DISEÑO DE UN ALGORTIMO PARA LA GENERACION Y NAVEGACION POR RUTAS MEDIANTE UNA INTERFAZ PARA ROBOTS MOVILES

CESAR JULIAN BERNAL GOMEZ

FUNDACION UNIVERSITARIA LOS LIBERTADORES

FACULTAD DE INGENIERIA

INGENIERIA ELECTRONICA

BOGOTA D.C

2017

DISEÑO DE UN ALGORTIMO PARA LA GENERACION Y NAVEGACION POR RUTAS MEDIANTE UNA INTERFAZ PARA ROBOTS MOVILES

CESAR JULIAN BERNAL GOMEZ

Trabajo de grado para optar el título de Ingeniero Electrónico

Director

Iván Darío Ladino

Ingeniero Electrónico

FUNDACION UNIVERSITARIA LOS LIBERTADORES

FACULTAD DE INGENIERIA

INGENIERIA ELECTRONICA

BOGOTA D.C

2017

Nota De Aceptación

_____

_____

_____

_____

_____

Firma:

Nombre:

Presidente Del Jurado

_____

Firma:

Nombre:

Jurado

_____

Firma:

Nombre:

Jurado

Bogotá DC, 31 Julio de 2017

A Ricardo Bernal B, María E Gómez, Tatiana Ríos y demás familiares que me apoyaron y aportaron esfuerzos de alguna manera.

AGRADECIMIENTOS

Muchas gracias a Iván Darío Ladino y demás compañeros de la institución que soportaron y apoyaron el desarrollo de este trabajo.

INDICE

Pág.

INDICE DE FIGURAS **Pág.**

INDICE DE TABLAS **Pág.**

RESUMEN

En este documento se describe la forma utilizada para encontrar el método de búsqueda de una ruta eficiente y evasiva de obstáculos para el desplazamiento de un robot. Mediante la generación de una GUIDE que nos de la opción de manipular nuestro entorno predeterminado, así como la forma y la cantidad de obstáculos con las que podremos realizar la generación de rutas dependiendo el tipo de Teselacion escogida. Dependiendo el punto de inicio y final escogidos nuestro algoritmo propuesto realizara una búsqueda de la ruta más conveniente.

La ruta escogida y el tiempo de respuesta de inicio y final tendrán una variación dependiendo de la Teselacion escogida de las cuatro propuestas. Nuestro sistema tiene una excelente respuesta con el menor tiempo posible de búsqueda y navegación gracias a los siguientes métodos y algoritmos usados que se explicaran en el siguiente capítulo.

INTRODUCCION

Se define navegación como la metodología (o arte) que permite guiar el curso de un robot móvil a través de un entorno con obstáculos. Existen diversos esquemas, pero todos ellos poseen en común el afán por llevar el vehículo a su destino de forma segura. La capacidad de reacción ante situaciones inesperadas debe ser la principal cualidad para desenvolverse, de modo eficaz, en entornos no estructurados.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos, para alcanzar el punto destino seleccionado; y el guiado del vehículo a través de la referencia construida. De forma simultánea, el vehículo puede interaccionar con ciertos elementos del entorno. Así, se define el concepto de operación como la programación de las herramientas de a bordo que le permiten realizar la tarea especificada.

Este capítulo introduce los conceptos fundamentales que definen la problemática de la realización de trabajos por parte de los robots móviles. Así, en primer lugar, se precisan los conceptos de misión, navegación y operación. A continuación, se plantean esquemas de navegación utilizados por robots móviles para realizar una tarea, identificando sus principales componentes. En los siguientes apartados se desarrollan dos de estos componentes: el planificador y el generador, los cuales constituyen el motivo de las aportaciones de esta tesis en los capítulos siguientes. Se procede a una revisión de ambos, dentro de los distintos tipos de metodologías, para con posterioridad realizar una formalización que los defina de manera precisa. Como ampliación de la labor planificación espacial que ejercen los componentes de planificación y generación, se introduce el concepto de trayectoria. La utilización de dicho concepto posibilita una planificación temporal de la tarea del robot móvil, con lo que se consigue un mayor rendimiento del comportamiento del mismo. Por último, se destacan los aspectos más relevantes de este capítulo en las conclusiones

OBJETIVOS

OBJETIVO GENERAL

Desarrollar un algoritmo de búsqueda, navegación y operación de la ruta más eficaz con una evasión de obstáculos.

OBJETIVOS ESPECIFICOS

1. Generar un algoritmo para la representación del mapa.
2. Crear una GUIDE para poder tener un control total del entorno y la creación de obstáculos.
3. Trazar la ruta más eficaz de un punto a otro sin chocar con los obstáculos.

1. MARCO REFERENCIAL

1.1. CONCEPTO DE MISIÓN, NAVEGACIÓN Y OPERACIÓN

El robot móvil se caracteriza por realizar una serie de desplazamientos (navegación) y por llevar a cabo una interacción con distintos elementos de su entorno de trabajo (operación), que implican el cumplimiento de una serie de objetivos impuestos según cierta especificación. Así, formalmente el concepto de misión en el ámbito de los robots móviles se define como la realización conjunta de una serie de objetivos de navegación y operación. En consecuencia, con las definiciones del párrafo anterior, el robot móvil debe poseer una arquitectura que coordine los distintos elementos de a bordo (sistema sensorial, control de movimiento y operación) de forma correcta y eficaz para la realización de una misión. El diseño de esta arquitectura depende mucho de su aplicación en particular, pero un esquema básico de los principales módulos que la componen y la interacción que existe entre los mismos es el presentado en la siguiente figura.

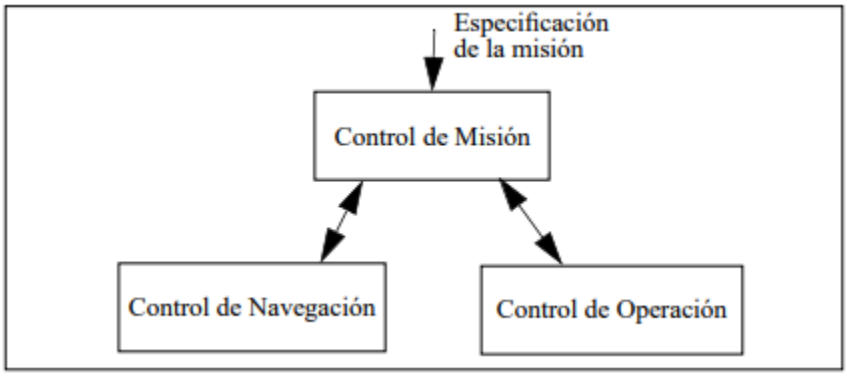


Figura 1.1. Esquema básico de la arquitectura necesaria en un robot móvil para realizar una misión.

En la mencionada figura, se presenta un módulo de control de misión dedicado a coordinar al controlador de desplazamientos (Control de navegación) con el controlador del elemento que interacciona con el entorno de trabajo (control de operación). Esta coordinación debe efectuarse de forma perfecta para cumplir los objetivos impuestos por la misión, definida de acuerdo con ciertas especificaciones de entrada. Formalmente, el control de misión debe analizar el problema y encontrar una estrategia para resolverlo, de suerte que el resultado de este análisis será un plan de navegación y otro de operación, los cuales se entregan a los módulos correspondientes de la parte inferior de la figura 1.

1.2. DIAGRAMA DE VORONOI

El diagrama de Voronoi es llamado así debido a su creador el matemático Voronoi en 1908. Básicamente este método puede verse con la partición de una región determinada en base a unos conjuntos de puntos, su visualización en su forma simple se ve en la Figura 1.2, donde se ha aplicado para un número de puntos aleatorios.

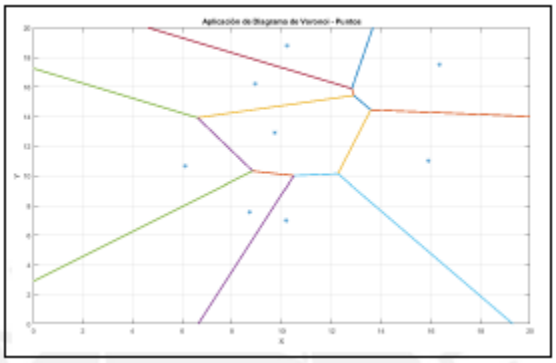


Figura 1.2. Diagrama de Voronoi para puntos aleatorios.

El diagrama de Voronoi se basa en un conjunto finito de puntos contenidos en un espacio de trabajo que definimos, de tal manera que la distancia del diagrama hacia los puntos es la mínima, respecto de otro punto.

$$d(x,p) = \inf \{ d(x,p) | \, p \in \mathcal{P} \}$$

La idea del diagrama de Voronoi es expandida hacia obstáculos poligonales: Cada obstáculo tiene un subespacio perteneciente de todos los puntos que están más cerca de este obstáculo. Una conectividad gráfica es creada, donde los ejes son los bordes de estos subespacios. Los vértices son la intercepción de dos o más bordes. Una ruta a través del gráfico es elegida, como se puede ver en la Figura 1.3.

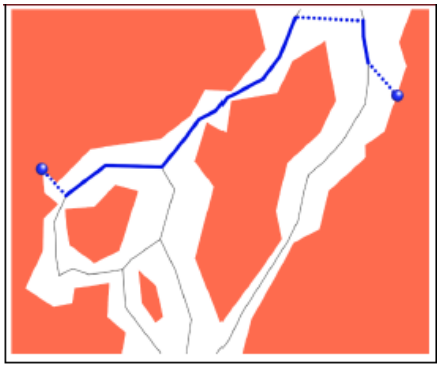


Figura 1.3. Diagrama de Voronoi aplicado.

1.3. ALGORTIMO DIJKSTRA

Este algoritmo es muy conocido por sus diversas aplicaciones en el campo de la busca de la ruta más corta en los métodos gráficos. Este algoritmo fue desarrollado en 1959. Este método asigna a cada nodo en la gráfica un valor de distancia, inicialmente es seteada cada distancia a infinito y el nodo de inicio es seteado a cero. El algoritmo considera todos los nodos excepto el inicial en una condición de no-vistado, y configura el nodo de inicio como el nodo actual. El algoritmo consiste en calcular la distancia tentativa, entre el nodo inicial y demás nodos no-visitados de la vecindad, y compararlo con el valor previo y actualizar si el valor calculado es menor y el nodo que genere este valor menor debe ser tomado como el nuevo nodo actual. Esto se va desarrollando hasta que el nodo de destino (Punto final) es visitado o no existe ninguna conexión entre los nodos visitados y no-visitados.

En la Figura 1.4, vemos que tenemos un punto inicial a y un punto objetivo b, que esta unidos a través de líneas las cuales simbolizar el peso o distancia entre cada punto de ruta, el algoritmo de Djikstra inicia una búsqueda de la menor ruta según el procedimiento descrito líneas arriba, al iniciar tiene 3 opciones las cuales se prueban y continúan hasta obtener la mejor ruta, en este caso vendría dada por los nodos 1-3-6-5.

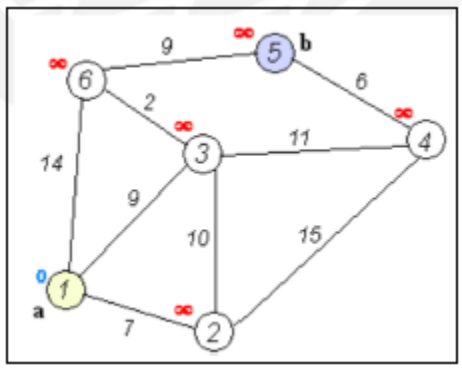


Figura 1.4 Algoritmo de Djikstra.

Una vez obtenidos los puntos, gracias al diagrama de Voronoi, se procede a utilizar el algoritmo de Djikstra para poder obtener los puntos que hagan menor el recorrido desde un punto hacia otro evitando los obstáculos.

1.4. ALGORITMO A*

Como todo algoritmo de búsqueda en amplitud, A* es un algoritmo completo: en caso de existir una solución, siempre dará con ella.

Si para todo nodo n del grafo se **g(n)=0** nos encontramos ante una búsqueda voraz. Si para todo nodo n del grafo se cumple **h(n)=0**, A* pasa a ser una búsqueda de coste uniforme no informada.

Para garantizar la optimización del algoritmo, la función **h(n)** debe ser heurística admisible, esto es, que no sobrestime el coste real de alcanzar el nodo objetivo.

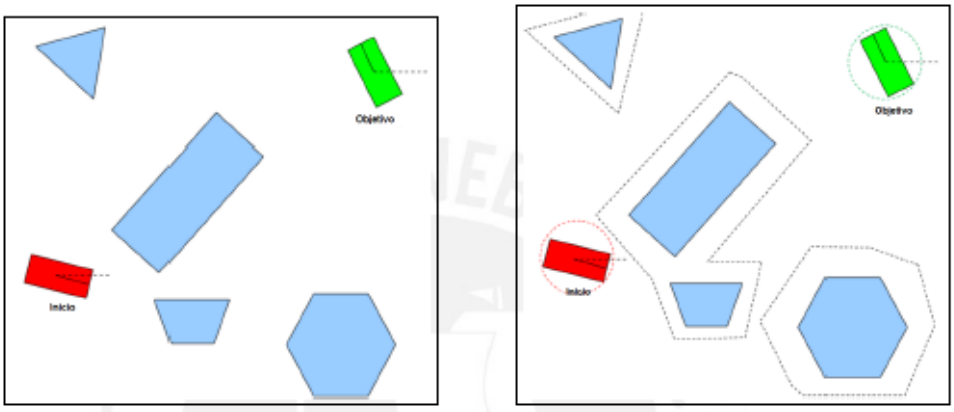
De no cumplirse dicha condición, el algoritmo pasa a denominarse simplemente A, y a pesar de seguir siendo completo, no se asegura que el resultado obtenido sea el camino de coste mínimo. Asimismo, si garantizamos que **h(n)** es consistente (o monótona), es decir, que para cualquier nodo n y cualquiera de sus sucesores, el coste estimado de alcanzar el objetivo desde **n** no es mayor que el de alcanzar el sucesor más el coste de alcanzar el objetivo desde el sucesor.

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

Figura 1.5. Ejemplo de aplicación del algoritmo A*.

1.5. OBSTACULOS

La representación de obstáculos en este trabajo será hecha por polígonos en dos dimensiones, siendo fácilmente extensible esto hacia obstáculos 3D. Naturalmente lo que se busca es evitar los obstáculos para llegar del punto inicial hacia el punto Final. Una de las ventajas de trabajar con polígonos es que son fáciles de incrementar en tamaño y se puede fácilmente asignar una distancia de tolerancia. (Ver Figura 1.6b).



(a) Sin considerar Tolerancia (b) Considerando Tolerancia

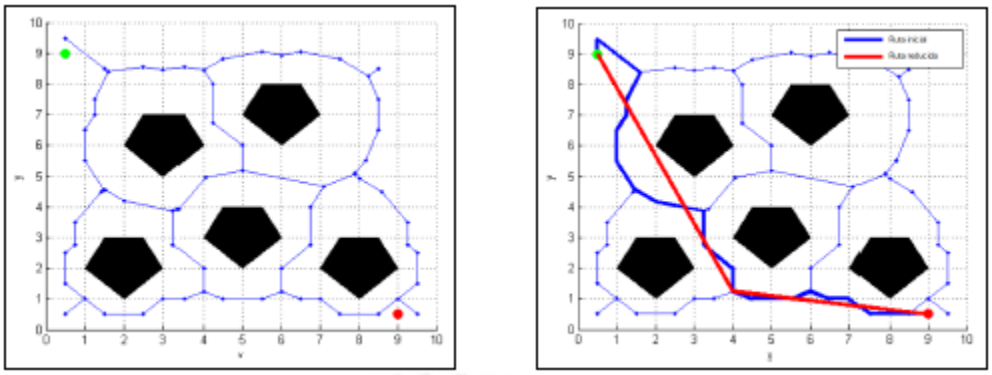
Figura 1.6. Representación de obstáculos gráficamente y consideración de tolerancia

Al tener los obstáculos como figuras geométricas se pueden aprovechar algunas funciones del software de simulación MATLAB. Para nuestro caso hemos usado las funciones polyxpoly, la cual

comprueba si existen intersecciones entre polígonos y la función inpolygon , que determina que valores de los puntos están dentro y fuera o sobre un eje de una región poligonal.

1.6. ELIMINACION DE PUNTOS NO NECESARIOS

Luego de tener la ruta con menor recorrido por la aplicación del diagrama Voronoi y el algoritmo de Djikstra, a se procede a realizar la implementación de un algoritmo que consiste simplemente en remover los puntos que causan cambio de ruta innecesaria y que esta no implique la colisión, también se toma en cuenta un valor de tolerancia de distancia respecto a los obstáculos. En la Figura 1.7a, se muestra un entorno al cual se le aplicó el diagrama de Voronoi, teniendo como partida el punto verde y como llegada el punto rojo, a partir de este diagrama se aplicará el algoritmo de Djikstra y se obtendrá la mejor ruta considerando los puntos dados por el diagrama de Voronoi, luego se aplicará el algoritmo de eliminación de puntos innecesarios, para obtener una mejor ruta y evitar los cambios de orientación.



(a) Diagrama de Voronoi con obstáculos (b) Ruta inicial y ruta reducida libre de obstáculos

Figura 1.7. Eliminación de puntos no necesarios.

Finalmente, como resultado del planeamiento del movimiento tenemos un conjunto de puntos de ruta que nos garantizan una trayectoria libre de obstáculos y con el menor cambio orientación.

1.7. TESELACIONES

Se llama Teselacion, de este modo, al patrón que se sigue al recubrir una superficie. La Teselacion requiere evitar la superposición de figuras y asegurar que no se registran espacios en blanco en el recubrimiento.

Para el desarrollo de la Teselacion, lo habitual es que se hagan reproducciones de una o más teselas hasta recubrir la totalidad de la superficie. Es importante destacar que se pueden realizar teselaciones irregulares, semirregulares o regulares.

1.7.1. Teselados Irregulares

Los teselados irregulares están construidos a partir de polígonos regulares e irregulares que, al igual que todas las teselaciones, cubren toda la superficie sin sobreponerse ni dejar espacios vacíos.

La distribución de los polígonos en los distintos vértices es cíclica; pueden darse 3, 4, 5 y más distribuciones que harán que la periodicidad sea más espaciada, requiriendo dibujar una gran porción de la tesela para poder ver un ciclo completo. Para ver tal efecto, observa dos ejemplos de la distribución del pentágono:

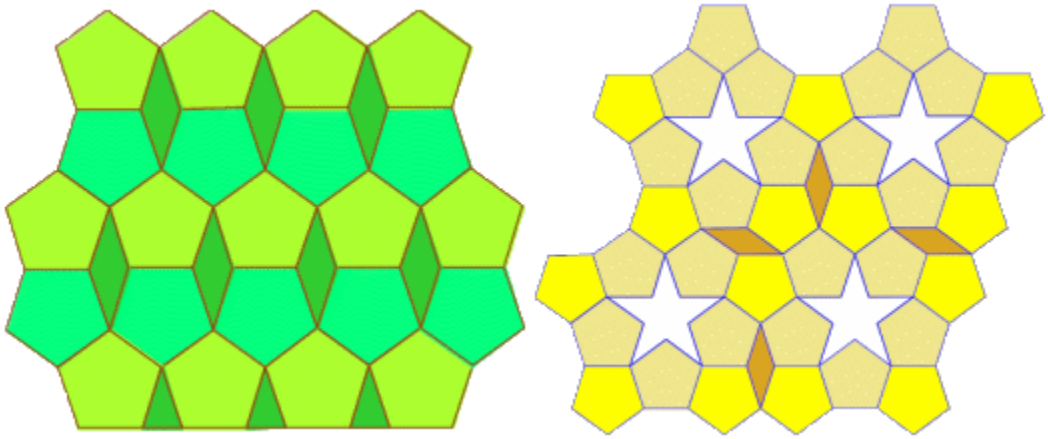


Figura 1.8. Teselados Irregulares

1.7.2. Teselados Regulares

Según Branko Grünbaum y Shephard (sección 1.3), se dice que un teselado es regular si el grupo de simetría del teselado opera transitivamente sobre los elementos del teselado, donde un elemento consiste de un vértice mutuamente incidente, una arista y una tesela. Esto significa que por cada par de elementos hay una operación de simetría que los asocia entre sí.

Esto es equivalente a un teselado arista con arista de polígonos regulares congruentes. Debe haber seis triángulos, cuatro cuadrados o tres hexágonos regulares en cada vértice, produciendo las tres teselaciones regulares.

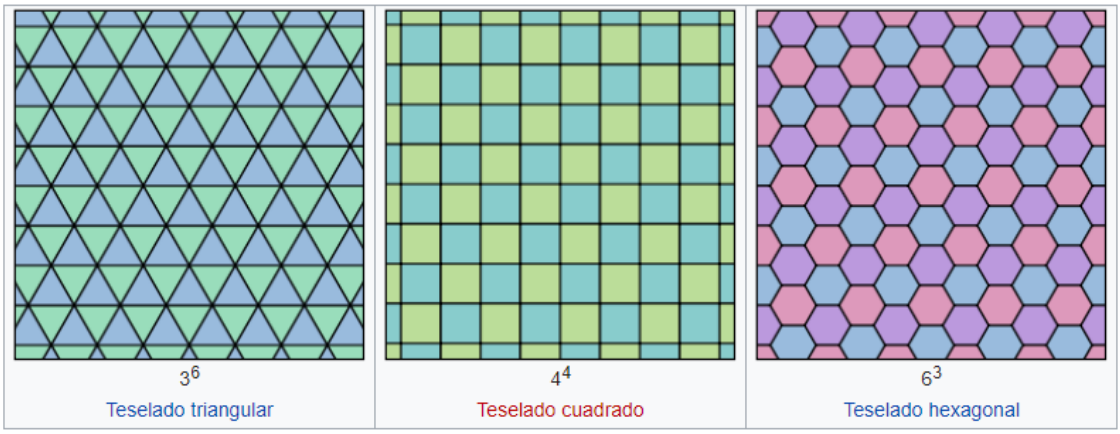


Figura 1.9. Teselados Regulares

1.7.3. Teselados Semirregulares

Son aquellos que contienen dos o más polígonos regulares en su formación.

Un teselado semirregular tiene las siguientes propiedades:

- Está formado sólo por polígonos regulares.
- El arreglo de polígonos es idéntico en cada vértice.
- Sólo existen ocho teselados semirregulares.

Figura 1.10. Teselados Semirregulares

2. MARCO METODOLOGICO

2.1. ESQUEMAS DE NAVEGACIÓN

Una primera definición del problema de la planificación, ya sea global o local, consiste en encontrar una ruta segura capaz de llevar al vehículo desde la posición actual hasta la especificada de destino. El concepto de ruta segura implica el cálculo de un camino al menos continuo en posición, que sea libre de obstáculos. En virtud de esta ruta, el generador construirá las referencias que se le entregan al control de movimientos. Por ello, en la especificación de esta ruta se obvian las características cinemáticas y dinámicas del vehículo, ya que el cómputo de una referencia adecuada que cumpla con estos atributos es tarea del generador de caminos. Por tanto, la ruta al tan sólo asegurar continuidad en posición, supone que únicamente los robots móviles omnidireccionales puedan seguir una referencia de tales características. En los siguientes subapartados se propone en primer lugar, una formalización del concepto de ruta, para, a continuación, describir métodos que, en atención a diferentes enfoques, realizan la acción de planificación.

2.1.1. Planificación De La Ruta Usando Espiral De Fermat

La espiral de Fermat presenta una curvatura suave que va desde cero hasta un valor deseado. Esta puede usarse para suavizar y reflejar una curva, como una opción alternativa a otras como las polinómicas o Clotoide.

Para realizar la curva mediante el espiral de Fermat, se debe trazar 2 gráficas donde una es el reflejo de la otra. Para la primera mitad de la curva, se elige la siguiente parametrización:

$$\mathbf{p_{Fermat}} = \begin{bmatrix} N_0 + \kappa\sqrt{\varpi\theta_{final}}\cos\left(\rho\varpi\theta_{final} + \chi_0\right) \\ E_0 + \kappa\sqrt{\varpi\theta_{final}}\sin\left(\rho\varpi\theta_{final} + \chi_0\right) \end{bmatrix}$$

Donde $P_0 = [N_0, E_0]^T$, es el punto de inicio para la trayectoria, K es una escala constante, θ_{final} $corresponde$ al ángulo polar correspondiente al final de la curva, ρ es la dirección de rotación y X_0 es el ángulo de curso inicial.

La curva simétrica tiene la siguiente parametrización:

$$\mathbf{p_{Fermat}} = \begin{bmatrix} N_{final} + \kappa\sqrt{\theta_{final} - \varpi\theta_{final}}\cos\left(\rho(\varpi\theta_{final} - \theta_{final}) + \chi_0\right) \\ E_{final} + \kappa\sqrt{\theta_{final} - \varpi\theta_{final}}\sin\left(\rho(\varpi\theta_{final} - \theta_{final}) + \chi_0\right) \end{bmatrix}$$

Se debe tener en cuenta que la espiral de Fermat, nos sirve para unir las partes donde hay cambio de ruta y las demás deben ser unidas por líneas rectas. Tomando como base esta parametrización evaluamos los puntos dados y obtenemos el resultado que se muestra en la Figura 2.1. Las dos primeras curvas corresponden a la posición X-Y, donde se ve claramente la parte definida por

líneas y curvas de Fermat, podemos ver que se presenta curvas más suaves, respecto a la gráfica de cambios de curso no tienen cambios abruptos, mientras la gráfica de curvatura muestra claramente un mejor comportamiento.

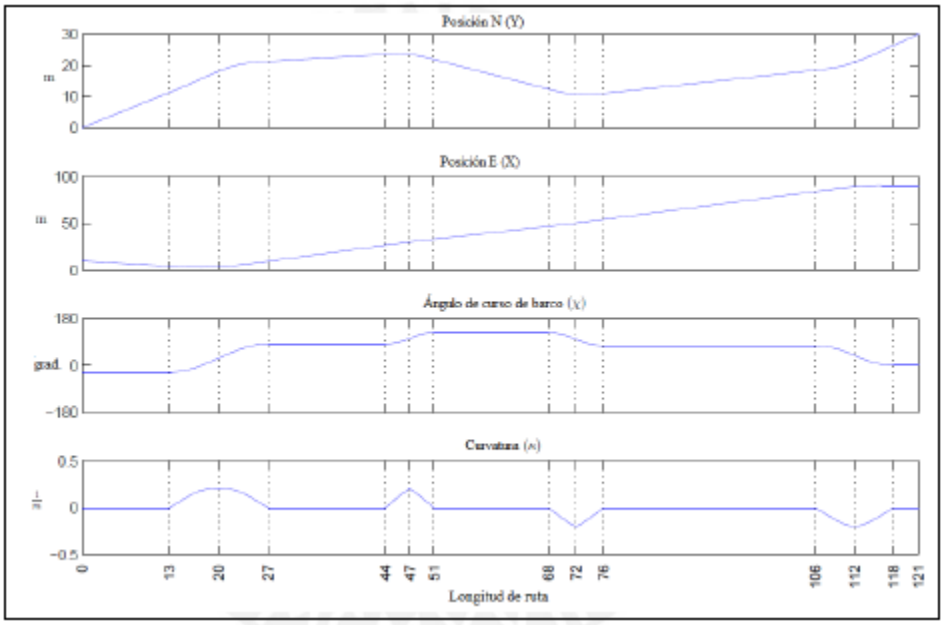


Figura 2.1. Espiral de Fermat y sus características geométricas.

Aquí solo se han presentado dos métodos que se pueden usar para la generación de rutas, se debe tener en cuenta que existen más técnicas como los que usan rectas y círculos, spline, espirales de Clotoide, funciones sigma, etc.

2.1.2. Métodos Clásicos De planificación

Todos ellos se fundamentan en una primera fase de construcción de algún tipo de grafo sobre el espacio libre, según la información poseída del entorno, para posteriormente emplear un algoritmo de búsqueda en grafos (por ejemplo, tipo A*) que encuentra el camino óptimo según cierta función de coste.

2.1.2.1. Planificación Basada En Grafos De visibilidad

Los grafos de visibilidad proporcionan un enfoque geométrico útil para resolver el problema de la planificación. Supone un entorno bidimensional en el cual los obstáculos están modelados mediante polígonos. Para la generación del grafo este método introduce el concepto de visibilidad, según el cual define dos puntos del entorno como visibles si y solo si se pueden unir mediante un segmento rectilíneo que no intersecte ningún obstáculo (si dicho segmento resulta tangencial a algún obstáculo se consideran los puntos afectados como visibles). En otras palabras, el segmento definido debe yacer en el espacio libre del entorno Cl.

Así, si se considera como nodos del grafo de visibilidad la posición inicial, la final y todos los vértices de los obstáculos del entorno, el grafo resulta de la unión mediante arcos de todos aquellos nodos que sean visibles.

En la figura 2.2 se muestra el grafo de visibilidad construido merced a los obstáculos poligonales existentes en el entorno y las configuraciones inicial qa y final qf.

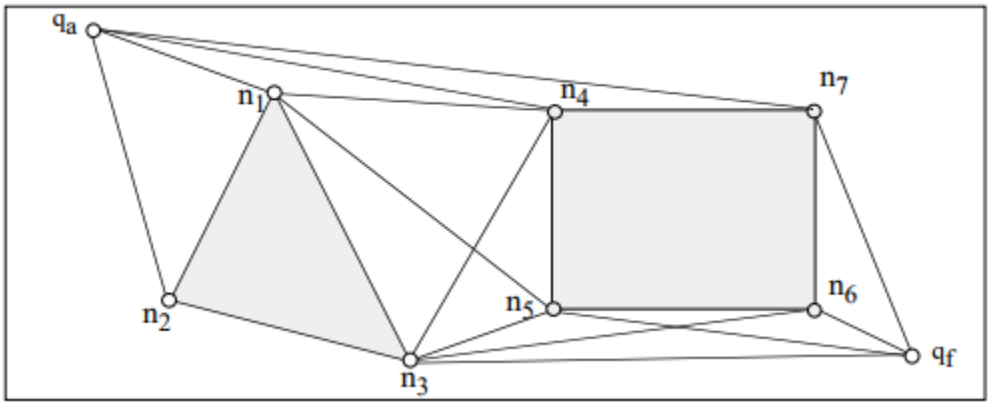


Figura 2.2. Grafo de visibilidad en un entorno de dos obstáculos

En el grafo mostrado (Figura 2.2), se puede observar cómo sólo están unidos los nodos directamente visibles, de tal forma que el conjunto de arcos estará formado por las aristas de los obstáculos, más el resto de líneas que relacionan los vértices de los diferentes polígonos. Mediante un algoritmo de búsqueda en grafos se elige la ruta que una la configuración inicial con la final minimizando alguna función de coste. La ruta que cumple el objetivo de la navegación queda definida como una sucesión de segmentos que siguen los requisitos especificados.

Aunque en principio el método está desarrollado para entornos totalmente conocidos, existe una versión denominada LNAV capaz de efectuar una planificación local a medida que se realiza la labor de navegación. Este algoritmo, que parte de una determinada posición, determina los nodos visibles desde el punto actual. Elige el más cercano de los nodos visibles, según distancia euclídea a la posición final, para desplazarse posteriormente al nodo seleccionado y marcarlo como visitado. Desde esta nueva posición se vuelve a iterar el proceso hasta llegar a la posición final, o bien no existen más nodos sin visitar.

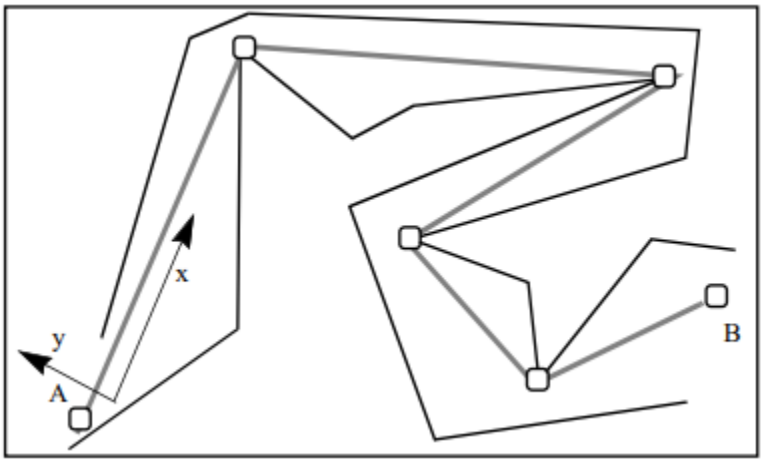


Figura 2.3. Planificación con el espacio libre de obstáculos modelado mediante cadenas.

Aunque están restringidos a esquemas de entornos muy concretos, el uso queda justificado debido a su bajo coste computacional. Como se puede observar en la figura 2.3, los algoritmos desarrollados para encontrar la ruta óptima bajo las condiciones descritas, se basan en enlazar los nodos situados en las zonas convexas del entorno tal que dos nodos consecutivos son visibles. El uso de métodos de planificación basados en grafos de visibilidad está muy extendido, debido a que se pueden construir algoritmos a bajo coste computacional que resuelvan el referido problema. Sin embargo, utilizar como nodos los vértices de los obstáculos implica que no son inmediatamente aplicables en la práctica, ya que un robot móvil real no consiste en un punto. Por ello, algunos autores denominan a la ruta planificada semi-libre de obstáculos.

2.1.2.2. Planificación Basada En Diagramas De Voronoi

De acuerdo a la anterior definición, un diagrama de Voronoi es la proyección del espacio libre del entorno en una red de curvas unidimensionales yacientes en dicho espacio libre. Se definen como una reacción con preservación de la continuidad. Si el conjunto Cl define las posiciones libres de obstáculos de un entorno, la función retracción RT construye un subconjunto Cv continuo de Cl.

$$RT(q)\colon Cl \rightarrow Cv / Cv \subset Cl$$

De esta forma se dice que existe un camino desde una configuración inicial qa hasta otra final qf siendo las dos libres de obstáculos, si y solo si existe una curva continua desde $RT(qa)$ hasta $RT(qf)$.

La definición de la función retracción RT implica la construcción del diagrama de Voronoi. La idea fundamental, es construir la ruta lo más alejada posible de los obstáculos, con ello se elimina el problema de los grafos de visibilidad de construir rutas semi-libres de obstáculos, es decir, ampliar al máximo la distancia entre el camino del robot y los obstáculos. Por ello, el diagrama de Voronoi

resulta el lugar geométrico de las configuraciones que se encuentran a igual distancia de los dos obstáculos más próximos del entorno.

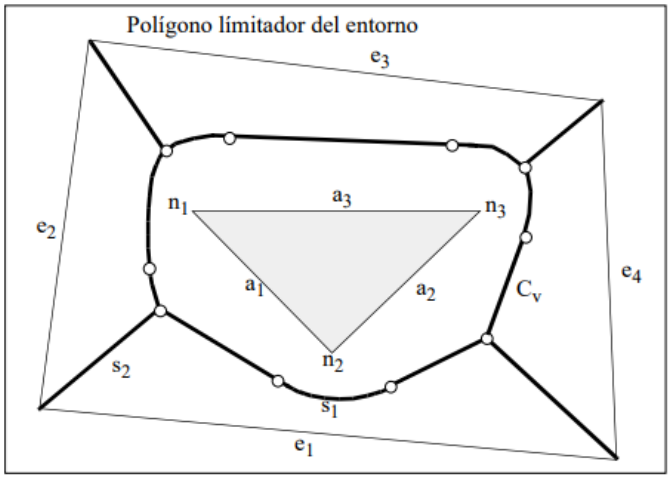


Figura 2.4. Retracción del espacio libre en un diagrama de Voronoi.

En la figura 12 muestra el entorno delimitado por un polígono de aristas {$e1$,$e2$,$e3$,$e4$} y un obstáculo triangular de vértices {$n1$, $n2$, $n3$} y aristas {$a1$, $a2$, $a3$}. La retracción del espacio libre en una red continua de curvas es el diagrama de Voronoi Cv, representado mediante las líneas de trazo grueso. Los dos tipos de segmento utilizados en la construcción del diagrama pueden distinguirse en la mencionada figura, así, el segmento $S1$ es el lugar geométrico de los puntos equidistantes entre la arista $e1$, y el vértice $n2$. Por otra parte, puede observarse como el segmento rectilíneo $S2$ cumple la misma condición, pero con respecto a las aristas $e1$ y $e2$.

Dado una configuración q no perteneciente a Cv, existe un único punto p más cercano perteneciente a un vértice o arista de un obstáculo. La función $RT(q)$ se define como el primer corte con Cv de la línea que une p con q.

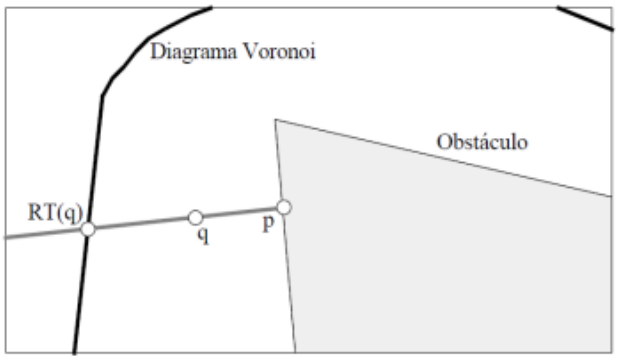


Figura 2.5. Imagen de una configuración q en el diagrama de Voronoi.

Este algoritmo consiste principalmente en encontrar la secuencia de segmentos Si del diagrama de Voronoi tal que conecten $RT(qa)$ con $RT(qf)$, dicha secuencia conforma la ruta buscada, los pasos que conforman este algoritmo son:

- Calcular el diagrama de Voronoi
- Calcular $RT(qa)$ y $RT(qf)$
- Encontrar la secuencia de segmentos $\{S1, \dots, Sp\}$ tal que $RT(qa)$ pertenezca a $S1$ y $RT(qf)$ a Sp.
- Si se encuentra la secuencia, devolver ruta. Si no, indicar condición de error.

Al igual que los grafos de visibilidad el método de diagramas de Voronoi también trabaja en entornos totalmente conocidos y con obstáculos modelados mediante polígonos.

2.1.2.3. Planificación basada En Descomposición De Celdas

Este método consiste en dividir el problema en dos niveles: uno global que resuelve la trayectoria por zonas y otro, más simple, a nivel local. Se trata de dividir el entorno en un conjunto de celdas. Estas han de tener la función de que resulte sencilla la conexión de dos puntos cualesquiera de su interior, ya sea por ser muy próximos, porque la celda sea convexa, etc. Una vez discretizado el espacio de configuraciones libres de colisión en un conjunto de celdas, se procede a resolver su conectividad a un nivel superior, es decir, construyendo un grafo de adyacencia, cada celda se le asigna un nodo, y estos se unirán o no en función de que las celdas que representen sean adyacentes. Existen dos tipos de este método:

1. Descomposición exacta:
2. Descomposición vertical

Se usa este término cuando la descomposición genera un conjunto de celdas cuya unión resulta idéntica al espacio de configuraciones libres de colisión, es decir, las celdas deben estar definidas de tal modo que se adapten a la configuración de los obstáculos. Un entorno en el que el escenario y los obstáculos tienen límites poligonales, se pueden trazar rectas verticales en cada vértice. De esta forma, todo el espacio de configuraciones libres de colisión queda fragmentado en celdas trapezoidales o triangulares en su totalidad. En esta basta por elegir los puntos medios de los segmentos adyacentes como puntos de paso. Trazando segmentos rectos entre los mismos se obtiene la solución.

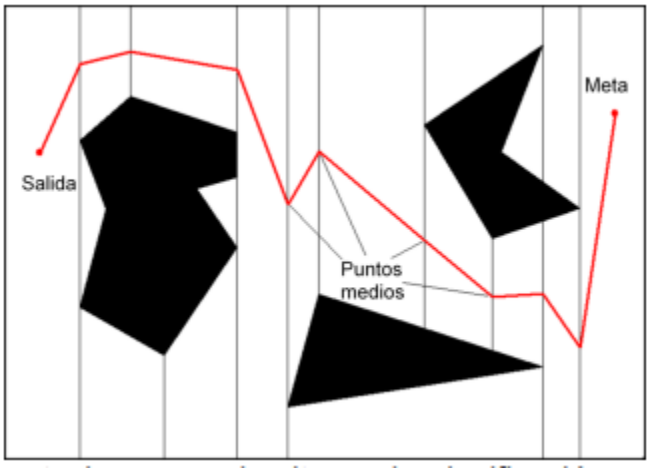


Figura 2.6. Trayectoria definida por descomposición vertical

2.1.2.4. Planificación Basada En Descomposición Delaunay.

Otro ejemplo de descomposición exacta la constituye la llamada triangulación de Delaunay. Esta considera inicialmente el conjunto de puntos formado por los vértices de los obstáculos y luego une en un segmento los pares de puntos tales que sean posible hallar una circunferencia que los contenga sin inscribir ningún otro punto. Esta exclusión implica que los puntos del par son los más cercanos al centro de la circunferencia hallada. Además, dicho centro equidista de los puntos a unir, puesto que, para dos puntos cualesquiera de una circunferencia, la bisectriz del segmento que definen contiene siempre al centro de la misma. Los dos resultados obtienen que este centro pertenecerá al diagrama de Voronoi que definen los vértices de los obstáculos. Pero el método no resuelve el diagrama de Voronoi, si no que únicamente define una descomposición de la configuración libre de colisión en celdas triangulares delimitados por los segmentos definidos de con la propiedad mencionada.

Sobre estas celdas se construye un grafo de adyacencia, y a partir de esto se realiza lo mismo que para el primer ejemplo mencionado en este apartado.

Este método resulta algo más complejo, los puntos medios de los segmentos de estas celdas se distancian con mayor eficiencia de los obstáculos, y la solución obtenida por este es de mayor margen de seguridad para evitar colisiones.

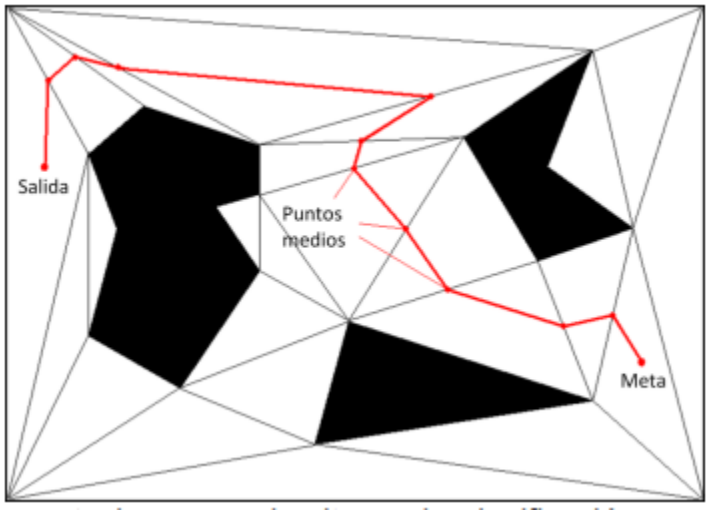


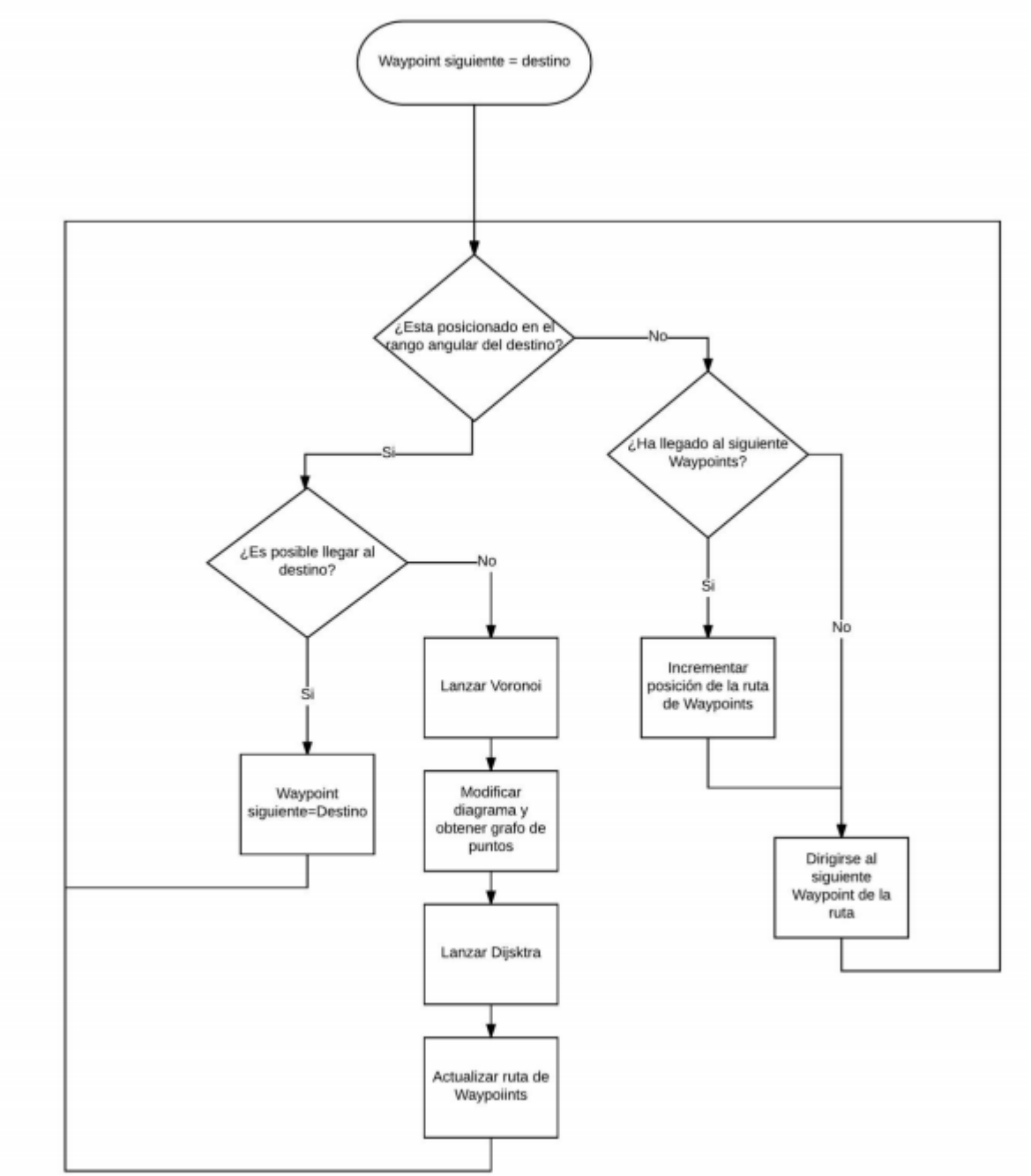
Figura 2.7. Trayectoria definida por descomposición por Delaunay.

2.2. GENERACIÓN DEL CAMINO

Una vez modelados los obstáculos detectados, se debe establecer un plan de actuación según la situación del robot y estos obstáculos. Al comienzo el robot se dirige al punto de destino. Durante la navegación del robot y con una cierta frecuencia se irán analizando las lecturas del sensor, para comprobar si el robot puede alcanzar el destino o puede colisionar con un obstáculo. En caso de posible colisión, se lanza el algoritmo de Voronoi y el algoritmo Djikstra para actualizar camino generando nuevos waypoints que aseguren la evitación de la colisión.

De esta forma, solo se computará el algoritmo de evitación de obstáculos cuando sea necesario, evitando una posible ralentización del sistema. Respecto al seguimiento de caminos, cuando el robot esté a una distancia menor de 10 cm del waypoint actual se dirigirá al próximo waypoint. Esto aporta fluidez y continuidad a las trayectorias, pues la velocidad lineal es reducida y no da lugar a problemas de aproximamiento excesivo a los obstáculos.

Se puede entender de forma más visual con el siguiente diagrama de flujo:



Para determinar la posición de los waypoints respecto al sistema de coordenadas global debemos realizar una traslación y rotación de los ejes coordenados, pues el sensor da las coordenadas en su sistema de referencia local en coordenadas polares (módulo y ángulo), que posteriormente hemos pasado a un sistema de coordenadas cartesianas local del sensor con la matriz de obstáculos

3. SIMULACION DEL ALGORITMO

A continuación, veremos paso por paso como se realizó la creación de cada parte del menú y su respectivo código que nos permita la correcta utilización de ellas.

3.1. CREACIÓN DE LAS GUIDE

Las GUI (también conocidas como interfaces gráficas de usuario o interfaces de usuario) permiten un control sencillo (con uso de ratón) de las aplicaciones de software, lo cual elimina la necesidad de aprender un lenguaje y escribir comandos a fin de ejecutar una aplicación.

Las apps de MATLAB son programas autónomos de MATLAB con un frontal gráfico de usuario GUI que automatizan una tarea o un cálculo. Por lo general, la GUI incluye controles tales como menús, barras de herramientas, botones y controles deslizantes. Muchos productos de MATLAB, como Curve Fitting Toolbox, Signal Processing Toolbox y Control System Toolbox, incluyen apps con interfaces de usuario personalizadas. También es posible crear apps personalizadas propias, incluidas las interfaces de usuario correspondientes, para que otras personas las utilicen.

GUIDE (entorno de desarrollo de GUI) proporciona herramientas para diseñar interfaces de usuario para Apps personalizadas. Mediante el editor de diseño de GUIDE, es posible diseñar gráficamente la interfaz de usuario. GUIDE genera entonces de manera automática el código de MATLAB para construir la interfaz, el cual se puede modificar para programar el comportamiento de la app.

3.1.1. Creación del Espacio De Trabajo

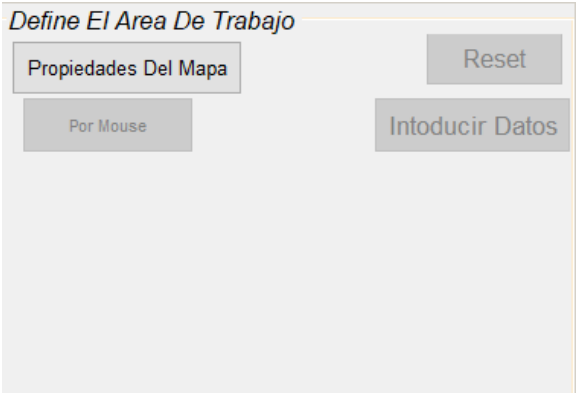


Figura 3.1. Menú Espacio De Trabajo

3.1.1.1. Propiedades del Mapa

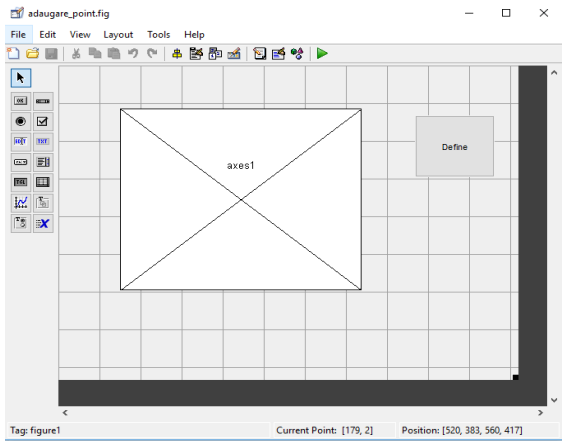


Figura 3.2. GUIDE Propiedades Del Mapa

3.1.1.2. Creación del Mapa

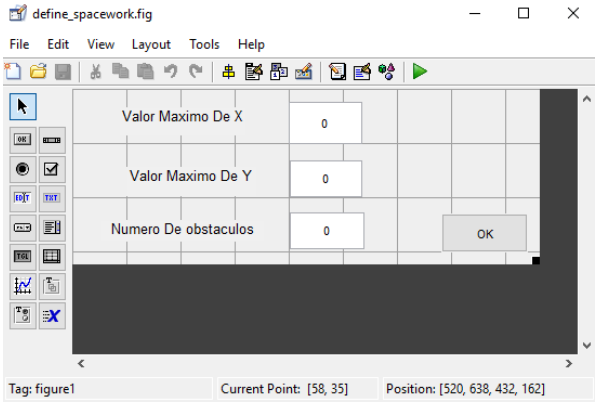


Figura 3.3. GUIDE creación del Mapa

3.1.2. Trayectoria

3.1.2.1. Menú de punto de Inicio y Final del Trayecto

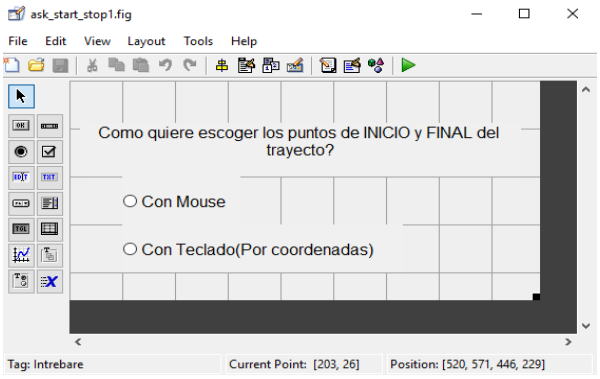


Figura 3.4. GUIDE Punto de INICIO/FINAL del trayecto

3.1.2.2. Generación de Inicio y Final por Coordenadas (Teclado)

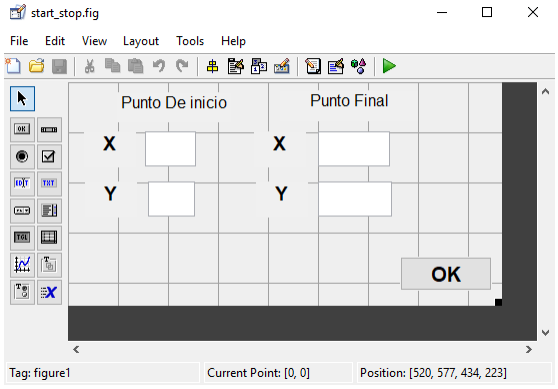


Figura 3.5. GUIDE Punto INICIO/FINAL por Coordenadas

3.1.2.3. Menú Para el Tipo de Búsqueda

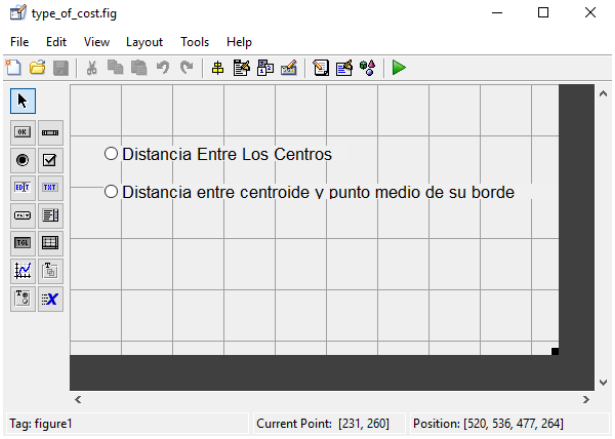


Figura 3.6. GUIDE Tipo De Búsqueda

3.1.3. Unión del menú general

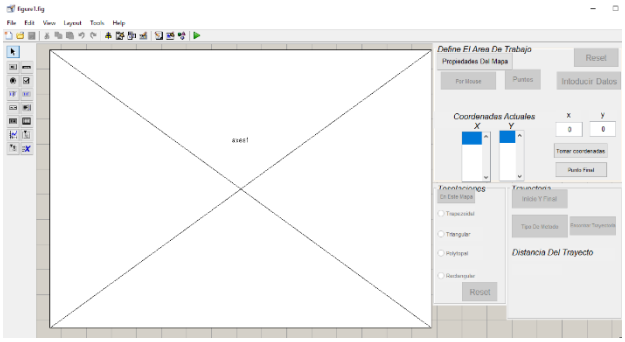


Figura 3.7. GUIDE Menú General

3.1.4. Creación del Menú Total por teclado y por Mouse

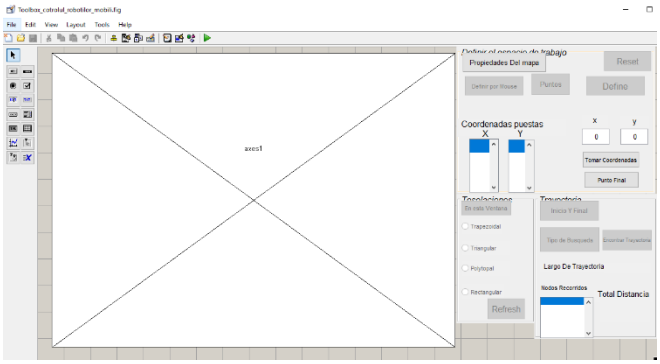


Figura 3.8. GUIDE Menú por Mouse

3.1.5. Creación De Mensajes de errores

3.1.5.1. Inserción de menos puntos en la generación de obstáculos

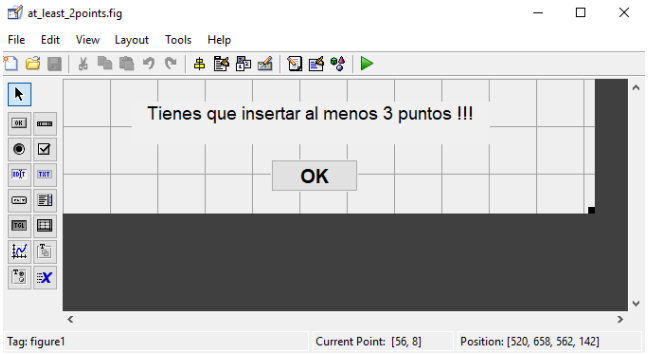


Figura 3.9. GUIDE Error de Obstáculos

3.1.5.2. Error Por no Definición Del Espacio De Trabajo

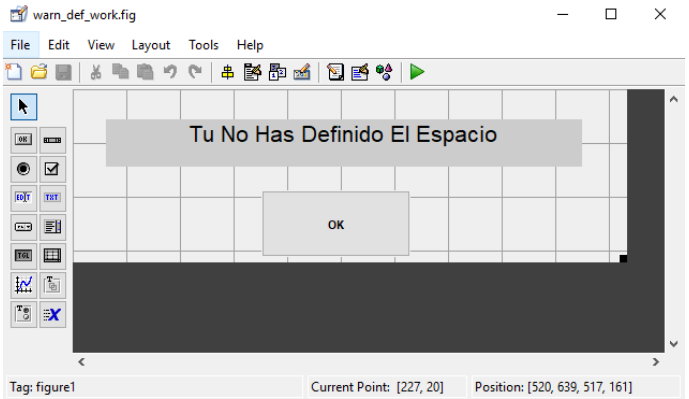


Figura 3.10. GUIDE Error De Espacio de Trabajo

3.1.5.3. Error por no colocar suficientes puntos en los obstáculos

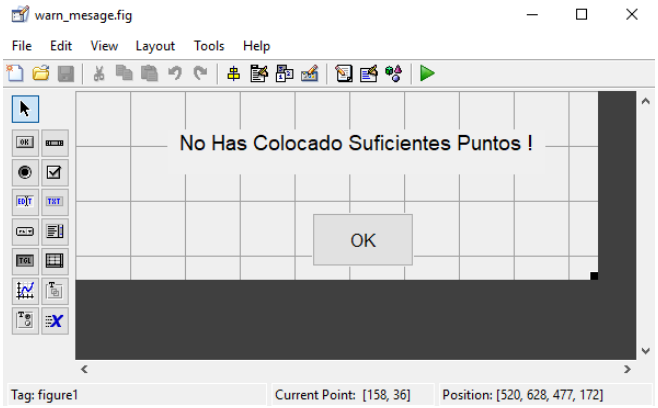


Figura 3.11. GUIDE Error Puntos De Obstáculos

3.1.5.4. Error por tratar de generar más obstáculos de los estipulados

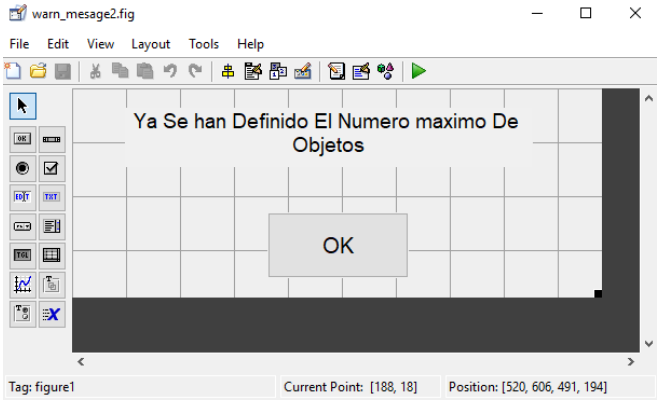


Figura 3.12. GUIDE Error Puntos de Obstáculos Excedidos

3.1.5.5. Error por Digitar Coordenadas mayores a la estipuladas

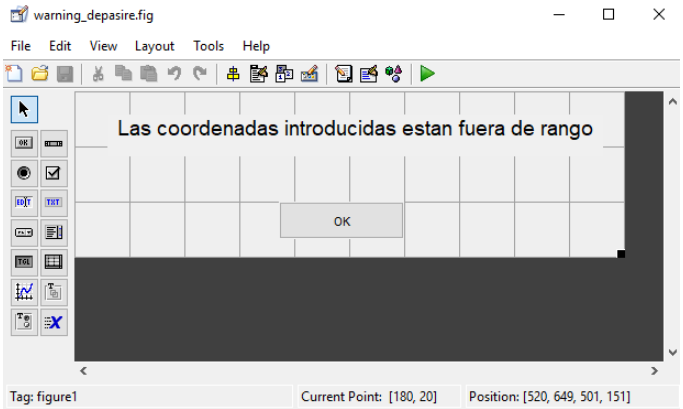


Figura 3.13. GUIDE Error Coordenadas Excedidas

3.1.5.6. Error por colocar los Puntos de INICIO/FINAL dentro de un obstáculo

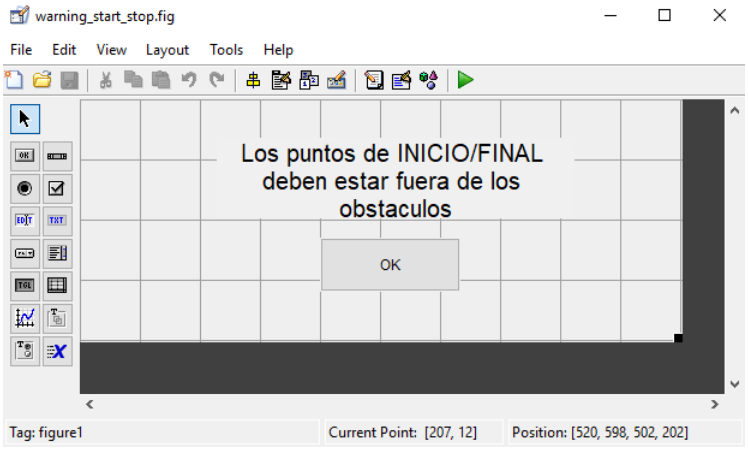


Figura 3.14. GUIDE Error puntos INICIO/FINAL

3.2. CREACIÓN DE LOS CÓDIGOS

3.2.1. Generación Del Mapa

```
function varargout = adaugare_point(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @adaugare_point_OpeningFcn, ...
                   'gui_OutputFcn',  @adaugare_point_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin Del Codigo Inicial


% --- Se ejecuta justo antes de que adaugare_point se haga visible
function adaugare_point_OpeningFcn(hObject, eventdata, handles,
varargin)
% Esta función no tiene args de salida
```

```
% Elija la salida de línea de comandos predeterminada para
adaugare_point
handles.output = hObject;

guidata(hObject, handles);

% UIWAIT hace que adaugare_point espere a la respuesta del usuario
global h;

% --- Las salidas de esta función se devuelven a la línea de comandos.
function varargout = adaugare_point_OutputFcn(hObject, eventdata,
handles)

% Obtener salida de línea de comandos predeterminada desde la estructura
de controladores
varargout{1} = handles.output;

function axes1_CreateFcn(hObject, eventdata, handles)

% --- Ejecuta en la prensa del mouse sobre el fondo de los ejes.
function axes1_ButtonDownFcn(hObject, eventdata, handles)

global h;
h = gca
handles.output = hObject

guidata(hObject, handles);

% --- Ejecuta el boton pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

S.fh = figure('units','pixels',...
    'position',[400 400 400 150],...
    'menubar','none',...
    'numbertitle','on',...
    'name','Lungime axe',...
    'resize','off');
S.ed(1) = uicontrol('style','edit',...
    'unit','pix',...
    'position',[10 80 70 30],...
    'string','0');
S.ed(2) = uicontrol('style','edit',...
    'unit','pix',...
    'position',[110 80 70 30],...
    'string','0');
S.tx(1) = uicontrol('style','text',...
    'unit','pix',...
    'position',[10 120 80 30],...
    'string','X  ',...
    'fontsize',16);
S.tx(2) = uicontrol('style','text',...
    'unit','pix',...
```

```
    'position',[105 120 80 30],...
    'string','Y  ',...
    'fontsize',16);
S.pb = uicontrol('style','pushbutton',...
    'units','pix',...
    'position',[280 20 80 30],...
    'string','plot',...
    'callback',{@pb_call,S});

function [] = pb_call(varargin)
% Devolucion para pushbutton

S = varargin{3};
N = str2double(get(S.ed(1:2),'string'));% Estos son los numeros para
operar.
world_dim=[0; 7;0 ;7]
x=N(1);
y=N(2);
 x_max=7;
 y_max=7;
 H = gca
 get(gcf,'CurrentAxes')
 d=guidata(h)
 global h;
    axes(h);
findall(handles.axes1,'type','axes')
set(handles.axes1,'Parent', axes_handle)
axis([0,x_max,0,y_max]);
title('Axes 1');
plot(x,y,'.k')


% ---Se ejecuta durante la creación del objeto, después de establecer
todas las propiedades
```

3.2.2. Generación de Punto de INICIO/FINAL

```
function varargout = ask_start_stop1(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ask_start_stop1_OpeningFcn, ...
                   'gui_OutputFcn',  @ask_start_stop1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del codigo de inicializacion


% --- Se ejecuta justo antes de que ask_start_stop1 se haga visible.

function ask_start_stop1_OpeningFcn(hObject, eventdata, handles,
varargin)

%Elija la salida de línea de comandos predeterminada para
ask_start_stop1
handles.output = hObject;

guidata(hObject, handles);

%UIWAIT hace que ask_start_stop1 espere la respuesta del usuario (vea
UIRESUME)
% uiwait(handles.Intrebare);


% --- Las salidas de esta función se devuelven a la línea de comandos.
function varargout = ask_start_stop1_OutputFcn(hObject, eventdata,
handles)

% Obtener salida de línea de comandos predeterminada desde la estructura
de controladores

varargout{1} = handles.output;


% --- Ejecuta la pulsación de botón en radio1.
function radio1_Callback(hObject, eventdata, handles)

% Sugerencia: get (hObject, 'Value') devuelve el estado de alternancia
de radio1
if (get(hObject,'Value') == get(hObject,'Max'))
    % El botón de radio está seleccionado-tome la acción apropiada

   set(handles.radio2,'Enable','off');
   value1= 'mouse';
   setappdata(0,'some_variable1', value1);
else
    setappdata(0,'some_variable1', 'nimic');

end
close;
```

```
% --- Ejecuta la pulsación de botón en radio2.
function radio2_Callback(hObject, eventdata, handles)

% Sugerencia: get (hObject, 'Value') devuelve el estado de alternancia
de radio2
if (get(hObject,'Value') == get(hObject,'Max'))
% El botón de radio está seleccionado-tome la acción apropiada

   set(handles.radio2,'Enable','off');
   value1= 'keyboard';
   setappdata(0,'some_variable1', value1);
else
    setappdata(0,'some_variable1', 'nimic');

end
 close;

% --- Se ejecuta durante la eliminación de objetos, antes de destruir
propiedades.
function Intrebare_DeleteFcn(hObject, eventdata, handles)

function Untitled_1_Callback(hObject, eventdata, handles)
```

3.2.3. creación de obstáculos en el mapa

```
function varargout = at_least_2points(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @at_least_2points_OpeningFcn, ...
                   'gui_OutputFcn',  @at_least_2points_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% --- Se ejecuta justo antes de at_least_2points se hace visible.
function at_least_2points_OpeningFcn(hObject, eventdata, handles,
varargin)
```

```
% Elige la salida de línea de comandos predeterminada para
at_least_2points
handles.output = hObject;

% Actualizar estructura de manejadores
guidata(hObject, handles);

% UIWAIT hace que at_least_2points espere la respuesta del usuario
(consulte UIRESUME)

% --- Las salidas de esta función se devuelven a la línea de comandos.
function varargout = at_least_2points_OutputFcn(hObject, eventdata,
handles)

% Obtener salida de línea de comandos predeterminada desde la estructura
de controladores
varargout{1} = handles.output;


% --- Ejecuta la pulsación del botón en el pulsador2.
function pushbutton2_Callback(hObject, eventdata, handles)
close
```

3.2.4. Creación de coordenadas y numero de obstáculos

```
function objects = define_regions(x_max,y_max)

world_dim=[0; x_max;0 ;y_max]
obs_no=input('Number of regions: ');

fprintf('\nFor defining a region:\n\t - left-click = pick a vertex\n\t -
right-click = pick last vertex\nRegions should be convex and non-
overlapping\n\nPress any key to begin.\n')
pause();

init_wrld_h=figure();
axis(world_dim);
hold on
grid
% Leer los vértices del obstáculo
for i=1:obs_no   % I = número de objeto

    j=1; % J = no. De vértices para el objeto actual
    but=1;
    while but==1
        [x,y,but]=ginput(1);
        plot(x,y,'.k')
        objects{i}(:,j)=[x;y];
        j=j+1;
    end
```

```
    % Crear obstáculos convexos y dibujarlos
    k=convhull(objects{i}(1,:),objects{i}(2,:));
    objects{i}=objects{i}(:,k(1:length(k)-1));
    pause(0.3)
    fill(objects{i}(1,:),objects{i}(2,:),'k','FaceAlpha',0.5);
end
```

3.2.5. Union de nodos para los Obstaculos

```
function [path, cost] = find_path(adj,s,d)

n=size(adj,1); % Número de nodos

visited=zeros(1,n); % Número de nodos% si se consideró un nodo, tiene
valor 1 en el vector "visitado" es

dist=Inf(1,n); % De distancia de s a nodos (se modificará)
dist(s)=0;
predec=zeros(1,n); % Predecesor de cada nodo de búsqueda hasta que todos
los nodos se visitan
while sum(visited)~=n
    d_m=min(dist(find(visited==0)));
    x=find(dist==d_m & visited==0);
    x=x(1); % Si hay más nodos a la misma distancia (se considerarán en
las iteraciones siguientes)
    visited(x)=1;
    neigh=find(adj(x,:)~=0 & visited==0); % Vecinos no visitados del
nodo actual x (es adj no era escaso, con "inf" en lugar de 0, use
isfinite (adj (x, :))
    if length(neigh)==0
       continue % Del nodo actual no tiene más vecinos no visitados,
continúe con las iteraciones siguientes (otras "ramas")

    end
    for i=neigh % Para cada vecino no visitado, compruebe si la
distancia al pasar por x a ella es menor que la de hasta ahora
        if (dist(i) > dist(x)+adj(x,i))
            dist(i)=dist(x)+adj(x,i);   % De actualización de la
distancia al nodo i (más pequeño que el anterior)
            predec(i)=x;   % X se convierte en el predecesor de i

        end
    end
end

% Tenemos distancias mínimas de s a todos los nodos en el gráfico
if dist(d)~=Inf
    path=[d]; % Pone el nodo de destino en la ruta y comienza a pasar
por los predecesores (búsqueda "hacia atrás"), hasta que se alcance s
```

```
    while path(1)~=s    % Encontrar todos los nodos entre s y d (i),
hasta que s se alcanza (porque sabemos que hay una trayectoria s-> d)

        path=[predec(path(1)) path];  % Agrega un predecesor en el
camino, delante de los otros; S estará en la primera posición en el
camino

    end
else
    path=[];
end
cost=dist(d);    % Del costo de la ruta
```

3.2.6. Encontrar la Trayectoria

```
function [trajectory, distance, path, cost_path] =
find_trajectory(C,adj,mid_X,mid_Y,x_st,y_st,x_fin,y_fin,varargin)


%% Buscar células de inicio y destino
start_cell=0;
destination_cell=0;
for i=1:length(C)   % Índices de células iniciales y finales
    if inpolygon(x_st,y_st,C{i}(1,:),C{i}(2,:))
        start_cell=i;
    end
    if inpolygon(x_fin,y_fin,C{i}(1,:),C{i}(2,:))
        destination_cell=i;
    end
end

if start_cell==0    % Inicio célula no encontrada (ya sea en el
obstáculo, o descomposición aproximada como rectangular)
    disp('varargin{1}');
    celldisp(varargin{1});
% Obstacles = varargin {1}; % Si se pasó este argumento
    for i=1:length(obstacles)
        if inpolygon(x_st,y_st,obstacles{i}(1,:),obstacles{i}(2,:))
            fprintf('\nStart position inside obstacle O_%g.\n',i);
            trajectory=[]; distance=inf; path=[]; cost_path=inf;
            return;
        end
    end
   % No en el obstáculo, elija como célula de inicio que tiene centroide
más cercano
    min_dist=inf;
    for i=1:length(C)   % Índices de células iniciales y finales
        if norm([x_st;y_st]-mean(C{i},2))<min_dist
            min_dist=norm([x_st;y_st]-mean(C{i},2));
            start_cell=i;
        end
```

```
    end
    fprintf('\nStart position not inside a cell. Moved to centroid of
cell c_%g.\n',i);
end

if destination_cell==0    % Célula de inicio no encontrada (ya sea en
obstáculo, o descomposición aproximada como rectangular)
    obstacles=varargin{1};  % Si se pasó este argumento
    for i=1:length(obstacles)
        if inpolygon(x_fin,y_fin,obstacles{i}(1,:),obstacles{i}(2,:))
            fprintf('\nDestination position inside obstacle O_%g.\n',i);
            trajectory=[]; distance=inf; path=[]; cost_path=inf;
            return;
        end
    end
    %not in obstacle, choose as start cell having closest centroid
    min_dist=inf;
    for i=1:length(C)   % No en el obstáculo, elija como célula de
inicio que tiene centroide más cercano
        if norm([x_fin;y_fin]-mean(C{i},2))<min_dist
            min_dist=norm([x_fin;y_fin]-mean(C{i},2));
            destination_cell=i;
        end
    end
    fprintf('\Destination position not inside a cell. Moved to centroid
of cell c_%g.\n',i);
end

%% path de búsqueda

[path, cost_path] = find_path(adj,start_cell,destination_cell);  % De
secuencia de células a seguir
if isempty(path)
    trajectory=[]; distance=inf; path=[]; cost_path=inf;
    fprintf('\nA path cannot be found.\n');
    return;
end
if length(path)==1  % Caso particular (inicio y meta en la misma celda)
    trajectory=[[x_st;y_st] , [x_fin;y_fin]];
    distance=norm([x_st;y_st]-[x_fin;y_fin]);
    return
end

%% encuentra puntos a lo largo de la trayectoria continua (segmentos de
línea conectados)
trajectory=zeros(2,length(path)+1);   % Add punto de inicio, que medio
de segmento de línea compartida con celda siguiente, y así sucesivamente
hasta la celda final

distance=0; % Para añadir distancia recorrida
trajectory(:,1)=[x_st;y_st];    %posición inicial
for i=2:length(path)
```

```
    trajectory(:,i) = [mid_X(path(i-1),path(i)) ; mid_Y(path(i-
1),path(i))];  % Punto medio del segmento compartido por dos estados
sucesivos

    distance = distance + norm(trajectory(:,i)-trajectory(:,i-1));
end
trajectory(:,end) = [x_fin;y_fin]; %Punto de meta

distance = distance+norm(trajectory(:,end)-trajectory(:,end-1));
```

3.2.7. Dibujar Obstaculos en el mapa

```
function  plot_environment(handl,objects,x_max,y_max,varargin)

 axis([0 x_max 0 y_max]);

  set(handl,'Box','on');


switch nargin
    case 4
        %Los obstaculos ya estan dibujados
    case 5  %Dibujar la descomposiion de las celulas
        disp('acesta este varargin');
        C=varargin{1}
        %Representar las celulas:
        for i=1:length(C)
            fill(C{i}(1,:),C{i}(2,:),'w','FaceAlpha',0.5);
        end

        %Escribir el numero de celulas
        for i=1:length(C)
            centr=mean(C{i},2)';

text(centr(1),centr(2),sprintf('c_{%d}',i),'HorizontalAlignment','center
','Color','k','FontSize',10,'FontAngle','italic','FontName','TimesNewRom
an');
        end
        set(gca,'Box','on','XTick',[],'YTick',[]);

    case 6 %Dinujar la descomposicion de las celulas
        C=varargin{1};
        adj=varargin{2};
        %Representar las celulas:
        for i=1:length(C)
            fill(C{i}(1,:),C{i}(2,:),'w','FaceAlpha',0.5);
        end

        centr=zeros(length(C),2);    %Memoria de Centroides
        for i=1:length(C)
```

```
            centr(i,:)=mean(C{i},2)';
        end
        gplot(adj,centr,':b');  %Representar y graficar

        %Escribir el numero de Celulas
        for i=1:length(C)

text(centr(i,1),centr(i,2),sprintf('c_{%d}',i),'HorizontalAlignment','ce
nter','Color','k','FontSize',10,'FontAngle','italic','FontName','TimesNe
wRoman');
        end
        set(gca,'Box','on','XTick',[],'YTick',[]);

    case 8
        C=varargin{1};
        adj=varargin{2};
        middle_X=varargin{3};
        middle_Y=varargin{4};
        %Representar las celulas:
        for i=1:length(C)
            fill(C{i}(1,:),C{i}(2,:),'w','FaceAlpha',0.5);
        end

        centr=zeros(length(C),2);   %Memoria de Centroides
        for i=1:length(C)
            centr(i,:)=mean(C{i},2)';
        end
%         gplot(adj,centr,':b');  %Representar y graficar

        for i=1:length(C)
            for j=setdiff(find(adj(i,:)),1:i)
                plot(middle_X(i,j),middle_Y(i,j),'*r');
            end
        end
        %Escribir el numero de la celula
        for i=1:length(C)

text(centr(i,1),centr(i,2),sprintf('c_{%d}',i),'HorizontalAlignment','ce
nter','Color','k','FontSize',10,'FontAngle','italic','FontName','TimesNe
wRoman');
        end
        set(gca,'Box','on','XTick',[],'YTick',[]);

    otherwise
        fprintf('\nCheck # of arguments for the plot_environment
function\n');
end

for i=1:length(objects)
    fill(objects{i}(1,:),objects{i}(2,:),'r');
    centr=mean(objects{i},2)';
    axis([0 x_max 0 y_max]);
    set(handl,'Box','on');
```

```
    text(fix(centr(1)) ,fix( centr(2))
,sprintf('O_{%d}',i),'HorizontalAlignment','center','Color','g','FontSiz
e',12,'FontWeight','bold','FontAngle','italic','FontName','TimesNewRoman
');

end
```

3.2.8. Creación de color para los obstáculos al generar las teselaciones

```
function ph = plotfilledcircle(circle_radius,circlecenter, fcol)

if nargin < 2
 circlecenter = [0 0];
end
if nargin < 3
 fcol = [0 0 0];
end
theta = linspace(0,2*pi,100);
x = circle_radius*cos(theta) + circlecenter(1);
y = circle_radius*sin(theta) + circlecenter(2);
ph = fill(x, y, 'k');
set(ph, 'FaceColor', fcol);
box off; axis equal;
end
```

3.2.9. Ubicación del punto De Inicio y Final de la ruta por Coordenadas

```
function varargout = start_stop(varargin)
% START_STOP MATLAB code for start_stop.fig
%      START_STOP, by itself, creates a new START_STOP or raises the
existing
%      singleton*.
%
%      H = START_STOP returns the handle to a new START_STOP or the
handle to
%      the existing singleton*.
%
%      START_STOP('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in START_STOP.M with the given input
arguments.
%
%      START_STOP('Property','Value',...) creates a new START_STOP or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before start_stop_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
```

```
%      stop.  All inputs are passed to start_stop_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @start_stop_OpeningFcn, ...
                   'gui_OutputFcn',  @start_stop_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before start_stop is made visible.
function start_stop_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to start_stop (see VARARGIN)

% Choose default command line output for start_stop
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes start_stop wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = start_stop_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
xstart= get(handles.edit1,'String') ;
ystart= get(handles.edit2,'String') ;
xstop= get(handles.edit3,'String') ;
ystop= get(handles.edit4,'String') ;

setappdata(0,'xstart', xstart);
setappdata(0,'ystart', ystart);
setappdata(0,'xstop', xstop);
setappdata(0,'ystop', ystop);

close


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as
a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as
a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as
a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function pushbutton1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

3.2.10. Tipo De Teselacion

```
function varargout = type_of_cost(varargin)
% TYPE_OF_COST MATLAB code for type_of_cost.fig
%      TYPE_OF_COST, by itself, creates a new TYPE_OF_COST or raises the
existing
%      singleton*.
%
%      H = TYPE_OF_COST returns the handle to a new TYPE_OF_COST or the
handle to
%      the existing singleton*.
%
%      TYPE_OF_COST('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in TYPE_OF_COST.M with the given input
arguments.
%
%      TYPE_OF_COST('Property','Value',...) creates a new TYPE_OF_COST
or raises the
```

```
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before type_of_cost_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to type_of_cost_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @type_of_cost_OpeningFcn, ...
                   'gui_OutputFcn',  @type_of_cost_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%Terminar del codigo


% --- Ejecuta justo antes de que el tipo de Teselacion se haga visible.
function type_of_cost_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to type_of_cost (see VARARGIN)

% Choose default command line output for type_of_cost
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes type_of_cost wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = type_of_cost_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
if (get(handles.radiobutton1,'Value') == get(hObject,'Max'))
    % Radio button is selected-take appropriate action

   % set(handles.radio2,'Enable','off');
   cost_type= '1';
   setappdata(0,'cost_type', cost_type);
   close;

end

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2
if (get(handles.radiobutton2,'Value') == get(hObject,'Max'))
    % Radio button is selected-take appropriate action

   % set(handles.radio2,'Enable','off');
   cost_type= '2';
   setappdata(0,'cost_type', cost_type);
   close;

end

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
```

```
if (get(handles.radiobutton3,'Value') == get(hObject,'Max'))

   cost_type= '3';
   setappdata(0,'cost_type', cost_type);
   close;

end

function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (get(handles.radiobutton4,'Value') == get(hObject,'Max'))
    % El boton de Radio selected-take esta escogiendo la seleccion
    % apropiada

   cost_type= '4';
   setappdata(0,'cost_type', cost_type);
   close;

end
```

3.2.11. Error Del Espacio De Trabajo

```
function varargout = warn_def_work(varargin)
% WARN_DEF_WORK MATLAB code for warn_def_work.fig
%      WARN_DEF_WORK, by itself, creates a new WARN_DEF_WORK or raises
the existing
%      singleton*.
%
%      H = WARN_DEF_WORK returns the handle to a new WARN_DEF_WORK or
the handle to
%      the existing singleton*.
%
%      WARN_DEF_WORK('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in WARN_DEF_WORK.M with the given input
arguments.
%
%      WARN_DEF_WORK('Property','Value',...) creates a new WARN_DEF_WORK
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before warn_def_work_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
```

```
%      stop.  All inputs are passed to warn_def_work_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help warn_def_work

% Last Modified by GUIDE v2.5 12-Mar-2013 17:51:47

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warn_def_work_OpeningFcn, ...
                   'gui_OutputFcn',  @warn_def_work_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before warn_def_work is made visible.
function warn_def_work_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warn_def_work (see VARARGIN)
set(hObject,'WindowStyle','modal');
% Choose default command line output for warn_def_work
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warn_def_work wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
```

```
function varargout = warn_def_work_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

3.2.12. Error De Suficientes Puntos

```
function varargout = warn_mesage(varargin)
% WARN_MESAGE MATLAB code for warn_mesage.fig
%      WARN_MESAGE, by itself, creates a new WARN_MESAGE or raises the
existing
%      singleton*.
%
%      H = WARN_MESAGE returns the handle to a new WARN_MESAGE or the
handle to
%      the existing singleton*.
%
%      WARN_MESAGE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in WARN_MESAGE.M with the given input
arguments.
%
%      WARN_MESAGE('Property','Value',...) creates a new WARN_MESAGE or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before warn_mesage_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to warn_mesage_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help warn_mesage

% Last Modified by GUIDE v2.5 15-Jun-2017 05:01:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warn_mesage_OpeningFcn, ...
                   'gui_OutputFcn',  @warn_mesage_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before warn_mesage is made visible.
function warn_mesage_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warn_mesage (see VARARGIN)

% Choose default command line output for warn_mesage
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warn_mesage wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = warn_mesage_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

3.2.13. Error Del Numero Maximo De Objetos

```
function varargout = warn_mesage2(varargin)
% WARN_MESAGE2 MATLAB code for warn_mesage2.fig
%      WARN_MESAGE2, by itself, creates a new WARN_MESAGE2 or raises the
existing
%      singleton*.
%
%      H = WARN_MESAGE2 returns the handle to a new WARN_MESAGE2 or the
handle to
%      the existing singleton*.
%
%      WARN_MESAGE2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in WARN_MESAGE2.M with the given input
arguments.
%
%      WARN_MESAGE2('Property','Value',...) creates a new WARN_MESAGE2
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before warn_mesage2_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to warn_mesage2_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help warn_mesage2

% Last Modified by GUIDE v2.5 15-Jun-2017 05:02:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warn_mesage2_OpeningFcn, ...
```

```
                   'gui_OutputFcn',  @warn_mesage2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before warn_mesage2 is made visible.
function warn_mesage2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warn_mesage2 (see VARARGIN)

% Choose default command line output for warn_mesage2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warn_mesage2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = warn_mesage2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

3.2.14. Error De Coordenadas Fuera De Rango

```
function varargout = warning_depasire(varargin)
% WARNING_DEPASIRE MATLAB code for warning_depasire.fig
%      WARNING_DEPASIRE, by itself, creates a new WARNING_DEPASIRE or
raises the existing
%      singleton*.
%
%      H = WARNING_DEPASIRE returns the handle to a new WARNING_DEPASIRE
or the handle to
%      the existing singleton*.
%
%      WARNING_DEPASIRE('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in WARNING_DEPASIRE.M with the given
input arguments.
%
%      WARNING_DEPASIRE('Property','Value',...) creates a new
WARNING_DEPASIRE or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before warning_depasire_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to warning_depasire_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help warning_depasire

% Last Modified by GUIDE v2.5 15-Jun-2017 05:02:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warning_depasire_OpeningFcn, ...
                   'gui_OutputFcn',  @warning_depasire_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before warning_depasire is made visible.
function warning_depasire_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warning_depasire (see VARARGIN)

% Choose default command line output for warning_depasire
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warning_depasire wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = warning_depasire_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

3.2.15. Error De Puntos De Inicio/Final

```
function varargout = warning_start_stop(varargin)
% WARNING_START_STOP MATLAB code for warning_start_stop.fig
%      WARNING_START_STOP, by itself, creates a new WARNING_START_STOP
or raises the existing
```

```
%      singleton*.
%
%      H = WARNING_START_STOP returns the handle to a new
WARNING_START_STOP or the handle to
%      the existing singleton*.
%
%      WARNING_START_STOP('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in WARNING_START_STOP.M with the given
input arguments.
%
%      WARNING_START_STOP('Property','Value',...) creates a new
WARNING_START_STOP or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before warning_start_stop_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to warning_start_stop_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help warning_start_stop

% Last Modified by GUIDE v2.5 01-Jun-2013 18:23:32

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @warning_start_stop_OpeningFcn, ...
                   'gui_OutputFcn',  @warning_start_stop_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before warning_start_stop is made visible.
```

```
function warning_start_stop_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to warning_start_stop (see VARARGIN)

% Choose default command line output for warning_start_stop
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes warning_start_stop wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = warning_start_stop_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Close
```

4. ANALISIS 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.
- Tiempos de ejecución.
- Tiempos de ejecución parcial.
- Incremento de promedio de distancia como filtro y reductor de tiempos de ejecución
- Eficiencia dependiendo el tipo de Teselacion.

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

4.1. CANTIDAD DE OBSTACULOS

La cantidad de obstáculos extiende el tiempo de ejecución considerablemente y dificulta los trayectos dependiendo de la posición de los puntos 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 Centroide de ubicación distinta entre adyacentes(laterales) de las y de columnas a la recta. 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 línea recta hacia los puntos que más lo acercan al punto de llegada. Estas dos consideraciones las realiza de forma óptima, con la dificultad de lo engorroso del recorrido, requerirá más tiempo y ciclos en procesamiento.

4.3. TIEMPOS DE EJECUCION

Como se ha mencionado, los tiempos de procesamiento dependen de la maquina computacional. El algoritmo desarrollado se diseñó para acortar estos tiempos, por medio de la no repetición 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 está sujeto a la ubicación de los puntos de partida y llegada en cada caso.

4.4. TIEMPOS DE EJECUCION 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 teselacion es indispensable y también requiere una gran cantidad de ciclos. Además de ser el proceso que más requiere tiempo de ejecución.

4.5. INCREMENTO DE PROMEDIO DE DISTANCIA COMO FILTRO Y REDUCTOR DE TIEMPOS DE EJECUCION

Con el fin de obtener menores tiempos de procesamiento, en aquellos casos en que los recorridos son cortos, se implementó la utilización del promedio de distancias entre todos los puntos, como valor límite 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 incrementará por el producto del promedio, por un valor incremental en una unidad, en cada ciclo que no encuentre una ruta eficaz, hasta lograrlo.

4.6. EFICIENCIA DEPENDIENDO EL TIPO DE TESELACION

Con la finalidad de determinar la elección de ruta eficiente, se expondrá un ejemplo de ruta variando el tipo de Teselacion para corroborar las distancias seleccionas y el resultado gráficamente.

4.7. TABLAS DE DISTANCIA OPTIMA DE EJEMPLO DE RUTA

Las columnas Trapezoidal, Triangular, Polytopal y Rectangular son los tipos de teselaciones que usaremos para evaluar cuál de ellos nos dan un mejor resultado las dos con los mismos obstáculos y los mismos puntos de inicio y final.

La fila siguiente (Distancia) nos dará las diferencias entre ellas y de ahí podremos escoger cual será nuestra solución óptima.

A continuación, observaremos las imágenes de los obstáculos con los tipos de teselacion y luego las tablas de datos cambiando el tipo de búsqueda con los resultados obtenidos por el programa.

4.7.1. Teselacion Trapezoidal

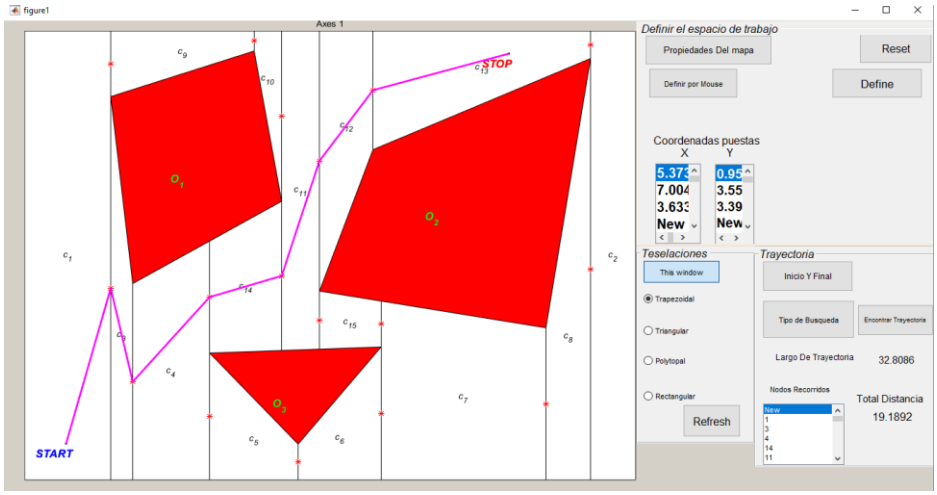


Figura 4.1. Búsqueda Por Teselacion Trapezoidal

Tabla 1: Tipo De Búsqueda Por Teselacion Trapezoidal

	Distancia Entre Los Centros
	Distancia Entre Centroide Y Punto Medio De Su Borde

4.7.2. Teselacion Triangular

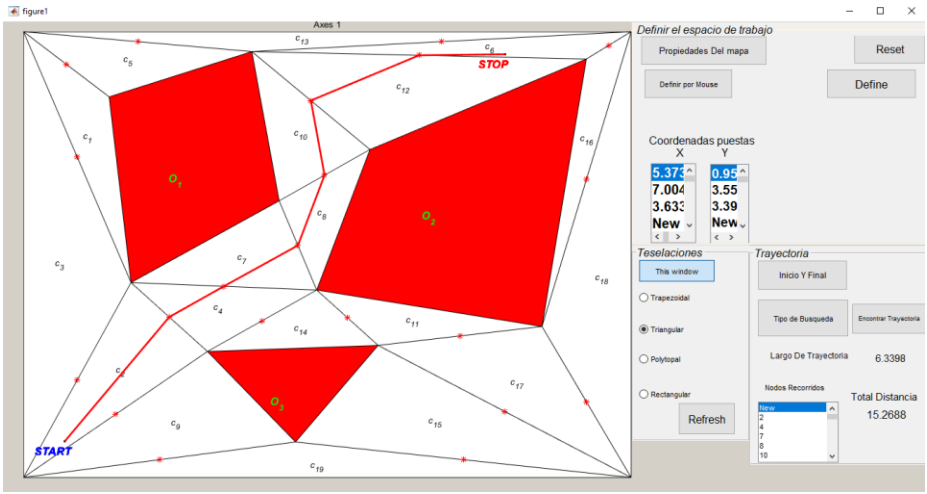


Figura 4.2. Búsqueda Por Teselacion Triangular

Tabla 2: Tipo De Búsqueda Por Teselacion Triangular

	Distancia Entre Los Centros
	Distancia Entre Centroide Y Punto Medio De Su Borde

4.7.3. Teselacion Polytopal

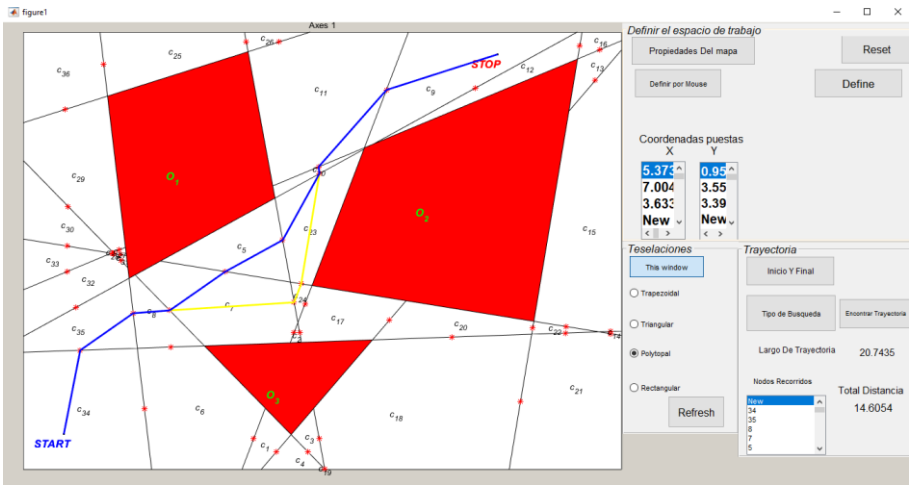


Figura 4.3. Búsqueda Por Teselacion Polytopal

Tabla 3: Tipo De Búsqueda Por Teselacion Polytopal

	Distancia Entre Los Centros
	Distancia Entre Centroide Y Punto Medio De Su Borde

4.7.4. Teselacion Rectangular

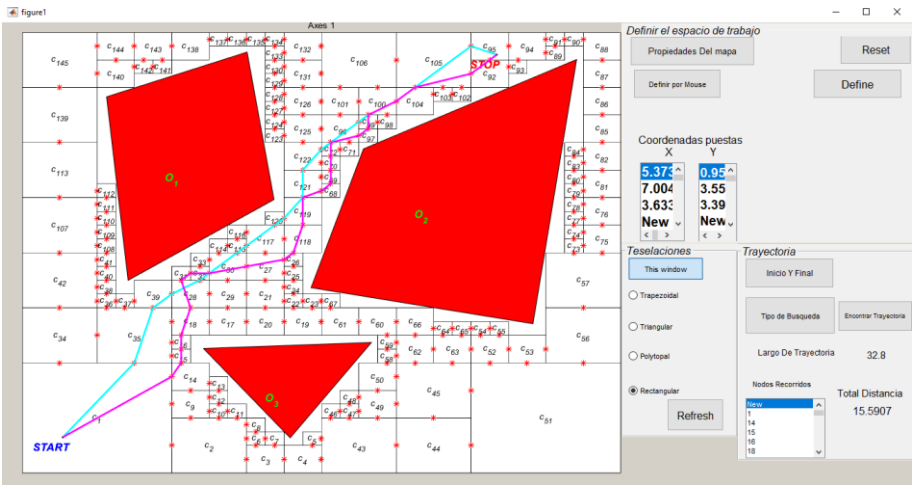


Figura 4.4. Búsqueda Por Teselacion Rectangular

Tabla 4: Tipo De Búsqueda Por Teselacion Rectangular

	Distancia Entre Los Centros
	Distancia Entre Centroide Y Punto Medio De Su Borde

TABLA 5: DISTANCIA ENTRE LOS CENTROS

Teselacion	Trapezoidal	Triangular	Polytopal	Rectangular
Distancia	19.18	15.26	15.73	14.41

TABLA 6: DISTANCIA ENTRE CENTROIDE Y PUNTO DE SU BORDE

Teselacion	Trapezoidal	Triangular	Polytopal	Rectangular
Distancia	19.18	15.26	14.60	15.59

5. CONCLUSIONES

- Se estudiaron dos métodos diferentes a fin de obtener la ruta partiendo de puntos, concluyendo que el método más eficiente, es el que usa la espiral de Fermat y líneas rectas, para las simulaciones realizadas se pudo comprobar esto, puesto que como resultado siempre genera menor ruta a recorrer.

- Las variables de error, en todas las simulaciones siempre tienen valores ≈0, garantizando un adecuado seguimiento de ruta.

- 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 incrementaran la optimización de la ruta.

- Aunque en los casos considerados se logra determinar una trayectoria desde el punto inicial al punto objetivo en algunas situaciones se observa la presencia de trayectorias con irregularidades y movimientos lineales, lo cual puede producir trayectorias no óptimas. Considerando lo anterior, en un trabajo futuro se puede desarrollar un algoritmo para suavizar las trayectorias irregulares y eliminar las trayectorias lineales.

- Con el análisis estadístico de los datos, se validan las observaciones efectuadas para los diferentes experimentos. Los parámetros que se modificaron para observar el comportamiento del algoritmo fueron seleccionados, considerando los análisis reportados del modelo y las simulaciones previas del mismo.

6. TRABAJO FUTURO

- Desarrollar el control aplicando otras técnicas para probar la eficiencia de la técnica backstepping y ver si se puede mejorar el tiempo de procesamiento.

- Revisar otras técnicas de planeamiento de movimiento, a fin de establecer la más eficiente para la aplicación.

- Investigar otras técnicas para generación de rutas, con el fin de reducir la carga computacional y obtener el mejor comportamiento dinámico del robot.

- Aplicar técnicas de identificación a fin de obtener los parámetros reales del robot autónomo a implementar.

- En trabajos futuros, se podría considerar otros modelos de partículas autopropulsadas y partículas brownianas, para implementar el algoritmo propuesto. También se puede aplicar el concepto presentado en este documento, para el desarrollo de algoritmos de optimización.

7. REFERENCIAS Y BIBLIOGRAFIA

[1] Alessandretti, A. P. Aguiar, and C. Jones. Trajectory-tracking and pathfollowing controllers for constrained underactuated vehicles using model predictive control. In 2013 European Control Conference (ECC), number EPFLCONF- 199227, pages 1371-1376. IEEE, 2013.

[2] M. Candeloro, A. Lekkas, A. Soerensen, and T. I. Fossen. Continuous curvature path planning using voronoi diagrams and fermat's spirals. In Control Applications in Marine Systems, volume 9, pages 132-137, 2013.

[3] BHAPTACHARIA,Subhrajit. Topological and geometric techniques in graph searchbased robot planning . Estado Unidos: Universidad de Pensilvania. 2012.

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

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

[6] 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.

[7] 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.

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

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

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

[11] 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.

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

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

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

[15] 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,.

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

[17] Gómez-Bravo F., Cuesta F., y Ollero Baturone Aníbal, (2003). Planificación de trayectorias en robots móviles basada en técnicas de control de sistemas no holónomos. XXIV Jornadas de Automática.

[18] Ollero Baturone Aníbal, (2001). Robótica Manipuladores y robots móviles. Marcombo.

[19] De Los Santos De La Rosa Erik Adolfo, (2004). Heurística para la generación de configuraciones en pasajes estrechos aplicada al problema de los clavos. Tesis de Maestría en Ciencias con Especialidad en Ingeniería en Sistemas Computacionales, Universidad de las Américas Puebla.

[20] Tu Jianping, and Yangt Simon, (2003). Genetic Algorithm Based Path Planning for a Mobile Robot. Proceedings of the IEEE International Conference on Robotics & Automation.

[21] Nilsson J. Nils, (1969). A Mobile Automaton: An Application of Artificial Intelligence Techiques. Proc. of the 1st. International Joint Conference on Artificial Intelligence.

[22] Brooks Rodney A., (1983). Solving the Find-Path Problem by Good Representation of Free Space. IEEE Transactions on Sysyems Man and Cybernetics.

[23] S. M. LaValle. Planning algorithms. Cambridge university press, 2006.

[24] M. Lekkas. Guidance and path-planning systems for autonomous vehicles. 2014.

[25] M. Lekkas and T. I. Fossen. Trajectory tracking and ocean current estimation for marine underactuated vehicles. In Control Applications (CCA), 2014 IEEE Conference on, pages 905-910. IEEE, 2014.

[26] M. Lekkas, T. Fossen, et al. Integral los path following for curved paths based on a monotone cubic hermite spline parametrization. Control Systems Technology, IEEE Transactions on, 22(6):2287-2301, 2014a.

[27] Rimon E., and Koditchek D.E., (1992). Exact robot navigation using artificial potential functions. IEEE Transactions on Robotics and Automation, Vol. 8 (5).

[28] Kim J.O., and Khosla P.K., (1991). Real-time obstacle avoidance using harmonic potential functions. IEEE Conference on Robotics and Automation, Sacramento, pp.790-796.

[29] Khatib O., (1986). Real-time obstacle avoidance for manipulators and mobile robots. In: International Journal of Robotic Research, Vol. 5 (1), 90p.

[30] Coelho Leandro dos Santos, and Sierakowski Cezar Augusto, (2006). Bacteria Colony Approaches With Variable Velocity Applied to Path Optimization of Mobile Robots. ABCM Symposium Series in Mechatronics, Vol. 2.

[31] 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.

[32] R. HUANG, A schedule-based pathnding algorithm for transit networks using patternrst search, . Geoinformatica, vol. 11, no. 2, pp. 269-285, 2007.

[33] 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.

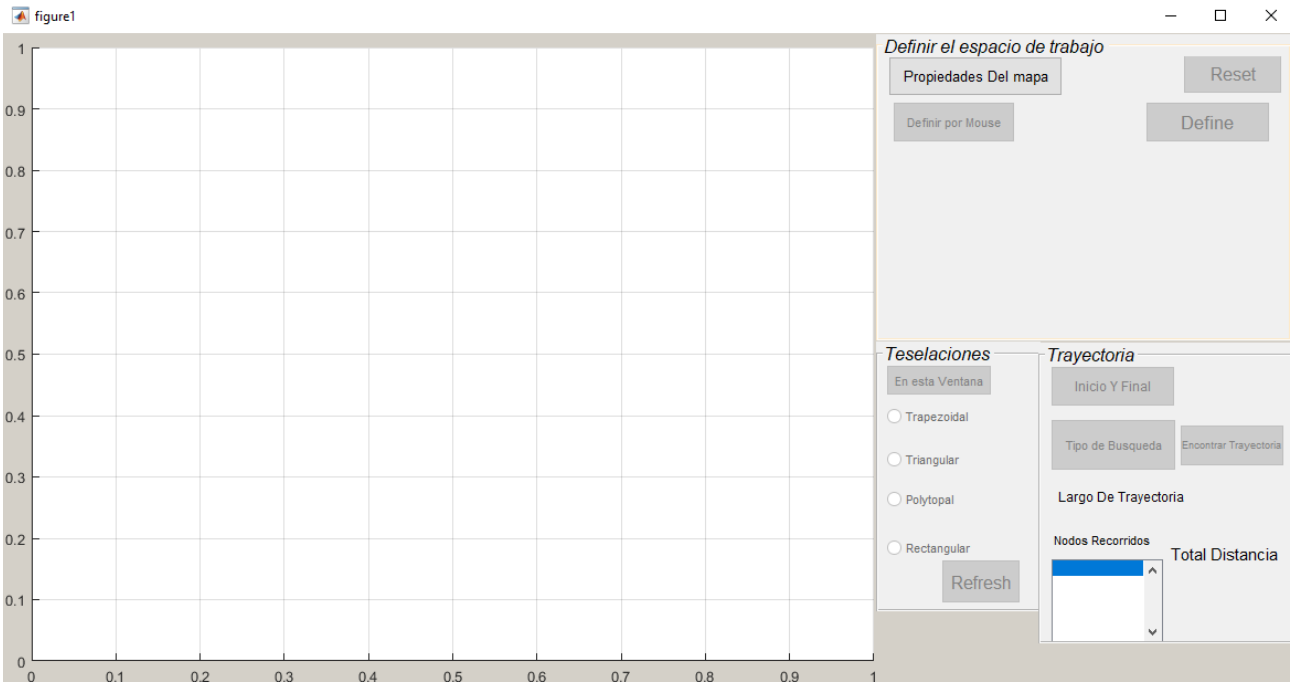
[34] GALLO, G. and PALLOTTINO, S., Shortest path methods: a unifying approach. Mathematical Programming Study., 26, pp. 38-64, 1986.

[35] 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,

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

8. ANEXOS

A continuación, se mostrará el algoritmo simulado en Matlab, que demostrara el funcionamiento del algoritmo, adjunto con una imagen del MENU que nos mostrará cuando se compile el código.




```
%Funcion para Que nos corra la imagen antes creada.
function varargout = figure1(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @figure1_OpeningFcn, ...
    'gui_OutputFcn',  @figure1_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
function create_workspace_CreateFcn(hObject, eventdata, handles)
% hObject    Nos sirve para crear nuestro espacio de trabajo
% eventdata  Se usa para verificar la version de matlab que se use

function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    Se encarga de la figura 1
% eventdata  Como anteriormente comentamos,sirve para verificar la
version de MatLab

function figure1_OpeningFcn(hObject, eventdata, handles, varargin)
%En esta funcion no tiene argumentos de salida

% varargin   Usamos los argumentos del comando de la figura 1


handles.cnt=0;

handles.output = hObject;
set(hObject,'WindowStyle','modal');
%Se actualiza la estructura de los manejadores
guidata(hObject, handles);

% UIWAIT Hace que la figura 1 espere la respuesta del usuario
uiwait(handles.figure1);

% --- Ejecucion del boton Take_Coord (Tomar Coordenadas).
function Take_Coord_Callback(hObject, eventdata, handles)

xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);

if( handles.obs_no==0)
    m2=warn_mesage2;
    set(handles.textX,'Visible','off');
    set(handles.textY,'Visible','off');
    set(handles.X_point,'Visible','off');
    set(handles.Y_point,'Visible','off');
    set(handles.Take_Coord,'Visible','off');
    set(handles.final,'Visible','off');
end


set(handles.list1,'Visible','on');
set(handles.list2,'Visible','on');
set(handles.afis_coord,'Visible','on');
ok1=0;
%Lea el punto X descrito por el usuario
handles.x_current=str2double(get(handles.X_point,'String'));
%Lea el punto Y descrito por el usuario
handles.y_current=str2double(get(handles.Y_point,'String'));
if(ok1==0)
```

```
    if(handles.x_current>xmax || handles.y_current>ymax ||
handles.x_current<0 || handles.y_current<0 )
        ok1=0;
        warning_depasire;
        waitfor(warning_depasire);
        handles.x_current=str2double(get(handles.X_point,'String'));
        %Lee el punto Y
        handles.y_current=str2double(get(handles.Y_point,'String'));
    else
        ok1=1;
        %%%----------Imprime las coordenadas previstas en la caja del
mapa-----------
        oldstr = get(handles.list1,'string'); %Se crea la cadena tal
como esta diseñada
        addstr = { get(handles.X_point,'String')}; %Ordenamos que la
cadena se añada a la fila
        if(strcmp(oldstr,' '))
            set(handles.list1,'str',{addstr{:} });
        else
            set(handles.list1,'str',{addstr{:},oldstr{:}});
        end


        %Añadimos el punto al mapa
        oldstr = get(handles.list2,'string');
        addstr = { get(handles.Y_point,'String')};
        if(strcmp(oldstr,' '))
            set(handles.list2,'str',{addstr{:} });
        else
            set(handles.list2,'str',{addstr{:},oldstr{:}});
        end
        %-------Creamos 2 vectores con puntos para el obstaculo actual -
---------------
        handles.x=[handles.x handles.x_current];
        handles.y=[handles.y handles.y_current];
        %-------El primer elemento sera siempre 0 y lo guardamos--------
--------
        if(handles.x(1)==0)
            for i=1:length(handles.x)-1
                handles.x(i)=handles.x(i+1);
            end

            for i=1:length(handles.y)-1
                handles.y(i)=handles.y(i+1);
            end
        end

    end
end

xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);
```

```
title('Axes 1');
axis([0,xmax,0,ymax]);
plot(handles.x_current,handles.y_current,'.k');

handles.output = hObject;
%Actualiza la estructura de los manejadores
guidata(hObject, handles);


function def_obj_Callback(hObject, eventdata, handles)

openfig('define_spacework.fig');

h=define_spacework;
waitfor(h);

handles.x_max = getappdata(0,'some_var1');
handles.y_max = getappdata(0,'some_var2');
handles.obs_no = getappdata(0,'some_var3');
handles.output = hObject;
handles.obs_no=str2double(handles.obs_no);
handles.var=handles.obs_no;

set(handles.togglebutton2,'Enable','on');

%Generamos la opcion de retorno para cuando el usuario no ha definido el
%espacio de trabajo correctamente
%  return
set(handles.create_workspace,'Enable','on');
set(handles.Reset,'Enable','on');

guidata(hObject, handles);

% --- Creacion del boton "Puntos"
function hints_Callback(hObject, eventdata, handles)

ind=hint;


% --- Creamos el boton "Crear espacio de trabajo".\\\\\\\\\\\
function create_workspace_Callback(hObject, eventdata, handles)
cla(handles.axes1);
getappdata(0,'some_var4');
if(strcmp(getappdata(0,'some_var4'),'start' ))
    handles.h1=0;
    hold on
    grid(handles.axes1,'minor');
    xmax=str2double(handles.x_max);
    ymax=str2double(handles.y_max);
    obs_no= handles.obs_no;
    title('Axes 1');
```

```
if((get(handles.togglebutton2,'Value') == 1))
    %nou
    set(handles.textX,'Visible','on');
    set(handles.textY,'Visible','on');
    set(handles.X_point,'Visible','on');
    set(handles.Y_point,'Visible','on');
    set(handles.Take_Coord,'Visible','on');
    set(handles.final,'Visible','on');
    handles.Start_keyb=1;
    set(handles.textX,'Enable','on');
    set(handles.textY,'Enable','on');
    set(handles.X_point,'Enable','on');
    set(handles.Y_point,'Enable','on');
    set(handles.Take_Coord,'Enable','on');
    set(handles.final,'Enable','on');
    set(handles. togglebutton3,'Enable','off');
else
    if((xmax~=0) && (ymax~=0))

        axis([0,xmax,0,ymax]);
        grid(handles.axes1,'minor');
        %Leemos los bordes de los obstaculos
        for i=1:obs_no

            j=1; %j = N° de vertices para el objeto actual
            but=1;
            a=zeros(1,3);

            set(handles.textX,'Visible','off');
            set(handles.textY,'Visible','off');
            set(handles.X_point,'Visible','off');
            set(handles.Y_point,'Visible','off');
            set(handles.Take_Coord,'Visible','off');
            set(handles.final,'Visible','off');

            while  but==1

                ok=0;
                while ok==0
                    [x,y,but]=ginput(1);
                    ok=1;
                    if(x>xmax || y>ymax || x<0 || y<0)
                        ok=0;
                    end
                end

                set(handles.list1,'Visible','on');
                set(handles.list2,'Visible','on');
                set(handles.afis_coord,'Visible','on');
                oldstr1 = get(handles.list1,'String');
                addstr1 = { num2str(x)};
                if(strcmp(oldstr1,' '))
```

```
                        set(handles.list1,'str',{addstr1{:}  });
                    else

set(handles.list1,'str',{addstr1{:},oldstr1{:}});
                    end

                    oldstr2 = get(handles.list2,'String');
                    addstr2 = { num2str(y)};
                    if(strcmp(oldstr2,' '))
                        set(handles.list2,'str',{addstr2{:} });
                    else

set(handles.list2,'str',{addstr2{:},oldstr2{:}});
                    end
                    plot(x,y,'.k');
                    objects1{i}(:,j)=[x;y];
                    j=j+1;

                end

                k=convhull(objects1{i}(1,:),objects1{i}(2,:));
                objects1{i}=objects1{i}(:,k(1:length(k)-1));

                pause(0.3)

fill(objects1{i}(1,:),objects1{i}(2,:),'k','FaceAlpha',0.5);
%
                oldstr = get(handles.list1,'String');
                addstr = { 'New' };
                oldstr2 = get(handles.list2,'String');
                addstr2 = { 'New'};
                if(i~=obs_no )
                    set(handles.list1,'str',{addstr{:},oldstr{:}});
                    set(handles.list2,'str',{addstr2{:},oldstr2{:}});
                end
            end
        end

        handles.obj=objects1;
        set(handles.textX,'Enable','off');
        set(handles.textY,'Enable','off');
        set(handles.X_point,'Enable','off');
        set(handles.Y_point,'Enable','off');
        set(handles.Take_Coord,'Enable','off');
        set(handles.final,'Enable','off');

    end
else
    f = openfig('warn_def_work.fig');

end
if(get(handles.togglebutton2,'Value') ~= 1)
    set(handles. togglebutton3,'Enable','on');
```

```
end
handles.output = hObject;

guidata(hObject, handles);

% ---Se ejecuta durante la creación del objeto, después de establecer
todas las propiedades.
function axes1_CreateFcn(hObject, eventdata, handles)

% ---Se ejecuta durante la eliminación de objetos, antes de destruir
propiedades
function figure1_DeleteFcn(hObject, eventdata, handles)

close all


% --- Se ejecuta durante la eliminación de objetos, despues de destruir
propiedades.
function hints_DeleteFcn(hObject, eventdata, handles)

function X_point_Callback(hObject, eventdata, handles)

function X_point_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Y_point_Callback(hObject, eventdata, handles)

function Y_point_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function final_Callback(hObject, eventdata, handles)

handles.fin=1;

set(handles.list1, 'String', ' ');
set(handles.list2, 'String', ' ');

if( handles.obs_no~=0)
    if(( (length(handles.x))>=3 && (length(handles.y))>=3))
```

```
        objects2 =[handles.x ; handles.y];

        plot(handles.x , handles.y ,'.k');
        try
            k=convhull(objects2 (1,:),objects2 (2,:));
            objects2 = objects2 (:,k(1:length(k)-1));
            fill(objects2 (1,:),objects2(2,:),'k','FaceAlpha',0.5);
            handles.obs_no=handles.obs_no-1;

            handles.x=[];
            handles.y=[];
        catch


        end
    else
        m1=warn_mesage;
    end
else
    m2=warn_mesage2;
end

handles.obj{handles.var-handles.obs_no} =objects2;
if(handles.obs_no==0)
set(handles.togglebutton3,'Enable','on');
end
handles.output = hObject;

guidata(hObject, handles);




function list1_Callback(hObject, eventdata, handles)

function list1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function list2_Callback(hObject, eventdata, handles)

function list2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


function uipanel1_CreateFcn(hObject, eventdata, handles)

%Establecemos la forma de teselacion tipo (Trapezoidal)
function Trapezoidal_Callback(hObject, eventdata, handles)

[C,adj,mid_X,mid_Y]=trapezoidal_decomposition(handles.obj,str2double(han
dles.x_max),str2double(handles.y_max));

set(handles.refresh_dec,'Enable','on');
%Borra el eje establecido
cla;

modify_cost(C,adj,mid_X,mid_Y,1);

modify_cost(C,adj,mid_X,mid_Y,2);

modify_cost(C,adj,mid_X,mid_Y,3);

hold on;
xmax=str2double(handles.x_max);

set(handles.Triangular,'Value',0);
set(handles.Rectangular,'Value',0);
set(handles.Polytopal,'Value',0);

handles.C=C;
handles.adj=adj;
handles.mid_X=mid_X;
handles.mid_Y=mid_Y;

if(get(handles.togglebutton3,'Value') ~= 1)


plot_environment_simplu(handles.obj,str2double(handles.x_max),str2double
(handles.y_max),C,adj,mid_X,mid_Y);
    handles.current_axes=gca;

else


plot_environment(handles.axes1,handles.obj,str2double(handles.x_max),str
2double(handles.y_max),C,adj,mid_X,mid_Y);
    handles.current_axes=gca;
end

set(handles.start_stop,'Enable','on');
handles.flag_dec='Trapezoidal';
```

```
handles.output = hObject;

guidata(hObject, handles);


%Establecemos la forma de teselacion tipo (Triangular)
function Triangular_Callback(hObject, eventdata, handles)
handles.flag_dec='Triangular';
[C,adj,mid_X,mid_Y]=triangular_decomposition(handles.obj,str2double(hand
les.x_max),str2double(handles.y_max));
%Borra el eje establecido
cla

set(handles.refresh_dec,'Enable','on');

if(get(handles.togglebutton3,'Value') ~= 1)


plot_environment_simplu(handles.obj,str2double(handles.x_max),str2double
(handles.y_max),C,adj,mid_X,mid_Y);
    handles.current_axes=gca;

else

plot_environment(handles.axes1,handles.obj,str2double(handles.x_max),str
2double(handles.y_max),C,adj,mid_X,mid_Y);
     handles.current_axes=gca;
end


handles.C=C;
handles.adj=adj;
handles.mid_X=mid_X;
handles.mid_Y=mid_Y;

set(handles.Rectangular,'Value',0);
set(handles.Polytopal,'Value',0);
set(handles.Trapezoidal,'Value',0);
set(handles.start_stop,'Enable','on');

handles.output = hObject;

guidata(hObject, handles);


%Establecemos la forma de teselacion tipo (Rectangular)
function Rectangular_Callback(hObject, eventdata, handles)
handles.flag_dec='Rectangular';
[C,adj,mid_X,mid_Y]=rectangular_decomposition(handles.obj,str2double(han
dles.x_max),str2double(handles.y_max),50);
%Borra el eje establecido
```

```
cla
handles.C=C;
handles.adj=adj;
handles.mid_X=mid_X;
handles.mid_Y=mid_Y;



set(handles.refresh_dec,'Enable','on');
hold on;

if(get(handles.togglebutton3,'Value') ~= 1)


plot_environment_simplu(handles.obj,str2double(handles.x_max),str2double
(handles.y_max),C,adj,mid_X,mid_Y);
    handles.current_axes=gca;

else

plot_environment(handles.axes1,handles.obj,str2double(handles.x_max),str
2double(handles.y_max),C,adj,mid_X,mid_Y);
      handles.current_axes=gca;
end



set(handles.Polytopal,'Value',0);
set(handles.Trapezoidal,'Value',0);
set(handles.Triangular,'Value',0);
set(handles.start_stop,'Enable','on');
handles.output = hObject;

guidata(hObject, handles);

%Establecemos la forma de teselacion tipo (Polytopal)
function Polytopal_Callback(hObject, eventdata, handles)
handles.flag_dec='Polytopal';
[C,adj,mid_X,mid_Y]=polytopal_decomposition(handles.obj,str2double(handl
es.x_max),str2double(handles.y_max));
%Borra el eje establecido
handles.C=C;
handles.adj=adj;
handles.mid_X=mid_X;
handles.mid_Y=mid_Y;
cla;

set(handles.refresh_dec,'Enable','on');

if(get(handles.togglebutton3,'Value') ~= 1)
```

```
plot_environment_simplu(handles.obj,str2double(handles.x_max),str2double
(handles.y_max),C,adj,mid_X,mid_Y);
    handles.current_axes=gca;

else

plot_environment(handles.axes1,handles.obj,str2double(handles.x_max),str
2double(handles.y_max),C,adj,mid_X,mid_Y);
      handles.current_axes=gca;
end

set(handles.start_stop,'Enable','on');
set(handles.Rectangular,'Value',0);
set(handles.Trapezoidal,'Value',0);
set(handles.Triangular,'Value',0);
handles.output = hObject;

guidata(hObject, handles);



function figure1_OutputFcn(hObject, eventdata, handles)



% --- Ejecuta el boton this_windm (En esta Ventana).
function this_wind_Callback(hObject, eventdata, handles)

if (get(hObject,'Value') == get(hObject,'Max'))

    set(handles.other_wind,'Enable','off');
else
    set(handles.other_wind,'Enable','on');
end
set(handles.Trapezoidal,'Enable','on');
set(handles.Triangular,'Enable','on');
set(handles.Polytopal,'Enable','on');
set(handles.Rectangular,'Enable','on');



% --- Ejecuta el boton other_wind (En otra Ventana).
function other_wind_Callback(hObject, eventdata, handles)

if (get(hObject,'Value') == get(hObject,'Max'))
    set(handles.this_wind,'Enable','off');
else
    set(handles.this_wind,'Enable','on');
end
set(handles.Trapezoidal,'Enable','on');
```

```
set(handles.Triangular,'Enable','on');
set(handles.Polytopal,'Enable','on');
set(handles.Rectangular,'Enable','on');


% --- Ejecuta el boton Reset.
function Reset_Callback(hObject, eventdata, handles)

  cla(handles.axes1) ;

 hold on;
xmax=str2double(handles.x_max)
ymax=str2double(handles.y_max)
axes(handles.axes1);
axis([0,xmax,0,ymax]);
grid on;

function test_limita(hObject, eventdata, handles)
xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);

handles.ok=0;
start_stop;
waitfor(start_stop);

while(handles.ok==0)
    try
        getappdata(0,'xstart');
        getappdata(0,'ystart');
        getappdata(0,'xstop');
        getappdata(0,'ystop');
        handles.start(1) = str2double(getappdata(0,'xstart'));
        rmappdata(0,'xstart');


        handles.start(2) = str2double(getappdata(0,'ystart'));
        rmappdata(0,'ystart');


        handles.stop(1) = str2double(getappdata(0,'xstop'));
        rmappdata(0,'xstop');


        handles.stop(2) = str2double(getappdata(0,'ystop'));

        rmappdata(0,'ystop');
    end
    if(handles.start(1)>xmax || handles.start(2)>ymax || handles.stop(1)
>xmax || handles.stop(2)>ymax)

        handles.ok=0;
```

```
        warning_depasire;
        waitfor(warning_depasire);
    else
        handles.ok=1;
        return;
    end

    start_stop;
    waitfor(start_stop);


end
handles.output = hObject;

guidata(hObject, handles);

function start_stop_Callback(hObject, eventdata, handles)

%El programa está tratando de evitar la posibilidad de inserción de más
%puntos de Inicio y Final

ask_start_stop1;
waitfor(ask_start_stop1);
xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);

d=getappdata(0,'some_variable1');

try
    rmappdata(0,'some_variable1');

    if(strcmp(d,'mouse')==1)

        if(get(handles.togglebutton3,'Value') == 1)
            disp('figure1');
            axes(handles.axes1);
            [x,y ]=ginput(1);
            plot( handles.axes1,x,y,'.k');
            handles.start=[x y];
            text(x-((2/100)*xmax ),y -((2/100)*ymax
),'START','HorizontalAlignment','center','Color','b','FontSize',12,'Font
Weight','bold','FontAngle','italic','FontName','TimesNewRoman');
            [x,y ]=ginput(1);
            plot( handles.axes1,x,y,'.k');
            handles.stop=[x y];
            text(x+((2/100)*xmax ),y +((2/100)*ymax
),'STOP','HorizontalAlignment','center','Color','r','FontSize',12,'FontW
eight','bold','FontAngle','italic','FontName','TimesNewRoman');

        else

            axes(handles.current_axes)
```

```
            [x,y ]=ginput(1);
            plot(  x,y,'.k');
            handles.start=[x y];
            text(x-((2/100)*xmax ),y -((2/100)*ymax
),'START','HorizontalAlignment','center','Color','b','FontSize',12,'Font
Weight','bold','FontAngle','italic','FontName','TimesNewRoman');
            [x,y ]=ginput(1);
            plot(  x,y,'.k');
            handles.stop=[x y];
            text(x+((2/100)*xmax ),y +((2/100)*ymax
),'STOP','HorizontalAlignment','center','Color','r','FontSize',12,'FontW
eight','bold','FontAngle','italic','FontName','TimesNewRoman');

        end

    else

        if(strcmp(d,'keyboard')==1)

            handles.ok=0;
            start_stop;
            waitfor(start_stop);

            while(handles.ok==0)
                try
                    getappdata(0,'xstart');
                    getappdata(0,'ystart');
                    getappdata(0,'xstop');
                    getappdata(0,'ystop');
                    handles.start(1) =
str2double(getappdata(0,'xstart'));
                    rmappdata(0,'xstart');

                    handles.start(2) =
str2double(getappdata(0,'ystart'));
                    rmappdata(0,'ystart');

                    handles.stop(1) = str2double(getappdata(0,'xstop'));
                    rmappdata(0,'xstop');

                    handles.stop(2) = str2double(getappdata(0,'ystop'));

                    rmappdata(0,'ystop');
                end
                if(handles.start(1)>xmax || handles.start(2)>ymax ||
handles.stop(1) >xmax || handles.stop(2)>ymax)

                    handles.ok=0;
                    warning_depasire;
                    waitfor(warning_depasire);
                else
                    handles.ok=1;
```

```
                    break;
                end

                start_stop;
                waitfor(start_stop);

            end
            axes(handles.current_axes);

            plot(  handles.start(1),handles.start(2),'.k');
            text( handles.start(1)-((2/100)*xmax ),handles.start(2) -
((2/100)*ymax
),'START','HorizontalAlignment','center','Color','b','FontSize',12,'Font
Weight','bold','FontAngle','italic','FontName','TimesNewRoman');

            plot(  handles.stop(1),handles.stop(2),'.k');
            text( handles.stop(1)-((2/100)*xmax ),handles.stop(2) -
((2/100)*ymax
),'STOP','HorizontalAlignment','center','Color','r','FontSize',12,'FontW
eight','bold','FontAngle','italic','FontName','TimesNewRoman');

        end
    end


    set(handles.create_workspace,'Enable','on');

    for i=1:handles.obs_no

IN1=inpolygon(handles.start(1),handles.start(2),handles.obj{i}(1,:),hand
les.obj{i}(2,:));
        if(sum(IN1)~=0)

            warning_start_stop;
            waitfor( warning_start_stop);
            refresh_dec_Callback(hObject, eventdata, handles);

            break;
        end

IN2=inpolygon(handles.stop(1),handles.stop(2),handles.obj{i}(1,:),handle
s.obj{i}(2,:));
        if(sum(IN2)~=0)
            warning_start_stop;
            waitfor( warning_start_stop);
            refresh_dec_Callback(hObject, eventdata, handles);

            break;
        end
    end
end
d=0;
```

```
set(handles.type_of_cost,'Enable','on');
handles.output = hObject;

guidata(hObject, handles);



function refresh_Callback(hObject, eventdata, handles)

function refresh_dec_Callback(hObject, eventdata, handles)

cla;
xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);
axis([0,xmax,0,ymax]);
grid(handles.axes1,'minor');

for i=1:length(cellfun(@length,handles.obj))
    fill(handles.obj{i} (1,:),handles.obj{i}(2,:),'k','FaceAlpha',0.5);
end


%Funcion de Encontrar el Trayecto
function find_traj_Callback(hObject, eventdata, handles)
set(handles.find_traj,'Enable','off');

axes(handles.current_axes);

 persistent CNT

if isempty(CNT)
   CNT = 1;
else
    CNT = CNT + 1;
end
if CNT==7
    CNT=1;
end
colors=['b','g','r','c','m','y','k'];
l_type=['-','--'];

xmax=str2double(handles.x_max);
ymax=str2double(handles.y_max);

axes(handles.current_axes);
plot(handles.axes1,handles.start(1),handles.start(2),'.k');
text(handles.start(1)-((2/100)*xmax ),handles.start(2) -((2/100)*ymax
),'START','HorizontalAlignment','center','Color','b','FontSize',12,'Font
Weight','bold','FontAngle','italic','FontName','TimesNewRoman');

plot(handles.axes1,handles.stop(1),handles.stop(2),'.k');
```

```
text(handles.stop(1)-((2/100)*xmax ),handles.stop(2) -((2/100)*ymax
),'STOP','HorizontalAlignment','center','Color','r','FontSize',12,'FontW
eight','bold','FontAngle','italic','FontName','TimesNewRoman');



handles.adj =
modify_cost(handles.C,handles.adj,handles.mid_X,handles.mid_Y,handles.co
st_type);

[trajectory, distance, path,
cost_path]=find_trajectory(handles.C,handles.adj,handles.mid_X,handles.m
id_Y,handles.start(1),handles.start(2),handles.stop(1),handles.stop(2)
);

set(handles.traj_len,'String',num2str(cost_path));

set(handles.dist,'String',num2str(distance));

for i= length(path):-1:1
    oldstr1 = get(handles.cells,'String');
    addstr1 = { num2str(path(i))};
    if(strcmp(oldstr1,' '))
        set(handles.cells,'str',{addstr1{:}  });
    else
        set(handles.cells,'str',{addstr1{:},oldstr1{:}});
    end
end

oldstr = get(handles.cells,'String');
addstr = { 'New' };
set(handles.cells,'str',{addstr{:},oldstr{:}});

for i=2:length(trajectory)

    line( [trajectory(1,i-1) trajectory(1,i )],[trajectory(2,i-1)
trajectory(2,i)], 'Color',colors(CNT),'LineWidth',2,
'LineStyle',l_type(mod(CNT,2)+1) );
    pause(0.4);
end
set(handles.find_traj,'Enable','on');


function togglebutton2_Callback(hObject, eventdata, handles)


if(get(hObject,'Value')==1)
    set( handles.togglebutton2,'String','Keyboard definition');
    set(handles.hints,'Enable','on');
    set(handles.hints,'Visible','on');
    handles.x=0;
```

```
    handles.y=0;

else
    set(handles.hints,'Enable','off');

    handles.x=0;
    handles.y=0;
    set( handles.togglebutton2,'String','Mouse definition');
end

handles.output = hObject;

guidata(hObject, handles);

% --- Ejecuta el boton togglebutton3.
function togglebutton3_Callback(hObject, eventdata, handles)


if(get(hObject,'Value')==1)
    set( handles.togglebutton3,'String','This window');
    set(handles.refresh_dec,'Visible','on');
else

    set( handles.togglebutton3,'String','In figures');
    set(handles.refresh_dec,'Visible','off');
end

set(handles.Trapezoidal,'Enable','on');
set(handles.Triangular,'Enable','on');
set(handles.Polytopal,'Enable','on');
set(handles.Rectangular,'Enable','on');


% --- Ejecuta el boton Tipo de Busqueda.
function type_of_cost_Callback(hObject, eventdata, handles)

h=type_of_cost;
waitfor(h);

c= getappdata(0,'cost_type');
handles.cost_type=str2double(c);
set(handles.find_traj,'Enable','on');
handles.flag_dec=1;
handles.output = hObject;

guidata(hObject, handles);


% --- Ejecuta el boton cambio de puntos.
function cells_Callback(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% ---Se ejecuta cuando la figura 1 se redimensiona.
function figure1_ResizeFcn(hObject, eventdata, handles)
```