View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Universidad de Zaragoza

Trabajo Fin de Grado

Implementación de algoritmos de planificación de trayectorias para robots móviles en entornos complejos

Implementation of path planning algorithms for mobile robots in complex environments

Autor

Javier Muñoz Mendi

Director

Eduardo Montijano Muñoz

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza

2019

Resumen

La planificación de trayectorias en robótica es un problema que ha recibido especial atención en los últimos años, debido a que los robots comienzan a estar muy presentes en la industria y en los hogares. Aunque estos robots pueden ser muy diferentes unos de otros, el problema de obtener trayectorias de un punto a otro del espacio evitando obstáculos es similar en todos ellos, ya sea un robot aspirador doméstico, como una *Roomba*, o un robot de rescate en entornos peligrosos.

Con la finalidad de aportar soluciones a este problema, este trabajo fin de grado tiene como objetivo principal la implementación y el estudio de diferentes algoritmos de planificación para obtener trayectorias válidas. Para ello es necesario disponer de (i) un modelo del entorno sobre el cual realizar la tarea de planificación, (ii) un origen y (iii) un destino.

En el TFG se comparan tres algoritmos con diferentes características y comportamientos. Por un lado, se ha estudiado el algoritmo A*, basado en búsqueda en grafos y muy utilizado en la actualidad. Por otro lado, como alternativas con menor coste de computación que el A*, se han estudiado el algoritmo basado en muestreo RRT, y una modificación *anytime-optimal* del mismo, el RRT*.

Con la finalidad de evaluar los diferentes algoritmos de planificación propuestos, se realizan ensayos en entornos de diferente tamaño y complejidad y se comparan los resultados obtenidos. Estos ensayos nos permiten observar las ventajas y desventajas de los algoritmos estudiados, pudiendo elegir el algoritmo idóneo para cada situación.

Por último, en el TFG se realiza una implementación en un robot real mediante la plataforma ROS.

Índice general

Índice de figuras 1

1. Introducción 2

1.1. Motivación y Contexto 2

1.2. Objetivos 3

1.3. Alcance 4

1.4. Estructura de la memoria 4

2. Algoritmos de planificación de trayectorias 6

2.1. Algoritmo A* 6

2.2. Algoritmo RRT 8

2.3. Algoritmo RRT* 10

3. Resultados y discusión 12

3.1. Diseño del experimento 12

3.2. Discusión 14

4. Implementación en robot real 17

4.1. Introducción a ROS 17

4.2. Implementación en ROS del algoritmo RRT* 18

4.3. Resultados 19

5. Conclusiones 21

5.1. Líneas de trabajo futuras 22

Bibliografía **23**

Índice de figuras

1.1. PRM 3

2.1. Proyecto Shakey 7

2.2. Discretización en celdas de una línea recta 9

2.3. Variación del radio del área de recableado con el número de nodos 11

3.1. Mapas utilizados para el experimento 12

3.2. Trayectorias para diferentes algoritmos en el mapa mediano 13

3.3. Resultados del experimento 14

3.4. Resultados del experimento para los mapas pequeño y mediano 15

3.5. Resultados de la comparación de ambas versiones del RRT* 16

4.1. Robots modelo Turtlebot 2 17

4.2. Estructura de ROS 18

4.3. Implementación en ROS 19

4.4. Simulaciones en RViz 20

Capítulo 1

Introducción

1.1. Motivación y Contexto

La planificación de trayectorias en robótica es un problema que ha recibido especial atención en los últimos años, debido a que los robots comienzan a estar muy presentes en la industria y en los hogares. Aunque estos robots pueden ser muy diferentes unos de otros, el problema de obtener trayectorias de un punto a otro del espacio evitando obstáculos es similar en todos ellos, ya sea un robot aspirador doméstico, como una *Roomba*, o un robot de rescate en entornos peligrosos.

Dado un robot con una descripción del entorno, un estado inicial y un estado final, el problema de la planificación de trayectorias se puede definir como el problema de encontrar una trayectoria que lleve al robot del estado inicial al estado final obedeciendo las normas del entorno, *e.g.,* no colisionar con obstáculos. Se dice que un algoritmo creado para llevar a cabo esta planificación está completo si termina en un tiempo finito, devolviendo una solución si esta existe, o no devolviendo nada en caso contrario.

Los planificadores prácticos aparecieron con el desarrollo de métodos de descomposición en celdas[1] [2], y campos de potencial. Estos métodos dependen de una representación explícita de los obstáculos en la configuración espacial, que se usa directamente para construir una solución. Esto puede dar como resultado un exceso de carga computacional en un número alto de dimensiones y entornos con un gran número de obstáculos. Evitar este tipo de representación es la principal motivación que llevó al desarrollo de algoritmos de muestreo (*sampling-based*) [3].

En lugar de utilizar una representación explícita del entorno, los algoritmos de muestreo dependen de una función de detección de obstáculos que proporciona información sobre la viabilidad de posibles trayectorias. Basándose en esta información, conectan un número de puntos muestreados sobre el espacio libre de obstáculos con el fin de construir

un grafo de trayectorias viables. Este grafo se usa más adelante para construir la solución al problema original de planificación de trayectorias.

Los algoritmos de muestreo proporcionan un gran ahorro computacional al evitar una construcción explícita de los obstáculos en el espacio. Estos algoritmos garantizan que las probabilidades de que el planificador no devuelva una solución, si existe, disminuyan hasta cero a medida que el número de muestras se aproxime al infinito. Un ejemplo de algoritmo de muestreo es el PRM (Figura 1.1).

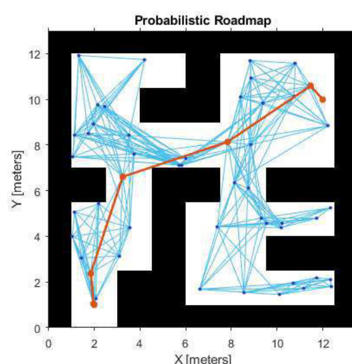


Figura 1.1: PRM

Sin embargo, la solución obtenida por los algoritmos basados en muestreo suele estar lejos de la solución óptima. En muchas situaciones la optimalidad de la trayectoria obtenida es de vital importancia, por lo que una vez alcanzada una solución factible, se dedica tiempo de computación adicional a mejorar la solución conseguida.

Esta idea de afinar la solución progresivamente se conoce en la literatura como algoritmos *anytime-optimal*. Estos algoritmos devuelven una solución que es más próxima a la óptima de forma proporcional al tiempo de ejecución. El algoritmo RRT* es una variación del algoritmo RRT que construye un árbol de manera incremental optimizando los costes de cada rama del árbol a medida que se va construyendo, proporcionando una solución que va convergiendo a la solución óptima con un coste computacional bajo.

1.2. Objetivos

El objetivo principal de este proyecto es estudiar e implementar diferentes algoritmos de planificación de trayectorias para robots móviles en entornos con obstáculos. Por un lado, el trabajo tiene una parte de estudio teórico en la que se analizan los algoritmos más utilizados en la literatura (A*, RRT, RRT*). Por otro lado, en base a un análisis empírico, se plantea la validación del algoritmo RRT* en una plataforma robótica real. Para ello se proponen los siguientes objetivos:

- Estudio de soluciones al problema de planificación de trayectorias.
- Selección de algoritmos de planificación a implementar.
- Estudio de los algoritmos seleccionados.
- Implementación y simulación de los algoritmos de planificación de trayectorias seleccionados en Matlab.
- Implementación en robot real.

1.3. Alcance

El proyecto ha consistido en el estudio teórico y experimental de los tres algoritmos descritos en la sección anterior: A*, RRT y RRT*.

En primer lugar se realizado un estudio teórico de cada uno de los algoritmos seleccionados. Los algoritmos RRT y RRT* se han estudiado directamente de la publicación original que los propuso [3] [4]. El algoritmo A* se ha estudiado de un proyecto fin de carrera [5].

En segundo lugar, se han implementado estos algoritmos en Matlab para realizar un análisis empírico de sus propiedades basado en experimentos Montecarlo. Los algoritmos RRT y RRT* se han implementado desde cero, mientras que la implementación del A* se obtiene de [5]. A partir de estos experimentos se han obtenido las métricas de evaluación deseadas: longitud de la trayectoria obtenida y tiempo de ejecución.

Por último, se ha implementado el algoritmo RRT* en un Turtlebot 2. Para ello ha sido necesario aprender el funcionamiento del software ROS y se ha estudiado la arquitectura de software y hardware de la plataforma disponible en el laboratorio de robótica.

1.4. Estructura de la memoria

La memoria se estructura de la siguiente manera:

- Capítulo 2: en este capítulo se presenta el problema concreto de planificación sobre el que se va a trabajar y los algoritmos utilizados para ello.
- Capítulo 3: en este capítulo se realiza una comparación de los resultados obtenidos con estos algoritmos y, en consecuencia, las aplicaciones recomendadas para cada uno de ellos.

- Capítulo 4: en este capítulo se explica la implementación de los algoritmos de planificación en un robot real, desde el estudio de la plataforma ROS hasta las pruebas de campo realizadas y los resultados obtenidos.
- Capítulo 5: en este capítulo se plantean las conclusiones del trabajo junto a las dificultades que se han encontrado durante su desarrollo. También se plantean líneas de trabajo futuras que podrían mejorar y ampliar los resultados obtenidos en este trabajo.

Capítulo 2

Algoritmos de planificación de trayectorias

El principal objetivo de este trabajo es resolver el problema de la planificación de trayectorias a partir de (i) un modelo del entorno sobre el cual realizar la tarea de planificación, (ii) un origen y (iii) un destino. El problema concreto que se plantea resolver es el de la planificación de trayectorias para un solo robot en dos dimensiones.

Para obtener el modelo del entorno se parte de una imagen binaria que se discretiza en celdas del tamaño de un píxel. Estas celdas pueden ser obstáculos o celdas libres. Para discernir entre celdas libres y obstáculos se utiliza la matriz obtenida a partir del procesado de la imagen. Las celdas que toman valor 0 en la matriz se consideran obstáculos, y las que toman valor 1 se consideran celdas libres.

Las celdas de origen y destino se designan mediante las correspondientes coordenadas(x,y) en el modelo del entorno.

Para resolver el problema de planificación es necesario llevar al robot desde la celda origen hasta la celda destino sin colisionar con obstáculos. Para ello se utilizan los algoritmos de planificación A*,RRT y RRT*.

El cálculo de las acciones de bajo nivel que es necesario aplicar al robot para seguir esta trayectoria queda fuera del alcance de este proyecto.

2.1. Algoritmo A*

El algoritmo A* fue presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael [6]. Fue desarrollado como una extensión del algoritmo de Edsger Dijkstra [7]. El algoritmo A* tiene un mejor rendimiento que este debido a que

usa la heurística para guiar su búsqueda. Fue creado como parte del proyecto Shakey (Figura 2.1), que tenía el objetivo de construir un robot móvil que pudiera planificar sus propias acciones.

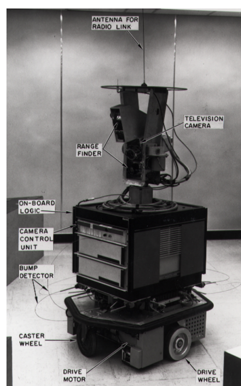


Figura 2.1: Proyecto Shakey

Este algoritmo es muy utilizado en el ámbito de la planificación de trayectorias y búsqueda en grafos debido a su eficiencia y precisión. La propiedad más destacada de este algoritmo es que, si existe, es capaz de encontrar la trayectoria de menor coste entre dos puntos.

En cada iteración el algoritmo decide cuál de sus caminos a expandir en función del coste $f(n)$. Esta función $f(n)$ consiste en la suma de la distancia desde el nodo de inicio hasta el nodo n ($g(n)$) y una estimación del coste requerido para extender la trayectoria hasta el objetivo ($h(n)$).

$$f(n) = g(n) + h(n). \tag{2.1}$$

La ecuación (2.1) describe la función de costes de cada celda.

Para comenzar la planificación se crean dos listas, la lista abierta y la lista cerrada. La lista cerrada maneja las celdas pertenecientes a la trayectoria y la lista abierta las celdas vecinas que se están analizando. La búsqueda parte de la celda inicial, analizando las celdas vecinas y sus costes. A continuación se elige la celda con menor $f(n)$ como sucesora, se añade a la lista cerrada y se realiza el mismo análisis. Si en alguna iteración del algoritmo se analiza una celda ya analizada previamente desde otra celda padre , su $f(n)$ se actualiza. Si esta $f(n)$ es menor que la que tenía previamente, se actualizan las coordenadas de su celda padre y se obtiene una trayectoria de menor coste.

La búsqueda de la trayectoria termina cuando la búsqueda alcanza la celda final. La sucesión de celdas desde la celda inicial con $g(n) = 0$ hasta la celda final con $h(n) = 0$, pertenecientes a la lista cerrada, conforman la trayectoria final. El pseudocódigo para el algoritmo A* se describe en Algoritmo 1.

Algorithm 1 Algoritmo A*

```
1: procedure PATH(x_init, x_goal)
2:     Inicializar Listas
3:     Añadir x_init a la Lista Cerrada
4:     while Lista Abierta no vacía do
5:         NuevaCelda = celda con mínima f en la Lista Abierta
6:         Mover NuevaCelda de la Lista Abierta a la Lista Cerrada
7:         Hijas = celdas adyacentes a NuevaCelda
8:         for  Hijas de NuevaCelda que no son obstáculos
9:             Calcular f(n)
10:            if f(n) < anterior f(n)
11:                Padre = NuevaCelda
12:            Explorar siguiente vecina
13:    if Lista Abierta vacía
14:            No existe solución
15:    if NuevaCelda = x_goal
16:            path = path entre x_init y x_goal mediante las celdas de la Lista Cerrada
17:    return path
```

2.2. Algoritmo RRT

Como alternativa a los algoritmos de búsqueda en grafos surgen los algoritmos basados en muestreo. Los algoritmos de muestreo más utilizados en la actualidad son el PRM y el RRT.

En esta sección se describe el algoritmo RRT, un algoritmo basado en muestreo desarrollado por Steven M. LaValle y James F. Kuffner Jr. en 1998 [3] para explorar espacios de grandes dimensiones construyendo un árbol. El árbol se construye de forma incremental a partir de muestras aleatorias del espacio y tiende a crecer hacia grandes áreas poco exploradas de forma inherente.

Una muestra es una celda del espacio, elegida de forma aleatoria, que no representa un obstáculo. Después de obtener cada muestra, se intenta realizar una conexión entre esta muestra y el nodo más cercano del árbol. Si esta conexión es factible, la muestra se añade como un nuevo nodo al árbol. Si la conexión al nodo más cercano no es posible, la conexión se realiza con el nodo del árbol con menor coste de entre los posibles candidatos que permiten una conexión libre de obstáculos.

Para detectar obstáculos entre dos puntos del espacio se utiliza el algoritmo de Bresenham [8]. Este algoritmo fue diseñado originalmente para dibujar líneas rectas en imágenes

binarias. En este trabajo se utiliza su capacidad de discretizar líneas rectas en celdas de la imagen binaria (Figura 2.2), con el fin de analizar si alguna de esas celdas es un obstáculo, y dar por imposible la conexión entre dos muestras del espacio.

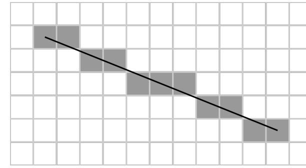


Figura 2.2: Discretización en celdas de una línea recta

Utilizando un muestreo uniforme del espacio, la probabilidad de expandir un estado existente es proporcional al tamaño de su región de Voronoi. Debido a que las regiones de Voronoi más grandes pertenecen a los nodos en la frontera de la búsqueda, el árbol se expande de forma preferente hacia grandes áreas inexploradas.

El principal inconveniente de este algoritmo es que la solución suele diferir en gran medida de la solución óptima.

El pseudocódigo para el algoritmo RRT se describe en el Algoritmo 2. La función RANDOM_STATE toma una muestra aleatoria del mapa para su posterior análisis. La función NEAREST_NEIGHBOR encuentra el nodo del árbol más cercano a la muestra. La función DETECT_OBSTACLE detecta si hay un obstáculo entre ambos. La función GET_PATH se encarga de computar la trayectoria a partir de las casillas inicial y final y las ramas del árbol que los unen.

Algorithm 2 Algoritmo RRT

1: **procedure** PATH(x_{init}, x_{goal})
2: Añadir x_{init} al árbol
3: **while** destino no alcanzado **do**
4: x_{rand} = RANDOM_STATE(Mapa)
5: x_{near} = NEAREST_NEIGHBOR(x_{rand}, árbol)
6: obstáculo = DETECT_OBSTACLE(x_{rand}, x_{near}, Mapa)
7: **if** not obstáculo
8: $x_{new} = x_{rand}$
9: Añadir x_{new} al árbol
10: path = GET_PATH(árbol, x_{init}, x_{goal})
11: **return** path

2.3. Algoritmo RRT*

En esta sección se describe el algoritmo RRT*, que hereda todas las propiedades del RRT y funciona de manera similar. Sin embargo, introduce dos nuevas características: búsqueda de vecinos cercanos y recableado(*rewiring*).

La búsqueda de vecinos cercanos encuentra el mejor nodo padre para el nuevo nodo antes de insertarlo en el árbol. La operación de recableado reconstruye las ramas del árbol que están dentro de esta área para mantener el mínimo coste posible entre los nodos del árbol. Esta reconstrucción sirve para eliminar las ramas redundantes, *i.e.*, ramas que no forman parte de la trayectoria de menor coste entre la raíz del árbol y un nodo. El proceso de reconstrucción asegura que la distancia desde la raíz del árbol hasta cada uno de sus nodos es siempre la mínima posible. A medida que se incrementa el número de nodos del árbol, el algoritmo RRT* mejora su trayectoria y su coste gradualmente debido a su comportamiento asintótico. Este comportamiento supone una mejora respecto al algoritmo RRT,que no mejora su trayectoria dentada y subóptima.

El pseudocódigo para el algoritmo RRT* se describe en Algoritmo 3. Las funciones RANDOM_STATE y GET_PATH realizan la misma función que en el algoritmo RRT. La función CHOOSE_PARENT conecta la muestra obtenida al nodo del árbol que garantiza el menor coste. REWIRE_TREE recablea los nodos en un área alrededor del nuevo nodo de forma que las ramas del árbol siempre garantizan el menor coste posible. La implementación de esta función se ha realizado mediante un bucle *for* que recorre todas las celdas del espacio dentro del área de recableado en busca de nodos del árbol existentes. Estos nodos se añaden a una lista para su análisis. A continuación se recalculan las distancias desde cada nodo hasta la raíz del árbol a través del nuevo nodo añadido. Si esta distancia es menor que la distancia obtenida anteriormente, se elimina la antigua conexión de dicho nodo y se conecta al nuevo nodo añadido al árbol. PATH_START busca el nodo de menor coste entre los nodos que han alcanzado la celda final.

Algorithm 3 Algoritmo RRT*

1: **procedure** PATH($x_{init}, x_{goal}, tiempo$)
2: Añadir x_{init} al árbol
3: **while** tiempo de ejecución < tiempo **do**
4: x_{rand} = RANDOM_STATE(Mapa)
5: x_{new} = CHOOSE_PARENT(x_{rand}, árbol)
6: REWIRE_TREE(x_{new}, árbol)
7: Añadir x_{new} al árbol
8: $path_{start}$ = PATH_START(x_{goal}, árbol)
9: path = GET_PATH(árbol, $path_{start}$, x_{init}, x_{goal})
10: **return** path

El radio del área de recableado es una variable que disminuye conforme aumenta el tamaño del árbol. El valor de esta variable se obtiene de [4] y se muestra en (2.2).

$$r(n) = \gamma_{RRT^*}(log(n)/n)^{1/d}, \tag{2.2}$$

donde n es el número de nodos del árbol, γ_{RRT^*} es una constante que depende de las dimensiones del mapa y d es el número de dimensiones del mapa, 2 en nuestro caso.

La variación del radio del área de recableado con el número de nodos del árbol para el mapa mediano se muestra en la Figura 2.3. Se observa que al inicio del proceso de planificación, cuando el árbol tiene pocos nodos, el radio del área de recableado es muy grande. Conforme se van añadiendo nodos al árbol, el radio de recableado disminuye.

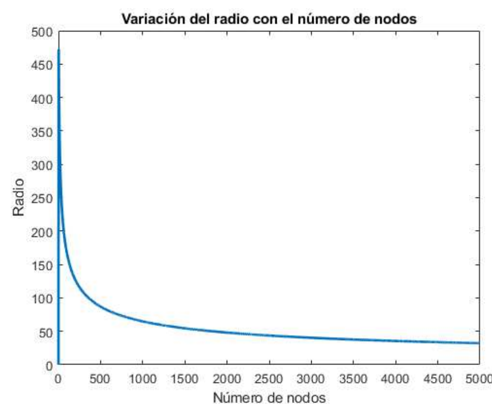


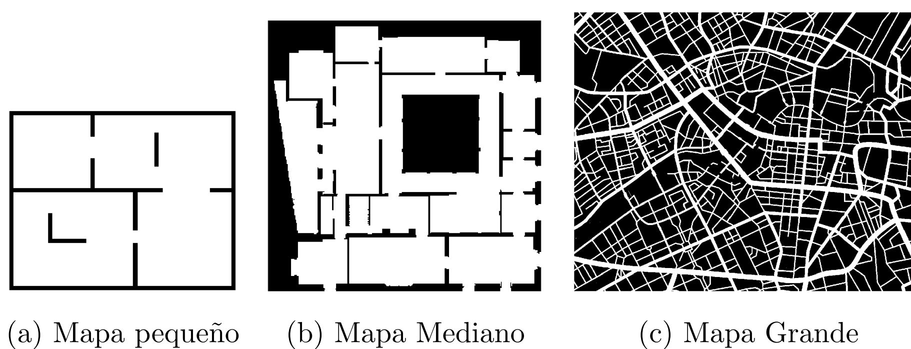
Figura 2.3: Variación del radio del área de recableado con el número de nodos

Capítulo 3

Resultados y discusión

3.1. Diseño del experimento

Para realizar un análisis empírico de los algoritmos planteados en el capítulo anterior, se han implementado y validado en Matlab. A continuación, se han construido 3 mapas de diferente complejidad y tamaño (Figura 3.1) para analizar el comportamiento de los algoritmos en diferentes entornos. Estos mapas corresponden a un entorno sencillo (mapa pequeño con unas dimensiones de 53x42 píxeles), al plano de un edificio (mapa mediano con unas dimensiones de 320x317 píxeles) y al mapa de una ciudad (mapa grande con unas dimensiones de 1069x861 píxeles). Estos mapas se discretizan en celdas del tamaño de un píxel, los píxeles blancos representan celdas libres de obstáculos y los píxeles negros representan obstáculos.



(a) Mapa pequeño (b) Mapa Mediano (c) Mapa Grande

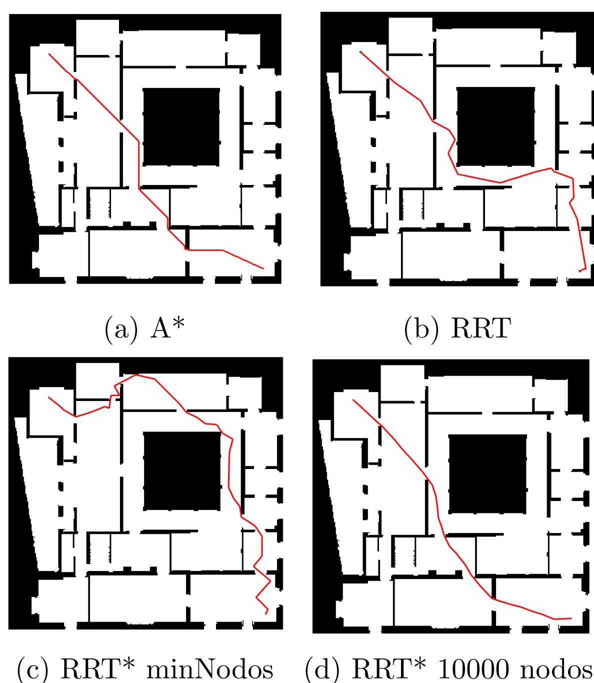
Figura 3.1: Mapas utilizados para el experimento

Para realizar el experimento se ejecuta cada algoritmo un número determinado de veces en cada mapa, a excepción del A* que solo se ejecuta en los mapas pequeño y mediano debido a su carga computacional en mapas grandes. Se ejecutan dos versiones del RRT*: la versión con área de recableado fija y la versión con área de recableado variable. El

número de ejecuciones es diferente para cada mapa: 1000 para el mapa pequeño, 500 para el mapa mediano y 100 para el mapa grande. Este número depende del tamaño del mapa debido a que las ejecuciones de los algoritmos son mucho más rápidas en el mapa pequeño que en el mapa grande. Por ello se elige un número de ejecuciones proporcional al tamaño del mapa. Para cada mapa se generan tantos valores aleatorios de celdas iniciales y finales como número de ejecuciones se van a realizar. Las métricas de evaluación son el tiempo de computación para cada trayectoria obtenida y la longitud de dicha trayectoria. Se eligen estas dos métricas porque se quiere contrastar la rapidez en el cálculo de la trayectoria con la optimalidad de la misma.

El experimento se ha realizado en un portátil *Mountain Onyx* con un procesador Intel(R) Core(TM) i7-4720HQ CPU 2.60GHz.

En la Figura 3.2 se muestra el comportamiento de cada algoritmo en el mapa mediano. Se observa como las ramas del árbol generado por el algoritmo RRT* devuelven un menor coste que las del RRT para la misma trayectoria. También se observa la diferencia entre ejecutar el algoritmo RRT* el mínimo tiempo necesario para calcular una trayectoria válida y fijar un tiempo de computación mayor para obtener una trayectoria óptima.



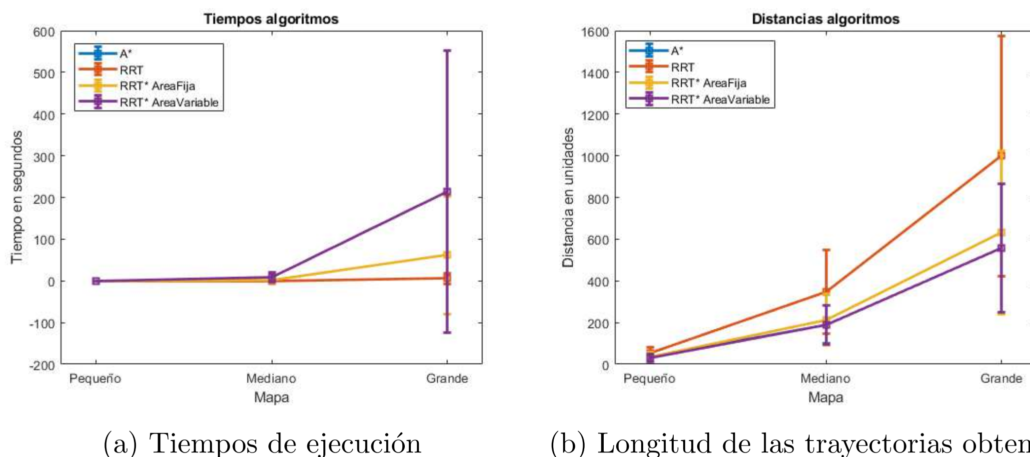
(a) A* (b) RRT

(c) RRT* minNodos (d) RRT* 10000 nodos

Figura 3.2: Trayectorias para diferentes algoritmos en el mapa mediano

3.2. Discusión

En la Figura 3.3 se pueden observar los resultados del experimento. En las gráficas se muestra la media y la desviación estándar de las longitudes de las trayectorias obtenidas y del tiempo de computación utilizado por el algoritmo A* en los mapas pequeño y mediano, y para el resto de algoritmos en los tres mapas.



(a) Tiempos de ejecución (b) Longitud de las trayectorias obtenidas

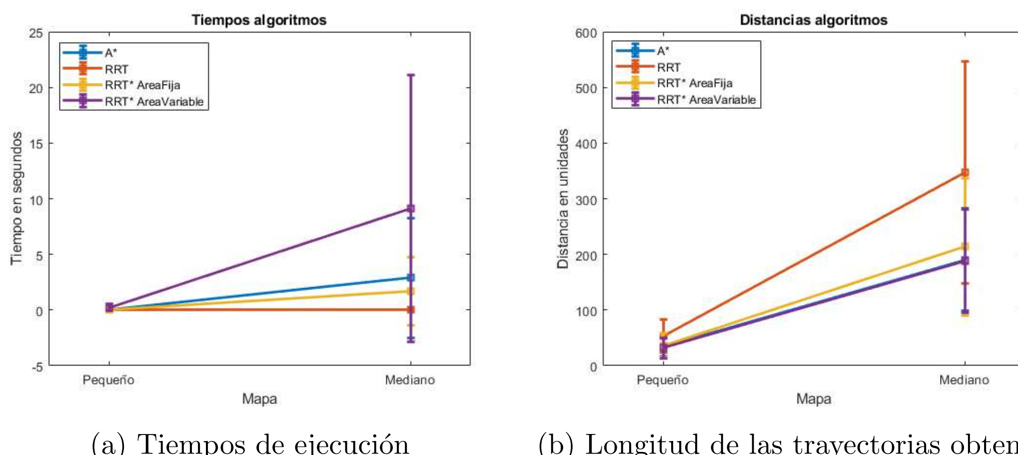
Figura 3.3: Resultados del experimento

El algoritmo RRT es el que menores tiempos de computación consigue en todos los mapas. Se observa una media de 0.038 s en el mapa mediano frente a los 1.67 s del algoritmo RRT* con área fija y los 2.89 s del algoritmo A*. Sin embargo, la longitud de las trayectorias obtenidas es mayor que en el resto de algoritmos en todos los mapas, una media de 347.2 en el mapa mediano frente a los 213.7 del algoritmo RRT* con área fija. Estos resultados demuestran que aunque el algoritmo RRT tiene unos tiempos de computación muy bajos debido a que está basado en muestreo, su solución está lejos de ser la óptima. Por ello es necesario la utilización de modificaciones *anytime-optimal* como el RRT*.

También destacan las diferencias entre el algoritmo RRT* con área fija y el algoritmo RRT* con área variable. La media de las longitudes de las trayectorias obtenidas para el algoritmo RRT* con área variable en el mapa grande es un 88 % de la media para algoritmo RRT* con área fija, pero su tiempo de computación medio es de 214 s frente a los 61.74 s de la versión con área fija.

Para el algoritmo A*, se demuestra que la longitud de la trayectoria obtenida es siempre la mínima posible y que los tiempos de computación son mayores que los de los algoritmos basados en muestreo, a excepción del RRT* con área variable. En comparación con el RRT*, el algoritmo A* no consigue unos tiempos de computación aceptables para mapas grandes, mientras que el algoritmo RRT* sí lo hace, y eligiendo un tiempo de

ejecución adecuado, podemos obtener una trayectoria con longitud mínima.



(a) Tiempos de ejecución (b) Longitud de las trayectorias obtenidas

Figura 3.4: Resultados del experimento para los mapas pequeño y mediano

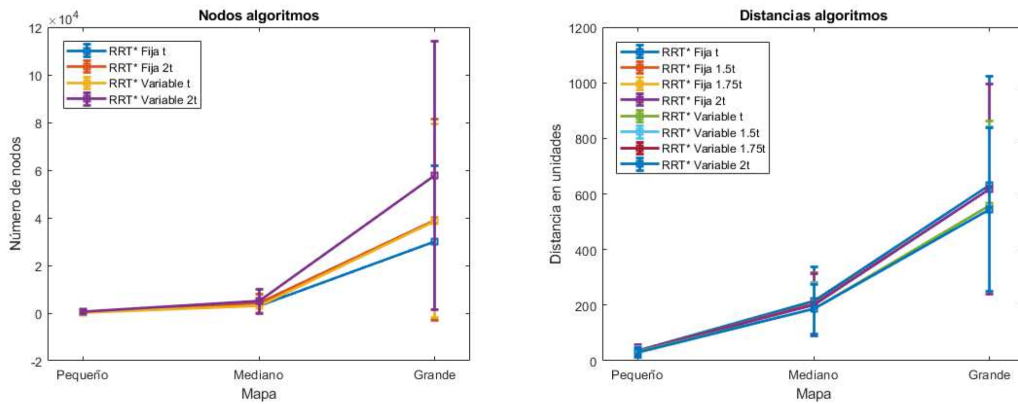
En la Figura 3.4 se observa que la longitud media de las trayectorias obtenidas mediante el algoritmo RRT* es ligeramente menor que para el algoritmo A*. Esto se debe a que el algoritmo A* solo calcula distancias a lo largo de la diagonal o de los laterales de la celda, mientras que el RRT* simplemente conecta puntos del mapa y mide la distancia entre ellos. Por tanto, el algoritmo RRT es útil si se quieren planificar trayectorias en el mínimo tiempo posible y no importa la longitud de la trayectoria obtenida, mientras que el RRT* es una muy buena opción si se quiere tener en cuenta la longitud de la trayectoria , ya que se puede elegir el tiempo de computación en función de la optimalidad de la trayectoria deseada. El algoritmo A* es muy útil cuando se quiere planificar una trayectoria óptima en mapas de tamaño moderado, ya que en estos casos el tiempo de computación no es muy elevado.

También se observa que los tiempos de computación del algoritmo RRT* con área variable son mucho mayores que los del RRT* con área fija, e incluso más altos que los del algoritmo A*, lo que no responde a las propiedades de un algoritmo basado en muestreo. Este resultado puede deberse a una implementación poco eficiente de la función de recableado del árbol. Podrían utilizarse técnicas más avanzadas de búsqueda o representación de variables para obtener mejores resultados.

A continuación se realiza una comparación en profundidad de las dos versiones del algoritmo RRT*. Se toman los tiempos de computación de cada uno de las versiones cuando alcanzan el objetivo (t) y la ejecución continúa hasta alcanzar un tiempo de ejecución ($2t$) para observar el comportamiento *anytime-optimal* del algoritmo RRT*. Se recogen el número de nodos añadidos al árbol y la longitud de la trayectoria obtenida para cada trayectoria cuando el tiempo transcurrido es t, $1.5t$, $1.75t$ y $2t$.

El primer dato a comentar es el del número de nodos generado por cada algoritmo.

En la Figura 3.5 se muestra el número de nodos generado por cada algoritmo en cada mapa. Para el algoritmo RRT* con área fija la diferencia de nodos entre los instantes $t = t_{fija}$ y $t = 2t_{fija}$ es de 29798 en el mapa grande, mientras que la diferencia entre los instantes $t = t_{fija}$ y $t = 2t_{fija}$ es de 10108. Este resultado se debe a que a medida que se añaden más nodos al árbol, más tiempo se invierte en el recableado del árbol, y menos en la generación de nuevos nodos. Para el caso del RRT* con área variable, el área se reduce conforme aumenta el número de nodos, obteniendo una diferencia de nodos entre los instantes $t = t_{variable}$ y $T = t_{variable}$ de 38743 en el mapa grande, mientras que la diferencia entre los instantes $t = t_{variable}$ y $t = 2t_{variable}$ es de 18976. Esto se debe a que el tiempo de computación invertido en el recableado se reduce conforme aumenta el tamaño del árbol.



(a) Número de nodos añadidos al árbol (b) Longitud de las trayectorias obtenidas

Figura 3.5: Resultados de la comparación de ambas versiones del RRT*

En la Figura 3.5 también se muestran las longitudes de las trayectorias obtenidas por cada versión del algoritmo en cada mapa. La media de la longitud de la primera trayectoria generada por el RRT* con área fija en un tiempo t_{fija} en el mapa grande es de 632.4, y se reduce a 616.9 para un tiempo $2t_{fija}$. Para la versión del RRT* con área variable la media de la longitud de la primera trayectoria obtenida en un tiempo $t_{variable}$ es de 557.4, y se reduce a 544.9 en un tiempo $2t_{variable}$. Esta reducción de la longitud de la trayectoria no es muy significativa con respecto al coste de computación asociado. Como se observa en la Figura 3.4, el RRT* con área variable ya obtiene una trayectoria muy cercana a la óptima con un tiempo de ejecución $t_{variable}$.

En conclusión, la versión del algoritmo RRT* con área variable obtiene trayectorias más cercanas a la óptima que la versión con área fija, pero su tiempo de ejecución es mucho mayor, lo que no compensa esta mejora en la longitud de la trayectoria. Sin embargo, este resultado puede deberse a una mala implementación del algoritmo RRT*. Además, eligiendo un área de recableado fija adecuada podemos reducir la longitud de la trayectoria con unos tiempos de ejecución bajos.

Capítulo 4

Implementación en robot real

En este capítulo se muestran los resultados de la implementación del algoritmo RRT* en un robot Turtlebot 2 (Figura 4.1) para comprobar su aplicación en un entorno real. La prueba se realiza conectando el planificador implementado en Matlab con el robot real mediante el software ROS (*Robot Operating System*) [9] y la Robotics Toolbox de Matlab.

Figura 4.1: Robots modelo Turtlebot 2

4.1. Introducción a ROS

ROS provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más.

La estructura de ROS se muestra en la Figura 4.2. Los conceptos básicos necesarios para entender el funcionamiento de esta estructura son:

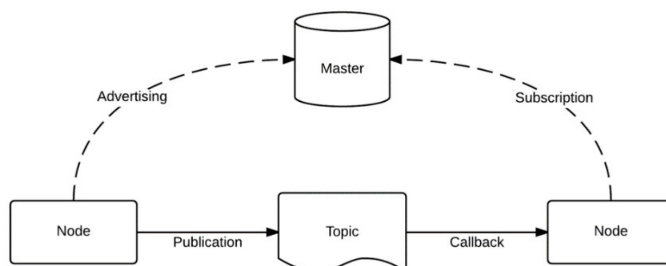


Figura 4.2: Estructura de ROS

- Master: proporciona servicios de denominación y registro al resto de los nodos del sistema ROS. También se encarga de establecer las conexiones entre los nodos. El Master es necesario en todo sistema que use ROS, ya que sin él es imposible establecer una comunicación entre nodos.
- Nodos: los nodos son los procesos encargados de realizar la computación. Es muy común que en los sistemas que utilizan ROS existan varios nodos, cada uno encargado de una función específica. Para el caso de un robot móvil, un nodo puede encargarse del sensor láser, otro de la odometría, otro de la localización, etc.
- Tópicos: los tópicos son buses mediante los cuales los nodos intercambian información. Los nodos pueden suscribirse o publicar a un tópico.

4.2. Implementación en ROS del algoritmo RRT*

El algoritmo seleccionado para la implementación en robots reales es el RRT*, debido a su gran versatilidad, ya que nos permite elegir el tiempo máximo que queremos dedicarle a la obtención de una trayectoria. Como se ha explicado en el capítulo anterior, la calidad de la trayectoria obtenida dependerá de dicho tiempo.

Dado que el objetivo de este trabajo de fin de grado es el estudio de algoritmos de planificación de trayectorias, para la gestión a bajo nivel de los robots y realizar la comunicación con Matlab se utiliza el software existente en el equipo de robótica. El trabajo realizado en este TFG ha sido llevar a cabo la comunicación entre el algoritmo RRT* y la capa de bajo nivel de los robots.

En la Figura 4.3 se muestra el esquema de la arquitectura necesaria para la implementación en un robot.

- AMCL: este nodo se encarga de la localización del robot en un mapa previamente calculado mediante un sensor láser y el algoritmo AMCL (Adaptative Monte Carlo Localization) y la publica en el tópico amcl_pose.

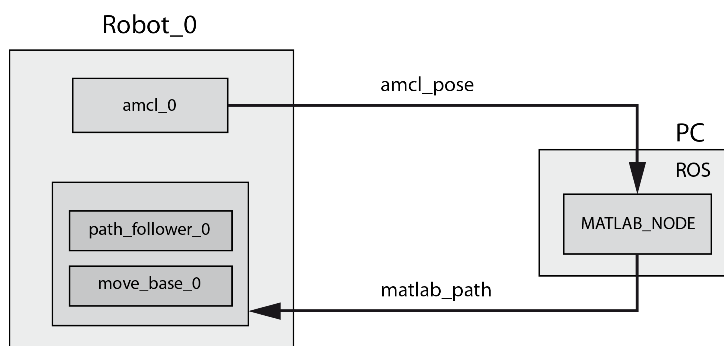


Figura 4.3: Implementación en ROS

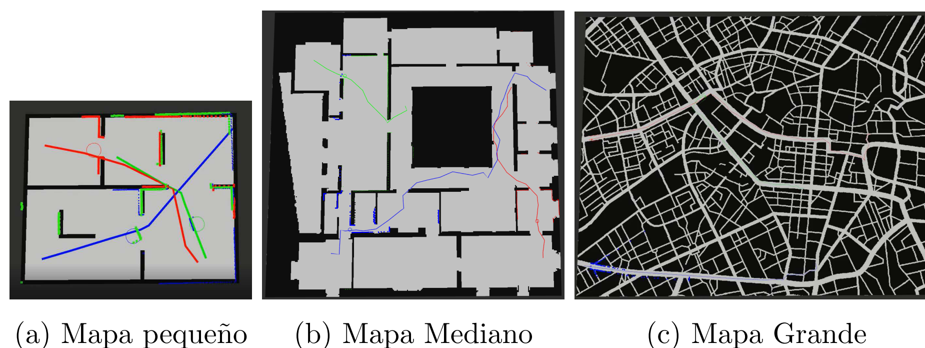
- MATLAB_NODE: se suscribe al tópico amcl_pose para obtener la localización y la orientación del robot, y utiliza esta información para realizar la planificación con el algoritmo RRT*. La trayectoria obtenida se publica en el tópico matlab_path.
- path_follower: se sucribe a matlab_path y obtiene la trayectoria que el robot debe seguir.
- move_base: usa un planificador local para llevar al robot a los diferentes puntos de la trayectoria. Debido a este planificador local, en ocasiones el robot no sigue la trayectoria planificada de forma estricta.

La simulación del movimiento de los robots se realiza en Stage y RViz. Stage es un simulador de robots que provee un mundo virtual poblado por robots móviles y sensores, junto a objetos que los robots pueden detectar y manipular. RViz es un visualizador 3D para la infraestructura ROS. Para ello ha sido necesaria la adaptación de los mapas a Stage y RViz mediante la modificación de los correspondientes archivos .world y .yaml.

Las simulaciones se realizan para cada uno de los mapas usando tres robots. Utilizando como base el software comentado anteriormente, se generan trayectorias para cada uno de ellos mediante el algoritmo RRT*. Estas trayectorias tienen como celda inicial la posición del robot, y como celda final una localización aleatoria del mapa. A continuación, cada robot sigue su trayectoria hasta alcanzar la celda final. Cuando uno de los robots alcanza su destino, se genera una nueva trayectoria y el robot comienza a moverse de nuevo.

4.3. Resultados

La Figura 4.4 muestra los resultados obtenidos para tres robots en los diferentes mapas en los que se han realizado las pruebas. En las imágenes se muestra la trayectoria asignada a cada robot.



(a) Mapa pequeño (b) Mapa Mediano (c) Mapa Grande

Figura 4.4: Simulaciones en RViz

Los resultados obtenidos demuestran la viabilidad de implementar el planificador en robots reales, debido a que la comunicación entre la simulación y el robot real es transparente. Las trayectorias obtenidas permiten a los robots alcanzar su destino sin colisionar con obstáculos y recorriendo la mínima distancia posible para el tiempo asignado.

Se observa que los robots tienen problemas de localización en el mapa grande, debido a que existen varias áreas del mapa con geometrías similares. Este problema podría solucionarse mediante el ajuste adecuado del algoritmo de localización Montecarlo(AMCL) utilizado.

Como consecuencia de haber realizado pruebas con tres robots en paralelo, pueden existir colisiones entre los mismos, ya que no se utiliza ningún algoritmo que las evite.

También se observa que el tamaño real de los robots es mayor que una celda, por lo que trayectorias viables para un “robot teórico” con el tamaño de una celda pueden no ser viables para un robot real.

Capítulo 5

Conclusiones

El objetivo principal de este proyecto ha sido estudiar e implementar diferentes algoritmos de planificación de trayectorias para robots móviles en entornos con obstáculos. Por un lado, el trabajo ha tenido una parte de estudio teórico en la que se han analizado los algoritmos más utilizados en la literatura (A*, RRT, RRT*). Por otro lado, en base a un análisis empírico, se ha planteado la validación del algoritmo RRT* en una plataforma robótica real.

La implementación en Matlab de los algoritmos A*, RRT y RRT* ha permitido estudiar su comportamiento en diferentes entornos.

El algoritmo A* ha demostrado ser muy eficiente en los mapas pequeño y mediano, pero su tiempo de computación es mayor que el de los algoritmos de muestreo, lo que imposibilita su utilización en mapas de grandes dimensiones.

El algoritmo RRT ha presentado unos tiempos de computación muy bajos en comparación con el algoritmo A*. Sin embargo, devuelve una trayectoria que, por lo general, está muy lejos de la óptima.

El algoritmo RRT* ha obtenido unos resultados mejores que los del algoritmo RRT, debido a la optimización de la trayectoria que realiza en cada iteración. Adicionalmente, el tiempo de computación elegido por el usuario determina la optimalidad de la solución obtenida. Por tanto, eligiendo un tiempo de computación adecuado podemos obtener una trayectoria óptima con un tiempo de computación menor que el del algoritmo A*. Se ha observado que en términos de tiempo de ejecución es mejor elegir un área de recableado fija adecuada, lo que devuelve buenos resultados y tiene un tiempo de ejecución menor que el de la versión del algoritmo RRT* con un área de recableado variable. Este resultado, sin embargo, puede deberse a una mala implementación de la función de recableado del árbol.

Los resultados obtenidos a partir de la implementación en ROS han demostrado la viabilidad de utilizar el planificador en robots reales, debido a que la comunicación entre la simulación y los robots reales es transparente. Las trayectorias obtenidas permiten a los robots alcanzar su destino sin colisionar con obstáculos y recorriendo la mínima distancia posible para el tiempo asignado.

5.1. Líneas de trabajo futuras

Hay varios puntos fuera del alcance de este trabajo, cuyo estudio sería interesante para mejorar los resultados obtenidos.

Como posible línea de trabajo futura se plantea una mejor implementación del algoritmo RRT*, en concreto de la función de recableado del árbol.

También se podrían estudiar otros algoritmos de muestreo como el PRM, y realizar una comparación con los algoritmos RRT y RRT*.

Otra posible línea de trabajo podría ser mejorar la integración del planificador con los robots, e implementar un algoritmo que evite colisiones entre ellos.

Bibliografía

[1] John Canny. *The complexity of robot motion planning.* MIT press, 1988.

[2] Rodney A Brooks and Tomas Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):224–233, 1985.

[3] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[4] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[5] Davinia Vera Soriano. Planificación y control con restricciones de formaciones de robots. 2011 Recuperado el 10 de febrero de 2017, de `https://zaguan.unizar.es/record/6479?ln=es`.

[6] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[9] Ros website. `http://wiki.ros.org/`.