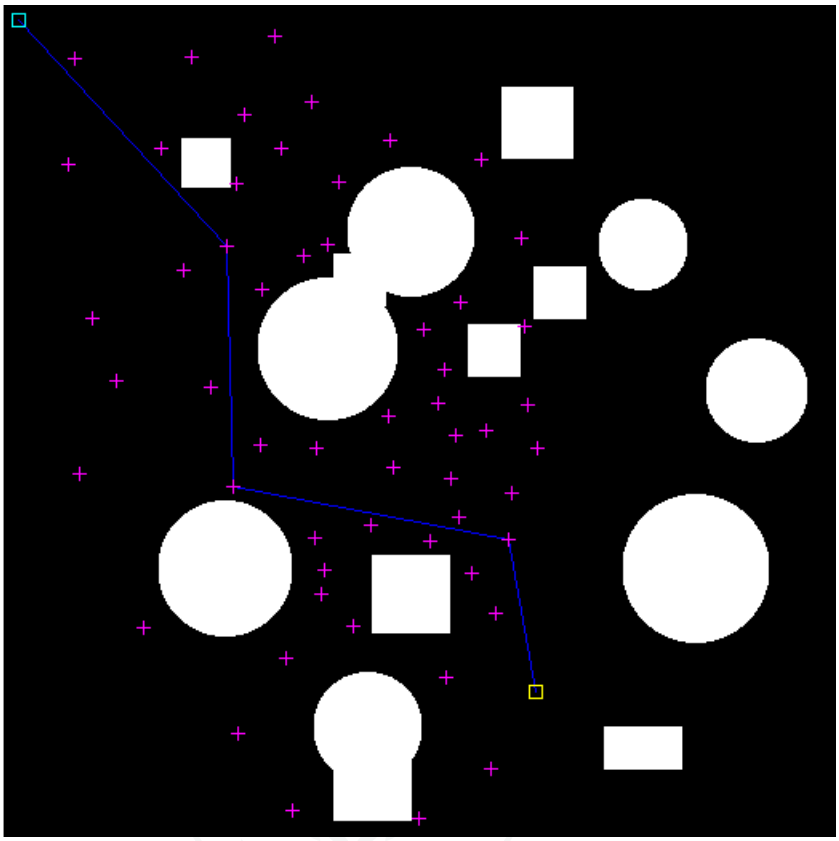


Figura 5.10: Visualización y almacenamiento de ruta eficaz, eficiente y evasiva de obstáculos. ej: heurística=300pixeles, 1*promedio. Sección 15.

4. ANÁLISIS DE RESULTADOS

Con el objetivo de analizar los resultados obtenidos, es necesario comparar los resultados entre varios aspectos importantes; con ello delimitamos el alcance del algoritmo, sus posibles desarrollos, y sus posibles aplicaciones. Estos aspectos son:

- Cantidad de obstáculos.
- Distancia de recorrido.
- Utilización de un solo obstáculo en todo el entorno en varios tamaños.
- Tiempos de ejecución.
- Tiempos de ejecución parcial.
- Comparación de aspectos con otro algoritmo de búsqueda de ruta.
- Cantidad de distancia en pixeles de puntos adyacentes a la recta (Heurística).
- Incremento de promedio de distancia como filtro y reductor de tiempos de ejecución.
- Eficiencia en la elección de ruta por distancia.

A continuación las conclusiones y evaluaciones de estos aspectos que determinaran el potencial del algoritmo.

4.1. CANTIDAD DE OBSTÁCULOS

La cantidad de obstáculos extiende el tiempo de ejecución considerablemente y dificulta los trayectos dependiendo de la posición de los punto de partida y llegada, ya que requiere de un aumento en los puntos de desplazamiento (heurística) y en la restricción de distancia promedio. Pero el algoritmo presenta respuesta ante la dificultad y traza la ruta.

4.2. DISTANCIA DE RECORRIDO

Se ubican puntos de partida y de llegada a distancias fijas de recorridos, para evaluar los tiempos de desempeño del algoritmo, en la selección de centroides de ubicación distinta entre adyacentes(laterales) de filas y de columnas a la recta. Esto valores de distancia pueden ser:180, 360, 540, 720 pixeles en diagonal. El algoritmo por medio del trazado obtiene la ruta sin atravesar obstáculos, lo cual lo hace eficaz; y también por evaluación de distancias, selecciona la linea recta hacia los puntos que mas lo acercan al punto de llegada. Esta dos consideraciones las realiza de forma optima, con la dificultad de lo engorroso del recorrido, requerirá mas tiempo y ciclos en procesamiento.

4.3. UTILIZACIÓN DE UN SOLO OBSTÁCULO EN TODO EL ENTORNO EN VARIOS TAMAÑOS

Debido a que el algoritmo esta diseñado a partir de la ubicación de los obstáculos y su centro de masa, no tiene dificultad para desplegar los puntos medios hacia el contorno, siempre que sea suficiente espacio para el desplazamiento(debe ser mayor a un pixel en linea recta), lo cual es suficiente para el recorrido que se necesite en todo el entorno disponible, siempre habrán puntos adyacentes(laterales) a la recta y dos trayectorias, cuya distancia evaluara el algoritmo buscando cual es la mas corta entre los puntos de partida y de llegada.

4.4. TIEMPOS DE EJECUCIÓN

Como se ha mencionado, los tiempos de procesamiento dependen de la máquina computacional. El algoritmo desarrollado se diseño para acortar estos tiempo, por medio de la no repeticiones de coordenadas en la ubicación de puntos; en la selección de punto necesarios; y en la aplicación de filtros de distancia de trayectorias. con el fin que ejecute lo esencial, pero todo esta sujeto a la ubicación de los puntos de partida y llegada en cada caso.

4.5. TIEMPOS DE EJECUCIÓN PARCIAL

la visualización de los procesos del algoritmo requiere bastante tiempo, lo cual es necesario para la ubicación de los puntos de partida y llegada, y evaluar la eficacia de la ruta, y la eficiencia del recorrido. El proceso de recorrido de las trayectorias punto a punto sin atravesar obstáculos, requiere una gran cantidad de ciclos, pero es absolutamente necesario para obtener una ruta segura.
Al igual de necesario, el proceso de teselación es indispensable y también requiere una gran cantidad de ciclos. Ademas de ser el proceso que mas requiere tiempo de ejecución.

4.6. COMPARACIÓN DE ASPECTOS CON OTRO ALGORITMO DE BÚSQUEDA DE RUTA

Debido a que es un algoritmo que conoce la posición de los obstáculos, tiene un desarrollo parecido al algoritmo A^*[5] ya que enfoca su recorrido hacia un objetivo fijo, para el caso del algoritmo planteado, se rige por una recta entre puntos de partida y llegada. Con la cual establece un trayectoria idónea. Otros algoritmos estiman el costo de desplazamiento y con ello despliegan los puntos de recorrido mas eficientes, pero para ello requieren de funciones heurísticas que establecen los menores costos, por ello requieren ciclos de procesamiento extensos y de información experimental o empírica respecto al entorno. Como resultado obtienen trayectorias mas seguras y eficientes, pero con respectos al algoritmo propuesto, que requiere de procesos menos extensos, se obtuvieron resultado equiparables en cuanto a eficacia, ademas que no necesita extensos bancos de memoria para guardar datos de recorrido y rutas fallidas para descarte.

4.7. CANTIDAD DE DISTANCIA EN PIXELES DE PUNTOS ADYACENTES A LA RECTA (HEURÍSTICA)

Quizá una de las limitaciones primarias durante el desarrollo del algoritmo, es el numero de puntos disponibles para el desplazamiento. Los cuales son dependientes del valor de la

heurística, ya que decide que puntos NO coincidentes entre columnas y filas, a una distancia establecida por debajo de su valor, harán parte del trayecto eficaz y eficiente. La finalidad del procesos es obtener los puntos mas dispersos posibles, y lograr una reducción en los tiempos de ejecución en la búsqueda de ruta eficaz. Pero debido a la dificultad de los trayectos en cada caso, este debe aumentar, y por ende retomar otra ves el proceso, desde el despliegue de puntos adyacentes hasta lograr que la ruta llegue a su destino. Estas razones identifican a esta particularidad del algoritmo como la sección que realiza la optimización del recorrido, por que despliega los puntos estrictamente necesarios para establecer trayectorias rectas extensas hacia los puntos cercanos al punto de llegada o meta.

4.8. INCREMENTO DE PROMEDIO DE DISTANCIA COMO FILTRO Y REDUCTOR DE TIEMPOS DE EJECUCIÓN

Con el fin de obtener menores tiempos de procesamiento, en aquellos casos en que los recorridos son cortos, se implemento la utilización del promedio de distancias entre todos los puntos, como valor limite de trayectorias entre puntos, ya que puede ser útil para agilizar el procesamiento. Esta distancia, se establece con la utilización de espacios métricos aplicados a unidades de imagen, que son pixeles, con la utilización de la distancia euclidiana que es aplicable a este tipo de entornos discretos.
Este valor se incrementara por el producto del promedio, por un valor incremental en una unidad, en cada ciclo que no encuentre una ruta eficaz, hasta lograrlo.

4.9. EFICIENCIA EN LA ELECCIÓN DE RUTA POR DISTANCIA

Con la finalidad de determinar la elección de ruta eficiente, se expondrá un ejemplo de ruta para corroborar las distancias seleccionas y el resultado gráficamente. A continuación la tabla de datos obtenida:

4.10. TABLA DE DISTANCIA ÓPTIMA DE EJEMPLO DE RUTA

Las columnas **vect_res_x_def** y **vect_res_y_def** representan los vectores donde se encuentran almacenados los puntos adyacentes a la recta seleccionados como opción de transito de ruta. La columna de **Distancias** representa la distancia entre las coordenadas respectivas y el punto final o de llegada.
Vemos que las coordenadas seleccionadas corresponden a las subrayadas en las tablas con el orden del item, de mayor a menor, de la siguiente manera: 1, 9, 19, 38, 44, 43 y 57. Esto lo podemos corroborar en la gráfica o figura 6.1 y 6.2, que se presenta después de las tablas; en ella aparecen también las coordenadas en cada caso, lo que concluye que toma el trayecto que mas lo acerca al punto final.

Tablas 1: Tabla de eficiencia de ejemplo de ruta

Item	**vect_res_x_def**	**vect_res_y_def**	**Distancias (pixeles)**
1	**10**	**10**	**529**
2	170	20	406
3	118	33	432
4	45	34	486
5	193	61	360
6	151	69	383
7	242	85	310
8	99	90	409
9	**174**	**90**	**352**
10	299	97	269
11	41	100	452
12	210	111	311
13	146	112	359
14	324	146	214
15	203	150	290
16	140	151	341
17	188	157	298
18	113	166	357
19	**162**	**178**	**308**
20	286	186	205
21	56	196	397
22	326	201	167
23	263	203	210
24	276	228	185
25	71	235	371
26	130	239	313
27	272	249	178
28	328	250	130

Tablas 2: Tabla de eficiencia de ejemplo de ruta

Item	**vect_res_x_def**	**vect_res_y_def**	**Distancias (pixeles)**
29	241	257	202
30	302	266	143
31	283	269	159
32	161	275	274
33	196	277	239
34	334	277	110
35	244	289	190
36	48	293	383
37	280	296	153
38	**144**	**301**	**286**
39	318	305	114
40	285	320	144
41	230	325	199
42	195	333	234
43	**316**	**334**	**113**
44	**267**	**335**	**162**
45	201	353	228
46	293	355	137
47	199	368	232
48	308	380	129
49	219	388	217
50	88	389	345
51	177	408	262
52	277	420	175
53	147	455	307
54	305	477	190
55	181	503	300
56	260	508	243
57	**429**	**333**	**0**

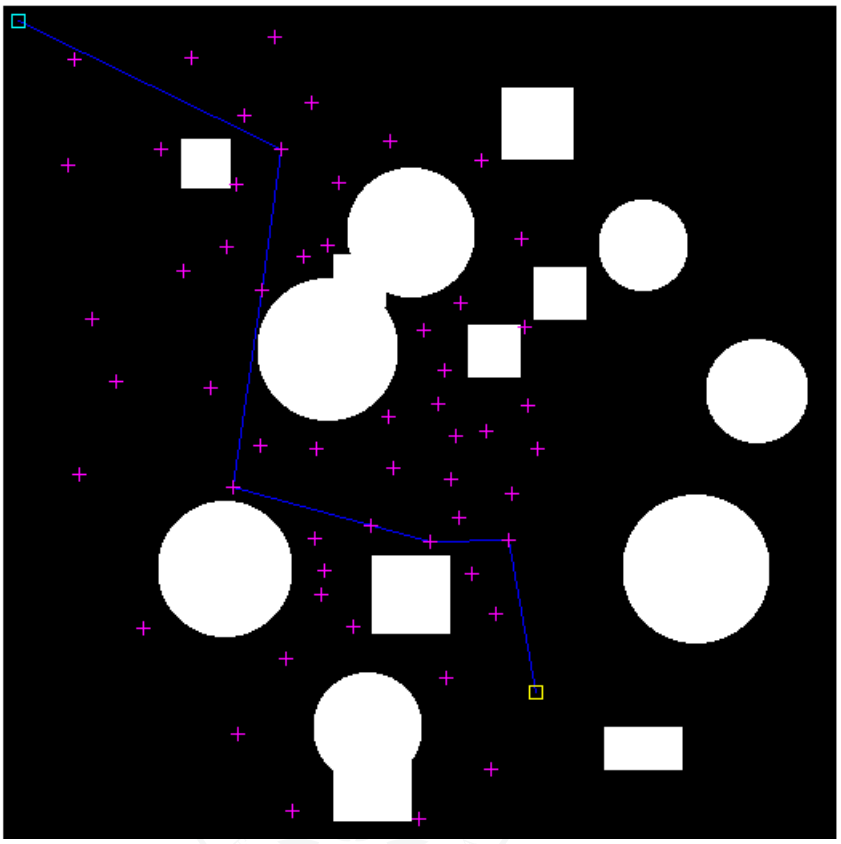


Figura 6.1: Gráfica que representa los puntos seleccionados para la ruta por distancia

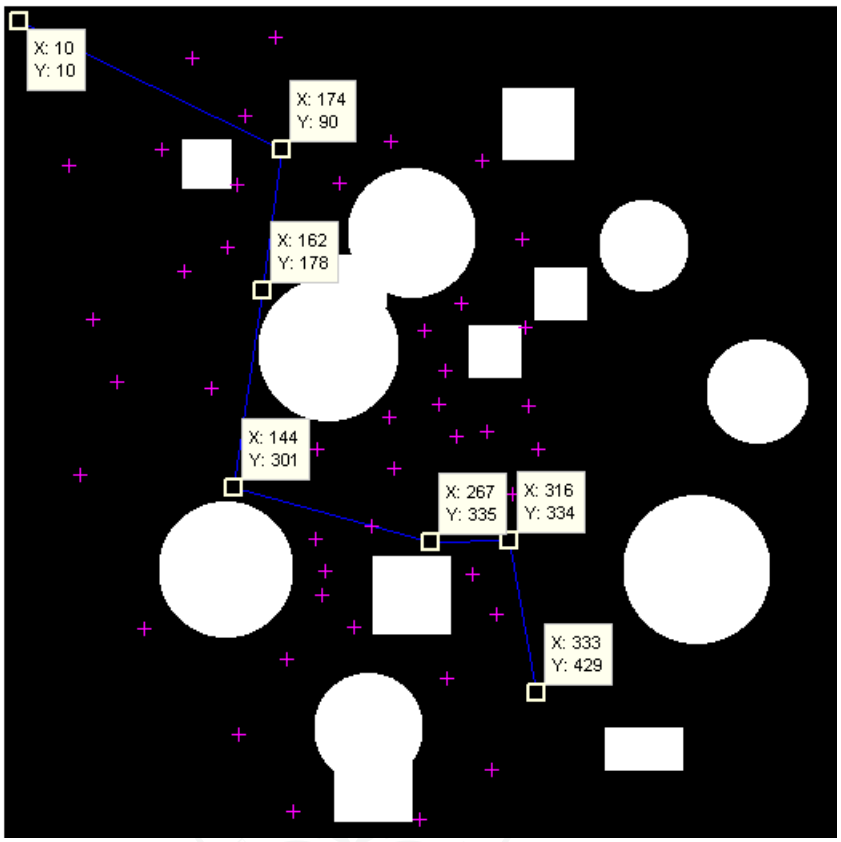


Figura 6.2: Gráfica que representa la elección de ruta por distancia

5. CONCLUSIONES Y TRABAJO FUTURO

Sin duda un factor determinante en la optimización de ruta, es la utilización de heurística. Pero al hacerlo, se interpone una restricción en el desarrollo del algoritmo de búsqueda, ya que la elección de los puntos adyacentes(laterales) a la recta puede descartar puntos que fuesen muy útiles para la ruta, pero que debido a la condicional de que sean distantes un determinado valor en ambos ejes a la recta; o sea cual sea, la condicional que se elija, se descartaran puntos de desplazamiento importantes que incrementarían la optimización de la ruta.

Para trabajos futuros, se podría plantear el reemplazo de esta heurística por una función matemática de optimización adaptativa*; una vez ubicados los puntos de partida y de llegada se desarrolle la recolección de parámetro de esa función y así determinar con bastante certeza una ruta optima.

Como se menciono en los resultados, los tiempos de ejecución son una característica propia de cada algoritmo dada su extension en procesamiento o en código. En el algoritmo desarrollado de búsqueda de ruta, los tiempo son bastante extensos, más puntualmente, en el reconocimiento del entorno con la ubicación de los puntos de origen de las teselaciones y las teselaciones propiamente dichas. No obstante, son procesos necesarios para asegurar un espacio suficiente para el desplazamiento, y este tiempo es también dependiente de las capacidades de la máquina subyacente.

Es importante, en trabajos futuros, encontrar otro tipo de instrumentos de medida de procesos de ejecución que no resulten influenciados por los factores antes mencionados. Una alternativa podría ser utilizando la información de entrada, de forma controlada, específica y cuantificada, analizando los resultados en cada caso y determinar la solución para agilizar el proceso global o los procesos parciales. Otra forma es reducir la extension de ejecución de operaciones cuando el trazado de ruta lo permita.

Una aclaración importante, a partir del glosario utilizado en el ámbito de generación de algoritmos de búsqueda de rutas, es que la solución encontrada con este algoritmo de búsqueda de

*Por decir un ejemplo

ruta, es una solución ***sub-óptima***. Puesto que la ruta seleccionada, evaluada visualmente por el observador, lograría determinar una ruta ideal, y en cuyo caso el algoritmo no determino detalles importante, detalles que no fueron especificados ni condicionados por que prolongarían ejecuciones o descartarían información valiosa en casos específicos.De modo que esta clasificación esta supeditada a la percepción humana, por lo cual se eligió también un entorno discreto que valida la aceptación dentro de unos limites.

6. REFERENCIAS Y BIBLIOGRAFÍA

[1] OSCAR,Penagos. *Diseño y simulación de un algoritmo de reconocimiento de entorno robótico basado en descomposición del plano en teselaciones usando frentes de onda con rombos* . Proyecto de grado de Ingeniero Electrónico. Bogotá. Colombia: Fundación Universitaria los libertadores. 2016.

[2] BHAPTACHARIA,Subhrajit. *Topological and geometric techniques in graph search-based robot planning* . Estado Unidos: Universidad de Pensilvania. 2012.

[3] PREPARATA, Franco and SHAMOS, Michael . *Computacional Geometry: An introduction* . pag 204-220. 1985.

[4] EDSGER, W. Dijkstra. *A note on two problems in connexion with graphs. Numerische Mathe- matik* . pag 1:269-271. 1959.

[5] W. ZENG, R. L. CHURCH *Finding shortest paths on real road networks: the case for A** . 08 Jun 2009.

[6] HANSEN Eric A. and ZHOU Rong.*Anytime heuristic search. Journal of Artificial Intelligence Research (JAIR)* . pag 28:267-297. 2007.

[7] STENTZ, A. *The focussed D* algorithm for real-time replanning. In Proceedings of the Interna- tional Joint Conference on Artificial Intelligence (IJCAI)* . pag 1652-1659. 1995.

[8] LIKHACHEV, Maxim and STENTZ, Anthony . *R∗ search. In Proceedings of the National Conference on Artificial Intelligence (AAAI)* . 2008.

[9] HART, P. E. NILSSON, N. J. and Raphael, B. *A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems. Science. and Cybernetics* . SSC-4(2):pag 100-107. 1968.

[10] MONTEMERLO, Michael and THRUN, Sebastian . *FastSLAM A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics* . pag 1-2. 2007.

[11] RIISGAARD, Søren and MORTEN,Rufus Blas *SLAM for Dummies* . pag 6,16,19-25.

[12] DURRANT-WHYTE, John J Leonard *Mobile robot localization by tracking geometric beacons* .

[13] CHEESMAN, Smith, Self, *Estimating uncertain spatial relationships in robotics* .

[14] ZUNINO *SLAM in realistic environments* .

[15] ROY *Foundations of state estimation* .

[16] WELCH, Bishop *An introduction to the Kalman Filter* .

[17] MUNKRE James. *Topology* . pag 264-269, 2000.

[18] B. LIU, *Intelligent Route Finding: Combining Knowledge, Cases and An Efficient Search Algorithm,* . in 12th European Conference on Artificial Intelligence, 1996, pp. 1–5.

[19] X. CAO, L. Chen, G. Cong, and X. Xiao, *Keyword-aware Optimal Route Search,* . VLDB, vol. 5, no. 11, pp. 1136–1147, 2012.

[20] M. Latendresse, M. Krummenacker, and P. D. Karp, *Optimal metabolic route search based on atom mappings,* . Bioinformatics, vol. 30, no. 14, pp. 2043–2050, 2014.

[21] C. Y. Hsiao, Z. L. Li, and C. S. Chiu, *A diamond search algorithm of travel route planning,* . in Proceedings - 2012 International Symposium on Computer, Consumer and Control, IS3C 2012, 2012, pp. 258–261.

[22] N. D. Pham and H. Choo, *Energy efficient expanding ring search for route discovery in MANETs,* . in IEEE International Conference on Communications, 2008, pp. 3002–3006.

[23] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli, *Age matters: efficient route discovery in mobile ad hoc networks using encounter ages,* . in Proceedings ACM MobiHoc'03 on Mobile ad hoc networking, 2003, pp. 257–266.

[24] J.-F. Cordeau, G. Laporte, and a Mercier, *Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows,* . J. Oper. Res. Soc., vol. 55, no. 5, pp. 542–546, 2004.

[25] V. Schmid, *Hybrid large neighborhood search for the bus rapid transit route design problem,* . Eur. J. Oper. Res., vol. 238, no. 2, pp. 427–437, 2014.

[26] B. A. Tolson and C. A. Shoemaker, *Dynamically dimensioned search algorithm for computationally efficient watershed model calibration,* . Water Resour. Res., vol. 43, no. 1, 2007.

[27] Y. KANZA, E. Safra, Y. Doytsher, and Y. Sagiv, *Heuristic algorithms for route-search queries over geographical data,* . in Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems - GIS '08, 2008, p. 1.

[28] X. JING and L. P. Chau, *An efficient three-step search algorithm for block motion estimation,* . IEEE Trans. Multimed., vol. 6, no. 3, pp. 435–438, 2004.

[29] Y. NAGATA and O. Bräysy, *A powerful route minimization heuristic for the vehicle routing problem with time windows,* . Oper. Res. Lett., vol. 37, no. 5, pp. 333–338, 2009.

[30] M. H. Baaj and H. S. Mahmassani, *Hybrid route generation heuristic algorithm for the design of transit networks,* . Transp. Res. Part C, vol. 3, no. 1, pp. 31–50, 1995.

[31] R. HUANG, *A schedule-based pathfinding algorithm for transit networks using pattern first search,* . Geoinformatica, vol. 11, no. 2, pp. 269–285, 2007.

[32] R. J. Szczerba, P. Galkowski, I. S. Glicktein, and N. Ternullo, *Robust algorithm for real-time route planning,* . Aerosp. Electron. Syst. IEEE Trans., vol. 36, no. 3, pp. 869–878, 2000.

[33] GALLO, G. and PALLOTTINO, S., *Shortest path methods: a unifying approach. Mat-*

hematical Programming Study ., 26, pp. 38–64, 1986.

[34] CHERKASSKY, B.V., GOLDBERG, A.V. and RADZIK, T. *Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming: Series A and B* ., 73,pp. 129–174, 1996,

[35] ZHAN, F.B. and NOON, C.E.,*Shortest path algorithms: an evaluation using real road networks. Transportation Science* ., 32, pp. 65–73, 1998.

[36] GOLDEN, L.B. and BALL, M., *Shortest paths with Euclidean distance: an exploratory model. Networks* ., 8, pp. 297–314, 1978.

[37] SHEKHAR, S., COYLE, M. and KOHLI, A., *Path computation algorithms for advanced traveler information system. In Proceeding of the International Conference on Date Engineering (IEEE Computer Society)* . Available online at: http://www.spatial.cs.umn.edu/paper_ list.html (accessed 7 September 2006) , 1993,.

[38] SHEKHAR, S. and FETTERER, A., *Path computation in advanced traveler information systems* . In Proceeding 9th International IEEE Conference on Intelligent Transportation Systems Available online at: http://www.spatial.cs.umn.edu/paper_ list.html (accessed 7 September 2006) , 1996,.

[39] BELLMAN, R., *On a routing problem. Quarterly of Applied Mathematics* ., 16, pp. 87–90, 1958,.

[40] HART, E.P., NILSSON, N.J. and RAPHAEL, B., *A formal basis for the heuristic determination of minimum cost paths* . IEEE Transactions on System Science and Cybernetics, 4, pp. 100–107, 1968,.

7. ANEXOS

7.1. CÓDIGO DE ALGORITMO DE BÚSQUEDA PROPUESTO COMPILADO EN MATLAB 2013B

A continuación se mostrarán los algoritmos simulados en Matlab, que demostraran el funcionamiento del algoritmo. El orden de ejecución se describe a continuación:

1. `Director_busqueda_IG_3`
2. `Tesela_romb_IG_definitivo_3`
3. `Busqueda_IG_definitiva_3`

Únicamente ubicando todos lo archivos (*.m*) en la carpeta de programas del entorno **Matlab 2013b** y compilando el archivo **[1.]** con los datos que solicita el programa.

7.1.1. Director_busqueda_IG_3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%IMPLEMENTACION Y COMPROBACION DE ALGORITMO DE BUSQUEDA DE RUTA EFICAZ
%Y EFICIENTE CON FRENTES DE ONDA TIPO ROMBO.
%FUNDACION UNIVERSITARIA LOS LIBERTADORES
%Facultad de ingenieria
%Ingenieria Electronica
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%CREADO POR: Ignacio Gutierrez Gomez 14/05/2017
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%**********************************************************
%FUNCION QUE DIRIGE LA UTILIZACION Y EJECUCION DE LAS OTRAS FUNCIONES
%PARA EL TRAZADO DE RUTA EFICAZ Y EFICIENTE.
%**********************************************************

clear all
close all, clc
tic
%formato de coordenadas:
%function Busqueda_IG_definitiva( PosPartX,PosPartY,PosLLegX,PosLLegY);
%PosPartX:coordenadas en x del punto de partida entre (20-500)
%PosPartY:coordenadas en Y del punto de partida entre (20-500)
%PosLLegX:coordenadas en x del punto de llegada entre (20-500)
%PosLLegY:coordenadas en y del punto de llegada entre (20-500)
%Valores menores o mayores que (20-500) se ubicaran en los limites
%del contorno, y no contara con espacio de giro el robot.
%(20 > x < 500)
%(20 > y < 500)
%por defecto X y Y aleatorio.
%-------------------------------------------------------------------
%Heurit:variable de distancia de la recta a los centroides cercanos en
%columnas o filas en pixeles.
%valor por defecto=200 pero se puede aumentar o disminuir entre
%(20< Heurist <500). entre mas centroides, mas demorado.
%-------------------------------------------------------------------
%los valores de PosPartX, PosPartY, PosLLegX, y PosLLegY  "DEBEN SER
%ENTEROS ENTRE 1 Y 519" y no deben ser los mismo ya que no funcionara
%el algoritmo.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% convension de marcadores de grafica
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Color: r (Red), g(Green), b(Blue), c(Cyan), m
% (Magenta), y (Yellow), k (blacK), w (White)

%  Marcadores: +, o, *, ., x, s (square), d
% (diamante), ^ v > o < (triángulos en distinto
% sentido), p (estrella 5 puntas, pentagram), h
% (estrella 6 puntas, hexagram)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


global PosPartX;
global PosPartY;
global PosLLegX;
global PosLLegY;
global const_prom

%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%asignacion de coordenadas por parte del ejecutor del programa
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 PosPartX=10
 PosPartY=10
 PosLLegX=333
 PosLLegY=429
```

```
65
66   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
67   %asignacion de heuritica inicial
68  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69  Heurist=200
70  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
71  %asignacion de valor de inicio de incremento de promedio
72  %de distancias
73  %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
74  const_prom=1
75  %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
76  %!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
77  %DECLARACION DE FUNCIONES NECESARIAS PARA LA EJECUCION DEL ALGORITMO
78  %!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
79  %funcion de genracion del entorno
80
81  %num_obst=Entorno_aleatorio_IG_3();
82
83  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
84  %funcion de generacionde puntos, teselaciones y limites de teselaciones
85
86  P_vali=Tesela_romb_IG_definitivo_3();
87
88  %#####################################################################
89  %varibles para obtener informacion entre las funciones
90  global centx;
91   global centy;
92  %>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
93  %funcion trazado de trayectorias viables para encontrar la ruta eficaz
94  %y eficiente.
95
96  VEC_val=Busqueda_IG_definitiva_3(PosPartX,PosPartY,PosLLegX,PosLLegY,Heurist);
97
98  %>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
99  %varibles para obtener informacion entre las funciones
100 global vect_res_x_def;
101 global vect_res_y_def;
102 global mat_hab_tr;
103 global cop_vect_r_x_d  cop_vect_r_y_d;
104
105 %//////////////////////////////////////////////////////////////////////
106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107 %INICIO DE MOTOR DE BUSQUEDA DE RUTA EFICAZ Y EFICIENTE
108 %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
109 %//////////////////////////////////////////////////////////////////////
110 %visualizacion de punto seleccionados bajo heuritica y promedio
111 I = imread('I.png');
112 figure(14)
113     imshow(I)
114     hold on
115      plot(vect_res_y_def,vect_res_x_def,'+m')
116
117          %asignacion de coordenadas de partida y llegada a variables
118         %auxiliares
119          actual_x= PosPartX;
120         actual_y= PosPartY;
121
122         fin_x= PosLLegX;
123         fin_y= PosLLegY;
124
125         %varibles para los indices de ubicacion en la metriz de
126         %de unos y ceros o de trayectorias que no atraviezan
127         %obstaculos
128         mem_i=1;
```

```
129          mem2_i=1;
130
131      %visualizacion de punto partida y llegada
132      plot(PosPartX,PosPartY,'sc')
133
134      plot(PosLLegX,PosLLegY,'sy')
135
136    %varible de distancia comparable en la primera seccion
137      distanciast=1000;
138       memoria_dist=1000; %variable de memoria de distancia comparable
139           %a gran distancia
140      visu_dist_x=0;
141      visu_dist_y=0;
142
143     %varible de conteo en tiempo en caso que no encuentre ruta hacia el
144     %punto de llegada
145      b=0;
146   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
147   %%%%%%%%%%%%%%%%%%MOTOR DE BUSQUEDA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
148   %$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
149  hold on
150         while(actual_y ~= fin_y || actual_x ~= fin_x)
151      %determina las distancias mas cortas al punto de llegada
152      % sin atravezar un obstaculo por medio de la matriz de unos y ceros
153      %%%%%"mat_hab_tr(i,j)"%%%%
154
155              for i = 1 : length(cop_vect_r_x_d)
156                  if(mat_hab_tr(mem_i,i) == 1)
157                      destino_x = cop_vect_r_x_d(i);
158                      destino_y = cop_vect_r_y_d(i);
159                      distanciast = sqrt((destino_x - fin_x)^2 + (destino_y -
fin_y)^2);
160                      if (distanciast < memoria_dist)
161                          %guarda la distancia menor en cada iteracion
162                          %hasta la ultima coordenda (X, Y)
163                          memoria_dist = distanciast;
164                          memoria_x = destino_x;
165                          memoria_y = destino_y;
166                          mem2_i=i;
167                      end
168                  end
169              end
170              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
171              %solucion en caso deque se encuentre en un ciclo
172              %en que la distancia mas corta se encentre equidistante
173              % al punto de llegada y sea el punto vecino, y cuando
174              %se desplace alli y no pueda avanzar por obstaculo o
175              %no tener registrado trayectoria por ese punto
176              % se devuelva al anterior y que tiene la misma dificultad
177              mat_hab_tr(mem_i,mem2_i) = 0;
178              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
179              %guarda memoria del indice de fila en la matriz para puntos
180              %cercanos en esa fila
181              mem_i=mem2_i;
182              %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
183              %VISUALIZACION DE RUTA EFICAZ Y EFICIENTE POR TRAZOS
184              %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
185              plot([actual_x memoria_x],[actual_y memoria_y],'b')
186              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
187              %Almacenamiento de ruta eficaz y eficiente por trazos
188              %//////////////////////////////////////////////////////////
189              visu_dist_x=[visu_dist_x [actual_x memoria_x]];
190              visu_dist_y=[visu_dist_y [actual_y memoria_y]];
191              %\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
```

```
192             %asignacion de nuevo trazado hasta llegar al punto de llegada
193             %############################################################
194             actual_x = memoria_x;
195             actual_y = memoria_y;
196             %;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
197             %Seccion en caso que no encuentre la ruta aumente la
198             %la heuritica y el nivel del promedio al
199             %multiplicarlo por la constante "const_prom" que
200             %incrementa en una unidad.
201             %;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
202             if(b>1000 && actual_y ~= fin_y && actual_x ~= fin_x)
203             Heurist=Heurist+100
204             const_prom=const_prom+1
205             VEC_val=Busqueda_IG_definitiva_3
    (PosPartX,PosPartY,PosLLegX,PosLLegY,Heurist);
206                 b=0;
207                 disp('%%%%%%%%%%%%%%%% CALCULANDO NUEVA TRAYECTORIA %%%%%%%%%%%%%%
    %%%')
208                 disp('%%%%%%%%%%%  variando Heuristica y const_prom  %%%%%%%%%%%%
    %%%')
209
210             end
211             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
212             %seccion encaso de NO encontrar una ruta hacia el punto de
213             %de llegada con el maximo de heuritica e incremento de promedio
214             %de distancia,TERMINE TODOS LOS CICLOS Y EL PROGRAMA
215             %CON UN ERROR VOLUNTARIO.
216             %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
217             b=b+1
218             Heurist
219             const_prom
220             if (Heurist>= 800 || const_prom >= 8)
221
222                 error('PROGRAMA TERMINADO%% NO ENCOTRO RUTA %%')
223             end
224
225         end
226
227   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
228   %almacenamiento de ruta eficiente como resultado del motor
229   %de busqueda
230   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
231     visu_dist_x(visu_dist_x==0) = [];
232     visu_dist_y(visu_dist_y==0) = [];
233
234   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
235   %Seccion para eliminar coordenadas repetidas
236
237   k=1;
238   n=length(visu_dist_x);
239   while k<=n
240   j=1;
241           while j<=n
242               if k~=j
243                   if (visu_dist_x(k)==visu_dist_x(j) && visu_dist_y(k)==visu_dist_y
    (j))
244                   visu_dist_x(j)=[];
245                   visu_dist_y(j)=[];
246                   n=length(visu_dist_x);
247                   end
248               end
249           j=j+1;
250           end
251   k=k+1;
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %VISUALIZACION DE DE RUTA EFICAZ Y EFICEINTE
 %************************************************************************
 figure(14)
    imshow(I)
    hold on
    plot(visu_dist_x,visu_dist_y,'b')
     plot(vect_res_y_def,vect_res_x_def,'+m')
      plot(PosPartX,PosPartY,'sc')

    plot(PosLLegX,PosLLegY,'sy')

    xlabel('%%TRAYECTORIA EFICIENTE %% FIN %%')

       %end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 disp('%%%%%%%%%%%%%%% TRAYECTORIA EFICIENTE %%%%%%%%%%%%%%%%%%')
 disp('%%%%%%%%%%%%%%%%%%%%%% FIN %%%%%%%%%%%%%%%%%%%%%%%')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% FIN  DE ALGORITMO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

toc
```

7.1.2. Tesela_romb_IG_definitivo_3

```
%--------------------------------------------------------------------------
%Tomado del proyectos de grado:
%Diseño y simulación de un algoritmo de reconocimiento de entorno robótico
%basado en descomposición del plano en teselaciones usando frentes de onda con
triángulos
%Por:OSCAR,Penagos
%--------------------------------------------------------------------------
%Para implementacion y comprobacion de algoritmo de busqueda de ruta eficaz
%y eficiente con frentes de onda tipo rombo.
%FUNDACION UNIVERSITARIA LOS LIBERTADORES
%Facultad de ingenieria
%Ingenieria Electronica
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%Modificado por: Ignacio Gutierrez Gomez 14/05/2017
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%**********************************************************
%FUNCION DE GENERACION DE PUNTOS DE REFERENCIA, TESELACION,
%LIMITE DE TESELACIONES Y UBICACION DE PUNTOS DEFINITIVOS PARA
%TRAZADO DE RUTA EFICAZ Y EFICIENTE.
%**********************************************************

function [centx,centy]=Tesela_romb_IG_definitivo_3()

%clear all, close all

global tesel
porcentaje = 100;
porcentaje = porcentaje / 100;
%%%%%%%%%%%%%% Inicialización de variables y lectura del entorno %%%%%%%%%
I = imread('I.png'); % Entorno
[f, c] = size(I); % Lectura del tamaño del entorno
[L, num]= bwlabel(I);

%----------------------------------------
I_pro=zeros(f, c);
areas_obs=zeros(1,num);
conti = 0;
centx_obs=zeros(1,num);
centy_obs=zeros(1,num);
cont_ast_c=0;
%----------------------------------------

figure(2)
imshow(I) % Grafica el entorno
hold  on

if (num>0)

                %buscar las secciones en blanco(centroides) de la imagen y las
cuenta
                for i = 1 : f
                    for j = 1 : c
                        if L(i,j) == 255
                            conti = conti + 1;
                        end
                    end
                end

                tesy = zeros(1,conti);
                tesx = zeros(1,conti);
                contt = 1;

                %ubica la coordenadas de los pixeles en blanco

```

```
                for k=1:num
                    for i = 1 : f
                        for j = 1 : c
                            if L(i,j) == 255
                                tesx(1,contt) = i;
                                tesy(1,contt) = j;
                                contt = contt + 1;
                            end

                        end
                    end
                end

 %================================================================
 % seccion de ubicacion de centro de masa en los obstaculos
 %================================================================
                tq_1 = 0;
                M_1 = 0;
                tq_2 = 0;
                M_2 = 0;
                c1=0;
                o = ones(f,1);
                p = ones(1,c);

                for k=1:num
                   for i = 1:f
                        for j = 1 : c
                           if L(i,j)==k
                              tq_1 = i*dot(p,L(i,:))+tq_1;%sumatoria de
columnasa
                              M_1 = dot(p,L(i,:))+M_1;
                           end
                        end
                   end

                    for i = 1:f
                        for j = 1 : c
                            if L(i,j)==k
                               tq_2 = j*dot(o,L(:,j))+tq_2;%sumatoria de filas
                               M_2 = dot(o,L(:,j))+M_2;
                            end
                        end
                    end;

                Cmx_1 = round(tq_1 / M_1);
                Cmx_2 = round(tq_2 / M_2);
                centx_obs(1,k)=Cmx_2;
                centy_obs(1,k)=Cmx_1;

                plot(Cmx_2,Cmx_1,'*m')


                tq_1 = 0;
                M_1 = 0;
                tq_2 = 0;
                M_2 = 0;
                c1=0;

                end
%================================================================
%----------------------------------------------------------------
%seccion ubica el centro de distancia entre los centros de masa
%de los obstaculos
%----------------------------------------------------------------
```

```
                %El centro entre los centroides de los obstaculos
                C_cx1=zeros(1,num);
                C_cy1=zeros(1,num);
                C_cx2=zeros(1,num);
                C_cy2=zeros(1,num);
                index_c1=0;
                index_c2=0;

                if(num>1)

                    for i=1:num
                        for j=1:num
                            if i<j
                                index_c1=index_c1+1;
                                C_cx1(index_c1)=centx_obs(i) + centx_obs(j);
                                C_cy1(index_c1)=centy_obs(i) + centy_obs(j);
                                C_cx1(index_c1)=round(C_cx1(index_c1)/2);
                                C_cy1(index_c1)=round(C_cy1(index_c1)/2);
                                if L(C_cy1(index_c1),C_cx1(index_c1)) == 0
                                    index_c2=index_c2+1;
                                    C_cx2(index_c2)=C_cx1(index_c1);
                                    C_cy2(index_c2)=C_cy1(index_c1);
                                    plot(C_cx1(index_c1),C_cy1(index_c1),'*g')
                                    cont_ast_c=cont_ast_c+1;
                                end
                            end
                        end
                    end
                end

%-----------------------------------------------------------------------
%Seccion que determina los limites en 8 puntos de los obstaculos
%[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
                %limites de obstaculos

                limitx_obs_ej=zeros(1,f);
                limity_obs_ej=zeros(1,f);
                limitx_obs_dia=zeros(1,f);
                limity_obs_dia=zeros(1,f);
                index_c7=1;
                l_1=0;
                l_2=0;
                l_3=0;
                l_4=0;
                l_5=0;
                l_6=0;
                l_7=0;
                l_8=0;
                h=10;


                for k=1:num

    %%%%%%%%%%%%%%%% para ejes
                   i=1;
                   condicion=true;
                   while (condicion)

                           if (L(centy_obs(k)+ i,centx_obs(k))== k )
                               l_1=L(centy_obs(k)+ i + 1,centx_obs(k));
                               limitx_obs_ej(1,(k*h))= centy_obs(k)+ i;
                               limity_obs_ej(1,(k*h))= centx_obs(k);
                               index_c7=index_c7+1;

```

```
                             end

                            if((L(centy_obs(k) , centx_obs(k)+i) == k))
                                l_2=L(centy_obs(k) , centx_obs(k)+i +1);
                                limitx_obs_ej(1,(k*h)+1)= centy_obs(k);
                                limity_obs_ej(1,(k*h)+1)= centx_obs(k) + i;
                                index_c7=index_c7+1;

                            end


                             if (L(centy_obs(k)- i,centx_obs(k))== k )
                                l_3=L(centy_obs(k)- i - 1,centx_obs(k)) ;
                                limitx_obs_ej(1,(k*h)+2)= centy_obs(k)- i;
                                limity_obs_ej(1,(k*h)+2)= centx_obs(k);
                                index_c7=index_c7+1;

                            end

                            if(L(centy_obs(k),centx_obs(k)-i)== k )
                                l_4=L(centy_obs(k),centx_obs(k)-i- 1);
                                limitx_obs_ej(1,(k*h)+3)= centy_obs(k);
                                limity_obs_ej(1,(k*h)+3)= centx_obs(k) - i;
                                index_c7=index_c7+1;

                            end


                                i=i+1;

                            if((l_1+ l_2+ l_3+ l_4)==0 || centy_obs(k)+ i==520 ||
centy_obs(k)- i ==1  ||centx_obs(k)+ i==520  ||centx_obs(k)- i ==1)

                                 condicion=false;
                            end

                    end

                i=1;

    %%%%%%%%%%%%%%%%%para diagonales
                    condicion=true;
                    while (condicion)
                            if (L(centy_obs(k)+i,centx_obs(k)+i)==k )
                                l_5=L(centy_obs(k)+i + 1,centx_obs(k)+i + 1);
                                limitx_obs_dia(1,(k*h))= centy_obs(k) + i;
                                limity_obs_dia(1,(k*h)) = centx_obs(k) + i;
                                index_c7=index_c7+1;

                            end
                            if (L(centy_obs(k)+i,centx_obs(k)-i)==k )
                                l_6=L(centy_obs(k)+i +1,centx_obs(k)-i- 1);
                                limitx_obs_dia(1,(k*h)+1)= centy_obs(k) + i;
                                limity_obs_dia(1,(k*h)+1) = centx_obs(k) - i;
                                index_c7=index_c7+1;

                            end
                            if (L(centy_obs(k)-i,centx_obs(k)-i)==k )
                                l_7=L(centy_obs(k)-i - 1,centx_obs(k)-i - 1);
                                limitx_obs_dia(1,(k*h)+2)= centy_obs(k) - i;
                                limity_obs_dia(1,(k*h)+2) = centx_obs(k)- i;
                                index_c7=index_c7+1;

                            end
```

```
                            if (L(centy_obs(k)-i,centx_obs(k)+i)==k )
                                l_8=L(centy_obs(k)-i - 1,centx_obs(k)+i + 1);
                                limitx_obs_dia(1,(k*h)+3)= centy_obs(k) - i;
                                limity_obs_dia(1,(k*h)+3) = centx_obs(k)+ i;
                                index_c7=index_c7+1;

                            end

                            i=i+1;
                            if(( l_5+ l_6+ l_7 +l_8)==0 || centy_obs(k)+ i==520 || centy_obs(k)- i ==1  ||centx_obs(k)+ i==520  ||centx_obs(k)- i ==1)

                                 condicion=false;
                            end
                    end



                 end

%[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
                %elimina los ceros de la matriz
                limitx_obs_ej(limitx_obs_ej==0) = [];
                limity_obs_ej(limity_obs_ej==0) = [];
                limitx_obs_dia(limitx_obs_dia==0) = [];
                limity_obs_dia(limity_obs_dia==0) = [];

%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%seccion que dibuja los asteriscos calculados para ejes y diagonales
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                   index_c8=1;
                   for s=1:(num*4)%dibuja los asteriscos calculados para ejes

                        if(limitx_obs_ej(1,s)~=0  && limity_obs_ej(1,s)~=0 && limitx_obs_ej(1,s)~=1  && limity_obs_ej(1,s)~=1)

                                limitx_obs_ej(1,index_c8)= limitx_obs_ej(1,s);
                                limity_obs_ej(1,index_c8)= limity_obs_ej(1,s);
                                index_c8=index_c8+1;

                        end

                   end
             plot(limity_obs_ej,limitx_obs_ej,'*b')

                %%%%%%%%%%%%%%%%%%%%%%
                index_c8=1;
                   for s=1:(num*4)%dibuja los asteriscos calculados para diagonales

                        if(limitx_obs_dia(1,s)~=0  && limity_obs_dia(1,s)~=0 && limitx_obs_dia(1,s)~=1  && limity_obs_dia(1,s)~=1)

                                limitx_obs_dia(1,index_c8)= limitx_obs_dia(1,s);
                                limity_obs_dia(1,index_c8)= limity_obs_dia(1,s);
                                index_c8=index_c8+1;

                        end

                   end
                plot(limity_obs_dia,limitx_obs_dia,'*b')

%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
%seccion que determina puntos medios entre borde obstaculo
```

```
%y borde de contorno
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<
                %rutina puntos medios entre borde obstaculo y borde de contorno

                index_c5=0;
                index_c6=0;
                cont_ast_e=0;

                condicion=true;
                i=1;
                while (condicion)

    %%%%%%%%%%%%%%%%%%%%%%%%%puntos a los bordes del contorno

                            index_c5=index_c5+1;
                            C_cx5(index_c5)= limity_obs_ej(i);
                            C_cy5(index_c5)= limitx_obs_ej(i) + f;
                            C_cx5(index_c5)=round(C_cx5(index_c5));
                            C_cy5(index_c5)=round(C_cy5(index_c5)/2);

                            if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
                                index_c6=index_c6+1;
                                C_cx6(index_c6)=C_cx5(index_c5);
                                C_cy6(index_c6)=C_cy5(index_c5);
                                plot(C_cx6(index_c6),C_cy6(index_c6),'*w')
                                cont_ast_e=cont_ast_e+1;

                            end
                            i=i+1;
                            index_c5=index_c5+1;
                            C_cx5(index_c5)=limity_obs_ej(i) + c;
                            C_cy5(index_c5)=limitx_obs_ej(i);
                            C_cx5(index_c5)=round(C_cx5(index_c5)/2);
                            C_cy5(index_c5)=round(C_cy5(index_c5));

                            if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
                                index_c6=index_c6+1;
                                C_cx6(index_c6)=C_cx5(index_c5);
                                C_cy6(index_c6)=C_cy5(index_c5);
                                plot(C_cx6(index_c6),C_cy6(index_c6),'*w')
                                cont_ast_e=cont_ast_e+1;

                            end
                            i=i+1;
                            index_c5=index_c5+1;
                            C_cx5(index_c5)=limity_obs_ej(i);
                            C_cy5(index_c5)=limitx_obs_ej(i)+1 ;
                            C_cx5(index_c5)=round(C_cx5(index_c5));
                            C_cy5(index_c5)=round(C_cy5(index_c5)/2);

                            if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
                                index_c6=index_c6+1;
                                C_cx6(index_c6)=C_cx5(index_c5);
                                C_cy6(index_c6)=C_cy5(index_c5);
                                plot(C_cx6(index_c6),C_cy6(index_c6),'*w')
                                cont_ast_e=cont_ast_e+1;

                            end
                            i=i+1;
                            index_c5=index_c5+1;
                            C_cx5(index_c5)=limity_obs_ej(i) +1;
                            C_cy5(index_c5)=limitx_obs_ej(i);
                            C_cx5(index_c5)=round(C_cx5(index_c5)/2);
                            C_cy5(index_c5)=round(C_cy5(index_c5));
```

```
377
378                              if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
379                                  index_c6=index_c6+1;
380                                  C_cx6(index_c6)=C_cx5(index_c5);
381                                  C_cy6(index_c6)=C_cy5(index_c5);
382                                  plot(C_cx6(index_c6),C_cy6(index_c6),'*w')
383                                  cont_ast_e=cont_ast_e+1;
384
385                              end
386
387
388                             if(i==index_c8-1)
389
390                                  condicion=false;
391                             end
392                             i=i+1;
393                 end
394                 i=1;
395
396                 condicion=true;
397                 while (condicion)
398     %%%%%%%%%%%%%%%%%%%puntos diagonales a las esquinas del entorno
399                              index_c5=index_c5+1;
400                              C_cx5(index_c5) = limity_obs_dia(i) + c;
401                              C_cy5(index_c5)= limitx_obs_dia(i) + f;
402                              C_cx5(index_c5)=round(C_cx5(index_c5)/2);
403                              C_cy5(index_c5)=round(C_cy5(index_c5)/2);
404
405                              if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
406                                  index_c6=index_c6+1;
407                                  C_cx6(index_c6)=C_cx5(index_c5);
408                                  C_cy6(index_c6)=C_cy5(index_c5);
409                                  plot(C_cx6(index_c6),C_cy6(index_c6),'*c')
410                                  cont_ast_e=cont_ast_e+1;
411
412                              end
413                              i=i+1;
414                              index_c5=index_c5+1;
415                              C_cx5(index_c5)=limity_obs_dia(i) + 1;
416                              C_cy5(index_c5)=limitx_obs_dia(i) + f;
417                              C_cx5(index_c5)=round(C_cx5(index_c5)/2);
418                              C_cy5(index_c5)=round(C_cy5(index_c5)/2);
419
420                              if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
421                                  index_c6=index_c6+1;
422                                  C_cx6(index_c6)=C_cx5(index_c5);
423                                  C_cy6(index_c6)=C_cy5(index_c5);
424                                  plot(C_cx6(index_c6),C_cy6(index_c6),'*c')
425                                  cont_ast_e=cont_ast_e+1;
426
427                              end
428                              i=i+1;
429                              index_c5=index_c5+1;
430                              C_cx5(index_c5)=limity_obs_dia(i) + 1;
431                              C_cy5(index_c5)=limitx_obs_dia(i) + 1;
432                              C_cx5(index_c5)=round(C_cx5(index_c5)/2);
433                              C_cy5(index_c5)=round(C_cy5(index_c5)/2);
434
435                              if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
436                                  index_c6=index_c6+1;
437                                  C_cx6(index_c6)=C_cx5(index_c5);
438                                  C_cy6(index_c6)=C_cy5(index_c5);
439                                  plot(C_cx6(index_c6),C_cy6(index_c6),'*c')
440                                  cont_ast_e=cont_ast_e+1;
```

```

                                end
                                i=i+1;
                                index_c5=index_c5+1;
                                C_cx5(index_c5)=limity_obs_dia(i) + c;
                                C_cy5(index_c5)=limitx_obs_dia(i) + 1;
                                C_cx5(index_c5)=round(C_cx5(index_c5)/2);
                                C_cy5(index_c5)=round(C_cy5(index_c5)/2);

                                if L(C_cy5(index_c5),C_cx5(index_c5)) == 0
                                    index_c6=index_c6+1;
                                    C_cx6(index_c6)=C_cx5(index_c5);
                                    C_cy6(index_c6)=C_cy5(index_c5);
                                    plot(C_cx6(index_c6),C_cy6(index_c6),'*c')
                                    cont_ast_e=cont_ast_e+1;

                                end


                               if(i==index_c8-1)

                                    condicion=false;
                               end
                               i=i+1;
                    end

%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<
      %%%%%%%%%%%%%%%%%elimina los ceros de la matriz
                C_cx2(C_cx2==0) = [];
                C_cy2(C_cy2==0) = [];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %se reunen en un solo vector todo los puntos posibles para teselacion
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                global C_cx7
                global C_cy7

                C_cx7=[C_cx2, C_cx6];
                C_cy7=[C_cy2, C_cy6];

                figure(3)
                imshow(I)
                hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 %Ubicacion de los puntos ubicados anteriomente con
 %rojo para expansion de teselaciones
 %<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>
                    x=length(C_cx7)+1;
                    tf = round(f/x);% División de filas
                    tc = round(c/x);% División de columnas
                    tes = ones(f,c); % Matriz de unos que corresponderá al entorno de salida
                    %tes = 255.*tes; % Se vuelve blanca la matriz en formato uint8
                    tes = 255.*tes; % Se vuelve blanca la matriz en formato uint8
                    global tesv; % Matriz de teselaciones válidas como variable global
                    tesv = zeros(x,x); % Tamaño de la matriz correspondiente al número de teselaciones
                    count = 0;% Contador de picseles del espacio de configuración
                    coltes = zeros(x-1,x-1); % Matriz para definir la tonalidad
```

```
de cada teselación
                    cont = 1; % Contador para definir el tamaño de cada teselación
                    m = 1; % Codicional para conocer si el entorno se encuentra teselado en su totalidad
                    % <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                    %%%% Ciclo para reconocer el espacio de configuración %%%%%
                    for i = 1 : f % Recorrido por filas del entorno
                        for j = 1 : c % Recorrido por columnas del entorno
                            if I(i,j) == 0 % Compara un picel si se encuentra sobre el espacio de configuración
                                count = count + 1; % count aumenta en uno por cada picsel
                            end
                        end
                    end
                    % <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                    %para puntos entre ejes - contorno y diagonales-esquinas de contorno
                    %%%%%% Ciclo para definir la tonalidad de cada teselación y la separacipon
                    %%%%%% entre cada punto de origen %%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%
                    for i = 1 : x-1% Ciclo para cantidad de teselaciones por filas
                        for j = 1 : x-1 % Ciclo para cantidad de teselaciones por columnas

                            px = C_cx7(j); % Define la posición en X de cada teselación
                            py = C_cy7(j); % Define la posición en Y de cada teselación
                            if (x-1) >= 15 % Define un rango máximo de teselaciones para diferenciar la tonalidad
                                kk = 15; % Limita el número de tonos (entre 1 y 254, 255 y 0 no cuentan)
                            else
                                kk = (x - 1); % En caso contrario la tonalidad depende del número de teselaciones
                            end
                            coltes(i,j) = (255/(((kk)^2)+1))*(((i-1)*(kk))+j); % Define la tonalida dividiendo 255 en la posición de cada teselación
                            while coltes(i,j) > 255 % Condicional paratonalidades superiores a 255
                                coltes(i,j) = coltes(i,j) - 255; % Se resta 255 hasta que sea menor a 255
                            end
                            if coltes(i,j) == 255 % Si alguna teselación es igual a 255
                                coltes(i,j) = 1; % Se le impartirá una tonalidad de uno a esa teselación
                            end

                                plot(px,py,'*r') % Graficará un asterisco rojo sobre el centro de una teselación válida
                                tesv(px,py) = 1; % El punto sobre la matriz de teselaciones será uno, indicando que es válida

                        end
                    end
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>
%Generacion de teselaciones tipo rombo en los punto ubicado
```

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    count1 = round(count*porcentaje);


                   %%%%% Ciclo para la expansión en frente de onda de cada
teselación %%%%%%%
                   while m < count1 % Ciclo para confirmar la teselación del
espacio de configuración
                       for k = 1 : x-1 % Ciclo por columnas de las teselaciones
                           for l = 1 : x-1 % Ciclo por filas de las teselaciones
                               for j = 0 : cont % Contador j para definir el
tamaño de las teselaciones
                                   if tesv(C_cx7(l),C_cy7(k)) == 1 %
Condicional si es una teselación válida
                                       % Primer cuadrante
                                       yc = (C_cx7(l)) + j; % Incremento en Y
igual la posición en Y mas el conteo de j
                                       xc = (C_cy7(l)) + cont - j; % Incremento
en X igual a la posición en X mas el decremento de cont

                                       if yc >= c % Limitación en Y si
sobrepasa el entorno
                                           yc = c; % Valor máximo de Y igual a
la última columna
                                       end
                                       if xc >= f % Limitación en X si
sobrepasa el entorno
                                           xc = f; % Valor máximo de X igual a
la última fila
                                       end
                                       % Condicional para poder expandir el
frente de onda
                                       % Si la imagen original el picsel se
encuentra sobre el espacio de
                                       % configuración y en la imagen de salida
el picsel aum no
                                       % ha sido expandido por alguna teselación
                   %                     if I(xc,yc) == 0 && tes(xc,yc) == 255
                   %                         tes(xc,yc) = coltes(l,k); % Define
la tonalidad del picsel dependiendo de la teselación correspondiente
                   %                         m = m + 1; % Aumenta el contador
de picseles expandidos
                                        if I(xc,yc) == 0 && tes(xc,yc) == 255
                                           tes(xc,yc) = coltes(l,k); % Define
la tonalidad del picsel dependiendo de la teselación correspondiente
                                           m = m + 1; % Aumenta el contador de
picseles expandidos

                                       end
                                       % Segundo cuadrante
                                       yc = (C_cx7(l)) + j; % Incremento en Y
igual la posición en Y mas el conteo de j
                                       xc = (C_cy7(l)) - (cont - j); %
Incremento en X igual a la posición en X mas el incremento de cont

                                       if yc >= c % Limitación en Y si
sobrepasa la altura del entorno
                                           yc = c; % Valor máximo de Y igual a
la última columna
                                       end
                                       if xc <= 1 % Limitación en X si
sobrepasa el entorno
                                           xc = 1; % Valir mínimo de X igual a
```

```
    la primera fila
581                                         end
582                                         % Condicional para poder expandir el
    frente de onda
583                                         % Si la imagen original el picsel se
    encuentra sobre el espacio de
584                                         % configuración y en la imagen de salida
    el picsel aum no
585                                         % ha sido expandido por alguna teselación
586                                         if I(xc,yc) == 0 && tes(xc,yc) == 255
587                                             tes(xc,yc) = coltes(l,k); % Define
    la tonalidad del picsel dependiendo de la teselación correspondiente
588                                             m = m + 1; % Aumenta el contador de
    picseles expandidos
589                                         end
590                                         % Tercer cuadrante
591                                         yc = (C_cx7(l)) - j; % Incremento en Y
    igual la posición en Y menos el conteo de j
592                                         xc = (C_cy7(l)) + cont - j; % Incremento
    en X igual a la posición en X mas el decremento de cont
593
594                                         if yc <= 1 % Limitación en Y si
    sobrepasa el entorno
595                                             yc = 1; % Valor mínimo de Y igual a
    la primera columna
596                                         end
597                                         if xc >= f % Limitación en X si
    sobrepasa el entorno
598                                             xc = f; % Valor máximo de X igual a
    la última fila
599                                         end
600                                         % Condicional para poder expandir el
    frente de onda
601                                         % Si la imagen original el picsel se
    encuentra sobre el espacio de
602                                         % configuración y en la imagen de salida
    el picsel aum no
603                                         % ha sido expandido por alguna teselación
604                                         if I(xc,yc) == 0 && tes(xc,yc) == 255
605                                             tes(xc,yc) = coltes(l,k); % Define
    la tonalidad del picsel dependiendo de la teselación correspondiente
606                                             m = m + 1; % Aumenta el contador de
    picseles expandidos
607                                         end
608                                         % Cuarto cuadrante
609                                         yc = (C_cx7(l)) - j; % Incremento en Y
    igual la posición en Y menos el conteo de j
610                                         xc = (C_cy7(l)) - (cont - j); %
    Incremento en X igual a la posición en X mas el incremento de cont
611
612                                         if yc <= 1 % Limitación en Y si
    sobrepasa el entorno
613                                             yc = 1; % Valor mínimo de Y igual a
    la primera columna
614                                         end
615                                         if xc <= 1 % Limitación en X si
    sobrepasa el entorno
616                                             xc = 1; % Valir mínimo de X igual a
    la primera fila
617                                         end
618                                         % Condicional para poder expandir el
    frente de onda
619                                         % Si la imagen original el picsel se
    encuentra sobre el espacio de
```

```
620                                         % configuración y en la imagen de salida
    el picsel aum no
621                                         % ha sido expandido por alguna teselación
622                                         if I(xc,yc) == 0 && tes(xc,yc) == 255
623                                             tes(xc,yc) = coltes(l,k); % Define
    la tonalidad del picsel dependiendo de la teselación correspondiente
624                                             m = m + 1; % Aumenta el contador de
    picseles expandidos
625                                         end
626                                     end
627                                 end
628                             end
629                         end
630                         cont = cont + 1; % Aumento del tamaño del frente de onda
    para las teselaciones
631                     end
632
633 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
634 %%%%%%%%%%%%%%%%%%% Graficacion de la imagen de salida de las %%%%%%%%%%%%%%%
635 %%%%%%%%%%%%%%%%%%% teselaciones expandidas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
636 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
637                     r = tesv; % Igualar la matriz de teselaciones a la variable
    global
638                     tes = uint8(tes); % Igualar la imagen de salida al formato
    uint8
639
640                     figure(4)
641                     imshow(tes) % Graficar la imagen de salida
642                     imwrite(tes,'J.png') % Guardar la imagen
643                     hold on
644
645 %<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
646 %verificacion de teselaciones con los punto de origen
647 %<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
648                     %para puntos entre ejes - contorno y diagonales-esquinas de
    contorno
649                     %%%%%% Ciclo para definir la tonalidad de cada teselación y
    la separacipon
650                     %%%%%% entre cada punto de origen %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%
651                     for i = 1 : length(C_cx7)% Ciclo para cantidad de
    teselaciones por filas
652                         for j = 1 : length(C_cy7) % Ciclo para cantidad de
    teselaciones por columnas
653
654                             px = C_cx7(j); % Define la posición en X de cada
    teselación
655                             py = C_cy7(j); % Define la posición en Y de cada
    teselación
656                             if (x-1) >= 15 % Define un rango máximo de
    teselaciones para diferenciar la tonalidad
657                                 kk = 15; % Limita el número de tonos (entre 1 y
    254, 255 y 0 no cuentan)
658                             else
659                                 kk = (x - 1); % En caso contrario la tonalidad
    depende del número de teselaciones
660                             end
661                             coltes(i,j) = (255/(((kk)^2)+1))*(((i-1)*(kk))+j);
    % Define la tonalida dividiendo 255 en la posición de cada teselación
662                             while coltes(i,j) > 255 % Condicional
    paratonalidades superiores a 255
663                                 coltes(i,j) = coltes(i,j) - 255; % Se resta 255
    hasta que sea menor a 255
664                             end
```

```
                            if coltes(i,j) == 255 % Si alguna teselación es
igual a 255
                                coltes(i,j) = 1; % Se le impartirá una tonalidad
de uno a esa teselación
                            end

                                plot(px,py,'*r') % Graficará un asterisco rojo
sobre el centro de una teselación válida
                                tesv(px,py) = 1; % El punto sobre la matriz de
teselaciones será uno, indicando que es válida

                        end
                    end


else
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
%EN CASO DE NO TENER OBSTACULOS SE ASIGNAN x NUMERO DE PUNTOS DE TESELACIONES
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
x=3+1;
tf = round(f/x);% División de filas
tc = round(c/x);% División de columnas
tes = ones(f,c); % Matriz de unos que corresponderá al entorno de salida
tes = 255.*tes; % Se vuelve blanca la matriz en formato uint8
global tesv; % Matriz de teselaciones válidas como variable global
tesv = zeros(x,x); % Tamaño de la matriz correspondiente al número de
teselaciones
count = 0;% Contador de picseles del espacio de configuración
coltes = zeros(x-1,x-1); % Matriz para definir la tonalidad de cada teselación
cont = 1; % Contador para definir el tamaño de cada teselación
m = 1; % Codicional para conocer si el entorno se encuentra teselado en su
totalidad
%[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[

C_cx_obs=[260,round(rand*(260 - 1))+1,round(rand*(260 - 1))+1];
C_cy_obs=[round(rand*(260 - 1))+1,260,round(rand*(260 - 1))+1];

%%%% Ciclo para reconocer el espacio de configuración %%%%%
for i = 1 : f % Recorrido por filas del entorno
    for j = 1 : c % Recorrido por columnas del entorno
        if I(i,j) == 0 % Compara un picel si se encuentra sobre el espacio de
configuración
            count = count + 1; % count aumenta en uno por cada picsel
        end
    end
end
%]]]]]]}}}}}}}}}}]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]}}

%para puntos entre ejes - contorno y diagonales-esquinas de contorno
%%%%%% Ciclo para definir la tonalidad de cada teselación y la separacipon
%%%%%% entre cada punto de origen %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1 : x-1% Ciclo para cantidad de teselaciones por filas
    for j = 1 : x-1 % Ciclo para cantidad de teselaciones por columnas
        px = C_cx_obs(j); % Define la posición en X de cada teselación
        py = C_cy_obs(j); % Define la posición en Y de cada teselación
        if (x-1) >= 15 % Define un rango máximo de teselaciones para diferenciar
la tonalidad
            kk = 15; % Limita el número de tonos (entre 1 y 254, 255 y 0 no
cuentan)
        else
            kk = (x - 1); % En caso contrario la tonalidad depende del número de
teselaciones
        end
        coltes(i,j) = (255/((kk)^2)+1))*(((i-1)*(kk))+j);  % Define la tonalida
```

```
dividiendo 255 en la posición de cada teselación
719         while coltes(i,j) > 255 % Condicional paratonalidades superiores a 255
720             coltes(i,j) = coltes(i,j) - 255; % Se resta 255 hasta que sea menor
    a 255
721         end
722         if coltes(i,j) == 255 % Si alguna teselación es igual a 255
723             coltes(i,j) = 1; % Se le impartirá una tonalidad de uno a esa
    teselación
724         end
725             plot(px,py,'*r') % Graficará un asterisco rojo sobre el centro de
    una teselación válida
726             tesv(px,py) = 1; % El punto sobre la matriz de teselaciones será
    uno, indicando que es válida
727
728     end
729 end
730
731  count1 = round(count*porcentaje);
732 %%%%%% Ciclo para la expansión en frente de onda de cada teselación %%%%%%%
733 while m < count1 % Ciclo para confirmar la teselación del espacio de
    configuración
734         for k = 1 : x-1 % Ciclo por columnas de las teselaciones
735             for l = 1 : x-1 % Ciclo por filas de las teselaciones
736                 for j = 0 : cont % Contador j para definir el tamaño de las
    teselaciones
737                     if tesv(C_cx_obs(l),C_cy_obs(l)) == 1 % Condicional si es
    una teselación válida
738                         % Primer cuadrante
739                         yc = (C_cx_obs(l)) + j; % Incremento en Y igual la
    posición en Y mas el conteo de j
740                         xc = (C_cy_obs(l)) + cont - j; % Incremento en X igual a
    la posición en X mas el decremento de cont
741
742                         if yc >= c % Limitación en Y si sobrepasa el entorno
743                             yc = c; % Valor máximo de Y igual a la última columna
744                         end
745                         if xc >= f % Limitación en X si sobrepasa el entorno
746                             xc = f; % Valor máximo de X igual a la última fila
747                         end
748                         % Condicional para poder expandir el frente de onda
749                         % Si la imagen original el picsel se encuentra sobre el
    espacio de
750                         % configuración y en la imagen de salida el picsel aum no
751                         % ha sido expandido por alguna teselación
752                         if I(xc,yc) == 0 && tes(xc,yc) == 255
753                             tes(xc,yc) = coltes(l,k); % Define la tonalidad del
    picsel dependiendo de la teselación correspondiente
754                             m = m + 1; % Aumenta el contador de pixeles
    expandidos
755
756                         end
757                         % Segundo cuadrante
758                         yc = (C_cx_obs(l)) + j; % Incremento en Y igual la
    posición en Y mas el conteo de j
759                         xc = (C_cy_obs(l)) - (cont - j); % Incremento en X igual
    a la posición en X mas el incremento de cont
760
761                         if yc >= c % Limitación en Y si sobrepasa la altura del
    entorno
762                             yc = c; % Valor máximo de Y igual a la última columna
763                         end
764                         if xc <= 1 % Limitación en X si sobrepasa el entorno
765                             xc = 1; % Valir mínimo de X igual a la primera fila
766                         end
```

```
767                         % Condicional para poder expandir el frente de onda
768                         % Si la imagen original el picsel se encuentra sobre el
    espacio de
769                         % configuración y en la imagen de salida el picsel aum no
770                         % ha sido expandido por alguna teselación
771                         if I(xc,yc) == 0 && tes(xc,yc) == 255
772                             tes(xc,yc) = coltes(l,k); % Define la tonalidad del
    picsel dependiendo de la teselación correspondiente
773                             m = m + 1; % Aumenta el contador de picseles
    expandidos
774                         end
775                         % Tercer cuadrante
776                         yc = (C_cx_obs(l)) - j; % Incremento en Y igual la
    posición en Y menos el conteo de j
777                         xc = (C_cy_obs(l)) + cont - j; % Incremento en X igual a
    la posición en X mas el decremento de cont
778
779                         if yc <= 1 % Limitación en Y si sobrepasa el entorno
780                             yc = 1; % Valor mínimo de Y igual a la primera
    columna
781                         end
782                         if xc >= f % Limitación en X si sobrepasa el entorno
783                             xc = f; % Valor máximo de X igual a la última fila
784                         end
785                         % Condicional para poder expandir el frente de onda
786                         % Si la imagen original el picsel se encuentra sobre el
    espacio de
787                         % configuración y en la imagen de salida el picsel aum no
788                         % ha sido expandido por alguna teselación
789                         if I(xc,yc) == 0 && tes(xc,yc) == 255
790                             tes(xc,yc) = coltes(l,k); % Define la tonalidad del
    picsel dependiendo de la teselación correspondiente
791                             m = m + 1; % Aumenta el contador de picseles
    expandidos
792                         end
793                         % Cuarto cuadrante
794                         yc = (C_cx_obs(l)) - j; % Incremento en Y igual la
    posición en Y menos el conteo de j
795                         xc = (C_cy_obs(l)) - (cont - j); % Incremento en X igual
    a la posición en X mas el incremento de cont
796
797                         if yc <= 1 % Limitación en Y si sobrepasa el entorno
798                             yc = 1; % Valor mínimo de Y igual a la primera
    columna
799                         end
800                         if xc <= 1 % Limitación en X si sobrepasa el entorno
801                             xc = 1; % Valir mínimo de X igual a la primera fila
802                         end
803                         % Condicional para poder expandir el frente de onda
804                         % Si la imagen original el picsel se encuentra sobre el
    espacio de
805                         % configuración y en la imagen de salida el picsel aum no
806                         % ha sido expandido por alguna teselación
807                         if I(xc,yc) == 0 && tes(xc,yc) == 255
808                             tes(xc,yc) = coltes(l,k); % Define la tonalidad del
    picsel dependiendo de la teselación correspondiente
809                             m = m + 1; % Aumenta el contador de picseles
    expandidos
810                         end
811                     end
812                 end
813             end
814         end
815         cont = cont + 1; % Aumento del tamaño del frente de onda para las
```

```
teselaciones
end


   disp('Entorno sin obstaculos, se generan 3 teselaciones aleatoria')
   figure(3)
   tes = uint8(tes); % Igualar la imagen de salida al formato uint8
   imshow(tes) % Graficar la imagen de salida
   imwrite(tes,'J.png') % Guardar la imagen
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% CODIGO DE LIMITE DE TESELACIONES %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

I = imread('J.png');
H = imread('I.png');
[L_obs, num] = bwlabel(H);
[f, c] = size(I);
tesx = ones(f,c);
tesx = 255.*tesx;
hold on
%%%%%%%%%%%%%%%%%%Dibuja contorno negros de dos pixeles en los limites de la
%%%%%%%%%%%%%%%%%%teselaciones
for i = 1 : f - 1
    for j = 1 : c - 1
         if (I(i,j) ~= I(i+1,j)) || (I(i,j) ~= I(i,j+1)) || (I(i,j) ~= I(i+1,j+1)) || (I(i,j) == 255)
            tesx(i,j) = 0;
            tesx(i+1,j) = 0;
            tesx(i,j+1) = 0;
            tesx(i+1,j+1) = 0;
        end
    end
end

max = 0;
figure(5)
imshow(tesx)
hold on
L = bwlabel(tesx);

for k = 1 : f
    for l = 1 : c
        if L(k,l) >= max
            max = L(k,l);
        end
    end
end
counter = 0;
z = zeros(f,c);
centx = zeros(1,max);
centy = zeros(1,max);
tesy = zeros(f,c);
for m = 1 : max
    for g = 1 : c
        for h = 1 : f
            for b=1:num
                if L(g,h) == m && L_obs(g,h) ~= b
                    z(g,h) = 255;
                    counter = counter + 1;
                end
            end
```

```
        end
    end
    if counter >= 50

  %**************************************************************
  %Dibuja los centros de masa de la espaciones con teselaciones
  % con color verde
  %**************************************************************
    tq_1 = 0;
    M_1 = 0;
    tq_2 = 0;
    M_2 = 0;
    c1=0;
    o = ones(f,1);
    p = ones(1,c);
    for i = 1:f
       tq_1 = i*dot(o,z(i,:))+tq_1;
       M_1 = dot(o,z(i,:))+M_1;
    end
    Cmx_1 = round(tq_1 / M_1);


    for j = 1:c
       tq_2 = j*dot(p,z(:,j))+tq_2;
       M_2 = dot(p,z(:,j))+M_2;
    end
    Cmx_2 =round(tq_2 / M_2);

    z = zeros(f,c);
    global centx;
    global centy;
    centx(1,m) = Cmx_2;
    centy(1,m) = Cmx_1;
    counter = 1;
    end

    tesy(Cmx_1,Cmx_2) = 255;

end
 plot(centx,centy,'*g')
 %*****************************************************************
tesy = uint8(tesy);
%figure
%imshow(tesy)
%imshow(tesx)

imwrite(tesx,'K.png')%registra delimitación de teselaciones
imwrite(tesy,'M.png')%registra los centroides de las teselaciones

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% FIN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end
```

7.1.3. Busqueda_IG_definitiva_3

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%IMPLEMENTACION Y COMPROBACION DE ALGORITMO DE BUSQUEDA DE RUTA EFICAZ
%Y EFICIENTE CON FRENTES DE ONDA TIPO ROMBO.
%FUNDACION UNIVERSITARIA LOS LIBERTADORES
%Facultad de ingenieria
%Ingenieria Electronica
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%CREADO POR: Ignacio Gutierrez Gomez 14/05/2017
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%*********************************************************
%FUNCION DE GENERACION DE PUNTOS DE REFERENCIA, TESELACION,
%LIMITE DE TESELACIONES Y UBICACION DE PUNTOS DEFINITIVOS PARA
%TRAZADO DE RUTA EFICAZ Y EFICIENTE.
%*********************************************************

function [vect_res_x_def, vect_res_y_def, visu_dist_x, visu_dist_y]=Busqueda_IG_definitiva_3( PosPartX,PosPartY,PosLLegX,PosLLegY,Heurist)
%clear all,
%close all, clc
%decraracion de variables
global PosPartX;
global PosPartY;
global PosLLegX;
global PosLLegY;
global actual_x actual_y fin_x fin_y;
global cop_vect_r_x_d  cop_vect_r_y_d;
global tesel;
global C_cx7
global C_cy7
global centx
global centy
global vect_res_x_def
global vect_res_y_def
global mat_hab_tr
global const_prom

I = imread('I.png');

porcentaje = 100;
porcentaje = porcentaje / 100;
count = 0;% Contador de pixeles del espacio de configuración
m = 1; % Codicional para conocer si el entorno se encuentra teselado en su totalidad
%tamaño de la imagen
[f, c] = size(I);
contador = 0;
[L, num]= bwlabel(I);
cantidad = 0;
grises = 0;

vect_tras_x=zeros(1, length(centx));
vect_tras_y=zeros(1, length(centy));
tesy = zeros(1,length(centy));
tesx = zeros(1,length(centx));

tesx =[centx];
tesy=[centy];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define los puntos inicio y fin para la busqueda de la
%trayectoria o ruta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(PosPartX==0 && PosPartY==0 && PosLLegX==0 && PosLLegY==0 && Heurist==0)
```

```
    actual_y = round(rand*(length(centy)-1))+1;
    fin_y = round(rand*(length(centy)-1))+1;

    actual_x = tesx(1,actual_y);
    actual_y = tesy(1,actual_y);
    fin_x = tesx(1,fin_y);
    fin_y = tesy(1,fin_y);
    Heurist=200;


else
    actual_x = PosPartX;
    actual_y = PosPartY;
    fin_x = PosLLegX;
    fin_y = PosLLegY;
end

figure(6)
imshow(I)
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Seccion de validacion de cordenadas y actualizacion en caso
%de que no cumpla las condiciones
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    while( PosPartX==0 || PosPartY==0 || PosLLegX==0 || PosLLegY==0 || PosPartX>=520 ...
            || PosPartY>=520 || PosLLegX>=520 || PosLLegY>=520 ...
            || PosPartX<1 || PosPartY<1 || PosLLegX<1 || PosLLegY<1 || I(PosPartY,PosPartX) ==255 ...
            || I(PosLLegY,PosLLegX) ==255 || PosPartX==PosLLegX || PosPartY==PosLLegY)

         plot(actual_x,actual_y,'sr')

    plot(fin_x,fin_y,'dg')

    disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    disp('PTOS DE PARTIDA O LLEGADA SE UBICAN EN OBSTACULO, CAMBIAR COORDENADAS')
    disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    xlabel('PTOS DE PARTIDA O LLEGADA SE UBICAN EN OBSTACULO, CAMBIAR COORDENADAS')

    disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    disp('%% DIGITE NUEVAS COORDENADAS PTOS DE PARTIDA Y LLEGADA %%%%%%%%')
    disp('%%%%%LOS VALORES DEBEN SER ENTEROS ENTRE 1 Y 519%%%%%%%%%%%%%')
    disp('%%%%% LOS VALORES  NO DEBEN SER LOS MISMO PARA %%%%%%%%%%%%%%')
    disp('%%%%% PosPartX=PosLLegX Y PosLLegy=PosPartX %%%%%%%%%%%%%%%')
    disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')


    PosPartX=input('Ingrese el valor de PosPartX=: ')
    PosPartY=input('Ingrese el valor de PosPartY=: ')
    PosLLegX=input('Ingrese el valor de PosLLegX=: ')
    PosLLegY=input('Ingrese el valor de PosLLegY=: ')
     actual_x = PosPartX;
    actual_y = PosPartY;
    fin_x = PosLLegX;
    fin_y = PosLLegY;

    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%importante asignacion
```

```
I(actual_x ,actual_y )=0;
I(fin_x ,fin_y )=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
%visualizacion de puntos validos para trazado de ruta
%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
contcond = 0;
plot(centx,centy,'*g')

figure(7)
imshow(I)
hold on

%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
%visualizacion de trazado de recta entre puntos de partida
%y llegada con la cual se determinaran los puntos mas
% cercanos a la recta segun la heuritica
%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

if actual_y ~= fin_y && actual_x~=fin_x && I(actual_y,actual_x) ~=255  && I
(fin_y,fin_x) ~=255


        distanciast = sqrt((actual_x - fin_x)^2 + abs(actual_y - fin_y)^2);
        distanciast=distanciast - 1;

        xx = abs(fin_x - actual_x);
        yy = abs(fin_y - actual_y);

        mem_x = actual_x;
        mem_y = actual_y;

        if xx >= yy
            pend = (fin_y - actual_y) / (fin_x - actual_x);
            cond = 1;
        else
            pend = (fin_x - actual_x) / (fin_y - actual_y);
            cond = 2;
        end

        mov_x = actual_x;
        mov_y = actual_y;


            l=0;
            cont = 1;

            while l == 0
                if cond == 1
                    if fin_x > actual_x && fin_y > actual_y
                        mov_x = mov_x + 1;
                        x = 1;
                        mov_y = actual_y + round(cont*pend);
                    elseif fin_x > actual_x && fin_y < actual_y
                        mov_x = mov_x + 1;
                        mov_y = actual_y + round(cont*pend);
                        x = 2;
                    elseif fin_x < actual_x && fin_y < actual_y
                        mov_x = mov_x - 1;
                        mov_y = actual_y - round(cont*pend);
                        x = 3;
                    elseif fin_x < actual_x && fin_y > actual_y
                        mov_x = mov_x - 1;
```

```
                    mov_y = actual_y - round(cont*pend);
                    x = 4;
                elseif fin_y == actual_y && fin_x > actual_x
                    mov_x = mov_x + 1;
                    mov_y = actual_y;
                    x = 5;
                elseif fin_y == actual_y && fin_x < actual_x
                    mov_x = mov_x - 1;
                    mov_y = actual_y;
                    x = 6;
                end
            elseif cond == 2
                if fin_x > actual_x && fin_y > actual_y
                    mov_y = mov_y + 1;
                    x = 7;
                    mov_x = actual_x + round(cont*pend);
                elseif fin_x > actual_x && fin_y < actual_y
                    mov_y = mov_y - 1;
                    mov_x = actual_x - round(cont*pend);
                    x = 8;
                elseif fin_x < actual_x && fin_y < actual_y
                    mov_y = mov_y - 1;
                    mov_x = actual_x - round(cont*pend);
                    x = 9;
                elseif fin_x < actual_x && fin_y > actual_y
                    mov_y = mov_y + 1;
                    mov_x = actual_x + round(cont*pend);
                    x = 10;
                elseif fin_x == actual_x && fin_y > actual_y
                    mov_y = mov_y + 1;
                    mov_x = actual_x;
                    x = 11;
                elseif fin_x == actual_x && fin_y < actual_y
                    mov_y = mov_y - 1;
                    mov_x = actual_x;
                    x = 12;
                end
            end

            if I(mov_x,mov_y) == 0
                grises = grises + 1;
            end

            if mov_x == fin_x && mov_y == fin_y
                l=1;
                 grises
                 if grises > 30

                     actual_x = mem_x;
                     actual_y = mem_y;
                     grises = 0;
                 else
                     actual_x = fin_x;
                     actual_y = fin_y;
                     grises = 0;
                 end
            end
            plot(mov_x,mov_y,'.g')
            vect_tras_x(cont)=mov_x;
            vect_tras_y(cont)=mov_y;

            cont = cont + 1;
        end
```

```
end

%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

        plot(actual_x,actual_y,'sr')

        plot(fin_x,fin_y,'dc')

        plot(centx,centy,'*g')
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
 %*******************************************************************
 %%%%%%%%%%%%%%%%CODIGO DE RUTA EFICIENTE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %___________________________________________________________________
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%declaracion de variables y vectores necesarios
            vect_res_x=zeros(1,length(centx));
            vect_res_y=zeros(1,length(centy));
            vect_res_x_2=zeros(1,length(centx));
            vect_res_y_2=zeros(1,length(centy));
            vect_res_x_def=zeros(1,length(vect_tras_y));
            vect_res_y_def=zeros(1,length(vect_tras_y));
            vect_res_x_def_2=zeros(1,length(vect_tras_x));
            vect_res_y_def_2=zeros(1,length(vect_tras_y));
            indic_1=0;
            indic_2=0;

%###############################################################
%SECCION IMPORTANTE PARA DETERMINAR LOS PUNTOS CERCANOS A LA
%RECTA TRAZADA EN LA SECCION ANTERIOR
%###############################################################
       i=1;
       i2=1;
       i3=1;
       condicion=true;
        while  (condicion)
            %resta cada punto de la recta con las coordenadas de
            %los puntos disponibles y encuentra los valores cercanos
            %a cero o minimos.
           vect_x= abs((centx - vect_tras_x(i)*ones(1,length(centx))));
           vect_y= abs((centy - vect_tras_y(i)*ones(1,length(centy))));

             [val_min_x,indic_1]=min(vect_x);
             [val_min_y,indic_2]=min(vect_y);


            val_x=indic_1;
            val_y=indic_2;

                    if(i2==1 ||  i2==length(vect_tras_x))
                        vect_res_x(i2)=vect_tras_x(i2);
                        vect_res_y(i2)=vect_tras_y(i2);
                    end
       %seccion donde opera la heuristica y selecciona
       % los puntos para filas y luego para columnas
            aux_min_y=abs(centy(val_x)- vect_tras_y(i));
                if (aux_min_y < Heurist)

                    i2=i2+1;
                     vect_res_x(i2)=centx(val_x);
                     vect_res_y(i2)=centy(val_x);

```

```
                end

                 if(i3==1 || i3==length(vect_tras_y))
                        vect_res_y_2(i3)=vect_tras_y(i3);
                        vect_res_x_2(i3)=vect_tras_x(i3);
                end

            aux_min_x=abs(centx(val_y)- vect_tras_x(i));
                if (aux_min_x < Heurist)


                    i3=i3+1;
                    vect_res_y_2(i3)=centy(val_y);
                    vect_res_x_2(i3)=centx(val_y);

                end



            i=i+1;
            if((i>length(vect_tras_x)|| i>length(vect_tras_y)))

                    condicion=false;
            end

        end
 %#############################################################

   %elimina valores cero en los vectores ya que son coordenadas
        vect_res_x_2(vect_res_x_2==0) = [];
        vect_res_y_2(vect_res_y_2==0) = [];
        vect_res_x(vect_res_x==0) = [];
        vect_res_y(vect_res_y==0) = [];
        vect_tras_x(vect_tras_x==0) = [];
        vect_tras_y(vect_tras_y==0) = [];
    %%%%%%%visualiza nuevamente los puntos posibles y la recta%%%%
       figure(8)
        imshow(I)
        hold on
        plot(actual_x,actual_y,'+r')
        %hold on
        plot(fin_x,fin_y,'*c')
         plot(vect_tras_x,vect_tras_y,'-y')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %%%%%%%visualiza nuevamente los puntos posiblesn %%%%
 %%%%%%%%y los seleccionados por la heuristica%%%%%%%
 %%%%%%%%%%%%%%en color rojo y azul%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     plot(centx,centy,'*g')

     plot(vect_res_x,vect_res_y,'*b')

     plot(vect_res_x_2,vect_res_y_2,'*r')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        figure(9)
        imshow(I)
        hold on
        plot(actual_x,actual_y,'sr')

        plot(fin_x,fin_y,'dc')
```

```
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<
%Seccion importante de selecion de puntos diferentes o no
%comunes entre filas y columnas cercanosa la recta dibujada
%anteriormente, y se asignan a un nuevo vector.
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<
  a=1;
  i=1;
condicion=true;
while condicion

    if ((vect_res_x(i)~=vect_res_x_2(i) && vect_res_y(i)~=vect_res_y_2(i) || ...
            (vect_res_x(i)==vect_res_x_2(i) && vect_res_y(i)~=vect_res_y_2
(i)|| ...
            vect_res_x(i)~=vect_res_x_2(i) && vect_res_y(i)==vect_res_y_2(i))))


         vect_res_y_def(a)=vect_res_x(i);
         vect_res_x_def(a)=vect_res_y(i);

        a=a+1;


    end


                if(i==length(vect_res_x) || i==length(vect_res_x_2) )
                    condicion=false;
                end
               i=i+1;

end
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%eliminacion de ceros
vect_res_x_def(vect_res_x_def==0) = [];
vect_res_y_def(vect_res_y_def==0) = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Seccion para ordenar vectores%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [VALx,Ind_ordx] = sort(vect_res_x_def);
    vect_res_x_def = VALx;
    vect_res_y_def = vect_res_y_def(Ind_ordx);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%eliminacion de ceros
vect_res_x_def(vect_res_x_def==0) = [];
vect_res_y_def(vect_res_y_def==0) = [];

%*********************************************
%visulaizacion de puntos selecionados bajo la
%heuritica y demas restriciones anteriores
%***********************************************

plot(vect_res_y_def,vect_res_x_def,'+m')

%**********************************************

disp('%%%%%seccion de trazado de ruta entre los puntos seleccionados %%%%')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%seccion de trazado de ruta entre los puntos seleccionados
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
438
439  %composicion de vector copia del vector principal con las coordendas
440  %de partida y llegada
441  cop_vect_r_x_d=[PosPartX, vect_res_y_def, PosLLegX ];%almacenamiento de vectores
     seleccionados
442  cop_vect_r_y_d=[PosPartY, vect_res_x_def, PosLLegY ];%copia para modificacion
     durante la funcion
443  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
444
445  %*********************************************
446  %visulaizacion de puntos selecionados bajo la
447  %heuritica y demas restriciones anteriores en
448  %otra ventana
449  %*********************************************
450  figure(10)
451      imshow(I)
452      hold on
453       plot(actual_x,actual_y,'sr')
454
455      plot(fin_x,fin_y,'sc')
456      plot(vect_res_y_def,vect_res_x_def,'+m')
457
458  aux_x=PosPartX;%almacenamiento de punto de partida elegida
459  aux_y=PosPartY;
460  fin_x=PosLLegX;%almacenamiento de punto de llegada elegida
461  fin_y=PosLLegY;
462
463  aux_x_2=0;
464  aux_y_2=0;
465
466  %############################################################
467  %Seccion importante de trazado de trayectorias entre los
468  %puntos seleccionados que comuniquen los punto de partida
469  % y de llegada
470  %############################################################
471  %declaracion de vectores necesaros
472  vect_dis=zeros(1, length(cop_vect_r_x_d));
473  vect_dis_prom=zeros(1, length(cop_vect_r_x_d));
474  visu_dist_x=zeros(1, length(cop_vect_r_x_d));
475  visu_dist_y=zeros(1, length(cop_vect_r_x_d));
476
477  %*****************************************************************
478  %Declaracion de matriz de validacion de trazos que no atravizan
479  % obstaculos
480  %*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
481  mat_hab_tr=zeros(length(cop_vect_r_x_d), length(cop_vect_r_y_d));
482
483  %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
484
485   b=1;
486   cont=0;
487   grises = 0;
488
489   j=1;
490
491   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
492   %seccion para evaluar el promedio de distancia
493   %{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{}}
494   disp('%%%%%%%%%%% evaluar distancia para promedio%%%%%%%%%%%%%')
495   b=1;
496    for i=1: length(cop_vect_r_x_d)
497          for j=1:length(cop_vect_r_x_d)
498
499              if(i<j)
```

```
500
501             dist_trazad = sqrt(abs((cop_vect_r_x_d(i) - cop_vect_r_x_d(j)))^2 +
    abs((cop_vect_r_y_d(i) - cop_vect_r_y_d(j)))^2);
502
503             vect_dis_prom(b) = dist_trazad;
504
505              b=b+1;
506
507           end
508          end
509    end
510    %{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{}}
511  %%%%%%%%%%%%%%%%%%%%%%%%valor promedio%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
512
513  prom_dist=(const_prom*mean(vect_dis_prom))
514
515  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
516
517  %******************************************
518  %visulaizacion de puntos selecionados bajo la
519  %heuritica y demas restriciones anteriores en
520  %otra ventana
521  %******************************************
522
523   figure(11)
524      imshow(I)
525      hold on
526       plot(vect_res_y_def,vect_res_x_def,'+m')
527      plot(actual_x,actual_y,'sg')
528          %hold on
529      plot(fin_x,fin_y,'sb')
530
531   %############################################################
532  %Inicio de seccion importante de trazado de trayectorias entre los
533  %puntos seleccionados que comuniquen los punto de partida
534  % y de llegada
535  %############################################################
536
537   disp('%%%%%%%%%%% funcion para trazado de ruta %%%%%%%%%%%%')
538   b=1;
539
540  for i=1: length(cop_vect_r_x_d)
541      for j=1:length(cop_vect_r_x_d)
542
543      if(i<j)
544  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
545  %Calcula distancia parcial entre las coordenadas seleccionadas
546  %por i y j
547
548             dist_trazad = sqrt(abs((cop_vect_r_x_d(i) - cop_vect_r_x_d(j)))^2 +
    abs((cop_vect_r_y_d(i) - cop_vect_r_y_d(j)))^2);
549
550             vect_dis(b) = dist_trazad;
551
552  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
553  %condicional para distancias de trayectoria dibujadas menores al
554  %promedio
555
556  if(dist_trazad < prom_dist)
557
558          actual2_x=cop_vect_r_x_d(i);
559          actual2_y=cop_vect_r_y_d(i);
560
561          fin2_x= cop_vect_r_x_d(j);
```

```
562          fin2_y= cop_vect_r_y_d(j);
563
564   %%%%%%%%%condicional para permitir el trazado siempre que
565   %%%%%%%%%las coordenadas nos sean iguales o se halla terminado
566   %%%%%%%%%%el recorrido
567
568      if(actual2_y ~= fin2_y && actual2_x~=fin2_x && I(actual2_x,actual2_y) ~=255
      && I(fin2_x,fin2_y) ~=255)
569
570
571          xx = abs(fin2_x - actual2_x);
572          yy = abs(fin2_y - actual2_y);
573
574          mem_x = actual2_x;
575          mem_y = actual2_y;
576
577          if xx >= yy
578              pend = (fin2_y - actual2_y) / (fin2_x - actual2_x);
579              cond = 1;
580          else
581              pend = (fin2_x - actual2_x) / (fin2_y - actual2_y);
582              cond = 2;
583          end
584
585          mov_x = actual2_x;
586          mov_y = actual2_y;
587
588              l=0;
589              cont = 1;
590
591              while l == 0
592                  if cond == 1
593                      if fin2_x > actual2_x && fin2_y > actual2_y
594                          mov_x = mov_x + 1;
595                          x = 1;
596                          mov_y = actual2_y + round(cont*pend);
597                      elseif fin2_x > actual2_x && fin2_y < actual2_y
598                          mov_x = mov_x + 1;
599                          mov_y = actual2_y + round(cont*pend);
600                          x = 2;
601                      elseif fin2_x < actual2_x && fin2_y < actual2_y
602                          mov_x = mov_x - 1;
603                          mov_y = actual2_y - round(cont*pend);
604                          x = 3;
605                      elseif fin2_x < actual2_x && fin2_y > actual2_y
606                          mov_x = mov_x - 1;
607                          mov_y = actual2_y - round(cont*pend);
608                          x = 4;
609                      elseif fin2_y == actual2_y && fin2_x > actual2_x
610                          mov_x = mov_x + 1;
611                          mov_y = actual2_y;
612                          x = 5;
613                      elseif fin2_y == actual2_y && fin2_x < actual2_x
614                          mov_x = mov_x - 1;
615                          mov_y = actual2_y;
616                          x = 6;
617                      end
618                  elseif cond == 2
619                      if fin2_x > actual2_x && fin2_y > actual2_y
620                          mov_y = mov_y + 1;
621                          x = 7;
622                          mov_x = actual2_x + round(cont*pend);
623                      elseif fin2_x > actual2_x && fin2_y < actual2_y
624                          mov_y = mov_y - 1;
```

```
                        mov_x = actual2_x - round(cont*pend);
                        x = 8;
                    elseif fin2_x < actual2_x && fin2_y < actual2_y
                        mov_y = mov_y - 1;
                        mov_x = actual2_x - round(cont*pend);
                        x = 9;
                    elseif fin2_x < actual2_x && fin2_y > actual2_y
                        mov_y = mov_y + 1;
                        mov_x = actual2_x + round(cont*pend);
                        x = 10;
                    elseif fin2_x == actual2_x && fin2_y > actual2_y
                        mov_y = mov_y + 1;
                        mov_x = actual2_x;
                        x = 11;
                    elseif fin2_x == actual2_x && fin2_y < actual2_y
                        mov_y = mov_y - 1;
                        mov_x = actual2_x;
                        x = 12;
                    end
                end

                if I(mov_y,mov_x) == 255
                    grises = grises + 1;
                end

%*****************************************************
%condicional para asignar un cero a esa trayecotria
%si atravieza un obstaculo quiere decir
%grises mayor que cero
%+++++++++++++++++++++++++++++++++++++++++++++++++++++
                if mov_x == fin2_x && mov_y == fin2_y
                    l=1;
                     grises;
                     if grises > 0

                         actual2_x = mem_x;
                         actual2_y = mem_y;
                         grises = 0;
          %%%%%%%%%%%%%%%%%%%matriz (guardar 0)
                         mat_hab_tr(i,j)=0;

                     else
%*****************************************************
%condicional para asignar un uno a esa trayecotria
%si NOOO atravieza un obstaculo
%+++++++++++++++++++++++++++++++++++++++++++++++++++++
                         plot([cop_vect_r_x_d(i) cop_vect_r_x_d(j)],
[cop_vect_r_y_d(i) cop_vect_r_y_d(j)],'r')
          %%%%%%%%%%%%%%%%%%%matriz(guardar 1)
                         actual2_x = fin2_x;
                         actual2_y = fin2_y;
                         grises = 0;
                         mat_hab_tr(i,j)=1;

                     end
                end

                cont = cont + 1;
            end

            b=b+1;

    end
    end
```

```matlab

    end
end
end

disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('%%%%%%%%%%%Fin de trazado de rutas%%%%%%%%%%%%%%%%')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% convension de marcadores de grafica
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Color: r (Red), g(Green), b(Blue), c(Cyan), m
% (Magenta), y (Yellow), k (blacK), w (White)


%  Marcadores: +, o, *, ., x, s (square), d
% (diamante), ^ v > o < (triángulos en distinto
% sentido), p (estrella 5 puntas, pentagram), h
% (estrella 6 puntas, hexagram)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% FIN  DE FUNCION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```