Comparación de algoritmos de planeación de rutas para robots de servicio doméstico

Luis Sergio Cano Olguin

Tutor: Dr. Marco Antonio Negrete Villanueva

II

Resumen

Índice general

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

En los últimos años, el desarrollo de robots móviles autónomos ha cobrado gran importancia, debido a que esta área ha tenido considerables avances en cuanto funcionalidad y aplicaciones, llegando a tener un gran impacto social(De Graaf and Allouch, 2016).

El progreso de de la robótica en conjunto con otras disciplinas ha logrado muchos avances, de los cuales podemos resaltar la autonomía que un robot posee para realizar tareas y el grado de complejidad del ambiente en el que puede efectuarlas. A causa de esto ha crecido la cantidad de problemas en los que estos pueden ser usados. Las primeras aplicaciones donde los robots mostraron ser de gran utilidad fue en el área industrial, no obstante ahora esta es solo una parte de un grupo mas grande de sectores en los que los robots están involucrados, como por ejemplo la medicina, agricultura, minería etc (Rubio et al., 2019). Debido a esto los robots se pueden clasificar en dos grandes grupos: industriales y de servicio.

Los robots de servicio tienen la capacidad de trabajar en entornos no estructurados, en condiciones ambientales cambiantes y con una estrecha interacción con los humanos. En 1995 fue creado por la *IEEE Robotics and Automation Society*, el comité *Technical Committee on Service Robots*, este definió de manera más especifica las aplicaciones de los robots de servicio,

las cuales se pueden dividir en dos partes: 1) sectores productivos no manufactureros tales como edificación, agricultura, naval, minería, medicina, etc. y 2) sectores domésticos: asistencia personal, limpieza, vigilancia, educación, entretenimiento, etc (Aracil et al., 2008).

Se puede considerar un robot de servicio doméstico a todo aquel que cuente con la capacidad de convivir con personas y de realizar tareas que incidan directamente en su forma de vida (Hüttenrauch and Severinson-Eklundh, 2006). Un robot de este tipo debe realizar las tareas con total autonomía, sin más intervención de las personas que una programación previa de sus actividades (Park et al., 2008).

El primer sistema que responde a las características de uno de estos robots y que ha penetrado en el mercado de los equipos domésticos ha sido el robot aspiradora *Roomba*. Estos son programables y poseen varios sensores, utilizan algoritmos que les permiten cubrir todo el suelo del recinto a aspirar, se desplazan siguiendo el límite de las paredes y alrededor de los muebles, a pesar de ello sus movimientos no son planeados y la navegación se limita a la evasión de obstáculos, sin embargo son capaces de dirigirse al punto de alimentación eléctrica cuando detectan que sus baterías están bajas.

Existen varias competencias que se encargan de fomentar el desarrollo de robots de servicio, como Robocup@Home (Holz et al., 2013) y RoCKIn (Lima et al., 2013). Estas competencias proporcionan un ambiente estandarizado para probar el desempeño de robots de servicio doméstico. En los robots de estas competencias se ha observado que la solución más común para la navegación autónoma es el uso de paquetes de ROS tal como el *Navigation Stack* (`http://wiki.ros.org/navigation`) el cual incluye nodos para la planeación de rutas, localización, evasión de obstáculos y control, resultando así en una poderosa herramienta para el problema de la planeación de movimientos.

El problema de la navegación ha tenido grandes avances en el área de los vehículos autónomos, sin embargo, la planeación de rutas en los robots

de servicio domestico pude ser muy diferente, debido a los entornos con los que estos interactúan, por lo que en ciertas circunstancias es necesario usar métodos distintos.

Algunos problemas con los que también un robot de este tipo debe lidiar son: visión computacional, reconocimiento de patrones, interacción humano-robot, y nodos de control, entre otros. Este trabajo se enfocara en el problema de la navegación autónoma, específicamente, el problema de la planeación de rutas.

1.1. Motivación

El problema de la planeación de movimientos en ambientes dinámicos y estáticos, implica que el robot debe ser capaz de localizarse a sí mismo, contar con un mapa del entorno o en caso de ser necesario, mapear el ambiente en el que se encuentra, y de esta manera, poder de hallar un camino libre de colisiones desde un punto a otro (Choset et al., 2005).

La navegación autónoma desempeña un papel importante para los robots en general, sin embargo en este trabajo nos centraremos en los robots de servicio domestico. Estos deben de tener una buena planeación de rutas debido a su gran interacción con los humanos y a su desempeño en ambientes dinámicos.

El problema de la planeación de movimientos implica varias tareas. La primera de ellas se conoce como localización y mapeo simultáneos (*SLAM* por sus siglas en inglés). Esta se aplica cuando el robot no tiene acceso a un mapa del entorno ni conoce su posición en el mismo. En estos caso, el robot deberá armar un mapa del entorno apoyándose constantemente en la información de su ubicación para tener mayores probabilidades de éxito. Esta tarea resulta compleja por el hecho que para poder localizarse de forma precisa se necesita un mapa, y por otro lado, para poder crear un mapa es necesario estar localizado de forma precisa (Durrant-Whyte and Bailey,

2006).

Una vez obtenida una representación del ambiente se puede proceder con la planeación de movimientos libres de colisiones, esta consiste en varios componentes muy relacionados entre si, como los controles de bajo nivel y localización. Debido a esto se han desarrollado una gran cantidad de métodos y estrategias para encontrar la mejor manera de generar rutas óptimas entre un punto inicial y final.

A pesar de la variedad de métodos para resolver el problema de la planificación, la complejidad del entorno, el número de grados de libertad y las restricciones cinemáticas y dinámicas que puede presentar el robot son factores que suelen exceder las limitaciones de estos métodos o incrementar excesivamente el tiempo de cómputo para hallar la solución (López García et al., 2011).

Existe una gran cantidad de trabajo reportado en la literatura científica sobre el problema de la planeación de movimientos en robots autónomos, no obstante, éste es aún un problema abierto y desafiante cuando se trata de robots navegando en ambientes reales (Banino et al., 2018).

1.2. Planteamiento del problema

Existen diferentes tipos de algoritmos para la planeación de rutas, los cuales son capaces de encontrar un camino óptimo sujeto a las restricciones del entorno. Estos poseen diferentes técnicas, métodos y estrategias para resolver este tipo de problemas.

El ambiente en el que estos algoritmos es aplicado determinará en gran parte la efectividad del mismo, ya que para algunos de ellos es más fácil y rápido encontrar una ruta en un entorno con ciertas características. Debido a esto se busca explorar las ventajas que presenta cada uno en la planificación óptima de rutas para determinar qué algoritmos son mas adecuados bajo cierto tipo de condiciones en la navegación autónoma en robots móviles.

1.3. Hipótesis

En el desarrollo de este trabajo se consideraron las siguientes hipótesis:

- El entorno en el que se desarrolla un algoritmo influye en su desempeño.
- Los distintos métodos implementados pueden presentar diferencias significativas dependiendo del ambiente en que se usen.
- Se pueden generar mejores rutas si se conoce en qué circunstancias se desarrolla el proceso seleccionado.

1.4. Objetivos

- Comparar diferentes algoritmos de planificación de rutas en diversos entornos y ambientes, utilizando modelos simulados de robots de servicio doméstico.
- Determinar el desempeño de los algoritmos frente a diversas situaciones.
- Obtener resultados que muestren el comportamiento y eficiencia de estos.
- Identificar y describir los algoritmos más adecuados para determinadas condiciones.

1.5. Descripción del documento

El contenido de este trabajo está organizado de la siguiente manera: En el Capitulo 2 se muestran los conceptos, herramientas y problemas básicos presentes en la navegación de los robots autónomos. En el Capitulo 3 se describen los métodos basados en grafos, en qué consisten y cómo funcionan, y de esta manera presentar el primer algoritmo implementado en este trabajo:

el algoritmo $A*$. En el Capitulo 4 se explica en qué consisten los métodos basados en muestreo, después se desarrolla el algoritmo RRT y sus variantes para la planeación de rutas: *RRT-Ext y RRT-Connect*, los cuales son los otros métodos implementados en este documento. En el Capitulo 5 se muestran las herramientas usadas para la implementación y evaluación de los algoritmos desarrollados, como el uso del meta-sistema operativo ROS, el robot HSR y las estrategias de comparación utilizando pruebas estadísticas. En el Capitulo 6 se presentan los resultados obtenidos con las estrategias de comparación empleadas sobre los algoritmos (*tiempo de ejecución, longitud y tortuosidad*), junto con los mapas en los que se probaron. Por último, en el Capitulo 7 se exponen las conclusiones y se plantea el trabajo a futuro.

Capítulo 2

Antecedentes

En este capitulo se describen los conceptos básicos que serán desarrollados y utilizados en el resto del documento. Primero se explica en qué consiste un robot de servicio, sus características y algunas limitaciones a tener en cuenta. Después se abordará el tema de la planeación de movimientos y los problemas que esta presenta, junto con el planteamiento de algunos métodos, técnicas y herramientas que pueden ser de gran ayuda al momento de realizar esta tarea.

2.1. Los robots de servicio doméstico

La Organización Internacional de Normalización define un "robot de servicio" como un robot "que realiza tareas útiles para humanos o equipos, excluyendo las aplicaciones de automatización industrial" (ISO, 2012).

Esta norma también establece que los robots requieren "un grado de autonomía", lo cual hace referencia a la capacidad de realizar tareas previstas en función del estado actual en conjunto con la detección de su entorno sin intervención humana. Para los robots de servicio, esto va desde la autonomía parcial, incluida la interacción del humano-robot, hasta la autonomía total, sin ninguna intervención humana. La Federación Internacional de Robótica

clasifica a los robots de servicio según el uso personal o profesional. Tienen muchas formas y estructuras, así como áreas de aplicación (IFR, 2018).

Los robots de servicio de uso personal están enfocados al desarrollo de tareas no comerciales, generalmente son usados por personas no profesionales, por ejemplos las sillas de ruedas automatizadas, sistemas de vigilancia, y todos aquellos que realizan tareas en el hogar. Por el contrario los robots de servicio profesional están hechos para el uso comercial y son usados por especialistas capacitados, estos son utilizados en la médica, el campo, la minería, etc.

La planeación de movimientos en la robótica es muy importante ya que tiene como finalidad la generación de un conjunto de acciones que permitan a un robot de servicio moverse a través de un entorno conocido o desconocido para alcanzar uno o más objetivos partiendo de su posición inicial. Una de las tareas más comunes que se pueden resolver mediante la planeación de movimientos es la navegación en entornos con una gran cantidad de obstáculos (Sieira and Molina, 2011).

El problema puede ser resuelto con diferentes técnicas y métodos. En cualquier caso, para un correcto funcionamiento de un robot real, es necesario tener en cuenta la incertidumbre del entorno, así como la de los sensores y actuadores del robot. Si la planificación se realiza a bajo nivel, se seleccionan las acciones de control, aunque cuando el mapa del entorno es desconocido, las soluciones encontradas pueden ser subóptimas, o incluso no encontrarse. Otra alternativa es realizar una planificación a alto nivel, donde no se tenga en cuenta el control del robot y el objetivo sea la obtención de una ruta libre de obstáculos. Estos métodos fallan si alguno de los pasos propuesto por el planificador no puede ser ejecutado a bajo nivel. Por ello, es muy importante tener en cuenta las restricciones de movimiento del robot al obtener una ruta. No es lo mismo planificar para un robot con un movimiento de tipo Ackerman (ver figura 2.1), que para uno de tipo diferencial (ver figura 2.2), que permite giros sobre sí mismo (Sieira and Molina, 2011).

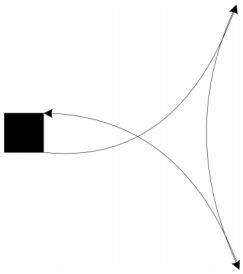


Figura 2.1: Movimiento de tipo Ackerman

Figura 2.2: Robot Takeshi (HSR)

2.2. El problema de la planeación de movimientos

Una tarea fundamental de la robótica es planear movimientos libres de colisión para cuerpos complejos desde un estado inicial hasta una posición objetivo a través de una colección de obstáculos estáticos. Aunque relativamente

simple, este problema puede ser computacionalmente difícil. Las extensiones de este planteamiento tienen en cuenta problemas adicionales que se heredan de las limitaciones mecánicas y sensores de los robots reales, como las incertidumbres, la retroalimentación y las restricciones de movimiento, que complican aún más el desarrollo de los planificadores automáticos. Los algoritmos modernos han tenido bastante éxito al abordar instancias difíciles del problema básico y se ha dedicado mucho esfuerzo a extender sus capacidades a instancias más desafiantes. Estos algoritmos han tenido un éxito generalizado en aplicaciones más allá de la robótica, como la animación por computadora, la creación de prototipos virtuales y la biología computacional (Reif, 1979).

Las características más importantes de la planeación de movimientos dependen del problema que se desea resolver. Se consideran cuatro tareas principales: navegación, cobertura, localización y mapeo (Choset et al., 2005). La navegación consiste en encontrar un camino libre de colisiones para el sistema de un robot, con el fin de moverse desde una configuración inicial a una final. La cobertura es el problema de usar un sensor o herramienta sobre todos los puntos en un espacio. La localización es el problema de usar un mapa para interpretar los datos del sensor para determinar la configuración y acciones del robot. El mapeo consiste en explorar y detectar un entorno desconocido para construir una representación que sea útil para la navegación, la cobertura o la localización.

La planeación de movimientos puede ser clasificada en dinámica o estática dependiendo de si el ambiente es cambiante o no. En un problema estático toda la información acerca del entorno se conoce y el movimiento del robot se diseña a partir de ella. Mientras que en un problema dinámico solo se tiene información parcial del ambiente y el movimiento del robot se planifica con esta, conforme éste se mueve se obtiene más información para continuar con las trayectorias.

Los métodos de planeación de movimientos también pueden ser clasifi-

cados por su precisión y alcance. Los primeros de estos son clasificados en completos, heurísticos, de resolución completa y probabilísticamente completos. Los métodos completos encuentran una solución si esta existe o bien reportan falla en caso contrario y por su naturaleza son computacionalmente caros. Los heurísticos se enfocan en generar soluciones rápidas sin embargo pueden fallar o encontrar en algunas ocasiones soluciones erróneas. Los métodos de resolución completa discretizan el espacio de trabajo en una malla de configuraciones para encontrar un solución. Por otra parte, los métodos clasificados por alcance se subdividen en globales y locales. Los métodos globales trabajan con la información de todo el espacio de configuración y planifican una ruta desde la configuración inicial hasta la configuración final. En cambio los métodos locales no exploran todo el espacio de configuración, son utilizados como componentes de los métodos globales y se caracterizan por ser bastante rápidos al calcular una ruta dentro del espacio de configuraciones del robot (De la Rosa, 2004).

De igual manera la planificación se puede plantear como un problema de búsqueda de tal forma que se puede expresar la información relevante mediante un grafo (Cuchango, 2012). Por lo que se pueden usar métodos basados en estos, de los cuales se distinguen: los grafos de visibilidad, diagramas de Voronoi, modelado del espacio libre, descomposición en celdas, búsqueda a lo ancho y profundidad, algoritmo de Dijkstra y A*. Estos cuatro últimos serán abordados en el Capitulo 3.

Otro tipo de métodos emplean un enfoque estocástico y de muestreo, que da lugar al algoritmo de planeación aleatoria de trayectorias (RPP por sus siglas en inglés), al algoritmo de mapas probabilísticos (PRM por sus siglas en inglés), y al algoritmo de árboles de exploración rápida (RRT por sus siglas en inglés). Este último algoritmo y su implementación se describen en el capitulo 4.2.

En la mayor parte de los algoritmos de planeación, la búsqueda del camino se realiza teniendo en cuenta sólo la posición del robot, dejando el control del

mismo como un problema independiente a resolver; sin embargo, es posible incluir la velocidad angular, la velocidad lineal y la orientación del robot en el proceso de búsqueda, lo cual aporta dos ventajas sobre la aproximación anterior: por una parte se da solución al problema del control del robot junto con el de la obtención de una ruta, ya que se puede introducir información sobre los cambios de estas variables en las transiciones de estado. Por otra parte se introducen como condiciones de búsqueda las restricciones cinemáticas del vehículo utilizado, por lo que el camino obtenido no contiene giros imposibles o cambios de velocidad demasiado bruscos ni atraviesa zonas que debido a las restricciones cinemáticas del vehículo, son intransitables (Sieira and Molina, 2011).

2.2.1. Métricas para la planeación de rutas

Son usadas con el objetivo de determinar si una ruta es óptima, para lograr esto, se realiza una comparación implícita, algunos métodos pueden producir una ruta óptima porque consideran todas las rutas posibles entre puntos, esto puede llegar a ser computacionalmente costoso (Murphy, 2019). Sorprendentemente, un camino óptimo puede no parecerlo para el ojo humano. Por ejemplo, una ruta matemáticamente óptima de un mundo dividido en celdas o cuadrículas puede ser muy irregular en lugar de recta. La capacidad de producir y comparar todos los caminos posibles también supone que la planificación tiene acceso a un mapa del mundo preexistente, igualmente importante, supone que el mapa es preciso y está actualizado.

Los planificadores de rutas primero dividen el mundo en una estructura adecuada. Usan una gran variedad de técnicas para representar el mundo; ninguna técnica es dominante, aunque las cuadrículas regulares (celdas) parecen ser populares. La intención de cualquier representación es mostrar solo las características sobresalientes, o la configuración de objetos relevantes para la navegación en el espacio de interés; de ahí el término espacio de configuración, el cual se define como una estructura de datos que permite

al robot especificar la posición (ubicación y orientación) de cualquier objeto y de sí mismo. Una buena representación del espacio reduce el número de dimensiones con las que tiene que lidiar un planificador. Los algoritmos de planificación de rutas generalmente funcionan en casi cualquier representación del espacio de configuración, aunque como con cualquier algoritmo, algunos métodos funcionan mejor en ciertas estructuras de datos.

El método de cuadrícula regular superpone una cuadrícula cartesiana 2D del espacio. Si hay algún objeto en el área de un cuadrante, ese elemento se marca ocupado, debido a esto el método es fácil de aplicar. El centro de cada elemento en la cuadrícula puede convertirse en un nodo, lo que lleva a un grafo altamente conectado. Las cuadrículas se consideran con 4 u 8 conexiones, dependiendo de si se permite que un arco se dibuje diagonalmente entre nodos o no.

Desafortunadamente, este método no esta exento de problemas. Primero, introducen un sesgo de digitalización, lo que significa que si un objeto cae incluso en la porción más pequeña de un cuadrante, todo el elemento se marcara como ocupado. Esto conduce a espacio desperdiciado y a objetos muy irregulares. Para reducir el espacio desperdiciado, los cuadrantes para la representación del espacio son de un tamaño reducido. Esta pequeña representación conlleva un alto costo de almacenamiento y a una gran cantidad de nodos a considerar para un algoritmo de planificación de rutas.

No obstante, contar con un mapa detallado del entorno por donde debe navegar el robot no es siempre posible. En consecuencia, por todo lo dicho anteriormente, para que un robot móvil pueda ser realmente autónomo deberá contar con la habilidad esencial de explorar un entorno y crear un mapa de él. Sólo así podrá ser capaz de navegar eficientemente por cualquier espacio mientras realiza la misión que se le ha encomendado (Gil et al., 2008).

2.2.2. Localización y mapeo

El problema de crear un mapa mientras el robot se localiza dentro de él se denomina *SLAM* (por las siglas en inglés de localización y mapeo simultáneos). Dicho de otra manera, el problema del SLAM trata sobre la construcción de un mapa del entorno utilizando una secuencia de medidas obtenidas por un robot en movimiento; asimismo se hace uso de diversos métodos probabilísticos con el fin de resolver estos problemas, como el Filtro de Kalman o el Filtro de Partículas, estos sirven para reducir los errores que se generan debido a la incertidumbre del movimiento del robot, propio del modelo que lo gobierna.

Los algoritmos de SLAM no son los mismos en todas las condiciones, esto depende del lugar en el cual se va a desenvolver el robot, teniendo así algoritmos para entornos cerrados y estructurados, entornos cerrados y cambiantes, etc. Es así que la incertidumbre en tales ambientes va a ser mayor o menor dependiendo de ellos. Aquí radica una de las muchas complejidades que presenta el desarrollar un algoritmo robusto; otra dificultad a tomar en cuenta es el coste computacional de implementar un algoritmo de SLAM, más aun en tiempo real, ya que conforme aumenta el tamaño del mapa, también aumenta la cantidad de datos a procesar, y con esto aumenta cuadráticamente el número de operaciones del proceso. Aunque la solución por Filtro de Kalman Extendido brinda buenos resultados, si se aplica en entornos de grandes dimensiones, el coste computacional puede llegar ser demasiado alto (Narváez et al., 2014).

Filtro de Kalman Extendido

El Filtro de Kalman Extendido (EKF por sus siglas en inglés) es un método para estimar la posición inicial real del robot. El EKF considera que se tiene un modelo del sistema y un modelo de observación, es decir, un modelo que relaciona los estados del sistema con las mediciones realizadas.

El EKF sirve para estimar los estados del sistema cuando se tiene ruido presente tanto al modelo (ruido de proceso) como a las señales medidas (ruido de medición). La estimación con EKF consiste de dos etapas principales: predicción y actualización. En la primera etapa se estiman los estados de acuerdo con el modelo, es decir, como si no se tuviera ruido de proceso. En la etapa de actualización se corrigen las estimaciones con base en la diferencia entre las salidas estimadas y las medidas. Para poder localizar al robot empleando el EKF se requiere de una serie de mediciones y movimientos (Bar-Shalom et al., 2004).

El Filtro de Kalman es un algoritmo computacionalmente eficiente cuyas ecuaciones matemáticas minimizan el error producido por el ruido inherente de las mediciones. Con la estimación es posible conocer con precisión del estado pasado, presente y futuro de un sistema, incluso si éste tiene un modelo de naturaleza desconocida (Huang, 2019).

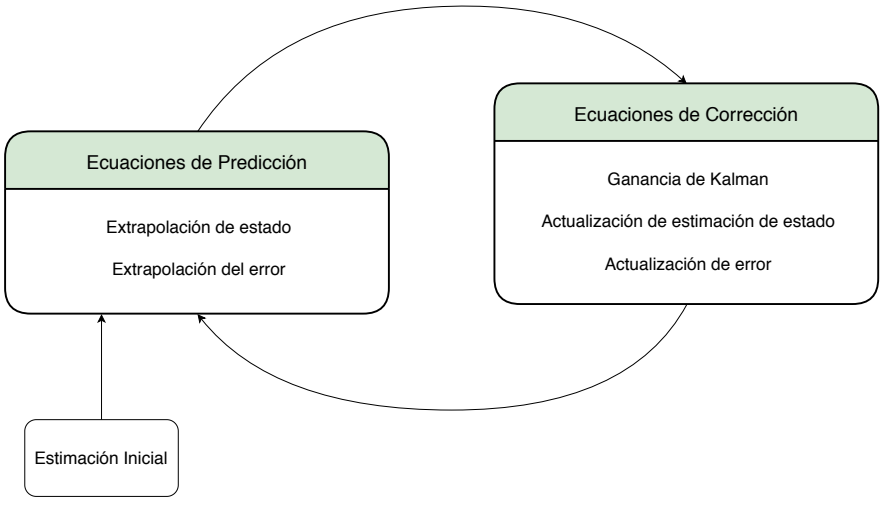


Figura 2.3: Diagrama del Filtro de Kalman

Filtro de Partículas

Los filtros de partículas intentan aproximar la distribución de probabilidad del estado del sistema utilizando métodos de Montecarlo. Para esto, mantienen un conjunto de partículas que son muestras de la distribución a actualizar. Al igual que el filtro de Kalman, actualizan esta distribución a medida que se encuentra disponible nueva información de sensado.

La dinámica de actualización de la función de densidad del filtro de partículas es similar al Filtro de Kalman. Esta consta de dos pasos, el de predicción y de actualización (Bar-Shalom et al., 2004).

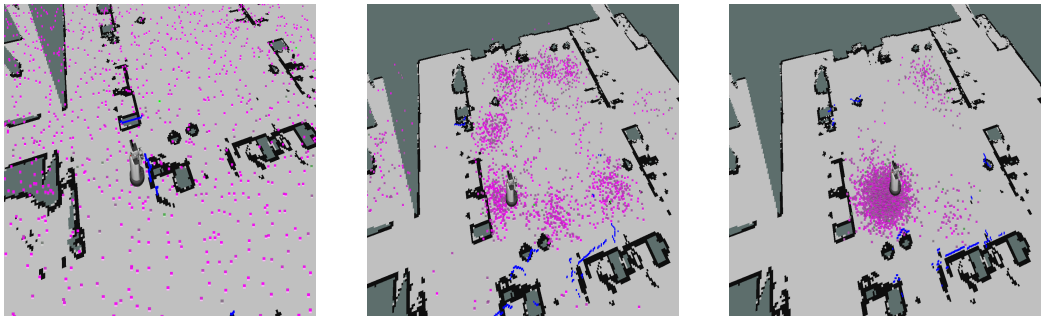


Figura 2.4: Proceso del Filtro de Partículas

Capítulo 3

Métodos basados en grafos

Los grafos se pueden usar para modelar un número sorprendentemente grande de problemas.La teoría de grafos es fundamental para la comprensión y resolución de problemas como también para el análisis de algoritmos; en matemáticas, ingeniería y ciencias de la computación, los grafos son colecciones de objetos llamados vértices (estados o nodos) conectados por líneas llamadas aristas que son el camino establecido para pasar de un vértice a otro (Thulasiraman and Swamy, 2011). Dependiendo del problema que se desee modelar se pueden usar diferentes tipos de grafos.

Este tipo algoritmos suponen que pueden acceder a toda la estructura de un grafo a través de una matriz o lista. Cuando la búsqueda se realiza en un espacio desconocido, se debe de obtener esta información mediante una búsqueda preliminar, mientras esta progresa se va generando una imagen parcial a partir de los nodos que se están explorando, en cada iteración un nodo se expande para generar todos los nodos adyacentes a los que se pueden llegar a través de él (Edelkamp and Schroedl, 2011).

3.1. Búsquedas a la ancho y profundidad

Se utilizan principalmente cuando se tiene poca información del entorno que se esta explorando. Se conoce la posición actual y cuantos posibles nodos se pueden explorar partiendo del estado actual. Si se encuentra la meta, la búsqueda termina pero de lo contrario el algoritmo seguirá buscando en todos los posibles estados no explorados.

Búsqueda a lo ancho

También conocida como búsqueda en amplitud, consiste en explorar todos los nodos sucesores al inicial, si ninguno de estos nodos es el deseado, se exploran los descendientes (hijos) de estos, así hasta el encontrar el deseado (si existe), o hasta recorrer todos los nodos disponibles. En algunas ocasiones se refieren a esta como búsqueda por niveles, por lo general se explora cada nivel de izquierda a derecha. En la figura 3.1 se muestra un diagrama de árbol (ó estados) donde cada nodo está representado por letras, siendo A (marcado con verde) el nodo inicial y el nodo M (marcado con rojo) el deseado.

Al realizar este tipo de búsqueda sobre el diagrama anterior, se puede observar una secuencia de exploración como la mostrada en la figura 3.2. Donde los nodos resaltados son los que han sido explorados mientras que los opacos solo han sido vistos. Se puede notar que para encontrar el nodo deseado se tubo que explorar una gran cantidad de estos.

Búsqueda en profundidad

Esta es muy similar a la búsqueda en amplitud, ya que se debe de explorar cada nodo hasta encontrar el deseado. En este caso se explora el primer nodo, si este no es el deseado la exploración continuará con cada uno de los descendientes, así sucesivamente mientras no se encuentre el nodo objetivo, si no se llega a la solución y se alcanza el ultimo nodo hijo, se realiza un

retroceso a un estado previo, si este estado no cuenta con mas descendientes a explorar el retroceso continua hasta encontrar un estado en el cual continuar la exploración. Esto continuará hasta llegar al nodo deseado, o cuando se se exploren todos los nodos. Podemos observar este procedimiento en la figura 3.3

La diferencia principal entre la búsqueda a lo ancho y la búsqueda en profundidad es la política para la inserción de nodos y la selección del siguiente nodo a explorar. En la búsqueda a lo ancho, la lista abierta es una cola y en la búsqueda en profundidad, es una pila.

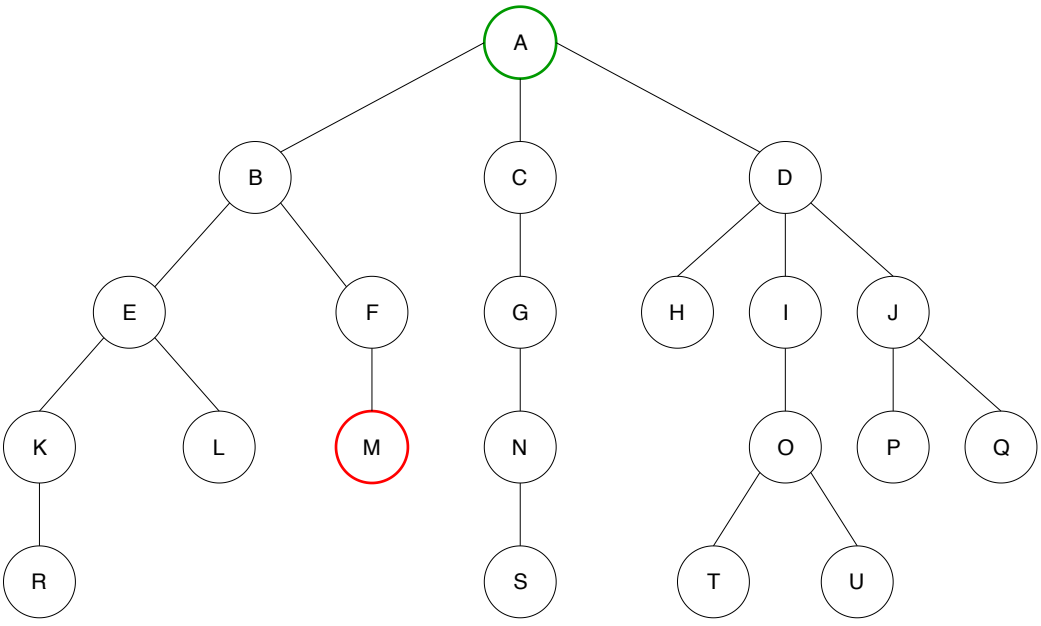


Figura 3.1: Diagrama de árbol

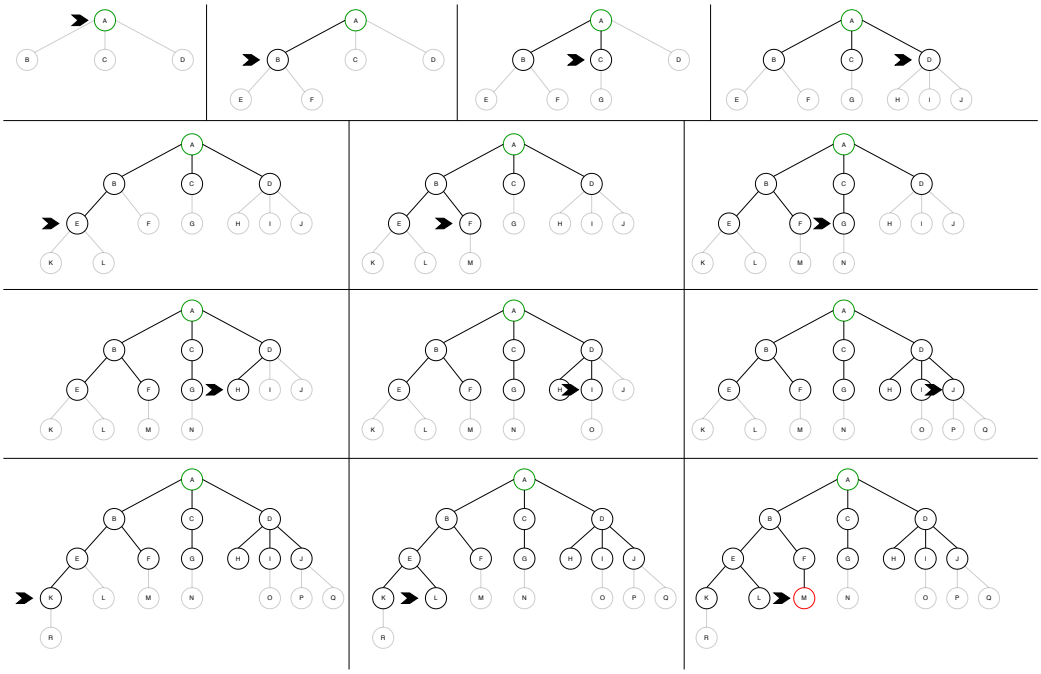


Figura 3.2: Exploración por amplitud

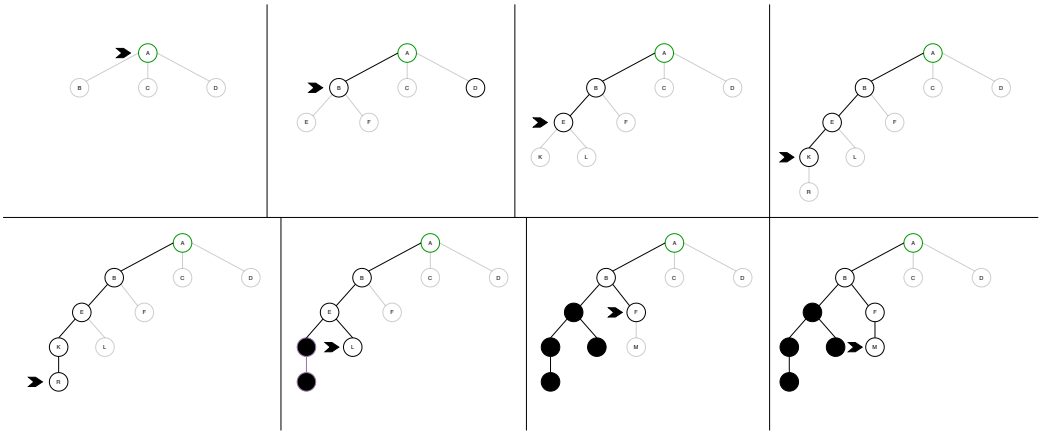


Figura 3.3: Exploración por profundidad

Algoritmo 1: Búsqueda a lo ancho

Datos: Espacio(Grafo,Mapa), Nodo inicial, Nodo objetivo
Resultado: Ruta encontrada

```
1  Lista;
2  Visitados[g];
3  Marcado[g];
4  Distancia[g];
5  Ruta[g];
6  Nodo_actual = Nodo_inicial;
7  Lista.agregar(Nodo_inicial);
8  Visitados[Nodo_inicial] = Cierto;
9  Distancia[Nodo_inicial] = 0;
10 mientras Nodo_actual ≠ Nodo_objetivo hacer
11 |   Nodo_actual = Lista.tomar_primer_nodo;
12 |   Marcado[Nodo_actual] = Cierto;
13 |   Vecinos = [Izquierda, Derecha, Arriba, Abajo];
14 |   Dist = Distancia[Nodo_actual] + 1;
15 |   para n ∈ Vecinos hacer
16 |   |   si Dist < Distancia[n] entonces
17 |   |   |   Distancia[n] = Dist;
18 |   |   |   Ruta[n] = Nodo_actual;
19 |   |   fin
20 |   |   si n ∉ Visitados[n] entonces
21 |   |   |   Visitados[n] = Cierto;
22 |   |   |   Lista.agregar(n);
23 |   |   fin
24 |   fin
25 fin
```

Algoritmo 2: Búsqueda en profundidad

Datos: Espacio(Grafo,Mapa), Nodo inicial, Nodo objetivo
Resultado: Ruta encontrada

```
1  Lista;
2  Visitados[g];
3  Marcado[g];
4  Distancia[g];
5  Ruta[g];
6  Nodo_actual = Nodo_inicial;
7  Lista.agregar(Nodo_inicial);
8  Visitados[Nodo_inicial] = Cierto;
9  Distancia[Nodo_inicial] = 0;
10 mientras Nodo_actual ≠ Nodo_objetivo hacer
11 |  Nodo_actual = Lista.tomar_ultimo_nodo;
12 |  Marcado[Nodo_actual] = Cierto;
13 |  Vecinos = [Izquierda, Derecha, Arriba, Abajo];
14 |  Dist = Distancia[Nodo_actual] + 1;
15 |  para n ∈ Vecinos hacer
16 |  |  si Dist < Distancia[n] entonces
17 |  |  |  Distancia[n] = Dist;
18 |  |  |  Ruta[n] = Nodo_actual;
19 |  |  fin
20 |  |  si n ∉ Visitados[n] entonces
21 |  |  |  Visitados[n] = Cierto;
22 |  |  |  Lista.agregar(n);
23 |  |  fin
24 |  fin
25 fin
```

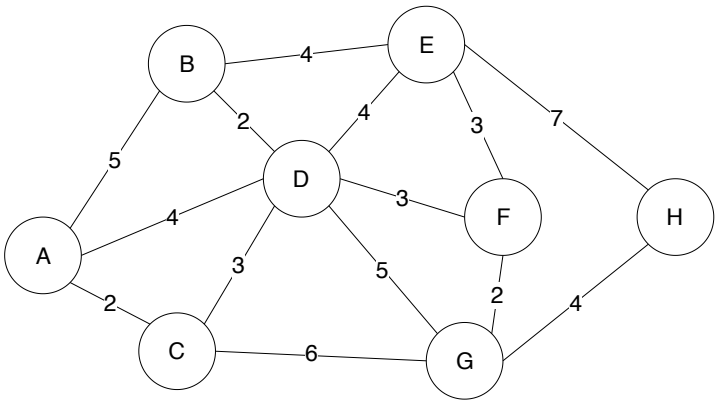


Figura 3.4: Grafo con pesos

3.2. Algoritmo de Dijkstra

Se trata de un algoritmo eficiente de complejidad $O(n^2)$, donde n es el número de vértices necesarios para encontrar el camino óptimo desde un nodo origen u hasta cualquiera de los demás nodos v del grafo(Torrubia and Terrazas, 2012).

Hay que tener en cuenta una serie de consideraciones a la hora de aplicar el algoritmo Dijkstra:

- Todos los pesos de las aristas deben ser positivos.
- Es necesario que el grafo sea conexo, es decir, para cualquier par de vértices o nodos u y v del grafo debe existir al menos una sucesión de nodos adyacentes, que no se repitan nodos, del nodo u al v.
- Sirve tanto para grafos dirigidos como no dirigidos.
- Es un algoritmo ávido que genera uno a uno los caminos de un nodo origen al resto en orden creciente de longitud.
  (Martorell Pons, 2017)

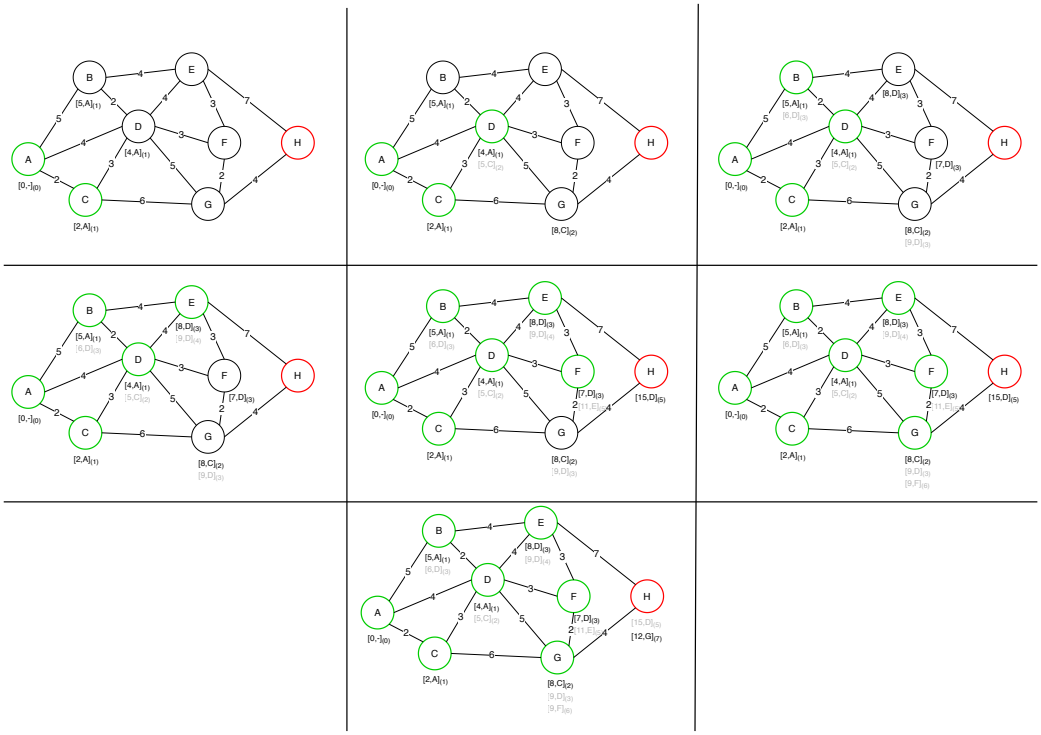


Figura 3.5: Pasos del algoritmo Dijkistra

En la figura 3.5 se describe el proceso del algoritmo de Dijkstra, donde en cada paso de denota al nodo evaluado de esta forma:

$$[P, N]_{(I)}$$

donde P representa el valor acumulado, el cual resulta de la suma del peso de cada arista recorrida hasta el nodo actual, N representa el nodo del cual se hizo el calculo, I representa el numero de iteración.

Se toma como punto de partida el nodo A y se marca como permanente, lo cual significa que no es necesario realizar mas operaciones sobre este, después se calcula el costo (peso) de moverse hacia sus nodos adyacentes, que en este caso son B, C y D, una vez evaluados se escoge el de menor valor siendo C el elegido y marcado como permanente, después de debe de hacer el mismo

proceso tomando como punto de partida este nuevo nodo permanente. Se puede observar que en algunas iteraciones se calcula el peso de una nueva ruta hacia un nodo que ya poseía un valor, cuando esto sucede se elige la de menor peso y la otra se descarta.

Algoritmo 3: Algoritmo de Dijkstra

```
   Datos: Espacio(Grafo,Mapa), Nodo inicial, Nodo objetivo
   Resultado: Ruta óptima encontrada
 1 Nodo_actual = Nodo_inicial;
 2 Lista.agregar(Peso, Nodo_inicial);
 3 Visitados[Nodo_inicial] = Cierto;
 4 Distancia[Nodo_inicial] = 0;
 5 mientras Nodo_actual ≠ Nodo_objetivo hacer
 6 |   Nodo_actual = Lista.tomar_nodo;
 7 |   Marcado[Nodo_actual] = Cierto;
 8 |   Vecinos = [Izquierda, Derecha, Arriba, Abajo];
 9 |   para n ∈ Vecinos hacer
10 |   |   Peso = Distancia[Nodo_actual] + 1;
11 |   |   si g < Distancia[n] entonces
12 |   |   |   Distancia[n] = Peso;
13 |   |   |   Ruta[n] = Nodo_actual;
14 |   |   fin
15 |   |   si n ∉ Visitados[n] entonces
16 |   |   |   Visitados[n] = Cierto;
17 |   |   |   Lista.agregar(Peso, n);
18 |   |   fin
19 |   fin
20 fin
```

3.3. Algoritmo A*

Es un algoritmo de búsqueda informada en el cual, comenzando desde el nodo origen hasta el objetivo, se genera una ruta óptima con cada iteración

y actualización; para realizar esto se consideran los nodos que podrían agregarse a la ruta y se elige el mejor para llegar al nodo destino. Se trata de una mejora del algoritmo Dijkstra mediante la implementación de una función heurística $h(n)$ dentro de la función de costo $f(n)$. Así la función del coste total es calculada como: $f(n) = g(n) + h(n)$, donde $f(n)$ mide qué tan buena es la ruta al nodo n, $g(n)$ es el coste de la trayectoria desde el nodo inicial al nodo n y $h(n)$ es la función heurística que estima el costo del mejor camino al nodo destino. Si $h(n) = 0$ se trata del algoritmo Dijkstra. Una característica importante de la función $h(n)$ es que siempre debe subestimar el costo real de llegar desde el nodo n al nodo meta.

En el siguiente ejemplo podemos observar cómo se aplica la formula usando distancia euclidiana. Suponga que una representación del espacio de configuración produjo el grafo de la figura 3.6. El algoritmo de búsqueda A* comienza en el nodo A y crea una estructura de árbol de decisión para determinar cuál es el mejor nodo posible que puede agregar a su ruta. Solo hay dos nodos para elegir: B y C. Para determinar qué nodo es el mejor para usar, el algoritmo de búsqueda A* evalúa el peso de añadir a B o C al observar los bordes. El valor de B como el próximo movimiento es:

$$f(B) = g(B) + h(B) = 1 + 2.24 = 3.24$$

donde $g(B)$ es el costo de ir de A a B, y $h(B)$ es el costo de ir de B a la meta E. El valor de C es:

$$f(C) = g(C) + h(C) = 1 + 1 = 2$$

donde $g(C)$ es el costo de ir de A a C, y $h(C)$ es el costo de ir de C a E. Como $f(B) > f(C)$, la ruta debe ir de A a C.

En el ejemplo de la figura 3.7 usará la distancia de Manhattan en un espacio de configuración representado por celdas y se utilizará una función de heurística diferente a la anterior:

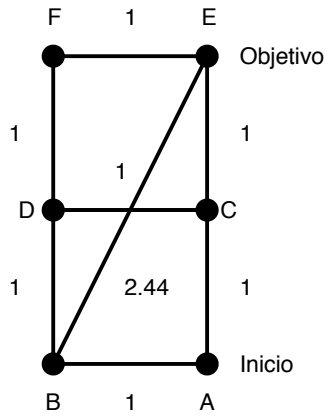


Figura 3.6: Espacio generado para A*

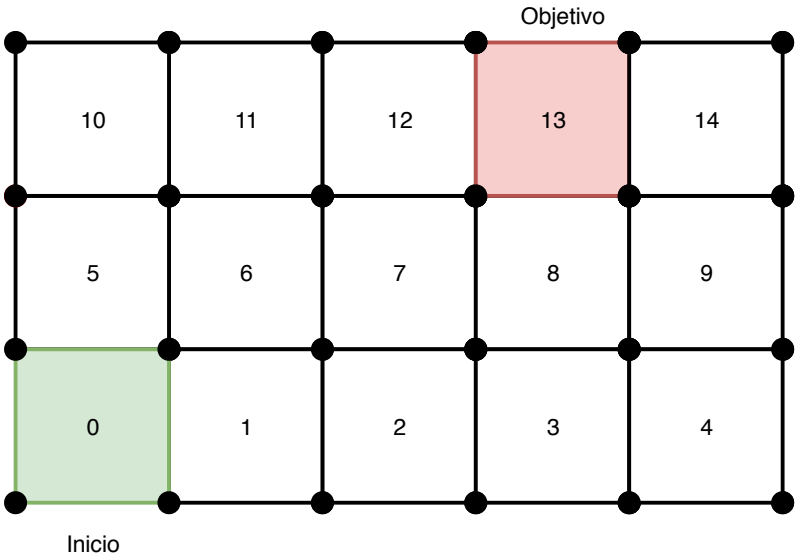


Figura 3.7: Espacio de celdas generado para A*

$$
\begin{aligned}
f(n) &= g(n) + h(n) \\
g(n) &= Acumulado + 1 \\
x &= |x_1 - x_2| \\
y &= |y_1 - y_2| \\
h(n) &= x + y
\end{aligned}
$$

donde x_1 y y_1 corresponden a la posición de la celda evaluada en el eje X y Y, mientras que x_2 y y_2 hacen referencia a la celda objetivo. El resultado de estas operaciones determina la distancia entre las celdas evaluadas, la cual será utilizada en la función de heurística.

En la figura 3.8 se muestra el proceso para encontrar una ruta óptima a través del espacio de celdas mostrado anteriormente, el cual consiste en evaluar las celdas alrededor del nodo actual, para este ejemplo se usará únicamente conectividad 4. Las celdas son evaluadas y se elige la que obtenga la $f(n)$ de menor costo, esta se marca con verde indicando que se añadió a la ruta que se está generando, mientras que las celdas que no fueron seleccionadas se marca con azul para evitar que se vuelvan a tomar en cuenta. Este proceso se repite hasta llegar a la celda objetivo marcada de color rojo.

Hay que tener en cuenta una serie de consideraciones a la hora de aplicar el algoritmo A*:

- Todos los pesos de las aristas deben ser positivos.
- Es necesario que el grafo sea conexo, es decir, para cualquier par de vértices o nodos u y v del grafo debe existir al menos una sucesión de nodos adyacentes, que no se repitan nodos, del nodo u al v.
- Sirve tanto para grafos dirigidos como no dirigidos.
- Debe emplear una heurística admisible, es decir, que no sobre-estime la distancia entre el nodo actual y el nodo destino.

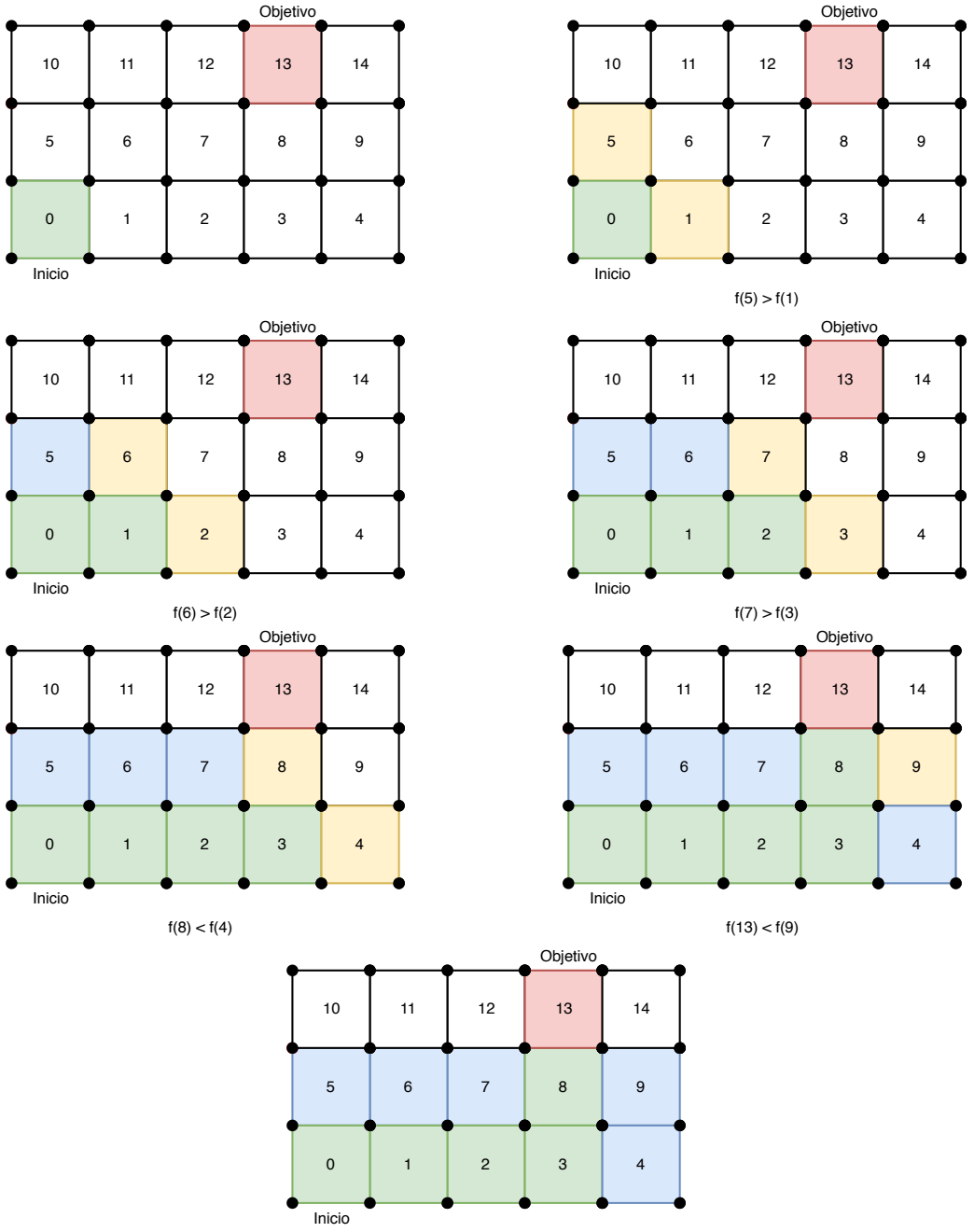


Figura 3.8: Espacio de celdas generado para A*

Algoritmo 4: Algoritmo A*

Datos: Espacio(Grafo,Mapa), Nodo inicial, Nodo objetivo
Resultado: Ruta óptima encontrada

```
1  Nodo_actual = Nodo_inicial;
2  Lista.agregar(Peso, Nodo_inicial);
3  Visitados[Nodo_inicial] = Cierto;
4  Distancia[Nodo_inicial] = 0;
5  mientras Nodo_actual ≠ Nodo_objetivo hacer
6  |  Nodo_actual = Lista.tomar_nodo;
7  |  Marcado[Nodo_actual] = Cierto;
8  |  Vecinos = [Izquierda, Derecha, Arriba, Abajo];
9  |  para n ∈ Vecinos hacer
10 |  |  g = Distancia[Nodo_actual] + 1;
11 |  |  h = Distancia Euclidiana o Manhattan al Nodo_objetivo;
12 |  |  f = g + h;
13 |  |  si g < Distancia[n] entonces
14 |  |  |  Distancia[n] = g;
15 |  |  |  Costo[n] = f;
16 |  |  |  Ruta[n] = Nodo_actual;
17 |  |  fin
18 |  |  si n ∉ Visitados[n] entonces
19 |  |  |  Visitados[n] = g;
20 |  |  |  Lista.agregar_prioridad(Costo[n], n);
21 |  |  fin
22 |  fin
23 fin
```

En este capítulo se describieron métodos basados en búsqueda en grafos. Los cuatro métodos expuestos tienen el mismo principio y varían solo en la estructura que se maneja en la lista abierta y en la política para seleccionar el siguiente nodo a expandir. En el siguiente capítulo se describe un conjunto diferente de métodos: los algoritmos de planeación basados en muestreo.

Capítulo 4

Métodos basados en muestreo

En este capítulo se describen algunas técnicas de muestreo, sus características y funciones. Posteriormente se explica en qué consiste algoritmo RRT, sus variantes y mejoras, al igual que sus aplicaciones en la exploración y planeacion de rutas.

4.1. Muestreo

El muestreo es una herramienta para seleccionar una parte de la población, cuya observación permita extender la información obtenida del conjunto de estudio. Para que las conclusiones sobre la población sean adecuadas es necesario que la selección de las unidades se realice de tal manera que sean lo más representativas posible, para esto es fundamental planificar adecuadamente el método usado para la selección (Azorín, 1994).

4.1.1. Muestreo aleatorio

En este tipo de muestreo todos los elementos tienen la misma probabilidad de ser elegidos. Los individuos que formarán parte de la muestra se elegirán al azar mediante números aleatorios. También puede realizarse de maneras

distintas, las más frecuentes son el muestreo simple, sistemático, estratificado y por conglomerados (Casal, 2003).

- Muestreo aleatorio simple: Es el método conceptualmente más simple. Consiste en extraer todos los individuos al azar de un conjunto.
- Muestreo sistemático: En este caso se elige el primer individuo al azar y el resto esta condicionado por este.
- Muestreo aleatorio estratificado: Se divide la población en grupos en función de un carácter determinado y después se muestrea cada grupo aleatoriamente, para obtener una parte proporcional de la muestra. Este método se aplica para evitar que por azar algún grupo este menos representado que otro.
- Muestreo aleatorio por conglomerados: Se divide la población en varios grupos de características parecidas similares y luego se analizan completamente algunos de los grupos, descartando los demás. Dentro de cada conglomerado existe una variación importante, pero los distintos grupos son parecidos. Requiere una muestra más grande, pero suele simplificar la obtención de estas.
- Muestreo mixto: Cuando la población es compleja, cualquiera de los métodos descritos puede ser difícil de aplicar, en estos casos se aplica un muestreo mixto que combina dos o más de los métodos anteriores sobre distintas unidades de la muestra.

4.1.2. Campos potenciales

Son una herramienta utilizada para establecer campos imaginarios de repulsión al rededor de los obstáculos como se muestra en la figura 4.1. Los campos pueden variar de acuerdo a la distancia que existe hasta ellos o geométricamente de acuerdo a un valor definido (Holland, 2004). Este método

fue propuesto inicialmente por (Khatib, 1986) para el control y planeación de movimientos de robots manipuladores, sin embargo, actualmente se utiliza en muchos otros tipos de robots, principalmente para evasión de obstáculos.

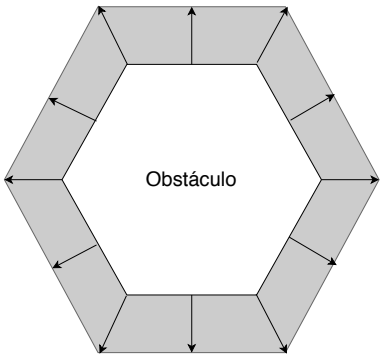


Figura 4.1: Campos potenciales

4.1.3. Planificación con generación aleatoria

Gracias al desarrollo del método de campos potenciales surgió la primera aproximación a los métodos de planificación aleatoria (Latombe et al., 1991). Los campos potenciales son un procedimiento que genera una trayectoria de acuerdo a un vector gradiente derivado del campo potencial artificial. Esta técnica permite que el sistema siga siempre la dirección que minimiza el valor del campo potencial. El objetivo es alcanzar el mínimo absoluto que estará situado en la configuración destino. Sin embargo, el entorno donde se desarrolla el método puede tener una serie de mínimos locales, más cercanos que el mínimo absoluto. El problema aparece cuando el sistema alcanza uno de estos. En este caso, el potencial no es nulo, pero el gradiente mantiene al sistema en la configuración alcanzada (López et al., 2006).

Para resolver este inconveniente se plantearon movimientos aleatorios que permitan que el sistema abandone el mínimo local y, a continuación, aplicar de nuevo el método del gradiente. Este proceso continuará hasta hallar un nuevo mínimo (Latombe et al., 1991).

El éxito de los métodos aleatorios sugirió la posibilidad de usar dichas técnicas de forma exclusiva, eliminando así el coste del procesamiento para el cálculo del campo potencial. Estas debían de ser más simples, para competir en velocidad y suplir la carencia de una inteligencia en la búsqueda de un camino o ruta. Uno de estos métodos es el llamado método de árboles de exploración rápida ("Rapidly Exploring Random Trees", RRT) (LaValle, 1998).

4.2. Algoritmo RRT

Este algoritmo se basa en la construcción de un árbol de configuraciones que crece explorando el espacio a partir de un punto origen.

Para entender este algoritmo se usarán los siguientes conceptos:

- C es el conjunto de todas las configuraciones posibles en un espacio dado.
- C_{libre} es el subconjunto de C, donde las configuraciones que no tienen intersección con ninguno de los obstáculos existentes en dicho espacio.
- ϵ es una métrica definida dentro de C, que corresponde a la longitud del segmento de crecimiento entre las configuraciones. Puede ser distancia Euclideana, Manhattan u otra ponderación de proximidad que pueda usarse.
- q_{ini} es la configuración inicial (en el caso de un robot que se mueve en un plano, representa las coordenadas x, y de un punto de referencia y la orientación del vehículo respecto a uno de los ejes del sistema).
- q_{fin} es la configuración que se desea alcanzar.
- q_{ale} es una configuración aleatoria que genera el algoritmo.

- $q_{cercano}$ es la configuración más cercana a q_{ale}, de entre las existentes en un árbol, en el sentido definido por ϵ.

- q_{nuevo} es la configuración que se va a añadir al árbol.

El objetivo del método RRT consiste en construir un árbol de exploración que cubra uniformemente todo el espacio disponible. Para ello, se desarrolló el método que se muestra en el algoritmo 5 (LaValle, 1998).

Algoritmo 5: Algoritmo de RRT

Datos: Espacio(Mapa), q_{ini} Punto origen
Resultado: Espacio explorado

```
1 Árbol[0] = q_ini;
2 k_ini = 0;
3 mientras k_ini ≠ k_max hacer
4 |   q_ale = ConfiguracionAleaoria();
5 |   Extiende(Árbol, q_ale);
6 |   k_ini + +;
7 fin
8 devolver Árbol;
```

Dicho algoritmo tiene como objetivo seleccionar un punto (q_{ale}) de forma aleatoria y extender un árbol de configuraciones hacia este. Para ello se hace uso de la función *Extiende*(), dicha función tiene la finalidad de ampliar el árbol en el sentido que marca q_{ale}, determinando en el proceso la existencia de un camino libre de colisiones (López et al., 2006). El pseudocódigo de esta función se presenta en el Algoritmo 6.

El algoritmo comienza inicializando la tabla asociada al árbol con la configuración origen. A continuación, se entra en un bucle, limitado por un valor K_{max}, cuya función es finalizar el algoritmo una vez se ha realizado un número prefijado de iteraciones. Este valor, se utilizará posteriormente para parar el algoritmo en el caso en que no se alcance la configuración final.

Dentro del bucle del algoritmo RRT existen dos funciones. Con la prime-

ra se obtiene un punto al azar dentro del espacio libre de colisión (C_{libre}); la segunda hace crecer el árbol en dirección a la configuración aleatoria anteriormente obtenida.

Algoritmo 6: Función Extiende()

Datos: Árbol, q_{ale}
Resultado: Nuevo punto aceptado o rechazado

```
1 q_cercano = NodoMásCercano(Árbol, q_ale);
2 q_nuevo = NuevoNodo(q_cercano, q_ale);
3 si Obstaculo(q_nuevo) entonces
4 |   AñadeVértice(q_nuevo, Árbol);
5 |   devolver Continuar;
6 en otro caso
7 |   devolver Rechazado;
8 fin
```

El crecimiento del árbol se realiza con la función *Extiende*(). El procedimiento se realiza con el cálculo de $q_{cercano}$, el cual es el punto mas cercano a q_{ale}, este valor se obtiene usando la función *NodoMasCercano*(), la cual se muestra en el algoritmo 7.

Algoritmo 7: Función NodoMásCercano()

Datos: Árbol, q_{ale}
Resultado: Nodo mas cercano a q_{ale}

```
1 Nodo = Árbol[0];
2 para n ∈ Árbol hacer
3 |   si Distancia(n, q_ale) < Distancia(Nodo, q_ale) entonces
4 |   |   Nodo = n;
5 |   fin
6 fin
7 devolver Nodo;
```

Posteriormente, la función *NuevoNodo*() calcula q_{nuevo}, el cual será el nuevo nodo a agregar mediante un salto de tamaño ϵ en dirección a q_{ale}, como

se muestra en la figura 4.2. Para poder obtener q_{nuevo} se toma en cuenta si existe alguna colisión en el desplazamiento, de ser así el nuevo nodo no se agregará al Árbol.

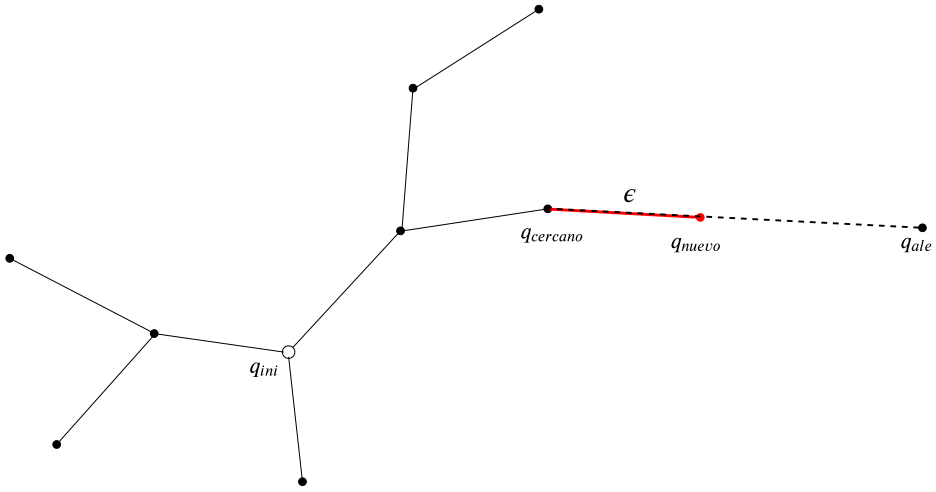
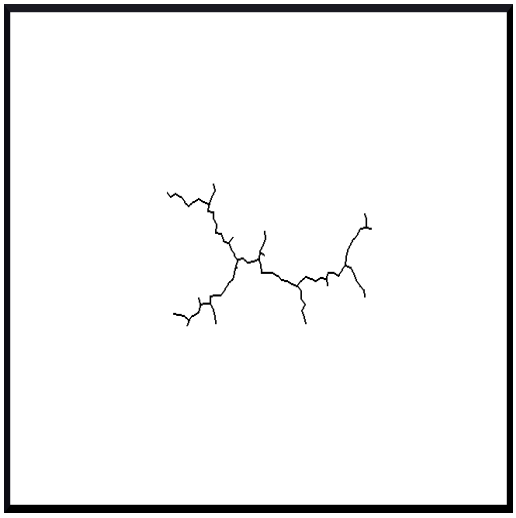


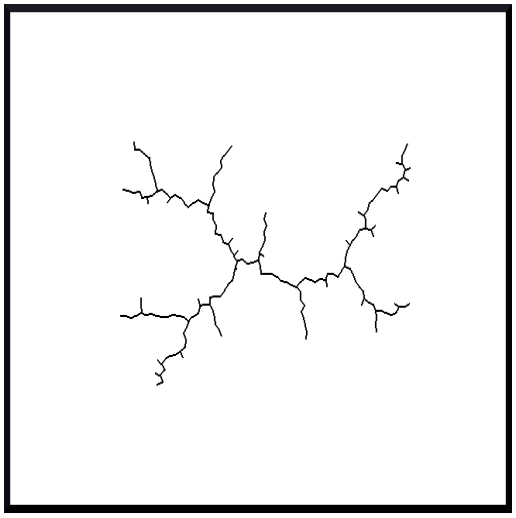
Figura 4.2: Crecimiento del árbol

El comportamiento de este algoritmo es mejor con respecto a otros en cuanto a la homogeneidad del espacio explorado (López et al., 2006). La naturaleza del algoritmo RRT le permite extenderse hacia zonas inexploradas con mayor facilidad, ya que ahí es más probable encontrar puntos q_{ale} factibles. Esto puede observarse con el desarrollo del método en distintos entornos.

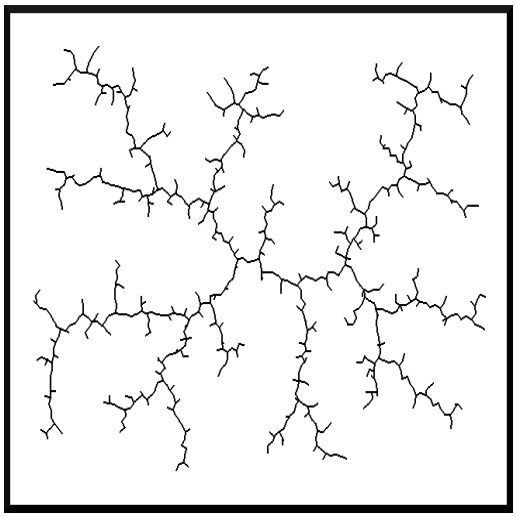
En los ejemplos de la figura 4.3 se muestra el desarrollo del algoritmo en un espacio totalmente libre de colisiones. Su crecimiento es totalmente aleatorio, sin ningún tipo de preferencia en cuanto a su dirección. Un hecho importante a observar es la inexistencia de una alta densidad de ramas en punto inicial del árbol, lo cual es una propiedad del RRT, esto puede ser especialmente efectivo en espacios en los que se utilicen campos potenciales.



(a) RRT 100 iteraciones



(b) RRT 200 iteraciones



(c) RRT 800 iteraciones

Figura 4.3: RRT

En la figura 4.4, se representa la progresión del algoritmo RRT en un espacio diferente, donde se observa que desde el punto de origen ubicado en el centro del rectángulo pequeño se genera una rama que crece en mayor proporción que las demás. Esto se debe a la alta probabilidad de encontrar puntos aleatorios en una mayor área inexplorada libre de obstáculos.

Tras las suficientes iteraciones, puede observarse que la densidad de las ramas es bastante homogénea, muy similar al caso anterior a pesar de la gran variación del espacio disponible.

Todo esto permite apreciar que el algoritmo RRT es una buena herramienta para explorar un espacio desconocido, debido a que este no es muy demandante en cuanto a recursos, y resulta más conveniente si se utiliza en conjunto con otros métodos.

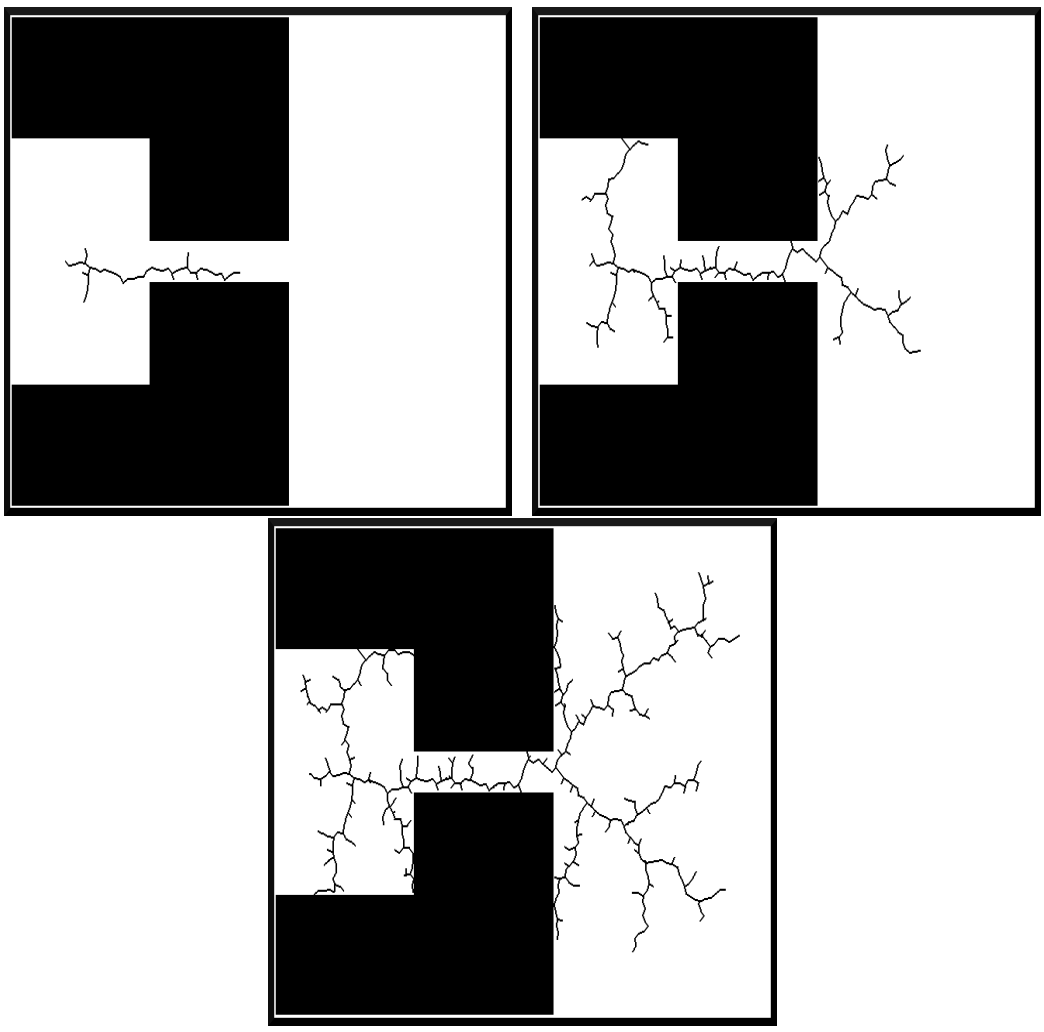


Figura 4.4: RRT en un entorno asimétrico

Los primeros intentos de utilizar el RRT como planificador de rutas autónomo consistían en generar un árbol tomando como nodo raíz al punto de partida, posteriormente el desarrollo del algoritmo explora el espacio vacío, aproximándose eventualmente a la configuración final deseada.

Con un numero de iteraciones suficientemente grande algunas de las nuevas ramas agregadas se acercarán al punto objetivo para obtener una trayectoria. Sin embargo, las aplicaciones de este procedimiento pueden resultar demasiado lentas.

A causa de esto se han desarrollado algunas extensiones para mejorar el método original, las cuales permiten establecer un camino entre una configuración inicial y una objetivo (López García et al., 2011). .

RRT-GoalBias

Se trata de la primera modificación introducida para mejorar el rendimiento de la planificación de rutas, consiste en sustituir q_{ale} por q_{fin} en una fracción de las iteraciones, o asignar una función de probabilidad que cambie entre q_{ale} y la configuración final (q_{fin}) (LaValle et al., 2001).

Esta versión, incluso con una baja probabilidad, resulta mucho más rápida que el RRT sin modificar. Aunque dicha función se muestra determinante en el comportamiento del planificador, debido que si este valor es muy elevado puede caer en problema de mínimos locales.

RRT-GoalZoom

Esta variación del algoritmo surgió para evitar el problema del método anterior, la cual consiste en no usar exactamente la configuración final (q_{fin}), si no puntos cercanos a esta. La función para generar el punto q_{ale} toma dos posibles casos: que q_{ale} sea un punto del espacio libre, o que sea un punto cercano a la configuración final dentro de una vecindad de radio R_{max}.

De esta manera en vez de tener un valor q_{fin} se obtiene una esfera en el

espacio capaz de atraer el crecimiento del árbol. El radio de dicha esfera se actualiza en cada iteración, por lo que durante el crecimiento del árbol los puntos q_{ale} surgen cada vez mas cerca del objetivo, mejorando así el rendimiento del algoritmo en comparación con el anterior. Sin embargo aun con estas modificaciones el método no esta exento del todo de caer en mínimos locales.

Debido a que estas versiones del RRT solo se ocupan de generar un árbol para explorar el espacio disponible, no son el método mas adecuado para la planeación de rutas. Por lo anterior, se abordarán más extensiones y mejoras.

RRT-Bidireccional

Esta adaptación del algoritmo se basa en la construcción de dos árboles que parten de la configuración inicial y objetivo de manera simultanea, estos crecen explorando el espacio buscándose entre si, este proceso concluye cuando dichos árboles se conectan, obteniendo de esta manera una trayectoria. Si el valor de k_{max} es alcanzado y los árboles no han coincidido, se devuelve un mensaje de error, esto puede suceder cuando el espacio a explorar es demasiado a grande, o no existe una posible ruta.

Cabe mencionar que ambos árboles comparten el mismo punto q_{ale} para crecer, de esta forma se reduce el coste computacional de calcular un nuevo valor, sin afectar las propiedades del RRT. La funcion $IntercambiarArbol()$ tiene como tarea alternar el crecimiento de los arboles, para que ambos se desarrollen de manera equilibrada, de lo contrario el $ArbolB$ solo crecería cuando el $ArbolA$ no presente ningún problema.

Este algoritmo admite ciertas mejoras, por ejemplo el crecimiento de los árboles puede ser dirigido de forma que tiendan el uno hacia el otro, en lugar de crecer por zonas donde la interconexión puede resultar difícil. Para conseguir esto se ha desarrollado el algoritmo RRT-Ext (LaValle et al., 2001).

Algoritmo 8: Algoritmo de RRT-Bidireccional

Datos: Espacio(Mapa), q_{ini} Punto origen, q_{fin} Punto final
Resultado: Ruta Obtenida

```
1  ÁrbolA[0] = q_ini;
2  ÁrbolB[0] = q_fin;
3  k_ini = 0;
4  Cruzar = False;
5  mientras k_ini ≠ k_max & !Cruzar hacer
6      q_ale = ConfiguracionAleaoria();
7      si Extiende(ÁrbolA, q_ale) ≠ Rechazado entonces
8          si Extiende(ÁrbolB, q_ale) == Final entonces
9              Cruzar = Verdadero;
10             devolver Camino(ÁrbolA, ÁrbolB);
11         fin
12     fin
13     IntercambiarÁrbol(ÁrbolA, ÁrbolB);
14     k_ini++;
15 fin
16 devolver Árbol;
```

4.2.1. RRT-Ext

Esta versión permite agilizar la conexión entre los árboles del algoritmo anterior. Añadiendo algunas variaciones, es posible atribuirle la capacidad de que cada árbol pueda dirigir su crecimiento hacia su homólogo, de igual forma se agrega una función de búsqueda, la cual trata de encontrar una intersección entre ambos árboles cada que una nueva configuración es agregada. De esta manera se logra que este método sea más eficaz que otros puramente aleatorios.

Estos cambios pueden verse en el Algoritmo 9, donde el punto q_{ale} se sus-

tituye en el segundo árbol por q_{nuevo}, es decir, el punto recién agregado al árbol anterior.

Algoritmo 9: Algoritmo de RRT-Ext

Datos: Espacio(Mapa), q_{ini} Punto origen, q_{fin} Punto final
Resultado: Ruta Obtenida

```
1  ÁrbolA[0] = q_ini;
2  ÁrbolB[0] = q_fin;
3  k_ini = 0;
4  Cruzar = False;
5  mientras k_ini ≠ k_max & !Cruzar hacer
6      q_ale = ConfiguracionAleaoria();
7      si Extiende(ÁrbolA, q_ale, ÁrbolB) == Final entonces
8          Cruzar = Verdadero;
9          devolver Camino(ÁrbolA, ÁrbolB);
10     fin
11     si !Cruzar entonces
12         si Extiende(ÁrbolB, q_nuevo, ÁrbolA) == Final entonces
13             Cruzar = Verdadero;
14             devolver Camino(ÁrbolA, ÁrbolB);
15         fin
16     fin
17     IntercambiarÁrbol(ÁrbolA, ÁrbolB);
18     k_ini + +;
19 fin
20 devolver Árbol;
```

También se incluyen algunas alteraciones para adaptarse a las modificaciones de la función *Extiende*() las cuales se muestran en el algoritmo 10, este ahora cuenta con la función $Unir()$ que intenta conectar los árboles cada que un nuevo nodo es agregado.

Algoritmo 10: Función Extiende()

Datos: Árbol, q_{ale}, ÁrbolC
Resultado: Nuevo punto aceptado o rechazado

```
1  q_cercano = NodoMásCercano(Árbol, q_ale);
2  q_nuevo = NuevoNodo(q_cercano, q_ale);
3  si Obstaculo(q_nuevo) entonces
4  |   AñadeVértice(q_nuevo, Árbol);
5  |   si Unir(q_nuevo, ÁrbolC) entonces
6  |   |   devolver Final;
7  |   en otro caso
8  |   |   devolver Continuar;
9  |   fin
10 en otro caso
11 |   devolver Rechazado;
12 fin
```

En la función podemos ver el nuevo procedimiento para aceptar un nuevo nodo. Primero se evalúa si es posible agregarlo, después se comprueba si este es capas de conectar ambos árboles usando la función $Unir()$, en caso de ser así se retorna $'Final'$ y algoritmo termina, en caso contrario el método continua hasta encontrar un nodo que una los dos arboles o se alcance el numero máximo de iteraciones.

4.2.2. RRT-Connect

En esta variación del algoritmo RRT-Ext, se sustituye la función $Extiende()$, que agrega un nuevo nodo al árbol en cada interacción, por otra denominada $Conecta()$ (ver 13) la cual es más ambiciosa y agrega nodos consecutivos hasta alcanzar a q_{ale}, o que un obstáculo la detenga (Kuffner and LaValle, 2000).

Algoritmo 11: Algoritmo de RRT-Connect

Datos: Espacio(Mapa), q_{ini} Punto origen, q_{fin} Punto final
Resultado: Ruta Obtenida

```
1  ÁrbolA[0] = q_ini;
2  ÁrbolB[0] = q_fin;
3  k_ini = 0;
4  Cruzar = False;
5  mientras k_ini ≠ k_max & !Cruzar hacer
6      q_ale = ConfiguracionAleaoria();
7      si Conecta(ÁrbolA, q_ale, ÁrbolB) == Final entonces
8          Cruzar = Verdadero;
9          devolver Camino(ÁrbolA, ÁrbolB);
10     fin
11     si !Cruzar entonces
12         si Conecta(ÁrbolB, q_nuevo, ÁrbolA) == Final entonces
13             Cruzar = Verdadero;
14             devolver Camino(ÁrbolA, ÁrbolB);
15         fin
16     fin
17     IntercambiarÁrbol(ÁrbolA, ÁrbolB);
18     k_ini + +;
19 fin
20 devolver Árbol;
```

Ahora, los segmentos agregados son múltiples, con lo que para conectar basta con que exista un camino rectilíneo libre de colisiones entre el punto q_{nuevo} y el nodo más cercano del otro árbol.

Debido a estos cambios se debe agregar un nuevo caso en la función *Extiende*() (ver 12), el cual consiste en detectar cuando los nodos agregados alcanzaron a q_{ale}, en ese momento se detiene el ciclo de la función *Connecta*() y continua la siguiente parte del algoritmo *RRT-Connect*.

Algoritmo 12: Función Extiende()

Datos: Árbol, q_{ale}, ÁrbolC
Resultado: Nuevo punto aceptado o rechazado

```
1  q_cercano = NodoMásCercano(Árbol, q_ale);
2  q_nuevo = NuevoNodo(q_cercano, q_ale);
3  si Obstaculo(q_nuevo) entonces
4  |  AñadeVértice(q_nuevo, Árbol);
5  |  si Unir(q_nuevo, ÁrbolC) entonces
6  |  |  devolver Final;
7  |  si no, si q_cercano == q_nuevo entonces
8  |  |  devolver Alto;
9  |  en otro caso
10 |  |  devolver Continuar;
11 |  fin
12 en otro caso
13 |  devolver Rechazado;
14 fin
```

La única desventaja de esta variante consiste en el mayor coste computacional de la función *Conecta*(), que se compensa en muchos escenarios con la mayor eficiencia del algoritmo. En la práctica parece observarse un mejor comportamiento de este algoritmo con respecto al *RRT-Ext* cuando el entorno no está congestionado de obstáculos (Kuffner and LaValle, 2000).

Algoritmo 13: Función Conecta()

Datos: Árbol, q_{ale}, ÁrbolC
Resultado: Nuevo camino agregado

```
1 repetir
2 |  Camino = Extiende(Árbol, q_ale, ÁrbolC);
3 mientras Camino == Continuar;
```

En este capítulo se describieron varios métodos de planeación de rutas basados en muestreo. En el capítulo 3 se describieron métodos basados en búsqueda en grafos. Como se mencionó en la introducción, el objetivo de este trabajo es comparar diferentes métodos de planeación de rutas, para ello, en el siguiente capítulo se describe la forma en que estos algoritmos se implementaron así como las estrategias para compararlos.

Capítulo 5

Implementación

En este capitulo se explican las herramientas usadas para la implementación de los algoritmos descritos en los capítulos anteriores, comenzando con las características y objetivos de *ROS*, después se describe al robot *HSR* y al simulador *Gazebo*. También se muestran los nodos y servicios creados para realizar las pruebas, así como la interacción entre estos. Por ultimo se presentan los métodos usados para las estrategias de comparación.

5.1. La plataforma ROS

La Wiki define a ROS (`http://www.ros.org/`) como un meta-sistema operativo de código abierto (*open-source*) el cual proporciona los servicios necesarios para el desarrollo de aplicaciones en el ámbito de la robótica, los cuales se esperarían de un sistema operativo, tales como: abstracción de hardware, control de dispositivos de bajo nivel, envió de mensajes entre procesos y gestión de distintos tipos de paquetes. También proporciona herramientas y bibliotecas para obtener, crear, escribir y ejecutar código en varias computadoras.

ROS es la abreviatura de *Robot Operating System* (Sistema Operativo de Robots traducido al español). Sería lógico pensar que se trata de un siste-

ma operativo, sin embargo una descripción más precisa sería la de un *Meta Sistema Operativo*, aunque no es un término definido en el diccionario, describe a un sistema que realiza procesos tales como programación, ejecución, monitoreo y manejo de errores utilizando una capa de virtualización entre aplicaciones y recursos informáticos distribuidos. (Yoonseok Pyo, 2017)

Por lo tanto, ROS no es un sistema operativo convencional como *Windows, Linux, o Android*, sino una plataforma que se ejecuta dentro de un sistema operativo existente. A menudo, se utiliza *Ubuntu*, que es una de las distribuciones de Linux. No obstante, es posible usarse en distintos sistemas, tal y como se muestra la Figura 5.1.

Figura 5.1: ROS en distintos sistemas

Objetivos de ROS

Existen diversas plataformas de software para la robótica (OpenRTM, OPRoS, Player, YARP, Orocos, CARMEN, Orca, MOOS, Microsoft Robotics Studio), las cuales tienen distintos propósitos y objetivos. ROS en especifico

se puede decir que se centra en construir un entorno que permita el desarrollo de software robótico a nivel mundial. Es decir, ROS se centra en maximizar la reutilización de código en la investigación y el desarrollo de la robótica, y de esta forma hacer crecer el entorno mismo (Quigley et al., 2009). Para lograr esto el sistema tiene las siguientes características:

- Distribución de procesos: Están programados en unidades mínimas de procesamiento (nodos). Cada uno de estos procesos se ejecuta de manera independiente y es capaz de intercambiar datos con otros de manera sistemática.

- Manejo de paqueterías: Cuando varios procesos tienen propósitos similares, estos se manejan dentro de un paquete que haga los procesos más fáciles de usar, desarrollar, modificar y distribuir.

- Repositorios públicos: Cada paquete se hace público dentro de un repositorio (por ejemplo GitHub) para que la comunidad de desarrolladores puedan acceder a él.

- API (Interfaz de Programación de Aplicaciones): Cuando se desarrolla un programa en ROS, generalmente se utilizan funciones ya existentes las cuales se pueden agregar fácilmente dentro del código que se esté construyendo.

- Soporte de distintos lenguajes de programación: La plataforma ROS posee una biblioteca de clientes que facilita el trabajo de los programadores. Debido a que puede importar lenguajes de programación que son bastantes populares, tales como Python, C++, Java, Ruby, Lips, entre otros.

5.2. El robot HSR

Ha habido un creciente interés en el desarrollo de manipuladores móviles que sean capaces de realizar trabajo físico en entornos domésticos. Debido a esto, Toyota en su línea de robots de apoyo humano desarrolló una plataforma de investigación compacta y segura, *Human Support Robot (HSR)* (Robot de apoyo humano) el cual se muestra en la figura 5.2, este se ha proporcionado a varios institutos de investigación para establecer una comunidad de desarrolladores.

Figura 5.2: Robot HSR

Para lograr que este robot ejecute tareas en un entorno real se requiere un enorme desarrollo de software, además de que el hardware pueda coexistir con las personas en su espacio vital. A causa de esto *HSR* se desarrolló como una plataforma de investigación con un cuerpo compacto que tiene la capacidad y

seguridad de realizar pruebas de campo en entornos domésticos (Yamamoto et al., 2019).

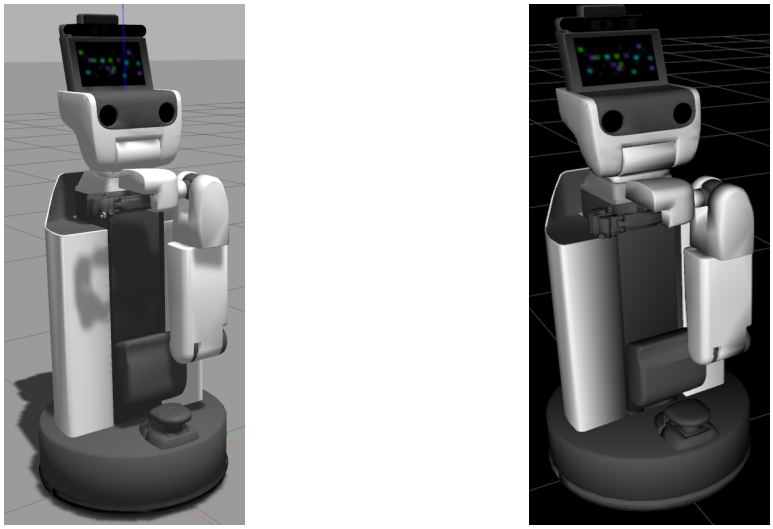
En los últimos años las competiciones internacionales de robots han llamado la atención como un enfoque eficaz para acelerar la investigación y el desarrollo de robots. HSR fue adoptado como la plataforma estándar en la liga *DSPL* de la RoboCup, y la competencia *World Robot Summit (WRS).*

5.3. El simulador Gazebo

La simulación de robots es una herramienta esencial para el desarrollo del campo. Un simulador bien elaborado permite probar rápidamente algoritmos, diseñar robots, realizar pruebas de regresión y entrenar un sistema de inteligencia artificial utilizando escenarios realistas. Gazebo ofrece la capacidad de simular con precisión y eficiencia robots en entornos complejos interiores y exteriores, genera información realista de sensores, así como las interacciones entre los objetos físicamente plausibles. Cuenta con un motor de física robusto, gráficos de alta calidad e interfaces gráficas convenientes, además posee una gran comunidad de desarrolladores (`http://gazebosim.org/`).

Debido a las características de este simulador, Toyota realizó un modelo de su robot *HSR*, el cual cuenta con las herramientas y habilidades del robot real, excluyendo algunos sensores como el de fuerza y el reconocimiento de voz. En esta representación del robot se pueden agregar una gran cantidad de objetos y realizar el diseño de diferentes escenarios.

Gracias a todas estas aplicaciones y capacidades del modelo *HSR* es posible realizar simulaciones que representen de manera muy exacta el comportamiento del robot en un entorno real.



(a) HSR Gazebo (b) HSR Rviz

Figura 5.3: Modelo HSR

5.4. Implementación de los algoritmos comparados

Para comparar el desempeño entre algoritmos basados en muestreo y los basados en búsqueda en grafos, se seleccionaron tres algoritmos: A*, RRT-Ext y RRT-Connect. Se seleccionó A* debido a que se ha observado que es el más eficiente del grupo de los métodos basados en búsqueda en grafos. Este algoritmo es el que se ha utilizado en el robot HSR y otros del Laboratorio de Bio-Robótica y es por esto que se seleccionó para la comparación. Por otro lado, los algoritmos RRT-Ext y RRT-Connect se seleccionaron porque son los que se espera que tengan un mejor desempeño del grupo de métodos basados en muestreo, como se explicó en el capítulo 4.

Para poner en funcionamiento estos algoritmos se hizo uso del software elaborado por el Laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM, el cual cuenta con herramientas que facilitaron su desarrollo

y aplicación, de esta manera se lograron realizar las simulaciones y pruebas correspondientes.

Se elaboró una Interfaz Gráfica de Usuario (GUI) como se muestra en la figura 5.4. Mediante esta herramienta podemos manipular el hardware del robot HSR, el cual abarca brazo, pinza, cabeza, torso y base del robot. También se añadieron funciones que permiten realizar tareas de navegación, planeación de rutas y la obtención de datos.

- Arm and Manipulation: Permite modificar la posición del brazo, junto con el uso de la pinza que este posee. Esto resulta muy útil cuando se requiera que el robot tome objetos.

- Head: Controla la orientación de la cabeza, lo cual es de gran ayuda cuando se desea hacer reconocimiento y detección de objetos, debido a que esta contiene una cámara y otro tipo de sensores que ayudan a estas tareas.

- Torso: Este funciona principalmente de apoyo a los a grupos antes mencionados, debido a que les permite cambiar la altura a la que pueden realizar sus funciones.

- Manual Drive: Manipula la base, lo que modifica la posición actual del robot en el mapa.

- Obtaining Data: Mediante esta función podemos conseguir los datos necesarios de los algoritmos implementados.

- Mobile Base and Navigation: En esta sección se pueden realizar diferentes tareas,como mover el robot a una posición especifica, o trazar una ruta, todo esto usando alguno los métodos de planeación disponibles. También proporciona información de la ubicación actual del robot en el mapa.

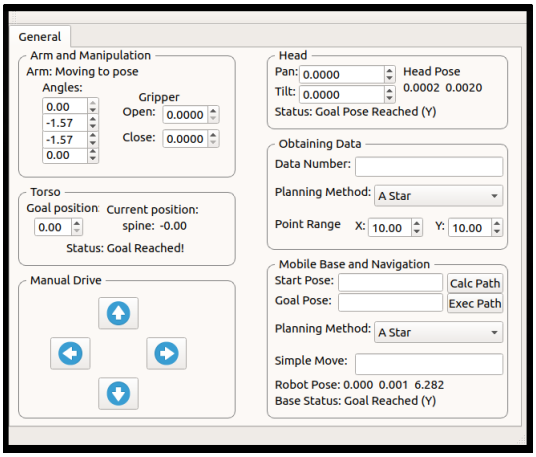
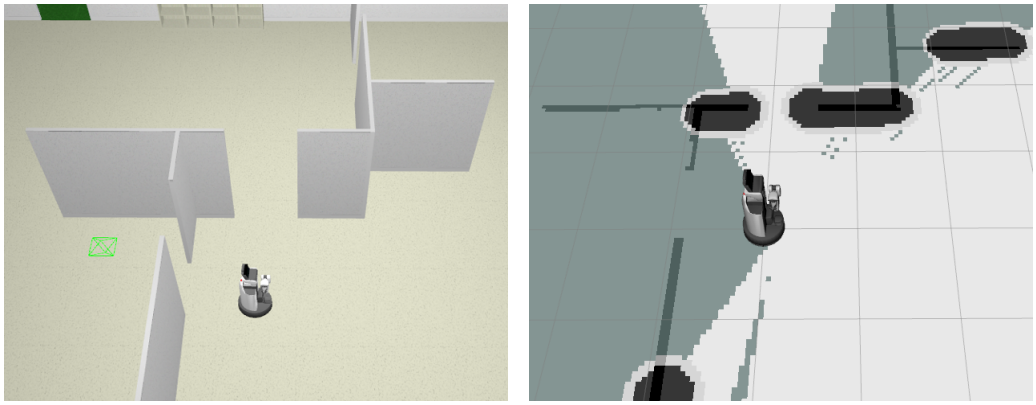


Figura 5.4: GUI HSR

La representación del robot HSR es posible gracias a los paquetes proporcionadas por Toyota, estos permiten tener una interacción entre un entorno creado en Gazebo y un mapa del espacio en Rviz. También proveen las funciones que hacen posible usar las herramientas del robot en simulaciones y el mundo real.



(a) Mundo Gazebo (b) Mapa Rviz

Figura 5.5: Simulaciones

Esto se logra con la implementación de diferentes nodos y servicios, los cuales se comunican con la *GUI* usando algunos datos de entrada o modificando algunos valores establecidos:

- **Arm and Manipulation**
  - **Arm Moving to pose:** Mueve cada una de las cuatro articulaciones del brazo en un rango establecido.
  - **Gripper:** Abre y cierra la pinza del brazo para sujetar objetos.
- **Head**
  - **Pan:** Gira la cabeza del robot de izquierda a derecha, para buscar cosas a su alrededor.
  - **Tili:** Mueve la cabeza del robot de arriba a abajo para detectar objetos cercanos a el.
- **Torso**
  - **Goal position:** Levanta o baja el torso del robot para aumentar el alcance de los grupos antes descritos.
- **Manual Drive**
  - **Botones de dirección** Utiliza las flechas de *izquierda* y *derecha* para modificar la orientación del robot, con el fin de cambiar los puntos de referencia y permitir que el robot se mueva en diferentes direcciones usando las flechas de *adelante* y *atrás*.
- **Mobile Base and Navigation**
  - **Planning Method:** Su objetivo consiste en mostrar el método seleccionado para la planeación de rutas.

- **Start Pose, Calc Path, Goal Pose:** Estos nodos requieren dos parámetros de posición (X, Y), los cuales corresponden a las coordenadas de la ubicación a la que se desea llegar, después se obtienen los valores de la posición actual usando una función que se comunica con el mapa y la base del robot, también se utiliza la información del método de planeación elegido.

  Una vez que se cuenta con esta información, se hace uso de un servicio que se comunica con el nodo que contiene el método seleccionado. El servicio mantendrá un canal de comunicación abierta esperando una respuesta que indique si es posible establecer un camino o no. Estos nodos comienzan evaluando si la posición inicial o final esta ocupada, es decir tiene un obstáculo, de estar libre comienza el proceso de obtener una ruta entre ambos puntos, en caso de encontrar una, esta se mostrará en el mapa del visualizador Rviz.

- **Exec Path:** Es un complemento de los nodos anteriores por lo que también requiere de dos parámetros de ubicación y uno de ángulo θ (el cual es opcional, debido a que este determinara la orientación que tendrá el robot al finalizar el recorrido), después realiza el mismo procedimiento para obtener una ruta. Posteriormente si no hubo ningún problema durante el proceso de planeación comenzará a moverse el robot por el camino trazado en el mapa hasta la posición deseada.

- **Obtaining Data**

  - **Data Number:** Recibe un número que determina la cantidad de datos que se desea generar para el análisis.

  - **Planning Method:** Su objetivo consiste en mostrar el método seleccionado de los tres disponibles para la obtención de datos.

- **Point Range:** Determina el rango de los números aleatorios generados, para ajustarse al mapa que se está utilizando, por lo que se deben conocer previamente sus dimensiones.

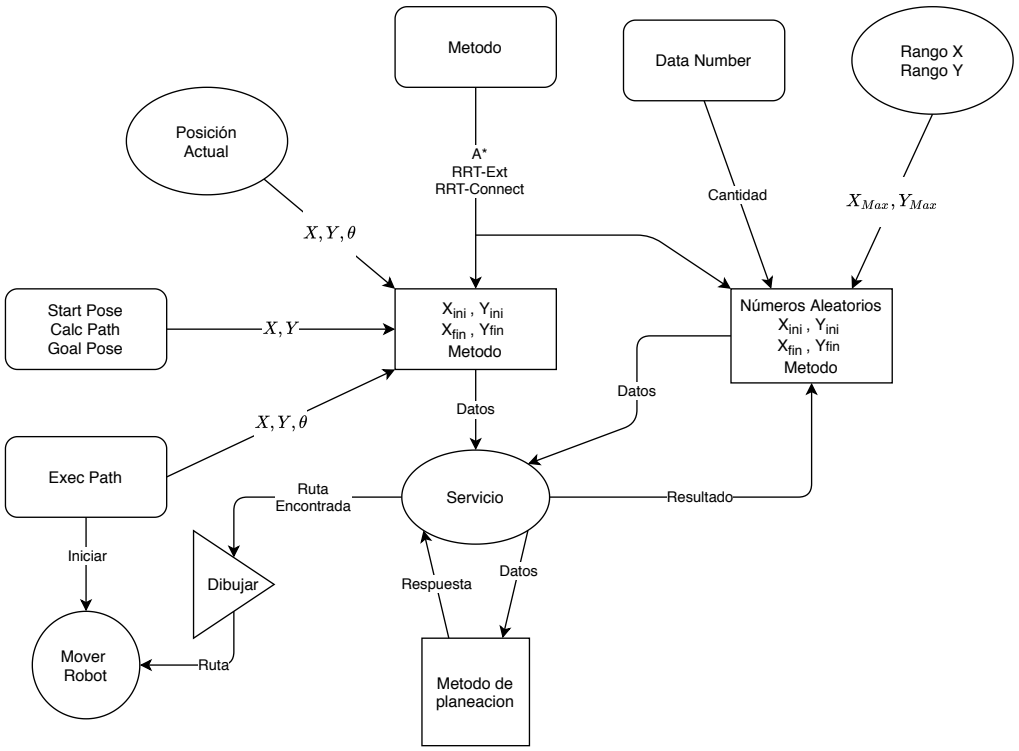


Figura 5.6: Sistemas de navegación y Obtención de datos

El planeador de rutas elegido funciona de manera independiente a los algoritmos y nodos de control, debido a que estos se ejecutan utilizando únicamente información del mapa en conjunto con los datos del origen y destino. Por lo tanto estos algoritmos no dependen del hardware que se esté utilizando, de modo que pueden funcionar en cualquier otro robot y no solo en el HSR. Para lograr esta autonomía se implementa un nodo cuya función es atender el servicio *PathFromMap* que se encuentra en el paquete *navig_msgs*, que esta compuesto por:

- **Datos del mapa:** Estos pertenecen al mensaje tipo (*OccupancyGrid*) del paquete (*nav_msgs*), el cual simboliza una cuadricula 2-D donde cada celda contiene información del espacio representado (`http://docs.ros.org/melodic/api/nav_msgs/html/msg/OccupancyGrid.html`).

- **Posición inicial y final ($X_{ini}, Y_{ini}, X_{fin}, Y_{fin}$):** Forman parte del mensaje (*Pose*) del paquete (*geometry_msgs*), cuya función es es almacenar datos de posición y orientación (`http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Pose.html`).

y estos se denotan dentro del servicio de la siguiente forma:

```
geometry_msgs/Pose start_pose
geometry_msgs/Pose goal_pose
nav_msgs/OccupancyGrid map
---
nav_msgs/Path path
```

de esta manera se puede almacenar la información necesaria para la planeación de rutas.

5.5. Estrategias de comparación

Con el propósito de evaluar el desempeño de los algoritmos en diferentes entornos, se optó por generar usa serie de posiciones aleatorias para medir el tiempo que toma encontrar un camino entre dos de ellas, así como su longitud y tortuosidad, dicho de otra manera se pretende calcular el tiempo de ejecución de los métodos de planeación al momento de obtener una ruta, junto con el tamaño y la cantidad de curvas que esta posee.

Debido a esto se creó uno de los nodos descrito en la sección 5.4, *Obtaining Data*, este permite producir una cantidad determinada de datos generando una serie de puntos aleatorios distribuidos de manera uniforme entre los

limites del mapa seleccionado. Una vez obtenido un punto inicial y final comienza el proceso de planeación, al conseguir un resultado, se procederá con los siguientes valores aleatorios, y de esta manera obtener tener todos los datos solicitados.

Estos se guardan en un archivo individual para cada método, con el formato $'.xls'$ compatible con Excel, el cual contiene las coordenadas del punto inicial y final, junto como el tiempo de ejecución medido en milisegundos, la longitud de la ruta en metros y su tortuosidad en radianes.

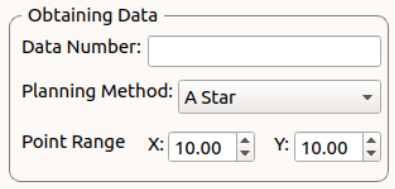


Figura 5.7: Obtención de Datos usando A*

Una vez obtenida la información de los algoritmos se pueden realizar diversas pruebas que determinen qué método es más rápido que otro, para esto se puede hacer uso de las pruebas estadísticas.

Pruebas estadísticas

Una de las aplicaciones de la estadística es extraer $inferencias$ en poblaciones a partir del estudio de muestras. Uno de los aspectos que permite esta deducción es determinar si existe o no asociación entre diferentes variables. Para realizar este proceso se parte de una hipótesis, es decir, de suposiciones cuya validez se debe confirmar o rechazar. Y para llevar a cabo esta comprobación se aplican pruebas estadísticas, las cuales son de significación estadística, es decir, cuantifican hasta qué punto la variabilidad de la muestra puede ser responsable de los resultados de un estudio en particular.

La H_n **(hipótesis nula)** representa la afirmación de que no existe asociación entre las variables estudiadas y la H_a**(hipótesis alternativa)** afirma

que hay algún grado de relación o asociación entre las dos variables. Dicha decisión puede ser afirmada con seguridad a través del nivel de significación.

El proceso de aceptación o rechazo de la hipótesis lleva implícito un riesgo que se cuantifica con el valor de $'p'$, que es la probabilidad de aceptar la hipótesis alternativa como cierta, cuando la correcta podría ser la hipótesis nula. El valor de $'p'$ indica si la asociación es estadísticamente significativa. Cuando rechazamos la H_n y aceptamos la H_a hay una asociación ($p < 0.05$), asimismo, si ($p > 0.05$) aceptamos la H_n .

5.5.1. Prueba t-student

Supongamos que tenemos dos muestras aleatorias con medias $X1$, $X2$ y queremos saber si estas dos son significativamente distintas a un nivel de ($p < 0.05$), es decir, que si afirmamos que hay una diferencia entre las muestras tenemos un 95 % de probabilidad de tener razón. Entonces al realizar esta prueba podemos verificar la valides de una premisa. Existen algunas variantes en la forma de realizar la prueba, dependiendo de las características de la muestra. Para este trabajo se utilizó la prueba suponiendo varianzas diferentes y utilizando la distribución de dos colas.

5.5.2. Tiempo de ejecución

Haciendo uso de esta prueba podemos postular la hipótesis de que un método es más rápido que otro si ($p < 0.05$), del mismo modo si ($p > 0.05$) determinamos que no existe diferencia entre ellos. Con el propósito de evaluar esta premisa, se busca encontrar diferencias significativas entre los tiempos de ejecución de cada algoritmo, para ello generamos una cierta cantidad de datos usando el procedimiento *Obtaining Data*, descrito anteriormente, después evaluamos estos resultados, a fin de indicar si un método es similar o mejor que otro.

5.5.3. Longitud

Posteriormente se calculara el tamaño de la ruta, midiendo la distancia *euclidiana* que existe entre cada uno de los puntos que la conforman, haciendo uso del siguiente método:

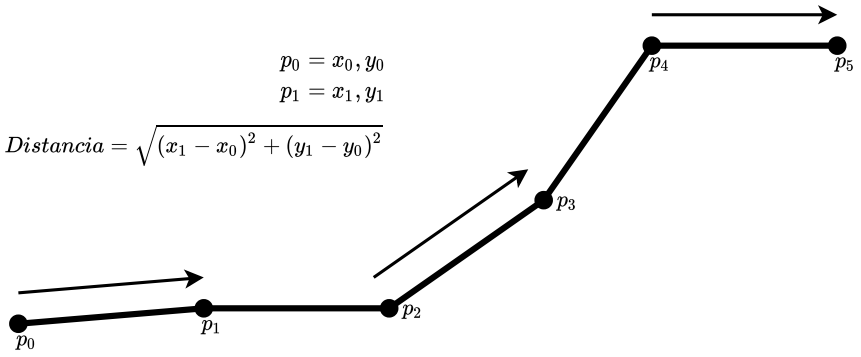


Figura 5.8: Calculando la distancia Euclidiana

teniendo dos puntos compuestos por coordenadas (x, y), se puede obtener la extensión del espacio entre ellos, utilizando la formula de *Distancia* mostrada en la Figura 5.8. Para obtener la longitud final de la ruta se usa la siguiente operación:

$$Longitud = \sum_{i=0}^{n-1} x_i = x_i + Distancia$$

5.5.4. Tortuosidad

Por ultimo se analizará la complejidad de las rutas, evaluando sus giros y curvas durante el trayecto generado. Con el propósito de lograr esto se utilizó el procedimiento ilustrado a continuación:

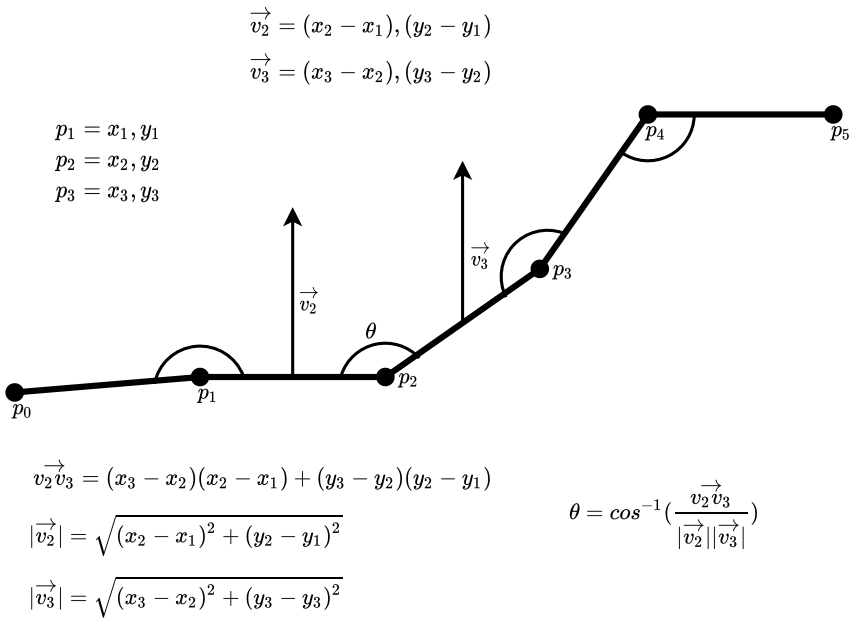


Figura 5.9: Cálculo de los Ángulos

para determinar los ángulos que componen a la ruta encontrada, debemos analizar grupos de tres puntos, para ello comenzamos planteando los vectores directores, es decir, $\vec{v_2} = P_2 - P_1$ y $\vec{v_3} = P_3 - P_2$, una vez obtenidos debemos realizar el producto punto entre ellos $(\vec{v_2} * \vec{v_3})$, posteriormente dividimos el resultado entre la multiplicación de los módulos de los vectores $(|\vec{v_2}| * |\vec{v_3}|)$, por ultimo aplicaremos el *coseno inverso* (cos^{-1}) al resultado final, encontrando así el ángulo que existe entre esos putos. Para concluir obtendremos el resultado final de tortuosidad usando la siguiente operación:

$$Tortuosity = \sum_{i=0}^{n-2} x_i = x_i + \theta$$

En el siguiente capítulo se describen los resultados obtenidos con diferentes mapas y algoritmos para determinar si un método es mejor que otro, evaluando si es significativamente más rápido, genera rutas significativamente más cortas o con menor tortuosidad.

Capítulo 6

Resultados

En este capitulo se muestran los resultados de los algoritmos implementados, obteniendo la media (*Mean*) y la desviación estándar (*SD*), así como los datos obtenidos de la prueba **t-student**, utilizados para comparar y discutir qué métodos tuvieron mejor rendimiento, usando cada uno de los diferentes mapas mostrados en las siguientes secciones del documento.

6.1. Descripción del experimento

Con el fin obtener los datos necesarios para la evaluación de los algoritmos implementados, se utilizo el simulador *Gazebo* y el visualizador *Rviz*, empleando cuatro mapas de diferentes tamaños y complejidades. Los cuales fueron proporcionados por el software de Toyota, estos corresponden a oficinas, laboratorios y apartamentos.

Se corrió el programa *Obtaining Data* descrito en la sección 5.4, el cual tiene como objetivo automatizar la obtención de información, generando 200 puntos aleatorios, después se mide el tiempo de ejecución que toma planear una ruta entre dos de ellos, mediante el siguiente proceso:

```
struct timeval t_ini, t_fin;
gettimeofday(&t_ini, NULL);
```

```
/*
    Algoritmo en ejecución
*/
gettimeofday(&t_fin, NULL);
---
sec = (t_fin.tv_sec - t_ini.tv_sec)*1000;
usec = (t_fin.tv_usec - t_ini.tv_usec)/1000;
executionTime = sec + usec;
```

Esta información se calcula y guarda junto con la longitud y tortuosidad de la ruta, sólo si durante la ejecución del algoritmo se obtuvo un camino entre los puntos proporcionados por el programa anterior.

6.2. Pruebas de desempeño

En la Tabla 6.1 se muestra la media (*Mean*) y la desviación estándar (*SD*) de los 100 tiempos de ejecución de cada uno de los métodos planteados en este trabajo medidos en milisegundos, haciendo uso de los mapas mostrados en las Figuras 6.4, 6.5, 6.7 y 6.6.

Posteriormente en la Tabla 6.2 podemos ver los resultados de la prueba **t-student** realizada sobre los datos de la tabla anterior. Donde **R-C** significa *RRT-Connect*, **R-E** es *RRT-Ext* y por ultimo **ND**, no hay diferencia.

Debido a esto si el resultado es ($p < 0.05$) la hipótesis de que el algoritmo con la menor media es mas rápido es aceptada y se toma como el mejor método, en caso contrario, es decir,($p > 0.05$) ninguno de los dos métodos es significativamente mas rápido que otro.

Como se puede observar en la Tabla 6.2, los métodos basados en muestro (*RRT-Ext y RRT-Connect*) fueron mas rápidos que el método basado en grafos ($A*$) en la mayoría de los mapas, siendo el *RRT-Connect* el mas veloz de todos.

Tiempos de ejecución [milisegundos]						
Mapa	A*		RRT-Ext		RRT-Connect	
	Mean	SD	Mean	SD	Mean	SD
1	710.25	250.82	1669	4603.75	332.54	1108.47
2	4498.39	1816.42	800.1	1911.19	748.55	2979.39
3	1215	659.85	2970.53	4466.03	532.37	1440.21
4	5108	2321	6610.32	11206.17	315.75	419.61

Tabla 6.1: Comparación de Medias y Desviación Estándar

Tiempos de ejecución						
Mapa	A* vs RRT-Ext		A* vs RRT-Connect		RRT-Connect vs RRT-Ext	
	Mejor	$Valor_p$	Mejor	$Valor_p$	Mejor	$Valor_p$
1	ND	0.0818	R-C	0.0005	R-C	0.0114
2	R-E	1.96×10^{-30}	R-C	1.72×10^{-21}	N-D	0.5028
3	A*	0.0004	R-C	2.05×10^{-5}	R-C	2.39×10^{-6}
4	N-D	0.240	R-C	1.41×10^{-37}	R-C	4.75×10^{-7}

Tabla 6.2: Comparación de tiempos de ejecución usando la prueba t-student

En las siguientes tablas podemos ver el análisis de la longitud de las rutas generadas, empezando por la tabla 6.3, mostrando la media (*Mean*) y la desviación estándar (*SD*) de todos los métodos.

Después en la tabla 6.4 están los resultados de la prueba **t-student** aplicada sobre los datos anteriores, haciendo uso de la misma notación de la tabla 6.2.

Se puede observar que ningún método es significativamente mejor produciendo rutas con menor longitud. Debido a que las diferencias en el tamaño son mínimas lo cual se puede ver en los datos de la Tabla 6.3, donde sin importar el mapa los resultados son similares entre si. Esto era un resultado esperado, pues todos los métodos de planeación de rutas tratan de encontrar la ruta más corta entre dos puntos.

Longitud [metros]						
Mapa	A*		RRT-Ext		RRT-Connect	
	Mean	SD	Mean	SD	Mean	SD
1	13.09	6.69	12.34	6.83	13.96	7.63
2	25.26	12.8	23.6	13.38	24.96	14.26
3	27.18	16.67	26.43	16.93	30.56	17.79
4	31.45	14.81	30.79	18.44	34.37	19.49

Tabla 6.3: Comparación de Medias y Desviación Estándar

Longitud						
Mapa	A* vs RRT-Ext		A* vs RRT-Connect		RRT-Connect vs RRT-Ext	
	Mejor	$Valor_p$	Mejor	$Valor_p$	Mejor	$Valor_p$
1	ND	0.5470	ND	0.3939	ND	0.1602
2	ND	0.3690	ND	0.7433	ND	0.2418
3	ND	0.3212	ND	0.3166	ND	0.0515
4	ND	0.6794	ND	0.1534	ND	0.0960

Tabla 6.4: Comparación de la longitud usando la prueba t-student

En las ultimas tablas se presenta el análisis de la tortuosidad de las rutas, es decir, las curvas y pendientes que esta posee.

En la tabla 6.5, se muestra al igual que antes, la media (*Mean*) y la desviación estándar (*SD*) de todos los métodos.

Por ultimo en la tabla 6.6 se presentan los resultados de la prueba **t-student** implementada sobre los datos de la tabla anterior, y de igual forma se usa la notación de las tablas 6.6 y 6.4.

En este caso se puede notar que el método basado en grafos ($A*$), produce rutas con menor tortuosidad que los métodos basados en muestro (*RRT-Ext y RRT-Connect*), al mismo tiempo estos últimos no presentan diferencias significativas entre si.

Tortuosidad [radianes]						
Mapa	A*		RRT-Ext		RRT-Connect	
	Mean	SD	Mean	SD	Mean	SD
1	8.79	4.66	17.63	12.24	18.43	12.41
2	9.74	4.99	25	19.11	27.3	18.07
3	9.07	4.36	23.74	15.34	22.66	14
4	10.11	3.76	26.1	17.3	22.15	14.04

Tabla 6.5: Comparación de Medias y Desviación Estándar

Tortuosidad						
Mapa	A* vs RRT-Ext		A* vs RRT-Connect		RRT-Connect vs RRT-Ext	
	Mejor	$Valor_p$	Mejor	$Valor_p$	Mejor	$Valor_p$
1	A*	3.04×10^{-10}	A*	5.74×10^{-11}	ND	0.7532
2	A*	4.12×10^{-13}	A*	6.62×10^{-16}	ND	0.5873
3	A*	1.44×10^{-14}	A*	1.26×10^{-15}	ND	0.8012
4	A*	2.57×10^{-14}	A*	9.73×10^{-15}	ND	0.2446

Tabla 6.6: Comparación de tortuosidad usando la prueba t-student

En las siguientes Figuras 6.1, 6.2 y 6.3 se puede apreciar que en algunas ocasiones los algoritmos *RRT-Ext y RRT-Connect* producen rutas ligeramente más cortas que $A*$, pero al mismo tiempo son mas tortuosas, dicho de otra manera, pueden producir rutas mas pequeñas pero con mas pendientes y giros.

Sin embargo los resultados de las pruebas mostraron que la longitud de cada una no presenta diferencias significativamente grandes, y por el contrario la tortuosidad de los métodos *RRT-Ext y RRT-Connect* sí es significativamente mayor a la de $A*$.

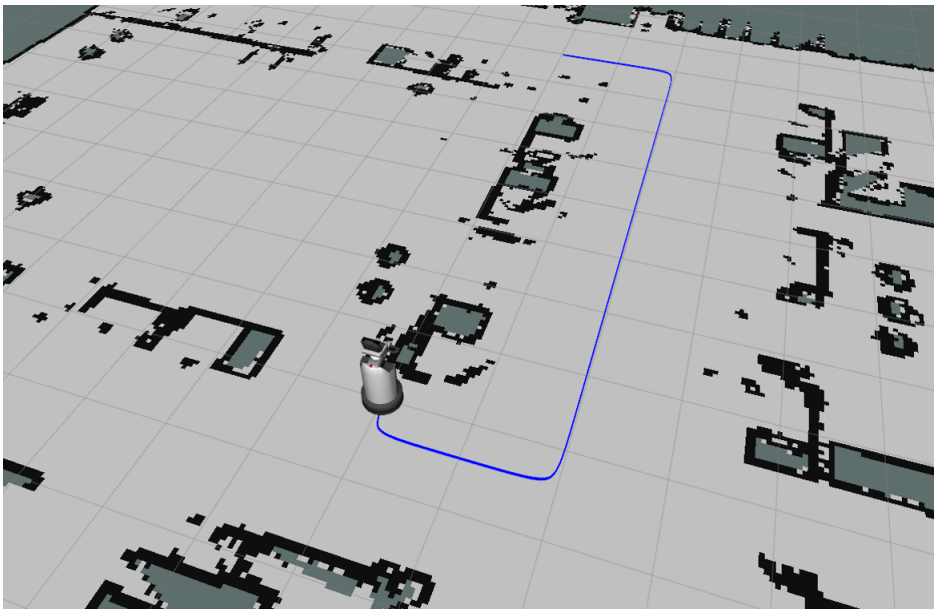


Figura 6.1: Ruta con A*

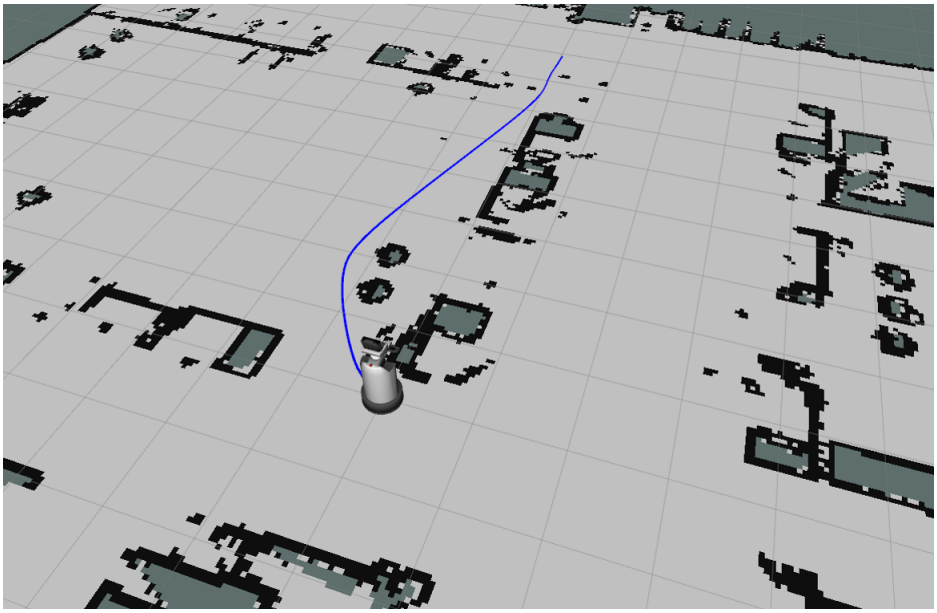


Figura 6.2: Ruta con RRT-Ext

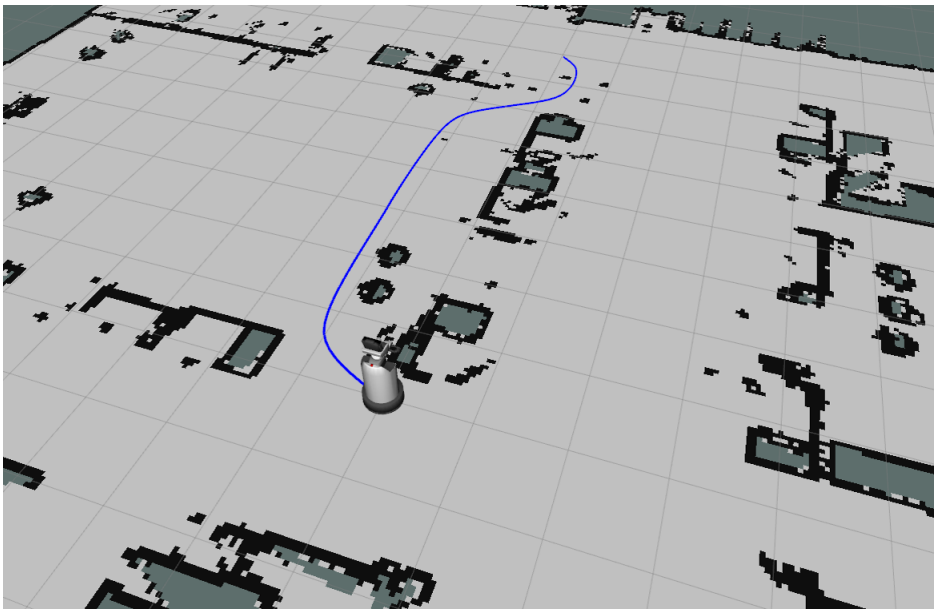


Figura 6.3: Ruta con RRT-Connect

6.3. Mapas usados

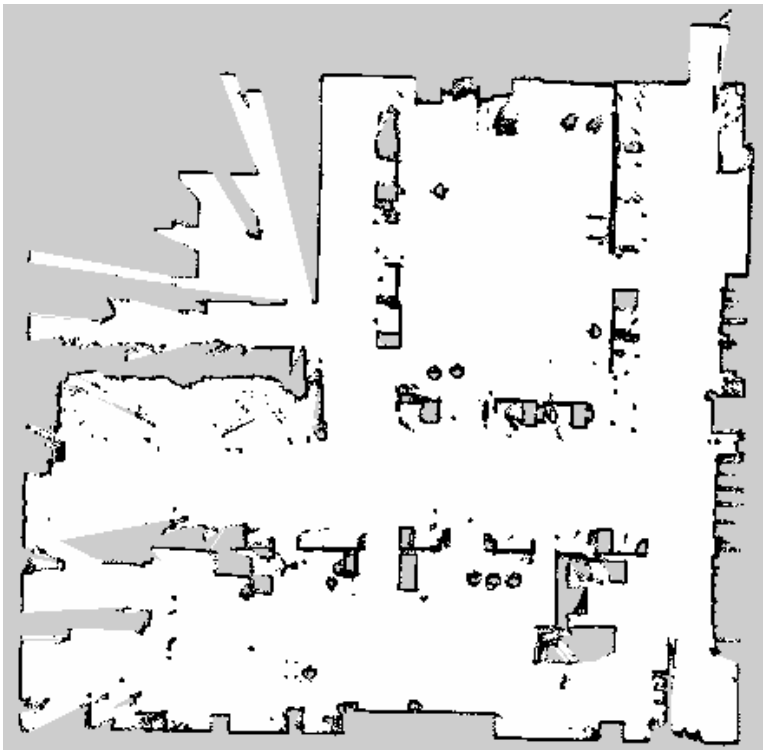


Figura 6.4: Mapa 1 - Departamento

Figura 6.5: Mapa 2 - Laboratorio

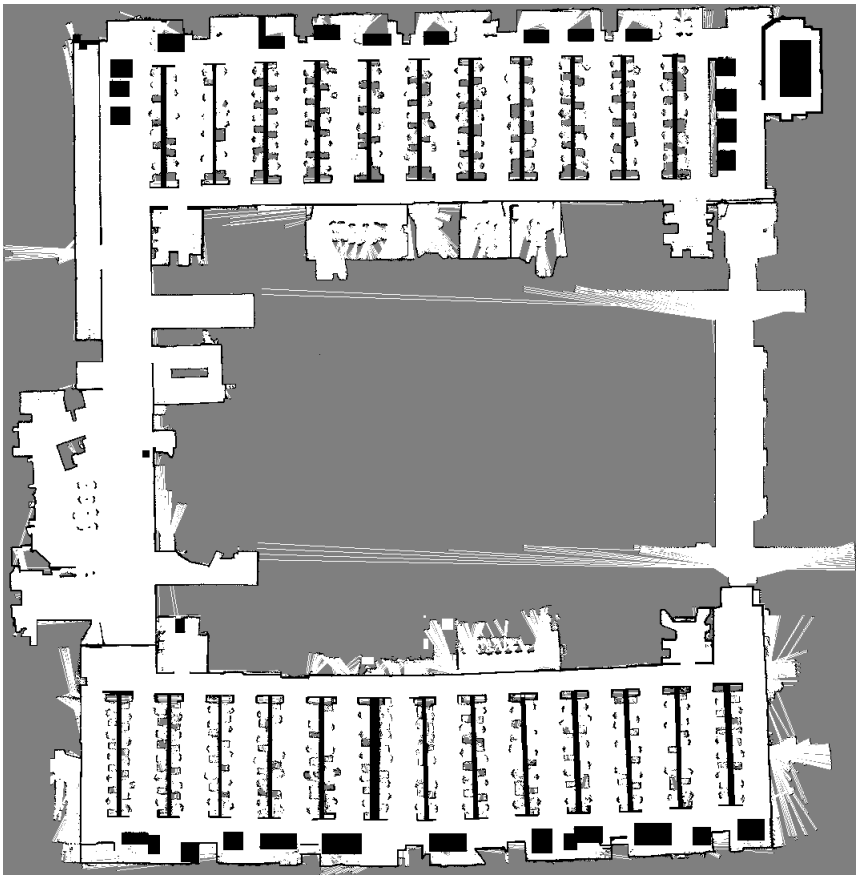


Figura 6.6: Mapa 3 - Oficina

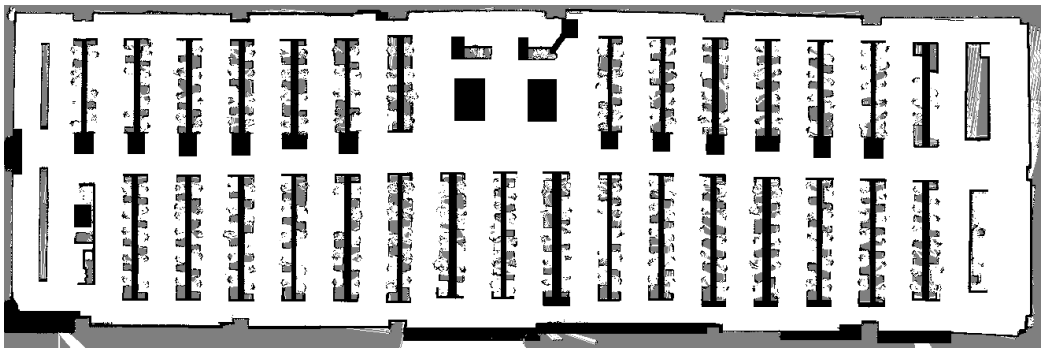


Figura 6.7: Mapa 4 - Oficina

Integración del robot HSR

Todos los procedimientos descritos en los capítulos anteriores, fueron desarrollados para ser transparentes al hardware, es decir, pueden ser implementados en cualquier robot que cumpla con los estándares necesarios para usar este sistema de navegación autónomo.

Sin embargo estos fueron implementados con las herramientas de Toyota, lo que proporciona un fácil acoplamiento con el robot HSR, permitiendo que las funciones de la simulación en *Gazebo* y *Rviz* puedan ser usadas sin mayor problema que establecer un enlace con el robot, otorgándole la capacidad de moverse usando el sistema de navegación e implementar los planeadores de rutas desarrollados en este trabajo. También permite la manipulación de hardware descrita en el Capitulo 5.4, debido a esto se pueden añadir funciones adicionales, como la manipulación y detección de objetos.

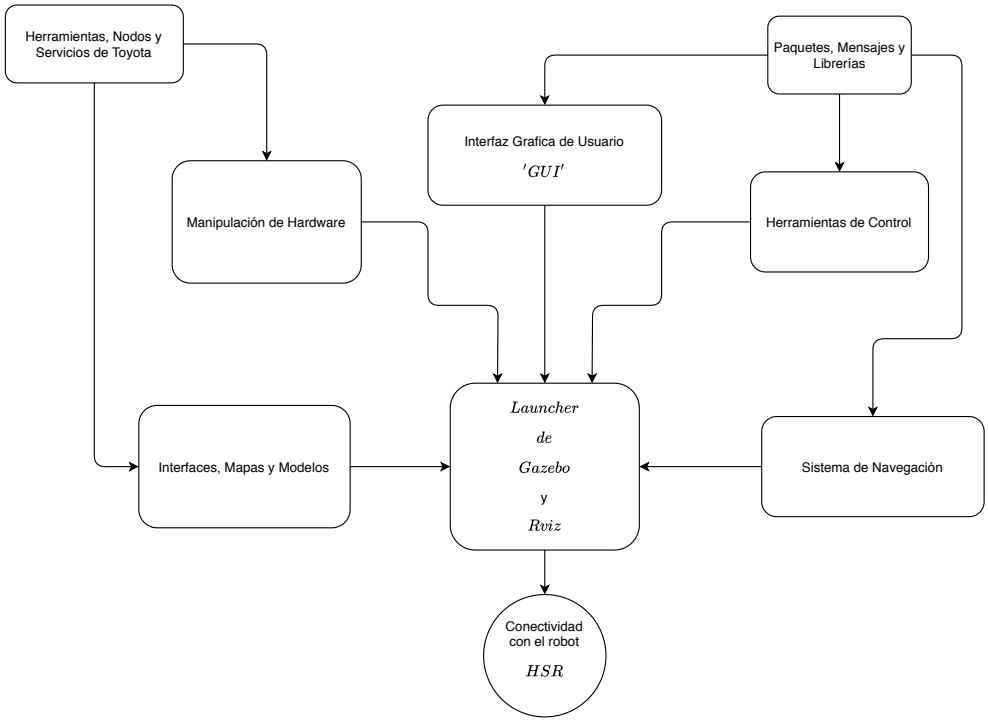


Figura 6.8: Integración de Funciones

Capítulo 7

Discusión

7.1. Conclusiones

Se desarrolló un sistema de planeación de rutas capaz de integrarse con el software elaborado por el Laboratorio de Bio-Robotica de la Facultad de Ingeniería de la UNAM. Con este sistema se implementaron los algoritmos descritos en este trabajo y se pudo evaluar el desempeño del robot en conjunto con el resto de subsistemas ya implementados: localización, evasión de obstáculos, planeación y seguimiento de rutas.

El sistema se puso en funcionamiento en una simulación del robot HSR utilizando las herramientas provistas por el meta-sistema ROS y las funciones de $Toyota$.

Se realizaron pruebas para evaluar el desempeño de cada uno de los algoritmos propuestos en este trabajo usando diferentes mapas. Como métricas de comparación se emplearon el cálculo del tiempo de ejecución medido en *milisegundos*, la longitud en *metros* y tortuosidad en *radianes*, como se describe el sección 5.5.

Gracias a todo esto se pudieron comparar los diferentes algoritmos desarrollados, usando los distintos escenarios proporcionados por el software de $Toyota$ mostrados en la sección 6.3. Debido a esto se logró determinar el

desempeño de los algoritmos usando ambientes con diferentes características y complejidades, los cuales van desde un departamento, hasta un laboratorio y un par de oficinas.

La información resultante mostró que existen diferencias significativas en el tiempo de ejecución del los métodos basados en muestreo (*RRT-Ext y RRT-Connect*) y el método basado en grafos ($A*$), a causa de esto se determinó que los métodos basados en muestro pueden encontrar rutas mas rápidamente en ambientes como los mostrados en este trabajo, los cuales tienen muchos obstáculos. Sin embargo otro de los parámetros de comparación indicó que también producen rutas más tortuosas que que los métodos basados en grafos. Mientras que los resultados conseguidos en la comparación de la longitud, mostraron que el tamaño de las rutas generadas no presenta diferencias significativas en ninguno de los mapas utilizados.

Con todos los resultados obtenidos se puede concluir que los métodos basados en muestreo (*RRT-Ext y RRT-Connect*) son significativamente mas rápidos y tortuosos que los métodos basados en grafos ($A*$). Sin embargo el uso de estos métodos puede afectar a robots que posean algún tipo de restricción de movimiento, por lo cual se deben tomar en cuenta ciertas consideraciones al momento de usar este sistema de navegación autónomo.

7.2. Trabajo futuro

Como trabajo futuro se planea realizar una comparación mas extensa de algoritmos, usando un mayor numero de métodos para la planeación de movimientos y rutas, de igual forma se tendrán que utilizar mas ambientes y escenarios, con diferentes complejidades y dimensiones para probar su desempeño.

El uso de mas parámetros de comparación también es una parte esencial, debido a que de esta manera podemos obtener resultados que determinen mejor el comportamiento y características de cada método.

Se deben realizar pruebas experimentales en robots reales, debido a que un entorno real puede presentar condiciones que afectan el uso de los algoritmos implementados y estas deben ser tomadas en cuenta. Al mismo tiempo se espera que se realice una correcta integración con otras funciones del robot, como la manipulación y detección de objetos.

Por ultimo, se espera poder utilizar algunas funciones de este sistema de planeación de rutas para robots de servicio domestico en otras áreas, como la planeación de movimientos para manipulación de objetos o en la navegación de vehículos sin conductor.

Referencias

Aracil, R., Balaguer, C., and Armada, M. (2008). Robots de servicio. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 5(2):6–13.

Azorín, Francisco y Sánchez-Crespo, J. L. (1994). *Métodos y aplicaciones del muestreo*. Alianza Madrid.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433.

Bar-Shalom, Y., Li, X. R., and Kirubarajan, T. (2004). *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons.

Casal, Jordi y Mateu, E. (2003). Tipos de muestreo. *Rev. Epidem. Med. Prev*, 1(1):3–7.

Choset, H. M., Hutchinson, S., Lynch, K. M., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Cuchango, Helbert Eduardo Espitia y Esmeral, J. I. S. (2012). Algoritmo para planear trayectorias de robots móviles, empleando campos potenciales y enjambres de partículas activas brownianas. *Ciencia e Ingeniería Neogranadina*, 22(2):75–96.

De Graaf, M. M. and Allouch, S. B. (2016). Anticipating our future robot society: The evaluation of future robot applications from a user's perspective. In *2016 25th IEEE international symposium on robot and human interactive communication (RO-MAN)*, pages 755–762. IEEE.

De la Rosa, E. (2004). *Heurística para la generación de configuraciones en pasajes estrechos aplicada al problema de los clavos. Capítulo I. Planificación de movimientos*. PhD thesis, Tesis Maestría en Ciencias con Especialidad en Ingeniería en Sistemas ....

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110.

Edelkamp, S. and Schroedl, S. (2011). *Heuristic search: theory and applications*. Elsevier.

Gil, A., Reinoso, O., Payá, L., and Ballesta, M. (2008). Influencia de los parámetros de un filtro de partículas en la solución al problema de slam. *IEEE LATIN AMERICA TRANSACTIONS*, 6(1).

Holland, J. M. (2004). *Designing autonomous mobile robots: inside the mind of an intelligent machine*. Elsevier.

Holz, D., Iocchi, L., and Van Der Zant, T. (2013). Benchmarking intelligent service robots through scientific competitions: The robocup@ home approach. In *2013 AAAI Spring Symposium Series*. Citeseer.

Huang, G. (2019). Visual-inertial navigation: A concise review. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9572–9582. IEEE.

Hüttenrauch, H. and Severinson-Eklundh, K. (2006). To help or not to help a service robot: Bystander intervention as a resource in human–robot collaboration. *Interaction Studies*, 7(3):455–477.

IFR (2018). https://ifr.org/service-robots. *Service Robots*.

ISO (2012). *Robots and robotic devices – vocabulary(ISO n 8373-2012)*. Standard, Geneva, Standard: International Organization for Standardization.

Khatib, O. (1986). The potential field approach and operational space formulation in robot control. In *Adaptive and Learning Systems*, pages 367–377. Springer.

Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE.

Latombe, J.-C., Lazanas, A., and Shekhar, S. (1991). Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1):1–47.

LaValle, S. M. (1998). *Rapidly-exploring random trees: A new tool for path planning*. Citeseer.

LaValle, S. M., Kuffner, J. J., Donald, B., et al. (2001). Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 1(5):293–308.

Lima, P. U., Nardi, D., Iocchi, L., Kraetzschmar, G., and Matteucci, M. (2013). Rockin@ home: Benchmarking domestic robots through competitions. In *International Conference on Advanced Robotics, ICAR, Montevideo, Uruguay*.

López, D., Gómez-Bravo, F., Cuesta, F., and Ollero, A. (2006). Planificación de trayectorias con el algoritmo rrt. aplicación a robots no holónomos. *Revista Iberoamericana de Automática e Informática Industrial*, 3(3):56–67.

López García, D. A. et al. (2011). *Nuevas aportaciones en algoritmos de*

planificación para la ejecución de maniobras en robots autónomos no holónomos. Universidad de Huelva.

Martorell Pons, L. (2017). Implementació i comparació d'algoritmes per la obtenció del camí òptim. aplicació en sistemes de weather routing. B.S. thesis, Universitat Politècnica de Catalunya.

Murphy, R. R. (2019). *Introduction to AI robotics*. MIT press.

Narváez, V., Yandún, F., Pozo, D., Morales, L., Rosero, J., Rosales, A., and Auat, F. (2014). Diseño e implementación de un sistema de localización y mapeo simultáneos (slam) para la plataforma robótica robotino®. *Revista Politécnica*, 33(1).

Park, K.-H., Lee, H.-E., Kim, Y., and Bien, Z. Z. (2008). A steward robot for human-friendly human-machine interaction in a smart house environment. *IEEE Transactions on Automation Science and Engineering*, 5(1):21–25.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, page 5. Kobe, Japan.

Reif, J. H. (1979). Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427. IEEE.

Rubio, F., Valero, F., and Llopis-Albert, C. (2019). A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596.

Sieira, A. G. and Molina, M. M. (2011). Planificación de movimientos en robótica móvil utilizando retículas de estados.

Thulasiraman, K. and Swamy, M. N. (2011). *Graphs: theory and algorithms*. John Wiley & Sons.

Torrubia, G. and Terrazas, V. (2012). Algoritmo de dijkstra. un tutorial interactivo. *VII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2001)*.

Yamamoto, T., Terada, K., Ochiai, A., Saito, F., Asahara, Y., and Murase, K. (2019). Development of human support robot as the research platform of a domestic mobile manipulator. *ROBOMECH journal*, 6(1):4.

Yoonseok Pyo, Hancheol Cho, L. J.-D. L. (2017). *ROS Robot Programming*. ROBOTIS.