GENERACIÓN DE TRAYECTOS EN UN ENTORNO DINÁMICO USANDO UNA PLATAFORMA ROBÓTICA DIFERENCIAL.

BRAYAN RICARDO GUEVARA RONCANCIO

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ D. C.
2017

GENERACIÓN DE TRAYECTOS EN UN ENTORNO DINÁMICO USANDO UNA PLATAFORMA ROBÓTICA DIFERENCIAL.

BRAYAN RICARDO GUEVARA RONCANCIO

TRABAJO DE GRADO PRESENTADO PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

DIRECTOR
ANDRÉS CAMILO JIMÉNEZ ALVAREZ
MsC. INGENIERIA ELECTRÓNICA

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE ELECTRÓNICA
BOGOTÁ D. C.
2017

Nota de Aceptación

_____

_____

_____

_____

_____

_____

Firma del Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

BOGOTÁ D.C. MARZO 2017

Las directivas de la Universidad de América, los jurados calificadores y el cuerpo docente no son responsables por los criterios e ideas expuestas en el presente documento. Estos corresponden únicamente a los autores

DEDICATORIA

A MI FAMILIA

AGRADECIMIENTOS

Deseo expresar mi agradecimiento especial en primer lugar, al Ingeniero Andrés Camilo Jiménez Álvarez, por el acompañamiento, apoyo e interés durante todo el proceso que conllevo la culminación de este documento, aportando su conocimiento y experiencia al proyecto.

A la Ingeniera Marcela Ríos Romero, coordinadora de laboratorios de ingeniería de la Fundación Universitaria Los Libertadores, por la atención, la cooperación, y la eficaz gestión para el préstamo de equipos, salas o talleres.

CONTENIDO

Pág.

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

Pág.

ANEXOS

Pág.

INTRODUCCIÓN

El problema de planificación se define en general, como el movimiento que debe llevar un cuerpo, desde un punto de configuración inicial hasta un punto final, dentro de un espacio de configuración libre de colisiones. Esto ha sido abordado en la literatura, por Jean – Claude Latombe con el libro “Robot Motion Planning” [1] , quien es uno de los primeros en abarcar el tema en particular, dando a conocer que existen muchos métodos efectivos que dan solución al problema de planificación en tiempo real, tales como campos de potencial, grafos de visibilidad, diagramas de Voronoi y métodos probabilísticos. Pero, cuanto más complejo es el espacio de configuraciones, mayores son las limitaciones en los métodos y mayor el tiempo necesario para encontrar la solución[2].

El método de planificación de campos de potencial, siendo uno de los métodos más rápidos, presenta el problema de la existencia de mínimos locales, interrumpiendo la generación de la trayectoria. Para dar solución a esta limitación se recurre a algoritmos de generación aleatoria (métodos probabilísticos) que permiten realizar planificaciones locales hacia un nuevo mínimo.

Este proyecto introduce los métodos de planificación más relevantes, y hace una comparación destacando al método por generación aleatoria RRT por sus siglas en inglés “Rapidly Exploring Random Tree” [3]. De este modo, se describe el algoritmo y se implementa mostrando una descripción grafica de su forma de operar en un entorno con obstáculos. En concreto, el método propuesto se aplica al caso de una plataforma robótica diferencial.

Este documento se presenta en el siguiente orden: primero, se hace una introducción a los conceptos más relevantes que se encontraran a lo largo del documento, segundo, se investigan y se dan a conocer los métodos de planificación más significativos, tercero, se hace una comparación de estos identificando las ventajas y desventajas de cada uno, demostrando que el método por generación aleatoria cumple con mayores ventajas a los demás, cuarto se describe el algoritmo RRT, la forma como se implementó y la solución que se le dio a los errores que presentó, teniendo como resultado una trayectoria que cumple con el objetivo en particular, pero esta, no es una trayectoria suave que pueda seguir la plataforma robótica diferencial, para cumplir esto, en quinto lugar, se presenta una propuesta para el suavizado de la trayectoria, y se implementa teniendo una trayectoria totalmente limpia, es decir, una trayectoria continua, sin cambios bruscos; en sexto y séptimo lugar se hace una presentación de manera formal, sobre la navegación en robots móviles y el modelamiento cinemático y dinámico del Robot. El proyecto finaliza con las conclusiones y como anexo, el código usado para la implementación del algoritmo y las herramientas usadas como metodología para lograrlo.

OBJETIVOS

OBJETIVO GENERAL:

Diseñar un algoritmo de navegación basado en un método de planificación para la generación de trayectorias de una plataforma robótica en un entorno dinámico.

OBJETIVOS ESPECIFICOS:

1. Investigar los métodos de planificación roadmap, descomposición en celdas, campos de potencial artificial y métodos probabilísticos.

2. Comparar los métodos: Roadmap, descomposición de celdas, campos de potencia artificial y RRT, para la planificación en robots móviles.

3. Seleccionar y diseñar el algoritmo de planeación que mejor se ajuste en entornos estáticos.

4. Validar el algoritmo de navegación en Python.

5. Realizar el modelamiento dinámico y cinemático del agente robótico diferencial.

6. Modelar y evaluar el comportamiento del robot en un entorno dinámico evadiendo obstáculos estimando y reduciendo los errores por odométria y fallas en el terreno

PLANTEAMIENTO DEL PROBLEMA

La planificación de trayectorias es uno de los mayores problemas que debe enfrentar un robot autónomo móvil cuando está en operación, debido al constante cambio que puede tener el terreno, como pasar de una superficie totalmente lisa a una superficie rugosa, desviándolo cada vez más del objetivo o meta a cumplir, ya sea por fricción en las llantas o cambios inesperados en el entorno, como obstáculos, y la mala lectura de los sensores.

En este proyecto se usa un robot móvil de tipo diferencial, siendo esto una gran ventaja, ya que el problema de planificación se reduce por las características de este modelo de robot, pero aun así, sigue siendo un reto, además el problema específico para este proyecto es generar la trayectoria desde un punto inicial cualquiera a un punto final cualquiera en un entorno inicialmente conocido, el cual puede ser diseñado o implementado partiendo de un mapa real. Pero, ese entorno solo será un mapa ideal, ya que al interactuar el robot en el entorno en tiempo real, este puede encontrarse con obstáculos que no se contemplan en el mapa inicial.

JUSTIFICACIÓN

El problema de planificación de movimientos en robots móviles consiste en determinar una trayectoria admisible o conjunto de movimientos que permitan llevarlo desde una configuración inicial a una final dentro del espacio de configuraciones libres de colisión. Teniendo en cuenta que la trayectoria debe cumplir con las restricciones cinemáticas de los robots móviles.

La importancia que conlleva resolver este problema, se centra en la necesidad de proponer un método que genere una trayectoria para el desplazamiento de cualquier robot móvil de tracción diferencial, el cual trabaje en un ambiente con objetos en el suelo como obstáculos, personas haciendo distintas labores que se encuentren en constante movimiento o para resolver tareas autónomas en ambientes peligrosos, por ejemplo, un robot realizando trabajos de carga en un laboratorio químico, en donde debe evitar tanto obstáculos estáticos, como personas que muy probablemente interrumpirán su trayecto mientras realizan sus deberes.

El sentido que tiene este proyecto es solucionar el problema de planificación para un robot diferencial, el cual pueda ser usado para otros fines, como realizar tomas de datos de temperatura, humedad, tomar muestras para laboratorios científicos, realizar trabajos de transporte de cargas, realizar trabajos en entornos que son de difícil alcance para las personas, en fin, que puedan ser usados incluso para realizar una acción en entornos peligrosos, para no arriesgar la vida de una persona.

ESTADO DEL ARTE

En la década de los ochenta y noventa la popularidad de los robots y sus usos en la industria [4] hicieron que los investigadores y programadores se dedicaran a resolver el problema de concebir trayectorias libres de colisiones entre los robots y su entorno, una de las metas de la robótica es desarrollar sistemas autónomos, y lo que para el hombre parece fácil e intuitivo, desplazarse de un sitio a otro o manipular objetos, no lo es para las maquinas.
Una primera aproximación a los métodos de planificación, fue realizada por Lozano-Pérez [5] y una década después se dio la aparición del libro de Jean – Claude Latombe [1] el cual genero enorme interés con numerosas publicaciones que lo sitúan como tema de actualidad en el campo de la planificación de trayectorias.

El acercamiento en el procesamiento computacional ha permitido que estos métodos de planificación de caminos hagan parte de aplicaciones innovadoras como es el caso de la cirugía robotizada, la animación en 3D, el uso de robots asistentes en museos y hospitales, en la química realizando el modelamiento de cadenas moleculares y en la exploración de otros mundos empleando robots móviles.

Existen varios métodos de origen geométrico para dar solución al problema básico de planeación de trayectorias y desde Latombe 1991 se han destacado 3 métodos: campos de potenciales artificiales, Roadmaps (mapas de carreteras) y descomposición en celdas. En la última década se ha desarrollado una nueva metodología llamada mapas probabilísticos, debido a que en entornos complejos, con restricciones cinemáticas, los métodos geométricos pueden presentar limitaciones en cuanto al tiempo de cómputo necesario y el éxito en la generación de trayectorias.

Con objeto de dar a conocer los métodos dentro de un marco general y lograr una visión más intuitiva de los sistemas de planificación se describen los siguientes:

Métodos “Roadmap”: son aquellos que construyen una descripción del espacio de configuraciones libres de colisión en forma de una red de curvas unidimensionales, que formaran parte del espacio de soluciones. Entre estos métodos están los grafos de visibilidad y los diagramas de Voronoi[6].

Métodos de descomposición por celdas: se basan en la descomposición del espacio de configuraciones libres de colisión en regiones convexas denominadas celdas, cada una de estas celdas se representa por un nodo en el denominado grafo de adyacencia. Cuando dos de dichas celdas tienen un límite en común, se agrega un arco al grafo que une sus correspondientes nodos[7]. Este grafo se usara para hallar la solución. La descomposición vertical y las técnicas quadtree son ejemplos de esta categoría.

Métodos de campos de potencial: se genera una función escalar sobre el espacio de configuraciones de modo que su gradiente pueda ser utilizado como una guía en la que se trace la trayectoria. Estos campos de potencial se crean a partir de los obstáculos y la configuración inicial y final[8].

Métodos Probabilísticos: son aquellos que utilizan generación de configuraciones aleatorias para identificar su entorno[9]. Ejemplos de estos métodos son los RRT y los PRM.

1 MARCO TEÓRICO

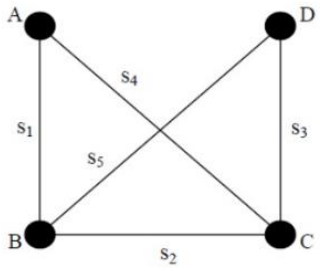
Se presentan algunos conceptos generales de robótica, planificación de trayectorias y navegación en Robots móviles, todo bajo un enfoque hacia la propuesta que se describe para la generación de trayectorias en un robot móvil, con el objetivo de aclarar estos conceptos manejados más adelante y que serán de gran apoyo a lo largo del documento.

1.1 Grafo

Los grafos son un lenguaje, notación, que permite hacer una relación entre objetos en un conjunto. Formalmente, un grafo G es un conjunto de V (vértices) y de A (aristas) tomado de la colección de subconjuntos de dos elementos de V. Una arista de G es un subconjunto como $\{a, b\}$, con $a, b \in V, a \neq b$.
De forma general un grafo es una colección de vértices que se unen por aristas, los puntos en el plano son las vértices y las aristas son líneas que unen estos puntos[10] (ver figura 1).

El siguiente grafo G(figura 1), consta de cuatro vértices, A, B, C y D y cinco aristas o segmentos. $S1 = \{A, B\}, S2 = \{B, C\}, S3 = \{C, D\}, S4 = \{A, C\}, S5 = \{B, D\}$.

Figura 1 Grafo



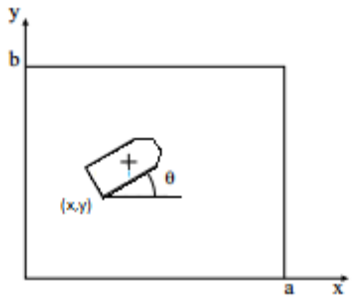
Fuente: Autor

1.2 Espacio de las configuraciones

Es el conjunto de todas las posibles configuraciones en las que se puede situar el robot. Este concepto es una herramienta de representación, fue introducido por Lozano-Pérez[5], y ocupara un papel importante en la descripción de este documento. Cada punto del espacio de configuraciones es una tupla, de una cierta dimensión, en las que se especifican los datos para los parámetros que se corresponden con los grados de libertad del robot.

La posición y orientación del robot en su espacio de trabajo, quedan completamente determinadas si se especifican los valores para los grados de libertad del robot, es decir, si suponemos que un robot móvil que puede moverse y girar libremente sobre un plano con unas dimensiones $[0, a]$ x $[0, b]$ [11], como se muestra en la figura 2. Para describir una configuración del robot, y todos los puntos que lo componen, es necesario fijar la posición de uno de los puntos del robot y la orientación de este respecto a un sistema de referencia fijo. La configuración del robot vendrá dada por una tupla de tres dimensiones (x, y, θ), y el espacio de las configuraciones estaría formado por todas las posibles configuraciones, donde x e y se determinan por las dimensiones del espacio y theta tomara valores en el intervalo $[0,2\pi]$.

Figura 2 Robot Móvil



Fuente: Autor

1.3 Grados de libertad

Los grados de libertad de un robot A son el número de parámetros necesarios para especificar sus movimientos, estos pueden ser desplazamientos y rotaciones.

Un robot móvil que se mueve sobre un plano, normalmente tiene 3 grados de libertad los cuales se miden respecto a un punto de referencia fijo, estos son la posición del robot (x, y) y su orientación (θ)[12].

1.4 Robot móvil

Existen muchas clasificaciones de los robots, estos pueden ser por su geometría, por su método de control o por su función, en esta última categoría se encuentran

los robots manipuladores y los robots móviles, los primeros se desempeñan en aplicaciones de manufactura y los robots móviles se emplean en aplicaciones de exploración. En el caso de los robots móviles, existen varios tipos según su forma de locomoción:

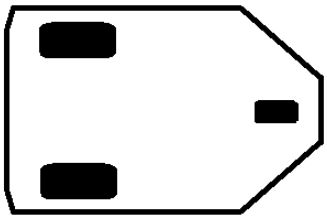
- Terrestres: robots de ruedas o de patas.
- Acuáticos
- Aéreos
- Espaciales

Los robots con ruedas son los más usados por su fácil construcción, buena capacidad de carga y baja complejidad en su sistema de control. Los robots que emplean ruedas están limitados a desplazarse en terrenos planos, ya que tienden a presentar problemas si las diferencias en el terreno son mayores al radio de las ruedas.

1.5 Robot Diferencial

Es uno de los esquemas más sencillos, consiste de dos ruedas en un eje común, donde cada rueda se controla independientemente. Esta configuración permite realizar desplazamientos en forma recta, de arco, o girar sobre su eje. En esta configuración se puede utilizar una rueda para mantener su balance, como se muestra en la figura 3.

Figura 3 Esquema de un robot móvil de tracción diferencial

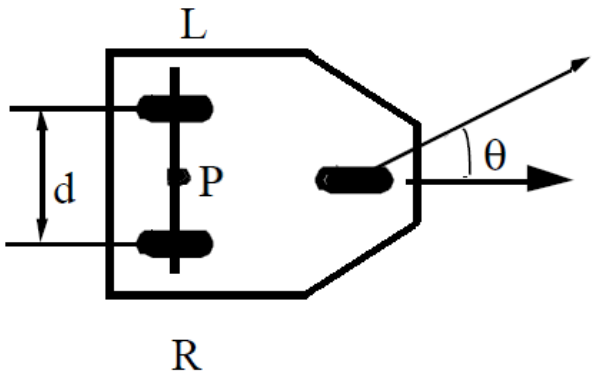


Fuente: Autor

1.6 Odométria

Es una técnica que tiene por objeto estimar la posición y orientación de un robot móvil a partir del número de vueltas dadas por sus ruedas. La idea principal de la odométria es la integración temporal del movimiento, lo cual lleva inevitablemente a la acumulación de errores[13].

Figura 4 Esquema Robot tipo Diferencial



Fuente: Autor

En la actualidad, para llevar la cuenta del número de vueltas dadas, se usan codificadores ópticos de alta precisión sobre los ejes de las ruedas, para poder estimar la posición, se requiere el registro odométrico de las dos ruedas, para mostrar el modo de operación de esta técnica, se considera la figura 4, en cuyas ruedas traseras lleva codificadores. Se toma el punto medio entre ambas ruedas (punto P de la figura 4).La distancia recorrida en un intervalo de tiempo por el punto P viene dado por:

$$\Delta s = \frac{1}{2}\left(\Delta s_R + \Delta s_L\right) \tag{1}$$

Donde s_R y s_L son las distancias recorridas por las ruedas. La variación en la orientación del vehiculo viene dada por:

$$\Delta\theta = \frac{1}{d}\left(\Delta s_R - \Delta s_L\right) \tag{2}$$

Donde d es la distancia entre ambas ruedas.

Al integrar las ecuaciones 1 y 2 se obtiene, respectivamente:

$$s(t) = \frac{1}{2}\left(s_R(t) + s_L(t)\right) \tag{3}$$

Y

$$\theta(t) = \theta(0)\frac{1}{d}\,(\mathrm{s_R}(t) + \mathrm{s_L}(t)) \tag{4}$$

Donde s (t) es la distancia recorrida por el punto P y $\theta(t)$ es la orientación del vehiculo, ambas para un tiempo t.

La ecuación 4 permite conocer la orientación del robot móvil en cualquier instante de tiempo. Para saber las coordenadas x e y del punto P se usan las siguientes ecuaciones[14]:

$$x(t) = x(0) + \int_0^t \frac{1}{2}(\mathrm{s_R}(t) + \mathrm{s_L}(t))\cos(\theta(t))dt \tag{5}$$

$$y(t) = y(0) + \int_0^t \frac{1}{2}(\mathrm{s_R}(t) + \mathrm{s_L}(t))\cos(\theta(t))dt \tag{6}$$

Las ventajas más relevantes de esta técnica son su simplicidad, bajo costo, y permite altas tasas de muestreo. Sin embargo, se necesita una calibración debido al desgaste de las ruedas, es vulnerable a imprecisiones por el deslizamiento de las ruedas, las irregularidades del terreno, entre otras.

2 ALGORITMOS DE PLANIFICACIÓN

La planificación se define como la búsqueda de la ruta más libre de obstáculos desde una posición inicial hasta una final a través del entorno de trabajo, en este caso, la plataforma robótica diferencial [15]. Esta acción se construye mediante el uso de la información que posee el entorno, la descripción del trabajo de navegación y la metodología a usar como estrategia. Es decir el planificador se define por el entorno de trabajo y el algoritmo de búsqueda utilizado.

Se definirán los métodos de planificación más significativos, elegidos entre los más acordes al objetivo general. Cada uno de ellos tiene sus ventajas e inconvenientes. Por ejemplo se debe tener en cuenta si el robot tiene restricciones cinemáticas o dinámicas ya que el problema se vuelve más complejo. Estos factores hacen interesante conocer las cualidades de las plataformas robóticas por lo cual se exponen a continuación. Finalmente interesa conocer las características que se comparan con el algoritmo RRT que se explicara más adelante, justificando en lo posible el por qué se eligió como base para este modelo de búsqueda.

2.1 METODOS “ROADMAP”

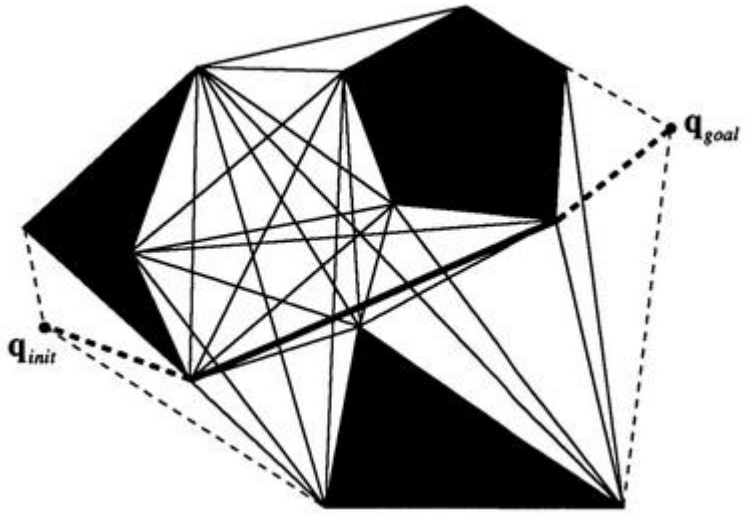
2.1.1 GRAFOS DE VISIBILIDAD

Es un método que se define como la construcción de un grafo que representa en sus nodos configuraciones factibles del robot, y en sus arcos el coste de conexión directa entre dos vértices. Al trazarse el grafo, denominado grafo de visibilidad[16], se halla un camino entre el origen y el destino y se reduce a conectar los puntos con el grafo y buscar una secuencia de nodos dentro de este [17]. El término de visibilidad para este método significa el enlace entre vértices en los que existe visión directa, es decir, sin obstáculos que lo impidan.

El método de grafos de visibilidad como otros planificadores, consiste de una descripción poligonal de los obstáculos. Si se asume el modelo de un robot puntual, tal y como se expresó anteriormente, esto implica que los obstáculos coincidan con los obstáculos de configuración. Así, el entorno de estudio queda definido por los límites del plano en el que se desplaza el robot y un conjunto de polígonos que identifican los obstáculos. Dentro del espacio de configuraciones libres de colisión se sitúan las posiciones de partida y meta, quedando completamente definido el problema.
El proceso comienza generando una lista de los vértices de los polígonos a los que se añaden la posición inicial y final. Con estos puntos se genera una matriz cuadrada con tantas filas como puntos. A continuación se evalúa para cada par de puntos su posibilidad de enlace directo, es decir, si el robot ubicado en uno de los puntos puede alcanzar el otro desplazándose en línea recta. Si un obstáculo no lo impide, esto será posible, y la longitud del tramo a recorrer se almacena en la matriz indicando el coste de realizar dicho enlace. Si por el contrario existe obstáculo, en

la matriz se anotará un valor que indique tal eventualidad. Esta matriz no es más que la descripción de un grafo que une los puntos entre los que es posible el enlace directo (ver figura 5).

Figura 5 Grafo de visibilidad con espacio de configuración poligonal



Fuente: Robot Motion Planning. Jean-Claude Latombe.1991.p13

2.1.2 DIAGRAMA DE VORONOI

2.1.2.1 Problemas de aproximación

Es uno de los métodos más importantes en la geometría computacional, para llevar a cabo una definición más directa primero se definirán los llamados problemas de aproximación [18]. Estos se presentan cuando se necesita calcular la distancia euclidiana entre puntos, líneas, polígonos y círculos. Los siguientes son algunos de ellos:

- Pares más cercanos: hallar los dos pares de puntos más cercanos en un conjunto S de n puntos.

- El vecino más cercano: para todo punto P en un conjunto S, hallar el punto más cercano.

- Arboles de longitud mínima (Euclidean-minimum-spanning-Tree) conectar todos los puntos de un conjunto S, mediante un grafo de árbol de longitud mínima.

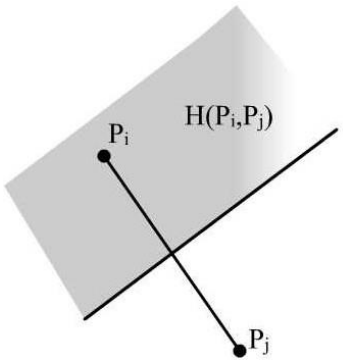
- Máximo circulo vacío: hallar el mayor circulo que no contenga puntos de S y cuyo centro se interior a la envolvente convexa de S

2.1.2.2 Definiciones básicas

Los problemas de proximidad, enunciados anteriormente, puedes ser resueltos usando una herramienta que contiene toda la información sobre la proximidad de los puntos. El diagrama de Voronoi es uno de los conjuntos más representativos[19].

Supongamos que tenemos un conjunto S formado por n puntos en el plano. Para cada pareja de puntos del plano más cercano a p_i que a p_j en un semiplano determinado por el bisectriz del segmento $\overline{p_i p_j}$ y se denota por $H\ (p_i, p_j)$.

Figura 6 Hiperplano que contiene a $\boldsymbol{p_i, p_j}$



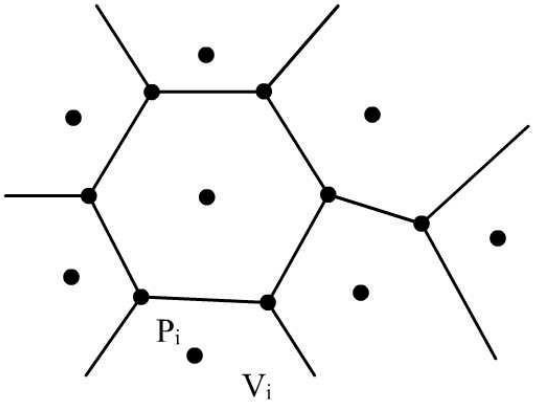
Fuente: Geometría computacional. Francisco Rivero M. p 52

Si para cada punto pi en S, denotamos por Vi el conjunto de puntos del plano más cercano a pi que a los restantes puntos de $S - \{p_i\}$ entonces se puede probar que.

$$V_i = \bigcap_{j \neq i} H(p_i, p_j) \tag{7}$$

Teniendo en cuenta esto, un diagrama de Voronoi para un conjunto $S = \{p_1, \dots, p_n\}$ de puntos en el plano, es una partición del plano en n regiones poligonales convexas $V_1, \dots, V_n$ Para cada i todos los puntos de la región V_i están mas cercanos a p_i que a cualquier otro punto de $S - \{p_i\}$ (ver figura 7).

Figura 7 Diagrama de Voronoi



Fuente: Geometría computacional. Francisco Rivero M. p 53

En la figura 7 se muestra un diagrama de Voronoi para un conjunto S de 8 puntos. El polígono convexo V_i que contiene al punto p_i se llama Polígono de Voronoi del punto p_i, los vértices del diagrama se llaman Vertices de Voronoi y los segmentos de recta del diagrama se llaman los Lados de Voronoi[6].

Teniendo en cuenta lo anterior se puede concluir que:

Si tenemos un punto p_i en S, entonces su vecino mas próximo se halla en alguno de los polígonos de Voronoi adyacentes a V_i, y si se ordena en una lisata cada punto p_i con su vecino más cercano, entonces se puede buscar el par de elementos de S más cercanos.

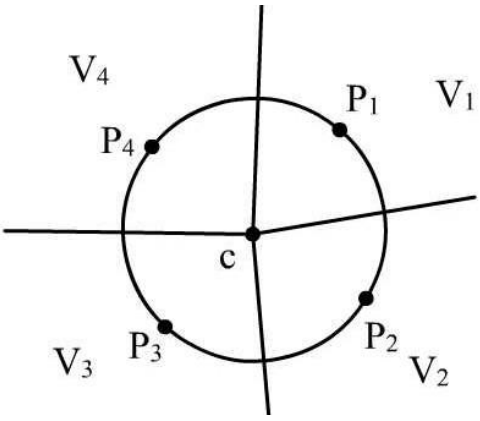
Esto permite resolver los dos problemas de aproximación enunciados anteriormente, el vecino más cercano y el par más cercano.

2.1.2.3 Propiedades del diagrama de Voronoi

- Si los cuatro puntos de un conjunto S están sobre un círculo C, entonces el centro del círculo es un vértice de Voronoi de grado 4.

Sean los puntos p_1, p_2, p_3, p_4 los puntos de S circulares, entonces cada segmento de recta bisectora de $\overline{p_i p_j}$ pasa por el centro del círculo C. Luego el punto c es un vértice de Voronoi, pues allí se produce intersección de dos segmentos bisectores.

Figura 8 Cuatro puntos cocirculares



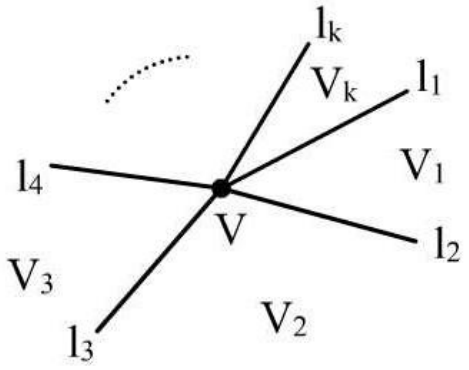
Fuente: Geometría computacional. Francisco Rivero M. p 54

C es el único nodo en $V_{or}(S)$ y es de orden 4, pues allí concurren las 4 rectas bisectoras.

- Todo vértice de Voronoi de S, tiene grado de orden tres.

Sea s $S = \{p_1, p_2, \dots, p_n\}$ y v un vértice de Voronoi, cada vértice $V_{or}(S)$ se obtiene como una interseccion de los lados de los poligonos de Voronoi. Supongamos que los lados de Voronoi $\{l_1, l_2, \dots, l_k\}$ convergen en v, y ademas l_i divide a los polígonos de Voronoi V_i y $V_i + 1$, Y l_i divide a V_1 y V_k.

Figura 9 Nodo de grado 3



Fuente: GEOMETRIA COMPUTACIONAL. Francisco Rivero M. p 55

Entonces, v es equidistante de los puntos p_1 y p_2, puesto que v esta sobre l_2, la línea que divide las regiones V_1 y V_2. De igual manera v es equidistante de $p_3, \dots, p_k$.

Luego los puntos $p_1, \dots, p_k$ estan sobre un circulo y por lo tanto $K \leq 3$ pues no hay 4 puntos cocirculares en S.

Si $K = 2$ entonces l_1 y l_2 dividen a los polígonos V_1 y V_2 y por lo tanto ambos lados pertenecen al segmento bisector de $\overline{p_1 p_2}$. Luego v no es vértice de Voronoi.

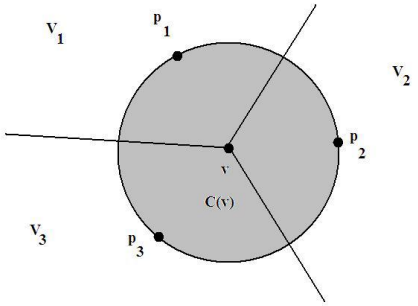
Si $K = 1$, entoces hay una sola semirecta que divide la región V_1 y por lo tanto esta no es convexa, lo cual es una contradicción.

Luego $K = 3$ y todo nodo de $V_{or}(S)$ tiene grado exactamente tres.

- Sea v un vértice de Voronoi. Entonces el circulo $C(v)$ no contiene puntos de S en su interior[18]

El circulo de Voronoi tangente en v, denotado por $C(v)$, es el circulo con centro en v y tangente a cada uno de los puntos p_1, p_2 y p_3.

Figura 10 Círculo de Voronoi



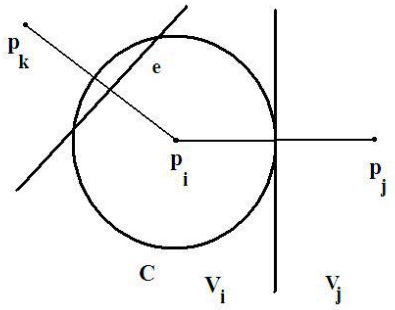
Fuente: Geometría computacional. Francisco Rivero M. p 56

Sean V_1, V_2 y V_3 los polígonos que coinciden en v. Estos vienen determinados por los puntos de S, p_1, p_2 y p_3. Si q es otro punto de S dentro de $C(v)$, entonces v estaría mas cerca de q que de p_1 y por lo tanto el polígono de Voronoi $V(q)$ debe incidir en v, lo cual es imposible. Luego no hay puntos de S en $C(v)$.

- Sea p_1 un punto en S, y p_j el punto en S mas cercano. Entonces las correspondientes regiones de Voronoi V_i y V_j comparten un lado en común[18].

Sea v el punto medio del segmento $\overline{p_i p_j}$. Sea C el circulo de radio $r = \frac{p_i p_j}{2}$ y con centro de p_i, entonces este circulo debe estar contenido en el polígono de Voronoi V_i, que contiene a p_i. Entonces si el circulo se sale de V_i, es cortado por algún lado de Voronoi e.

Figura 11 El vecino más cercano



Fuente: Geometría computacional. Francisco Rivero M. p 57

Luego existe un punto u de e, contenido en el interior del círculo. Por otro lado e es parte de la recta bisectora del segmento $\overline{p_k p_j}$, donde p_k es un punto de S, y $p_k \neq p_j$.

Entonces p_k esta mas cercano a p_i que a p_j, pues

$$d(p_i, p_k) \leq d(p_i, u) + d(u, p_k) = 2d(p_i, u) < 2d(p_i, v) = d(p_i, p_j) \quad (8)$$

Esto es una contradicción, y por lo tanto $C \subseteq V_i$.

2.1.2.4 Planificación basada en diagramas de Voronoi

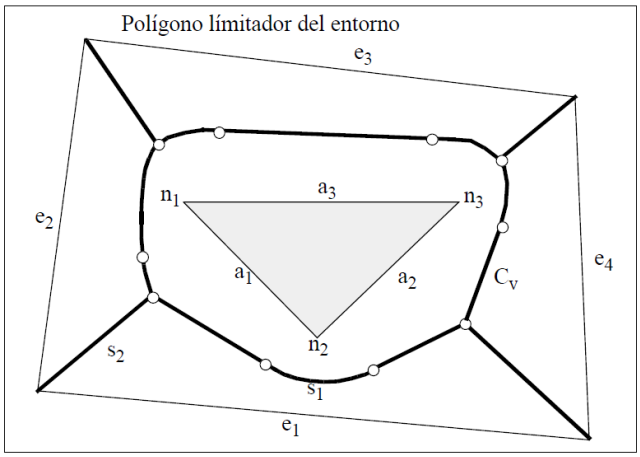
De acuerdo a la anterior definición, un diagrama de Voronoi es la proyección del espacio libre del entorno en una red de curvas unidimensionales yacientes en dicho espacio libre. Se definen como una reacción con preservación de la continuidad. Si el conjunto C_l define las posiciones libres de obstáculos de un entorno, la función retracción RT construye un subconjunto C_v continuo de C_l [18].

$$RT(q): C_l \rightarrow C_v / C_v \subset C_l \quad (9)$$

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

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 semilibres de obstáculos[19], 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 12 Retracción del espacio libre en un diagrama de Voronoi

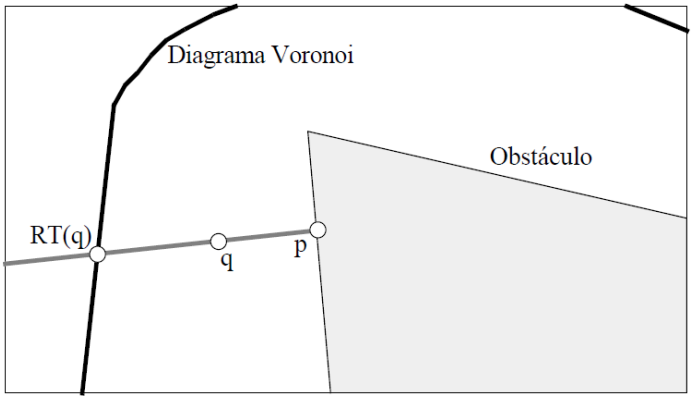


Fuente: Planificación de trayectorias para robots móviles. Víctor Fdo. Muñoz.1995.p 34

En la figura 12 muestra el entorno delimitado por un polígono de aristas $\{e_1, e_2, e_3, e_4\}$ y un obstáculo triangular de vértices $\{n_1, n_2, n_3\}$ y aristas $\{a_1, a_2, a_3\}$. 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 S_1 es el lugar geométrico de los puntos equidistantes entre la arista e_1, y el vértice n_2. Por otra parte, puede observarse como el segmento rectilíneo S_2 cumple la misma condición pero con respecto a las aristas e_1 y e_2.

Dado una configuración q no perteneciente ha C_v, 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 C_v de la línea que une p con q.

Figura 13 Configuración q en el diagrama de Voronoi



Fuente: Planificación de trayectorias para robots móviles. Víctor Fdo. Muñoz.1995.p 35

Este algoritmo consiste principalmente en encontrar la secuencia de segmentos S_i del diagrma de Voronoi tal que conecten $RT(q_a)$ con $RT(q_f)$, dicha secuencia conforma la ruta buscada, los pasos que conforman este algoritmo son:

- Calcular el diagrama de Voronoi
- Calcular $RT(q_a)$ y $RT(q_f)$
- Encontrar la secuencia de segmentos $\{S_1, \dots, S_p\}$ tal que $RT(q_a)$ pertenezca a S_1 y $RT(q_f)$ a S_p.
- 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.

2.2 MÉTODOS POR DESCOMPOSICIÓN EN 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,

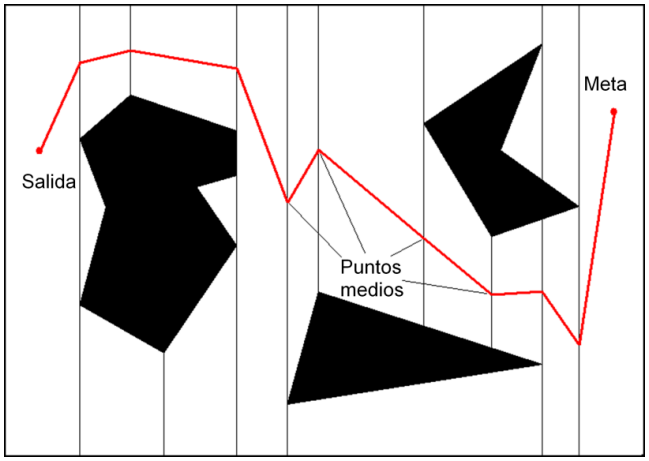
cada celda se le asigna un nodo, y estos se unirán o no en función de que las celdas que representen sean adyacentes. Existen dos tipos de este método:

2.2.1 Descomposición exacta:

2.2.1.1 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[1].
Un entorno en el que el escenario y los obstáculos tienen límites poligonales (ver figura 14), 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 14 Trayectoria definida por descomposición vertical



Fuente: Nuevas aportaciones en algoritmos de planificación para la ejecución de maniobras en robots autónomos no holónomos. Diego López. 2011. p 24.

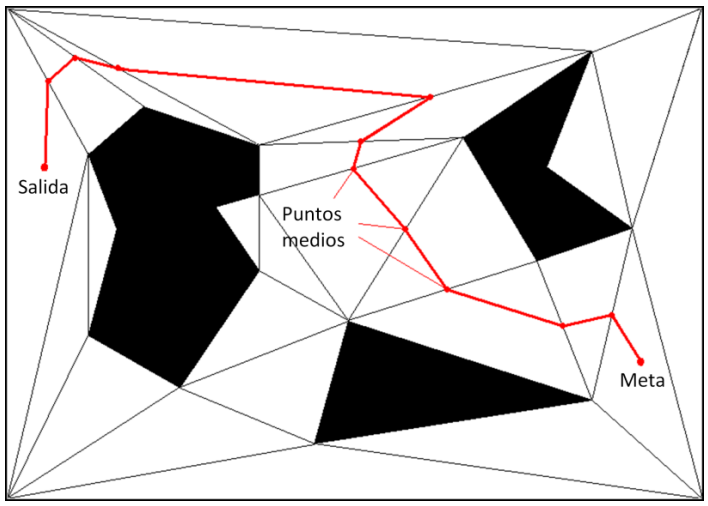
2.2.1.2 Descomposición de Delaunay

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[1].
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 15 Trayectoria definida por descomposición por Delaunay



Fuente: Nuevas aportaciones en algoritmos de planificación para la ejecución de maniobras en robots autónomos no holónomos. Diego López. 2011. p 25.

2.2.2 Descomposición aproximada

En este tipo de descomposición se usan celdas de diseño predefinido, lo más utilizado es de diseño rectangular. Al descomponer todo el escenario en dichas celdas, algunas solaparan bien completamente a un obstáculo. Estas serán descartadas, las demás serán las que conformen el grafo de adyacencia.
Por lo demás, el proceso es bastante similar a la descomposición exacta. La ventaja de este método es que reside en su sencillez de implementación. Como puede deducirse este tipo de descomposición no cumple con el requisito de la

descomposición exacta, debido a esas celdas descartadas que contiene en parte de configuraciones libres de colisión.

2.3 MÉTODO DE CAMPOS DE POTENCIAL ARTIFICIAL

Este método realiza una aproximación más directa para la planificación de movimientos, a diferencia de los métodos descritos anteriormente que se basan en la conectividad global del espacio de configuraciones libres de colisión en un grafo que luego se usa para realizar la búsqueda del camino, el método de campos de potencial artificial se enfoca en discretizar todo el espacio de configuraciones en una grilla regular de configuraciones donde realizar la búsqueda del camino. En este caso, debido al tamaño de la grilla, se requieren heurísticas de búsqueda suficientemente potentes. Una de las más exitosas se basa en funciones que son interpretadas como campos de potencial.

La metáfora que sugiere este método, es considerar el robot como una partícula que se mueva en el espacio de configuraciones bajo la influencia de capos artificiales de potencial producidos por la configuración objetivo y los obstáculos. Regularmente, aunque no necesariamente, la función de potencial es definida sobre el espacio de configuraciones libres de colisión como la suma de un potencial atractor, que atraiga el robot hacia la configuración objetivo, y un potencial repulsivo que lo aleja de los obstáculos.
En este enfoque, la planificación de movimientos es realizada de forma iterativa. En cada iteración la fuerza artificial:

$$\vec{F}(q) = -\vec{\nabla}U(q) \tag{10}$$

Inducida por la función de potencial, es considerada como la dirección prometedora para realizar el siguiente movimiento y la generación del camino se realiza a lo largo de esta dirección de a ciertos incrementos.

Originalmente, los campos artificiales de potencial fueron concebidos como una forma de realizar la evasión de obstáculos en tiempo real, aplicables en contextos donde no se dispone a priori de un modelo global del entorno y, por el contrario, se reconoce el entorno mediante sensado durante la ejecución de los movimientos.

2.3.1 Estructura general

El campo de fuerzas artificiales $\vec{F}(q)$ en C (espacio de configuraciones) es producido por una función de potencial con espacio de configuraciones libres de colisión:

$C_{free} U : C_{free} \rightarrow R,\ con\ \vec{F}(q) = -\vec{\nabla}U(q)$. En $C = R^n$ con $(n = 2\ o\ n = 3)$, q puede ser (x, y) o (x, y, z), con lo cual:

$$\vec{\nabla}U = \begin{pmatrix} \delta U/\delta x \\ \delta U/\delta y \end{pmatrix} o\ \vec{\nabla}U = \begin{pmatrix} \delta U/\delta x \\ \delta U/\delta y \\ \delta U/\delta z \end{pmatrix} \quad (11)$$

En orden de lograr que el robot sea atraído hacia la configuración destino mientras es repulsado por los obstáculos, U se define como la suma de dos funciones de potencial más elementales como se indica en la ecuación 11.

$$U(q) = U_{att}(q) + U_{rep}(q) \quad (12)$$

Donde U_{att} es el potencial atractor asociado a la configuración objetivo q_{goal} y U_{rep} es el potencial repulsivo asociado con las configuraciones de los obstáculos O. U_{att} es independiente de las configuraciones de los obstáculos O y U_{rep} es independiente de la configuración objetivo q_{goal}. De este modo $\vec{F}$ se define como la suma de dos vectores:

$$\vec{F}_{att}(q) = -\vec{\nabla}U_{att}\ y\ \vec{F}_{rep}(q) = -\vec{\nabla}U_{rep} \quad (13)$$

2.3.2 Potencial atractor

El potencial atractor puede definirse como una parábola, en los términos de la siguiente ecuación:

$$U_{att}(q) = \frac{1}{2}\xi\rho^2_{goal}(q) \quad (14)$$

Donde ξ es un factor de escalado y $\rho_{goal}(q)$ denota la distancia Euclidiana $\|q - q_{goal}\|$. Esta función positiva o nula y tiene un minimo en q_{goal}, donde $U_{att}\left(q_{goal}\right) = 0$. La función ρ_{goal} es diferenciable de todo es el espacio de modo que, en cualquier configuración q, la fuerza atractora artificial $\vec{F}_{att}$ derivada del potencial atractor U_{att} esta determinada por la ecuación:

$$\vec{F}_{att}(q) = -\vec{\nabla}U_{att}\ (q) = -\xi\rho_{goal}(q)\vec{\nabla}\rho_{goal}(q) = -\xi(q - q_{goal}) \quad (15)$$

2.3.3 Potencial repulsivo

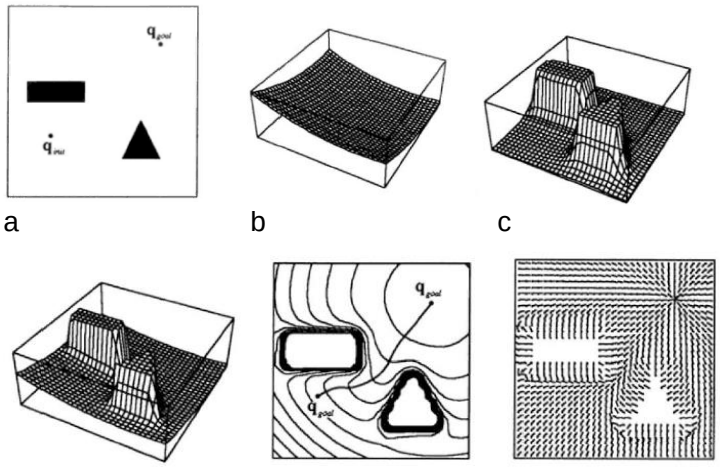
Este crea una barrera alrededor del espacio de configuración de obstáculos. Por el contrario, es deseable que el potencial repulsivo no afecte los movimientos del robot

cuando esté suficientemente alejado de los obstáculos. Una forma de lograrlo es definiendo la función de potencial repulsivo como lo indica la ecuación:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho 0}\right)^2 & si\ \rho(q) \leq \rho 0 \\ 0 & si\ \rho(q) > \rho 0 \end{cases} \quad (16)$$

Donde η es un factor de escalado, $\rho(q)$ denota la distancia desde q hasta el espacio de configuraciones de los obstáculos y $\rho 0$ es una constante positiva que representa la distancia de influencia de los obstáculos. La función U_{rep} es positiva o nula, tiende a infinito a medida que el robot se acerca a la región donde están los obstáculos y se anula cuando la distancia entre la configuración del robot y la de los obstáculos es mayor que $\rho 0$.

Figura 16 Campos artificiales de potencial



Fuente: Robot motion planning. J.-C. LATOMBE, 1991, p20.

El segmento a muestra un espacio de configuraciones de dos dimensiones conteniendo dos obstáculos. El potencial atractor tiene su mínimo en la configuración objetivo q_{goal} en el segmento b; mientras que el potencial repulsivo se aprecia en el segmento c. la suma de ambos potenciales se muestra en el segmento d, y finalmente en los segmentos e y f se pueden apreciar los diagramas de curvas de nivel junto a la trayectoria encontrada y una matriz conteniendo una

discretización del vector gradiente negado correspondiente al potencial total sobre el espacio de configuraciones libres, respectivamente.

2.4 MÉTODOS DE RESOLUCIÓN PROBABILÍSTICOS

Estos métodos no intentan representar fielmente el espacio de configuraciones libre, sino, por el contrario construir una representación simplificada del mismo, independientemente de su geometría.

Considerando que la posición y orientación del robot móvil que se mueve en el plano (x, y, z) y la representación del robot móvil con restricciones dinámicas $(\dot{x}, \dot{y}, \dot{z})$, resulta un espacio de configuraciones de seis dimensiones.
En este contexto, surgieron algoritmos simples basados en muestreo o sampling de configuraciones, como una herramienta poderosa para realizar planificación de movimientos en espacios de gran dimensión.

Existen dos clases principales de algoritmos, una computa previamente el roadmap permitiendo que, sobre un entorno estático, múltiples planificaciones puedan consultarse y realizarse rápidamente, esta es conocida como multi-query planning y la otra clase no realiza ningún calculo previo y construye un pequeño roadmap en simultaneo mientras procesa una sola planificación tan rápido como sea posible, se conoce como single- query planning y un ejemplo claro es el algoritmo RRT.

El algoritmo RRT originario se basa en la construcción de un árbol de configuraciones que crece explorando a partir de un punto origen. Para entender mejor el comportamiento del algoritmo, en la sección 4 se describe detalladamente ya que este algoritmo es en el que se basa este proyecto como método de exploración aleatoria.

3 VENTAJAS Y DESVENTAJAS DE LOS ALGORITMOS

Se presentan las principales ventajas y desventajas que presentan los algoritmos anteriormente descritos, desde el punto de vista que más convenga para la plataforma robótica usada en este caso y los entornos que idealmente pueda enfrentar.

Tabla 1 Ventajas y Desventajas

ALGORITMO	VENTAJA	DESVENTAJA
GRAFOS DE VISIBILIDAD	Escoge la mejor sucesión de vértices conexos.	Requieren algún tipo de pre-procesamiento que puede limitarse a la definición poligonal de los obstáculos en el espacio de configuración y en cuanto a su velocidad se caracterizan por ser lentos en la construcción de la trayectoria.
DIAGRAMA DE VORONOI	Capaz de alejarse de los obstáculos, eligiendo siempre el camino más despejado posible.	
DESCOMPOSICIÓN EN CELDAS	Fácil de implementar	
CAMPO DE POTENCIAL	Hace una subdivisión en celdas de todo el escenario y el cómputo de un determinado valor en cada una de ellas.	Generan una densidad de probabilidad no uniforme al crear la trayectoria. En cuanto a la representación gráfica y construcción es difícil de implementar. Método costoso desde el punto de vista computacional.
RRT(Rapidly exploring Random Tree)	Se inclina decididamente hacia los espacios inexplorados. Facilita el análisis de comportamiento y la implementación en cualquier escenario. Siempre permanecen conexos, aún con pocos vértices. Puede ser implementado en otros planificadores No requieren una definición explícita de Cfree, sólo utiliza una función que comprueba la existencia de colisión Es probabilísticamente completo (la probabilidad de encontrar un camino si existe tiende a 1 exponencialmente con el número de nodos).	Coste computacional alto. Necesita de una plataforma para la representación gráfica.

Fuente: Autor

4 RAPIDLY EXPLORING RANDOM TREE (RRT)

El algoritmo RRT originario se basa en la construcción de un árbol de configuraciones que crece explorando a partir de un punto origen.

```
RRT
T (0) ← q_start
    For i = 1 a K_max entonces:
        q_rand ← punto aleatorio
        Extiende (T, q_rand)
            q_near ← VecinomasCercano (T, q)
            if nuevoestado (q, q_near, q_new) entonces:
                Agreganodo (T, q_new )
                Agregaenlace (T, q_near, q_new)
                if q_new = q entonces:
                    devuelve Alcanzado
                else
                    devuelve Extendido
            devuelve Rechazado.
```

Donde:

$qinit$ = Es la configuración inicial (en el caso de un robot que se mueve en un plano, las coordenadas x e y de un punto de referencia y la orientación del vehículo respecto a uno de los ejes del sistema de referencia).

q = Es la configuración final que se desea alcanzar.

q_{rand} = Es una configuración aleatoria que genera el algoritmo dentro del espacio de configuraciones ($q_{rand} \in C$).

q_{near} = Es la configuración más próxima a q_{rand}, en el sentido definido por ρ, de entre las existentes en un árbol.

q_{new} = Es la configuración que se va a añadir al árbol.

Además se deben tener en cuenta algunas métricas que son importantes en la construcción del árbol.

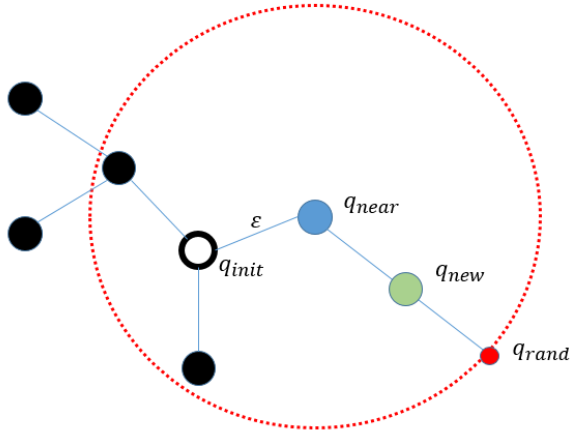
ρ = métrica definida dentro de C. Puede ser distancia euclídea u otra ponderación de proximidad que pueda interesar.

ε = Longitud del segmento de crecimiento. En realidad, es la distancia entre un punto del árbol y el siguiente con el que está conectado.

El algoritmo comienza inicializando la tabla asociada al árbol con la configuración origen. Seguidamente, entra en un bucle, limitado por un valor $Kmax$, cuya función es finalizar el algoritmo una vez se ha realizado un numero prefijado de iteraciones. Este valor se usara posteriormente para detener el algoritmo en el momento en que no se alcance o no encuentre la configuración final. Es importante tener en cuenta que la determinación de dicho valor depende de las características del espacio de configuraciones, el número de obstáculos.
Dentro del bucle del algoritmo se encuentran dos instrucciones importantes, la primera da un punto al azar dentro del espacio de configuraciones, y la segunda hace crecer el árbol en dirección a la configuración aleatoria anteriormente obtenida.

El crecimiento del árbol se realiza con la función Extiende, la estructura de esta función comienza con el cálculo de q_{near}. Esto se realiza gracias a la función $VecinoMasProximo$ que aplica la métrica ρ definida anteriormente a todos los vértices del árbol, obteniendo el punto más cercano a $qnear$ en dirección a q_{rand} [20].

Figura 17 Esquema de expansión del RRT



Fuente: Autor

La figura 17 muestra el esquema de expansión del RRT, se puede observar que el nuevo punto aleatorio definido con color rojo, se encuentra en un espacio muy alejado de la longitud del segmento de crecimiento, y la línea punteada nos muestra la configuración más cerca, es allí donde la nueva configuración toma su dirección y se sitúa a una distancia ε ya configurada

Para la obtención de q_{new} se tiene en cuenta si hay alguna colisión en dicho punto, devolviendo "falso" o "verdadero" según exista colisión o no.

Si no se detecta colisión, se agrega el nuevo punto al árbol teniendo en cuenta los dos casos, el primero es si q_{rand} coincide con los parámetros dados entonces q_{rand} será igual a q_{new} y se cumple el caso "$alcanzado$", si por lo contrario no se ha producido el alcance entonces devolverá el valor "$avanzado$". Por último, en caso de que la función $NuevaConfiguracion$ haya afirmado colisión no se agregara tal punto y el algoritmo seguirá en su bucle.

Para la implementación del algoritmo en Python, se necesita de una plataforma más dinámica para esto se importó una biblioteca llamada Pygame que nos permite visualizar la construcción del árbol en un entorno, ya sea diseñado o desarrollando los obstáculos a partir de un espacio cualquiera.

4.1 RAPIDLY EXPLORING RANDOM TREE EN PYTHON

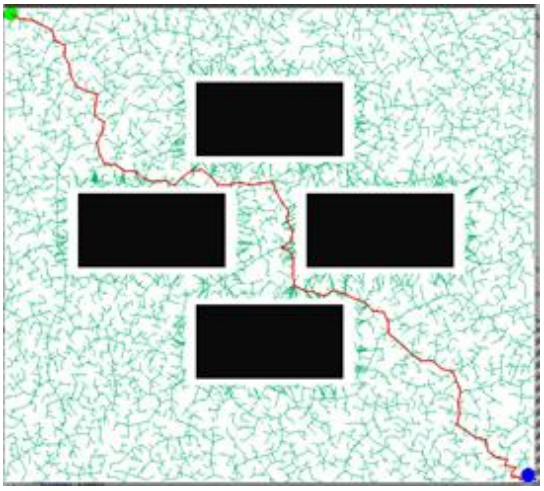
Como se mencionó anteriormente, una de las desventajas que abarca el algoritmo RRT es que necesita de una plataforma para su representación gráfica, para esto se importó Pygame, este es un conjunto multiplataforma de Python, que nos permite visualizar la construcción del árbol en un entorno, ya sea diseñado o desarrollado a partir de un espacio cualquiera, esto teniendo en cuenta que es un escenario en 2D.

Pygame trabaja sobre las librerías SDL (Simple DirectMedia Layer) que permiten la creación de videojuegos en dos dimensiones de una manera sencilla, las SDL son un conjunto de bibliotecas que proporcionan funciones básicas para realizar operaciones de dibujado 2D, gestión de efectos de sonido, música, carga y gestión de imágenes (y son importantes ya que Pygame trabaja sobre ellas).

4.2 PRIMERAS PRUEBAS DEL ALGORITMO RRT

Para inicializar el algoritmo, primero se debe definir el punto inicial del que parte el robot y posteriormente el punto al que se desea llegar; inmediatamente después el árbol empieza a crecer hasta que encuentre la trayectoria que une a estos dos puntos, esto se puede evidenciar en la siguiente figura.

Figura 18 Generación de Trayectoria por RRT



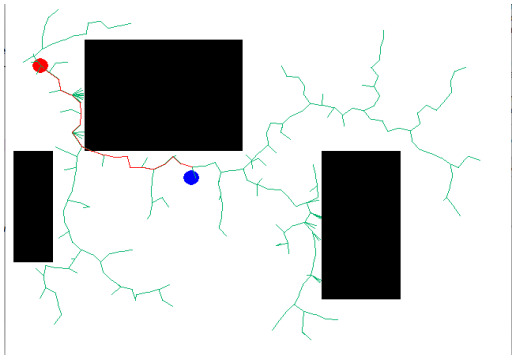
Fuente: Autor

En la figura 18, el punto inicial se identifica con el color verde mientras que el punto final con el color azul, ya que los puntos en este caso se definieron a los extremos del espacio, el árbol tuvo un comportamiento muy extenso e hizo un barrido en todo el entorno.

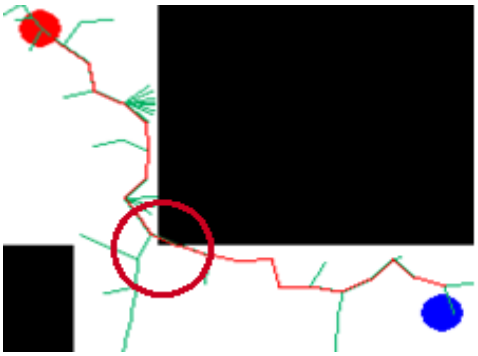
4.3 SOLUCION AL ERROR PRESENTADO

Al realizar varias pruebas se encontró el primer error en el algoritmo al hallar una trayectoria en la que si se apreció colisión, pero aun así el algoritmo no la detecto.

Figura 19 a) Error en Algoritmo RRT, b) Acercamiento al punto de colisión



a)



b)

Fuente: Autor

La figura 19 a) muestra la trayectoria en la que ocurrió el error y la figura 19 b) hace una ampliación de está identificando donde fue la colisión.

El error se identificó, ya que la función $collides$ se encarga de verificar si existe colisión en el punto de configuración nuevo, es decir esta función no identifica la colisión del tramo que se traza si no en solo los nuevos puntos de configuración.

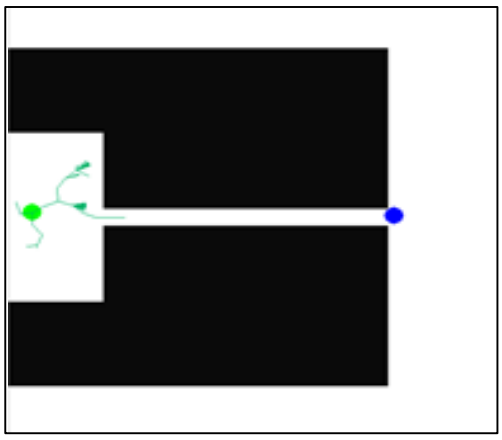
Para solucionar este error se creó la función $collidesRect$, esta es muy similar a la función $collides$ pero esta cumple la función de analizar si hay colisión con el robot que se sitúa en el nuevo punto generado en el árbol, y así se puede establecer que no se generará un tramo que colisione con algún obstáculo.

```
def collidesRect(r):
    for rect in rectObs:
        if rect.colliderect(r) == True:
            return True
    return False
```

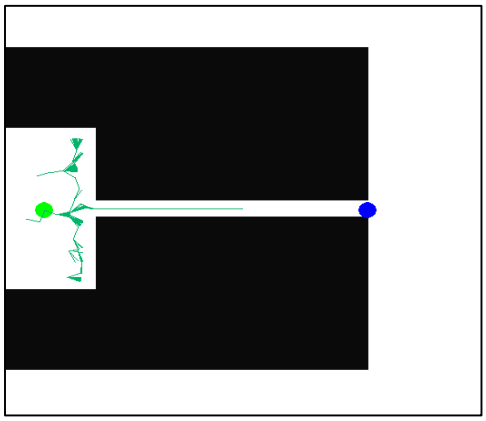
4.4 PROGRESIÓN DEL RRT EN UN ENTORNO COMPLEJO

En las figuras 20 a), b), c) y d) se representa la progresión del algoritmo RRT sobre un espacio más complejo. Para ello se sitúa el origen del árbol RRT a un lado del escenario. En la Evolución se aprecia que desde el punto origen, situado en la parte izquierda, parte una rama que crece con más preponderancia con respecto a las demás, la causa es la alta frecuencia con que aparecen puntos aleatorios a la derecha del origen del árbol.

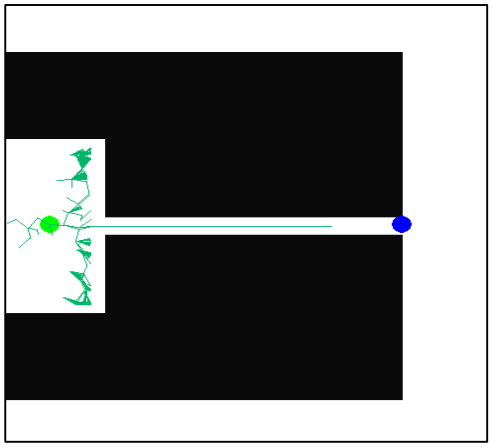
Figura 20 Evolución del RRT en un entorno asimétrico.



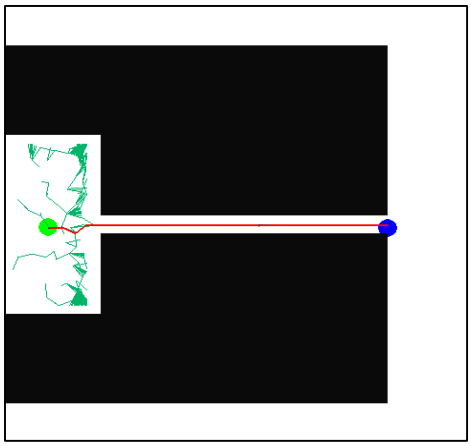
a) 100 iteraciones



b) 200 iteraciones



c) 500 iteraciones



d) Más de 500 iteraciones

Fuente: Autor

Gracias a este entorno mostrado en la figura 20, se puede apreciar que la expansión del algoritmo RRT se inclina decididamente hacia los espacios inexplorados, además la distribución de los obstáculos así como también la de los puntos inicial y final, hacen tender el árbol hacia una dirección.

5 ANÁLISIS Y SUAVIZADO DE LA TRAYECTORIA GENERADA POR RRT

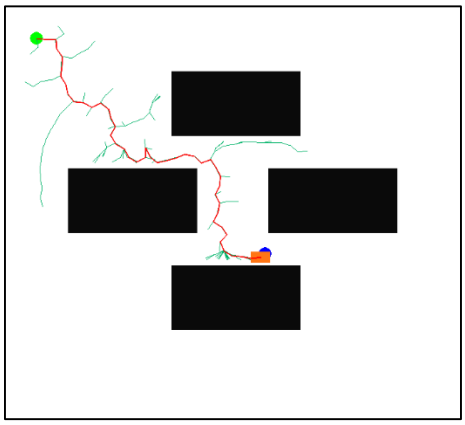
5.1 Regresión lineal

Ya que la trayectoria generada por RRT no es una trayectoria suave que puede seguir un robot móvil se decide establecer los puntos críticos de la trayectoria generada y con ayuda de un herramienta más practica como lo es Excel graficar la trayectoria y elegir aquellos puntos para luego trabajar sobre cada tramo que queda. Para lograr esto primero se crea la función guardar, creada específicamente para extraer las coordenadas de la lista en la que se encuentra la trayectoria producida. Esta función crea un archivo independiente de texto y puede actualizarlo las veces que se ejecute el programa.

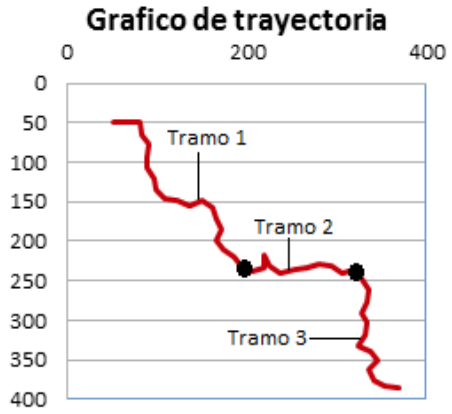
```
def guardar (file_txt, coor):
    archivo = open(file_txt, "a")
    for posX,posY in coor:
        archivo.write(str(posX)+","+str(posY)+"\n")
    archivo.close()
```

Una vez generada la trayectoria (figura 21a) y confirmado el archivo de texto con los datos de esta, se grafica en Excel (figura 21b) para identificar y elegir con mayor precisión los tramos en que se va a dividir esta.

Figura 21 a) Trayectoria generada por RRT y b) Grafica de la trayectoria



a)



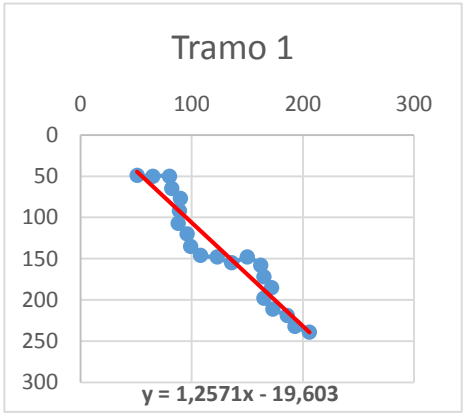
b)

Fuente: Autor

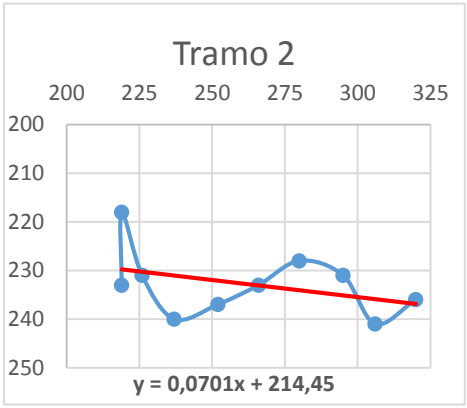
En la figura 21a se muestra la ruta generada por RRT en un entorno básico con 4 obstáculos, la figura 21b es la misma trayectoria pero graficada en Excel, identificando los puntos críticos y los tramos en los que se convirtió.

Teniendo los tramos identificados (tramo 1, tramo 2 y tramo 3) se procede a sacar la regresión lineal con ayuda de las herramientas de gráficos en Excel, ya que este modelo matemático (regresión lineal) se usa para aproximar la relación de dependencia.

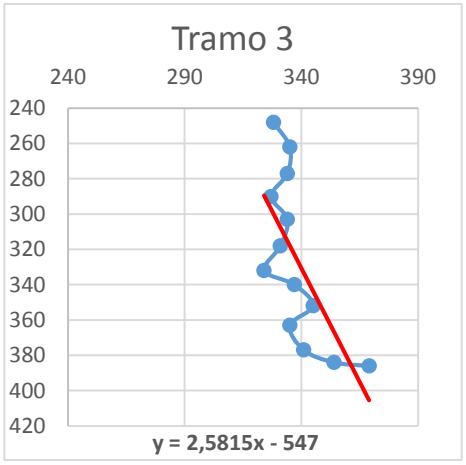
Figura 22 Tramos de la trayectoria y grafico resultante



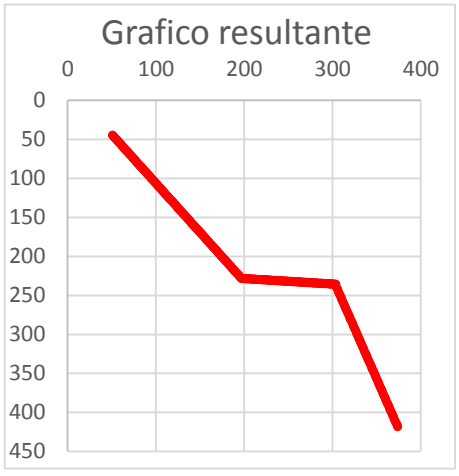
a)



b)



c)



d)

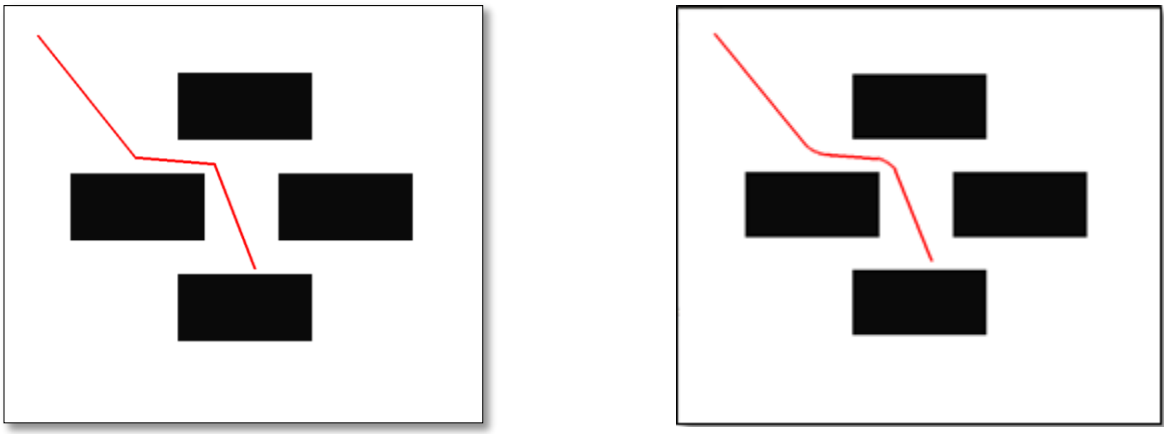
.

Fuente: Autor

Como se puede apreciar, en la figura 22(a), (b) y (c) los tramos correspondientes a la trayectoria mencionada anteriormente (tramos azules),cada uno de ellos con su respectiva ecuacion de regresion lineal y linea de tendencia (tramo rojo); la figura 22d nos muestra la trayectoria resultante.

El proceso que se realizo para tener como finalidad esta trayectoria, comienza encontrando los puntos criticos de la trayectoria generada por RRT, una vez identificados, esta se divide en tramos, los cuales se analizan independientemente calculando la linea de tendencia y graficandola; finalmente se grafican los nuevos tramos y se identifican los puntos de interseccion entre estas. Finalmente se grafica la trayectoria resultante y nuevamente se implementa en el entorno al que pertencia(ver figura 23a).

Figura 23 a) Trayectoria después del proceso de regresión lineal y b) trayectoria suavizada.



a) b)

Fuente: Autor

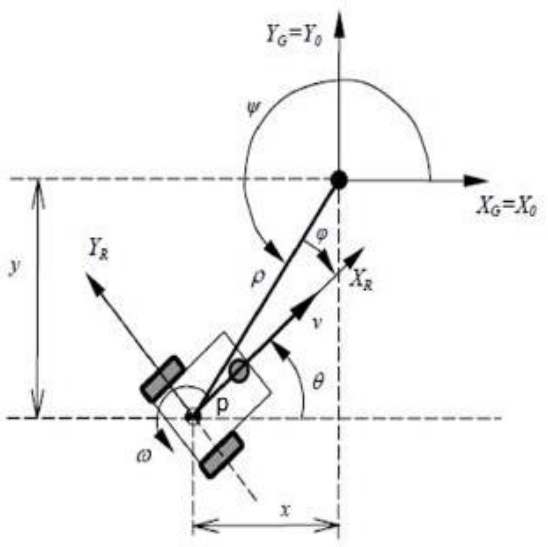
5.2 Control de movimiento para el suavizado.

La figura 23 b) entrega la trayectoria final que debe recorrer el robot, pues ya se encuentra suavizada.

El criterio que se tomó para suavizar la trayectoria, parte del control de movimiento para los robots de configuración diferencial, este control se basa en el modelo cinemático, este se describe formalmente más adelante junto con el modelo dinámico del Robot.

Este control de movimiento se realiza proporcionando las velocidades de las ruedas $v(t)$ y $w(t)$, variables de control[21].
Este criterio, consiste en un controlador por realimentación de estados, que es adecuado para el cumplimiento de misiones de navegación como movimiento de un punto a otro. Para su diseño se considera un estado cualquiera de posición y orientación del robot y se define un punto objetivo a alcanzar, como se muestra en la figura 24.

Figura 24 Situación considera para el Robot



Fuente: kinematic control of mobile robots to produce chaotic trajectories. Luiz S. Martins-Filho, et al.

Donde se define:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{17}$$

$$\varphi = 180 + \theta - \psi \tag{18}$$

En la figura 24, φ es el ángulo definido entre $(\pi, -\pi)$ y determina el sentido de giro del robot en w o $-w$. Mientras que ρ describe la distancia entre P y el objetivo, y ψ el ángulo entre la dirección en línea recta al objetivo y el eje X_0.

Así se establece el criterio de control para determinar las entradas del sistema v y w [21]:

$$v = k_1 \rho \cos\varphi \;\; ; \;\; w = -k_1 sen\varphi \cos\varphi - k_2 \varphi \tag{19}$$

6 NAVEGACIÓN EN PLATAFORMAS ROBÓTICAS MÓVILES

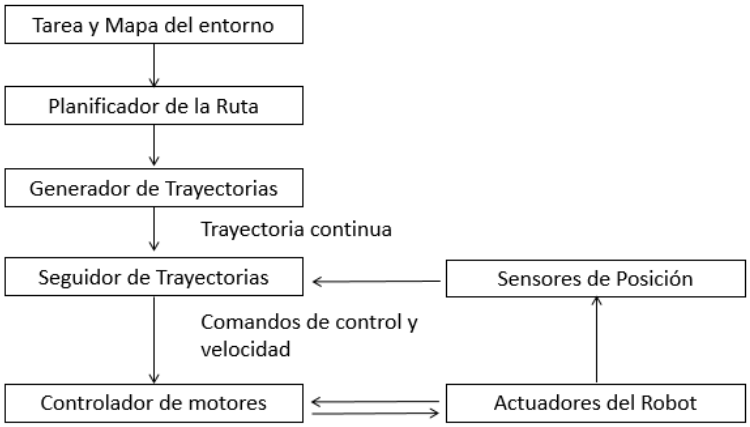
6.1 Esquemas de navegación

Una tarea de navegación para un robot significa recorrer un camino que lo lleve de una posición inicial hasta una final, atravesando por ciertas posiciones intermedias o subtemas. El problema de navegación se puede considerar en las siguientes etapas:

- Percepción del entorno: mediante uso de sensores externos, se recrea un mapa o modelo del entorno donde se desarrollara el trabajo de navegación y búsqueda de trayectorias.
- Planificación de la trayectoria: se crea una secuencia ordenada de metas o subtemas que deben ser seguidas por el robot. Esta secuencia se calcula usando el modelo o mapa del entorno, un algoritmo de planificación y búsqueda para desarrollar la tarea a realizar.
- Generación del camino: se define una función continua que interpola la secuencia de objetivos construida por el planificador para posteriormente proceder a la discretización con el fin de generar la trayectoria.
- Seguimiento del camino: se efectúa el movimiento del robot desplazándose según la trayectoria generada mediante el adecuado control de los actuadores de la plataforma robótica.

Estas etapas se pueden construir de una forma separada, aunque en el orden especificado. La interrelación de estas etapas conforma el control de navegación básico en un robot móvil, como se muestra en la figura 25.

Figura 25 Control básico para un robot móvil.



Fuente: Autor

El control básico para un robot móvil, parte de un mapa de entorno y de las especificaciones de la tarea de navegación. De estos datos se realiza la planificación de un conjunto de objetivos representados como una secuencia de puntos cartesianos que definen la ruta, este debe cumplir los requisitos para que esta trayectoria esté libre de obstáculos.
Al usar el generador de trayectorias se construye la referencia que utilizara el seguidor de caminos para generar los comandos de direccionamiento y velocidad que controlaran los motores del robot. Finalmente el uso de sensores de posición en conjunción con técnicas odométricas, se produce una estimación de la posición actual en el mapa, la cual realimentara la etapa de seguidor de caminos[22].

Para desarrollar esta tarea depende mucho del conocimiento que se posee del entorno de trabajo. Así la figura anterior considera que el mapa del entorno cuenta con una forma fiel a la realidad. Si se hace el uso adecuado del mismo se puede construir un camino que corresponda a los requisitos de la tarea de navegación.

Es posible que el modelo del entorno del que dispone el robot tenga ciertas imperfecciones al omitir detalles, y esto daría como resultado que el esquema anterior no sea muy eficaz, ya que no asegura la construcción de un camino libre de obstáculos.

6.2 Planificación de la ruta

Esta consiste en encontrar una ruta segura que sea capaz de indicar al robot la trayectoria que debe seguir desde la posición actual hasta la posición especificada de destino. Que sea segura se refiere al cálculo de un camino al menos continuo en posición, que sea libre de obstáculos. En esta ruta, el generador construirá las referencias que se le entregan al control de movimientos. Por eso, en la especificación de esta ruta se hacen triviales las características cinemáticas y dinámicas de robot, ya que el cómputo de los puntos de referencia adecuados que cumplan con todos los atributos es tarea del generador de trayectorias.

Se propone una formalización del concepto de ruta, para poder describir métodos que realizan la acción de planificación.

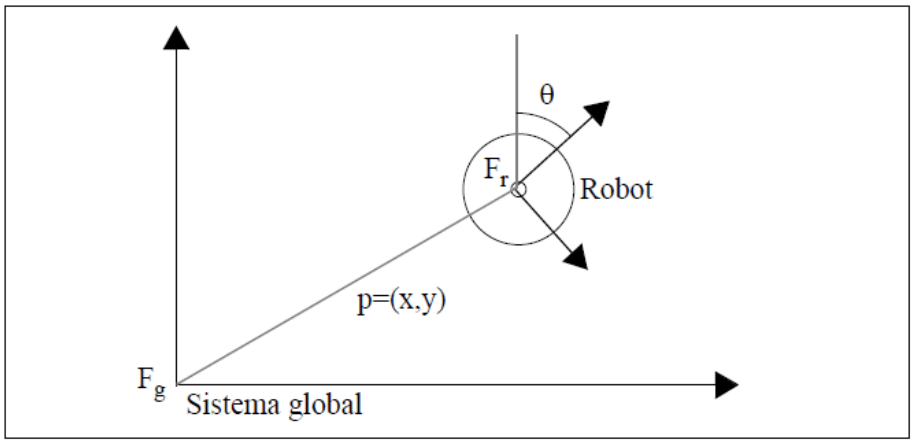
6.2.1 Formalización del problema de planificación

El entorno de trabajo en el cual un robot móvil ejecutara sus acciones, pueden ser considerados como un conjunto de configuraciones en las cuales se puede encontrar el robot en un determinado instante de tiempo.

Se define una configuración $\boldsymbol{q}$ como el vector de componentes que proporcionan información completa sobre el estado actual del robot. Un robot es un objeto rígido

el cual se puede asociar a un sistema de coordenadas móvil, la localización del vehículo en un instante de tiempo determinado queda definido por la relación que existe entre el sistema de coordenadas global Fg en el que está definido todo el entorno de trabajo y su sistema de coordenadas locales asociado Fr.

Figura 26 Sistema de coordenadas global y sistema local asociado a robot



Fuente: PLANIFICACIÓN DE TRAYECTORIAS PARA ROBOTS MÓVILES. Víctor Fdo. Muñoz.1995.p 29

El vector que da información sobre el estado actual del robot viene dado por dos componentes, la posición p y la orientación θ, por tanto se puede definir la configuración como:

$$q = (p\,, \theta) = (x, y, \theta) \tag{20}$$

Se denomina espacio de configuraciones C del robot R a todas las configuraciones q que puede tener el robot en su entorno de trabajo. El subconjunto de C ocupado por el robot R cuando este se encuentra en q, se denota por $R\,(q)$. Si el robot se modela como en la figura de una forma circular con radio ρ, $R(q)$ se define como:

$$R(q) = \{\, q_i \in C / |\, q, q_i \,| \leq \rho\} \tag{21}$$

En caso de un robot puntual, en la expresión (21), $\boldsymbol{\rho}$ es nulo, con lo cual se cumple:

$$R(q) = \{q\} \tag{22}$$

En el espacio de trabajo, donde el robot realizara su tarea, se encuentran distribuidos una serie de obstáculos definidos como un conjunto de objetos rígidos B y que se encuentran distribuidos por el espacio de configuraciones C.

$$B = \{b_1, b_2, b_3, \dots b_q\} \tag{23}$$

Todo el conjunto de configuraciones del espacio C ocupadas por un obstáculo se define por $bi(q)$, de esa forma el subconjunto de configuraciones de C, que definen el espacio libre de obstáculos viene dado por:

$$C_l = \{q \in C / \ R(q) \cap \left(\bigcup_{i=1}^{q} b_i(q) \right) = \emptyset \} \tag{24}$$

Según esta metodología, el problema de la planificación, como se ha definido, queda transformado en la búsqueda de una sucesión de posiciones q siendo la primera de ellas la posición actual del robot qa y la ultima de esta sucesión la posición objetivo qf. Todas estas posiciones deben pertenecer al subconjunto C_l definido en la expresión (24). Es decir, una ruta Qr que conecta q_a con q_f es:

$$Q_r = \{ q_a, \dots, q_f / \ q_i \in C_l \} \tag{25}$$

La especificación de este conjunto Qr, implica la construcción de una función de ruta definida de la siguiente forma:

$$\tau: [0,1] \rightarrow C_l \tag{26}$$

Tal que:

$$\tau(0) = q_a \qquad\qquad \tau(1) = q_f \tag{27}$$

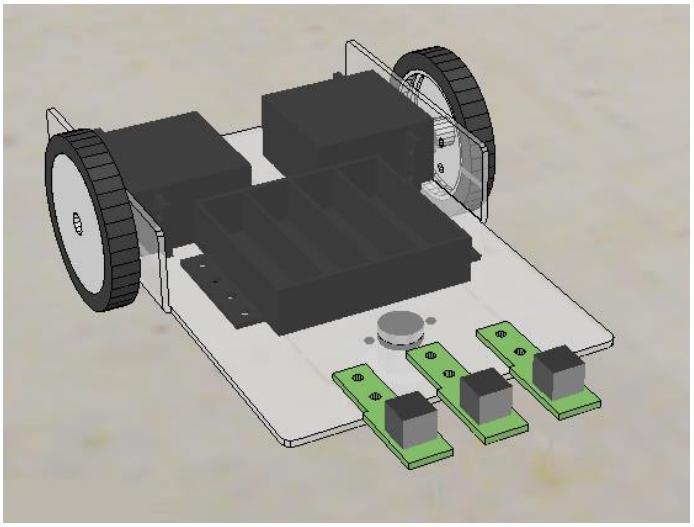
Además de la restricción (26), se le exige a la función τ, teniendo en cuenta la posición de un robot omnidireccional y la continuidad. Este concepto se refleja en la siguiente expresión

$$\lim_{s \to s_0} |\tau(s), \tau(s_0)| = 0 \tag{28}$$

7 AGENTE ROBÓTICO

Este agente robótico es de tipo diferencial, es decir no tiene ruedas directrices, su dirección está dada por la velocidad relativa de las ruedas traseras. La rueda delantera, es una rueda de castor que mantiene el balance del robot móvil (figura 27).

Figura 27 Robot Diferencial



Fuente: Autor

Una característica especial de los robots móviles es su naturaleza con restricciones no holónomicas [23]. Como se describe en los métodos de planificación, la posición de cada punto de un robot puede encontrarse en función de los parámetros de q. El espacio de configuraciones C es el conjunto de todas las posibles configuraciones de q.

Entonces si consideramos las restricciones, estas en su forma más general, consistirán en una relación entre las variables de configuración. Dado que estas suelen identificarse con posiciones y orientaciones de los sólidos rígidos que tiene el robot, las derivadas de estas variables con respecto al tiempo tienen un sentido físico de velocidades. La naturaleza de las restricciones cinemáticas y dinámicas pueden encontrarse mediante expresiones como:

$$F_j(\dot{q_1}, \dot{q_2}, \dots, \dot{q_n}, q_1, q_2, \dots, q_n, t) = 0 \quad ; j = 1, \dots, m \tag{29}$$

Donde n es el número de variables en el espacio de configuraciones y m es el número de restricciones. Si estas restricciones son integrables entonces se dice que son holónomas. Si en cambio no lo son, se denominan restricciones no holónomas.

Si las restricciones son holónomas, la ecuación (29) podría integrarse y obtenerse una expresión del tipo:

$$G_j(\, q_1, q_2, \dots, q_n, t) = 0 \quad ; j = 1, \dots, m \tag{30}$$

A partir de la ecuación (24) puede ser posible encontrar m ecuaciones que expresen m variables en función de las restantes $(n - m)$

$$q_j = g, (\, q_{m+1}, q_{m+2}, \dots, q_n, t) = 0 \quad ; j = 1, \dots, m \tag{31}$$

Solo es necesario definir el conjunto de parámetros $(\, q_{m+1}, q_{m+2}, \dots, q_n, t)$ para encontrar las restantes variables y asi obtener la configuración del robot. Esto implica que se puede reducir la dimensión del espacio de configuraciones en m variables. De esta forma se reduce el problema a un menor número de variables, y al ser todas independientes entre sí, el planificador puede generar cualquier tipo de trayectoria, cosa que no es este caso ya que este robot móvil con restricciones no holónomas.

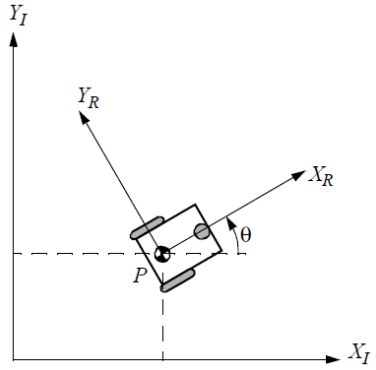
Cuando las restricciones son no holónomas, la dimensión no puede reducirse y además impone condiciones al movimiento. Es decir, si se quiere unir dos configuraciones dentro de C (sin obstáculos), las restricciones no holónomas limitaran el conjunto de trayectorias posibles, dado que la variación de cada una de las variables de configuración depende de las demás.

Las restricciones no holónomas no reducen la dimensión de C, pero si lo hacen en el espacio de las velocidades.

7.1 Modelamiento cinemático

Para el modelo cinemático se debe tener en cuenta cada rueda individualmente ya que contribuye al movimiento del robot y, al mismo tiempo, impone restricciones al movimiento. Las ruedas esta unidas en base a la geometría del chasis del robot y por lo tanto se combinan para formar restricciones en el movimiento de todo el chasis[24]. Pero la fuerza y las restricciones de cada rueda se deben expresar con respecto a un claro y coherente marco de referencia. Esto particularmente es importante en la robótica móvil debido a su autocontrol y naturaleza del robot; un mapeo claro dentro los marcos de referencia global y local es muy necesario.

Figura 28 El marco de referencia global y local del robot



Fuente: Autonomous Mobile Robots. Roland Siegwart, Illah R. Nourbakhsh 2004 .p 49

7.1.1 Representación de la posición del robot

A lo largo de este análisis se modela el robot como un cuerpo rígido sobre ruedas, operando sobre un plano horizontal. La dimensionalidad total del chasis es de tres, dos para el plano y uno para la orientación a lo largo del eje vertical, que es ortogonal al plano.

Para especificar la posición del robot en el plano se establece una relación entre el marco de referencia global y el marco de referencia local del robot como en la figura anterior. Los ejes XI y Yi definen una base inercial arbitraria en el plano como el marco de referencia global de algún origen O: $\{XI, YI\}$. Para especificar la posición del robot, se toma el punto P del chasis del robot como su punto de referencia.

Las bases $\{XR, YR\}$ definen dos ejes relacionados a p en el chasis del robot y por esto es marco de referencia local del robot. La posición de P en el marco de referencia local, está especificado por las coordenadas x e y, y la diferencia angular entre los marcos de referencia global y local está dada por θ. Se puede describir la postura del robot como un vector con estos tres elementos[25].

Se debe tener en cuenta el uso del subíndice I para aclarar la base de esta postura como referencia global.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{32}$$

Para describir el movimiento del robot en términos de movimientos de componentes, será necesario mapear a lo largo de los ejes el marco de referencia global para el movimiento de los ejes del robot en el marco local de referencia. Por supuesto, la

asignación es una función de la postura actual del robot. Este mapeo se logra utilizando la matriz de rotación ortogonal:

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (33)$$

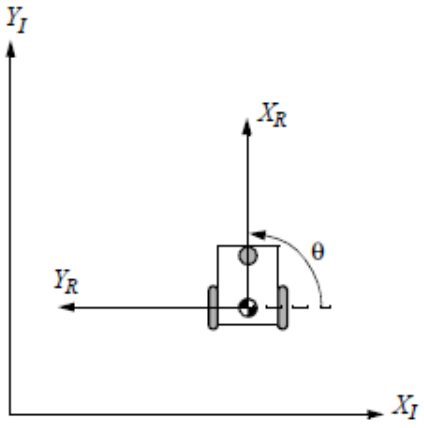
Esta matriz puede usarse para mapear el movimiento en el marco de referencia global $\{X_I, Y_I\}$ para moverse en términos del marco de referencia local $\{X_R, Y_R\}$. Esta operación es denotada por $(\theta)\xi_I$ porque el cálculo de esta operación depende del valor de:

$$\xi_R = R\left(\frac{\pi}{2}\right)\xi_I \quad (34)$$

Por ejemplo se puede considerar la figura 24 para este robot ya que $\theta = \frac{\pi}{2}$ y fácilmente se puede calcular la matriz de rotación instantánea R:

$$R\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (35)$$

Figura 29 Robot móvil alineado con un eje global



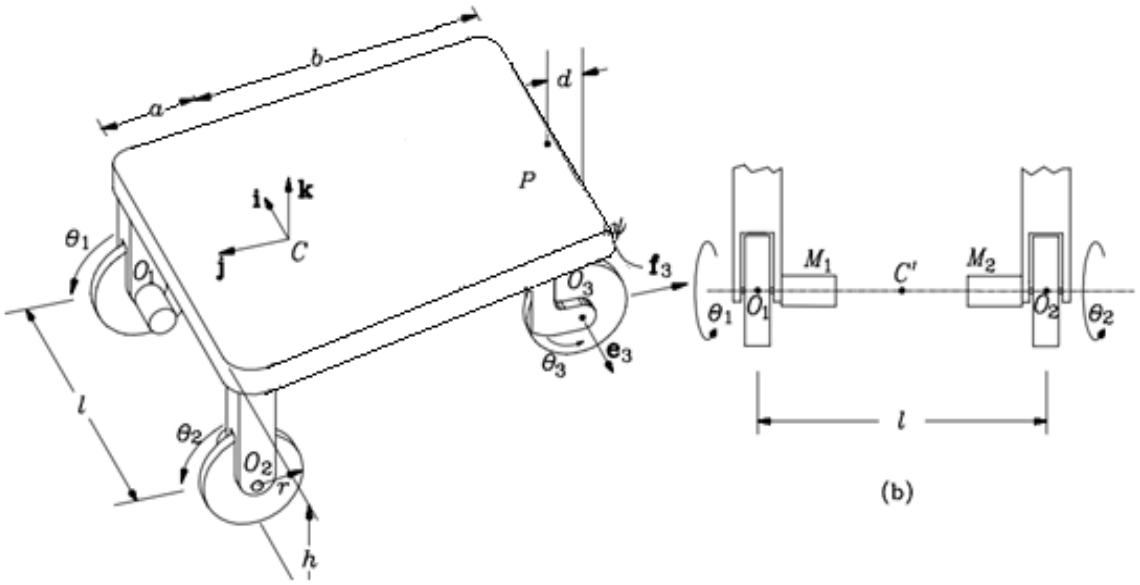
Fuente: Autonomous Mobile Robots. Roland Siegwart, Illah R. Nourbakhsh 2004 .p 50

Dada una cierta velocidad $(\dot{x}, \dot{y}, \dot{\theta})$ en el marco referencial global se puede calcular los componentes de movimiento a lo largo de los ejes locales del robot X_R y Y_R [25]. En este caso, ambos para el ángulo especifico del robot, el movimiento a lo largo X_R es igual a $\dot{y}$ y el movimiento a lo largo Y_R es -$\dot{x}$:

$$\dot{\xi}_R = R(\frac{\dot{\pi}}{2})\xi_I = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \quad (36)$$

7.1.2 Representación matricial del modelo cinemático diferencial directa.

Figura 30 Esquema del Robot Diferencial



Fuente: Modelo cinemático dinámico del mini robot móvil ricimaf, 2012.p 51

El modelo cinemático del robot se establece a partir de los parámetros anteriormente visualizados en la figura, estos son:

a - Distancia del centro del chasis al centro de las ruedas
b - Distancia entre el centro del chasis y el centro del soporte de la rueda castor.
d - Distancia entre el centro del soporte de la rueda loca y el centro de la rueda loca.
l - Distancia entre los centros de las ruedas traseras.
h - Alturas del chasis sobre el terreno.
$r1$ - Radio de las ruedas traseras.
$r2$ - Radio de la rueda castor.
Ψ - Ángulo de giro del chasis con respecto al sistema de coordenada en el chasis del robot $\{i, j\}$.
$\theta i\,, \theta d\,, \theta l$ - Ángulos de giro de las ruedas del sistema sobre su eje respectivamente.[26]

El modelo diferencial es:

$$\dot{p} = J(p)\dot{q} \tag{37}$$

Donde $J\,(p)$ es una matriz de derivadas parciales o jacobiana, esta se puede expresar en forma matricial y de esta manera obtener los elementos de la matriz jacobiana.

$$J\,(p) = \begin{vmatrix} -\sin\Psi & 0 \\ \cos\Psi & 0 \\ 0 & 1 \end{vmatrix} \tag{38}$$

Para obtener las ecuaciones cinemáticas diferenciales inversas es necesario invertir la matriz anterior, pero esta es una matriz singular por tanto es necesario obtener su pseudoinversa a partir de la ecuación (38), así que se multiplica por la transpuesta de la matriz Jacobiana en ambos lados de la ecuación:

$$|J(p)|^{T}\,\dot{p} = |J(p)|\,\dot{q}\,|J(p)|^{T} \tag{39}$$

Al despejar $\dot{q}$ y expresar nuevamente en forma matricial el modelo cinemático diferencial inverso del robot queda:

$$\begin{vmatrix} v \\ w \end{vmatrix} = \begin{vmatrix} -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \dot{x} \\ \dot{y} \\ \Psi \end{vmatrix} \tag{40}$$

Las velocidades angulares y lineales de las ruedas traseras están definidas como:

$$v = \frac{(vi + vd)}{2} = \frac{(\theta i + \theta d)}{2}\frac{r}{1} \tag{41}$$

$$w = \frac{(vi - vd)}{l} = \frac{(\theta i - \theta d)}{l}\frac{r}{1} \tag{42}$$

7.2 Modelamiento dinámico

El modelamiento dinámico del robot, representado por las ecuaciones del movimiento del agente robótico se realizó usando la ecuación (43), esta es la formulación de Euler y LaGrange.

$$\begin{vmatrix} \dot{x} \\ \dot{y} \\ \Psi \end{vmatrix} = \begin{vmatrix} -\sin\Psi & 0 \\ \cos\Psi & 0 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} v \\ w \end{vmatrix} \tag{43}$$

La energía total E de un robot de n grados de libertad (DOF) es la suma de las energías cinéticas y potenciales[23].

$$\mathcal{E}(q, \dot{q}) = \mathcal{K}(q, \dot{q}) + \mathcal{U}(q) \tag{44}$$

Se asume que la energía potencial se debe solo a las fuerzas conservadoras, como la energía gravitacional[25], principalmente se determina el Lagrangeano del sistema como se expresó en la ecuación (42), la energía potencial del robot se anula debido a que la altura es cero, porque se considera el movimiento sobre un terreno plano y horizontal y la energía cinética depende de las masas y las velocidades.

- Masa de ruedas traseras energizadas = $mi = md = mr$
- Masa de la rueda loca $ml0$.
- Masa del chasis mc

$$L(q, \dot{q}) = \mathcal{K}(q, \dot{q}) - \mathcal{U}(q) \tag{45}$$

Las velocidades que adquieren las masas anteriormente mencionadas son:

- Velocidad lineal de las ruedas traseras y rueda loca $vi,\ vd, vl0$**.**
- Velocidad rotacional de las ruedas alrededor de su eje $\dot{\theta}i\ , \dot{\theta}d, (\theta l0)\dot{}$.
- Velocidad Angular de las ruedas $wi, wd, wl0$.
- Velocidad lineal del centro del chasis o plataforma vc**.**
- Velocidad angular del centro del chasis W**.**

Las velocidades lineales de los centros de las ruedas son:

$$\mathrm{vi} = \mathrm{r}\dot{\theta}\mathrm{i};\ \mathrm{vd} = \mathrm{r}\dot{\theta}\mathrm{d};\quad \mathrm{vl0} = -\mathrm{r}\dot{\theta}\mathrm{lo} \tag{46}$$

La velocidad del centro del chasis es:

$$\sum_{\mathrm{K}=1}^{2}\left[\mathrm{Vck} = \mathrm{r}\dot{\theta}\mathrm{k} + (\mathrm{C} - \mathrm{Ok})\dot{\Psi}\right] \tag{47}$$

Restando los términos de la ecuación (47)

$$0 = \mathrm{r}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right) - (\mathrm{O1} - \mathrm{O2})\dot{\Psi} \tag{48}$$

Se obtiene la velocidad angular del centro del chasis w.

$$\mathrm{W} = \dot{\Psi} = \frac{\mathrm{r}}{\mathrm{l}}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right) \tag{49}$$

Si se suman los términos de la ecuación (47)

$$\mathrm{Vc} = \frac{\mathrm{r}}{2}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right) + \mathrm{a}\dot{\Psi} \tag{50}$$

La velocidad lineal del centro del chasis Vc es:

$$\mathrm{Vc} = \mathrm{a}\frac{\mathrm{r}}{\mathrm{l}}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\mathrm{i} + \frac{\mathrm{r}}{2}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\mathrm{j} \tag{51}$$

Las ecuaciones (49) y (50) constituyen la cinemática directa diferencial del robot.

La velocidad angular de las ruedas energizadas.

$$\mathrm{Wi} = \dot{\theta}\mathrm{i}\mathbf{i} + \dot{\Psi}\mathbf{k} = \dot{\theta}\mathrm{i}\mathbf{i} + \frac{\mathrm{r}}{\mathrm{l}}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\mathbf{k} \tag{52}$$

$$\mathrm{Wd} = \dot{\theta}\mathrm{d}\mathbf{i} + \dot{\Psi}\mathbf{k} = \dot{\theta}\mathrm{d}\mathbf{i} + \frac{\mathrm{r}}{\mathrm{l}}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\mathbf{k} \tag{53}$$

La velocidad angular de la rueda castor es:

$$\mathrm{Wlo} = -\dot{\theta}\mathrm{lo}\mathbf{f3} + (\mathrm{Wc} + \dot{\sigma}) \tag{54}$$

$$\mathrm{Wlo} = -\dot{\theta}\mathrm{lo}\mathbf{f3}\mathrm{f3} + \frac{\mathrm{r}}{\mathrm{l}}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right) + \dot{\sigma} \tag{55}$$

Finalmente se deben calcular las energías cinéticas correspondientes:

Para esto se parte de la ecuación principal

$$\mathrm{k} = \mathrm{k_T} + \mathrm{k_R} + \mathrm{k_C} \tag{56}$$

Donde KT es la energía trasnacional, KR es la energía rotacional y Kc es la aportación de la energía cinética del centro de masas de gravedad de todo el robot al punto central, ya que este punto puede no coincidir donde ocurre la intersección del eje de simetría del sistema con el eje horizontal el cual giran las dos ruedas y la velocidad queda definida por las velocidades debidas a las restricciones[23].

$$\mathrm{KT} = \frac{1}{2}(\mathrm{M} + \mathrm{m_C})\mathrm{Vc}^2 \tag{57}$$

Donde M es la suma total de las masas de las ruedas, $\mathrm{m_C}$ es la masa del chasis y Vc es la velocidad al centro del chasis:

$$\mathrm{k_R} = \left[\mathrm{I_{ri}W_i^2} + \mathrm{I_{rd}W_d^2} + \mathrm{I_{rlo}W_{lo}^2}\right] + \frac{1}{2}\mathrm{I_c}\Psi^2 \tag{58}$$

El momento de inercia está definido por $\boldsymbol{I_r}$, $\boldsymbol{W}$ es la velocidad angular de los ejes horizontales de las ruedas y la $\boldsymbol{r}$ el radio de cada rueda.

$$\mathrm{I_C} = m_\mathrm{C} z \tag{59}$$

Partiendo del momento de inercia en el centro del chasis, ecuación (59), con respecto a un eje vertical que se levanta en la intersección del eje de simetría y eje de las ruedas motrices.
Usando las restricciones cinemáticas se obtiene la contribución de la energía cinética del centro de la masa dad por el producto de la masa del chasis m_C por lo que z es el producto de las velocidades de restricción cinemática.

$$z = \frac{r}{l}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)(\dot{x}sin\Psi - \dot{y}cos\Psi) \tag{60}$$

$$\mathrm{k_C} = \frac{1}{2}\mathrm{m_c}z = \frac{1}{2}\mathrm{m_c}\frac{r}{l}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)(\dot{x}sin\Psi - \dot{y}cos\Psi) \tag{61}$$

$$\mathrm{k_C} = \mathrm{m_c}\frac{r}{2l}\left[(\dot{x}sin\Psi - \dot{y}cos\Psi)\dot{\theta}\mathrm{i} - (\dot{x}sin\Psi - \dot{y}cos\Psi)\dot{\theta}\mathrm{d}\right] \tag{62}$$

Para que las expresiones queden en función de las velocidades de la rueda, se usa la ecuación (51) en (57) quedando de la siguiente manera:

$$\mathrm{k_R} = M(\dot{\mathrm{x}}_\mathrm{c}^2 + \dot{\mathrm{y}}_\mathrm{c}^2) + \mathrm{m_c r}^2\left[\left(\frac{a}{l}\right)^2\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2 + \frac{1}{4}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2\right] \tag{63}$$

Usando las ecuaciones (49) (52) (53) y simplificando:

$$\mathrm{k_R} = \frac{1}{2}\mathrm{I_r}\left[(\dot{\theta}\mathrm{i}^2 + \dot{\theta}\mathrm{d}^2) + 2\left(\frac{r}{l}\right)^2\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2\right] + \tag{64}$$
$$\frac{1}{2}\mathrm{I_{rlo}}\left[\left(\frac{a+b}{l}\right)^2 cos^2\Psi\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2 + \left(\frac{1}{4}\right)sin^2\Psi\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2 + \left(\frac{a+b}{l}\right)sin\Psi cos\Psi\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right)\right] + \frac{r^2}{2l^2}\mathrm{I_c}\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)^2$$

En robótica móvil la ecuación de Euler-Lagrange es general para cualquier robot y está en función del Lagrangeano y de las coordenadas generalizadas del robot:

$$\tau = \frac{\partial}{\partial t}\left[\frac{\partial}{\partial \dot{q}}(L(q,\dot{q})\right] - \frac{\partial}{\partial q}[(L(q,\dot{q})] \tag{65}$$

Derivando las expresiones de la energía cinética con respecto a las velocidades de las ruedas del agente robótico, se obtuvo esto mediante la ecuación de Euler – LaGrange, recordando que la energía potencial es constante cuando el robot móvil se está desplazando.

$$\frac{d}{dt}\left[\frac{\partial}{\partial \dot{q}}[K] = \frac{d}{dt}\left[\frac{\partial}{\partial \dot{q}}(\mathrm{k_T} + \mathrm{k_R} + \mathrm{k_C})\right]\right] \tag{66}$$

$$\mathrm{T}_1 = \frac{\partial \mathrm{k}_\mathrm{T}}{\partial \dot{q}} = 2\left(\dot{x}_2 + \dot{y}_2\right) + m_c r^2 \left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right) \tag{67}$$

$$\mathrm{T}_2 = \frac{\partial \mathrm{k}_\mathrm{R}}{\partial \dot{q}} = \mathrm{I}_\mathrm{r}\left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right) + \mathrm{I}_\mathrm{rlo}\left\{\frac{1}{2} sin^2\Psi\left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right) + \left(-\frac{1}{2}\left(\frac{a+b}{l}\right)\right) sin\Psi cos\Psi\left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right) + \left(\frac{a+b}{l}\right) sin\Psi cos\Psi\left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\right\} \tag{68}$$

$$\mathrm{T}_3 = \frac{\partial \mathrm{k}_\mathrm{C}}{\partial \dot{q}} = \mathrm{m}_\mathrm{c} \frac{r}{l}\left[(sin\Psi - cos\Psi)\dot{\theta}\mathrm{i} + (cos\Psi - sin\Psi)\dot{\theta}\mathrm{d}\right] \tag{69}$$

El primer término de la ecuación de Euler-LaGrange es la diferenciación de los términos.

$$\frac{d}{dt}\frac{\partial(\mathrm{k}_\mathrm{T} + \mathrm{k}_\mathrm{R} + \mathrm{k}_\mathrm{C})}{\partial \dot{q}} \tag{70}$$

Se deriva con respecto al tiempo las ecuaciones (67) (68) (69), y se representan los términos de estas de forma matricial:

$$M(q)\ddot{q} = \begin{vmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{vmatrix} \tag{71}$$

$$\dot{M}(q)\dot{q} = \begin{vmatrix} m_{15} \\ m_{25} \\ m_{35} \\ m_{45} \end{vmatrix} \qquad q = \begin{vmatrix} x_c \\ y_c \\ \theta_i \\ \theta_d \end{vmatrix} \tag{72}$$

Donde estos valores quedan indicados como coeficientes no lineales del vector de aceleración generalizado θ y de la velocidad lineal v.

$$m_{11} = 2\ddot{x}_c; \quad m_{12} = 2\ddot{y}_c; \quad m_{13} = \mathrm{m}_\mathrm{c} r^2\, \ddot{\theta}_i\,; \quad m_{14} = \mathrm{m}_\mathrm{c} r^2\, \ddot{\theta}_d\,; \quad m_{15} = 0 \tag{73}$$

$$m_{21} = 0; \quad m_{22} = 0; \tag{74}$$

$$m_{23} = \left\{\mathrm{I}_\mathrm{r} + \mathrm{I}_\mathrm{rlo}\left[\left(\frac{a+b}{l}\right) sin\Psi cos\Psi + \left(\frac{1}{2}\right) sin^2\Psi\right]\right\}\ddot{\theta}$$

$$m_{24} = \left\{-\mathrm{I_r} + \mathrm{I_{rlo}}\left[-\left(\frac{a+b}{l}\right) sin\Psi cos\Psi + \left(\frac{1}{2}\right) sin^2\Psi\right]\right\} \ddot{\theta}_d \, ;$$

$$m_{25} = \mathrm{I_{rlo}}\left(\frac{a+b}{l}\right)(cos^2\Psi - sin^2\Psi) + \left(\dot{\theta}\mathrm{i} - \dot{\theta}\mathrm{d}\right)\dot{\Psi} + \mathrm{I_{rlo}} sin\Psi cos\Psi\left(\dot{\theta}\mathrm{i} + \dot{\theta}\mathrm{d}\right)\dot{\Psi}$$

$$m_{31} = 0; \qquad m_{32} = 0; \quad m_{33} = -\mathrm{m_c}\frac{r}{l}(cos\Psi - sin\Psi)\ddot{\theta}_i; \qquad (75)$$

$$m_{33} = -\mathrm{m_c}\frac{r}{l}(cos\Psi - sin\Psi)\ddot{\theta}_d;$$

$$m_{45} = 0 \qquad (76)$$

Se calculan las derivadas parciales de la enérgica cinética con respecto al vector posición.

$$\frac{1}{2}\left[\frac{\partial}{\partial t}(\mathrm{k_T} + \mathrm{k_R} + \mathrm{k_C})\right] \qquad (77)$$

$$\frac{\partial \mathrm{k_T}}{\partial \dot{q}} = 0; \qquad \frac{\partial \mathrm{k_R}}{\partial \dot{q}} = 0; \qquad \frac{\partial \mathrm{k_C}}{\partial \dot{q}} = 0; \qquad (78)$$

Por último se toman los coeficientes de Euler – LaGrange y las restricciones no holonómicas y se representan de la siguiente manera:

$$\tau = M(q)\ddot{q} + C(q)\ddot{q} + A^T(q)\lambda\boldsymbol{\lambda} \qquad (79)$$

8 ANÁLISIS Y RESULTADOS

8.1 Simulación en V-rep

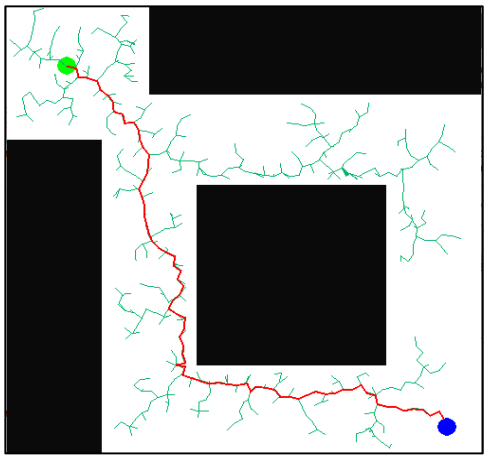
Como resultados se presentan dos mapas, los cuales, tienen una configuración diferente, es decir, la ubicación de los obstáculos cambia, así como también los puntos de configuración inicial y final.
Una vez que han sido generados los trayectos de cada uno por el algoritmo RRT y analizados para el suavizado del mismo, se implementa cada uno de los mapas, con las mismas dimensiones, en el software de simulación V-rep.

Las dos trayectorias fueron elegidas como las mejores, entre varias simulaciones con el mismo entorno y la misma configuración respectivamente, y el único criterio de elección fue la ruta más corta.

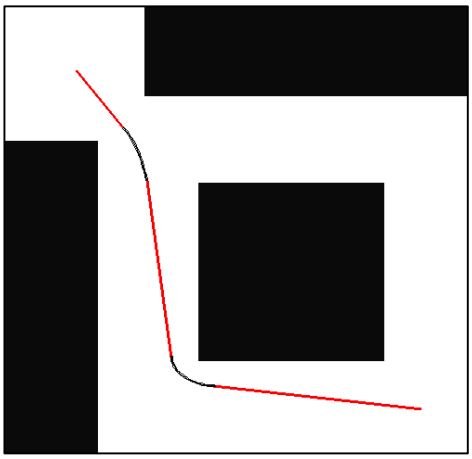
Para realizar el modelamiento de los mapas en V-rep, se debe tener en cuenta una tercera dimensión, pero está no es de gran importancia, ya que, al implementar el mapa en Python se realizó en el plano 2d, como no es de interés esta característica, se le dio un valor de 1 para el modelamiento de los obstáculos, es decir los obstáculos tendrán una altura, cosa que no se evidencia en los mapas iniciales.

El primer mapa que se presenta, es un entorno en el que se evidencian tres obstáculos, dos de forma rectangular en los límites del plano superior e izquierdo y uno cuadrado en el centro. Ya que el programa para generar la trayectoria permite ubicar el punto inicial y final en cualquier espacio libre de obstáculos, en este caso se ubicó el punto inicial en la parte superior izquierda del mapa, y el punto final en la parte inferior derecha, es decir a los extremos del mapa. Una vez ubicados los puntos el algoritmo empieza a construir el árbol, hasta que encuentra la trayectoria señalada con el color rojo, como se puede evidenciar en la figura 31a.
Una vez encontrada la trayectoria se realiza el suavizado, tal y como se describió en la sección 0, en la figura 31b, podemos observar el resultado de este proceso e incluso con color negro la curva que se obtiene. Las figuras 31 c, d, e y f nos muestran el comportamiento del robot sobre la trayectoria generada, y se evidencia el recorrido del robot en diferentes puntos de la trayectoria.
El segundo mapa, es un entorno un poco más complejo, ya que los cuatro obstáculos ubicados uniformemente en el centro del espacio, se separan por un espacio mínimo, pero que puede atravesar el robot, así, ponemos a prueba el algoritmo, y aunque no están evidenciadas las demás trayectorias que género, si logro crear una trayectoria a través de los obstáculos (figura 32).

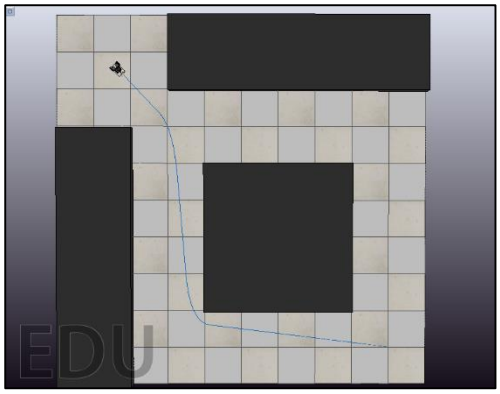
Figura 31 Mapa 1



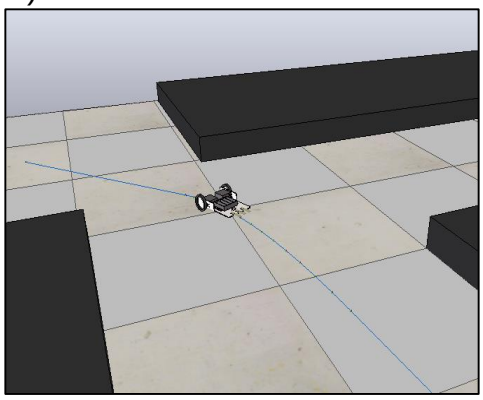
a)



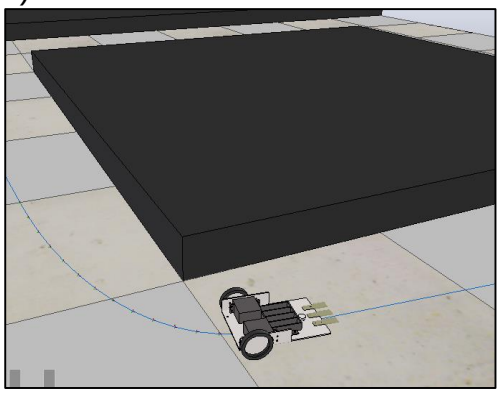
b)



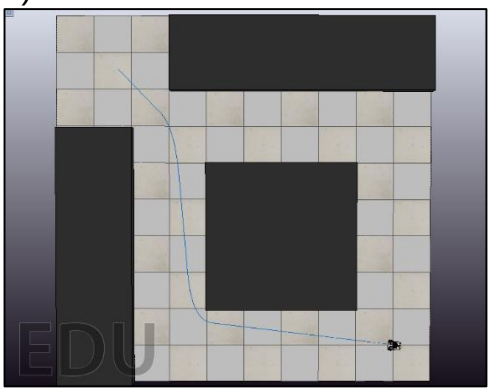
c)



d)



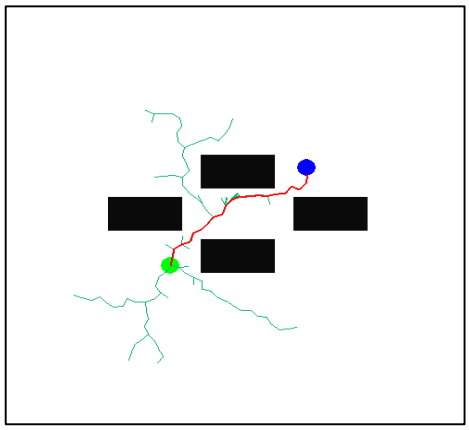
e)



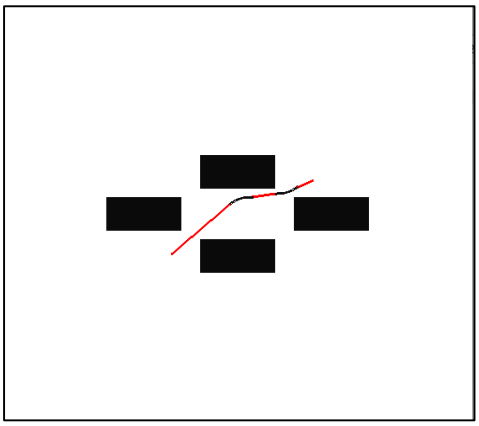
f)

Fuente: Autor

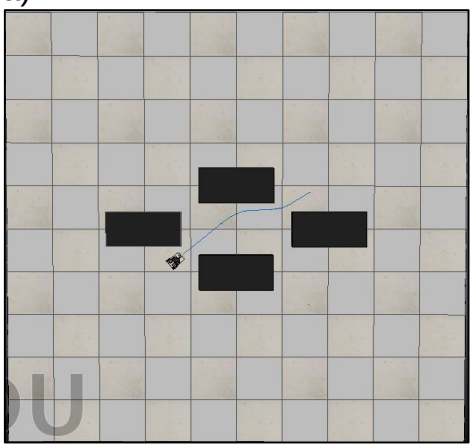
Figura 32 Mapa 2



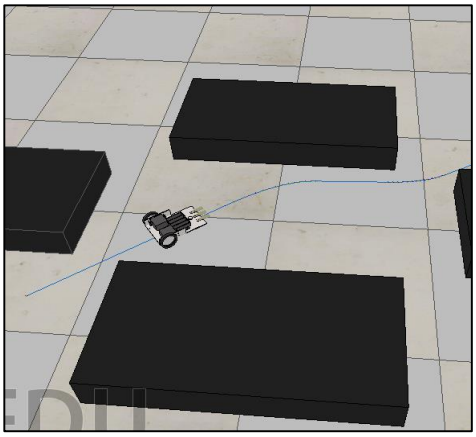
a)



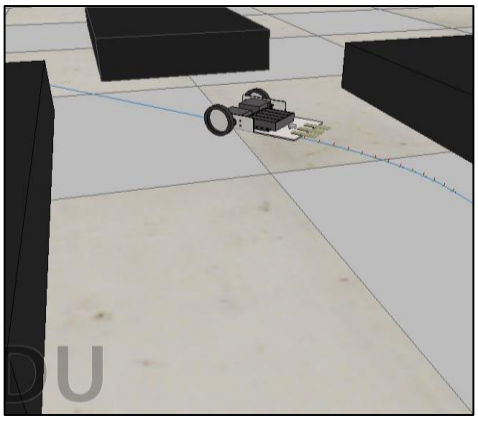
b)



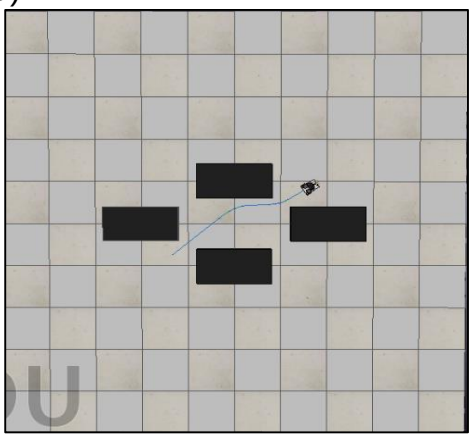
c)



d)



e)



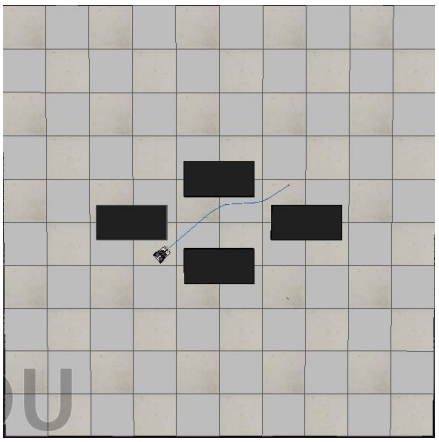
f)

Fuente: Autor

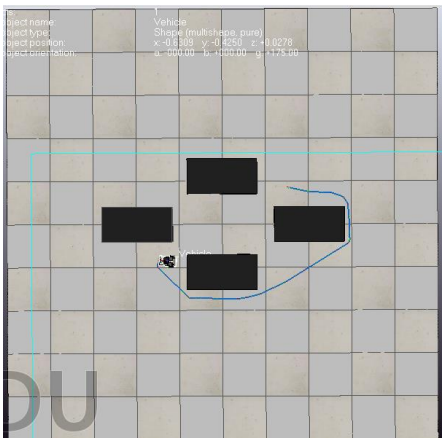
8.2 Tiempo de recorrido y comparación con OMPL

OMPL (Open Motion Planning Library), consiste en un conjunto de algoritmos de planificación de movimientos basados en el muestreo, como, PRM's, RRT, EST (Enhanced Transmission Selection), SBL (Sparse Bayesian Learning), KPIECE (Kinodynamic Motion Planning by Interior-Exterior Cell Exploration), SyCLoP(synergistic combination of layers of planning)[27] [28]. El contenido de la biblioteca se limita a estos algoritmos, asi, que no hay especificación del entorno, ni detección de obstáculos o visualización.

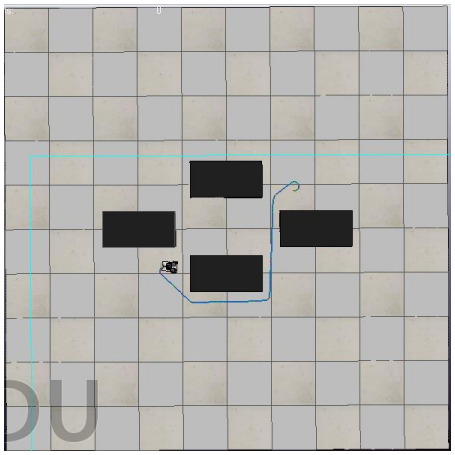
Figura 33 Mapa de comparación 1



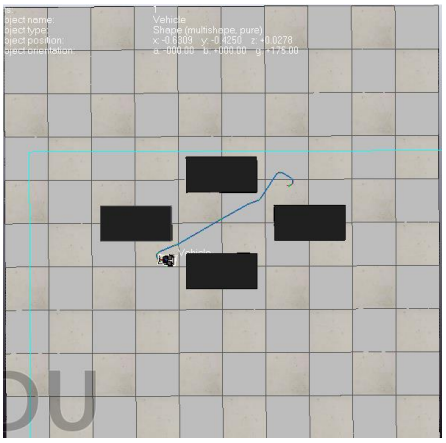
a) trayectoria generada por Autor



b) Primer Trayectoria generada por OMPI



c)) Segunda trayectoria generada por OMPI



c) Tercera trayectoria generada por OMPI

Fuente: Autor

La figura 33a muestra la trayectoria generada en la sección 8.1, las trayectorias de las figuras 33b, c y d, son trayectorias generadas por el simulador V-rep, que usa la biblioteca OMPL para hacer el cálculo del trayecto.

En esta sección mostramos una comparación del tiempo que toma el robot en recorrer la trayectoria generada por la propuesta en este proyecto, y la generada por el propio simulador, obviamente ubicando los puntos de configuración inicial y final iguales.

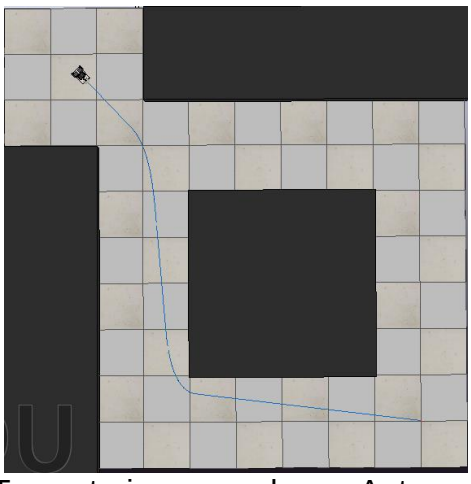
Tabla 2 Tiempo de recorrido del Robot en Mapa 1

TRAYECTORIA	TIEMPO (seg)
Generada por Autor	18
Primera generada por OMPL	39
Segunda generada por OMPL	28
Tercera generada por OMPL	17

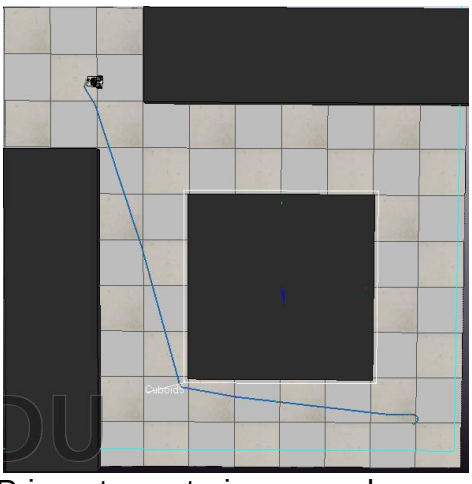
Fuente: Autor

Se decidió generar 3 trayectorias por OMPL (para generarlas ver anexo ##), ya que la trayectoria generada por autor, fue elegida entre varias simulaciones como la mejor, entonces la comparación se hace de una manera más justa por decirlo así, como podemos ver en la tabla, el menor tiempo que recorrió el robot del punto inicial al final evitando obstáculos, fue la tercera trayectoria generada por OMPL(figura##), pero la trayectoria generada por autor solo pierde un segundo en comparación, con la tercera, y es óptima en comparación con la primera y segunda generadas por OMPL, que tuvieron un tiempo muy alto y la trayectoria fue muy extensa.

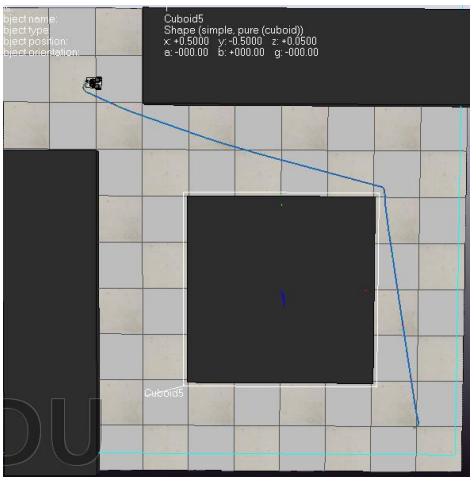
Figura 34 Mapa de comparación 2



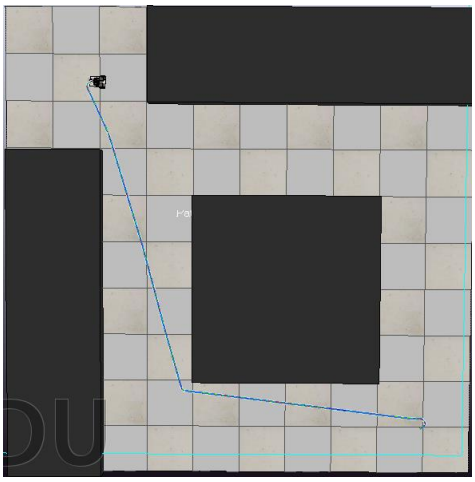
a) Trayectoria generada por Autor



b) Primer trayectoria generada por OMPI



c) Segunda trayectoria generada por OMPI



d) Tercera trayectoria generada por OMPI

Fuente: Autor

En este mapa (figura 34), podemos evidenciar que, al estar ubicados los puntos inicial y final en esa posición, a los extremos del mapa, solo existen dos espacios en los que se pueden generar las trayectorias, por la parte izquierda o derecha, así, el tiempo de recorrido no debería ser de gran diferencia entre los trayectos generados, pues la distancia es aproximadamente la misma.

La tabla 3, evidencia lo anteriormente descrito. Como se puede observar los tiempos de recorrido son muy cercanos, ya que el mapa se presta para esto. Aun así, la trayectoria que más tiempo ocupo fue la trayectoria generada por la propuesta de este proyecto, pero no es un dato muy importante, ya que la diferencia fue mínima en comparación con las trayectorias generadas por OMPI.

Tabla 3 Tiempo recorrido del Robot en Mapa 2

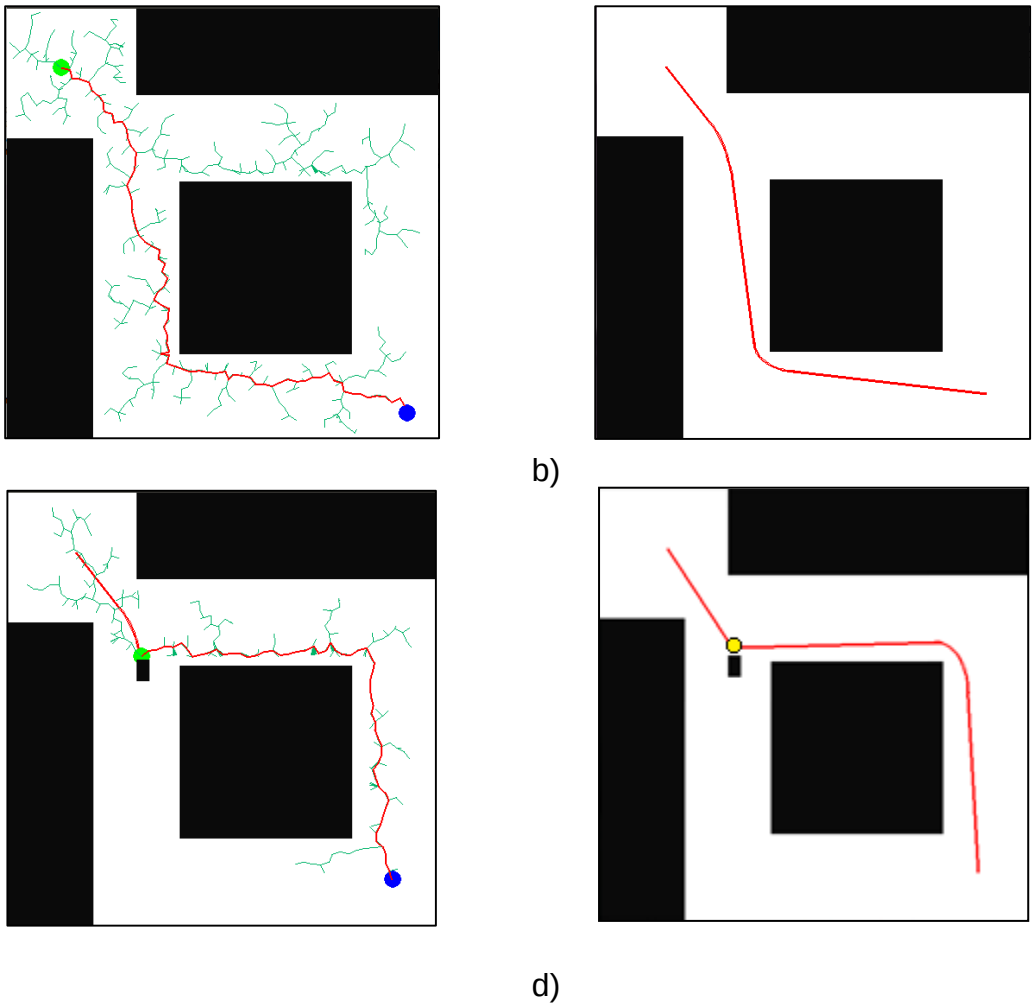
TRAYECTORIA	TIEMPO (seg)
Generada por Autor	65
Primera generada por OMPI	62
Segunda generada por OMPI	63
Tercera generada por OMPI	61

Fuente: Autor

8.3 Comportamiento en entornos dinámicos.

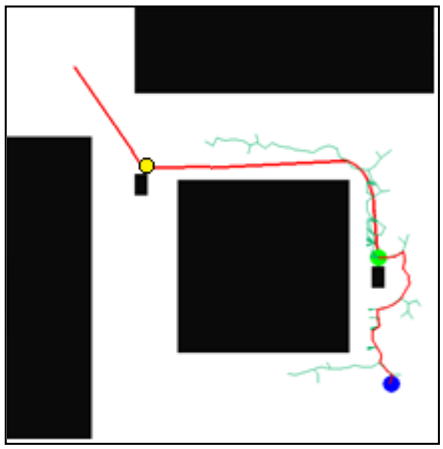
Para crear el entorno dinámico, se suponen obstáculos en los mapas que no se contemplaron inicialmente , pero estos no tendrán una dimensión exacta, es decir, solo el robot será capaz de sensar si existe o no obstáculos mientras recorre la trayectoria, ya sea por medio de sensores de proximidad o sensores de ultrasonido, y al no conocer la dimensión de tal obstáculo, el algoritmo propuesto supone un obstáculo frente al robot con las mismas dimensiones de área del robot en el punto donde encontró obstáculo.

Figura 35 Comportamiento en un entorno dinámico mapa 1.

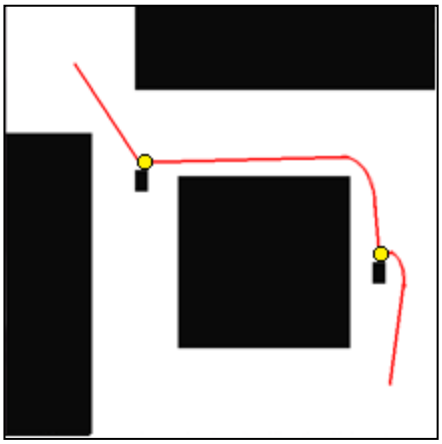


a) b)

c) d)



e)



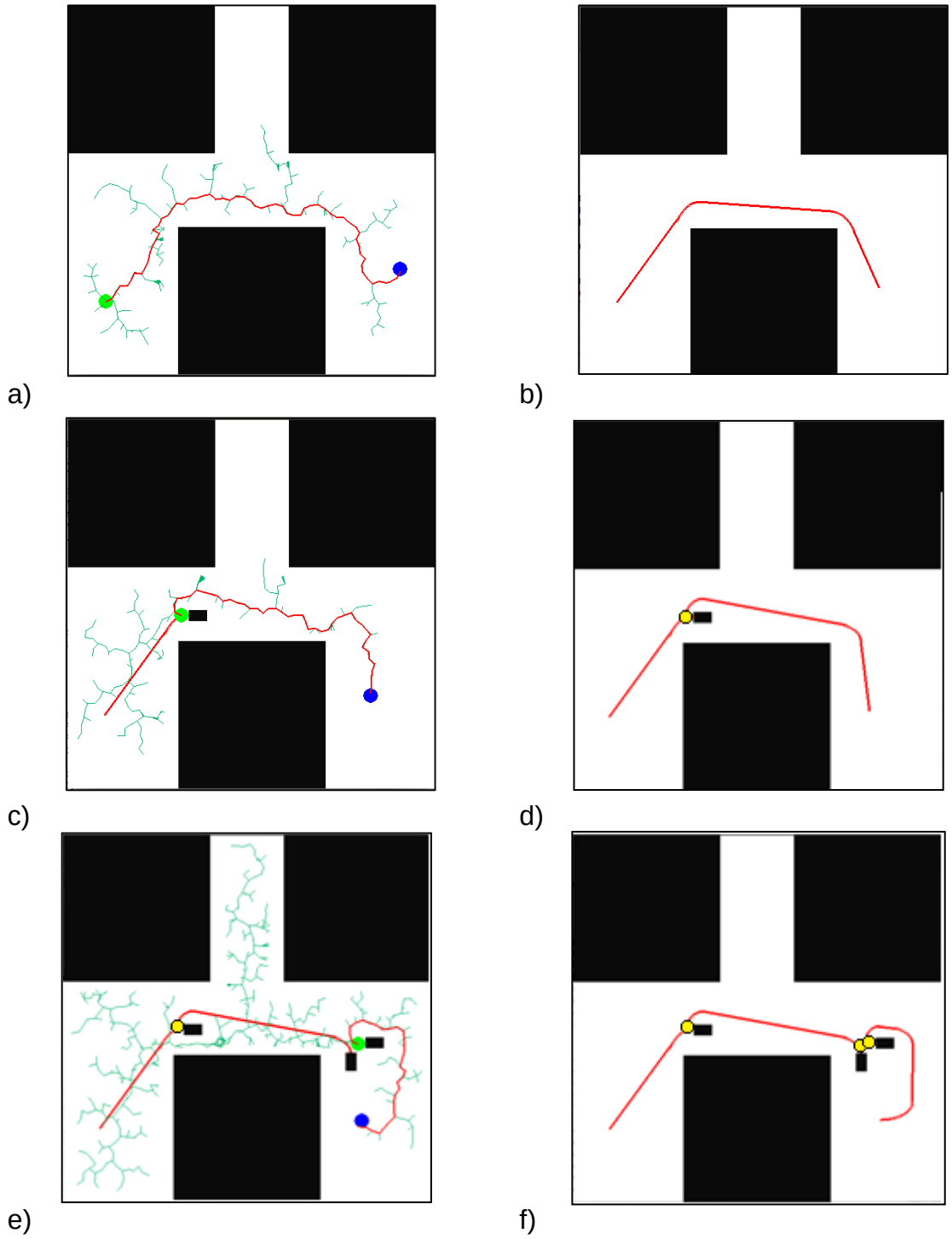
f)

Fuente: Autor

Se muestra un paso a paso de la forma como se comporta la generación de la trayectoria en un entorno dinámico(figura 35), podemos ver que la figura35a genera el trayecto con el entorno ya conocido, la figura 35b muestra la trayectoria final que debe recorrer el robot, la figura 35c, supone que el robot detecto un obstáculo en ese punto de la trayectoria, y ubica un obstáculo “imaginario” en tal punto con las mismas dimensiones del robot, así el algoritmo ubica un nuevo punto inicial en ese punto donde se detuvo el robot y genera un nuevo trayecto, que luego, en la figura 35d lo podemos ver suavizado, además, el punto amarillo que observamos indica que el robot cambio su orientación en su propio eje para ubicarse y seguir el nuevo trayecto. Pero este no es el único obstáculo que se supuso, la figura35e indica que el robot nuevamente encontró obstáculo en un punto de la nueva trayectoria, el algoritmo se comporta de la misma forma finalizando con la trayectoria que vemos en la figura 35f.

Se propone un mapa diferente, para visualizar una vez más el comportamiento en un entorno dinámico. Básicamente las etapas son las mismas del mapa 1 (figura 35), pero esta vez, suponemos una escena en la que el robot detecta un punto de colisión e inmediatamente al generarse la nueva trayectoria, el robot cambia su orientación para reiniciar su ruta y detecta nuevamente colisión, es por esto que observamos dos obstáculos “imaginarios” tan cerca, y también dos puntos amarillos por la misma causa(figura 36e), pero el algoritmo funciona correctamente rodeando los obstáculos que supone, y encuentra el punto de llegada con una trayectoria suavizada (figura 36f).

Figura 36 Comportamiento en un entorno dinámico mapa 2



Fuente: Autor

CONCLUSIONES

Para generar trayectorias en sistemas no holónomos, se deben tener en cuenta las características de estos sistemas, ya que estas se vinculan al tipo de trayectorias que puede seguir el robot; una configuración inicial y una final no pueden unirse mediante cualquier trayectoria, las restricciones cinemáticas del sistema imponen unas condiciones que solo algunos trayectos cumplirán.

Después de realizar la investigación de los métodos de planificación más importantes para la generación de trayectos, y al realizar una comparación de las ventajas y desventajas de cada uno de ellos, se pudo establecer que el algoritmo RRT sobresale por muchas características, siendo este un algoritmo muy útil para la generación de la trayectoria deseada, el cual no lograba solucionar los problemas en tiempo real, pero después de las modificaciones y el post-procesado para el suavizado que en este proyecto se implementó, se soluciona el problema de planificación en tiempo real y además de esto lo hace para sistemas no holónomos.

Los algoritmos puramente aleatorios de planificación de trayectos, tales como el RRT, constituyen una alternativa muy buena para el problema de planificación frente a los métodos de planificación convencionales para determinados entornos complejos y robots con restricciones.

El comportamiento del método propuesto por este proyecto, asegura que la trayectoria que seguirá el robot diferencial, aunque pueda cambiar en el transcurso del recorrido, seguirá evitando cualquier obstáculo, cumpliendo con el objetivo general, primero porque se genera la trayectoria, y segundo porque al detectar obstáculos mientras la recorre, indica que el entorno en el que se encuentra es un entorno dinámico.

Los requisitos computacionales que se necesitan para hacer viable su implementación en tiempo real, incluso para una posible re-planificación durante el trayecto ante obstáculos no modelados, no son de tan alto costo, además las maniobras restringidas como herramienta en el algoritmo, puede suponer una mejora interesante.

La trayectoria generada para un conjunto particular de entornos, al estilo que se presentaron en este documento, indica que el enfoque utilizado es metodológicamente factible y aplicable en sistemas reales de robots móviles.

BIBLIOGRAFÍA

[1] J.-C. Latombe, *Robot Motion Planning.pdf*. Springer Science & Business Media, 2012, 1991.

[2] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[3] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” *In*, vol. 129, pp. 98–11, 1998.

[4] D. Marshall and N. Chambers, “El amanecer del robot,” 2000.

[5] T. Lozano-PÉrez, “Automatic Planning of Manipulator Transfer Movements,” *IEEE Trans. Syst. Man Cybern.*, vol. 11, no. 10, pp. 681–698, 1981.

[6] P. Bhattacharya and M. L. Gavrilova, “Roadmap-based path planning - Using the voronoi diagram for a clearance-based shortest path,” *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, 2008.

[7] J. Sparbert and E. Hofer, “Numerical Path Optimization for Path Planning With Cell Decomposition Methods,” pp. 1822–1827, 2001.

[8] X. Yang, W. Yang, H. Zhang, and H. Chang, “A New Method for Robot Path Planning Based,” no. 2015, pp. 1294–1299, 2016.

[9] Y. S. Silveira and P. J. Alsina, “A New Robot Path Planning Method Based on Probabilistic Foam,” *2016 XIII Lat. Am. Robot. Symp. IV Brazilian Robot. Symp.*, vol. 55, no. 84, pp. 217–222, 2016.

[10] N.A, “Capitulo 8: Grafos,” 2003.

[11] C. C. Semester, “E190Q – Lecture 14 Autonomous Robot Navigation,” 2014.

[12] A. Ollero, “Modelos cinematicos de robots.,” *Robotica Manipuladores y robots moviles.* p. 422, 2001.

[13] J. Borenstein, H. R. Everett, and L. Feng, “Where am I? Sensors and methods for mobile robot positioning,” *Univ. Michigan*, vol. 119, p. 120, 1996.

[14] T. Abbas, M. Arif, and W. Ahmed, “Measurement and correction of systematic odometry errors caused by kinematics imperfections in mobile robots,” *2006 SICE-ICASE Int. Jt. Conf.*, vol. 12, no. 6, pp. 2073–2078, 2006.

[15] S. M. Lavalle, “Planning Algorithms,” *Cambridge*, p. 842, 2006.

[16] “Planificación de Caminos Mediante Grafos de Visibilidad.,” pp. 53–70.

[17] L. E. Kavraki *et al.*, “Probabilistic roadmaps for path planning in high-dimensionalconfiguration spaces,” *Robot. Autom. IEEE Trans.*, vol. 12, no. 4, pp. 566–580, 1996.

[18] F. R. Mendoza, “Geometría Computacional.”

[19] Nguyet Tran, Duy-Tung Nguyen, Duc-Lung Vu, and Nguyen-Vu Truong, “Global path planning for autonomous robots using modified visibility-graph,” *2013 Int. Conf. Control. Autom. Inf. Sci.*, pp. 317–321, 2013.

[20] D. E. T. Con *et al.*, “PLANIFICACIÓN DE TRAYECTORIAS CON EL ALGORITMO RRT. APLICACIÓN A ROBOTS NO HOLÓNOMOS,” vol. 3, pp. 56–67, 2006.

[21] E. E. N. M. Luiz S. Martins-Filho Ronilson Rocha, Romuel F. Machado, Laos A. Hirano, “Kinematic Control of Mobile Robots To Produce Chaotic Trajectories,” *ABCM Symp. Ser. Mechatronics*, vol. 2, pp. 258–264, 2006.

[22] V. F. Muñoz Martinez, “Planificación de Trayectorias para Robots Móviles,” pp. 21–52, 1995.

[23] P. Quintero, “MODELO CINEMATICO DINAMICO DEL MINI ROBÓT MÓVIL RICIMAF,” pp. 49–62, 2012.

[24] R. Illah, *Autonomous Mobile Robots*. .

[25] S. G. Tzafestas, “Mobile Robot Kinematics,” *Introd. to Mob. Robot Control*, pp. 31–67, 2014.

[26] J. Angeles, *Fundamentals of Robotic Mechanical Systems : Theory , Methods , and Algorithms , Second Edition*. 2003.

[27] M. Moll, L. E. Kavraki, and I. A. Şucan, “The Open Motion Planning Library,” *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

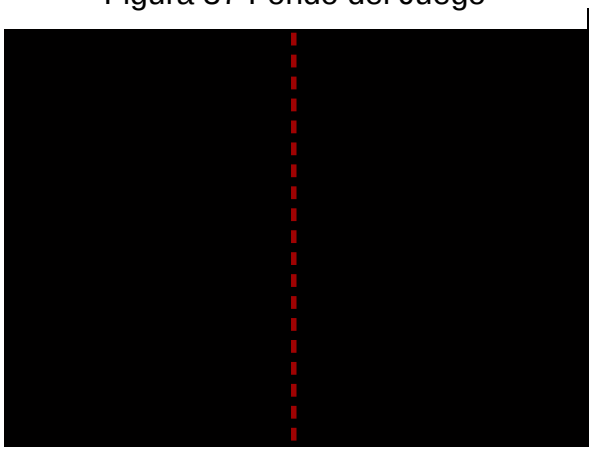
[28] M. Moll, I. A. Sucan, and L. E. Kavraki, “Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization,” *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, pp. 96–102, 2015.

ANEXOS

Anexo A Juego como metodología para el uso de Pygame

Lo primero que se realiza es cargar el fondo del juego, en este caso se le dio el nombre de fondo_pong, y este se guardó en una carpeta previamente creada llamada images.

Figura 37 Fondo del Juego



Fuente: Autor

Para el juego se necesita una pelota, para crear la pelota, se necesita la imagen de esta, y crear una clase, que será el sprite de nuestra pelota, es decir, un sprite es más que una imagen, es una superficie que puede interactuar, moverse y demás.

Figura 38 Bola del juego

Fuente: Autor

```
class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
```

```
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]
```

Una vez creada la clase bola, se debe crear antes del bucle del juego la sentencia

```
bola = Bola ()
```

Ahora solo se debe agregar a la ventana del juego con lo siguiente:

```
screen.blit(bola.image, bola.rect)
```

Es importante tener en cuenta que la bola se debe poner en pantalla después del fondo, ya que si no es así, la bola nos quedara perdida tras el fondo.

```
# Módulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# ---------------------------------------------------------------------

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

# ---------------------------------------------------------------------

# Funciones
# ---------------------------------------------------------------------

def load_image(filename, transparent=False):
        try: image = pygame.image.load(filename)
        except pygame.error, message:
                raise SystemExit, message
        image = image.convert()
        if transparent:
```

```
            color = image.get_at((0,0))
            image.set_colorkey(color, RLEACCEL)
        return image

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image('images/fondo_pong.png')
    bola = Bola()

    while True:
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()
    main()
```

En este punto el programa solo contiene el fondo y la bola, bien definidos, pero aun, la bola no se encuentra interactuando, para lograr esto, se crea un método llamado actualizar, dentro de la clase Bola, este controlara el movimiento de la pelota y si hay colisión con los límites de la ventana.

```
def actualizar(self, time):
    self.rect.centerx += self.speed[0] * time
    self.rect.centery += self.speed[1] * time
    if self.rect.left <= 0 or self.rect.right >= WIDTH:
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
    if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
        self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time
```

Ahora se crea un reloj para controlar el tiempo del juego. Es muy importante para el movimiento y la actualización de la pelota para situarla el espacio.

clock =pygame.time.Clock()

Esta línea se usa antes de entrar al bucle del juego y se utiliza para gestionar el tiempo.

Para saber cuánto tiempo para después de que se ejecuta una interacción del bucle, para ello dentro del bucle ponemos como primera línea:

time = clock.tick(60)

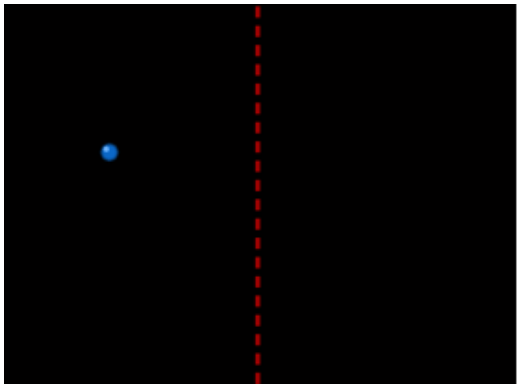
Por último debemos actualizar la posición de la bola antes de actualizarla en la ventana.

bola.actualizar(time)

```
# Módulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# ------------------------------------------------------------------

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
            self.speed[1] = -self.speed[1]
            self.rect.centery += self.speed[1] * time
```

```
# Funciones
# ----------------------------------------------------------------------

def load_image(filename, transparent=False):
        try: image = pygame.image.load(filename)
        except pygame.error, message:
                raise SystemExit, message
        image = image.convert()
        if transparent:
                color = image.get_at((0,0))
                image.set_colorkey(color, RLEACCEL)
        return image

# ----------------------------------------------------------------------

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image('images/fondo_pong.png')
    bola = Bola()

    clock = pygame.time.Clock()

    while True:
        time = clock.tick(60)
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        bola.actualizar(time)
        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()
    main()
```

Figura 39 Interacción de la bola en el juego



Fuente: Autor
La figura 39 muestra la pantalla del juego, aunque no se puede evidenciar, la bola mantiene movimiento a través de toda la pantalla.

Anexo B Implementación del algoritmo RRT

Lo primero que hay que hacer es importar las bibliotecas a usar como por ejemplo el módulo de Pygame, lo cual se hace simplemente como:

```
import pygame
from pygame import *
```

La segunda línea es muy recomendada, ya que se utiliza para importar una serie de constantes que tiene Pygame (como el uso de las teclas o los clics del mouse).

Se crea un objeto clase y se define el método init, el cual tendrá los argumentos self, point y parent que luego serán de gran utilidad.

```
class Node(object):
    def __init__(self, point, parent):
        super(Node, self).__init__()
        self.point = point
        self.parent = parent
```

```
XDIM = 720                    // Tamaño de ventana en X
```

YDIM = 640 // Tamaño de ventana en Y
WindowSize = [XDIM, YDIM] //Dimensión de Ventana
Delta = 15.0 // Máxima distancia de q_{new}
GAME_LEVEL = 4 // Nivel de entorno
GOAL_RADIUS = 10 // Radio de puntos q_{init} y q_{goal}
MIN_DISTANCE_TO_ADD = 1.0 // Mínima distancia del q_{new}
NUMNODES = 20000 // Numero de nodos máximo
pygame.init() // Inicialización del módulo Pygame
fpsClock = pygame.time.Clock() // Se inicia el reloj local de Pygame
screen = pygame.display.set_mode(windowSize) // Se inicia la ventana de trabajo
white = 255, 255, 255 // Color blanco en RGB
black = 10, 10, 10 // Color negro en RGB
red = 255, 0, 0 // Color rojo en RGB
green = 0, 255, 0 // Color verde en RGB
blue = 0, 0, 255 // Color azul en RGB
orange= 255,117,20 // Color naranja en RGB
robot = pygame.Rect((0,0),(30,17)) // Dibuja un tamaño de simulación del Robot móvil.

count = 0 // variable contadora
rectObs = [ ] // Lista de parámetros para el dibujo de obstáculos

La siguiente función dist, calcula la distancia entre los puntos $p1$ y $p2$, que según el algoritmo serían los puntos de configuración q_{init} y q_rand.

Para el cálculo de la distancia se usa la fórmula para determinar la distancia entre dos puntos de coordenadas conocidas.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (80)$$

```
def dist(p1,p2):
    return sqrt((p1[0]-p2[0])*(p1[0]-p2[0])+(p1[1]-p2[1])*(p1[1]-p2[1]))
```

La función step_from_to retorna el nuevo punto en caso de que el punto aleatorio no cumpla la condición de distancia con referencia al punto inmediatamente más cerca, para ello se debe ajustar en la misma dirección pero con una distancia delta anteriormente fijada.

Para calcular esa dirección se debe saber el ángulo que se forma entre los dos puntos, para ello se tiene la formula siguiente:

$$\theta = \tan \frac{y_2 - y_1}{x_2 - x_1} \quad (81)$$

```
def step_from_to(p1,p2):
    if dist(p1,p2) < delta:
        return p2
    else:
        theta = atan2(p2[1]-p1[1],p2[0]-p1[0])
        return p1[0] + delta*cos(theta), p1[1] + delta*sin(theta)
```

Collides es una función que retorna True (verdadero) mientras exista colisión del punto de configuración q_{new} con un obstáculo, si no lo es retorna False (falso).

```
def collides(p):
    for rect in rectObs:
        if rect.collidepoint(p) == True:
            return True
    return False
```

Para definir los diferentes obstáculos se creó la función init_obstacles, esta función cuenta con diferentes configuraciones de obstáculos para un mismo tamaño de ventana, con el fin de observar con más detalle el comportamiento del algoritmo en entornos diferentes.

Para entender con más facilidad la función primero se debe describir detalladamente como se dibuja un rectángulo en Pygame y los parámetros que necesita para su realización:

Para dibujar una figura rectangular se usa la línea:

pygame.draw.rect(Surface, color, Rect, width=0): return Rect

Dibuja una figura rectangular sobre una superficie, es decir el argumento Surface que en este caso será nuestra ventana de trabajo, un color y el parámetro Rect dado es el área del rectángulo. El argumento width es el espesor para dibujar el borde exterior de la figura. Si width se define con valor 0 entonces el rectángulo se pintará completo.

Conociendo esto esta función crea una lista llamada rectObs la cual tendrá los parámetros guardados en cada configuración y que posteriormente se dibujan al final de las líneas con una sentencia for.

```
def init_obstacles(configNum):
    global rectObs
    rectObs = []
    if (configNum == 0):
        rectObs.append(pygame.Rect((XDIM / 2.0 - 50, YDIM / 2.0 - 100),(100,200)))
    if (configNum == 1):
        rectObs.append(pygame.Rect((100,50),(200,150)))
        rectObs.append(pygame.Rect((400,200),(100,200)))
        rectObs.append(pygame.Rect((10,200),(50,150)))
        rectObs.append(pygame.Rect((560,200),(550,300)))

    for rect in rectObs:
        pygame.draw.rect(screen, black, rect)
```

La función reset es bastante sencilla, esta se encarga de reiniciar la ventana de trabajo con el entorno elegido así como también la variable global count.

```
def reset():
    global count
    screen.fill(white)
    init_obstacles(GAME_LEVEL)
    count = 0
```

Así el código completo queda de la siguiente forma:

```
import matplotlib.pyplot as plt
import math, sys, pygame, random
from math import *
from pygame import *
import numpy as np
from sympy import *
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

class Node(object):
    def __init__(self, point, parent):
        super(Node, self).__init__()
        self.point = point
```

```
        self.parent = parent

XDIM = 720
YDIM = 640
windowSize = [XDIM, YDIM]
delta = 10.0
GAME_LEVEL = 4
GOAL_RADIUS = 10
MIN_DISTANCE_TO_ADD = 1.0
NUMNODES = 20000
pygame.init()
fpsClock = pygame.time.Clock()
screen = pygame.display.set_mode(windowSize) #tamaño de ventana
white = 255, 255, 255
black = 10, 10, 10
red = 255, 0, 0
green = 0, 255, 0
blue = 0, 0, 255
cyan = 0,180,105
orange= 255,117,20
robot = pygame.Rect((0,0),(30,17))


count = 0
rectObs = []

def dist(p1,p2):    #distancia entre dos puntos
    return sqrt((p1[0]-p2[0])*(p1[0]-p2[0])+(p1[1]-p2[1])*(p1[1]-p2[1]))

def point_circle_collision(p1, p2, radius):
    distance = dist(p1,p2)
    print distance
    if (distance <= radius):
        return True
    return False

def step_from_to(p1,p2):
    if dist(p1,p2) < delta:
        return p2
    else:
        theta = atan2(p2[1]-p1[1],p2[0]-p1[0])
        return p1[0] + delta*cos(theta), p1[1] + delta*sin(theta)

def collides(p):    #revisa si existe colision con el obstaculo
    for rect in rectObs:
```

```
        if rect.collidepoint(p) == True:
            return True
    return False

def collidesRect(r):    #revisa si existe colision del robot con el obstaculo
    for rect in rectObs:
        if rect.colliderect(r) == True:
            return True
    return False


def get_random_clear():
    while True:
        p = random.random()*XDIM, random.random()*YDIM
        noCollision = collides(p)
        if noCollision == False:
            return p


def init_obstacles(configNum):  #initialized the obstacle
    global rectObs
    rectObs = []
    #print("config "+ str(configNum))
    if (configNum == 0):
        rectObs.append(pygame.Rect((XDIM / 2.0 - 50, YDIM / 2.0 - 100),(100,200)))
    if (configNum == 1):
        rectObs.append(pygame.Rect((100,50),(200,150)))
        rectObs.append(pygame.Rect((400,200),(100,200)))
        rectObs.append(pygame.Rect((10,200),(50,150)))
        rectObs.append(pygame.Rect((560,200),(550,300)))
    if (configNum == 2):
        rectObs.append(pygame.Rect((150,0),(710,80)))
        rectObs.append(pygame.Rect((640,0),(80,640)))
        rectObs.append(pygame.Rect((0,150),(80,630)))
        rectObs.append(pygame.Rect((0,560),(720,80)))
        rectObs.append(pygame.Rect((150,150),(420,60)))
        rectObs.append(pygame.Rect((150,150),(60,350)))
    if (configNum == 3):
        rectObs.append(pygame.Rect((0,150),(400,100)))
        rectObs.append(pygame.Rect((100,250),(300,90)))
        rectObs.append(pygame.Rect((100,360),(300,90)))
        rectObs.append(pygame.Rect((0,450),(400,100)))
    if (configNum == 4):
        rectObs.append(pygame.Rect((100,250),(200,100)))
        rectObs.append(pygame.Rect((260,100),(200,100)))
```

```
        rectObs.append(pygame.Rect((260,400),(200,100)))
        rectObs.append(pygame.Rect((410,250),(200,100)))

    for rect in rectObs:
        pygame.draw.rect(screen, black, rect)


def reset():
    global count
    screen.fill(white)
    init_obstacles(GAME_LEVEL)
    count = 0

def guardar(file_txt,coorde):
    archivo = open(file_txt, "a")
    for posX,posY in coorde:
        archivo.write(str(posX)+","+str(posY)+"\n")
    archivo.close()

def main():
    global count

    initPoseSet = False
    initialPoint = Node(None, None)
    goalPoseSet = False
    goalPoint = Node(None, None)
    currentState = 'init'

    nodes = []
    reset()


    while True:
        if currentState == 'init':
            #print('goal point not yet set')
            pygame.display.set_caption('Select Starting Point and then Goal Point')
            fpsClock.tick(10)
        elif currentState == 'goalFound':
            currNode = goalNode.parent
            pygame.display.set_caption('Goal Reached')
            goalNode = nodes[len(nodes)-1]
            #np.savetxt('file1.txt', goalNode)
            #screen.fill((255,255,255))
            init_obstacles(GAME_LEVEL)
            pygame.draw.rect(screen, orange, robot)
```

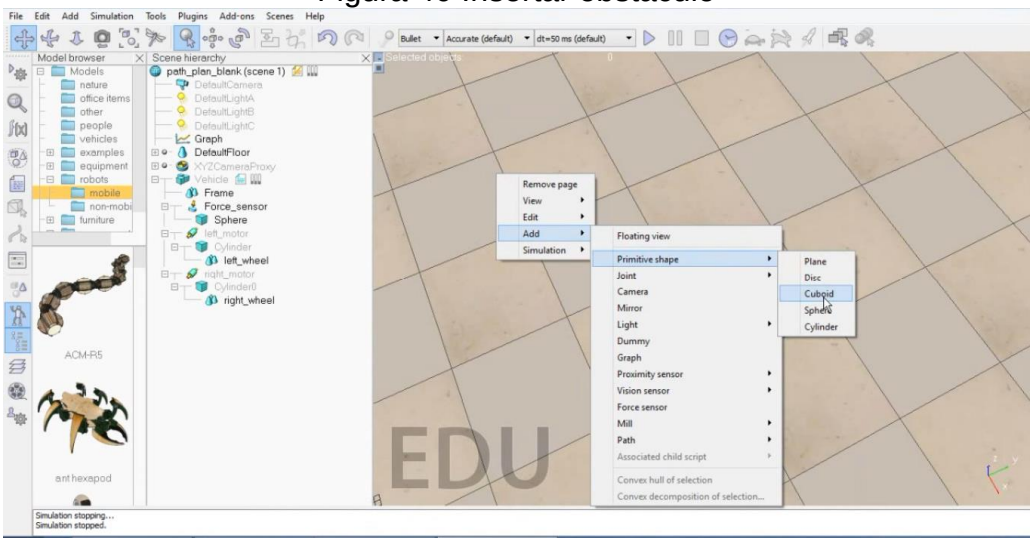
```
            f=[]
            h=[]
            while currNode.parent != None:

#pygame.draw.aalines(screen,red,currNode.point,currNode.parent.point,4)
                #pygame.draw.aalines(screen,red,parentNode.point,newnode,4)
                pygame.draw.aalines(screen,red,True,
[currNode.point,currNode.parent.point] ,100)
                pygame.draw.aalines(screen,red,True,         [parentNode.point,newnode]
,100)
                currNode = currNode.parent
                f.append(currNode.point)
                print('esto'+str(f))
                #np.savetxt('f.txt', f ,fmt='%10.5f')
                datos = guardar('coordenadas2.txt',f)
                optimizePhase = True


        elif currentState == 'optimize':
            fpsClock.tick(0.5)
            pass
        elif currentState == 'buildTree':
            count = count+1
            pygame.display.set_caption('Performing RRT')
            if count < NUMNODES:
                foundNext = False

                while foundNext == False:
                    rand = get_random_clear()
                    parentNode = nodes[0]
                    f=[]
                    for p in nodes:
                        if dist(p.point,rand) <= dist(parentNode.point,rand):
                            newPoint = step_from_to(p.point,rand)
                            robot.center=newPoint
                            #f=[robot.center]
                            #print('robot: '+str(robot.center))
                            if collidesRect(robot) == False:
                                parentNode = p
                                foundNext = True

                newnode = step_from_to(parentNode.point,rand)
                nodes.append(Node(newnode, parentNode))
                pygame.draw.line(screen,cyan,parentNode.point,newnode)
```

```
            if point_circle_collision(newnode, goalPoint.point, GOAL_RADIUS):
                currentState = 'goalFound'

                goalNode = nodes[len(nodes)-1]

            else:
                print("Ran out of nodes... :(")
                return;

        #handle events
        for e in pygame.event.get():
            if e.type == QUIT or (e.type == KEYUP and e.key == K_ESCAPE):
                sys.exit("Exiting")
            if e.type == MOUSEBUTTONDOWN:
                print('mouse down')
                if currentState == 'init':
                    if initPoseSet == False:
                        nodes = []
                        if collides(e.pos) == False:
                            print('initiale point set: '+str(e.pos))

                            initialPoint = Node(e.pos, None)
                            nodes.append(initialPoint) # Start in the center
                            initPoseSet = True
                            pygame.draw.circle(screen, green, initialPoint.point,
GOAL_RADIUS)
                    elif goalPoseSet == False:
                        print('goal point set: '+str(e.pos))
                        if collides(e.pos) == False:
                            goalPoint = Node(e.pos,None)
                            goalPoseSet = True
                            pygame.draw.circle(screen, blue, goalPoint.point,
GOAL_RADIUS)
                            currentState = 'buildTree'
                else:
                    currentState = 'init'
                    initPoseSet = False
                    goalPoseSet = False
                    reset()
        pygame.display.update()
        fpsClock.tick(100)

if __name__ == '__main__':

    main()
```

Anexo C V-rep configuracion de los obstaculos.

Para agregar un obstaculo en el espacio de pantalla, se hace click derecho y se eligen las opciones como se muestra en la figura:

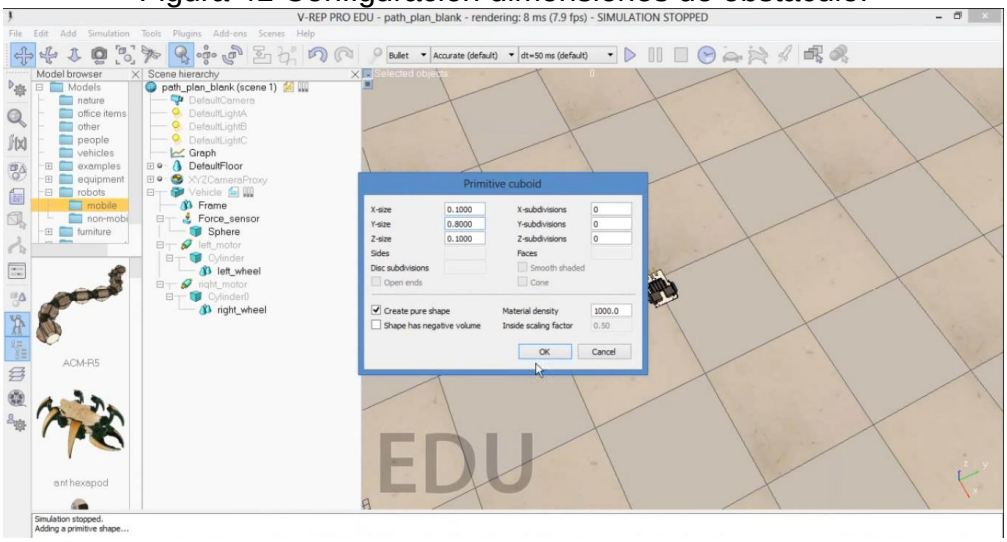
Figura 40 Insertar obstáculo



Fuente: Autor

Una ves seleccionado las opciones, se deben dar los parametros que tendra el obstaculo, es decir el area que va a ocupar en el espacio.

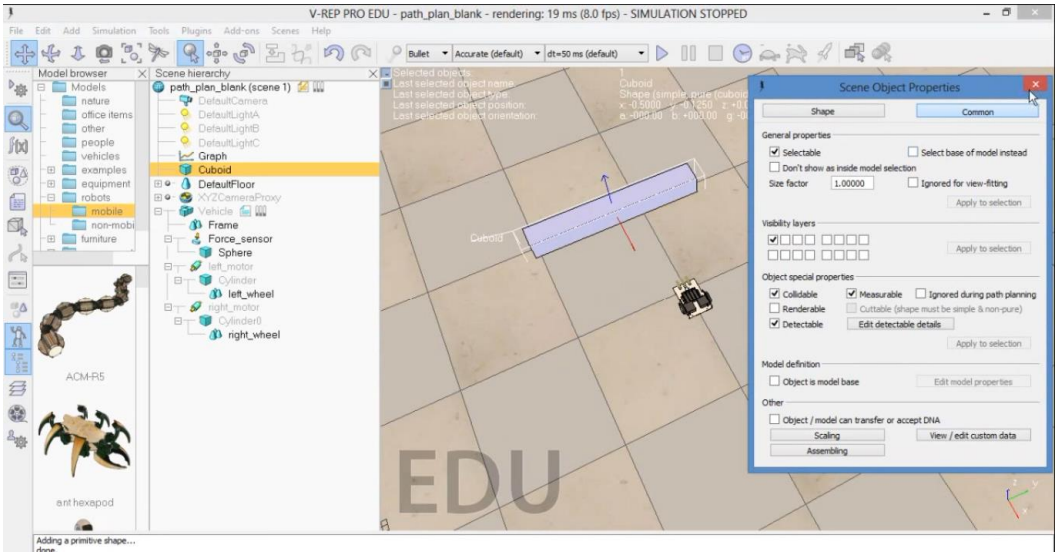
Figura 41 Configuración dimensiones de obstáculo.



Fuente: Autor

Una vez construidos los obstaculos, se deben activar algunas propiedades del objeto(obstaculo) en la escena, para ello se da doble click sobre el objeto y se activan las propiedades especiales como se muestra en la figura42.

Figura 42 Propiedades del obstáculo



Fuente: Autor

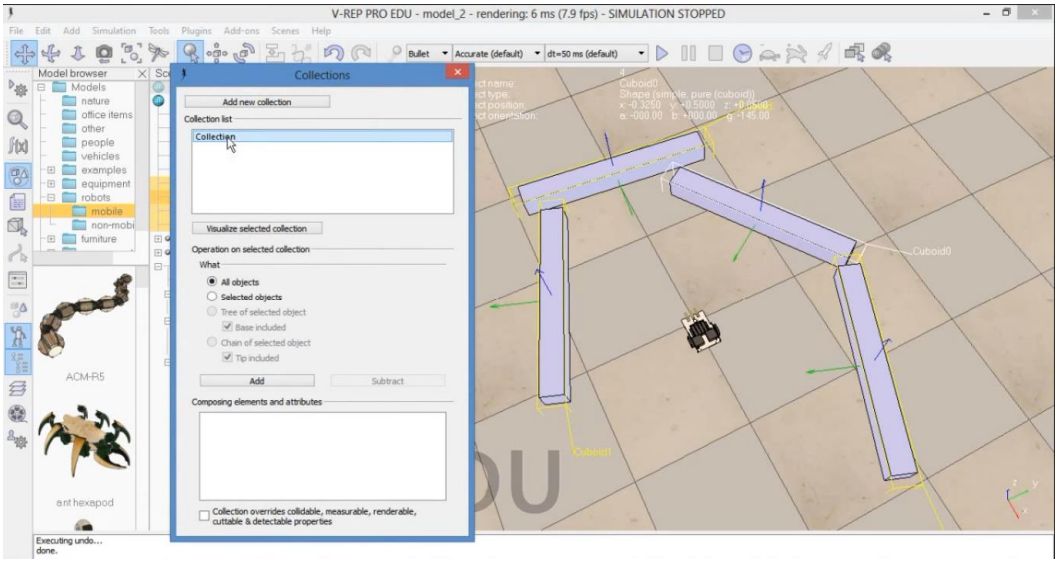
Si se desean crear mas obstaculos con las mismas propiedades, simplemente se puede hacer una copia y luego pegarla. Las herramientas del simulador nos permiten rotar o trasladar los objetos a gusto.
Si el entorno que se desea crear, contiene mas de un objeto o obstaculo, se debe crear una colección de estos, para facilitar la identificacion de las propiedades especiales que se ajustaron.

Figura 43 Botón Colección

Fuente: Autor

Para crear la colección, se seleccionan los obstaculos y se elige el boton colección en la barra de herramientas, como se muestra en la siguiente figura:

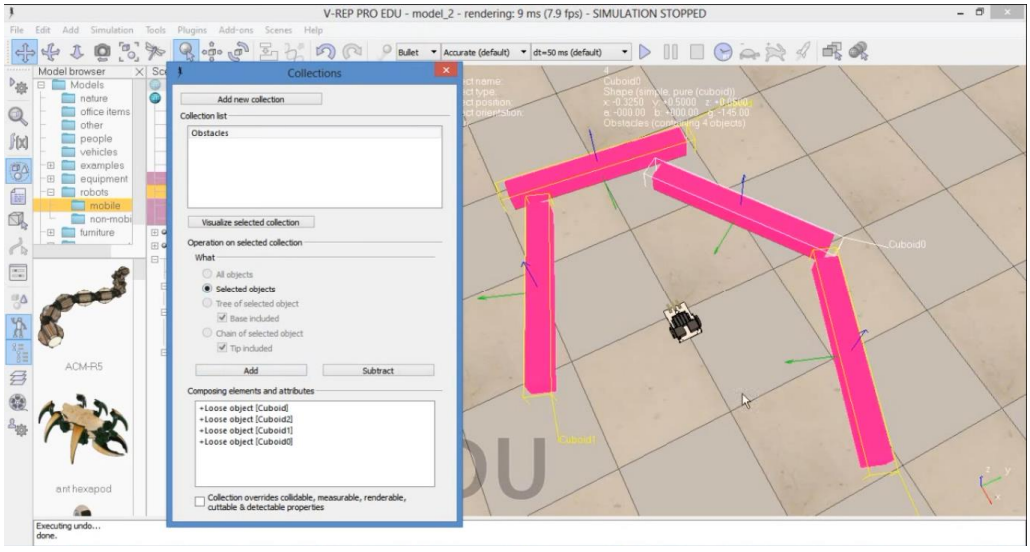
Figura 44 Crear colección de objetos



Fuente: Autor

Al elegir la opcion de objetos seleccionados, no permite agregar los objetos previamente seleccionados, estos se identificaran al cambiar su color a un tono rosa.

Figura 45 Agregar obstáculos a la colección

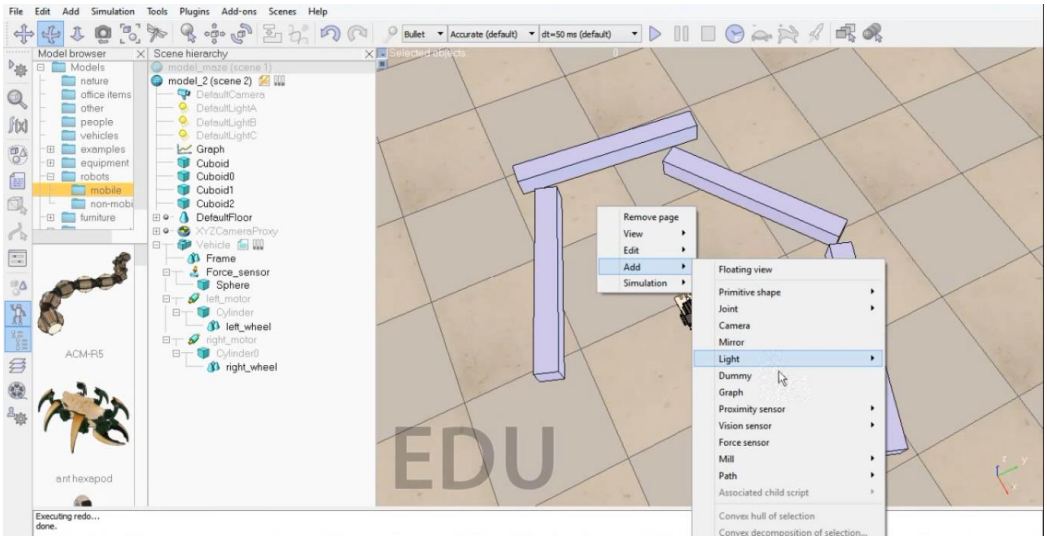


Fuente: Autor

Anexo D Configuración para la generacion del trayecto por OMPL en V-rep

Para la configuracion de la planificacion del trayecto se necesita 3 parametros.
El primero es el punto de partida o configuracion inicial , para esto damos click derecho, seleccionamos agregar y a continuacion en dummy. Una vez seleccionado, se agrega una configuracion en la barra de propiedades de la escena, alli cambiamos el nombre Dummy por Start, para facilitar diferenciarlos, el segundo es el punto de llegada o configuracion final, este se crea igual que el parametro de configuracion inicial, de hecho se puede hace una copia y cambiar el nombre.

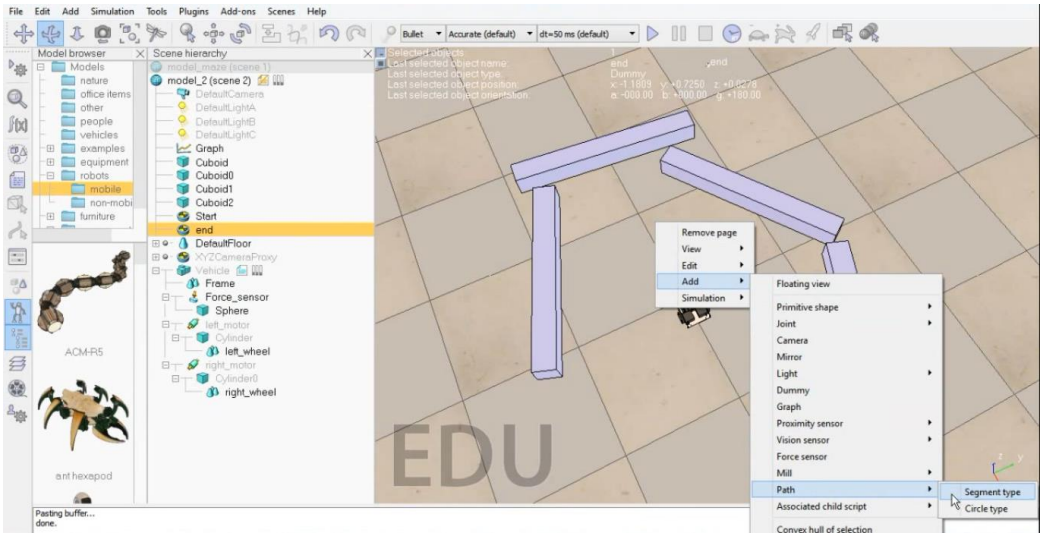
Figura 46 Insertar Dummy



Fuente: Autor

Y el tercer parametro es el objeto de trazado, es decir la ruta que se creara entre los dos puntos inicial y final, para crear este, se da click derecho sobre la escena, agregar, y a continuacion path, saldra un pestaña con dos opciones, elegiremos la de tipo segmento.

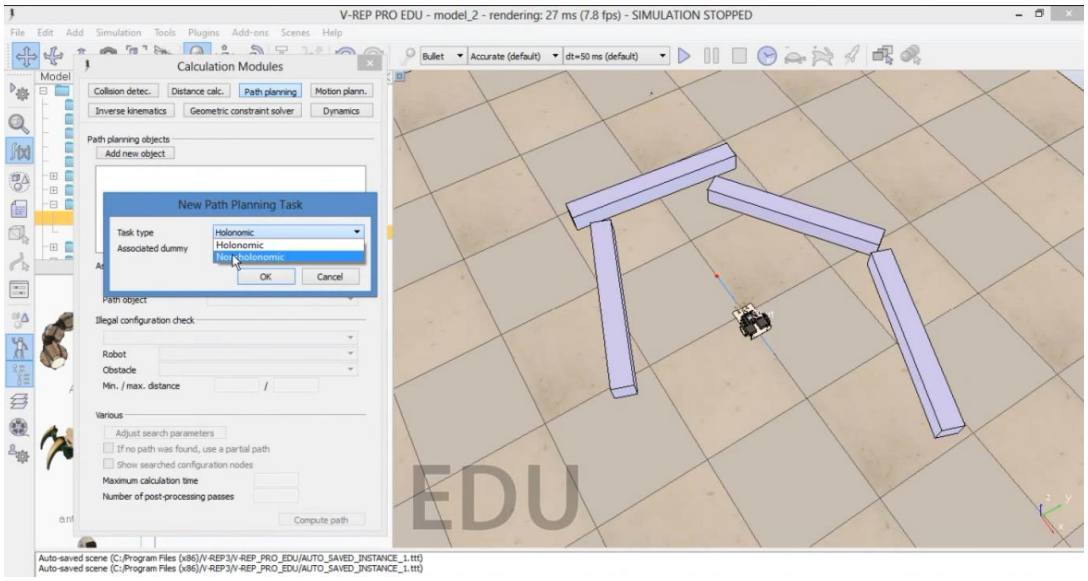
Figura 47 Insertar trayectoria tipo segmento



Fuente: Autor

En la parte izquierda de la pantalla, en la barra de herramientas, hay un icono que nos permite cambiar la configuracion del modulo de propiedades, en este crearemos las propiedades que tendra la trayectoria.

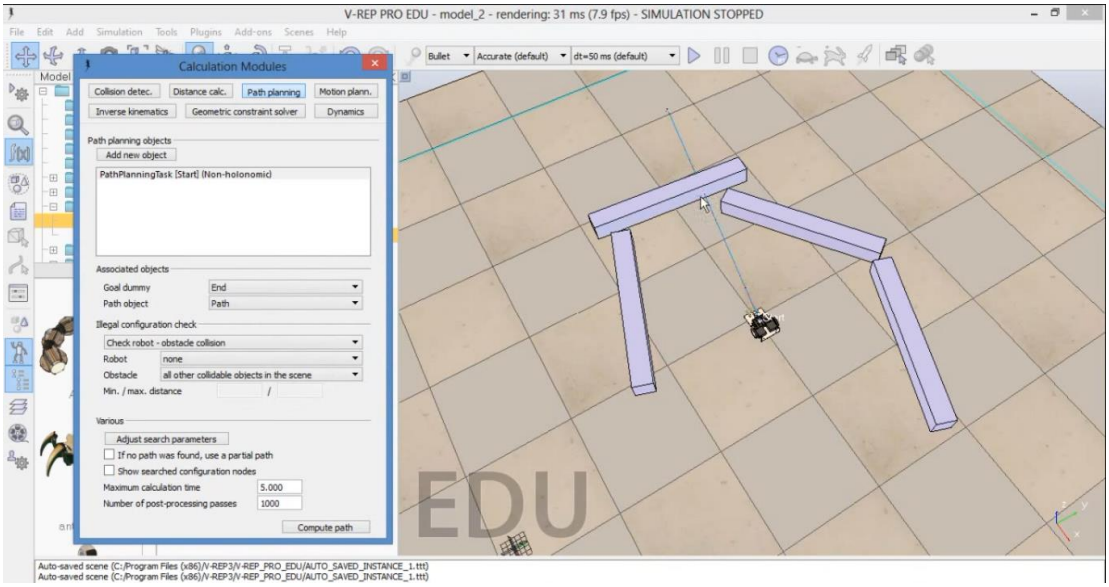
Figura 48 Configuración del tipo de robot para la planificación de la trayectoria



Fuente: Autor

Se agrega un nuevo objeto, y se configuran los 3 parámetros creados anteriormente. Además se establecen parámetros como: si el sistema es holónomo o no, tiempo máximo para el cálculo de la trayectoria, los obstáculos que se desean agregar, y demás. Por último, al configurar todos los parámetros necesarios, se da click en computar la trayectoria, esta se genera a partir de una librería llamada OMPI.

Figura 49 Configuración de la planificación de la trayectoria



Fuente: Autor

Anexo E Editar una trayectoria en V-rep

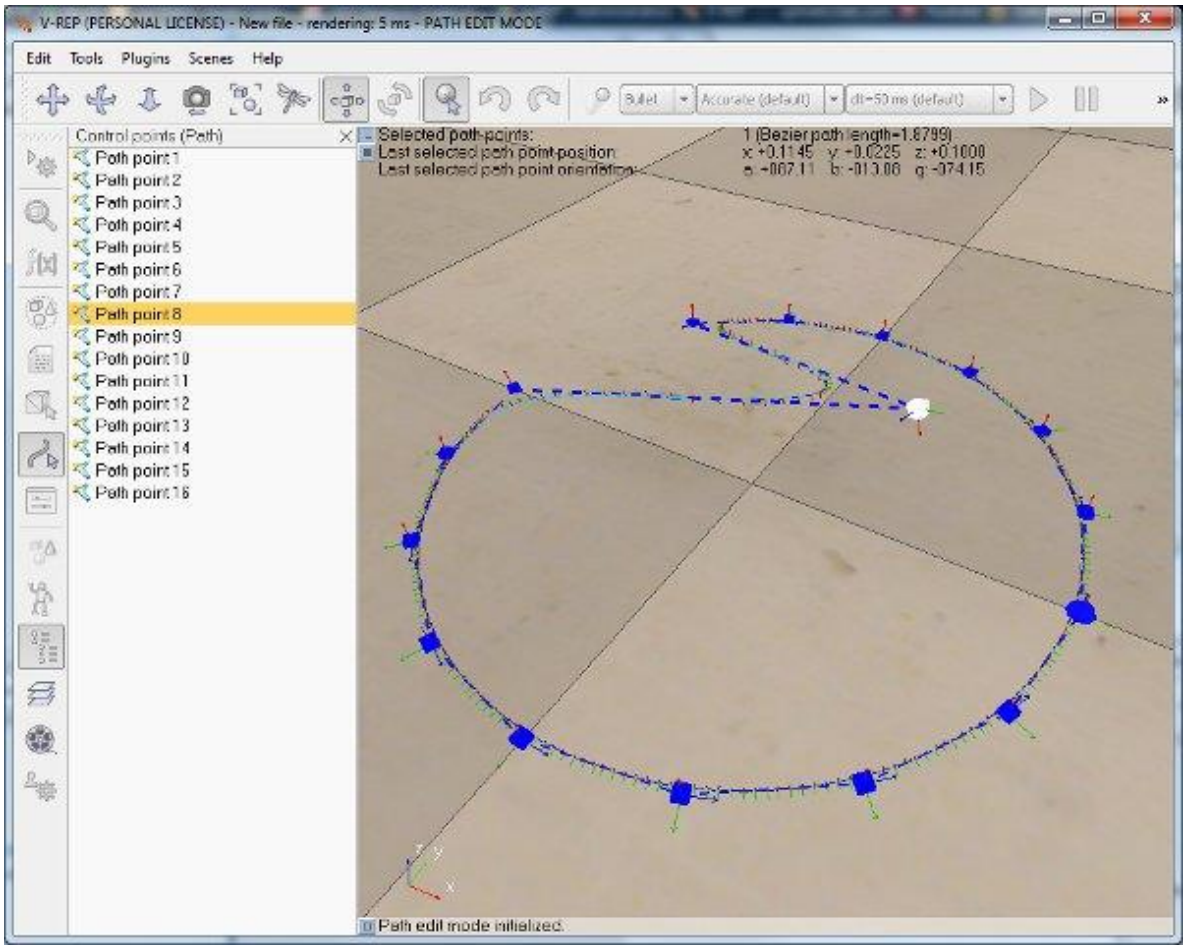
V_rep cuenta con una herramienta para crear manualmente una trayectoria, esta se encuentra en la parte izquierda, en la barra de herramientas de simulación.

Figura 50 Botón editar trayectoria

Fuente: Autor

Este botón solo se activa si se selecciona una ruta. Una vez en el modo de edición, podemos agregar o quitar puntos de control de ruta, para luego moverlos a voluntad y editar la ruta deseada.

Figura 51 Ruta editada manualmente



Fuente: Autor