UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

PLANIFICACIÓN DE MOVIMIENTO EN ROBÓTICA MÓVIL UTILIZANDO RETÍCULAS DE ESTADOS

MEMORIA
MÁSTER EN INVESTIGACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Presentada por:
Adrián González Sieira
Dirigida por:
Dr. Manuel Mucientes Molina
Santiago de Compostela, Julio de 2011

Índice

Resumen

Problema y solución presentada

La planificación de movimientos supone guiar el robot desde un punto de inicio a uno de fin, a través de una ruta libre de obstáculos. Un algoritmo de planificación de movimientos debe producir, a partir del objetivo que se pretende alcanzar y de los datos disponibles del entorno en el que se debe llevar a cabo la navegación, una serie de comandos que permitan al robot transitar de forma segura en un entorno con obstáculos hasta completar la tarea.

Para poder abordar el problema es necesario almacenar en todo momento información sobre el entorno del robot. Lo más adecuado para un problema de planificación de movimiento 2-D es un mapa tipo grid. Es importante que esta información esté lo más actualizada posible, ya que en base a ella el algoritmo de planificación toma decisiones sobre cuál es la mejor ruta de entre todas las posibles que conecta el estado actual y el objetivo.

Disponer de información *a priori* en el mapa del entorno ayuda a que la solución inicial aportada por el planificador se ajuste más al problema planteado, y por tanto se reduzcan el número de replanificaciones a realizar durante la ejecución. Tanto en caso de disponer de un mapa previo como en caso contrario, el sistema debe tomar en cada instante la información de los sensores con los que está equipado el robot, y combinar la información anterior con la nueva. Por esta razón la solución aportada no puede ser estática, en tanto que puede ser válida en ese momento, pero no si se producen cambios en el entorno que afectan al camino planificado. En este caso, de acuerdo a los cambios que se hayan producido, es necesario comprobar en qué medida afectan a la solución previa, y refinarla hasta que sea adecuada a la nueva disposición de obstáculos en el entorno del robot.

La planificación de movimiento se ha planteado como un grafo dirigido, donde los nodos son los estados del robot, definidos en cinco dimensiones: $(x, y, \theta, v, \omega)$, y los arcos son los movimientos discretos, o trayectorias primitivas, que son las acciones de control realizables por el robot que permiten transitar entre ellos. Cada uno de estos movimientos está asociado a una orden de control concreta, por lo que la solución aportada por el algoritmo de búsqueda es una composición de estos movimientos, y por tanto de órdenes de control, que permiten alcanzar el punto objetivo partiendo del estado actual del robot. Tanto el los estados del robot como el conjunto de acciones son discretos, por lo que se ha planteado la planificación como un proceso resoluble mediante un algoritmo de búsqueda.

El algoritmo que mejor se adapta al problema que se pretende resolver es el Anytime Dynamic A* (AD*). Una de las características de AD* es su capacidad de replanificación, fundamental en entornos dinámicos (como es el caso de la mayoría de entornos reales). Además, el algoritmo permite ajustar la optimalidad de la solución obtenida, cuanto menos restrictivo sea el nivel de optimalidad de la solución que se pretende conseguir, más rápida es la ejecución del proceso de búsqueda, por lo que este parámetro se puede ajustar en función del tiempo disponible para la obtención de una nueva solución.

Para poder guiar el algoritmo de búsqueda ha sido necesario definir dos heurísticas diferentes: una para aportar información sobre el coste de transición entre estados teniendo en cuenta la información de los obstáculos del entorno sin tener en cuenta las restricciones de movimiento impuestas por las trayectorias realizables por el robot, y otra que sí las utiliza, pero considera el entorno completamente libre de obstáculos.

Cuando una solución está disponible, es enviada al controlador, que envía los comandos de movimiento al robot. Todas las órdenes de control pueden ejecutarse en el robot, puesto que se han calculado utilizando su modelo de movimiento.

Todos los algoritmos se han desarrollado para que los procesos que realizan puedan ser independientes del conjunto de órdenes de control realizables por el robot, por lo que en caso de cambiar el modelo de robot, o la forma de movimiento que se quiere utilizar para planificar la ruta a los objetivos, no es necesario realizar ningún cambio sobre la implementación del planificador. Los resultados que se presentan en este trabajo son utilizando un robot *Pioneer 3-DX*, que permite giros sobre sí mismo.

Contribuciones

- Algoritmo capaz de llevar a cabo replanificación en entornos dinámicos, esto es esencial para evitar realizar todo el proceso de planificación desde el inicio cada vez que se detecta una necesidad de calcular una nueva solución.
- Algoritmo capaz de producir soluciones *anytime*, en función del tiempo disponible para el cálculo de la solución, se realiza el cálculo permitiendo un determinado nivel de sub-optimalidad.
- Se tienen en cuenta las restricciones cinemáticas del robot utilizado, las rutas que se generan son siempre realizables por el robot móvil.
- La introducción de las restricciones cinemáticas se realiza sin penalizar el funcionamiento en tiempo real del algoritmo, a pesar de la alta dimensionalidad del estado.

1. Introducción

La planificación de movimientos en robótica tiene como finalidad la generación de un conjunto de acciones que permitan a un robot móvil 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 planificación de movimientos es la navegación en entornos con una gran cantidad de obstáculos.

El problema puede ser resuelto en diferentes niveles de abstracción. En cualquier caso, para un correcto funcionamiento en el 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 como 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, realizando a más bajo nivel una conversión de las órdenes emitidas por el planificador a comandos de movimiento para ejecutar la ruta planificada. Este método falla 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 la ruta. No es lo mismo planificar para un robot con un movimiento de tipo Ackerman (figura 1), que para uno de tipo diferencial, que permite giros sobre sí mismo.

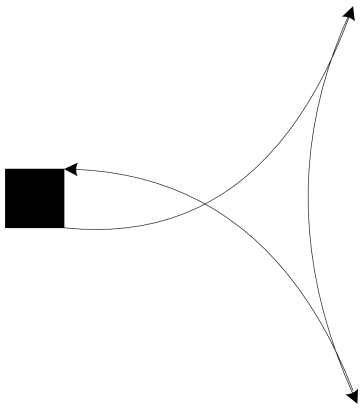


Figura 1: Movimiento de tipo Ackerman

La planificación se puede plantear como un problema de búsqueda de tal forma que expresando la información relevante del problema en forma de grafo, es posible ejecutar un algoritmo que minimice una función de coste en un camino sobre el grafo.

En la mayor parte de los algoritmos de planificación, la búsqueda del camino se realiza teniendo en cuenta sólo la posición del robot, dejando el control del mismo como un problema a resolver independientemente del anterior; 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 de control del robot al mismo tiempo que al problema de búsqueda del camino, ya que se introduce 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, no son transitables.

Para el desarrollo de este proyecto se ha utilizado un robot tipo *Pioneer 3-DX* [10] (figura 2). Es un robot terrestre de tres ruedas con movimiento de tipo diferencial, es decir: capaz de realizar giros parado o en movimiento.

Figura 2: Robot modelo *Pioneer 3-DX*

En este trabajo de fin de máster se han desarrollado un conjunto de algoritmos que, en su conjunto, generan rutas libres de obstáculos y que cumplen las restricciones cinemáticas del robot. El sistema funciona en tiempo real y permite, ademas, la construcción del mapa del entorno, así como la visualización del mismo y de la ruta generada. Los componentes que se han desarrollado a lo largo de este proyecto son:

- Un módulo de software que permite construir y actualizar un mapa a partir de la información que el robot recibe de sus sensores.
- Un algoritmo de planificación que obtiene una secuencia de órdenes de control que, partiendo del estado inicial, permite alcanzar el objetivo correctamente.
- Un conjunto de heurísticas que permiten estimar el coste de transitar entre diferentes estados del robot, necesarias para llevar a cabo el proceso de búsqueda de forma rápida y eficiente.
- Un módulo de control que permite secuenciar las órdenes de control devueltas por el algoritmo de planificación y aplicarlas a los motores del robot para llevar a cabo el movimiento que le permite alcanzar el punto de destino.

2. Estado del arte

Existen multitud de algoritmos de planificación en robótica, y la mayor parte de ellos se pueden agrupar en:

- Técnicas geométricas [6]
- Campos de potencial [1, 6]
- Algoritmos basados en muestreo [6]

A continuación se exponen las ventajas e inconvenientes de cada uno de ellos en comparación con la solución presentada en este trabajo.

Técnicas geométricas

Esta aproximación está basada en la diferenciación de las zonas libres del entorno y las ocupadas, manteniendo una representación del entorno que permita el cálculo de rutas en las regiones etiquetadas como libres. Estos algoritmos son completos, en el sentido que son capaces de conectar dos puntos cualesquiera del espacio libre, siempre que se cumplan dos condiciones:

- Accesibilidad: dos puntos cualesquiera del espacio libre están conectados entre sí por una ruta libre de obstáclos.
- Conservación de la conectividad: es posible representar el espacio libre de obstáculos mediante un grafo donde todas las rutas entre puntos del espacio libre estén presentes como conexiones entre nodos del grafo.

Estos algoritmos tienen su base en la representación del robot y el espacio ocupado por los obstáculos: cuando dicha representación reúne es de baja dimensionalidad y los obstáculos son de forma convexa, son capaces de proporcionar buenas soluciones.

Para algunos problemas de planificación especiales existen algoritmos basados en este método que son completos e independientes de la aproximación al problema, cuyas implementaciones son muy eficientes y ofrecen resultados con un tiempo de computación mucho menor que los métodos basados en muestreo. Para los problemas que no encajan en ninguno de estos algoritmos, existen otros de propósito general que pueden abordarlo, aunque con tiempos de computación mayores; su ventaja es que permiten calcular un tiempo máximo teórico de resolución del problema, mientras que para otros métodos es más complicado de estimar.

Uno de los problemas más relevantes de esta aproximación es que cuando el entorno tiene un gran número de obstáculos su representación requiere una gran capacidad computacional, y puede llegar a ser imposible encontrar una solución al problema en un tiempo razonable. Para resolver la planificación de movimientos es necesario que el control del robot sea fluido, y que puedan llevar a cabo replanificaciones en caso de actualizarse la información del entorno del robot sin tener que espear un tiempo muy elevado a que la nueva ruta sea calculada [6]. Aunque todos los algoritmos de planificación presentan un rendimiento mejor cuanto más simple sea el entorno en el que se ejecutan, no es asumible que en presencia de una gran cantidad de obstáculos, los requisitos computacionales sean tan elevados que hagan imposible la resolución del problema en un tiempo razonable.

Estos métodos de resolución son, por lo tanto, poco adecuados si se quieren obtener rutas muy largas, en entornos con una gran extensión, o en presencia de numerosos obstáculos que dificulten la navegación. En este caso, los métodos basados en muestreo presentan una complejidad computacional más controlable y predecible, y no son tan sensibles a la configuración de obstáculos presentes en ámbito del robot, sino que su rendimiento está basado en la implementación realizada.

Otra de las principales desventajas de este tipo de aproximaciones es la gran diferencia que hay entre distintos algoritmos. La forma de representación, resultados y tiempo de computación son muy diferentes, y no presentan el mismo comportamiento ante diferentes tipos de entornos. También hay que tener en cuenta la complejidad de implementación de algunos de ellos, que aunque presentan resultados muy prometedores, en la práctica son muy complejos de comprender y de aplicar al problema deseado.

Finalmente, este tipo de aproximaciones no modela adecuadamente la presencia de zonas desconocidas o de incertidumbre, y asumen que las zonas libres y ocupadas son estáticas. Cada vez que se produce un cambio en este sentido, es necesario modificar la representación realizada del entorno. Esto los hace no aplicables a entornos reales donde es frecuente la presencia de obstáculos móviles.

Campos de potencial

Los campos de potencial se utilizan para decidir cuál es la dirección más prometedora en cada instante para alcanzar la meta. Para conseguirlo, utilizan la teoría de los campos eléctricos y el robot, la meta y los obstáculos del entorno tienen asociado un potencial. Lo que se pretende conseguir es que el robot sea atraído por la meta, a la vez que es repelido por los obstáculos.

Una de las principales ventajas de estos métodos es que consiguen resolver problemas en entornos simples utilizando muy pocos recursos computacionales. Por contrapartida, no son métodos muy recomendables para abordar la resolución de problemas complejos, puesto que siguen una estrategia *greedy*: en cada momento se estudia la influencia de los potenciales emitidos por los diferentes objetos del entorno, para decidir cuál es la dirección de avance más prometedora. Esto hace que el método no sea robusto ante posibles mínimos locales, siendo necesaria una estrategia de más alto nivel para evitarlos.

Comparado con el método propuesto en este trabajo, los campos de potencial no suponen ninguna ventaja para planificar movimientos en entornos en presencia de una gran cantidad de obstáculos o para planificar rutas de gran longitud. Además del inconveniente de evitar mínimos locales, esta aproximación no asegura que, en caso de encontrar un camino entre la posición de inicio y la posición final, éste sea óptimo. Por último, estos métodos no tienen en cuenta las restricciones cinemáticas del robot.

Algoritmos basados en muestreo

Los algoritmos basados en muestreo tratan de resolver el problema de la planificación de movimientos evitando la descripción explícita de las regiones ocupadas por obstáculos, tratando de representar el espacio a partir de un esquema de discretización. Esta familia de algoritmos es completamente independiente del problema de detección de colisiones y, por tanto, también del modelo geométrico utilizado para describir el entorno del robot.

Este tipo de aproximación se puede considerar como la más exitosa de todas las que se encuentran en el estado del arte de la planificación de movimientos, y es ampliamente utilizada en robótica móvil, problemas de manufactura, y aplicaciones biológicas con muy alta dimensionalidad, que serían intratables desde las perspectivas anteriores y que requieren una representación completa del espacio de obstáculos.

Algunos de los métodos que se han descrito anteriormente, como los basados en combinatoria, aseguraban que si una solución existía, sería encontrada en un tiempo finito. Los algoritmos basados

en muestreo no son, en este sentido, completos, y la resolubilidad del problema depende del concepto de densidad; sí son, sin embargo, probabilísticamente completos, lo cual significa que, con un número suficiente de puntos, la probabilidad de encontrar una solución existente tiende a 1.

Esta representación del entorno del robot se puede combinar con técnicas de planificación discreta, como por ejemplo algoritmos de búsqueda, donde la ruta a obtener entre dos posiciones se expresa como una consulta a realizar a dicho algoritmo.

Una ventaja importante de esta aproximación respecto a las otras es que es posible introducir las limitaciones del modelo cinemático del robot en el proceso de planificación, definiendo además de un espacio de estados, un espacio de acciones realizables. Este espacio de acciones puede ser o no dependiente del espacio de estados, en cuyo caso cada acción tiene siempre definidos un estado de partida y uno de llegada. En el problema de la planificación de movimiento, existe una relación biunívoca entre cada uno de los elementos del espacio de acciones y las órdenes de control que permiten ejecutarlas, por lo que no es necesario determinar *a posteriori* los comandos de movimiento que hay que ejecutar para implementar la solución obtenida por el planificador. Muchas rutas obtenidas por un planificador que no tenga en cuenta esto son inválidas porque son irrealizables en la práctica. Los métodos geométricos o los campos de potencial no son válidos para dar solución a la planificación de movimientos con restricciones cinemáticas, ya que no los tienen en cuenta de ninguna forma.

El trabajo presentado en esta memoria es una aproximación basada en muestreo mediante retículas de estados que, además tiene en cuenta las restricciones cinemáticas del robot para realizar la planificación de movimientos. Utilizar esta aproximación tiene una clara ventaja, y es que el proceso de planificación es independiente de la representación del entorno del robot por lo que su rendimiento no se ve tan penalizado por la disposición o el número de obstáculos como en las aproximaciones combinatorias, y es robusto frente a mínimos locales, al contrario que los algoritmos basados en campos de potencial. La solución aportada en este trabajo utiliza un algoritmo de búsqueda para dar solución al proceso de planificación, por lo que en buena medida la complejidad computacional puede ser reducida hasta un límite inferior de $O(n)$, donde n es la longitud de la solución, en el caso de que la heurística estimase perfectamente el coste para el problema a resolver.

3. Arquitectura del sistema

Como ya se ha mencionado, para planificar una ruta es necesario conocer el entorno en el que se va a realizar el movimiento. En algunas ocasiones, se dispone de conocimiento *a priori*, mediante mapas del entorno de navegación. En otras, sin embargo, se parte de una situación de total desconocimiento, y la situación de los obstáculos se va conociendo a medida que el robot navega y procesa las medidas de los sensores. El mapeado del entorno es una parte importante de la aplicación, y su función principal es mantener actualizada la información disponible sobre los obstáculos presentes en la zona en tiempo real, de forma que el algoritmo de planificación siempre se ejecute con información actualizada.

Las partes más relevantes del sistema desarrollado son las que se encargan de ejecutar la planificación y el control. El módulo de planificación conecta el estado de partida y el estado objetivo del robot mediante una ruta que cumpla las restricciones de movimiento y libre de obstáculos. Cuando esta ruta se ha planificado, el módulo de control envía los comandos correspondientes a los motores del robot hasta que se alcanza el punto de destino. El algoritmo de control tiene capacidad de replanificación.

Esta situación se da cuando:

- Las zonas que forman parte del camino generado han cambiado su grado de ocupación.
- El robot se ha desviado de la ruta programada.
- La ruta planificada es sub-óptima, y existe posibilidad de mejorarla.

El módulo de control debe evaluar en cada una de las situaciones el grado de importancia que tiene la replanificación, y decidir si es necesario esperar a que el módulo de planificación obtenga una nueva ruta, o si es posible continuar la ruta actual y replanificar desde un punto futuro, para no detener la navegación del robot.

El conjunto de módulos desarrollados en este proyecto se detallan en la figura 3, las funcionalidades que implementan cada uno de ellos son:

- Módulo de actualización de mapa: permite la construcción del mapa del entorno, la carga de información a partir de un fichero de mapa, y su actualización en tiempo real, como se concreta en la sección 4.
- Módulo de planificación: permite la obtención de una ruta que conecta el estado actual del robot con el punto objetivo, como se detalla en la sección 5.
- Módulo de generación de trayectorias primitivas: permite la obtención del conjunto de trayectorias primitivas que conectan los diferentes estados del robot, como se explica en detalle en la sección 5.2.
- Módulo de generación de la tabla HLUT para la heurística FSH, definido en la sección 5.4.2.
- Módulo de control: permite extraer del camino generado por el planificador el conjunto de órdenes de control que permiten realizarlo, detecta en qué momento hay necesidad de replanificar el camino, y envía los comandos de movimiento a los motores del robot. Este módulo se explica más en detalle en la sección 6.
- Módulo de visualización del planificador: permite ver en forma gráfica el mapa del robot, el estado en el que se encuentra, y la ruta generada por el planificador.
- Módulo de visualización de las trayectorias primitivas: permite ver gráficamente todas las trayectorias primitivas generadas, y los detalles de cada una de ellas.

  Algunas de las herramientas que proporcionan los módulos implementados son ejecutadas de forma *offline* al proceso de planificación, como la generación de las trayectorias primitivas, la generación de la tabla HLUT, o la visualización de las trayectorias primitivas.

  Los módulos de actualización de mapa, de planificación y de control se deben ejecutar concurrentemente para que el sistema funcione. Cada uno de ellos ejecuta su función de forma independiente de los demás, comunicando los resultados al resto de módulos. La visualización es un módulo opcional que se puede ejecutar bajo demanda del usuario, pero no es necesario para la ejecución de la planificación.

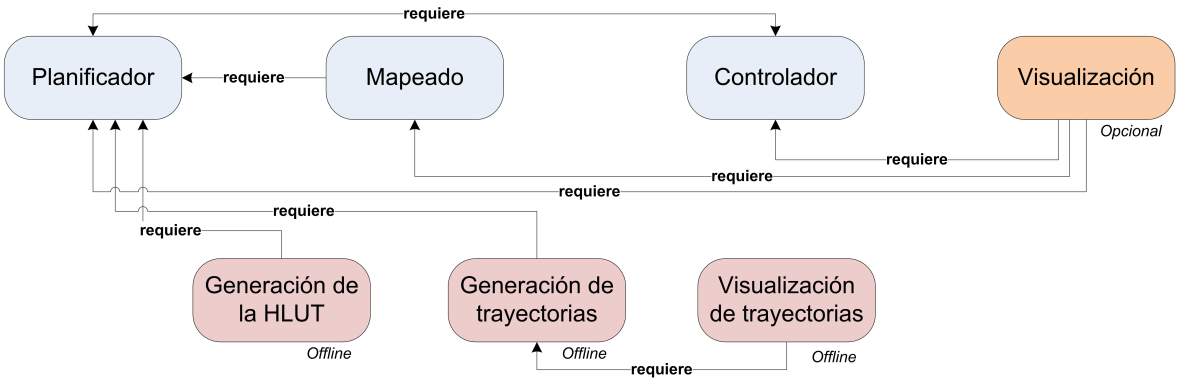


Figura 3: Diagrama de módulos

4. Mapeado del entorno

La planificación de rutas requiere conocer en tiempo real el estado del entorno en el que se encuentra el robot. Esto implica consultar periódicamente la información que ofrecen los sensores del robot, establecer en qué posiciones están situados los obstáculos respecto a la pose actual, transformar dichas posiciones del mapa local al mapa global, y actualizar la información más reciente en el mapa combinando la ya conocida con la nueva proporcionada por los sensores.

En primer lugar, es necesario definir el tipo de estructura que va a necesitar el algoritmo de planificación para consultar si una determinada posición está libre u ocupada. Mantener para cada punto del espacio información sobre su grado de ocupación sería inviable debido a la cantidad de información que habría que procesar. Además debido a la incertidumbre asociada a los sensores y a la posición del robot, lo habitual es trabajar con mapas de ocupación discretizados.

Por ello, se define el mapa como un conjunto de celdas de tamaño determinado, en una estructura de grid de celdas [13, 14]. Cada una de las celdas afecta a una región cuadrada del espacio, y contiene información sobre su probabilidad de ocupación, que se determina en función del número de veces que esa celda ha sido vista por los sensores del robot como libre u ocupada.

El mapa del entorno se construye inicialmente de tal forma que la posición inicial es la (0, 0), y el resto de posiciones se establecen de forma relativa a ésta. Este mapa, expresado en coordenadas globales, es el que contiene la información necesaria para ejecutar el proceso de planificación.

Para hacer la operación de actualización del mapa global realmente eficiente, y que el tiempo de actualización no escale con su tamaño, se utiliza un segundo mapa de dimensiones más reducidas, que abarca toda la zona alrededor del robot y con un radio igual al alcance de los sensores del robot. Las coordenadas de este segundo mapa son relativas a la posición del robot en cada momento, por lo que cuando el robot está en movimiento, el mapa local se mantiene centrado en él. Realizando un cambio de coordenadas, es posible establecer de forma inmediata la correspondencia entre las celdas de ambos mapas.

La correspondencia entre medidas de los sensores y celdas del mapa local se puede precalcular (figura 5).

Para cada celda se calcula el ángulo mínimo y máximo que forma respecto a la posición del robot.

Este intervalo define el conjunto de haces de los sensores que pasan a través de ella, y por tanto ofrecen información acerca de su grado de ocupación. El haz seleccionado para cada celda será aquel que pase más cercano al centro, ya que la información que ofrece es más representativa. Una vez conocido, se calculan los puntos de corte con los lados de la celda, y se seleccionan los dos más cercanos al centro de la misma (figura 4). La distancia de estos dos puntos a la posición relativa del robot respecto a la celda del mapa local determina el intervalo de valores de la medida entre los que su estado debe ser de celda ocupada. En las sucesivas actualizaciones del mapa local, el estado de cada celda se selecciona en función del valor de su medición asignada:

- Si $d_{medida} > d_{max}$, celda libre.
- Si $d_{medida<d_{min}}$, estado no conocido, ya que el valor de la medida no la alcanza.
- Si $d_{medida \geq d_{min}}$ y $d_{medida} \leq d_{max}$, celda ocupada.

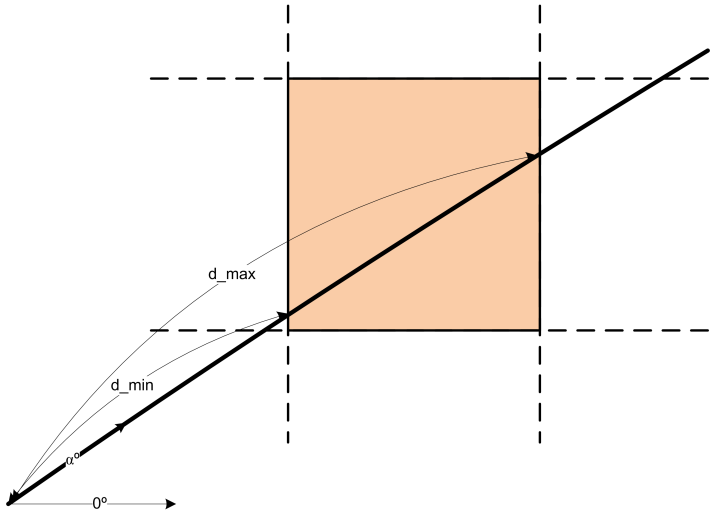


Figura 4: Determinación del intervalo $[d_{min}, d_{max}]$ que establece una celda como ocupada

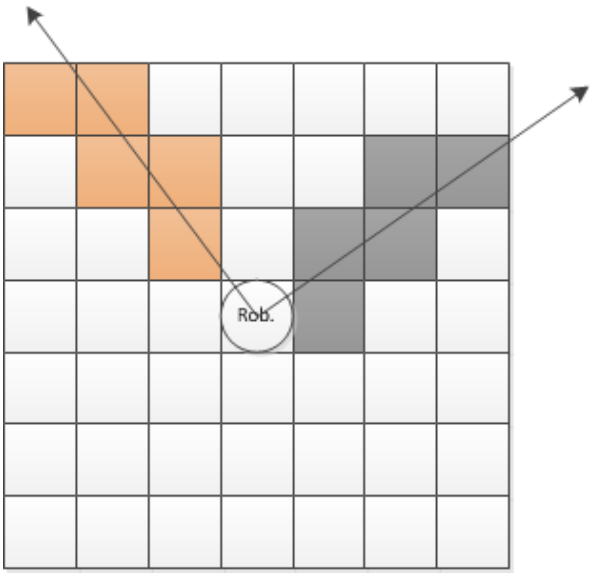


Figura 5: Celdas del mapa local que se ven afectadas por distintas medidas

La actualización del mapa local no tiene en cuenta la información de los sensores en instantes anteriores, por lo que éste será sólo una representación de las medidas de los sensores en un instante determinado.

Para actualizar el mapa global, se define un número máximo de actualizaciones (n_{act}) que van a ser tenidas en cuenta para el cálculo de la probabilidad de ocupación (p_{occ}). Este factor determina el número de veces que es necesario actualizar la celda para que tenga $p_{occ} = 1$. La probabilidad de ocupación de una celda se calcula como:

$$p_{occ} = \frac{1}{1 + (0{,}8^{n_{free} - n_{occ}} \cdot 0{,}2^{n_{occ} - n_{free}})} \tag{1}$$

Cuando la suma del número de veces que la celda ha sido vista como libre (n_{free}) y como ocupada (n_{occ}) es menor que el factor n_{act}, el proceso de actualización se reduce a actualizar estos valores según el nuevo estado de la celda. Cuando se alcanza dicho umbral, es necesario ajustar los valores, tal y como se describe en el algoritmo 1.

Para simplificar la información que contiene el mapa de cara al proceso de planificación, se transforma la probabilidad de ocupación, p_{occ} de cada una de las celdas en un conjunto predefinido de estados:

- Celda ocupada: $p_{occ} \geqslant 0{,}8$
- Celda desconocida: $0{,}2 < p_{occ} < 0{,}8$
- Celda libre: $p_{occ} \leqslant 0{,}2$

La inicialización del mapa asume que la probabilidad de ocupación inicial de las celdas es 0,5, lo que se corresponde con un estado de desconocimiento. Sin embargo en algunas ocasiones, el entorno es conocido *a priori*, por lo que el módulo de gestión de mapas permite cargar ficheros *ppm* con el mapa del entorno.

Algoritmo 1 Actualización de n_{free} y n_{occ}

Entrada: $n_{occ}(t-1)$, $n_{occ}(t)$, $n_{free}(t-1)$, $n_{free}(t)$

Salida: n_{occ}, n_{free}

if $n_{occ}(t) + n_{free}(t) \geq n_{act}$ **then**

$$n_{occ} = \frac{n_{occ}(t)}{n_{free}(t)} \cdot n_{act}$$

$$n_{free} = n_{act} - n_{occ}$$

else

$$dif_{free} = n_{free}(t-1) - n_{free}(t)$$

$$dif_{occ} = n_{occ}(t-1) - n_{occ}(t)$$

$$aux = \frac{dif_{occ}}{dif_{occ} + dif_{free}}$$

$$n_{occ} = aux \cdot (n_{act} - (n_{occ}(t) + n_{free}(t))) + n_{occ}(t)$$

$$n_{free} = (n_{act} - (n_{occ}(t) + n_{free}(t))) - aux \cdot (n_{act} - (n_{occ}(t) + n_{free}(t))) + n_{free}(t)$$

end if

Algoritmo 2 Actualización del mapa global

while forever **do**

leer los datos de los sensores y la pose real del robot

actualizar el grado de ocupación de las celdas del mapa local

for all $celda \in celdas_{mapaLocal}$ **do**

comprobar posición de la celda en el mapa global

obtener datos de la celda en el instante anterior, $n_{free(t-1)}$ y $n_{occ(t-1)}$

if celda en el mapa local está marcada como libre **then**

$n_{free(t)} = 1$ y $n_{occ(t)} = 0$

else if celda en el mapa global está marcada como ocupada **then**

$n_{free(t)} = 0$ y $n_{occ(t)} = 1$

end if

if $n_{free(t)} + n_{free(t-1)} + n_{occ(t)} + n_{occ(t-1)} > n_{act}$ **then**

obtener valores ajustados de n_{free} y n_{occ}

else

$n_{free} = n_{free(t-1)} + n_{free(t)}$

$n_{occ} = n_{occ(t-1)} + n_{occ(t)}$

end if

actualizar p_{occ} de la celda del mapa global

end for

end while

El tamaño del mapa no es ilimitado; en la mayor parte de los casos se pueden establecer unas dimensiones iniciales del mapa en función de la posición inicial del robot y de la posición de destino deseada. Un mapa demasiado grande ocupa mucho espacio en memoria, por lo que es necesario establecer un límite inicial que permita realizar la planificación adecuadamente sin desperdiciar los recursos del sistema. Una situación diferente es cuando se carga información del entorno a partir de un fichero *ppm*. En este caso, las dimensiones del mapa dependerán del número de píxeles del archivo y de la resolución indicada por el usuario, independientemente de la posición de inicio y de fin que se utilice para planificar. El tamaño se puede aumentar bajo demanda, si es necesario. Es posible añadir o eliminar nuevas celdas al mapa según las necesidades del sistema, sin que el proceso de actualización se vea afectado.

5. Algoritmo de planificación de rutas

Un problema de planificación de rutas en robótica puede expresarse de tal forma que sea resoluble mediante la aplicación de un algoritmo de búsqueda de la familia de A* [2, 7]. Este tipo de algoritmos de búsqueda exploran los diferentes nodos y sus relaciones en busca del camino de coste mínimo, y su eficiencia está basada en la heurística que utilizan.

Para que la solución devuelta por el algoritmo de búsqueda sea el camino de coste mínimo entre el nodo de inicio y el nodo de fin, es necesario que la heurística entre dos nodos, $h(S, S')$ cumpla las siguientes condiciones:

- Admisibilidad: el valor de heurística es optimista, y nunca es superior al valor de coste real de transición entre dos estados, cumple por tanto que $h(n) \leq g(s)$, donde $g(s)$ es el coste a la meta.
- Consistencia: el valor de heurística para cualquier nodo, más el coste de haber transitado desde el inicio hasta él, nunca debe ser superior al valor acumulado de coste y heurística de cualquier otro nodo del espacio de búsqueda, y por tanto $h(x) \leq c(n,\, a,\, n') + h(n')$, para cualquier acción a.

En este proyecto, la planificación de rutas se ha resuelto con un algoritmo (AD*), que permite replanificación y obtención de sucesivas soluciones con mayor grado de optimalidad. En el estado del arte se pueden encontrar una gran cantidad de algoritmos que están orientados a la replanificación en entornos dinámicos, y otros orientados a la producción de soluciones con un determinado umbral de sub-optimalidad que permita acelerar la búsqueda. El problema de la navegación móvil, sin embargo, exige la aplicación de un algoritmo de búsqueda que combine los beneficios de las dos aproximaciones anteriores, en tanto que es necesaria una etapa de replanificación cada vez que el robot detecta los cambios en el mapa para poder construir una nueva solución en caso de que sea necesario.

Anytime Dynamic A* [9] (AD*) es un algoritmo de búsqueda que se ejecuta sobre un grafo dirigido, capaz de producir soluciones con un determinado nivel de sub-optimalidad en un tiempo muy rápido, y que se va mejorando en sucesivas iteraciones del algoritmo. La calidad de la solución ofrecida por el algoritmo se puede ajustar en función del tiempo de que se dispone para realizar la búsqueda, y en cada paso del algoritmo de búsqueda se utiliza la información generada en los pasos previos, con lo que la planificación no se ejecuta desde el inicio en cada uno de los pasos. Además AD* permite

Variable	**Resolución**
Posición en X	0.5 m
Posición en Y	0.5 m
Orientación	0.78539 rad
Velocidad lineal	0.125 m/s
Velocidad angular	0.26179 rad/s

Cuadro 1: Valores de resolución utilizados en este proyecto.

aplicar los cambios detectados sobre el grafo de búsqueda, y en cada iteración se repara de forma incremental la solución previa. Su código se detalla en los algoritmos 3, 4, 5 y 6.

Para resolver el problema de la planificación mediante este algoritmo de búsqueda es necesario definir el concepto de nodo, y de relación entre nodos. Cada nodo del grafo se corresponde, en este caso, con un estado del robot, definido por la combinación de los siguientes valores:

- Pose del robot: (x, y, θ)
- Velocidad lineal: v
- Velocidad angular: ω

En el estado del arte existen propuestas para resolver problemas de planificación de movimiento utilizando aproximaciones parecidas a las propuestas en este trabajo, pero con una dimensionalidad menor del problema y basados en planificación de movimientos tipo Ackerman [3, 8] (figura 1).

Si se definen los nodos del espacio de búsqueda como estados del robot, es posible diseñar las conexiones entre los nodos de tal forma que se tengan en cuenta las restricciones cinemáticas del vehículo. Así, el planificador sólo construirá rutas que sea posible realizar con el modelo de movimiento del vehículo y evitando, por ejemplo, cambios de velocidad u orientación imposibles de ejecutar por los motores.

En este problema, una relación entre dos estados del robot está representada por la trayectoria que los conecta. El cálculo de las relaciones existentes entre nodos es un proceso demasiado costoso, que es necesario realizar *offline*, y se detalla en la sección 5.2.

5.1. Discretización del espacio de búsqueda

En este proyecto, se ha utilizado una discretización especial del espacio de estados del robot, la retícula de estados [3], que permite formular el problema de la planificación como un grafo dirigido. En ese grafo, los nodos son todos los estados alcanzables por el robot, y los arcos son los movimientos que permiten transitar entre distintos estados.

La retícula de estados requiere establecer unos valores de resolución, que van a definir el intervalo de repetición de la máscara de conexiones que forma el grafo de búsqueda. Los valores de resolución utilizados en este proyecto en las diferentes variables que forman el estado del robot se muestran en la tabla 1.

Cuanto menor sea la resolución utilizada para discretizar el espacio de estados, más capacidad de planificación se pierde, ya que los estados están más distantes entre sí y en entornos que requieren

Algoritmo 3 Función principal de AD*

$g(s_{start}) = rhs(s_{start}) = \infty$
for all $s_g \in GOALS g(s_g) = \infty$
$\epsilon = \epsilon_0$
$OPEN = CLOSED = INCONS = \emptyset$
insert for all $s_g \in GOALS$ insert s_g in $OPEN$ with $key(s_g)$
$computeOrImprovePath()$
publish ϵ-suboptimal solution
while forever **do**
  if $\epsilon = 1$ **then**
    wait for changes in edges cost or changing start node
  end if
  if $s_{newstart} \neq s_{start}$ **then**
    change node start
  end if
  if changes in edges cost detected **then**
    for all edges (u, v) with changed costs **do**
      update edge cost
      $updateState(u)$
    end for
  end if
  if significant costs detected **then**
    increase ϵ or replan from scratch
  else if $\epsilon > 1$ **then**
    decrease ϵ
  end if
  move states from $INCONS$ to $OPEN$
  update priorities for all $s \in OPEN$ with $key(s)$
  $CLOSED = \emptyset$
  $computeOrImprovePath()$
  publish ϵ-suboptimal solution
end while

Algoritmo 4 computeOrImprovePath()

while $(min_{s \in OPEN}(key(s) < key(s_{start})$ OR $rhs(s_{start})) \neq g(s_{start})$ **do**
  remove s with minimum key from $OPEN$
  if $g(s) > rhs(s)$ **then**
    $g(s) = rhs(s)$
    $CLOSED = CLOSED \cup \{s\}$
    for all $s' \in Pred(s)$ **do**
      $UpdateState(s')$
    end for
  else
    $g(s) = \infty$
    for all $s' \in Pred(s) \cup \{s\}$ **do**
      $UpdateState(s')$
    end for
  end if
end while

Algoritmo 5 $UpdateState(s)$

if s not visited before **then**
  $g(s) = \infty$
end if
if $s \neq s_{goal}$ **then**
  $rhs(s) = min_{s' \in Succ(s)}(c(s, s') + g(s'))$
end if
if $s \in OPEN$ **then**
  remove from $OPEN$
end if
if $g(s) \neq rhs(s)$ **then**
  if $s \notin closed$ **then**
    insert s in $OPEN$ with $key(s)$
  else
    insert s in $INCONS$
  end if
end if

Algoritmo 6 $key(s)$

if $g(s) > rhs(s)$ **then**
  return $[rhs(s) + \epsilon \cdot h(s_{start}, s); rhs(s)]$
else
  return $[g(s) + h(s_{start}, s); g(s)]$
end if

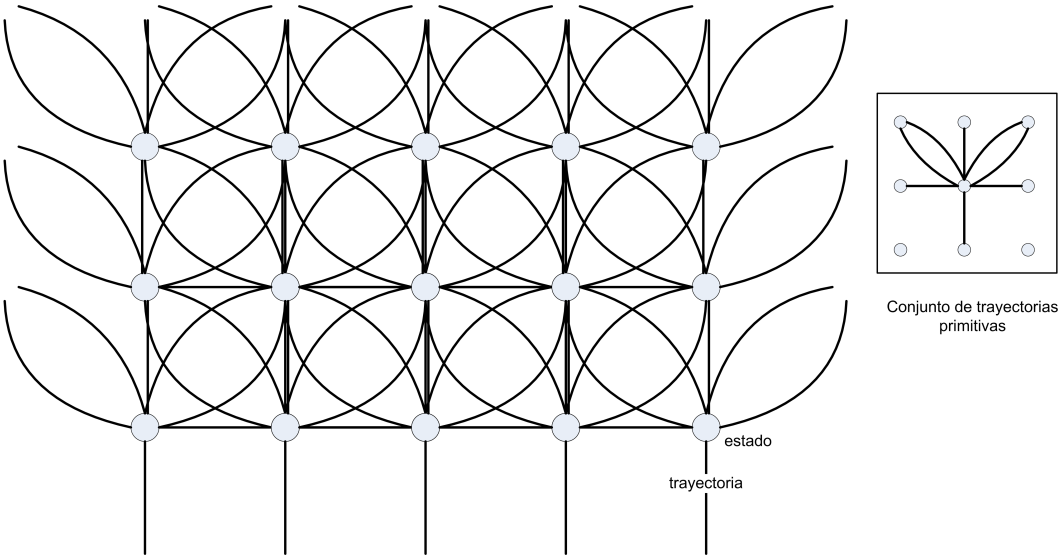


Figura 6: Regularidad en la retícula del espacio de estados.

maniobras más precisas el planificador podría fallar, tal y como se muestra en la figura 7. Utilizar niveles de resolución altos favorece, por una parte, la obtención de rutas con un nivel de optimalidad mayor entre el punto de inicio y el destino y, por otra, una mejor planificación en zonas con pasos estrechos (como pasillos, puertas, etc) o en presencia de una gran densidad de obstáculos; pero también aumenta de forma considerable el coste computacional de obtener la solución al problema.

5.2. Generación de las trayectorias primitivas

Una vez definida la discretización del espacio de búsqueda es necesario definir la conectividad que tienen los estados entre sí, mediante un conjunto de trayectorias primitivas que se calculan *a priori*, y de forma independiente al proceso de planificación. Este conjunto de trayectorias contiene todos los posibles movimientos realizables por el robot entre nodos vecinos.

Algoritmo 7 Esquema de generación de la máscara de trayectorias

```
for all θ_ini, θ_fin ∈ θ and v_ini, v_fin ∈ v and ω_ini, ω_fin ∈ ω do
  for all x, y ∈ vecinos do
    t = generarTrayectoria([0, 0, θ_ini, v_ini, ω_ini], [x, y, θ_fin, v_fin, ω_fin])
    if t ≠ null then
      añadir trayectoria a la máscara
      break
    end if
  end for
end for
```

El resultado de comprobar la conectividad entre todos los posibles estados de inicio y de fin es un conjunto de conexiones representada por un pequeño grafo dirigido, donde tanto los estados de inicio

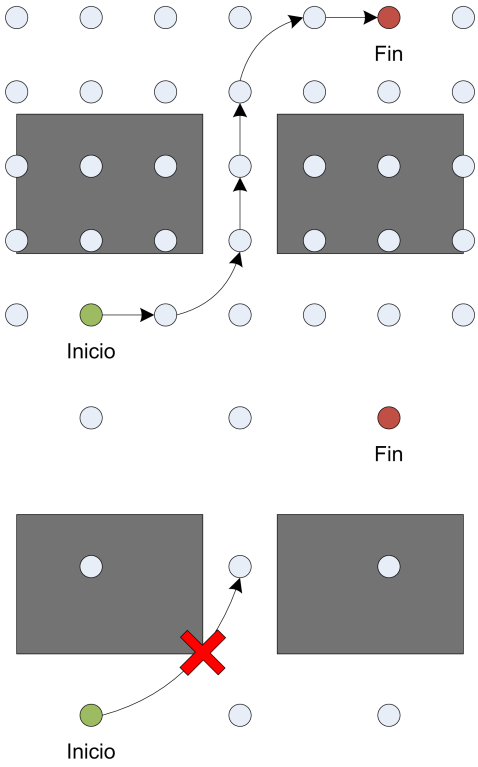


Figura 7: Planificación con diferentes resoluciones del espacio de estados

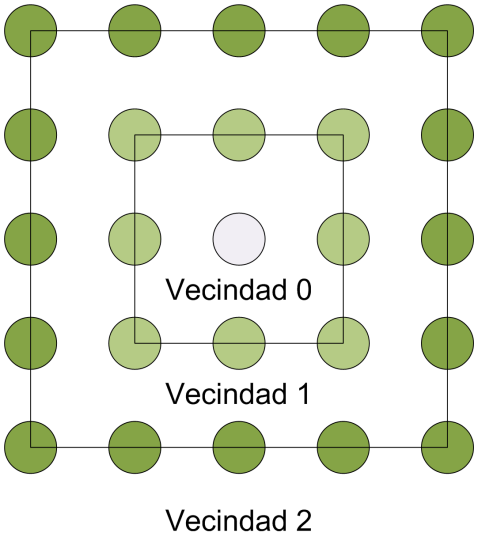


Figura 8: Niveles de vecindad para la generación de trayectorias primitivas

como los de fin están dentro del conjunto discreto de estados. Replicando la máscara de conexiones en intervalos regulares se forma una retícula [12], que permite ejecutar el algoritmo AD* de forma eficiente para estudiar el camino óptimo libre de obstáculos entre el nodo de inicio y de fin.

Algoritmo 8 Función $generarTrayectoria([0,0,\theta_{ini},v_{ini},\omega_{ini}],[x,y,\theta_{fin},v_{fin},\omega_{fin}])$

$t_{min} = max\{0{,}33; abs(v_i - v_f)/a_{max}; abs(\omega_i - \omega_f)/\phi_{max}\}$
$t_{max} = d_{neighbor}/v_{min} + 2 \cdot \alpha_{neighbor}/\omega_{min}$
for all $t_{actual} \in \{t_{min}$ to t_{max} by $t_c\}$ **do**
  generate profiles for v and ω
  filter ω profiles checking $error(\theta_{profile}, \theta_f)$
  for all combinations of ω filtered profiles and v profiles **do**
    generate points of trayectory from $t = 0$ to t_{actual} by t_cs
    while error getting down AND $t < t_{actual} + t_c$ **do**
      generate new point from with $t = t + t_m$
    end while
  end for
  select combination of profiles of v and ω with minimum error
  if $error < 1\,\%$ **then**
    return trayectory generated
  end if
end for
return null

Este proceso tiene en cuenta las simetrías de transformación, rotación y reflexión, con el fin de ahorrar tiempo de computación. El esquema de ejecución se detalla en los algoritmos 7 y 8.

Los perfiles de velocidad lineal y angular asociados a una trayectoria se han definido de forma trapezoidal. Estos perfiles recogen el conjunto de comandos a aplicar durante la ejecución de la trayectoria. Los perfiles trapezoidales permiten una mayor flexibilidad en las trayectorias generadas, puesto que constan de dos intervalos de tiempo de aceleración, y dos más en los que se mantiene una velocidad constante. Cada uno de los perfiles, velocidad lineal y angular, queda completamente descrito con 7 valores, tal y como se muestra en la figura 9.

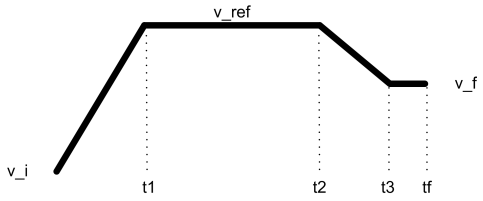


Figura 9: Forma trapezoidal de los perfiles de velocidad asociados a las órdenes de control.

Para comprobar si dos estados del robot están conectados, es necesario generar todos los perfiles de velocidad lineal y angular cuyos valores inicial y final se corresponden con los estados que se están comprobando. Como el tiempo y la velocidad son variables continuas, el número de trapecios que pueden conectar los valores inicial y final es infinito. Por esta razón, es necesario realizar una discretización en ambas, que mantenga la flexibilidad de los perfiles trapezoidales a la vez que se mantiene razonable el tiempo de computación necesario para estudiar la conectividad entre dos estados. Ésta se ha definido de la siguiente forma:

- Para las velocidades lineal y angular: se utilizan los mismos valores que pueden formar parte del

Estado	**Peso**
Libre	1
Ocupada	∞
Desconocida	1000

Cuadro 2: Pesos asignados a las celdas del mapa según su estado.

conjunto de todos los estados discretos del robot.

- Para el tiempo: en el intervalo $[0, t_f]$, las variables temporales que definen el trapecio (t_1, t_2, t_3), pueden tomar valores cada t_c, que es el tiempo entre comandos de control enviados al robot.

La generación de los puntos de la trayectoria se realiza aplicando los perfiles de velocidad al modelo de movimiento del robot. Las trayectorias generadas han sido obtenidas de tal forma que se minimice el tiempo empleado en realizar la conexión entre los dos estados del robot. Con la resolución utilizada en las variables de los estados del robot definida en la tabla 1, se obtiene un número de trayectorias muy elevado, superior a las 8.000. Sin embargo, es posible reducir este número significativamente, eliminando aquellas trayectorias con un coste superior a la combinación de otras trayectorias primitivas. Así, el número de trayectorias que se mantienen es cercano a las 5.000. En la figura 10 se pueden ver las trayectorias salientes, en azul, y las trayectorias entrantes, en rojo, para diferentes estados del robot.

5.3. Coste de transición entre estados

La función de coste de transición entre dos estados del robot asigna pesos a los arcos del grafo dirigido que forma el espacio de búsqueda. Para este proyecto se ha diseñado esta función de tal forma que el coste sea una combinación de dos variables: la información sobre los obstáculos que contiene el mapa, y el tiempo que se tarda en transitar del estado de inicio al estado de fin utilizando la trayectoria primitiva correspondiente.

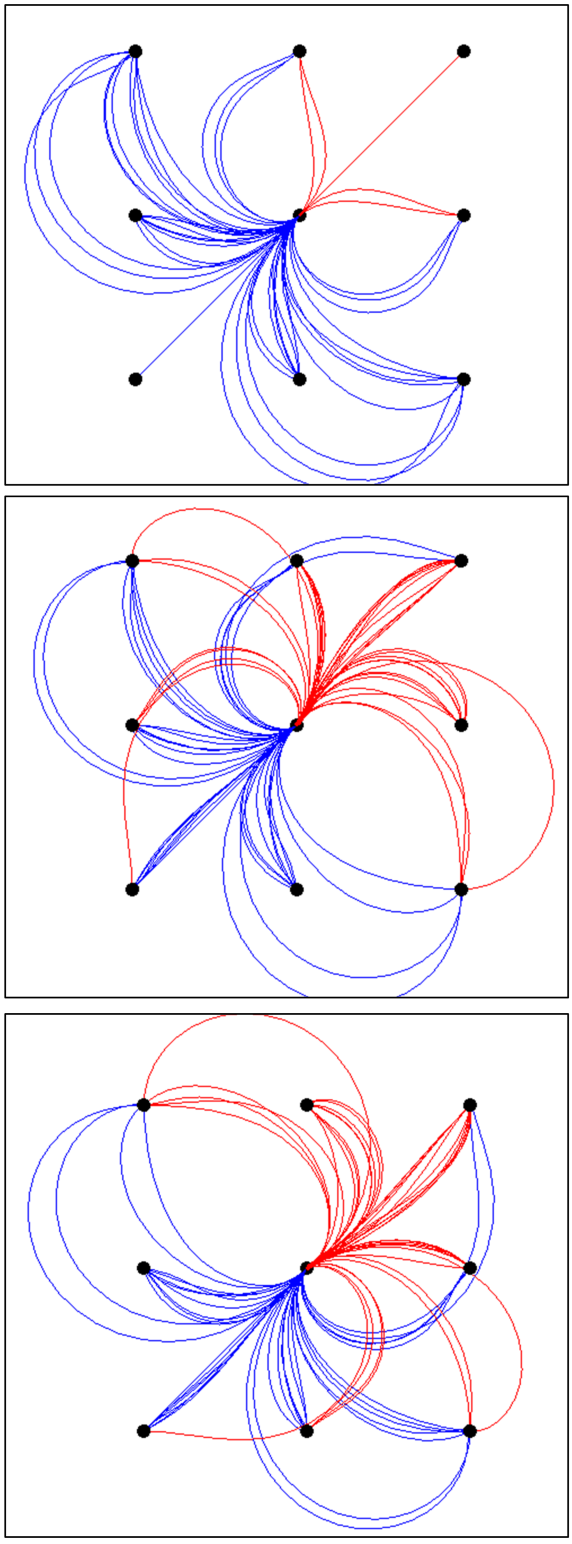
Para combinar la información del mapa con el tiempo que se tarda en realizar la trayectoria, es necesario consultar el grado de ocupación de las celdas que atraviesa. El coste de transición entre dos estados se define como:

$$coste(s, s') = peso_{celdasTrayectoria(s,s')} * tiempoTrayectoria(s, s') \tag{2}$$

El peso de una celda depende del estado en el que se encuentre. Como ya se ha definido en la sección 4, una celda puede estar en tres posibles estados: libre, ocupada o desconocida. A cada uno de estos estados se le asigna un peso, de acuerdo a la tabla 2.

Para combinar los pesos de varias celdas, se ha optado por hacer el promedio de los pesos de todas las celdas que forman parte del camino, pesado por el tiempo que permanece el robot en cada una de las celdas mientras realiza la trayectoria. De esta forma, la combinación entre los pesos de las celdas se realizaría de acuerdo a la ecuación siguiente:

$$c(s, s') = \left(\frac{\sum_{\alpha \in cells} N(\alpha) \cdot weight(\alpha)}{\sum_{\alpha \in cells} N(\alpha)} \right) \cdot timeTraj(s, s') \tag{3}$$



Máscara de trayectorias primitivas con velocidad lineal inicial 0 y velocidad angular inicial 0.

Máscara de trayectorias primitvas con velocidad lineal inicial 0 y velocidad angular inicial positiva.

Máscara de trayectorias primitvas con velocidad linal inicial 0 y velocidad angular inicial negativa.

Figura 10: Ejemplo de trayectorias primitivas de vecindad 1

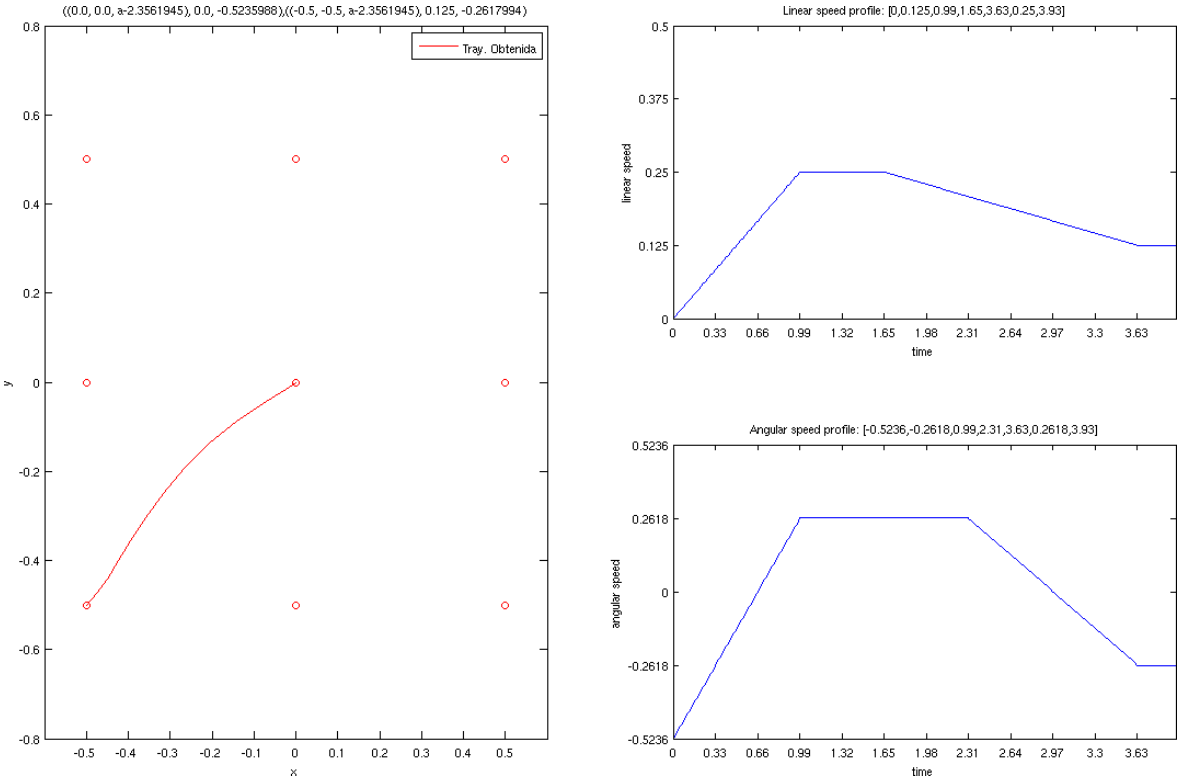


Figura 11: Perfiles de velocidad lineal y angular asociados a una trayectoria.

En las celdas que ocupa la trayectoria se incluyen, además de las celdas que atraviesa el centro, las celdas ocupadas por la huella del robot. La huella del robot es todo el espacio que ocupa en una posición y orientación determinadas. Al incluirla en el cálculo del coste asociado a una trayectoria evita que la solución calculada por el planificador pase por sitios demasiado estrechos para las dimensiones del robot, como se ilustra en la figura 12.

Para realizar este cálculo se define la huella del robot en base a un conjunto discreto de puntos, para ahorrar tiempo de computación de la huella. Para cada uno de los puntos muestreados de la trayectoria de la que se está calculando el coste, se centra la huella del robot en la posición y orientación correspondiente, y se anotan las celdas visitadas por el robot en ese punto. Se asume que en cada punto de la trayectoria el robot permanece t_c segundos. El término $N(\alpha)$ se obtiene a partir del número de veces que una celda fue visitada en esa trayectoria, multiplicado por el t_c. Este término representa el tiempo que el robot ha visitado la celda durante esa trayectoria.

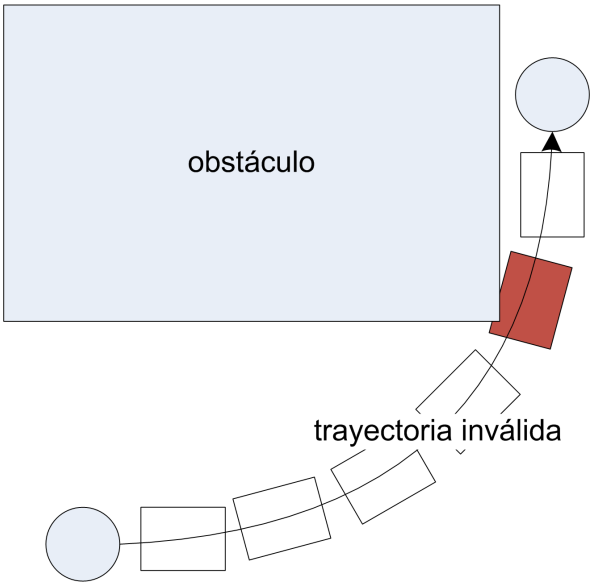


Figura 12: Trayectoria inviable considerando la huella del robot.

Utilizando esta función de coste de transición, el algoritmo de búsqueda minimiza el tiempo que se tarda en alcanzar el punto objetivo desde el estado inicial del robot, a la vez que evita los obstáculos presentes en el mapa, generando rutas que transitan por zonas libres.

5.4. Heurística

La heurística guía al algoritmo de búsqueda y prioriza los nodos que se van a expandir a continuación. Una buena heurística permite encontrar la solución al problema de una forma eficiente y muy rápida, minimizando el número de nodos que son expandidos por el algoritmo de búsqueda hasta su finalización y, con ello, el tiempo de computación necesario.

Típicamente, en los problemas de búsqueda de rutas en dos dimensiones, se utiliza la distancia euclídea como una buena heurística para estimar el coste de transitar desde el estado s al s'. Sin

embargo, el problema de la planificación de rutas para este proyecto se ha planteado en cinco dimensiones, con el fin de tratar las restricciones cinemáticas: posición en x, en y, orientación, velocidad lineal y velocidad angular del robot.

Para este tipo de problema, la distancia euclídea entre los nodos no es un buen estimador del coste de transición entre ellos, como se puede ver en la figura 13. Aunque es una heurística válida, es muy ineficiente, porque no tiene en cuenta nada más que las variables de posición del estado del robot, dejando al margen las dos velocidades y la orientación.

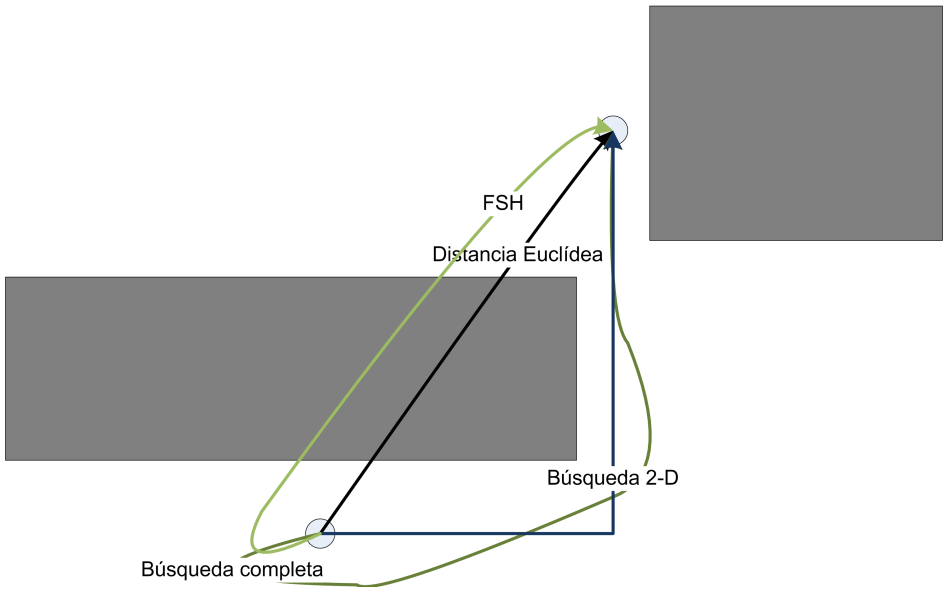


Figura 13: Comparativa de los caminos que representan diferentes formas de estimar el coste a la meta.

En este proyecto, se ha optado por utilizar dos heurísticas diferentes:

- **H2D**: Estima el coste de transitar del estado s al s' como el coste del camino que hay entre ambos, teniendo en cuenta el grado de ocupación de las celdas, pero sin tener en cuenta la orientación ni la velocidad lineal y angular de los estados.
- **FSH**: Estima el coste de transitar del estado s al s' como el coste de realizar el camino entre ambos, pero en este caso sin tener en cuenta el grado de ocupación de las celdas, y sí la orientación, la velocidad lineal y la velocidad angular de los estados del robot.

Para combinar los valores devueltos por las dos heurísticas utilizadas, se ha utilizado el criterio del máximo:

$$h(s, s') = max(h_{H2D}(s, s'), h_{FSH}(s, s')) \tag{4}$$

5.4.1. H2D

La heurística H2D representa el coste del camino que hay entre dos posiciones del mapa, teniendo en cuenta la información sobre el grado de ocupación de las celdas (figura 14). La forma más eficiente

de calcular el camino entre dos nodos definidos por su posición en x y en y, es utilizar un algoritmo de búsqueda simple, como el A*.

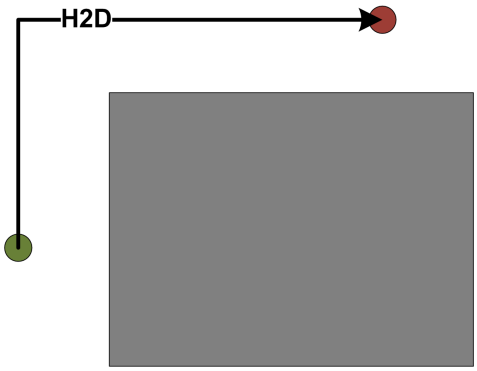


Figura 14: Ejemplo de heurística H2D

Para poder utilizar el A* y calcular la heurística H2D ha sido necesario definir una función de coste y una heurística propias para la obtención de este camino, teniendo en cuenta que cada uno de los nodos del espacio de búsqueda es, en este caso, una posición bidimensional. La resolución utilizada para discretizar el espacio de búsqueda tiene que ser necesariamente la misma que la definida para resolver el problema de la planificación de rutas, para que las posiciones utilizadas por H2D tengan posiciones coincidentes con los estados discretos del robot.

Ha sido necesario diseñar la función de coste de H2D siguiendo el mismo criterio que la función de coste del problema general, para que el valor de la heurística entre dos nodos fuera expresado en la misma magnitud que el coste real. En este caso, la función de coste de transitar entre dos posiciones se ha definido como:

$$coste_{H2D}(p, p') = pesoCeldas * tiempoTransicion_{H2D}(p, p') \tag{5}$$

Donde el tiempo de transición se ha definido como el tiempo de transitar en línea recta entre dos posiciones a la máxima velocidad lineal:

$$tiempoTransicion_{H2D}(p, p') = \frac{distanciaEuclidea(p, p')}{v_{max}} \tag{6}$$

En este caso, el peso de las celdas se calcula como el promedio de los pesos de las celdas correspondientes a la posición de inicio y la de fin. El tiempo de transición entre dos posiciones es el que se tarda en recorrer la distancia entre ambos estados a la máxima velocidad permitida por el robot.

La heurística H2D está a su vez dotada de una heurística muy simple que es utilizada por el A* para calcular el camino entre dos posiciones de un modo más eficiente, y así minimizar el tiempo de cálculo. Esta heurística se corresponde con la segunda componente de la función de coste definida anteriormente: el tiempo de transición entre dos posiciones:

$$h(p, p') = tiempoTransicion_{H2D}(p, p') \tag{7}$$

Para realizar este proceso de búsqueda, la vecindad utilizada ha sido la de un grid 8-conectado, esto es: cada posición tiene como vecinas las adyacentes en horizontal, en vertical, y en las diagonales.

5.4.2. FSH

La heurística FSH representa el coste del camino que hay que seguir para transitar entre dos estados del robot, sin tener en cuenta el grado de ocupación de las celdas del mapa, asumiendo que todo el espacio es libre (figura 5.4.2). En este caso, la función de coste de transición entre dos estados queda reducida al tiempo en completar la trayectoria que los conecta:

$$coste_{FSH}(s, s') = tiempoTransicion(s, s') \tag{8}$$

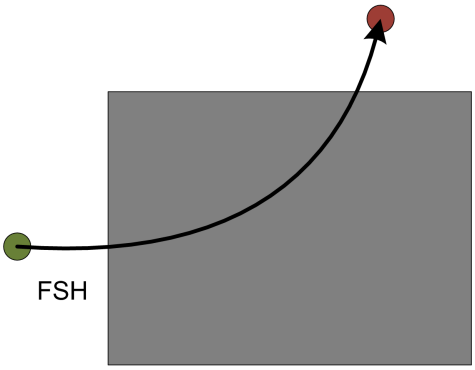


Figura 15: Ejemplo de heurística FSH

La heurística FSH es computacionalmente muy costosa, en tanto que supone realizar una búsqueda de la misma dimensionalidad que el problema que se pretende resolver inicialmente. Sin embargo, puesto que FSH asume el espacio siempre libre de obstáculos, los valores de heurística pueden ser calculados *offline*, y almacenados en una tabla para ser utilizados en el proceso de planificación. De esta forma, el rendimiento del algoritmo de búsqueda no se ve penalizado, al recuperar el valor a partir de una consulta a la tabla. Esta tabla se conoce con el nombre de Heuristic Look-Up Table, o HLUT [5].

Node1	Value1
Node2	Value2
Node3	Value3
...	
NodeN	ValueN

Figura 16: Tabla HLUT

El proceso de rellenado de la tabla HLUT se puede optimizar teniendo en cuenta:

- Los nodos introducidos en la HLUT deben tener un valor de heurística representativo, entendiendo por representativo que sea muy diferente al valor de H2D en el entorno tambien libre de obstáculos. Para medir objetivamente la representatividad se define el *trim ratio*, como $trim = H2D/FSH$. Sólo los nodos por debajo de un determinado valor de *trim* son introducidos en la HLUT.

- Todos los nodos por debajo de un umbral de distancia son siempre almacenados en la tabla.
- Como la retícula de estados es regular, se pueden aprovechar las simetrías para almacenar sólo los costes de las trayectorias que no se pueden expresar como rotación, traslación y reflexión de otras ya existentes, y de esta forma limitar el número de entradas que tiene la tabla. Estas operaciones se ilustran en la figura 17.

La introducción de valores en la HLUT se realiza en dos pasos consecutivos: para cada nodo de inicio, en primer lugar se ejecuta una búsqueda de Dijkstra especificando el nodo de inicio, pero sin un nodo de finalización definido. Cuando en la lista de abiertos no queda ningún nodo por debajo de un umbral de distancia determinado, significa que todos ellos han sido explorados ya, se encuentran en la lista de cerrados, y se pueden añadir a la HLUT independientemente del coste asociado a ellos.

En este punto, para un determinado nodo de inicio, en la HLUT se encuentran ya todos los nodos que están conectados con él y están a menos de una distancia límite. En algunos casos esta información no es una heurística suficiente, porque puede haber nodos más alejados que tengan un coste de transición hasta el destino muy diferente del que proporciona H2D.

Algoritmo 9 HORIZON

```
introducir todos los vecinos HORIZON de los nodos frontera en la trimList
while size(trimList) > 0 y min(trimList) > umbral do
  query = min(trimList)
  if HLUT no contiene query and query no ha sido explorada then
    ejecutar A* con query actual
    for all n ∈ CLOSED do
      if n no explorado previamente then
        calcular trimRatio del n
        marcar n y todos sus equivalentes como explorados
        if trimRatio < umbralTrim y n no contenido en la HLUT then
          añadir n a la HLUT
          generar todos los vecinos HORIZON de n, y añadirlos a la trimList con trim(n)
        end if
      end if
    end for
  end if
end while
```

Para introducir en la HLUT los nodos con un *trim* inferior a un umbral y que están fuera del radio de distancia definido anteriormente, se utiliza el algoritmo HORIZON [5], cuyo pseudocódigo se encuentra en el algoritmo 5.4.2, y que tiene las siguientes características:

- Su ejecución comienza partiendo de los nodos frontera de la zona de inclusión obligatoria en la HLUT definida por la distancia umbral.
- Es un algoritmo de priorización. Guarda en una lista todas las consultas que se van a realizar al A* para incluir entradas relevantes en la tabla. Esta lista contiene inicialmente los nodos vecinos

HORIZON de los estados frontera. El orden de esta lista depende del *trim ratio* asociado a las consultas.

- Para cada nodo que se añade a la HLUT, se introducen como nuevas consultas en la lista de priorización sus vecinos HORIZON, heredando como valor *trim* para la priorización el valor de *trim* del nodo que se acaba de introducir en la tabla.
- Finaliza cuando no queda ninguna consulta por realizar por debajo del *trim* umbral.

Los vecinos HORIZON para un estado determinado son aquellos estados adyacentes en horizontal, vertical o diagonal, con la misma velocidad lineal y angular que el estado de partida.

Conforme se ejecuta este algoritmo, la exploración va creciendo de forma radial alcanzando distancias cada vez mayores, pero de una forma controlada, ya que sólo son generados los vecinos HORIZON de los nodos que cumplen todas las restricciones para ser introducidos en la HLUT.

Cuando se realiza una consulta al A* utilizando este algoritmo, se puede aprovechar toda la información generada, y explorar la lista de cerrados para añadir todos los nodos a la HLUT que se encuentren en esa lista y cumplan las restricciones necesarias. De esta forma el algoritmo finaliza mucho más rápido al ejecutarse un número de consultas mucho menor, ya que la información almacenada en la lista de cerrados no tiene que volver a generarse posteriormente.

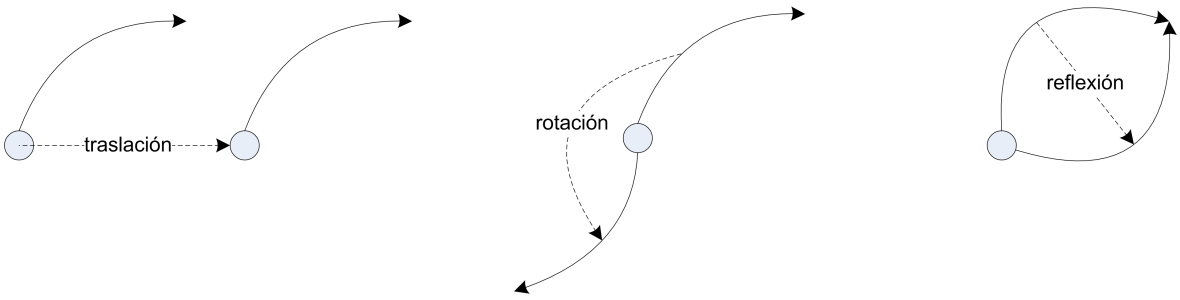


Figura 17: Transformaciones utilizadas a nivel de trayectoria.

Como se puede ver en la figura 18, H2D y FSH representan dos valores complemetarios, pues mientras que H2D estima el coste de transición en base a los obstáculos, FSH estima el coste en base a las restricciones cinemáticas del robot utilizado.

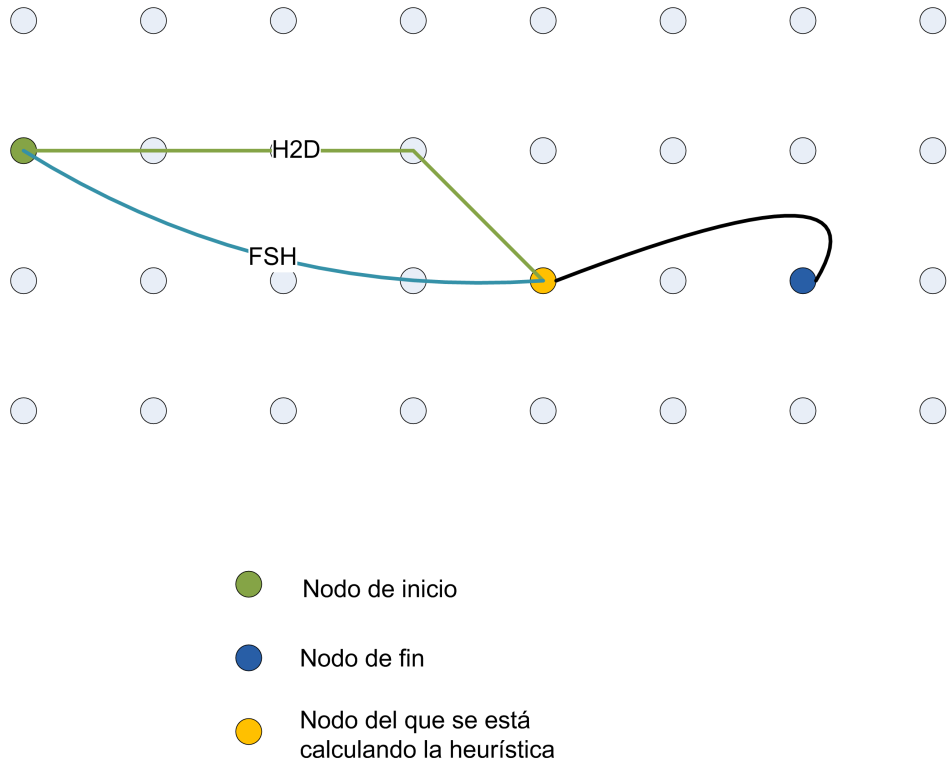


Figura 18: Cálculo de la heurística para un estado del robot en el planificador.

6. Control del robot

El módulo de control se ha diseñado de tal forma que el robot esté inactivo mientras no haya una solución válida disponible en el planificador. Una vez que existe una ruta planificada, se envía un comando de movimiento cada t_c.

Los comandos de movimiento se extraen de la solución que envía el planificador. Dicha solución está formada por un conjunto de estados por los que el robot tiene que pasar, y cada pareja de estados está conectada mediante una trayectoria. Como se ha visto en la sección 5.2, cada una de ellas lleva asociado un perfil de velocidad lineal y de velocidad angular, que definen la evolución de ambos valores a lo largo del tiempo que dura dicha trayectoria.

El módulo de control comienza la exploración de la solución en el estado inicial, y envía cada t_c un comando de movimiento a los motores del robot, formado por una pareja de valores de velocidad: (v_t, ω_t). Estos valores se extraen de los perfiles de velocidad asociados a las distintas trayectorias que conectan los estados del robot devueltos por el planificador como solución. Las órdenes de control no tienen necesariamente que acabar en un tiempo múltiplo de t_c, por lo que un determinado comando de movimiento puede tener un tiempo de aplicación mayor en algunos casos. Cuando esta situación se produce, el controlador envía un nuevo comando de movimiento tras esperar el tiempo correspondiente, en lugar del tiempo de control usual.

Este módulo también detecta cuándo se produce una situación en la cual es necesario replanificar.

En concreto:

- El estado del robot no se corresponde con el que debería estar según la ruta planificada, al acabar de realizar una trayectoria primitiva. Este tipo de situaciones se generan cuando el robot se desvía de la trayectoria original, por pequeños errores en la aplicación de los comandos de movimiento que se acumulan a lo largo del tiempo.
- Se detecta un cambio de costes en la ruta planificada, y es necesario actualizar la solución para asegurarse de que el camino planificado sigue siendo realizable, y no se han descubierto otras rutas mejores.
- La solución generada tiene un determinado nivel de suboptimalidad ($\epsilon > 1$) y es posible mejorarla.

En estos casos el módulo de control envía una orden al planificador para que recalcule la ruta desde el estado actual hasta el objetivo. Cuando se recibe la nueva solución, continúa enviando comandos de movimiento. Cuando el punto objetivo es alcanzado, este módulo envía una orden de finalización al resto de los módulos en ejecución.

7. Resultados, validación y conclusiones

En esta sección se muestran las capacidades del sistema de control inteligente implementado para este proyecto, donde las restricciones cinemáticas se han aplicado para el robot Pioneer3-DX [10], y el tamaño de celda del mapa se ha definido como 0.5 x 0.5 m.

Las pruebas sobre los diferentes módulos implementados se han llevado a cabo utilizando la herramienta de simulación Player/Stage [4, 11, 15], ya que permite trabajar en un entorno controlado y ver el comportamiento que tiene el sistema de una forma teórica.

La validación de este proyecto se ha llevado a cabo en tres partes:

- Módulo de mapeado: Se han realizado dos tipos de pruebas, sobre el mapa local y sobre el mapa global:
  - Mapa local: Se ha comprobado la actualización del mapa local realizando navegación aleatoria por varios entornos con diferente densidad de obstáculos: baja, media y elevada. Para cada uno de los entornos se ha comprobado la adecuación del tamaño de celda a los diferentes obstáculos, y se ha comprobado que este valor era suficiente para detectar con precisión los obstáculos en un entorno de interiores, sin perder fluidez en el proceso de actualización.
  - Mapa global: Utilizando los mismos entornos que en el caso anterior, se han realizado dos tipos de pruebas diferentes sobre cada uno de ellos: en primer lugar inicializando el mapa con información sobre el entorno, para comprobar que el proceso de actualización no era inconsistente con la información correcta y precargada de antemano; y un segundo tipo de prueba partiendo sin ningún tipo de información sobre el grado de ocupación de las celdas del mapa, para comprobar que se iba haciendo una reconstrucción correcta de toda la zona.
- Módulo de planificación: Para realizar las pruebas sobre este módulo se ha partido de un conjunto de entornos con distintos niveles de complejidad: con obstáculos muy grandes, distancias muy

elevadas y un elevado nivel de complejidad en la planificación. Para todos ellos se ha podido comprobar que el planificador era capaz de construir correctamente una solución entre el inicio y el fin, utilizando distintos niveles de sub-optimalidad y que ante la aparición o desaparición de obstáculos, el planificador se comportaba según lo esperado.

- Módulo de control: Las pruebas sobre este módulo de la aplicación fueron las más complejas de realizar, debido a su dependencia del funcionamiento tanto del módulo de mapeado como de planificación. Para reducir esta dependencia, se optó por realizar las pruebas con la información del mapa obtenida de antemano, de forma que no se tuviesen zonas de incertidumbre en el entorno para realizar la planificación, y comprobar el comportamiento del módulo en cuanto a la implementación de las sucesivas trayectorias primitivas, la detección de necesidades de replanificación y la integración con el resto de módulos de la aplicación.

En las pruebas finales del sistema, realizadas sobre el entorno simulado, se ha observado que, aunque el controlador realiza el envío de las órdenes de control correctamente, existe una diferencia notable en algunas de ellas en el resultado obtenido al aplicar la orden de control en la práctica, respecto a la teoría. Al no ser el mismo estado el alcanzado que el obtenido, a menudo el controlador detecta necesidad de replanificar, si bien esto no sería necesario si la orden de control estuviese ejecutada con un resultado más similar a la orden teórica.

En la figura 19 se observa el resultado de una planificación realizada en muy poco tiempo y con un determinado nivel de sub-optimalidad (ϵ), y el refinamiento de la ruta obtenida en sucesivas iteraciones, rebajando el nivel de sub-optimalidad en cada uno de los pasos. La situación inicial del robot en ambos casos es la misma: se encuentra en una habitación orientado hacia una de las paredes, y el objetivo es alcanzar la estancia anexa.

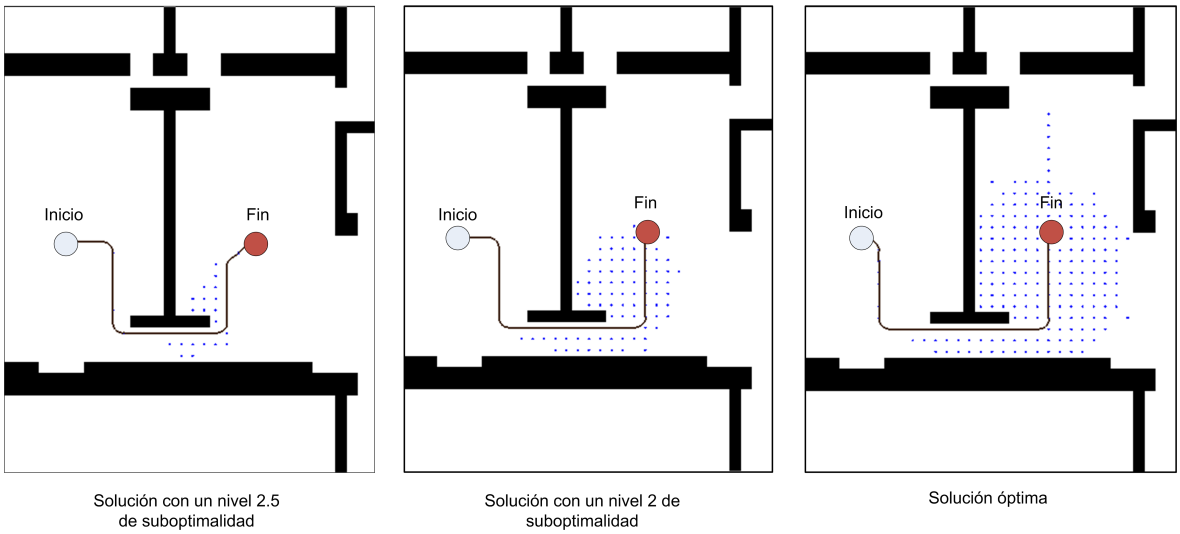


Figura 19: Evolución de la solución publicada por el planificador

En la figura 20 se muestra el comportamiento del planificador cuando hay un obstáculo que aparece después de haber planificado la ruta. En este caso se comparan la solución previa a la aparición del

obstáculo, y la solución posterior. Finalmente, en la figura 21 se puede ver el resultado del proceso de planificación para un ejemplo más complejo.

En `http://www.gsi.dec.usc.es/mucientes/VACCA/videos` se puede descargar un vídeo con un ejemplo de planificación realizada en este sistema.

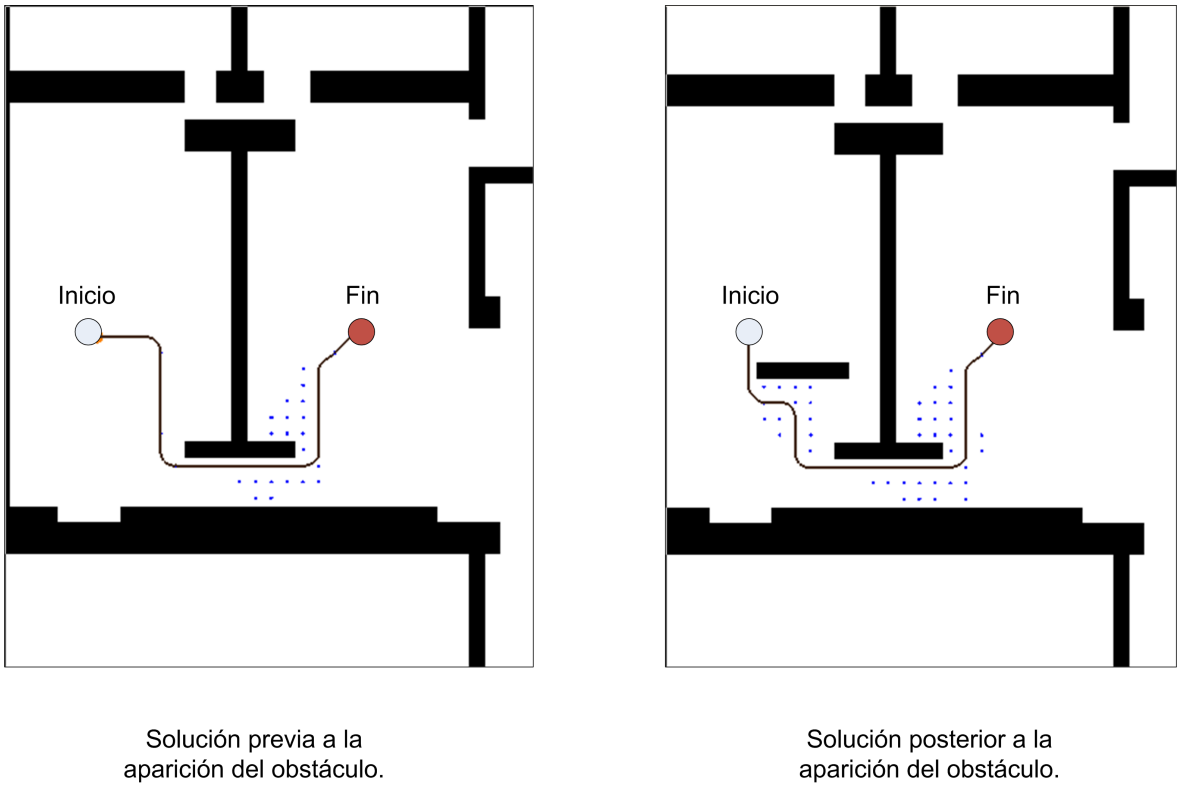


Figura 20: Evolución de la solución cuando aparece un obstáculo nuevo

7.1. Conclusiones

El objetivo principal de este Trabajo de Fin de Máster era la realización de un software de planificación y control para robótica móvil que integrase las restricciones de movimiento del robot utilizado en las rutas obtenidas, de forma que el robot pudiese ser capaz de navegar de forma autónoma tanto en entornos conocidos como desconocidos, realizando un mapeo constante de la zona de movimiento partiendo de la información recogida por los sensores.

En este documento se realiza una descripción pormenorizada del trabajo que se ha llevado a cabo para resolver el problema planteado. A continuación se realiza un breve resumen del trbajo realizado:

- Se ha implementado un módulo de gestión de mapas, capaz de cargar información de ficheros de imágenes para poder planificar en base a ella sin tener que partir de un total desconocimiento del entorno del robot. Este módulo permite además la actualización en tiempo real del mapa de la zona que almacena el robot basándose en la información recogida por los sensores.

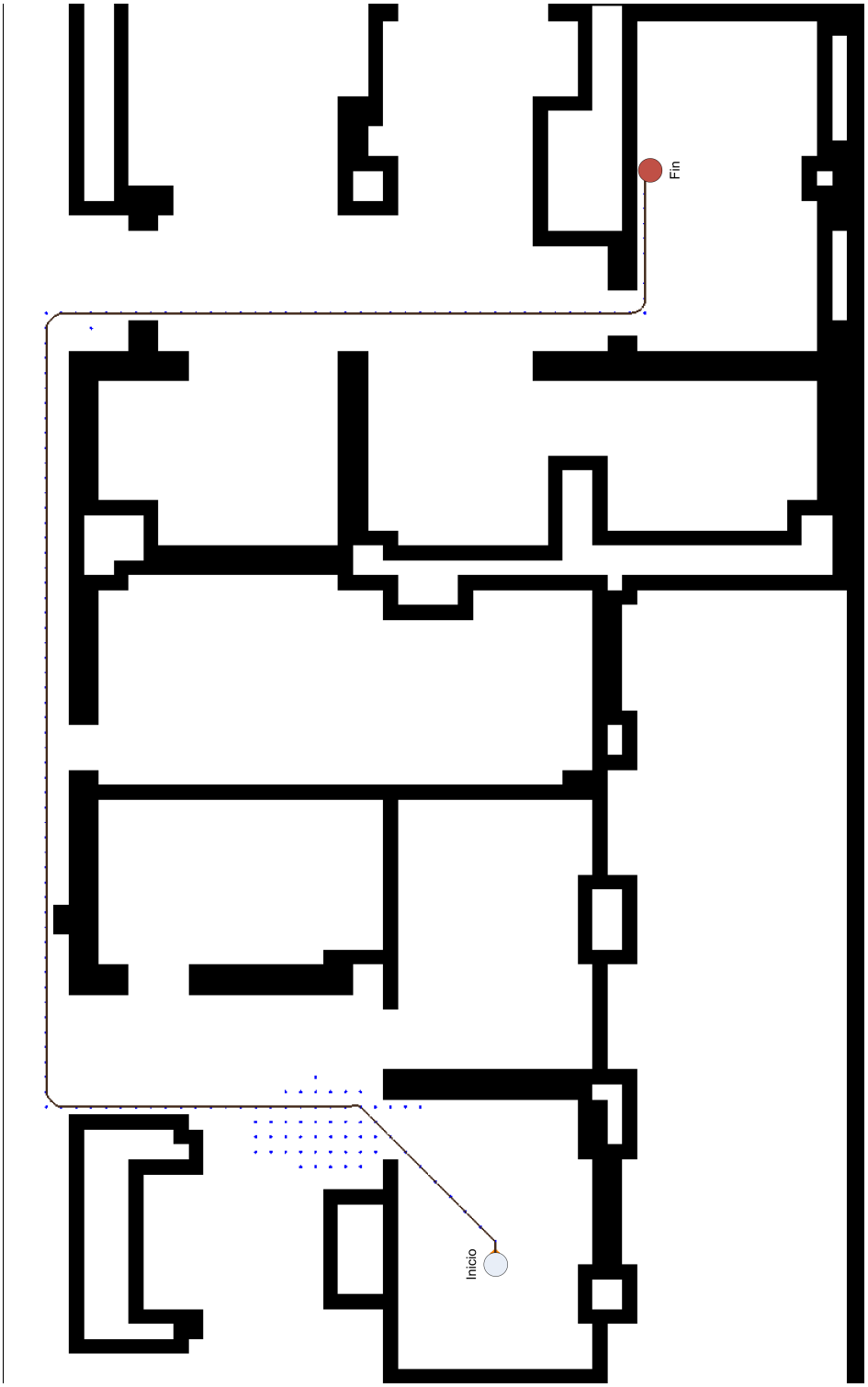


Figura 21: Solución a un problema de planificación complejo

- El módulo de planificación implementado es capaz de obtener soluciones entre el estado actual del robot y el objetivo basándose en la información contenida en el módulo de gestión de mapas, y ajustando el nivel de optimalidad de la solución al tiempo disponible para su cálculo. Los reajustes realizados sobre soluciones previas utilizan la información de cálculos anteriores y obtienen nuevas soluciones partiendo de las anteriores en un tiempo muy corto.

- El módulo de control utiliza la solución obtenida del planificador para extraer los comandos de movimiento que deben ser enviados a los motores del robot. En paralelo a este proceso, se realizan también comprobaciones sobre el mapa y al estado actual del robot para detectar posibles necesidades de replanificación que alterasen el proceso de control, debido a la necesidad de calcular una nueva solución desde el estado actual del robot.

- Se ha implementado además un módulo de generación de trayectorias primitivas, que es la base del proceso de planificación para la generación de soluciones que cumplan con las restricciones de movimiento del robot, y otro módulo de generación de la tabla HLUT para utilizar en el cálculo de la heurística FSH.

- Se ha desarrollado un módulo de visualización que permite ver de forma gráfica y en tiempo real el estado del mapa, de las expansiones realizadas por el algoritmo de búsqueda del planificador, de la solución obtenida y el estado actual del robot; y otro módulo de visualización diferente para explorar las trayectorias primitivas en detalle.

7.2. Trabajo futuro

Como trabajo futuro se plantean las siguientes líneas de mejora:

- Generación eficiente de las trayectorias primitivas: En este trabajo, la técnica de generación de trayectorias primitivas se ha basado en una exploración en bruto de todas las posibles acciones de control. Este punto se puede mejorar de forma notable utilizando algún método inteligente para limitar el número de trayectorias que se exploran.

- Multiresolución: Se puede reducir el número de nodos que se generan para formar el espacio de búsqueda, si se ajusta la resolución a la que son generados dependiendo de la densidad de obstáculos del mapa y la importancia de algunas zonas del mapa para realizar la planificación.

- Aprendizaje de modelos de movimiento de robots, aplicando estas técnicas a la robótica submarina.

Referencias

[1] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:224–241, 1992.

[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, GeorgeA Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*, chapter H, pages 521–546. MIT Press, Cambridge, MA, June 2005.

[3] D. Ferguson and M. Likhachev. Efficiently using cost maps for planning complex maneuvers. *Lab Papers (GRASP)*, page 20, 2008.

[4] Brian Gerkey et al. *The Player Robot Device Interface*. `http://playerstage.sourceforge.net/doc/Player-2.1.0/player/`, September 2005.

[5] R. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3375–3380, 2006.

[6] S. M. LaValle. *Planning Algorithms*. Cambridge Univ Pr, 2006.

[7] P. Lester. A* pathfinding for beginners. *Osoitteessa www.policyalmanac.org/games/aStarTutorial.htm*, 18:2005, 2005.

[8] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933, 2009.

[9] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, 2005.

[10] MobileRobots.inc, `www.mobilerobots.com/researchrobots/researchrobots/pioneerp3dx.aspx`. *The Pioneer 3-DX Robot*, 2010.

[11] Jennifer Owen. *The Player/Stage Tutorial 2nd Edition*. `www-users.cs.york.ac.uk/~jowen/player/playerstage-tutorial-manual.pdf`, April 2010.

[12] M. Pivtoraiko, R.A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[13] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*, chapter 6, pages 149–169. MIT Press, 2008.

[14] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*, chapter 9, pages 281–299. MIT Press, 2008.

[15] Richard Vaughan et al. *The Stage Robot Simulator*. `http://playerstage.sourceforge.net/doc/Stage-3.2.1/`, October 2009.