UNIVERSIDAD CARLOS III DE MADRID

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TESIS DE MÁSTER

ALGORITMOS DE PLANIFICACIÓN DE TRAYECTORIAS BASADOS EN FAST MARCHING SQUARE

Autor: Jose Pardeiro Blanco

Tutor: Dr. Ramón Barber Castaño

Director: Javier V. Gómez González

MÁSTER OFICIAL EN

ROBÓTICA Y AUTOMATIZACIÓN

LEGANÉS, MADRID

FEBRERO 2015

UNIVERSIDAD CARLOS III DE MADRID
MASTER OFICIAL EN ROBÓTICA Y AUTOMATIZACIÓN

El tribunal aprueba la tesis de Master titulada "**Algoritmos de planificación de trayectorias basados en Fast Marching Square**" realizada por **Jose Pardeiro Blanco**

Fecha: Febrero 2015

Tribunal:

Dr. Luis Moreno Lorente

Dr. Santiago Garrido Bullón

Dr. Cristina Castejón Sisamón

A toda la gente que ha confiado en mí y me ha dado la oportunidad de estar aquí.

Índice general

Índice de tablas

Índice de figuras

Agradecimientos

Quiero dedicarle esta tesis a toda la gente que me ha apoyado y ayudado en todo este tiempo.

Antes de nada, debo agradecerle a Javier V. Gómez que hace ya más de dos años tomara el rol de mentor y me abriera las puertas a la robótica.

También dar gracias a David Álvarez y Domingo Esteban, sin vuestra ayuda y apoyo esta tesis no hubiera llegado tan lejos.

Por supuesto, gracias a mis compañeros y amigos: Rubén Romero, Jorge Ortega, Isaac Rivero, Daniel Rodríguez, Víctor Aranda y Pablo Fernández. He aprendido mucho de vosotros y habéis sido un punto de apoyo siempre.

A Jorge Villagrá, mi otro mentor, del que llevo aprendiendo cada día desde hace más de un año.

Agradecerle también a Santiago Garrido, Ramón Barber y Jonathan Crespo el apoyo para sacar esta tesis adelante.

Y finalmente, a mis padres, mi hermana y mi pareja. Gracias por aceptar mi ausencia durante todo este tiempo.

Resumen

La planificación de trayectorias se considera un problema resuelto para multitud de investigadores. No obstante, aunque existen numerosos enfoques, a día de hoy no se ha desarrollado un método de planificación ideal. Un método de planificación de trayectorias se considera ideal cuando reúne características tales como baja carga computacional, fiabilidad, robustez, trayectorias suaves, o seguridad, tanto en entornos ideales como reales en los que el modelado resulta más complejo.

Esta Tesis de Máster busca resolver algunos de los problemas que aparecen para encontrar un método de planificación que cumpla todas las necesidades. Para ello se proponen métodos basados en Fast Marching Square que disminuyen su carga computacional y aumentan su adaptación a los entornos. Posteriormente los algoritmos son implementados sobre un robot real, concretamente un robot Turtlebot, con el fin de demostrar su adaptabilidad a entornos reales.

Palabras clave: Planificación de trayectorias, Fast Marching, Fast Marching Square, Fast Marching Square Star, Fast Marching Square Directional, Turtlebot.

Abstract

The path planning problem is considered solved for many researchers. However, despite the many different approaches, an ideal planning algorithm has not been proposed yet. A path planning method is considered ideal if it satisfies characteristics such as low computational cost, reliability, robustness, smooth paths, or safety, both in ideal and real environments whose modeling is quite complex.

This Master's Thesis aims to solve some of the problems that appear when pursuing an ideal planning method. Some methods based on Fast Marching Square are proposed to decrease computational cost and increase their adaptability to real environments. Subsequently the algorithms are implemented on a real robot, namely the Turtlebot, in order to demonstrate its adaptability to real environments.

Keywords: Path planning, Fast Marching, Fast Marching Square, Fast Marching Square Star, Fast Marching Square Directional, Turtlebot.

Capítulo 1

Introducción

El objetivo de este trabajo consiste en detallar nuevas aproximaciones a los métodos de planificación de trayectorias basados en Fast Marching Square que mejoren al original y demostrar que los métodos son válidos en entornos reales.

La planificación de trayectorias es un problema que actualmente se considera resuelto por parte de la comunidad investigadora, argumentando que los métodos actuales, sin ser perfectos, son suficientes como para resolver la mayoría de problemas relacionados con la robótica. No obstante, a pesar de la multitud de métodos existentes, aún no ha sido encontrada la solución perfecta y completamente eficiente en sistemas reales y que reúna todas las características deseadas para ser ideal: suavidad, eficiencia, bajo coste computacional, seguridad, etc.

Por ese motivo, actualmente sigue siendo un campo en constante evolución en el que la comunidad investigadora continúa haciendo avances para proponer soluciones más eficientes, rápidas, orientadas a propósitos más generales y viables en sistemas reales. Pese a ello, una cantidad importante de

aportaciones se enfocan a una aplicación o una plataforma concretos, dificultando su adaptación a otras situaciones o plataformas por lo que no pueden ser considerados métodos de propósito general.

1.1. Contexto y motivación

Como se ha mencionado antes, un buen método de planificación de trayectorias ha de ser capaz de resolver multitud de problemas, independientemente del entorno, las situaciones o el sistema en el que se aplica ya que no es óptimo variar el planificador en función de la situación o la plataforma. Esta variación conlleva que el sistema sea capaz de detectar las situaciones y variar el planificador de forma transparente, lo cual no solo añade complejidad al sistema sino un tiempo de ejecución que puede ser crítico. Además, en muchas ocasiones los métodos desarrollados solo contemplan el marco teórico, sin tener en cuenta las complicaciones que surgen al adaptar los algoritmos a un entorno real.

Dado este contexto la tesis se enfoca en la planificación de trayectorias utilizando como base el método Fast Marching Square (FM^2). Este método es adaptable a numerosas situaciones por lo que es utilizado como punto de partida para desarrollar algoritmos nuevos que mantengan su potencia mejorando además características como la eficiencia o la adaptación de la trayectoria al entorno. Dichos algoritmos son embarcados de forma independiente de la plataforma y probados en un robot para comprobar así la potencia del algoritmo sobre un robot real.

Como se verá en los siguientes capítulos, FM^2 y sus variaciones pueden ser opciones a tener en cuenta como algoritmos generales e independientes de la plataforma.

1.2. Estructura del documento

En el siguiente capítulo se detallará el estado del arte en el campo que ocupa la tesis y enfocado a los problemas que se busca resolver. En el capítulo 3 se describe el algoritmo FM^2, que será utilizado como base en los siguientes capítulos. Tras eso, en los capítulos 4 y 5 se desarrollan dos nuevas aproximaciones a la planificación de trayectorias basadas en FM^2, mostrando los resultados y comparativas entre ellos.

A continuación, en el capítulo 6 se realiza una implementación en un sistema real, concretamente un Turtlebot, y se muestran los resultados tanto en simulación como sobre el sistema real.

Finalmente, las conclusiones y el trabajo futuro de la tesis se encuentran detallados en la sección 7.

Capítulo 2

Estado del arte en planificación de trayectorias en robótica

La planificación de trayectorias sigue siendo un campo en activo en el que siguen apareciendo nuevas aproximaciones y soluciones a los problemas que surgen. Actualmente ninguno de los algoritmos es capaz de satisfacer todos los requisitos necesarios: baja carga computacional, fiabilidad, robustez, trayectorias suaves, soluciones óptimas, seguridad, etc.

Los algoritmos que proporcionan trayectorias con baja carga computacional suelen generar trayectorias poco suaves, por lo que un proceso de suavizado es necesario después. Por otro lado, los algoritmos que proporcionan soluciones suaves suelen ser rápidos en problemas de dos dimensiones (y según el algoritmo se podría llegar a tres) pero su complejidad aumenta a medida que lo hace la dimensionalidad del problema, y por tanto el coste computacional.

2.1. Clasificación entre métodos de planificación de trayectorias

Existe una amplia literatura referente a la planificación de trayectorias. Esta tesis se encuentra enfocada al uso de métodos geométricos, es decir, métodos que únicamente tienen en cuenta la geometría del espacio sobre el que se va a realizar la planificación, obviando cualquier otro elemento externo como pueda ser el modelo cinemático o dinámico del robot. No obstante también existen métodos de planificación que incluyen restricciones cinemáticas e incluso dinámicas.

Actualmente no se ha alcanzado un acuerdo en la forma de clasificar los métodos de planificación. La causa principal es que existen métodos utilizados en multitud de contextos por lo que establecer las fronteras es complejo. Basada en la clasificación que propone LaValle (LaValle, 2011)), los métodos geométricos se pueden dividir en los siguientes grupos:

- *Métodos combinatorios*: Se define como el conjunto de algoritmos que construyen estructuras de datos que capturan toda la información necesaria para la planificación de trayectorias (Berg, Kreveld, Overmars, y Schwarzkopf, 2008; LaValle, 2006). Uno de los métodos más habituales en este grupo son los basados en *mapas de ruta*, los cuales buscan encontrar la ruta más corta uniendo tramos de un conjunto de rutas precalculadas.

  Dentro de los métodos basados en mapas de rutas se puede introducir un método muy extendido basado en *grafos de visibilidad* (Asano, Asano, Guibas, Hershberger, y Imai, 1986; Ghosh y Mount, 1991). Los grafos de visibilidad de un conjunto de obstáculos poligonales que no interseccionan entre sí en el plano es un grafo sin dirección. Los vértices del grafo son los vértices de los obstáculos y los bordes son pares de vértices tales que la línea que une ambos no intersecta con ningún obstáculo. Aunque el enfoque es muy intuitivo, como se muestra en la

figura 2.1, los resultados al aplicarlo en entornos de tres dimensiones o más dejan de ser óptimos.

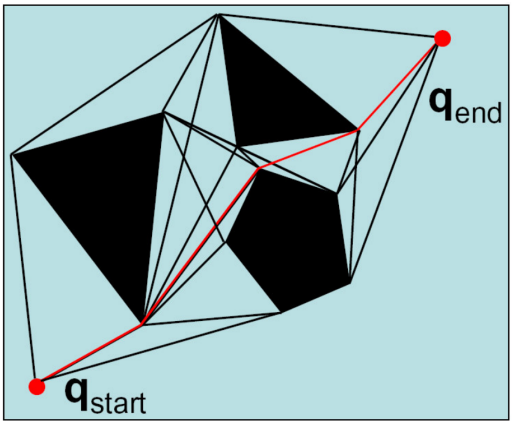


Figura 2.1: *Ejemplo de grafo de visibilidad. Imagen cortesía de 'Robot Motion Planning' de J.C. Latombe.*

Otro enfoque habitual consiste en generar la trayectoria utilizando la *triangulación Delaunay* del entorno (Delaunay, 1934). Este es uno de los algoritmos de triangulación más empleados porque minimiza la suma de ángulos de todos los triángulos obtenidos en la triangulación. Aunque la complejidad computacional del problema es $O(n\ log(n))$ en espacios de tres dimensiones para n puntos en superficies suaves (Attali, Boissonnat, y Lieutier, 2003), el principal inconveniente es que la triangulación tiene varias soluciones por lo que es posible que se generen trayectorias poco óptimas.

- *Métodos basados en muestreo*: Este tipo de métodos realizan búsquedas incrementales en el espacio para obtener una solución utilizando un algoritmo de detección de colisiones (LaValle, 2006). Los métodos más habituales en este campo están basados en *Rapidly-exploring Random Trees* (RRTs). Es un método basado en la extensión de ramas dentro de un mapa, equivalentes a las ramas de un árbol, las cuales se generan

aleatoriamente desde el punto inicial de la trayectoria. RRT es un método de planificación muy rápido, y además uno de los más utilizados actualmente.

Otro método muy común son los basados en *Probabilistic Roadmap* (PRMs). La idea principal de PRM es tomar muestras aleatorias del espacio de configuración del robot, en forma de grafo, comprobar cuales se encuentran dentro del espacio libre, y usar un planificador local para tratar de conectar cada configuración a otras configuraciones cercanas. La principal ventaja sobre RRT es que los grafos que se generan son independientes a los puntos de inicio y final, por lo que se pueden utilizar para generar distintas trayectorias sobre un mismo conjunto de configuraciones cambiando los puntos de inicio y final en el planificador local.

El mayor inconveniente de este tipo de métodos es que se tratan de métodos estocásticos, por lo que generalmente los grafos generados no reúnen características básicas como ser óptimos, seguros o suaves. Por ese motivo existen muchas aproximaciones diferentes que modifican estos métodos para hacerlos más útiles, como se verá en la sección 2.2.

- *Métodos basados en espacios discretos*: Este método puede ser considerado parte de los métodos basados en muestreo, pero existen diferencias entre ambos suficientes como para considerarlos métodos separados. Se caracterizan por discretizar el espacio antes de planificar, ya sea en grafos o en una rejilla compuesta por celdas, y asignar una conectividad entre elementos. Cada algoritmo establece un coste, basado en sus propios criterios, para viajar entre los nodos o las celdas. Dentro de los métodos basados en grafos se pueden enmarcar algoritmos de búsqueda clásicos, como *Dijkstra* (Dijkstra, 1959) o *A** (Hart, Nilsson, y Raphael, 1968)

  Las formas más habituales de representar las rejillas son como rectángulos o triángulos. La discretización puede provocar una pérdida

de precisión pero se puede solventar seleccionando un tamaño de celda acorde a la aplicación. Cada celda de la rejilla es un elemento conectado con sus vecinos (dependiendo del algoritmo la conectividad será de 4 u 8 en 2D) y posee un coste para llegar a él desde sus vecinos. Una vez que los costes se han asignado se aplica el mecanismo de búsqueda del algoritmo para encontrar el camino que une el punto de inicio con el final con el menor coste posible.

En este campo se encuentran los métodos basados en *potential field* (Hwang y Ahuja, 1992). En estos algoritmos el robot se trata como una carga eléctrica que se mueve por un campo de potencial, representando los obstáculos como cargas del mismo tipo que las del robot, y el destino como una carga opuesta. De este modo las cargas de los obstáculos repelen al robot mientras que la del objetivo lo atrae. El principal inconveniente de esta aproximación es que el sistema es propenso a generar mínimos locales y oscilaciones.

Finalmente, el método *Fast Marching Method* (FMM) (Sethian, 1999), en el cuál se basa esta tesis, utiliza un entorno discretizado como una malla (sea triangular o cuadrada). Los costes de llegada a cada celda se asignan mediante como el tiempo de llegada de una onda a cada celda, y a partir de ahí se genera la trayectoria. Una descripción más desarrollada del algoritmo se presenta en la sección 3.

En la figura 2.2 se presenta un esquema de la clasificación realizada. La clasificación utilizada, si bien está inspirada en la de LaValle, considera los métodos discretos como métodos independientes a los basados en muestreo. Dada la gran cantidad de métodos de planificación que existen, en la clasificación se presentan únicamente los básicos.

En las siguientes secciones del estado del arte se comentarán los avances más recientes en planificación de trayectorias.

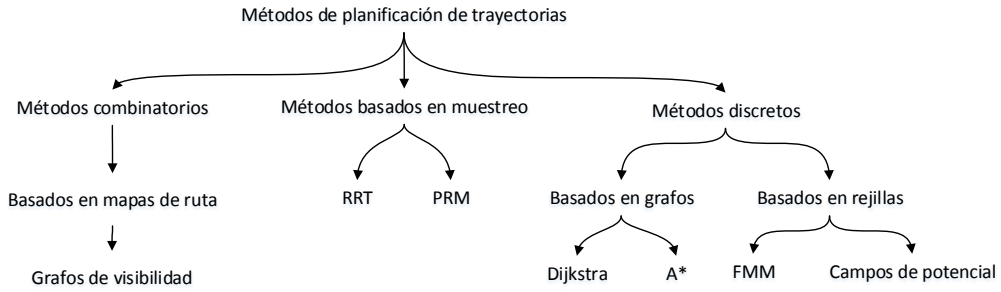


Figura 2.2: *Clasificación de los diferentes métodos de planificación de trayectorias.*

2.2. Últimos trabajos sobre planificación de trayectorias

Muchos investigadores consideran que la planificación de trayectorias es un problema actualmente resuelto. No obstante, la mayoría de las aproximaciones actuales bajan su rendimiento cuando se aplican a problemas de planificación complejos o de alta dimensionalidad. Las líneas de investigación actuales se encuentran enfocadas en una representación más realista del entorno y reformular el problema para obtener soluciones que se ajusten al mundo real.

A día de hoy, la detección de colisiones sigue siendo un cuello de botella dentro de la planificación de trayectorias basada en muestreo. Generalmente los algoritmos de planificación basados en muestreo integran la detección de colisiones como una caja negra en la que se encuentran encapsulados distintos algoritmos que se encargan de ello, independientes al métodos de planificación que se utilice. Para aplicar los detectores de colisión (Pan, Chitta, y Manocha, 2012) presenta una librería, *FCL*, que proporciona una interfaz unificada para realizar distintas consultas de proximidad y es capaz de manejar una amplia clase de modelos, incluyendo objetos rígidos y deformables, modelos articulados y nubes de puntos ruidosos.

Sobre estos algoritmos de detección de colisiones existen variaciones propuestas. Bialkowski propone un cambio en la forma de detectar las colisiones pasando de un resultado booleano a la distancia mínima de colisión

con los obstáculos (Bialkowski, Karaman, Otte, y Frazzoli, 2012). Con ello se incrementa el coste computacional pero también posibilita saltarse controles de colisión explícitos para una gran porción de las muestras posteriores, amortizando así el incremento de carga de cálculo. Aplicado sobre RRT se obtiene no solo una mejora en tiempos de cómputo sino una probabilidad mayor de que el algoritmo devuelva una trayectoria óptima.

Enfocado también a la detección de colisiones utilizando nubes de puntos como datos de sensores de entrada se presenta una nueva aproximación (Pan, Sucan, Chitta, y Manocha, 2013). Este enfoque convierte los datos en una estructura de datos octree, capaz de representar la incertidumbre del sensor, y desarrolla dos propuestas complementarias para llevar a cabo las consultas de colisión y proximidad. Por un lado se puede generar un árbol simple y mejorarlo a medida que se generan consultas de proximidad (ideal para entornos estáticos o simples), o directamente no inicializar el árbol y realizar las consultas directamente sobre el octree. Este segundo enfoque obtiene consultas más lentas pero sin el coste de inicialización, haciéndolo más óptimos para entornos de mayor complejidad. Con el fin de manejar oclusiones y la incertidumbre en los datos del sensor los algoritmos asumen que existe una probabilidad de ocupación. A medida que se reciben los datos del sensor, la probabilidad de ocupación se mantiene para ser un promedio de ocupación durante un número de tramas observadas anteriormente.

Aunque los métodos basados en muestreo son un área de trabajo muy común, no es la única. Uno de los aspectos básicos de los métodos basados en rejilla es la representación del espacio libre en mallas cuadradas o triangulares. La generación de este mallado es fundamental ya que debe ser capaz de recoger toda la información del entorno de la forma más precisa posible. Sobre este aspecto se presenta un método de refinado para mallados triangulares (Yershov y Frazzoli, 2014). Estas mallas se refinan para aumentar la precisión de los resultados, con el inconveniente de que hace falta un proceso de

inicialización en el que se genera un mallado basado en descomposición en celdas. No obstante las pruebas muestran que la ruta generada aplicando FMM es más óptima que el resultado de aplicar directamente RRT, por lo que se compensa ese coste. Sobre esta idea se plantea un método (Gomez, Álvarez, Garrido, y Moreno, 2013) en el cual se genera la malla triangular únicamente en la zona de interés, es decir, en el espacio que se encuentra entre el inicio y el fin de la trayectoria. Posteriormente y sobre dicha malla se aplica FMM para obtener la trayectoria, obteniendo resultados más cortos en distancia que los propuestos por RRT.

FMM, el método en el que se enmarca esta tesis, sigue siendo un campo en activo actualmente. Como se explicará en la sección 3, FMM es un método de planificación de orden $O(n\log(n))$ basado en la expansión de una onda, calculando el tiempo de llegada desde un punto a cada celda. Dicho tiempo de llegada se calcula utilizando la ecuación Eikonal, una ecuación diferencial compleja de la que existen distintas simplificaciones. Habitualmente los métodos proponen sistemas en los que el tiempo de llegada a una celda se calcula desde varios orígenes, pero (Jeong y Whitaker, 2008) propone un método en el que se evita el uso de estructuras de datos complejas al actualizar varios puntos en cada iteración y no únicamente el de menor valor como ocurre en FMM. Por otro lado también se propone otra solución al problema de la ecuación Eikonal, llamada *Group Marching Method* (GMM) (Seongjai Kim, 2000), la cual reduce la complejidad del problema a un orden $O(n)$ al evitar el uso de una cola de prioridad. Otro punto de vista es un método llamado *Untidy Fast Marching Method* (UFMM) (Yatziv, Bartesaghi, y Sapiro, 2005) en el cual se opta por crear una cola de prioridad sin ordenes complejos que permite mejorar la eficacia a costa de una pequeña bajada de precisión. Posteriormente se propone un método que estima el error introducir basado en un principio de comparación discreto (Rasch y Satzger, 2007). Finalmente en (Muñoz, 2014) se recoge un resumen y una comparativa de las distintas variaciones de FMM que

más se están utilizando actualmente, comparando distintas colas de prioridad y métodos para resolver la ecuación Eikonal entre los que se incluyen los descritos anteriormente.

La base para cualquier método enfocado a la expansión de ondas es la ecuación Eikonal, y a partir de ella se derivan distintos métodos que la implementan de forma eficiente. Por la construcción de los algoritmos basados en la expansión de ondas, el cálculo de la ecuación Eikonal es uno de los cuellos de botella. Para optimizar su resolución (Jeong, Ross, y Whitaker, 2007) propone un método basado en resolver la ecuación para un conjunto de celdas con actualizaciones iterativas de tal forma que la resolución es paralelizable utilizando multiprocesadores o GPUs. A su vez, existen también estudios sobre distintos métodos de paralelización con los que optimizar la resolución de la ecuación Eikonal (Borlaug, 2007). La ecuación Eikonal no solo se aplica a a mallas cuadradas, sino también a estructuras triangulares (Fu, Jeong, Pan, Kirby, y Whitaker, 2011), proponiendo un método optimizado basado en actualizaciones locales.

Aunque FMM es el enfoque más común para esta tarea existen alternativas muy extendidas como *Fast Sweeping Method* (FSM). Sobre este método se propone una optimización basada en paralelizar el cálculo de la ecuación Eikonal. Utilizando una mezcla de FMM y FSM se propone un método de mayor eficiencia computacional basado en la paralelización de la cola de prioridad, llamado *Heap-Cell Method* (HCM) (Chacon y Vladimirsky, 2013).

Reducir la complejidad computacional de los algoritmos de planificación es otra de las metas a alcanzar. En una parte importante de los métodos el algoritmo se expande en todas las direcciones, ya sean las ondas de FMM o el árbol de RRT. Intuitivamente la solución pasa por modificar el algoritmo de forma que su expansión vaya dirigida expresamente hacia el objetivo. Aunque es un campo en el que no se ha trabajado de forma extensiva, existen algunas aportaciones como un RRT direccionado utilizando campos de potencial

(Qureshi y cols., 2013) o el desarrollo de un nuevo algoritmo basado en campos de potencial y orientado hacia el objetivo llamado *nonlinear, anisotropic, harmonic potential fields* (Masoud y Masoud, 2002).

Además de estas soluciones para rebajar el coste computacional, en los últimos años se han desarrollado métodos específicos para espacios de configuración con alta dimensionalidad. Este problema se suele afrontar utilizando RRT con un procesado posterior que añada suavidad (Ettlin y Bleuler, 2006), (Karaman y Frazzoli, 2011) o variaciones sobre RRT para casos específicos. Las más comunes son *Transition-based Rapidly-exploring Random Trees* (T-RRT) (Jaillet, Cortés, y Siméon, 2010), que incluye un coste en las transiciones, *Manhattan-like Rapidly-exploring Random Trees* (ML-RRT) (Cortes, Jaillet, y Siméon, 2008) que incorpora una mezcla entre parámetros activos y pasivos, y *Manhattan-like Transition-based Rapidly-exploring Random Trees* (MLT-RRT) (Iehl, Cortés, y Siméon, 2012) que funciona como una combinación de los anteriores.

Además de los métodos basados en RRT existen otras propuestas como (Shkolnik y Tedrake, 2009), que trabaja únicamente bajo condiciones específicas, o la utilización de métodos derivados de FMM como *Fast Marching Tree* (FMT*) (Janson y Pavone, 2014). Este método está diseñado para minimizar el número de comprobaciones de colisión obteniendo una planificación óptima en términos de cómputo basándose en PRM, *mapas de ruta de árboles basados en muestreo* (SRT), RRT y FMM. Sobre este método se propone otra variante, *Bidirectional FMT** (BFMT*) (Starek y Pavone, 2014), que expande dos árboles, frente a uno que expandía FMT*, logrando así una solución óptima en menos tiempo. Basado también en FMT* se presenta el método *Motion Planning using Lower Bounds* (MPLB) (Salzman y Halperin, 2014). Este método se basa en observar el mapa de ruta introducido tras un conjunto de muestras para calcular cotas inferiores a la hora de calcular el coste para pasar de un nodo a otro, agilizando así las búsquedas.

A pesar de la amplia literatura existente, la mayor parte de las investigaciones en torno a planificación de trayectorias de los últimos años se encuentran enfocadas a obtener métodos que puedan ser utilizados bajo cualquier contexto. Pese a ello, la realidad es que las particularidades de cada método hacen que cada uno resulte óptimo para resolver tareas específicas, lo que resulta problemático. Dentro de este marco, esta tesis plantea distintos algoritmos de planificación basados en FMM y enfocados a un propósito general. En ellos se buscan obtener trayectorias intuitivas que reúnan suavidad y coste computacional reducido, dos de los mayores problemas que aparecen en este campo.

2.3. Fast Marching Method en planificación de trayectorias: trabajo previo

El uso de Fast Marching Method en planificación de trayectorias es un campo reciente, aunque posee multitud de usos actualmente. Al principio se usaba para encontrar caminos a lo largo de un diagrama de Voronoi (Garrido, Moreno, Abderrahim, y Martin, 2006). Posteriormente, FMM fue combinado con *Extended Voronoi Transform* (EVT) para crear el método *Voronoi Fast Marching* (VFM) (Garrido, Moreno, Blanco, y Muñoz, 2007). Este método se ha aplicado para planificación en ambientes desordenados (Garrido, Moreno, y Blanco, 2008) y formaciones de robots (Alvarez, Gomez, Garrido, y Moreno, 2014). Una mejora sobre el método VFM fue el método *Fast Marching Square* (FM^2) planning method (Garrido, Moreno, Abderrahim, y Blanco, 2009). Sobre FM^2 se desarrolló el método FM^{2*} (Valero-Gomez, Gomez, Garrido, y Moreno, 2013), que busca optimizar el tiempo de ejecución de FM^2 aplicando una heurística similar a la que propone A*.

Los métodos VFM y FM^2 han sido aplicados a otros problemas de planificación de trayectorias tales como planificación suave para robots no

holonómicos (Garrido, Moreno, Blanco, y Martin, 2009), localización y mapeado simultaneo para robots (Garrido, Moreno, y Blanco, 2009), planificación en exteriores (Malfaz, Garrido, y Blanco, 2012), (Sanctis, 2010), suavizar trayectorias de RRT (Jurewicz, 2010), esqueletos de tubo (Garrido, 2008) o planificación social (Javier V. Gómez y Garrido, 2013).

Recientemente, FMM y FM^2 se están utilizando para planificación de un enjambre de UAVs (González, Monje, y Balaguer, 2014), en terrenos con numerosos desniveles (Amorim y Ventura, 2014), replanificación dinámica cuando el mapa cambia (Xu, Stilwell, y Kurdila, 2013), (Gómez, 2012), o planificación para coches (Masehian, 2012), (Do, Mita, y Yoneda, 2014).

El rendimiento de los algoritmos de planificación y la comparación entre los mismos es un punto muy importante. Existen comparaciones en las que aparece FMM (Luo y Hauser, 2014) o FM^2 (Ansuategui y cols., 2014).

Capítulo 3

Fast Marching Square Planning Method

La planificación de trayectorias es un campo de amplio recorrido en el que se han desarrollado multitud de métodos o variaciones de los mismos para abordarlo. El presente documento está enfocado específicamente en el método Fast Marching Square, FM^2 (Garrido, Moreno, Abderrahim, y Blanco, 2009), y en variaciones sobre el mismo.

Este método se basa principalmente en la creación de campos de potencial artificial sobre un mapa generado por los sensores para obtener finalmente una trayectoria a partir de dichos campos. Para la creación de estos campos se aplica el método Fast Marching Method, FMM, permitiendo obtener además *campos libres mínimos locales*, solventando de este modo uno de los problemas más importantes a la hora de aplicar métodos de planificación de trayectorias.

Aunque FM^2 está basado en la creación de campos de potencial artificiales, este método no se encuentra dentro de los métodos de planificación basados en campos de potencial. Estos algoritmos poseen una base matemática muy diferente a la que tiene FM^2, ya que se basan en la colocación de dos campos de potencial diferentes: uno de atracción y otro de repulsión. De este modo el

robot es considerado como una partícula moviéndose bajo la influencia de estos campos. Sin embargo, como mayor inconveniente de estos métodos, es muy habitual alcanzar mínimos locales que desvirtúan el resultado del algoritmo, problema que no se encuentra presente en FM^2.

Por otro lado, FM^2 tiene en cuenta el concepto de *tiempo* en durante la creación de los campos de potencial. El robot es considerado una partícula que se mueve bajo la influencia del tiempo, eligiendo el camino por el que puede llegar a su objetivo en el menor tiempo posible. Por tanto, los caminos generados por FM^2 son los más óptimos en términos de tiempo.

Durante este capítulo se detallará la base matemática sobre la que se cimienta FM^2. Esto incluye de forma ineludible el desarrollo de FMM a lo largo de la sección 3.1 como base para el método FM^2, desarrollado en la sección 3.2.

3.1. Fast Marching Method (FMM)

El método Fast Marching Method (FMM) es un caso particular del Método del Conjunto de Nivel, desarrollado inicialmente por Osher y Sethian (Osher y Sethian, 1988). Se trata de un algoritmo numérico de gran eficiencia computacional orientado al seguimiento y modelado del movimiento de una interfaz física de onda. Este método, desde su base, es de propósito general por lo que es utilizado en multitud de campos de investigación, entre los que se encuentra la planificación de trayectorias.

3.1.1. Introducción a Fast Marching Method

Para entender FMM se puede pensar sencillamente en como se expande una onda en un fluido. Si se lanza una piedra a un charco de agua se originan ondas circulares concéntricas al punto en el que cae la piedra.

En este ejemplo se está suponiendo que el fluido es agua y es homogéneo. En caso de que el charco estuviera formado por una mezcla de distintos fluidos

las ondas circulares comenzarían a deformarse. Este fenómeno se explica con la velocidad de expansión de la onda. En el caso del agua, la velocidad es constante por lo que se generan círculos perfectos. Mientras, si está compuesto por varios fluidos, la velocidad de expansión es dependiente de la concentración de cada elemento en cada punto.

En cualquier caso la forma en la que dicha onda se expande hacia un punto es el camino más eficiente en términos de tiempo para alcanzar un objetivo, considerando además que la onda utiliza un perfil de velocidades para expandirse definido por la concentración en cada punto.

En FMM la onda es llamada interfaz, o Γ. La interfaz puede representarse como una curva plana en 2D, o como una superficie en 3D. No obstante, matemáticamente el modelo es generalizable a n dimensiones. FMM calcula cuanto tiempo, T, emplea una onda para alcanzar cada punto del espacio, partiendo de un punto de inicio. El número de puntos de inicio no se encuentra restringido a un único punto, ya que se puede situar distintos orígenes que generen otras tantas ondas. Dado que los tiempos calculados durante la expansión de la onda son relativos al origen, en cada uno de ellos el tiempo asociado será $T = 0$.

Dentro del contexto planteado por FMM se asume que la superficie que define la interfaz de la onda, Γ, es perpendicular en todo momento a la dirección de propagación. La velocidad, denominada F, no ha de ser constante a lo largo de la expansión, pero nunca podrá ser negativa. Dado un punto, el movimiento de expansión de la onda ha sido modelado mediante la ecuación Eikonal, presentada por Osher y Sethian (Osher y Sethian, 1988), y definida por:

$$1 = F(x)|\nabla T(x)|$$

donde x es la posición, $F(x)$ la velocidad de expansión de la onda en dicha posición, y $T(x)$ es el tiempo que requiere la onda para alcanzar la posición x.

La magnitud del gradiente de la función de llegada $T(x)$ es inversamente

proporcional a la velocidad:

$$\frac{1}{F(x)} = |\nabla T(x)|$$

La función $T(x)$ originada por una onda que se expande desde un único punto presenta únicamente un mínimo global, el punto de inicio, y ningún mínimo local, dado que se cumple siempre $F(x) \geq 0 \ \forall x$. Esta condición determina que la onda únicamente crece (se expande) y por tanto a medida que los puntos se alejan del origen el valor de T únicamente podrá crecer. Un mínimo local implicaría que existe un valor de T menor que el de su vecino más cercano al origen de la onda, caso imposible ya que a dicho punto se habrá llegado a través de su vecino más cercano.

En la sección 3.1.2 se presenta el modelo matemático en el que se basa FMM.

3.1.2. Formulación matemática de Fast Marching Method

La ecuación Eikonal se describe como:

$$\frac{1}{F(x)} = |\nabla T(x)|$$

Dado que la onda solo se puede expandir cumpliendo que $F(x) \geq 0 \forall x$, el tiempo de llegada, $T(x)$, tiene un único valor. Sethian propuso una solución discreta para resolver la ecuación Eikonal (Sethian, 1996). En el caso bidimensional el entorno se discretiza utilizando una rejilla de ocupación. Como notación se considerará que la rejilla está formada por i, j, donde i es la fila y j la columna de dicha rejilla, correspondiente a un punto $p(x_i, y_j)$ en el mundo real. La discretización del gradiente $\nabla T(x)$ se realiza de acuerdo al método propuesto en (Osher y Sethian, 1988), definido por la ecuación 3.1.

$$max(D_{ij}^{-x}T, 0)^2 + min(D_{ij}^{+x}T, 0)^2 + max(D_{ij}^{-y}T, 0)^2 + min(D_{ij}^{+y}T, 0)^2 = \frac{1}{F_{ij}^2} \quad (3.1)$$

o el método propuesto por Sethian en (Sethian, 1996), presentando en 3.2 una variación más sencilla a costa de una menor precisión:

$$max(D_{ij}^{-x}T, -D_{ij}^{+x}, 0)^2 + max(D_{ij}^{-y}T, -D_{ij}^{+y}, 0)^2 = \frac{1}{F_{ij}^2} \tag{3.2}$$

donde:

$$\begin{aligned} D_{ij}^{-x} &= \tfrac{T_{i,j}-T_{i-1,j}}{\triangle x} \\ D_{ij}^{+x} &= \tfrac{T_{i+1,j}-T_{i,j}}{\triangle x} \\ D_{ij}^{-y} &= \tfrac{T_{i,j}-T_{i,j-1}}{\triangle y} \\ D_{ij}^{+1} &= \tfrac{T_{i,j+1}-T_{i,j}}{\triangle y} \end{aligned} \tag{3.3}$$

y $\triangle x$ y $\triangle y$ representan el incremento de espacio en las direcciones x e y, y $T_{i,j}$ representa el tiempo de llegada en la celda (i, j). Si la ecuación 3.3 es sustituida en 3.2 se obtiene:

$$\begin{aligned} T &= T_{i,j} \\ T_1 &= min(T_{i-1,j}, T_{i+1,j}) \\ T_2 &= min(T_{i,j-1}, T_{i,j+1}) \end{aligned} \tag{3.4}$$

por tanto, para un espacio discreto en 2D, la ecuación Eikonal puede quedar definida tal y como se describe en la ecuación 3.5.

$$max\left(\frac{T-T_1}{\triangle x}, 0\right)^2 + max\left(\frac{T-T_2}{\triangle y}, 0\right)^2 = \frac{1}{F_{i,j}^2} \tag{3.5}$$

Como se ha comentado anteriormente, durante la expansión de la onda se asume que la velocidad será siempre positiva ($F > 0$), implicando así que T debe ser mayor que T_1 y T_2 siempre y cuando la onda no haya pasado previamente por las coordenadas i, j. Como consecuencia, la ecuación 3.5 puede ser resuelta como la ecuación cuadrática 3.6.

$$\left(\frac{T-T_1}{\triangle x}\right)^2 + \left(\frac{T-T_2}{\triangle y}\right)^2 = \frac{1}{F_{i,j}^2} \tag{3.6}$$

Siempre que se cumpla que $T > T_1$ y $T > T_2$. Dado que la onda se expandirá desde varios puntos sobre otro concreto, siempre se mantendrá el menor valor de T al resolver la ecuación 3.6. Si se da el caso en el que $T < T_1$ o $T < T_2$, se sustituirá su valor por 0 al resolver la ecuación 3.5 ($max\left(\frac{T-T_1}{\triangle x}, 0\right)$), y por tanto la ecuación Eikonal 3.5 queda reducida a la presentada en 3.7:

$$\left(\frac{T - T_1}{\triangle x}\right) = \frac{1}{F_{i,j}} \tag{3.7}$$

si el valor de T resultante es menor que T_1 al resolver la ecuación 3.6, o la ecuación 3.8:

$$\left(\frac{T - T_2}{\triangle y}\right) = \frac{1}{F_{i,j}} \tag{3.8}$$

si el valor de T es menor al de T_2 tras resolver la ecuación 3.6.

3.1.3. Detalles de implementación de Fast Marching Method

En la sección anterior se ha desarrollado la ecuación Eikonal, base matemática con la que definir la expansión de una onda, representada finalmente por la ecuación 3.6. Dicha ecuación puede ser resuelta de forma iterativa sobre una rejilla, para lo cual cada celda ha de ser etiquetada de algún modo. El convenio utilizado para la descripción de cada celda de la rejilla queda de definido del siguiente modo:

- *Unknown*: Celdas por las que la onda no ha pasado aún, por lo que su valor T no es conocido.
- *Narrow Band*: Celdas por las que se expandirá la onda en las siguientes iteraciones. Tienen asignado un valor de T pero puede variar en futuras iteraciones del algoritmo. La *narrow band* representa por tanto la interfaz de onda.
- *Frozen*: Celdas por las que ya ha pasado la onda y por tanto su valor de tiempo T es fijo.

El algoritmo se desarrolla en tres pasos: inicialización, bucle principal y finalización.

Inicialización El algoritmo comienza estableciendo un valor de tiempo $T = 0$ en cada celda que origina una onda. Estas celdas son etiquetadas directamente como *frozen* para evitar que se calcule su valor durante las iteraciones dado que no cambiará al no poderse dar un tiempo menor. Tras ese paso, se etiquetan todos sus vecinos Von Neumann *narrow band*, y se calcula el tiempo de llagada T, usando la ecuación 3.5, para cada uno de ellos.

Bucle principal En cada iteración del algoritmo se irá resolviendo de forma secuencial la ecuación Eikonal 3.5 para todas las celdas que pertenecen a la *narrow band* y no han sido etiquetadas como *frozen* aún, almacenando el menor valor de tiempo de llegada T. Cuando se ha finalizado el proceso con cada miembro de la *narrow band*, la celda se etiqueta como *frozen*. La narrow band mantiene una lista ordenada de celdas, utilizando como criterio el tiempo de llegada T (siguiendo un orden ascendente en el que el menor tiempo se sitúa primero), y se expande a través de los vecinos Von Neumann de la lista.

Finalización Cuando todas las celdas han sido etiquetadas como *frozen*, y por tanto la *narrow band* se encuentra vacía, el algoritmo concluye.

De forma ejemplificada para facilitar la compresión, en las figuras 3.1 y 3.2 se muestra el proceso de expansión. En la figura 3.1 la onda se origina desde un punto, mientras que en la figura 3.2 existen dos fuentes, representadas por el punto azul oscuro. Los puntos azul claro representan las celdas etiquetadas como *frozen*, lo que implica que su valor de tiempo de llegada T no cambiará. Las celdas etiquetadas como *narrow band* están representadas mediante puntos grises, mientras que los puntos blancos están etiquetados como *unknown*. Intuitivamente se puede apreciar que la onda se expande de forma circular y concéntrica a la fuente en ambos casos. En la figura 3.2 se muestra la expansión

simultanea de dos ondas desde distintas fuentes. El proceso expande por las celdas tal y como lo haría una onda en un espacio físico, expandiendo por las celdas con menor tiempo de llegada T. Esto quiere decir que en caso de confluir dos ondas en una misma celda y presentar distintos tiempos de llegada, se expandiría primero a través de la onda que presente menor tiempo de llegada en ese punto.

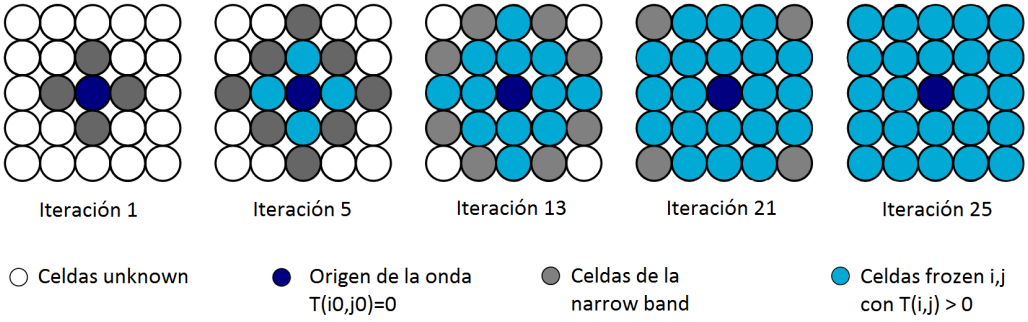


Figura 3.1: *Iteraciones de FMM en una rejilla 5x5 con un único origen de onda.*

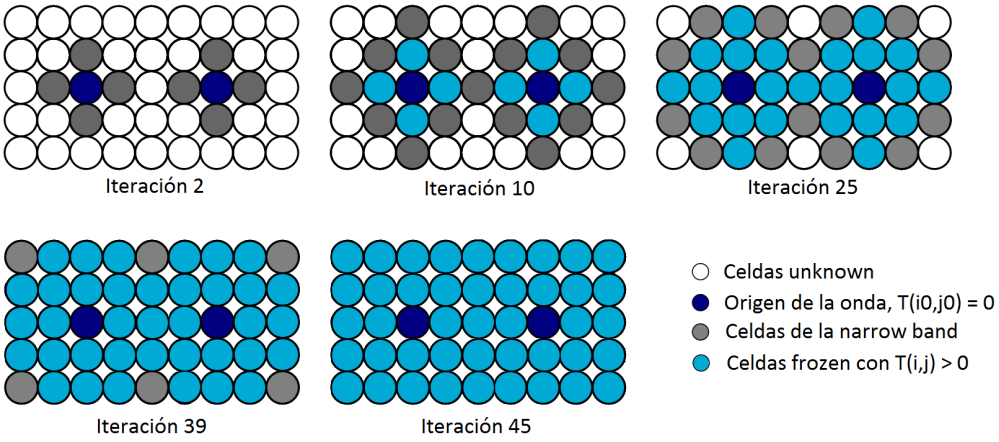
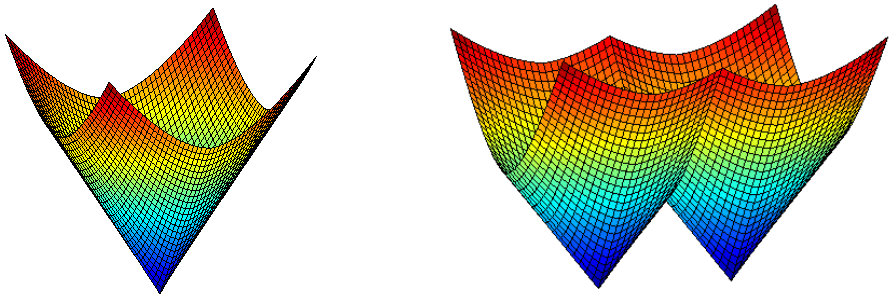


Figura 3.2: *Iteraciones de FMM en una rejilla 5x9 con dos orígenes de onda.*

Si se considera el valor del tiempo de llegada, T, como la tercera dimensión representada sobre el eje Z, queda como resultado las figuras 3.3a y 3.3b.

Dichas figuras representan la expansión de onda mostrada en las figuras 3.1 y 3.2 respectivamente. El resultado en la figura 3.3a muestra un mínimo global situado en el origen de la onda y un aumento de tiempo de llegada a medida que la onda se aleja del origen de la misma. Dicho resultado es predecible ya que la velocidad de expansión de la onda ha de cumplir siempre que $F > 0$, conllevando obligatoriamente que el tiempo de llegada T se vea incrementado. En la figura 3.3b aparecen dos mínimos locales, correspondientes en este caso a los dos orígenes de onda que se muestran en la figura 3.2. El punto en el que se unen las dos ondas es un punto de ensilladura.



(a) *Onda con una única fuente* **(b)** *Onda con dos fuentes*

Figura 3.3: *FMM aplicado sobre una rejilla. El eje z representa el tiempo de llegada T.*

Finalmente, el pseudo-código de FMM se muestra en el algoritmo 1.

```
input  : A grid map G of size m × n
input  : The set of cells Ori where the wave is originated
output: The grid map G with the T value set for all cells
Initialization;
foreach g_ij ∈ Ori do
    g_ij.T ← 0;
    g_ij.state ← FROZEN;
    foreach g_kl ∈ g_ij.neighbours do
        if g_kl = FROZEN then skip; else
            g_kl.T ← solveEikonal(g_kl);
            if g_kl.state = NARROW BAND then
            narrow_band.update_position(g_kl);
            if g_kl.state = UNKNOWN then
                g_kl.state ← NARROW BAND;
                narrow_band.insert_in_position(g_kl);
            end
        end
    end
    Iterations;
    while narrow_band NOT EMPTY do
        g_ij ← narrow_band.pop_first();
        foreach g_kl ∈ g_ij.neighbours do
            if g_kl = FROZEN then skip; else
                g_kl.T ← solveEikonal(g_kl);
                if g_kl.state = NARROW BAND then
                narrow_band.update_position(g_kl);
                if g_kl.state = UNKNOWN then
                    g_kl.state ← NARROW BAND;
                    narrow_band.insert_in_position(g_kl);
                end
            end
        end
    end
end
```

Algorithm 1: Pseudo-código de FMM

3.1.4. Aplicación de Fast Marching Method a la planificación de trayectorias

El método matemático propuesto hasta este momento plantea un modelo matemático de la expansión de una onda basada en los principios de Osher y Sethian (Osher y Sethian, 1988). Para poder aplicar dicha base física a la planificación de trayectorias se han de seguir varios pasos una vez se ha expandido la onda.

Comencemos considerando un mapa binario en el que las celdas libres están representadas por el valor 1 y las ocupadas por el valor 0. Estos valores pueden ser utilizados de forma directa como velocidad de expansión de la onda F sobre cada celda. Al tener velocidad 0, la onda nunca pasará por las celdas de obstáculos, ya que el tiempo de llegada será infinito, mientras que en las zonas libres la velocidad es constante e igual a 1. Para generar el camino más óptimo entre dos puntos definidos como p_0 y p_1, la onda se expandirá desde p_1 hasta p_0. Dadas las propiedades de FMM, la onda generada proporcionará el camino más óptimo en términos de tiempo entre ambos puntos. Dado que la velocidad de expansión es constante, la trayectoria será también la más óptima en cuanto a distancia.

La onda se origina desde el punto de destino, por lo que el tiempo de llegada de la onda $T(x)$ calculado tendrá un mínimo localizado en dicho punto. Por tanto, siguiendo la dirección del gradiente máximo desde el punto de inicio se confluirá en el punto de destino, obteniendo de este modo la trayectoria.

En la figura 3.4 se muestra un ejemplo. El sistema busca encontrar el camino más óptimo para alcanzar el punto de destino desde el punto de inicio. Primero se parte de un mapa binario obtenido mediante sensores. Por motivos de seguridad los obstáculos, representados en el mapa por el valor 0, se dilatan tanto como el tamaño del robot para evitar golpes. Tras ello se aplica FMM expandiendo la onda utilizando el punto de destino como origen de la misma. Cuando la interfaz de la onda alcanza el punto de destino la onda, Γ deja de

expandirse.

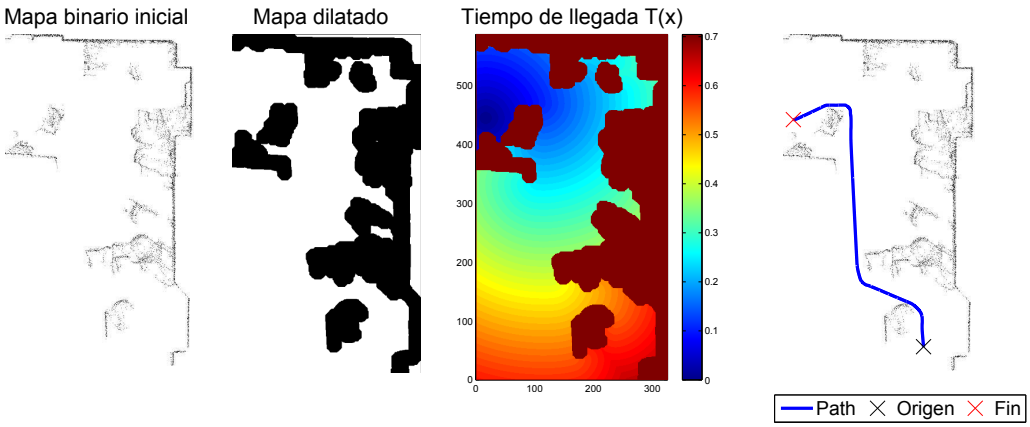


Figura 3.4: *Pasos para aplicar FMM a la planificación de trayectorias.*

La rejilla resultante contendrá el tiempo de llegada T a cada píxel desde el origen de la onda. Las isocurvas unen entre sí todos los puntos que han pasado durante el mismo instante de tiempo, formando así la interfaz de la onda. Si se calcula el gradiente de máxima dirección en cada punto de la rejilla se obtiene la dirección normal de la isocurva, lo que significa que se obtiene además la dirección de expansión de la onda. El gradiente de máxima dirección para cada una de sus dimensiones se calcula el valor medio de sus dos vecinos, para conocer el sentido de mayor crecimiento en cada dimensión. Ejemplificado en dos dimensiones, el gradiente se calcula como se describe en la ecuación 3.9.

$$\begin{aligned} grad_x &= \frac{T_{i-1,j} + T_{i+1,j}}{2} \\ grad_y &= \frac{T_{i,j-1} + T_{i,j+1}}{2} \end{aligned} \tag{3.9}$$

Para trazar el camino final solo se ha de seguir la dirección de máximo gradiente desde el punto de destino hasta el punto de inicio. El camino es calculado iterativamente, calculando los valores de $grad_{ix}$ y $grad_{iy}$ en cada punto p_i. Utilizando p_i se calcula p_{i+1} acorde a la ecuación 3.10. Este proceso se realiza hasta que el camino alcanza el mínimo global, es decir, que el tiempo de

llegada es 0. El resultado es el camino más cercano.

$$\begin{aligned} mod_i &= \sqrt{grad_{ix}^2 + grad_{iy}^2} \\ alpha_i &= \arctan(\tfrac{grad_{iy}}{grad_{ix}}) \\ p_{(i+1)x} &= p_{ix} + mod_i \cdot \cos(alpha_i) \\ p_{(i+1)y} &= p_{iy} + mod_i \cdot \sin(alpha_i) \end{aligned} \tag{3.10}$$

3.2. Fast Marching Square (FM^2)

En las secciones anteriores se ha descrito el método de planificación de trayectorias basados en el modelo de FMM descrito por Sethian (Sethian, 1996). Dichas trayectorias, como puede observarse en la figura 3.4, son óptimas acorde al criterio de la mínima distancia Euclídea. Pero dicha optimización implica varios inconvenientes, principalmente una gran brusquedad y la obligación de que las trayectorias pasen muy próximas a los obstáculos.

Estos factores hacen imposible el uso de FMM como planificador de trayectorias en robótica real en la mayoría de las aplicaciones. Principalmente, la cinemática y la dinámica de los robots impiden los cambios tan bruscos de sentido, por lo que la suavidad de la trayectoria se antoja imprescindible. Además, los errores en las medidas de los sensores y en la generación de los movimientos hacen necesaria una distancia de seguridad frente al obstáculo.

Para solucionar estos problemas se propone un método basado en FMM, Fast Marching Square (FM^2). El principio de FM^2 consiste en solucionar los problemas anteriores mediante la generación de un mapa de velocidades que modifique la expansión de la onda teniendo en cuenta la cercanía a los obstáculos.

Para ejemplificar se utilizará de nuevo un mapa binario con los obstáculos etiquetados como 0 y las zonas libres etiquetadas como 1. En la sección anterior se aplicó FMM sobre un mapa, utilizando un único punto como fuente, y obteniendo el resultado de la figura 3.4. La principal diferencia que propone

FM^2 es utilizar cada obstáculo como fuente de una onda, expandiendo por tanto una multitud de ondas al mismo tiempo. El resultado de este proceso sobre el mapa binario mostrado en la figura 3.4 se muestra en la primera imagen de la figura 3.5. Esta figura representa el campo de potencial sobre el mapa original, y se puede apreciar como a medida que la onda se aleja de los obstáculos el tiempo de llegada T es mayor. Este fenómeno puede ser enfocado como un *mapa de velocidades* del entorno binario. Si se considera el valor del tiempo de llegada T como el valor proporcional de la máxima velocidad permitida en dicho punto se puede ver como las velocidades son menores cuanto más cerca se encuentra del obstáculo, y aumentan a medida que se aleja la onda de los mismos. Tras el cálculo del potencial se reescalan las velocidades de forma relativa (entre 0 y 1) para obtener así el mapa de velocidades, que provee la velocidad segura para para el robot en cada punto del entorno. En la figura 3.6 se muestra un perfil de velocidades de una trayectoria mostrada en la tercera imagen de la figura 3.5. En dicha figura se puede apreciar como la trayectoria evita los obstáculos completamente.

La trayectoria se puede calcular siguiendo los pasos descritos en la sección 3.1.4, con una única excepción, y es la utilización del mapa de velocidades para extraer el valor de la velocidad F, que pasa de ser constante a 1 en todo el espacio libre a ser variable en función de la posición. Con esta información se expande la onda por la rejilla, considerando que la velocidad ha de cumplir que $F(x, y) = T(x, y)$, siendo $F(x, y)$ la velocidad en el punto x, y y $T(x, y)$ el valor del mapa de velocidades en el punto x, y, implicando inequívocamente que la velocidad de expansión depende de la posición, y dicha velocidad siempre será segura para el robot. Esta expansión de la onda a velocidad variable proporciona una trayectoria óptima en tiempo asumiendo que el robot se mueve a la velocidad máxima permitida en cada punto.

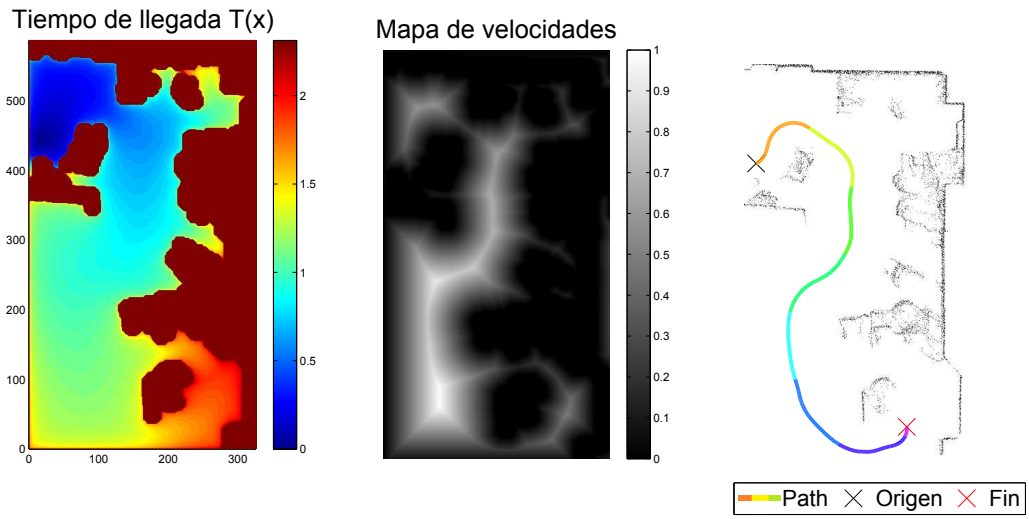


Figura 3.5: *Pasos de FM^2 aplicados a la planificación de trayectorias.*

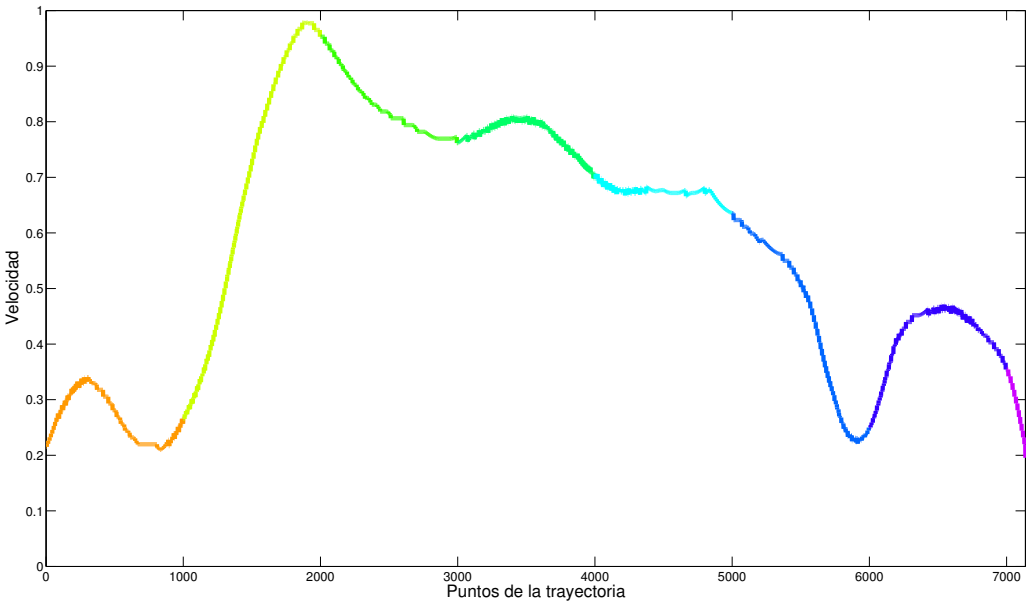


Figura 3.6: *Perfil de velocidades de la trayectoria mostrada en la figura 3.5*

Como consideraciones finales, FM^2 no requiere de la dilatación de los obstáculos ya que la propia generación del mapa de velocidades alejará al robot de las zonas cercanas a los mismos. No obstante el mapa puede ser dilatado previamente al cálculo del mapa de velocidades.

3.3. Conclusiones

En este capítulo se han presentado los fundamentos teóricos de FMM, desarrollando una implementación orientada a la planificación sobre rejillas de ocupación. En la sección 3.1.4 por su parte se expone como FMM puede aplicarse para calcular la trayectoria entre dos puntos del mapa.

Las limitaciones de este método radican en que la trayectoria generada, a pesar de ser la más óptima en tiempo y distancia, es también peligrosa dado lo cerca que pasa de los obstáculos, y además poco suave. En la sección 3.2 se detalla FM^2, presentado como solución a estos problemas. FM^2 calcula dos expansiones de onda sobre la rejilla. La primera de ellas sirve para generar un mapa de velocidades del cual se puede extraer la velocidad máxima a la que puede circular el robot en cada punto, mientras que la segunda utiliza el mapa de velocidades para expandir la onda desde el destino hasta el inicio de forma que el camino generado posteriormente aumenta en suavidad y seguridad.

FM^2 aparece como una solución segura y suave a la hora de generar trayectorias, pero como punto negativo, según la geometría del mapa puede resultar excesivamente restrictivo en cuanto a velocidad y distancia a los obstáculos. No obstante, como se mostrará a lo largo del presente documento, el algoritmo también se presenta como una base fuerte para el desarrollo de nuevos métodos, y permite ser implementado con éxito en plataformas reales.

Capítulo 4

Fast Marching Square Star Planning Method

En el capítulo 3 se ha realizado una introducción al método de planificación de trayectorias FM^2 y, por ende, también sobre FMM. En ambos casos se ha desarrollado la matemática necesaria para desarrollar los algoritmos y aplicarlos a planificación de trayectorias, mostrando al mismo tiempo sus ventajas e inconvenientes desde una visión crítica.

En esta sección se va a desarrollar una variación sobre el método FM^2 que busca obtener la trayectoria en un tiempo menor. Para ello añade una heurística inspirada en el método de planificación A*. En el caso de A* (Hart y cols., 1968) parte del algoritmo de Dijkstra (Dijkstra, 1959) para búsquedas de datos, añadiendo además una heurística que se suma al valor de coste original. Por su parte, FM^2 propone una heurística basada en añadir un tiempo pasado en la distancia de la celda al objetivo y la velocidad máxima permitida por el robot en el momento de resolver la ecuación Eikonal.

A lo largo del presente capítulo se desarrollarán dos variantes, la inicial planteada por Alberto Valero (Valero-Gomez y cols., 2013), y una variación propuesta para esta tesis que aporta una variación en el cálculo de la heurística.

Posteriormente se realizará una comparativa entre ellas, analizando tanto la suavidad de la trayectoria como el tiempo de ejecución para generar la misma trayectoria.

4.1. Fast Marching Square Star (FM^2*) clásico

En el capítulo 3 se ha descrito el funcionamiento del algoritmo FM^2 aplicado a planificación de trayectorias, analizando además sus virtudes y defectos. El método Fast Marching Square Star (FM^2*) busca alcanzar una trayectoria idéntica al método FM^2 en términos de distancia y suavidad pero mejorando el tiempo de cómputo. Para ello se añade una heurística que modifica el tiempo de llegada de la onda a cada celda basada en la distancia entre la celda analizada y el objetivo y la velocidad máxima del robot. La onda, al igual que en FM^2, finalizará su expansión cuando alcance el objetivo, evitando expandir la onda más de lo necesario y por tanto rebajando la carga computacional. De éste modo, las mejoras de tiempo aumentarán cuando lo haga la distancia entre el punto de inicio, ya que una onda direccionada hacia el objetivo alcanzará el punto de destino antes que la expandida sin heurística que la modifique.

De forma más detallada, el tiempo de llegada de la onda se calculará acorde a la ecuación 4.1.

$$T' = T + heuristic \tag{4.1}$$

donde T es el tiempo de llegada calculado con la ecuación Eikonal, $heuristic$ es el valor de la heurística, y T' el tiempo de llegada teniendo en cuenta la heurística. La heurística busca direccionar la onda añadiendo al tiempo de llegada un tiempo nuevo que relaciona la distancia a la que se encuentra la celda analizada del objetivo y la velocidad máxima del robot. Para calcular la distancia entre celdas se aplica la distancia euclídea, por lo que el valor de $heuristic$ queda definido según la ecuación 4.2.

$$heuristic = \frac{d\_E(start, goal)}{robot\_max\_speed} \tag{4.2}$$

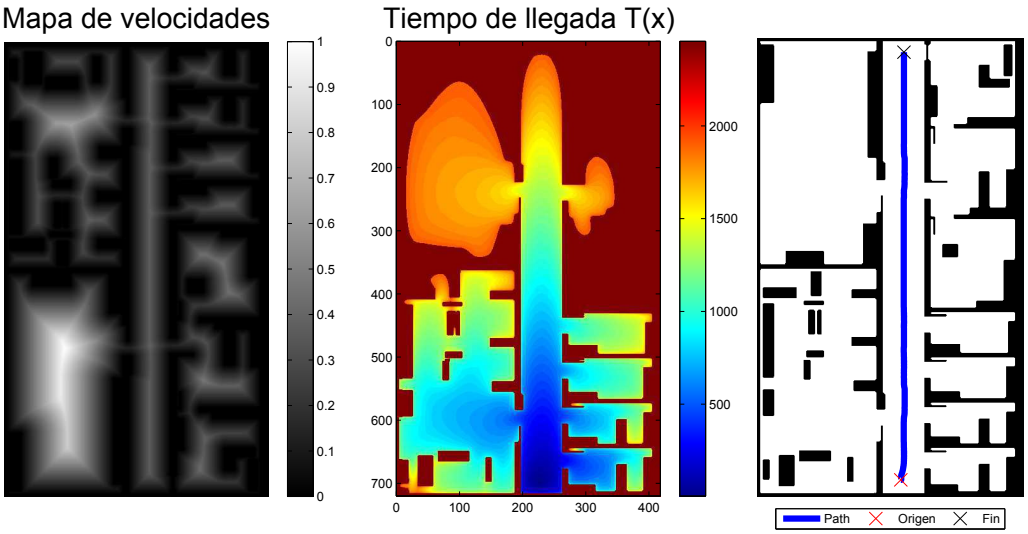
donde $d\_E(start, goal)$ es la distancia euclídea entre la celda por la que se expande la onda ($start$) y el objetivo ($goal$), $robot\_max\_speed$ es la velocidad máxima permitida por el robot y $heuristic$ el valor de la heurística.

La modificación del tiempo de llegada que propone FM^2* afecta directamente a la elección de celdas a partir de las que expandir la narrow band, dando que ésta se encuentra ordenada por tiempo de llegada. La aplicación de la heurística fuerza a que el tiempo de llegada sea más favorable para el vecino Von Neumann más cercano al objetivo, buscando así que la expansión de la onda continue a partir de ahí.

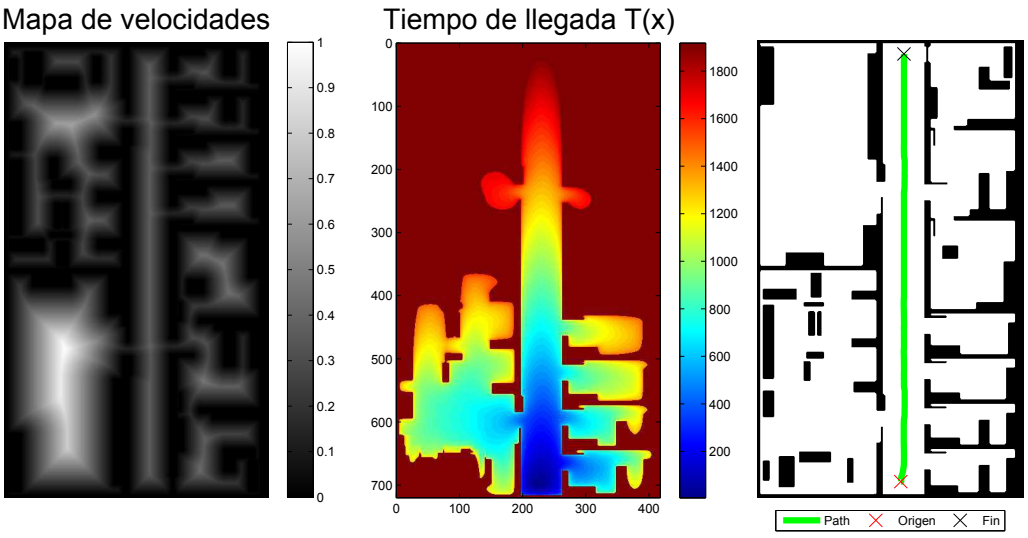
Dado que la heurística no tiene en cuenta la proximidad a los obstáculos puede ocasionar que el fin de FM^2, que no es otro que obtener trayectorias suaves y alejadas de los obstáculos, se pierda. Por ello se almacenan los datos en dos variables diferentes, la primera de ellas se corresponderá al tiempo obtenido mediante la ecuación Eikonal sin heurística, y la otra almacenará la heurística. De este modo la suma de ambas será utilizada para expandir la onda, mientras que el tiempo de llegada sin modificar será utilizado para calcular el gradiente y obtener de este modo la trayectoria.

De este modo se obtiene una variación análoga a la que propone A* sobre Dijkstra, es decir, direccionar la búsqueda añadiendo una heurística a la función de coste.

En la figura 4.1 se muestra una comparativa entre FM^2 y FM^2* clásico. En ella se puede apreciar como la expansión de la onda en el caso de FM^2* se encuentra notablemente más direccionada hacia el objetivo, entrando en menor medida dentro de las habitaciones, obteniendo así además una mejoría en términos de cómputo.



(a) *Camino utilizando FM^2.*



(b) *Camino utilizando FM^2* clásico.*

Figura 4.1: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2* clásico.*

Dados estos datos se puede destacar una trayectoria obtenida idéntica en ambos casos. Además se cumple con el objetivo de obtener una bajada en cuanto a tiempo de computación, que se verá incrementada a medida que el tamaño del

mapa y la distancia entre los puntos aumente, como se mostrará en siguientes secciones.

4.2. Fast Marching Square Star (FM^2*) propuesto

En este capítulo se detallará la alternativa propuesta al método FM^2* desarrollada en la sección 4.1. En ella se explica la necesidad de almacenar dos tiempos diferentes para cada celda, utilizando el tiempo con la heurística para expandir la onda y el tiempo sin ella para calcular el gradiente y extraer la trayectoria. Aunque en tiempo de cómputo la diferencia es pequeña al realizar los mismos cálculos, a nivel de memoria supone utilizar el doble de recursos.

Para solucionar este problema se propone un cambio en el cálculo de la heurística, propuesto en la ecuación 4.3.

$$heuristic = \frac{d\_E(start, goal)}{cell\_speed} \tag{4.3}$$

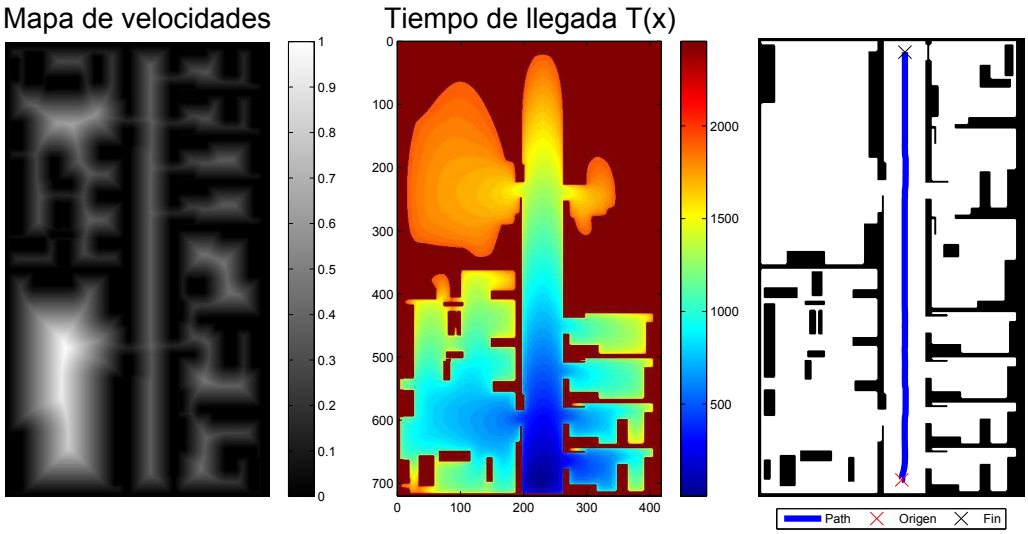
donde $d\_E(start, goal)$ es la distancia euclídea entre la celda por la que se expande la onda ($start$) y el objetivo ($goal$), $cell\_speed$ es la velocidad en la celda en la que se expande extraída del mapa de velocidades, y $heuristic$ el valor de la heurística.

De este modo se tiene en cuenta no sólo la distancia de la celda al objetivo, sino la distancia de la celda a un obstáculo a través de la velocidad máxima en dicho punto. Dicho de otra forma, la heurística penalizará a medida que la onda se acerque a los objetivos, logrando así no solo que la onda se expanda de forma direccional hacia el objetivo sino que además intenta alejarse de los obstáculos. Debido a esta ventaja no es necesario almacenar dos tiempos diferentes, ya que el cálculo con heurística contempla todas las variables.

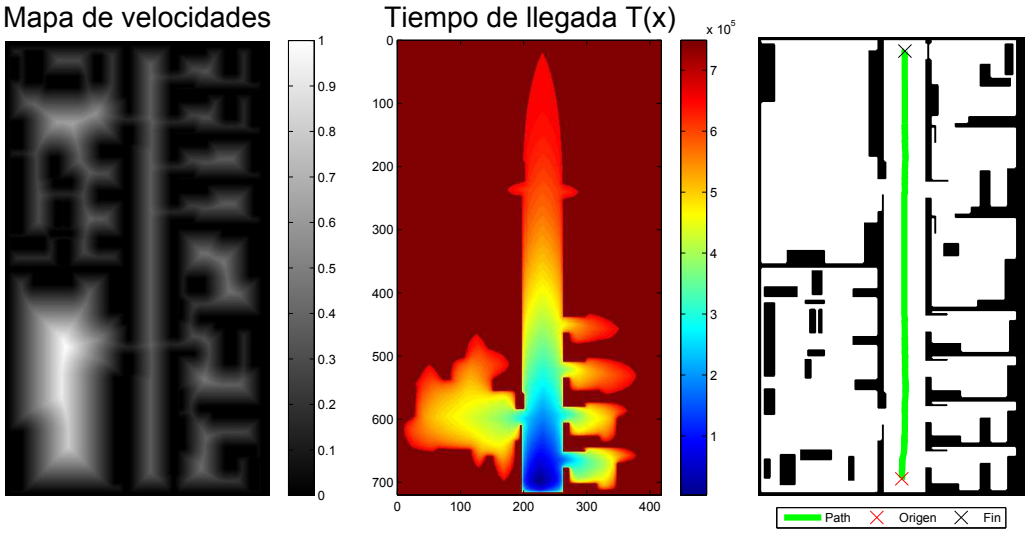
En la figura 4.2 se muestra una comparativa entre FM^2 y FM^2* propuesto, pudiendo apreciar como en este caso la expansión de la onda difiere notablemente, siendo que la de FM^2* propuesto es notablemente más

direccionada hacia el objetivo que la de FM^2. Analizando el tiempo de cómputo de la expansión de la segunda onda, la mejora queda patente ya que el tiempo se ha reducido un 15 % frente al utilizado por FM^2. La trayectoria, por su parte, es diferente entre ambos métodos, aunque manteniendo en ambos casos la suavidad y la búsqueda de un camino alejado de los obstáculos.

Dados estos datos se puede destacar una trayectoria obtenida muy parecida en ambos casos, cumpliendo además con el objetivo de obtener una bajada en cuanto a tiempo de computación. Esta mejoría se verá incrementada a medida que el tamaño del mapa y la distancia entre los puntos aumente, como se mostrará en siguientes secciones.



(a) *Camino utilizando FM^2.*



(b) *Camino utilizando FM^2* propuesto.*

Figura 4.2: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2* propuesto.*

4.3. Comparativa entre FM^{2*} clásico y propuesto

En esta sección se realizará una comparativa entre los dos métodos de FM^{2*} explicados en las secciones 4.1 y 4.2. La comparativa consistirá en realizar una trayectoria que enlace el mismo punto de inicio y final con ambas aproximaciones de FM^{2*}, y sobre el resultado comparar la expansión de la onda y el camino obtenido comparado con el que se obtiene aplicando FM^{2}. Posteriormente se realizará un análisis de tiempo que dura la segunda expansión de la onda, expansión en la cual se aplica la heurística, y de la suavidad de la trayectoria obtenida.

4.3.1. Detalles de implementación

Para comparar el tiempo de ejecución se parte de la implementación en C++ realizada de los tres algoritmos, suponiendo una optimización pareja en los tres casos. La base sobre la que se han implementado los distintos algoritmos parte de la implementación generalizada a n dimensiones de Fast Marching en C++ realizada por Javier V. Gómez[1]. A partir de su código se han implementado los distintos algoritmos, buscando siempre el mayor grado de optimización.

El tiempo de cómputo es orientativo ya que tiene una fuerte dependencia de la potencia del hardware sobre el que se ejecuta el código. El hardware utilizado para las pruebas consta de un microprocesador Intel i5 de cuatro núcleos a 3.3 GHz y 8 GB de memoria RAM.

La suavidad, por su parte, se calculará de un modo análogo al que se propone en la librería OMPL, orientada a planificación de trayectorias. La idea en la que se basa es analizar los triángulos que forman segmentos de la trayectoria consecutivos, calculando el ángulo entre ellos utilizando el teorema de Pitágoras. Posteriormente se normaliza el ángulo exterior al calculado para los segmentos de la trayectoria que sirve para analizar la suavidad. En el caso

[1]Código: https://github.com/jvgomez/fastmarching

de una línea recta, su suavidad será 0. Por tanto el resultado de esta operación será más suave cuanto más próximo a 0 resulte. La fórmula se define como 4.4.

$$smoothness = \sum_{i=2}^{n-1} \left(\frac{2\left(\pi - arccos\left(\frac{a_i^2+b_i^2+c_i^2}{2a_ib_i}\right)\right)}{a_i + b_i} \right)^2 \tag{4.4}$$

donde a es la distancia entre los puntos s_{i-2} y s_{i-1}, b es la distancia entre los puntos s_{i-1} y s_i y c es la distancia entre los puntos s_{i-2} y s_i.

Aunque el método es generalizable a n dimensiones, todas las pruebas se han realizado en entornos de 2 dimensiones. El motivo radica en que los algoritmos serán implementados en un robot Turtlebot para comprobar su funcionalidad aplicada a robótica móvil, trabajando únicamente en dos dimensiones. Todas las pruebas se realizarán sobre un mismo mapa con su mapa de velocidades asociado, mostrado en la figura 4.3.

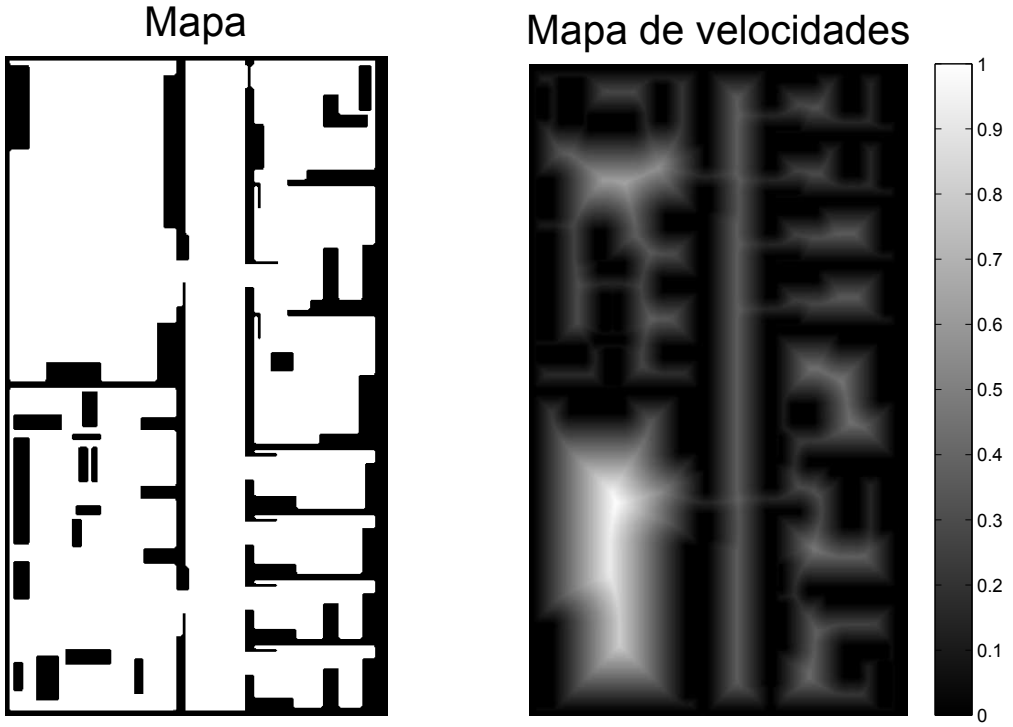


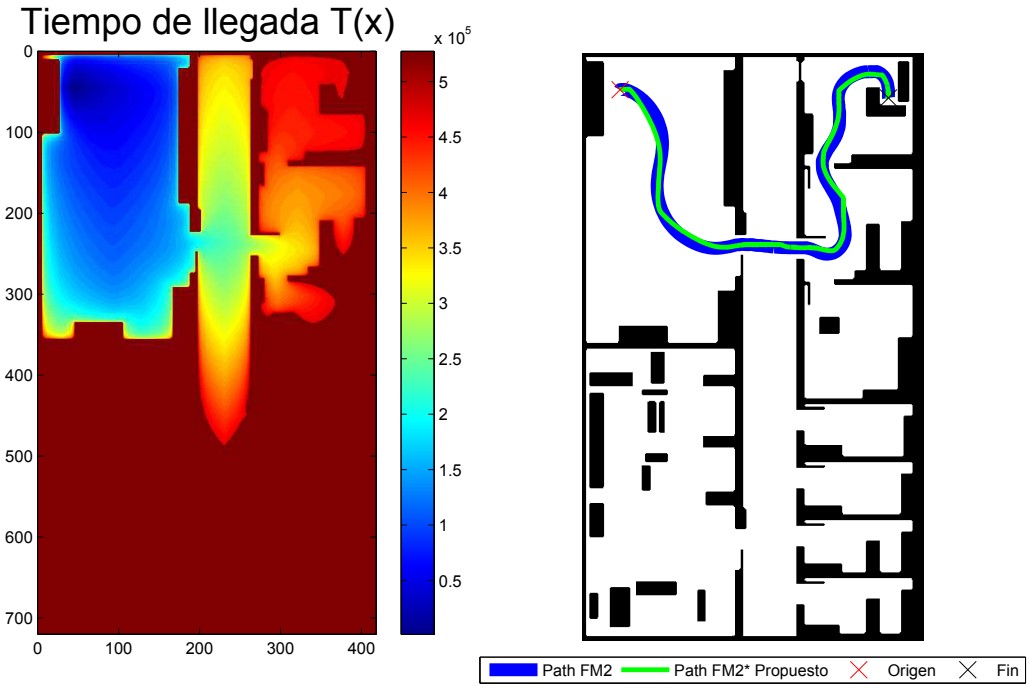
Figura 4.3: *Mapa y mapa de velocidades utilizado para las pruebas de FM^2*.*

4.3.2. Resultado de las pruebas

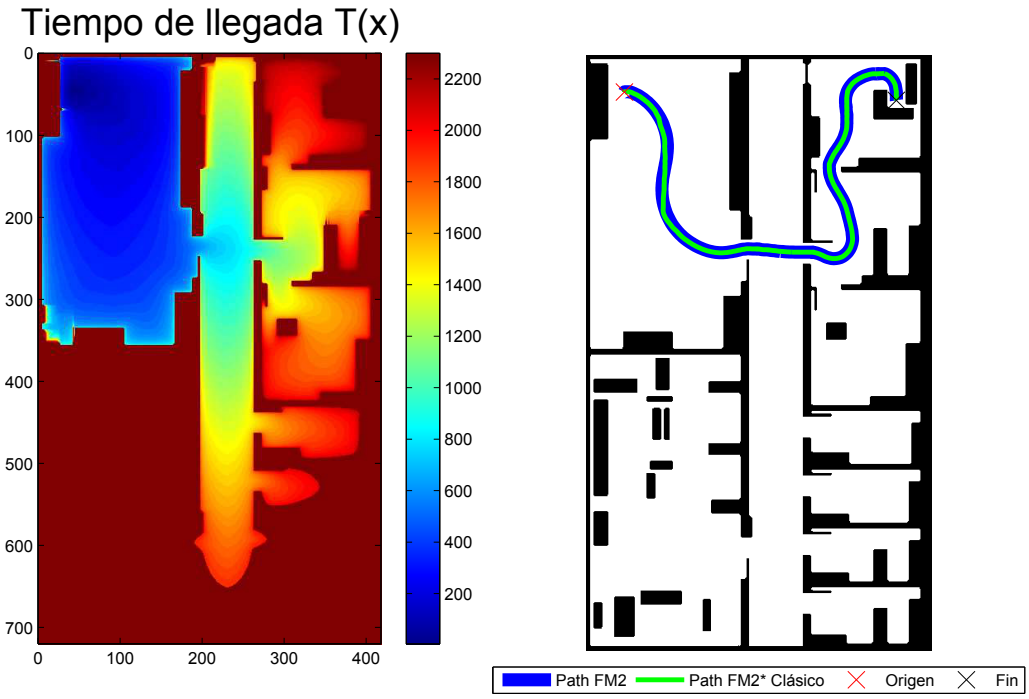
Durante esta sección se realizarán pruebas sobre un mismo mapa y varios puntos de inicio y final. En dichas pruebas se podrá evaluar la expansión de la onda y las trayectorias generadas en cada caso, realizando al final de los ensayos una muestra de tiempo de cómputo y suavidad de las trayectorias generadas en cada uno de los casos.

En la primera prueba, mostrada en la imagen 4.4, se puede ver como la expansión de la onda varía en ambos casos. Intuitivamente, la expansión de la onda en el caso del método de FM^{2*} propuesto se encuentra notablemente más direccionada que en el caso del método FM^{2*} clásico. No obstante, en la trayectoria aparece un resultado a valorar, y es que mientras que la trayectoria de FM^{2*} clásico se superpone de forma perfecta sobre la de FM^{2}, la de FM^{2*} muestra una trayectoria similar pero no idéntica.

Durante los siguientes ensayos, correspondientes a las figuras 4.5, 4.6, 4.7, 4.8 y 4.9, se puede ver de nuevo como la expansión de la onda varía en ambos casos, siendo de nuevo la del método de FM^{2*} propuesto la más direccionada. Comparando las trayectorias aparece el mismo punto a resaltar, ya que se vuelve a dar que la trayectoria de FM^{2*} clásico se superpone de forma prácticamente perfecta sobre la de FM^{2} mientras que la de FM^{2*} muestra un camino cercano pero no idéntico.

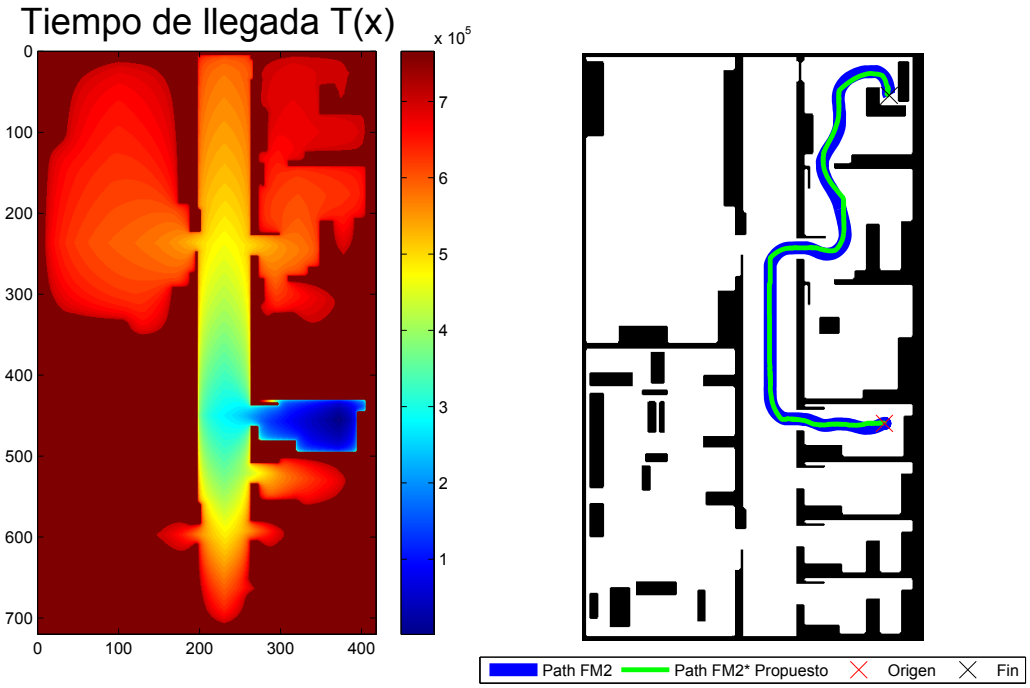


(a) *Camino utilizando FM^{2}* propuesto.*

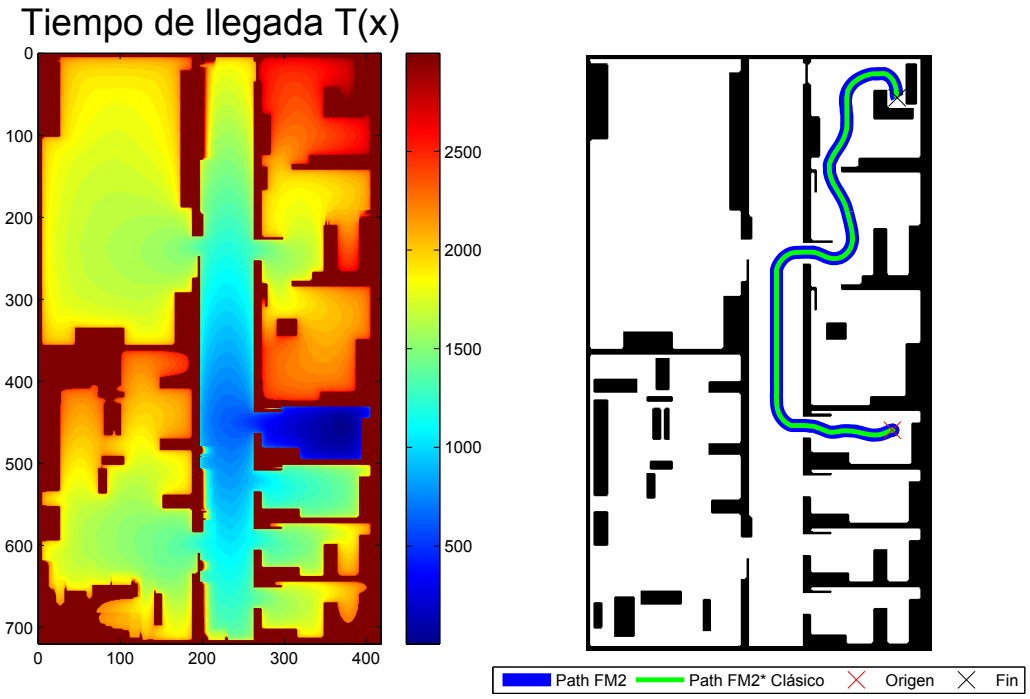


(b) *Camino utilizando FM^{2}* clásico.*

Figura 4.4: *Comparativa de trayectorias y expansión de onda de FM^{2}* propuesto frente a FM^{2}* clásico en caso 1.*

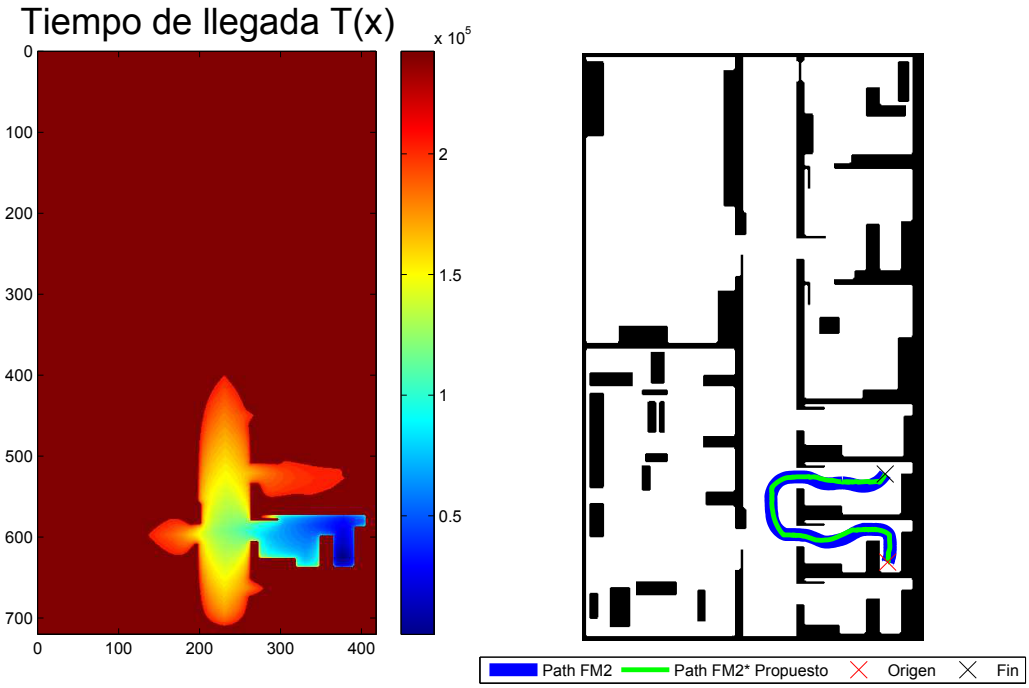


(a) *Camino utilizando FM^{2}* propuesto.*

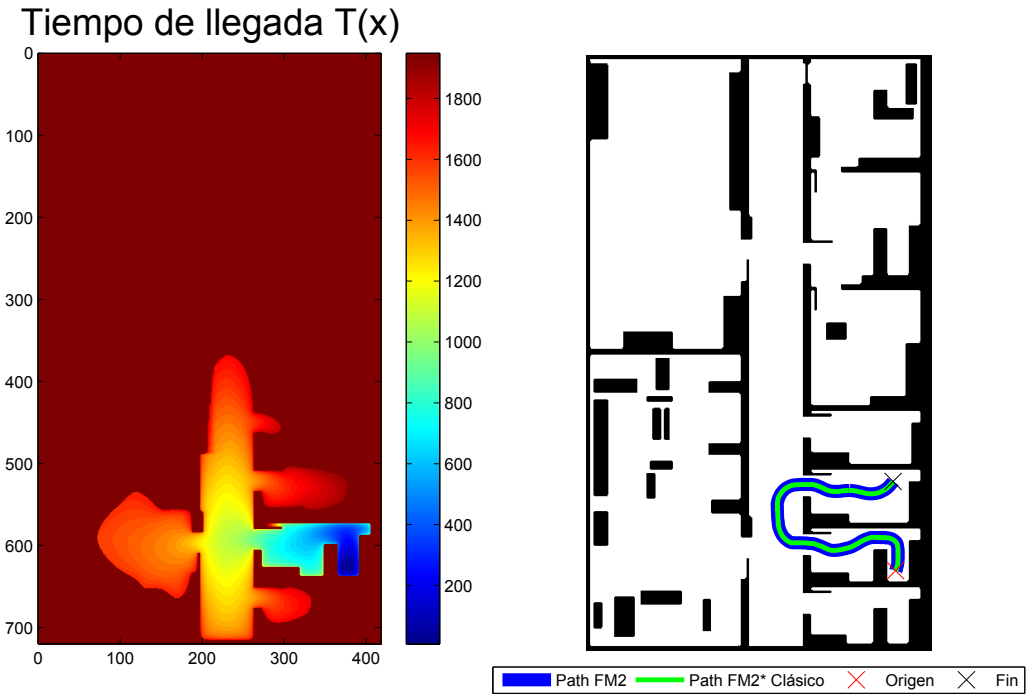


(b) *Camino utilizando FM^{2}* clásico.*

Figura 4.5: *Comparativa de trayectorias y expansión de onda de FM^{2}* propuesto frente a FM^{2}* clásico en caso 2.*

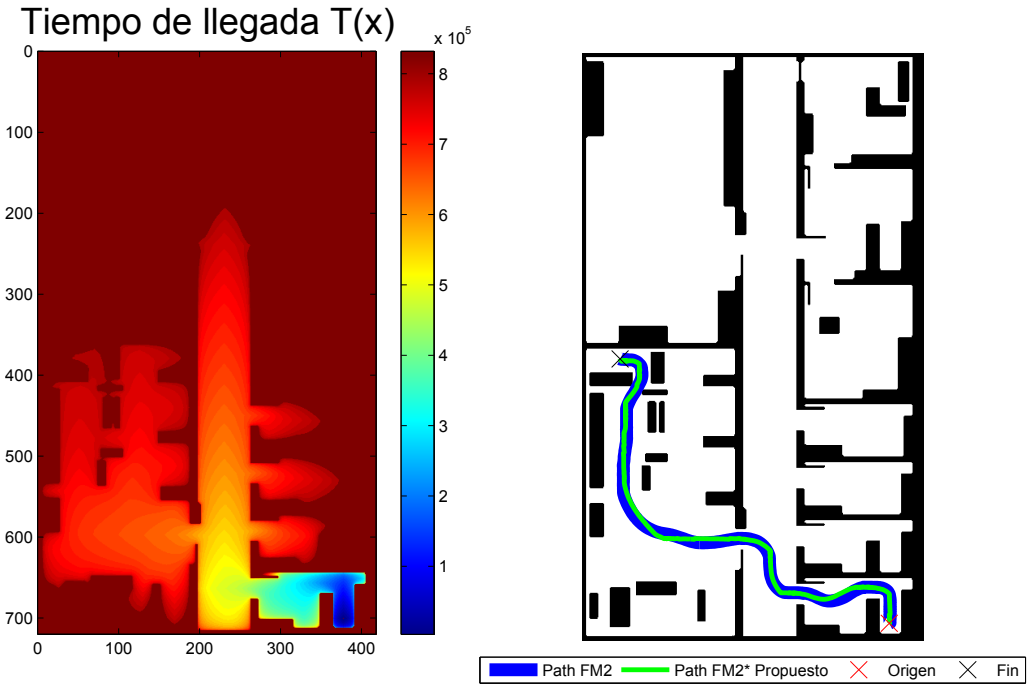


(a) *Camino utilizando FM^{2}* propuesto.*

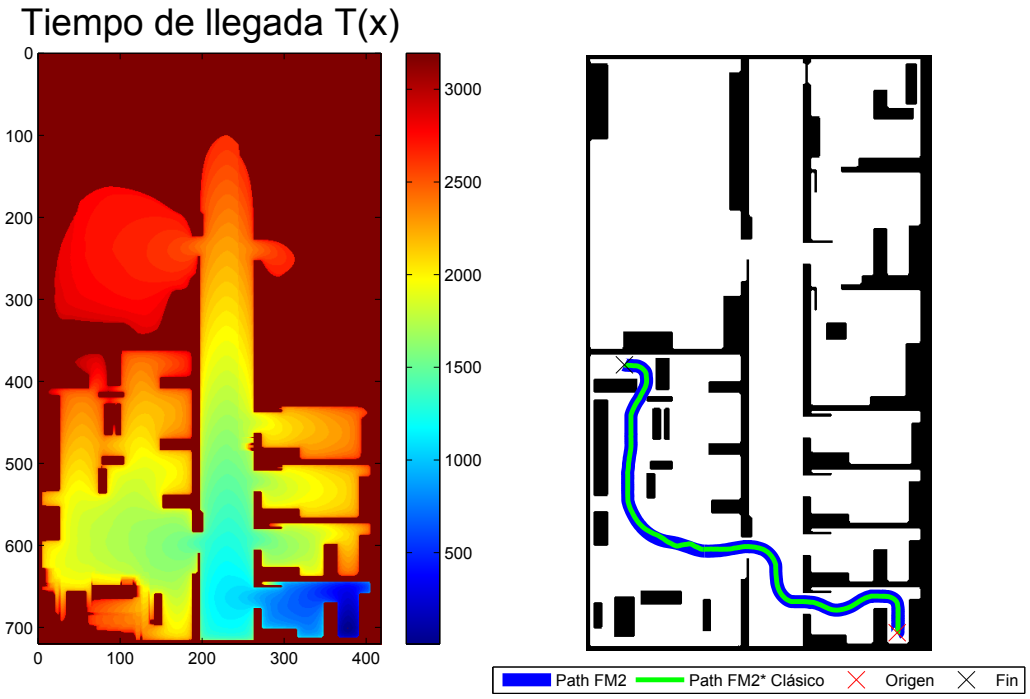


(b) *Camino utilizando FM^{2}* clásico.*

Figura 4.6: *Comparativa de trayectorias y expansión de onda de FM^{2}* propuesto frente a FM^{2}* clásico en caso 3.*

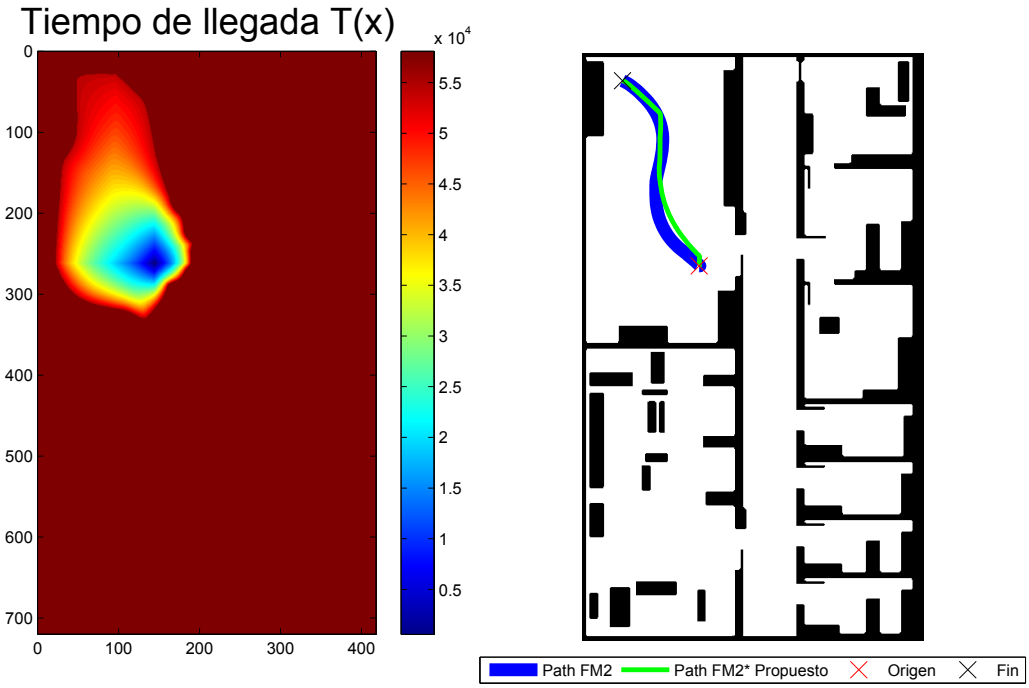


(a) *Camino utilizando FM^2* propuesto.*

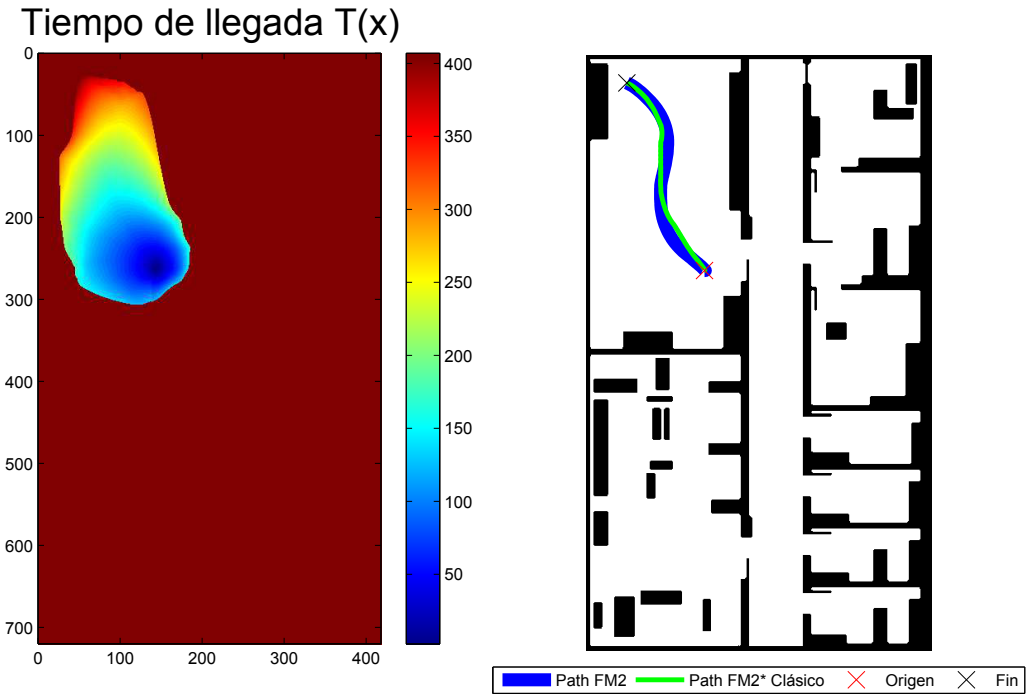


(b) *Camino utilizando FM^2* clásico.*

Figura 4.7: *Comparativa de trayectorias y expansión de onda de FM^2* propuesto frente a FM^2* clásico en caso 4.*

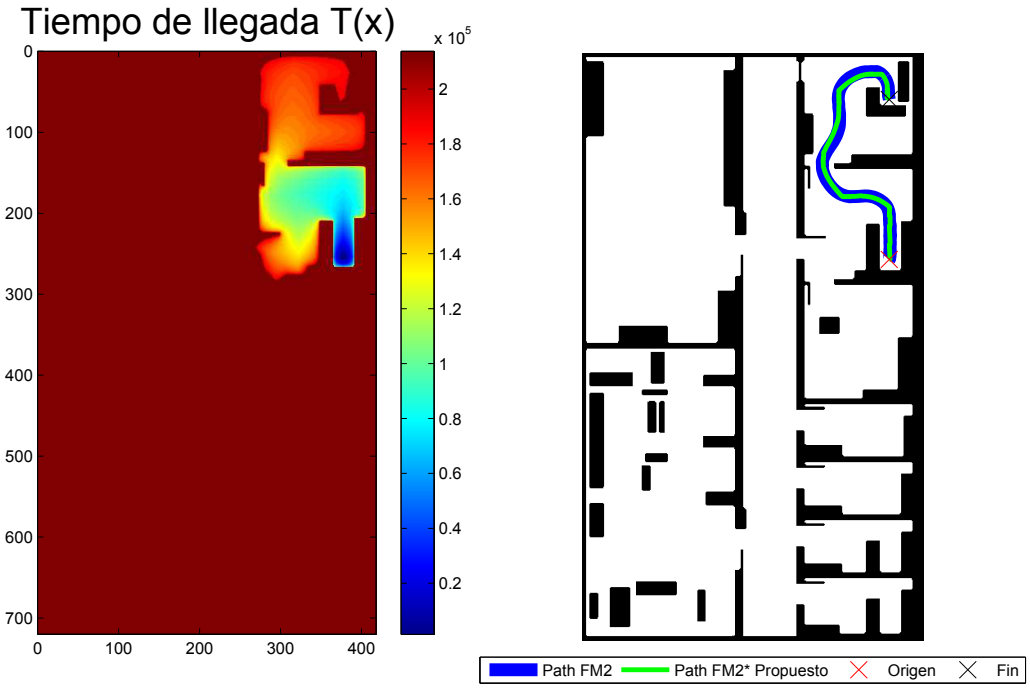


(a) *Camino utilizando FM^{2}* propuesto.*

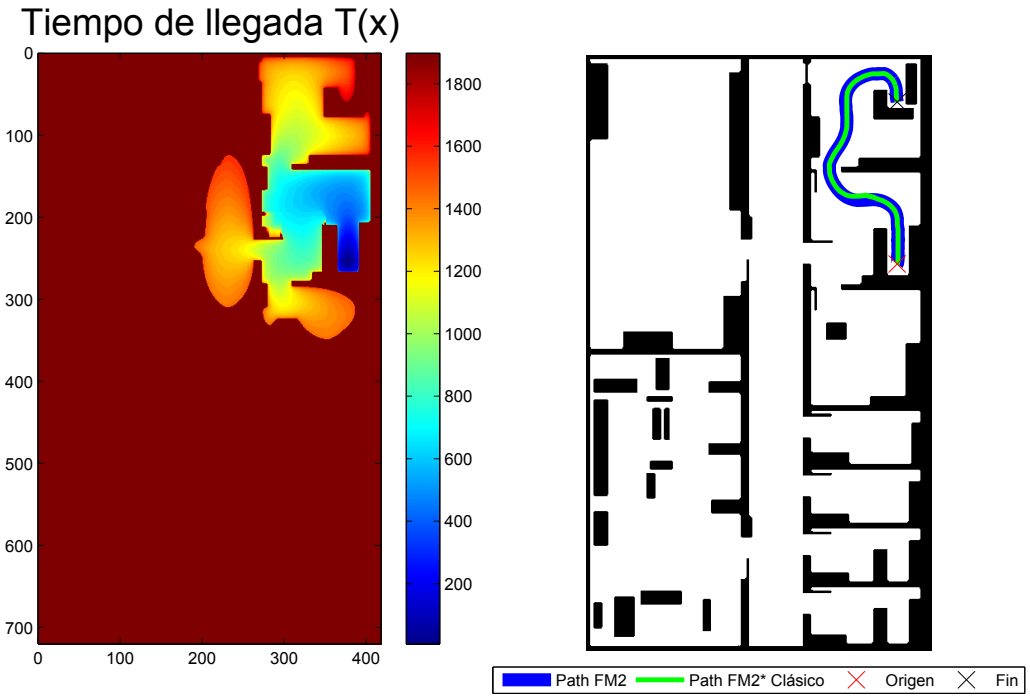


(b) *Camino utilizando FM^{2}* clásico.*

Figura 4.8: *Comparativa de trayectorias y expansión de onda de FM^{2}* propuesto frente a FM^{2}* clásico en caso 5.*



(a) *Camino utilizando FM^{2}* propuesto.*



(b) *Camino utilizando FM^{2}* clásico.*

Figura 4.9: *Comparativa de trayectorias y expansión de onda de FM^{2}* propuesto frente a FM^{2}* clásico en caso 6.*

En las figuras 4.10 y 4.11 se muestra una comparativa de los valores de suavidad en cada ensayo, calculados acorte a la fórmula 4.4. Como se puede ver, en el caso de FM^{2}* clásico la suavidad es muy próxima a la de FM^{2}, lógico dado que las trayectorias que proponen ambos métodos son las prácticamente idénticas. Mientras, en el caso de FM^{2}* propuesto, el nivel de suavidad baja ligeramente, pero no obstante sigue proporcionando un buen nivel de suavidad y por tanto no se puede considerar un motivo para rechazar el método.

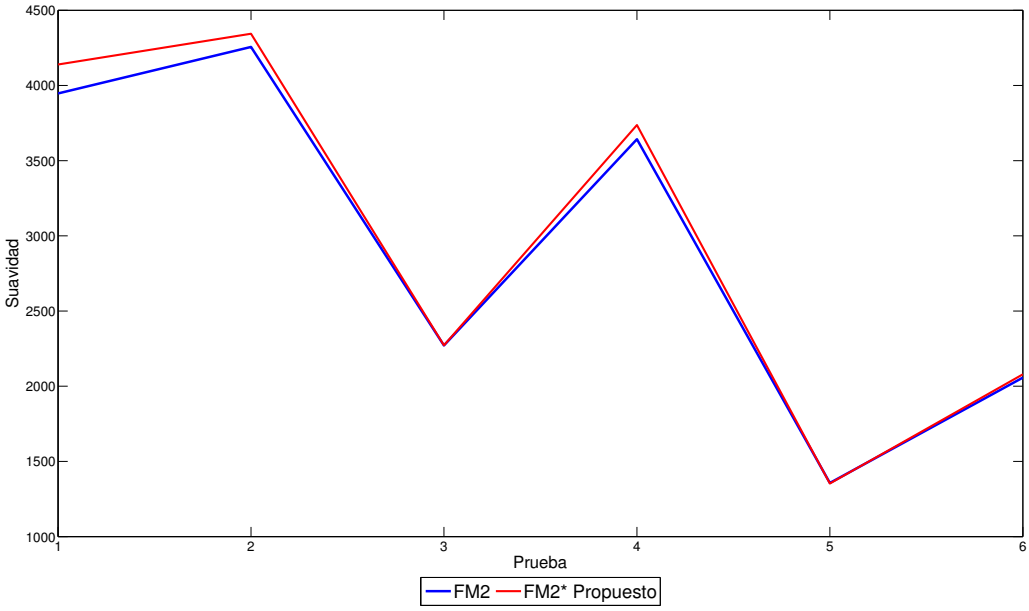


Figura 4.10: *Suavidad de la trayectoria utilizando FM^{2}* propuesto.*

Dado que la motivación para realizar este método es disminuir el tiempo de ejecución del algoritmo es inevitable realizar una análisis de tiempo de ejecución. Como se puede ver en la figura 4.12, los tiempos de ejecución del método FM^{2}* clásico frente a FM^{2} se ven disminuidos de forma visible. No obstante, el método FM^{2}* propuesto obtiene una bajada de tiempos muy superior a la que propone el método clásico.

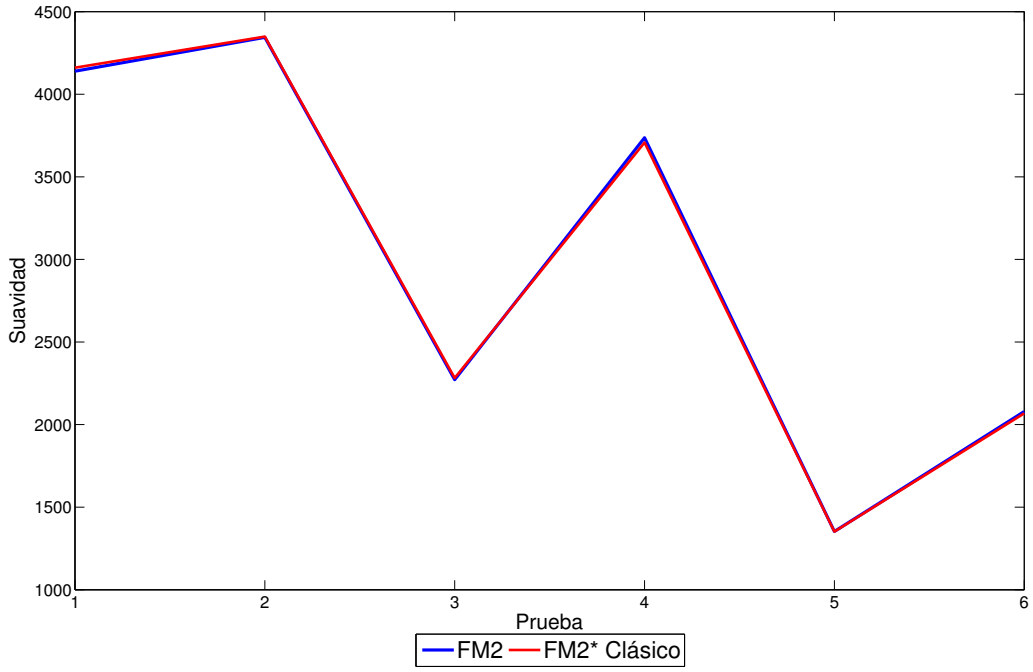


Figura 4.11: *Suavidad de la trayectoria utilizando FM^2* clásico.*

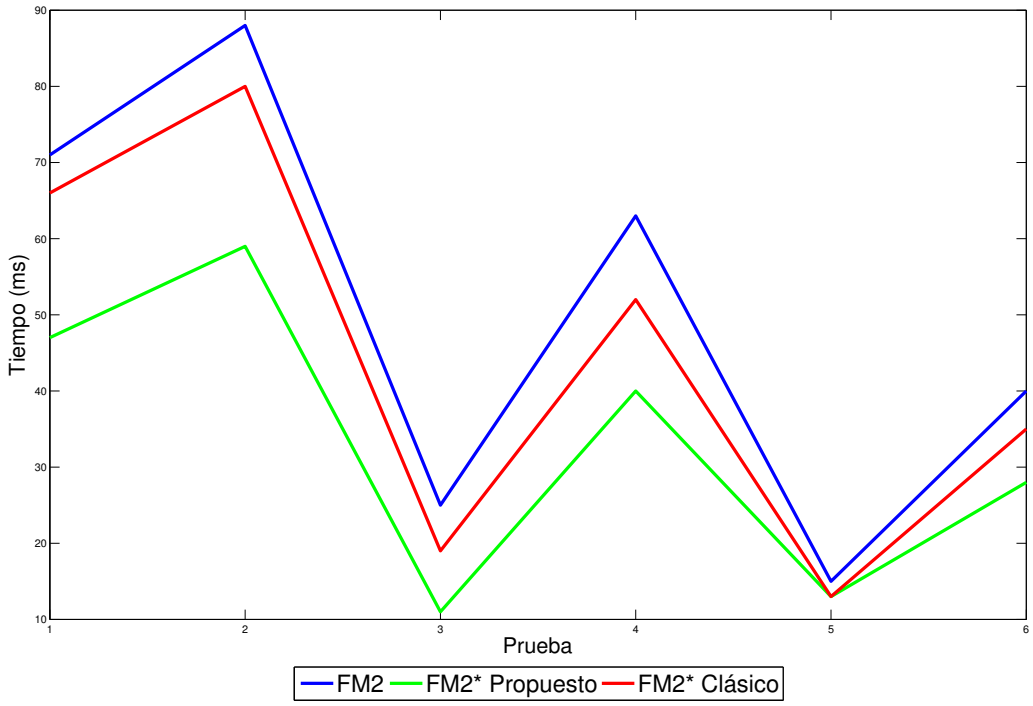


Figura 4.12: *Comparación de tiempo de ejecución de los distintos algoritmos.*

4.4. Conclusiones

En este capítulo se han presentado los fundamentos teóricos de dos aproximaciones al método FM^2*, una publicada en (Valero-Gomez y cols., 2013) y desarrollada en la sección 4.1 y otra desarrollada para este proyecto y presentada en la sección 4.2.

El objetivo final del desarrollo de FM^2* no es otro que obtener la potencia y las características de FM^2, pero disminuyendo el tiempo de cómputo. Para ello, la metodología aplicada consiste en modificar la expansión de la onda de forma que su dirección quede más orientada al objetivo. En ese punto radican las diferencias entre la aproximación a FM^2* clásica y la propuesta en este proyecto. Durante la primera, la onda se expande utilizando una heurística que modifica el tiempo de llegada, pero almacenando también los valores de tiempo de llegada sin modificar de cara a calcular el gradiente que genera la trayectoria posteriormente. En el método propuesto la heurística varía de tal forma que se almacena un único valor.

Durante los comparativas realizadas en la sección 4.3 se obtienen siempre mejores tiempos de cómputo, siendo esto causado por dos razones. La primera es que al manejar una única estructura de datos para el tiempo, los procesos se ven agilizados. La segunda se debe al cambio de heurística, que proporciona un direccionamiento más fuerte y por tanto se llega antes al objetivo. Dicha diferencia de tiempo, además, se hace más patente cuanto más lejos y más fácilmente accesible se encuentra el objetivo con respecto al origen. El punto negativo del nuevo método es un descenso en la suavidad, problema que no resulta grave dado que el descenso de suavidad sigue permitiendo obtener trayectorias de gran suavidad.

Dados todos estos datos, se puede determinar que el método de FM^2* propuesto ofrece unos resultados más óptimos y cumple con las motivaciones para desarrollar una variación de FM^2 de un modo más eficiente.

Capítulo 5

Fast Marching Square Directional Planning Method

En los capítulos 3 y 4 se han descrito y comparado los métodos de planificación de trayectorias FM^2 y FM^{2*}. En este capítulo se va a desarrollar una nueva variación sobre el método FM^2 que busca obtener una trayectoria más óptima en tiempo de recorrido y distancia. Tal y como se explicó en capítulos anteriores, la trayectoria que generan FM^2 y FM^{2*} busca ir siempre por el centro del mapa para ofrecer una mayor seguridad. Pero esta particularidad no siempre es lógica, ya que no tiene en cuenta que existan grandes zonas alejadas de obstáculos por las que circular libremente. Para resolver este problema se plantea una variación que tendrá en cuenta el sentido de avance del robot con respecto a los obstáculos, asumiendo que si el robot se aleja de los obstáculos es posible circular a máxima velocidad sin perder seguridad.

A lo largo del presente capítulo se desarrollará teóricamente el algoritmo y se realizarán comparativas de FM^2 Directional frente a FM^2 y FM^{2*} en términos de velocidad de cálculo, suavidad de la trayectoria, tiempo necesario para recorrer el camino y distancia recorrida.

5.1. Motivaciones

Una de las principales virtudes que plantea FM^2 es lograr que la trayectoria sea suave y se encuentre lo más alejada posible de los obstáculos. Esta virtud no siempre resulta tal, ya que es independiente a la geometría del escenario y los puntos de inicio y final de la trayectoria, haciendo que las trayectorias puedan recorrer un espacio mayor al requerido para ser seguras. En la figura 5.1 se puede ver un ejemplo de lo que sucede en un escenario libre de obstáculos. La trayectoria que intuitivamente se ve más óptima en cuanto a suavidad, distancia y velocidad entre dos puntos es una línea recta que los una. En cambio, como se puede ver en la imagen, la trayectoria no es tal, sino que presenta algunas zonas curvas. El resultado queda justificado por el mapa de velocidades que se genera al expandir el primer potencial, en el que se puede apreciar que en la zona abierta la velocidad se incrementa hacia el centro de la sala, produciendo la curva que se aprecia.

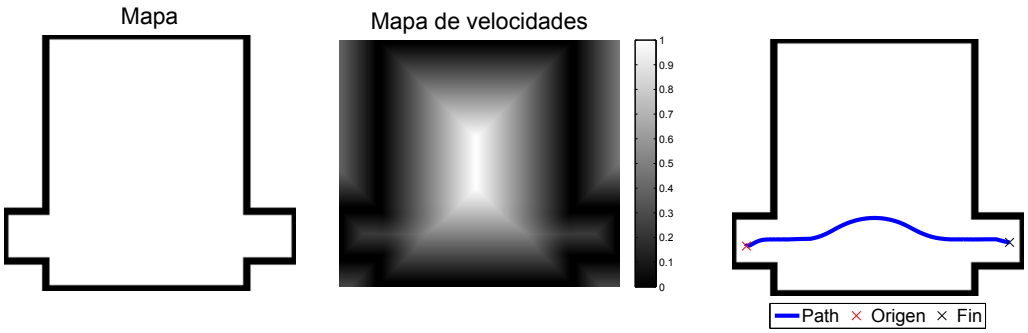


Figura 5.1: *Ejemplo de FM^2 en escenario vacío.*

En la figura 5.2 se puede ver el perfil de velocidades del ejemplo. Como se puede ver, la velocidad aumenta a medida que la trayectoria se acerca al punto medio, donde se puede encontrar la mayor velocidad en el mapa, bajando a medida que se aleja de dicho punto.

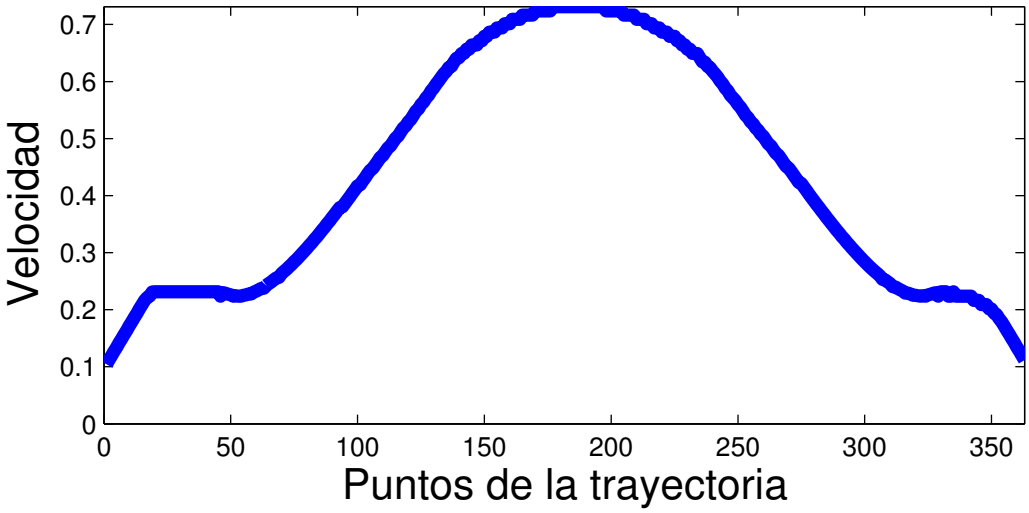


Figura 5.2: *Perfil de velocidades de FM^2 en escenario vacío.*

Aunque los resultados se ajustan a lo que se espera de FM^2, de forma intuitiva podemos pensar que dado que el escenario está libre de obstáculos y la recta que une los puntos de inicio y final no pasa cerca de ninguna pared, el robot podría tomar ese camino recto y circular a una velocidad máxima sin comprometer su seguridad.

Estos motivos justifican el desarrollo de una variedad de FM^2 que tenga en cuenta la geometría del escenario y los puntos de inicio y fin con el fin de obtener una trayectoria más óptima en distancia y tiempo de recorrido. Estas características se tienen en cuenta para modificar las velocidades y obtener una trayectoria más directa con un perfil de velocidades adecuado que permita recorrerla en menos tiempo. Con esta nueva vertiente se busca hacer todo ese proceso durante la planificación y no como un paso posterior, obteniendo de este modo trayectorias y perfiles de velocidad coherentes entre sí.

5.2. Fast Marching Square Directional (FM^2 Directional)

En la sección 5.1 se ha descrito la motivación para desarrollar este método alternativo basado en FM^2. El punto crítico que afecta a la trayectoria es el mapa de velocidades, que condiciona el tiempo de llegada de la onda a cada celda, y por tanto el resultado al aplicar el gradiente. Este mapa de velocidades es calculado en un paso previo y no tiene en cuenta los puntos de inicio y de final de la trayectoria, por lo que no tiene en cuenta los posibles movimientos del robot. Un robot que, aunque se encuentra cerca de los obstáculos, tome un camino que lo aleje de los mismos, puede moverse a velocidad máxima sin resultar peligroso independientemente de la velocidad extraída del mapa de velocidades en dicho punto.

El método propuesto, FM^2 Directional, busca integrar una solución matemática que resuelva ambos problemas. Para ello el método se basa en la comparación de la velocidad de la celda sobre la que se va a resolver la ecuación Eikonal con la de su origen. Dado que la onda viaja en sentido opuesto al camino resultante final, se asume que si la velocidad de celda sobre la que se va a expandir es menor que la de la fuente, la onda está alejándose de los obstáculos y por tanto puede expandirse a máxima velocidad. Esta solución no tiene en cuenta las limitaciones cinemáticas del robot, sino que se utiliza una referencia ideal.

Desarrollando la idea, en FM^2 Directional se propone un método muy similar al utilizado por FM^2 aunque añadiendo algunas variantes. La principal, como se ha comentado anteriormente, consiste en variar la velocidad de expansión de la onda bajo una condición comparativa. Hay que tener en cuenta que ese cambio se produce únicamente al resolver la ecuación Eikonal y que no se modifica el mapa de velocidades en ningún momento, por lo que si la onda pasa por la misma celda desde distintos orígenes la velocidad de expansión puede variar. Como se ha comentado antes, la comparación se realiza desde la celda con el valor de tiempo más bajo que resulta evaluada en la iteración

actual, que llamaremos i_{cell} que tendrá asociada una velocidad v_{cell} según el mapa de velocidades, y la celda desde la que se expande la onda, i_{source}, con una velocidad v_{source} asociada al mapa de velocidades. Con estos datos, teniendo en cuenta que la velocidad de cada celda x se representa como $F(x)$, se determina que:

$$v_{cell} = F(i_{cell})$$

$$v_{source} = F(i_{source})$$

Aplicando la comparativa sobre las velocidades se obtendrá una velocidad, v_{exp} que será utilizada para resolver la ecuación Eikonal.

$$v_{exp} = \begin{cases} 1, & \text{if } v_{source} > v_{cell} \\ v_{cell}, & \text{otherwise} \end{cases}$$

Si la onda se expande utilizando siempre el valor de v_{exp} correspondiente a cada celda la forma en la que lo hace resulta impredecible, por lo que es necesario añadir una capa extra. La onda se expande de forma normal, utilizando v_{cell} en todo momento como se haría con FM^2, pero además se guarda el valor de tiempo de llegada que quedaría utilizando v_{exp}. Sobre los valores de tiempo calculados con v_{exp} se aplica el gradiente para obtener la trayectoria.

Aplicando FM^2 Directional sobre el ejemplo utilizado en la sección 5.1 se obtiene el resultado en la figura 5.3. Como se puede ver la trayectoria resulta más directa comparada con la que ofrece FM^2. El mapa de velocidades mostrado es el utilizado para calcular la trayectoria, y es el resultado de modificar las velocidades al mismo tiempo que se expande la onda. Las velocidades se van modificando a medida que la onda se expande por lo que, por un lado, cuando la onda alcanza el objetivo se para y deja de modificar el mapa. Por otro lado se tiene en cuenta el valor de la velocidad de los vecinos durante la expansión de la onda para comprobar si la onda se expande, o no,

hacia un obstáculo, por lo que la modificación también depende de la dirección de la onda.

Es necesario tener en cuenta también que cuando la onda se expanda por zonas en las que el mapa de velocidades sin modificar tiene una velocidad alta la heurística no modifica la velocidad ya que no se cumple la condición. Como medida de seguridad que evite que las trayectorias se acerquen excesivamente al obstáculo se ha establecido una velocidad relativa mínima de 0.05, a partir de la cual el mapa de velocidades puede modificarse acorde a la heurística.

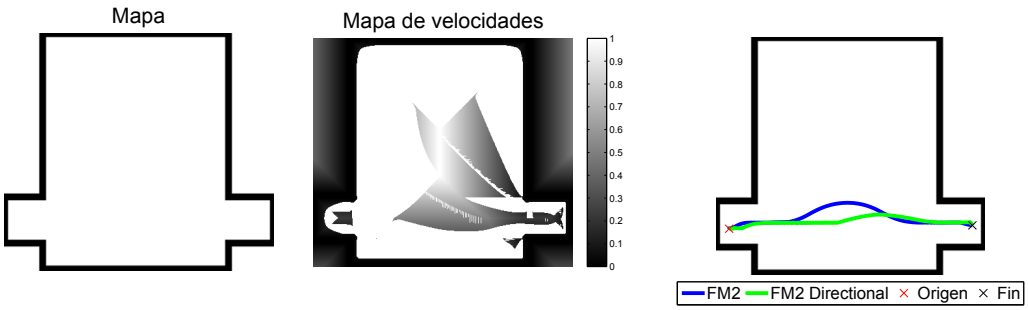


Figura 5.3: *Ejemplo de FM^2 Directional en escenario vacío.*

En la figura 5.4 se muestra el perfil de velocidades. En este perfil se pueden ver varios factores. En primer lugar, la velocidad baja en la llegada a las esquinas dado que son puntos críticos en los que se ha de aumentar la seguridad. En segundo lugar, la onda se expande a máxima velocidad hasta que llega a una zona del mapa de velocidades en la que se tiene velocidad alta, por lo que el perfil no se modifica salvo en algunos picos.

5.2.1. Alternativas propuestas

Una alternativa que resulta intuitiva consiste variar el criterio de comparación utilizado para modificar la velocidad, basándose esta vez en la comparación entre la dirección del gradiente de velocidad de la celda según el mapa de velocidades y el gradiente de dirección de expansión de la onda. De este modo se puede conocer si la onda se expande en el sentido de aumento de

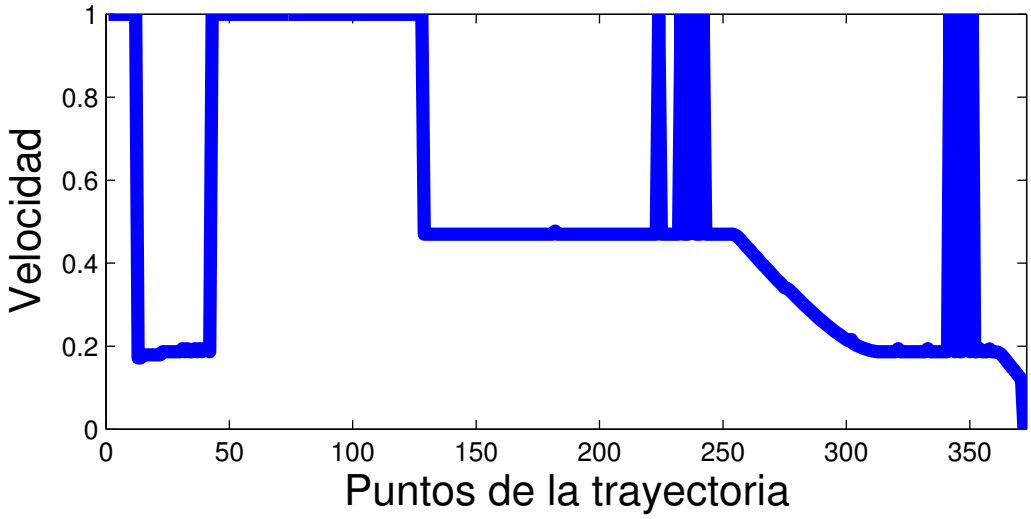


Figura 5.4: *Perfil de velocidad aplicando FM^2 Directional en escenario vacío.*

la velocidad del mapa de velocidades o en el sentido en el que decrece. Con dicha información se puede determinar si la onda se aleja de los obstáculos (sentido de aumento de la velocidad) o se acerca (sentido contrario al aumento de velocidad). Este método tampoco tiene en cuenta las limitaciones cinemáticas del robot, por lo que solo se proporciona una referencia ideal.

En la figura 5.5 se muestra un ejemplo de gradiente de velocidades en una pared. Como se puede apreciar, las flechas apuntan en el sentido contrario a la pared, indicando el sentido de crecimiento de la velocidad.

En la figura 5.6 se muestran los dos ejemplos más extremos de gradiente de velocidad (azul) frente a gradiente de expansión de onda (rojo). En el primer caso el sentido es completamente opuesto, lo que implica que la onda se expande hacia la zona libre dado que el camino final irá en sentido opuesto y es seguro que el robot pase esa zona a máxima velocidad. En cambio en el segundo ejemplo la onda y la velocidad se expanden en el mismo sentido, lo que significa que la onda se expande hacia un obstáculo y es necesario establecer un sistema de seguridad basado en la velocidad del mapa de velocidades.

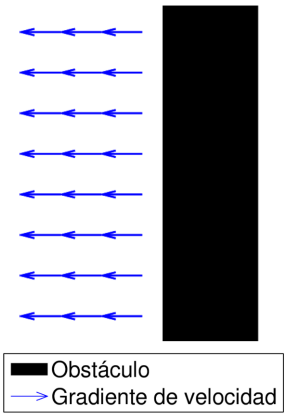
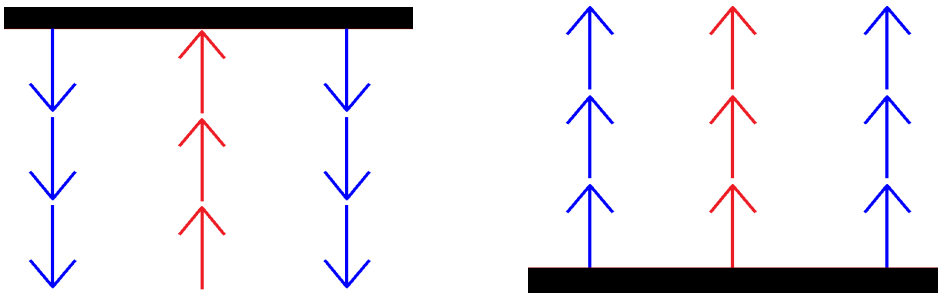


Figura 5.5: *Gradiente de velocidades en una pared.*

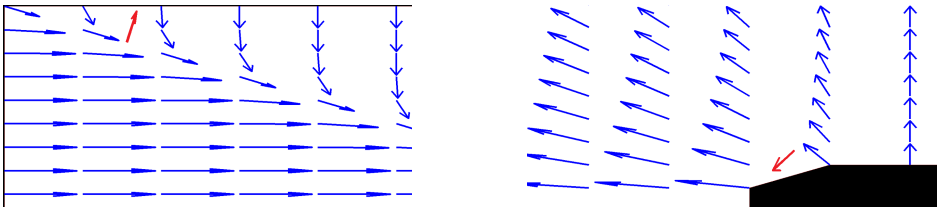


(a) *Ejemplo 1.* **(b)** *Ejemplo 2.*

Figura 5.6: *Comparativa de gradiente de expansión de la onda frente a gradiente de velocidad.*

El valor de esa diferencia determina si se resuelve la ecuación Eikonal con el valor de velocidad del mapa de velocidades o a máxima velocidad. En la figura 5.6 se ha mostrado un ejemplo intuitivo en el cual, dependiendo del ángulo de diferencia, se puede determinar si la onda se expande a máxima velocidad o no. No obstante los ejemplos mostrados se encuentran en los casos más extremos (0 y 180º) que no se darán por norma general. El caso ideal expandiría a máxima velocidad cuando la diferencia de los ángulos de los gradientes fuese 180º (es decir, la velocidad y la onda se expanden en sentido totalmente opuesto) pero no es viable debido a la discretización inherente del

mapa de velocidades. Para ello se ha de proponer un rango de ángulos aceptables, y es el motivo por el que se ha de descartar el método. La casuística en este problema es infinita por lo que en función de la geometría se pueden dar casos en los que un rango de diferencia de entre 90º y 180º se puede considerar válido (caso de la figura 5.5). Al igual, se pueden dar otros casos en los que incluso un rango muy reducido puede hacer que se modifique la velocidad de expansión a pesar de que la onda se expanda hacia el obstáculo, especialmente en las zonas próximas a las esquinas de los obstáculos. Este último caso se muestra en la figura 5.7, en la que aplicando un criterio con un ángulo situado entre 90º y 180º como se propone anteriormente la onda se expandiría hacia el obstáculo a máxima velocidad. Dado que los métodos que se utilizan buscan ser generalizables no es una solución válida. Este problema hace que esta implementación de FM^2 Directional no sea una solución válida actualmente.



(a) *Ejemplo 1.* **(b)** *Ejemplo 2.*

Figura 5.7: *Ejemplos de gradiente de velocidad en esquinas.*

En la figura 5.8 se muestra un ejemplo de esta alternativa. En la trayectoria generada se puede apreciar como en la mayor parte de las zonas cercanas a esquinas se producen picos o cambios de inclinación muy fuertes. Además, tanto las trayectorias como los mapas de velocidad tienen mucho ruido en las zonas en las que el ángulo que forman el gradiente de expansión de la onda y el gradiente de velocidades se encuentran bordeando el límite establecido. Como se puede ver, la solución es muy inconsistente y necesitaría de un procesado posterior para suavizar estos detalles, rompiendo así la motivación de integrar todos los

pasos en el propio planificador.

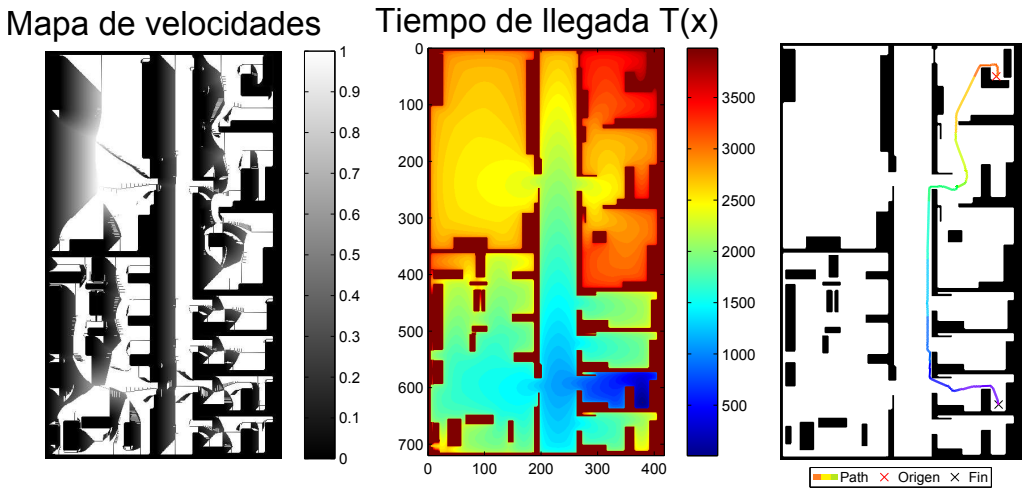


Figura 5.8: *Ejemplo de alternativa propuesta a FM^2 Directional.*

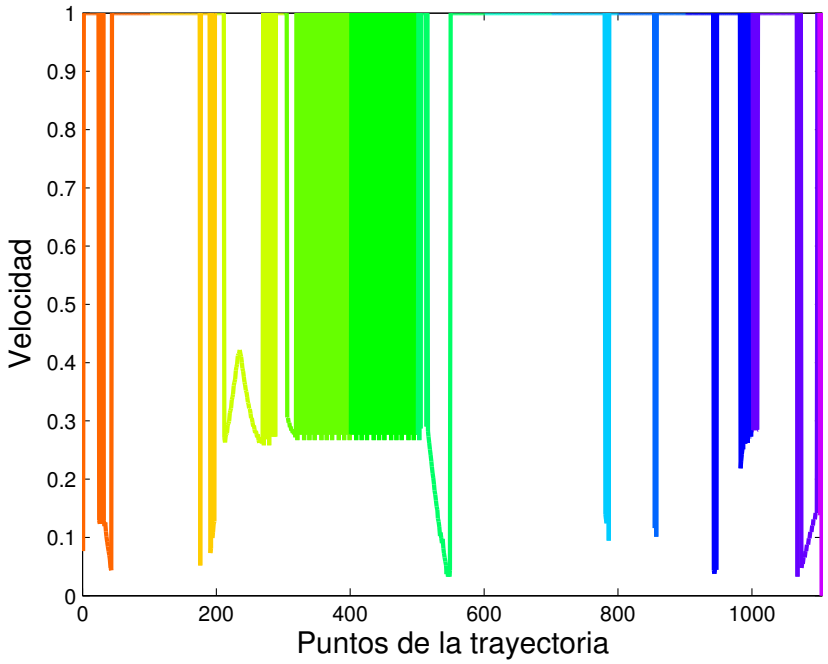


Figura 5.9: *Ejemplo de perfil de velocidad de alternativa propuesta a FM^2 Directional.*

Otra alternativa en la que se ha trabajado ha sido el desarrollo de un método que aúne las virtudes de FM^{2*} y FM^2 Directional, llamado FM^{2*} Directional. Como se ha explicado anteriormente, al expandir la onda utilizando FM^2 Directional se almacenan dos tiempos de llegada: uno utilizado para expandir la onda y otro para aplicar el gradiente y obtener la trayectoria. El tiempo de llegada utilizado para expandir la onda se calcula como en FM^2, es decir, utilizando la velocidad del mapa de velocidades. Mientras, el utilizado para generar la trayectoria se calcula con la velocidad obtenida por la heurística. Este método propone aplicar la misma heurística que la utilizada en FM^{2*} para calcular la expansión de la onda y obtener así unos resultados prácticamente idénticos a los de FM^2 Directional pero rebajando el tiempo de ejecución al expandir la onda de una forma más direccionada al objetivo.

Los resultados generales de FM^{2*} Directional son análogos a los obtenidos en la sección 4.3.2 al comparar FM^{2*} frente a FM^2. Por tanto, se obtiene una trayectoria prácticamente idéntica a la de FM^2 Directional rebajando además de forma notable los tiempos de ejecución frente a FM^2 Directional.

Esta mayor direccionalidad en la expansión de la onda potencia consigo la presencia cambios bruscos de velocidad. La gestión de la heurística, en la que se sustituye la velocidad de una celda por la máxima en caso de cumplir las condiciones, hace que existan saltos bruscos en los tiempos de llegada de la onda en lugar de unos saltos progresivos como sucede en FM^2. Aunque esto ocurre en cualquier aproximación realizada de FM^2 Directional, en este caso se ve notablemente acentuado por la mayor direccionalidad de la onda. Este evento se puede traducir en casos en los que el método no sea capaz de encontrar una solución al aplicar el gradiente, por lo que el método no resulta estable y robusto y no se puede aplicar de forma general.

5.3. Comparativa entre FM^2 y FM^2 Directional

En esta sección se realizará una comparativa entre los métodos de planificación de trayectorias FM^2, explicado en la sección 3.2, y FM^2 Directional, detallado en la sección 5.2. La comparativa consistirá en realizar una trayectoria que enlace el mismo punto de inicio y final con ambos métodos, y sobre el resultado comparar la expansión de la onda y el mapa de velocidades obtenido con cada uno de ellos. Posteriormente se realizará un análisis de tiempo de recorrido de la trayectoria, de la distancia recorrida y de la suavidad. Este análisis viene motivado porque se asume una pérdida de suavidad en las trayectorias de FM^2 Directional y es necesario cuantificar cuanta se pierde frente a la suavidad de FM^2. Finalmente se analizará el tiempo empleado para la segunda expansión de la onda, y así conocer el coste computacional del método.

Para el análisis de tiempo de recorrido de la trayectoria se ha utilizado una aproximación basada en el sumatorio de tiempos que emplea en recorrer el robot cada segmento basándose en la distancia entre puntos de la trayectoria y la velocidad del perfil de velocidades acorde a la ecuación 5.1.

$$tiempo = \sum_{i=1}^{n} \frac{\sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2}}{v_i} \tag{5.1}$$

Para el análisis de la distancia recorrida se ha utilizado una aproximación basada en el sumatorio de distancias entre puntos de la trayectoria descrito en la ecuación 5.2.

$$distancia = \sum_{i=1}^{n} \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2} \tag{5.2}$$

Para comparar el tiempo de ejecución se ha realizado una implementación siguiendo el mismo criterio que el descrito en la sección 4.3.1. En este caso, los elementos de mayor importancia a analizar en este método son la suavidad y el tiempo de recorrido, que resultará independiente al hardware.

Todas las pruebas se realizarán sobre un mismo mapa con su mapa de velocidades asociado, mostrado en la figura 5.10.

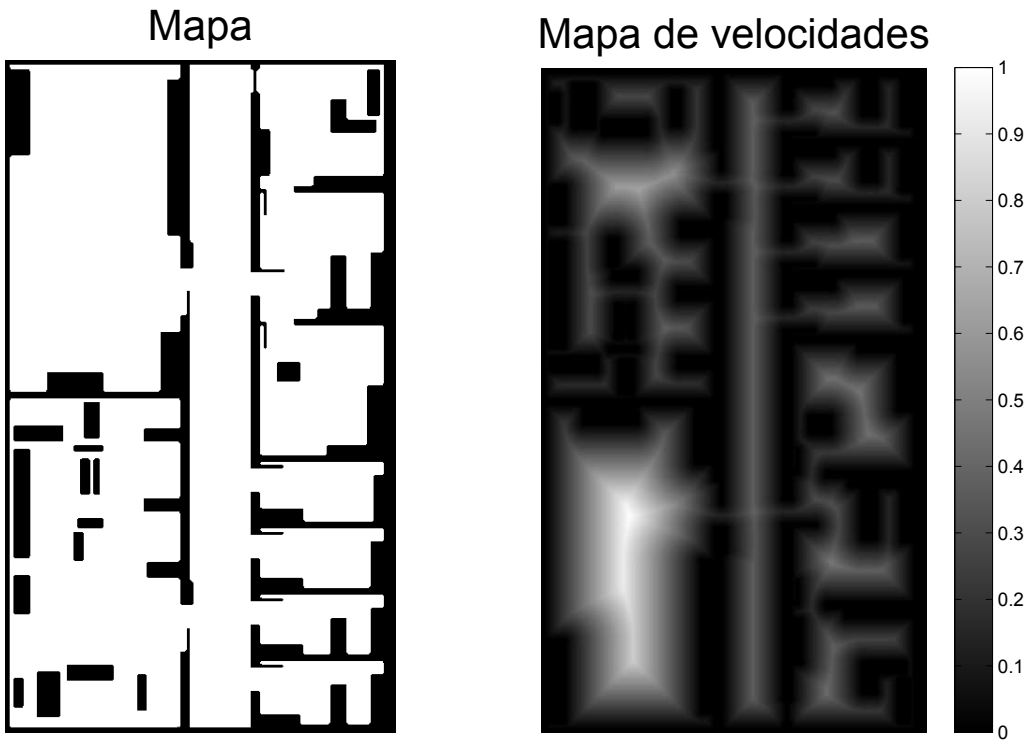


Figura 5.10: *Mapa y mapa de velocidades utilizado para las pruebas.*

5.3.1. Resultado de las pruebas

Durante esta sección se realizarán pruebas sobre un mapa y varios puntos de inicio y final. En dichas pruebas se podrá evaluar la expansión de la onda, las trayectorias generadas y perfiles de velocidad en cada caso. Al final de los ensayos se realizará un análisis de tiempo de recorrido de la trayectoria y de suavidad, concluyendo con un análisis de tiempo de cómputo. Se han realizado 22 ensayos para poder obtener resultados más robustos de suavidad y tiempo de recorrido, pero solo se mostrarán en detalle los que ayudan mejor a comprender las ventajas del algoritmo.

Para aportar una mayor claridad en la lectura de los resultados se ha optado por variar el color de la trayectoria y del mapa de velocidades cada 100 puntos con el fin de poder realizar una comparación más sencilla entre ambas.

En la primera prueba, mostrada en la imagen 5.11, se puede ver como la expansión de la onda resulta idéntica ya que se generan con los mismos datos, no así el mapa de velocidades resultante. El mapa de velocidades resultante sigue los criterios descritos en la sección 5.2.

La trayectoria generada, por su parte, es radicalmente diferente en ambos casos. Mientras que FM^2 busca siempre un camino situado de forma equidistante a todos los obstáculos, con una alta suavidad, FM^2 Directional pasa más cerca de los obstáculos pero acorta notablemente el camino, manteniendo parte de la suavidad característica de las trayectorias de FM^2. De este modo se obtiene una trayectoria acorde a la geometría real del mapa, evitando un exceso de seguridad para realizar caminos más cortos.

La comparación con respecto a los perfiles de velocidad mostrados en la figura 5.12 resulta evidente. Mientras que en el caso de FM^2 se obtienen velocidades que no superan el valor de 0.5 relativo dada la zona por la que ha de circular el robot, en FM^2 Directional el perfil de velocidad obtenido se encuentra la mayor parte del tiempo a máxima velocidad. Comparando los colores de la trayectoria y del perfil de velocidad se puede comprobar como la velocidad se mantiene al máximo por zonas libres de obstáculos, y baja a niveles más seguros en los tramos próximos a los obstáculos que ha de evitar.

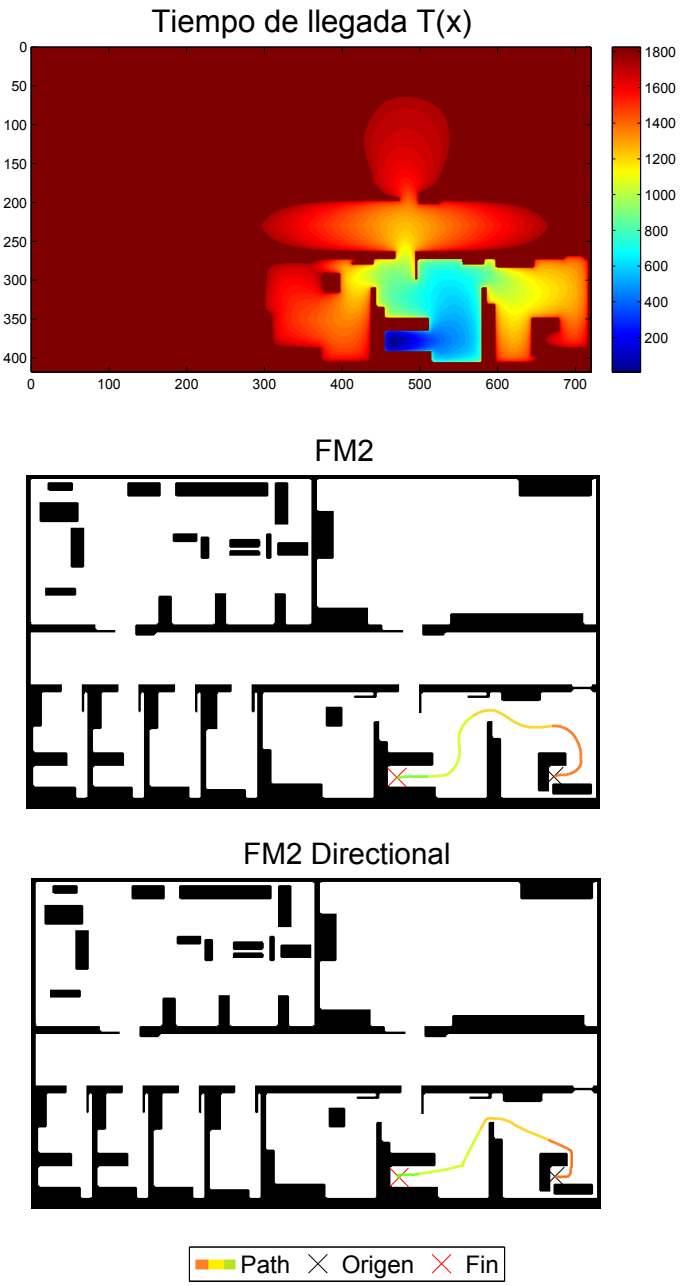
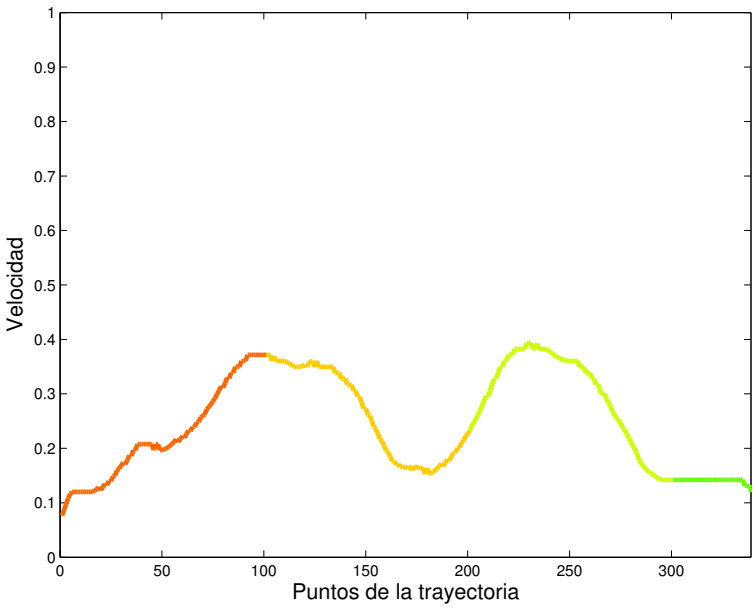
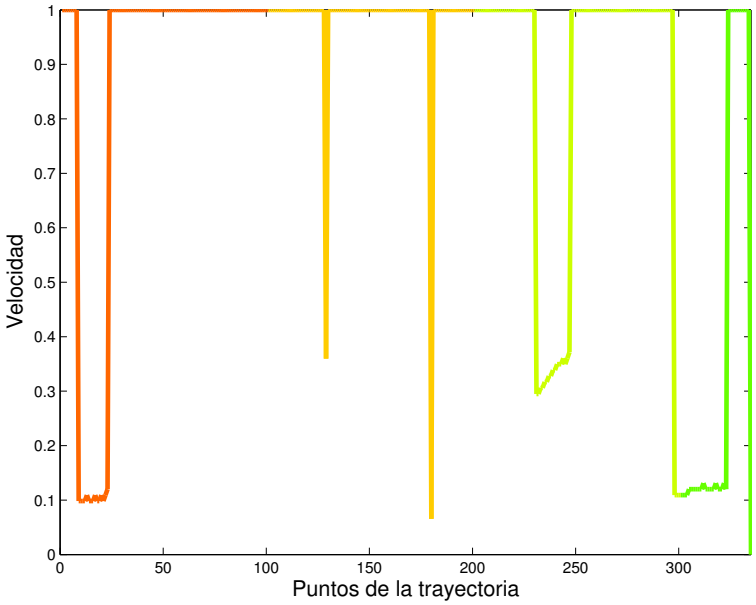


Figura 5.11: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 1.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.12: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 1.*

En las siguientes pruebas, correspondientes a las imágenes 5.13, 5.15, 5.17, 5.19, 5.21, 5.23, 5.25 y 5.27, se puede puede comprobar como las trayectorias generadas vuelven a resultar muy diferentes en todos los casos. Las trayectorias propuestas en FM^2 Directional pasan más cerca de los obstáculos aún manteniendo una distancia de seguridad suficiente y desarrollan trayectorias más directas.

La comparación de los perfiles de velocidad mostrados en las figuras 5.14, 5.16, 5.18, 5.20, 5.22, 5.24, 5.26 y 5.28 resultan muy diferentes. Mientras que en el caso de FM^2 se obtienen velocidades en torno al valor de 0.5 relativo, obteniendo raramente un valor alto de velocidad, en FM^2 Directional el perfil de velocidad obtenido se encuentra la mayor parte del tiempo a máxima velocidad. Realizando la comparación a partir de los colores de la trayectoria y del perfil de velocidad se puede comprobar como la velocidad se mantiene al máximo por zonas libres de obstáculos, y baja a niveles más seguros en los tramos próximos a los obstáculos que ha de evitar.

Aún así existen picos en los perfiles de velocidad propiciados por los puntos límites de la heurística. Estos puntos aparecen en menor cantidad que en la alternativa propuesta a FM^2 Directional y pueden ser atenuados por el seguidor de trayectorias del robot. Las trayectorias generadas por FM^2 o uno de sus derivados tiene una densidad de puntos muy alta por lo que el propio seguidor filtra el número de puntos y los reduce. Tras este proceso los picos quedan atenuados y posteriormente la propia dinámica del robot se encargaría de suavizar aún más la trayectoria. No obstante, y pese a que el objetivo del método es integrar todos los procesos durante la planificación, se podría aplicar un filtro posteriormente que suavizara el perfil de velocidades.

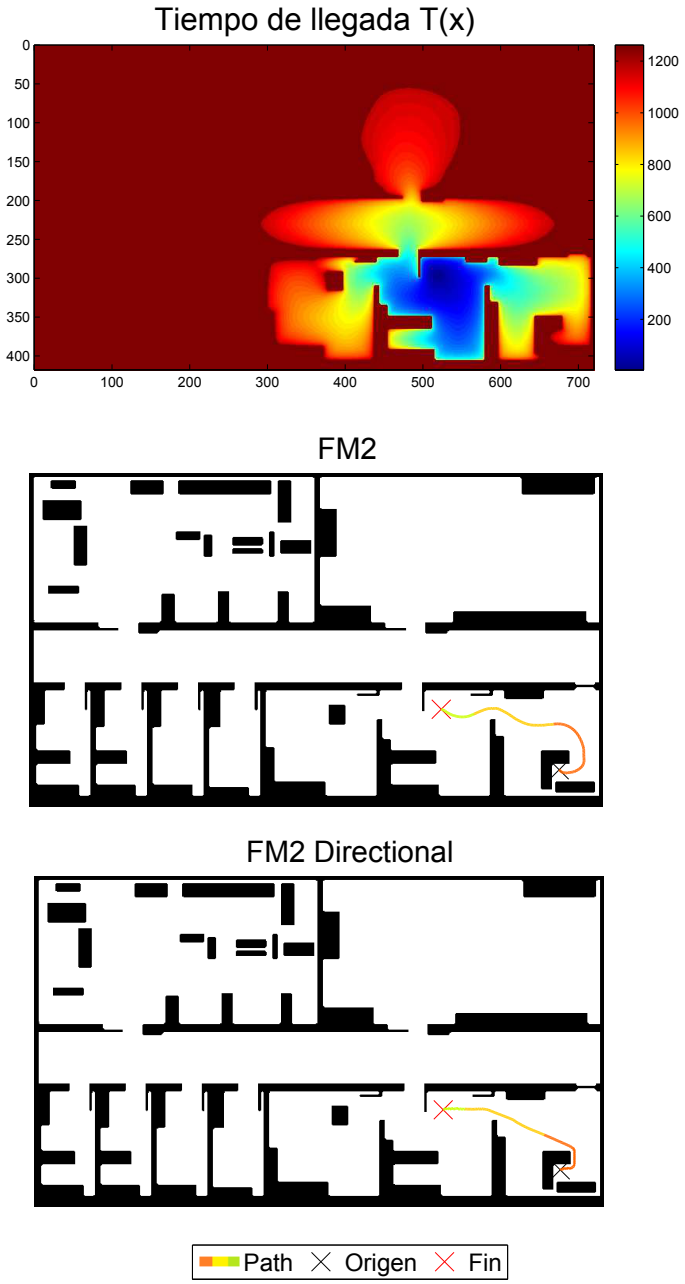
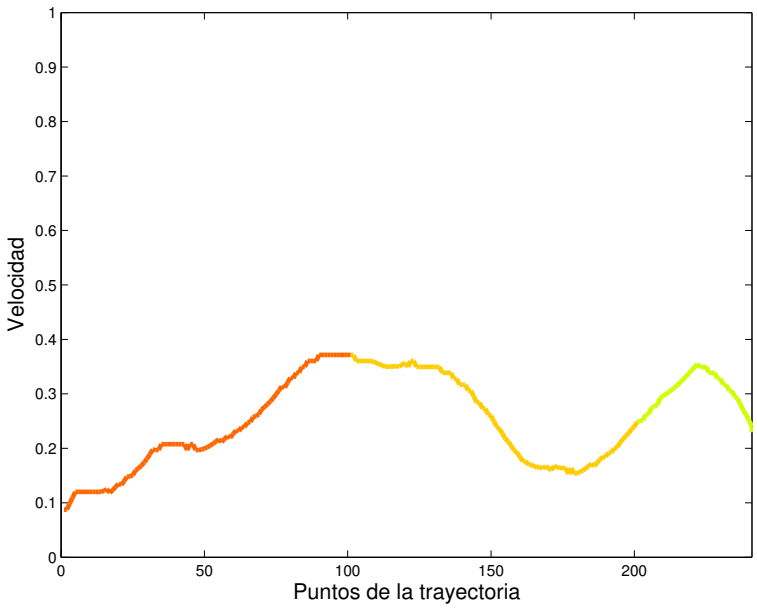
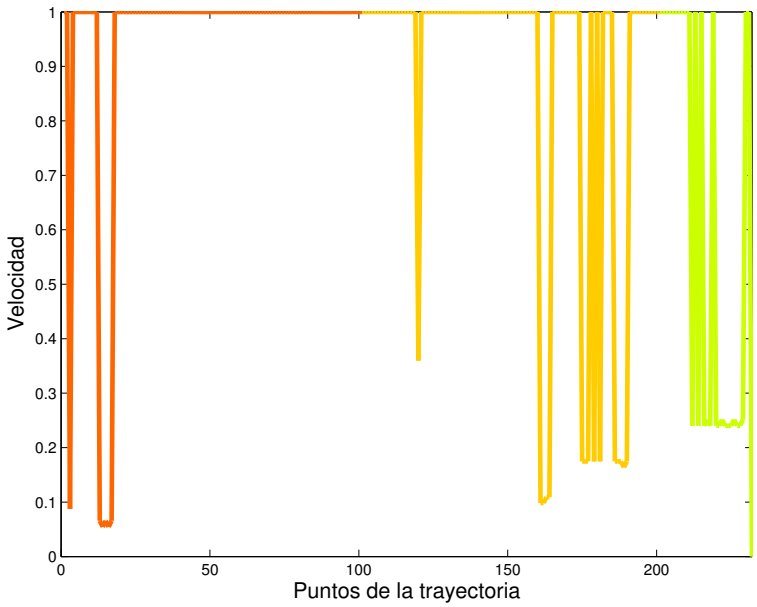


Figura 5.13: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 4.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.14: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 2.*

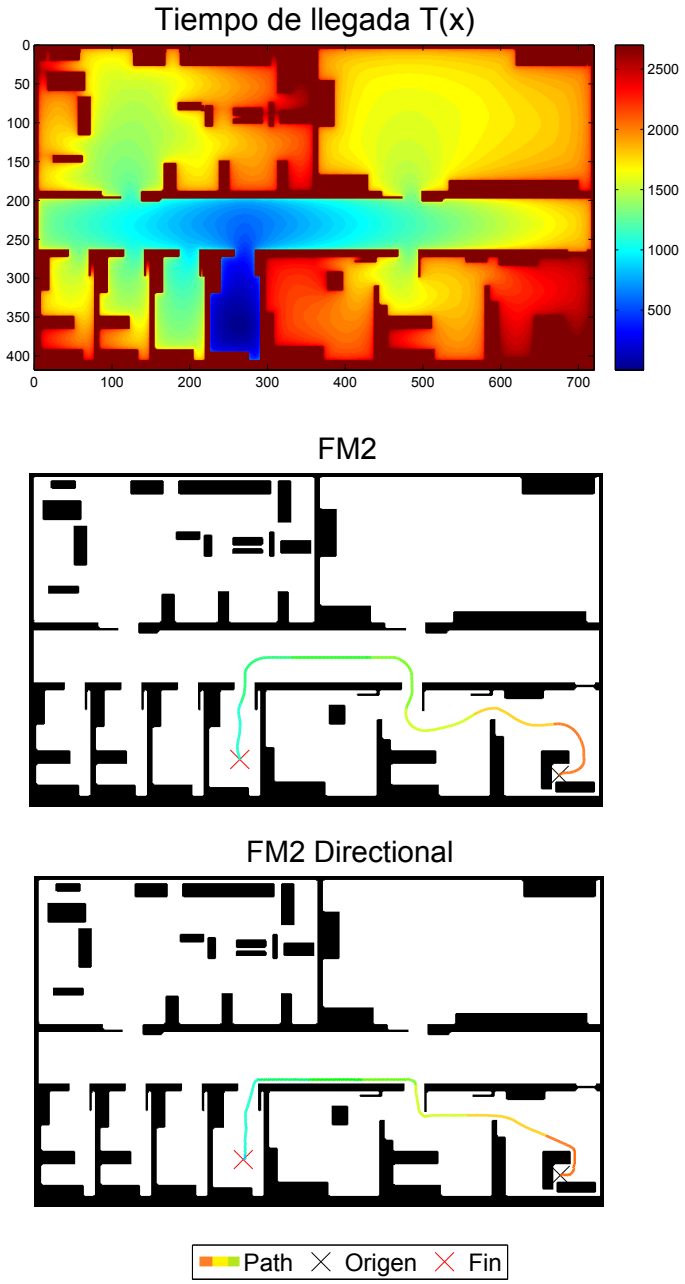
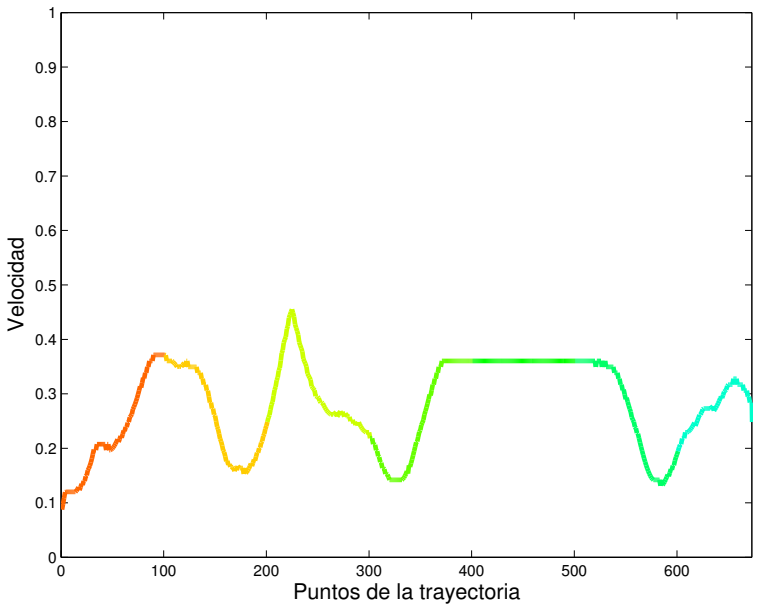
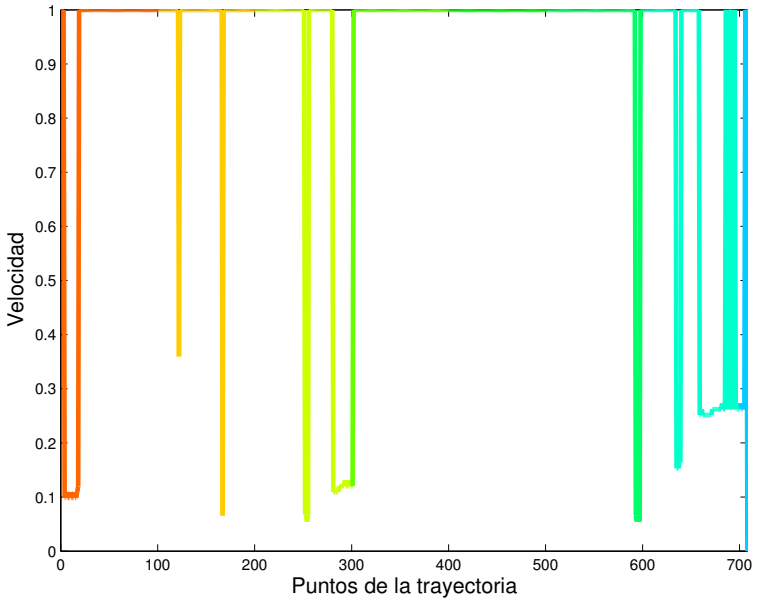


Figura 5.15: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 5.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.16: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 3.*

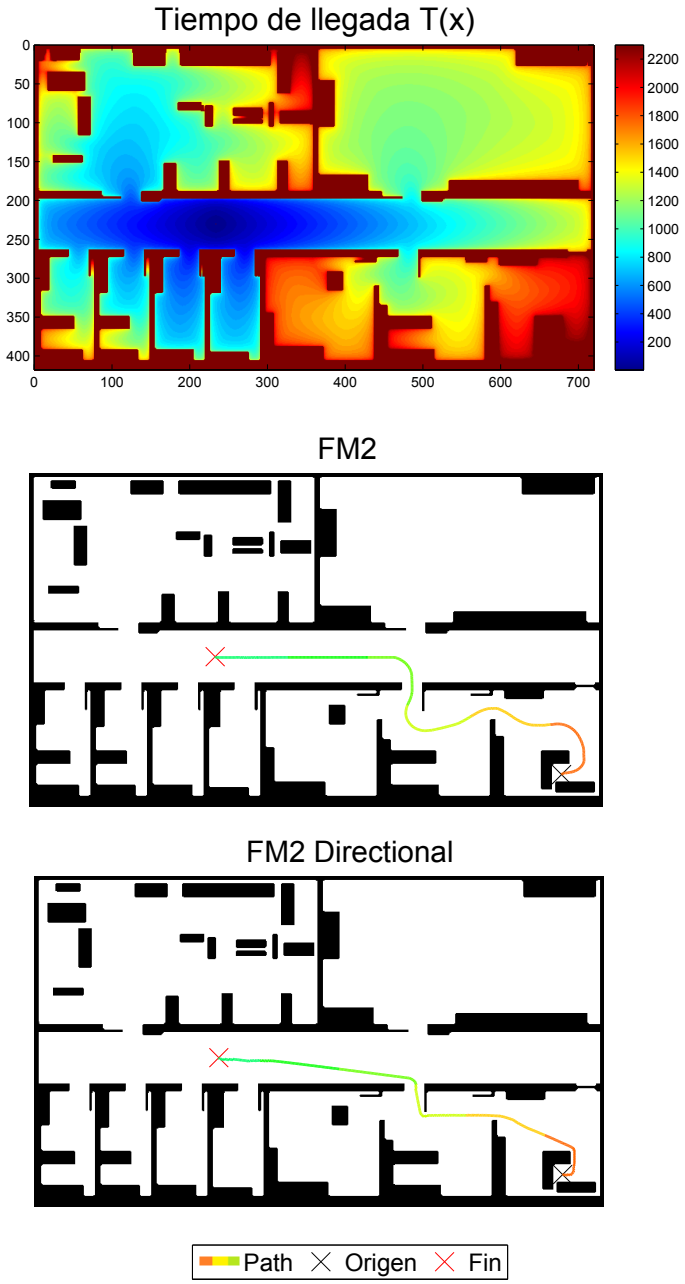
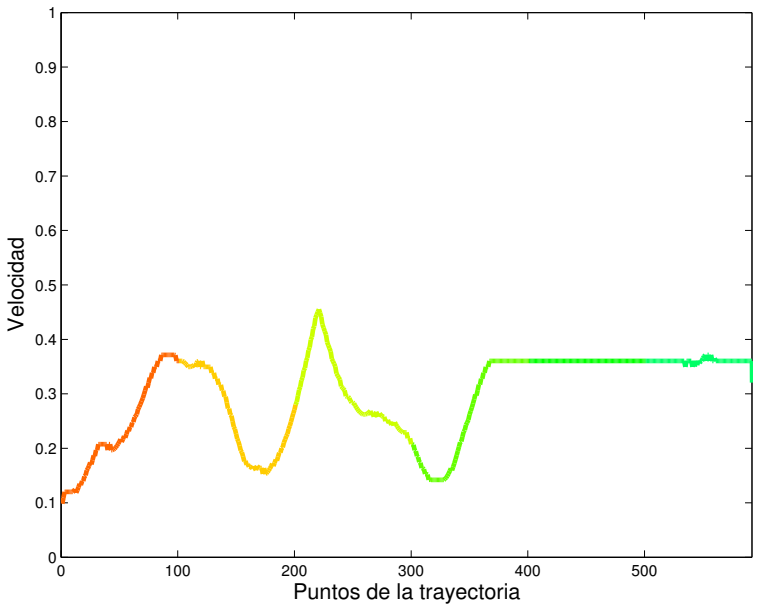
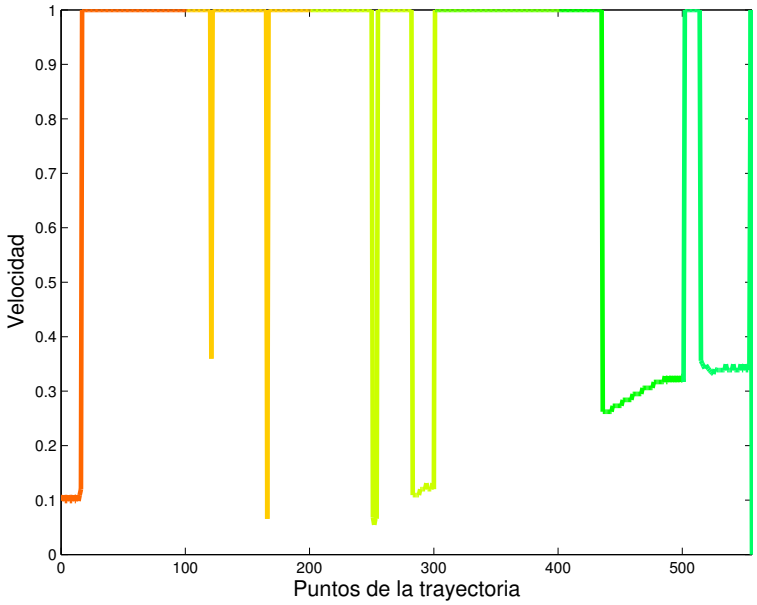


Figura 5.17: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 6.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.18: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 4.*

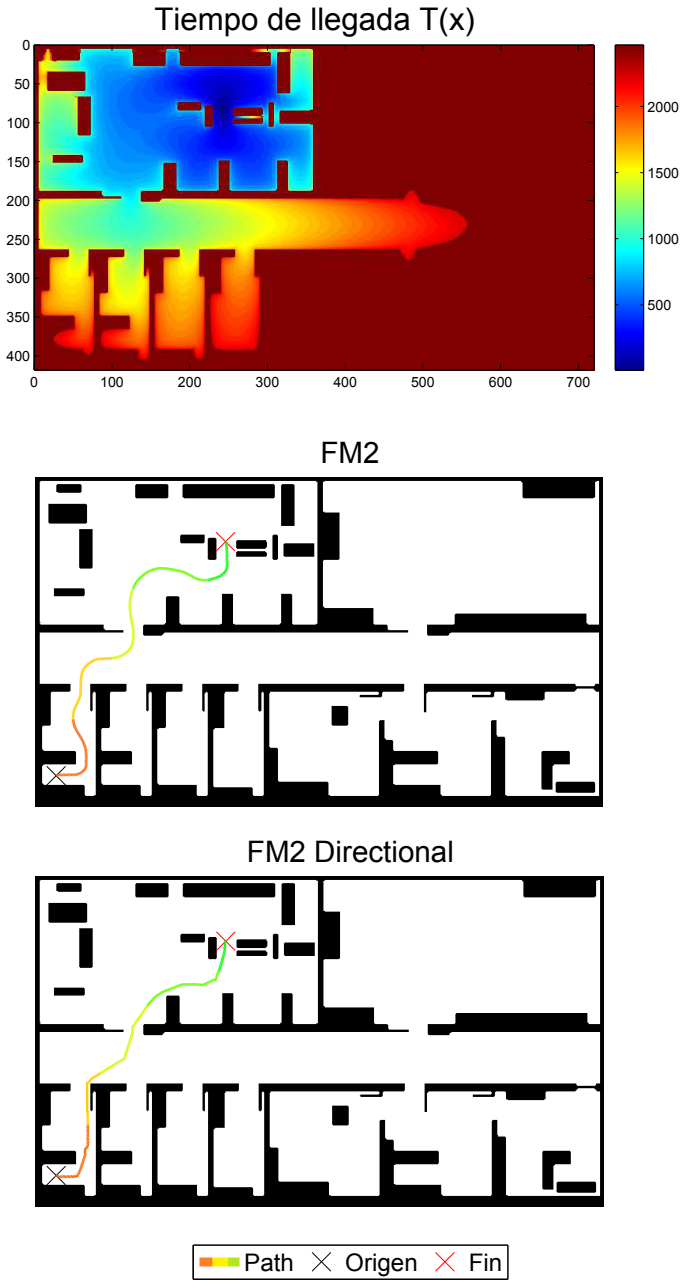
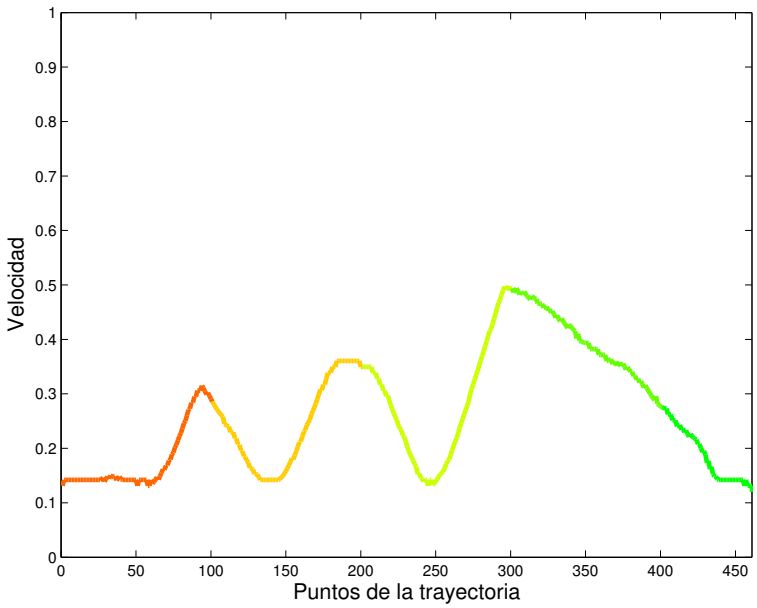
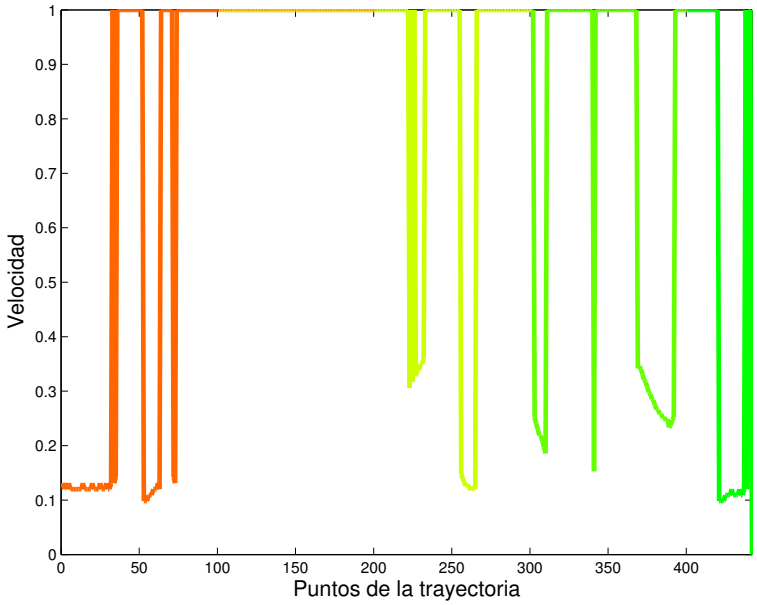


Figura 5.19: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 10.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.20: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 5.*

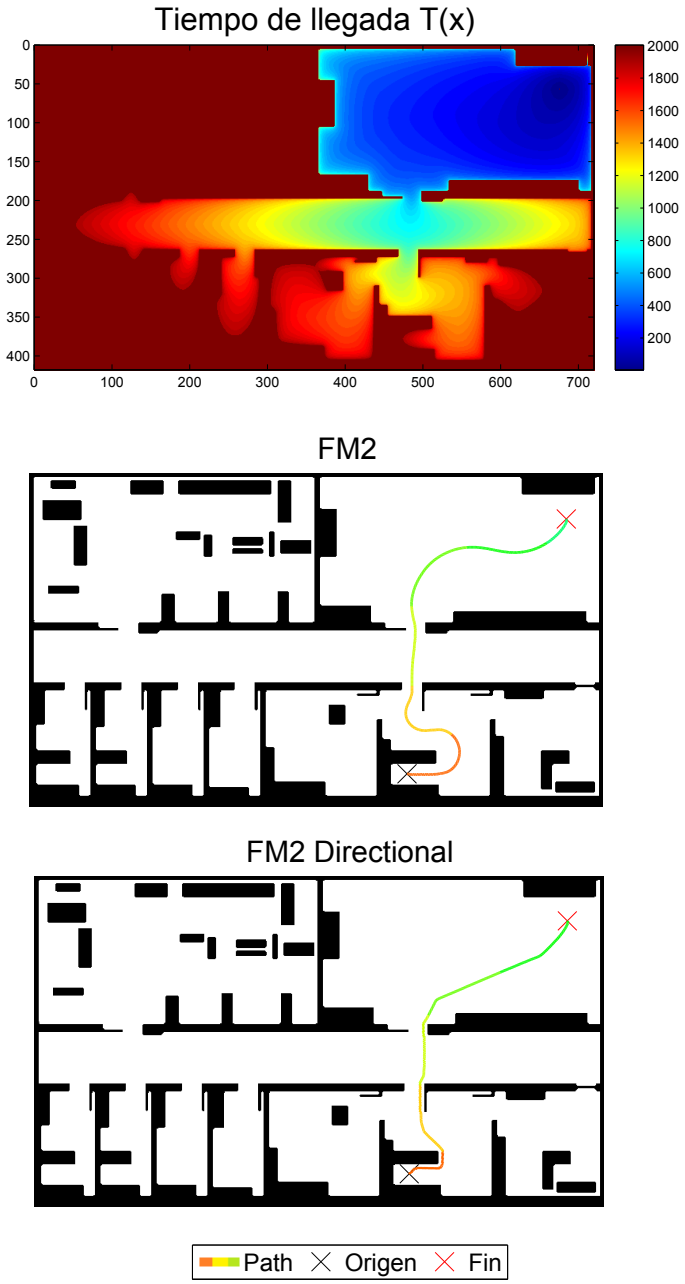
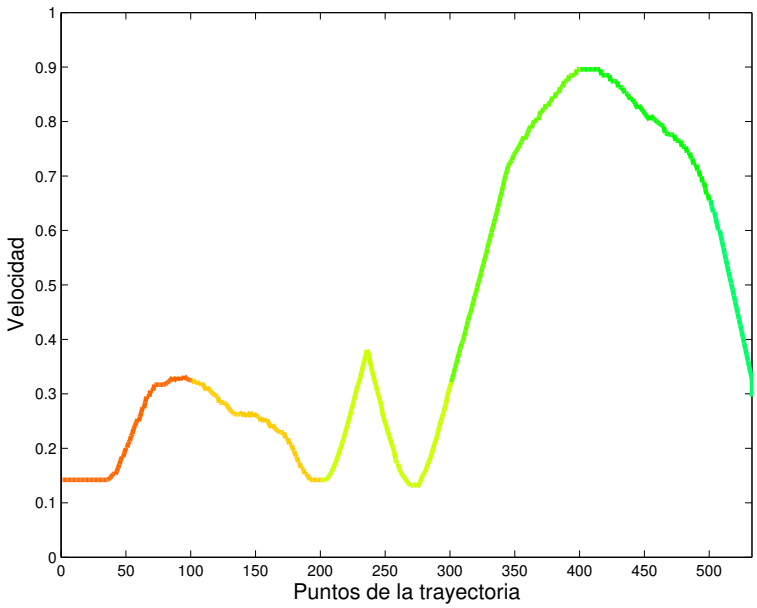
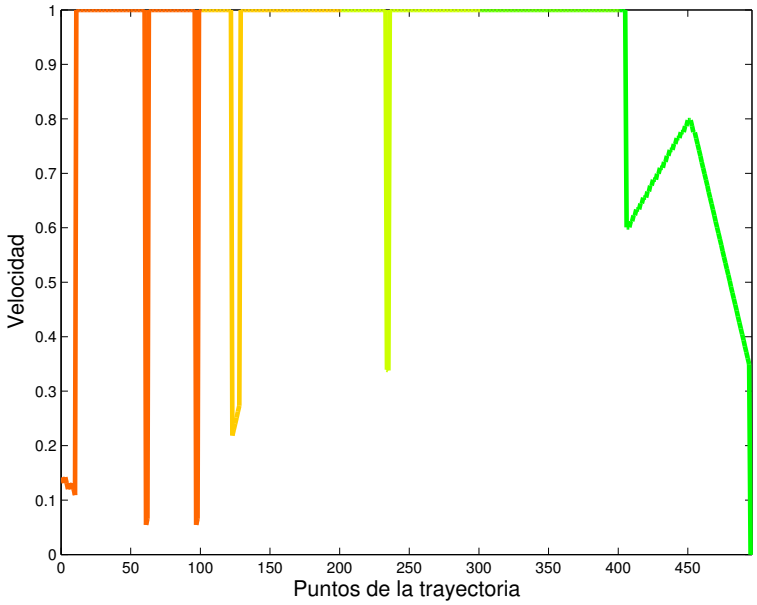


Figura 5.21: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 11.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.22: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 6.*

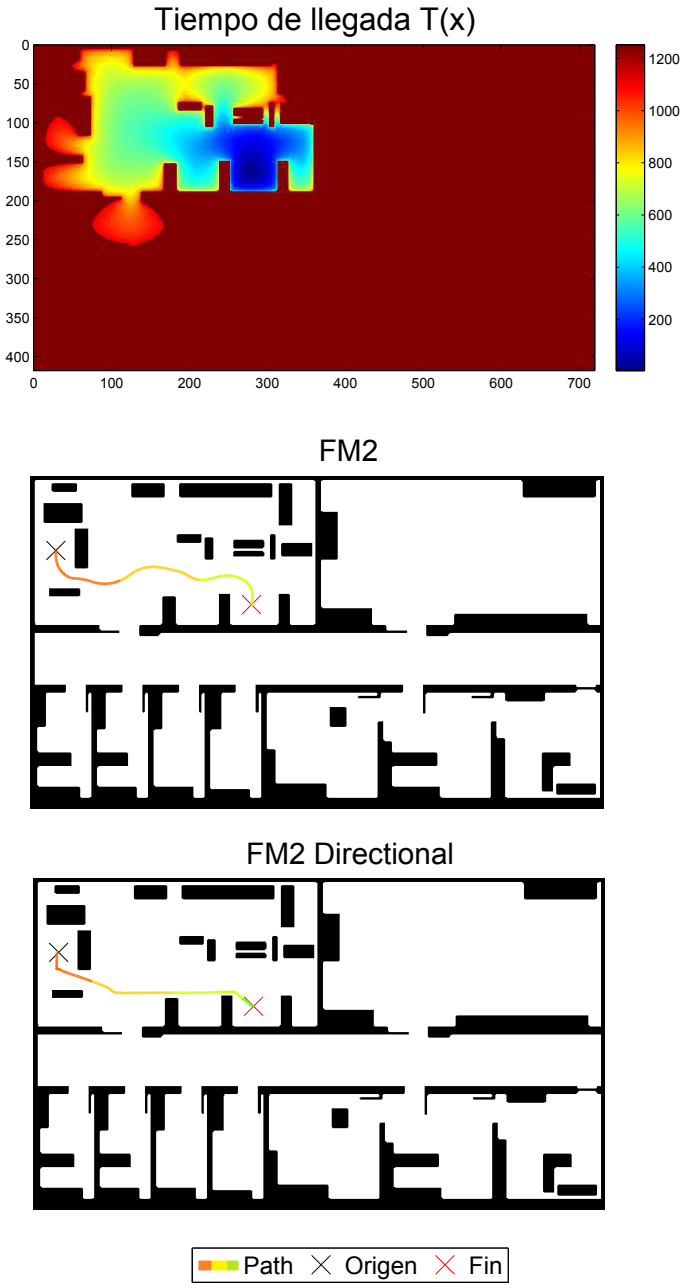
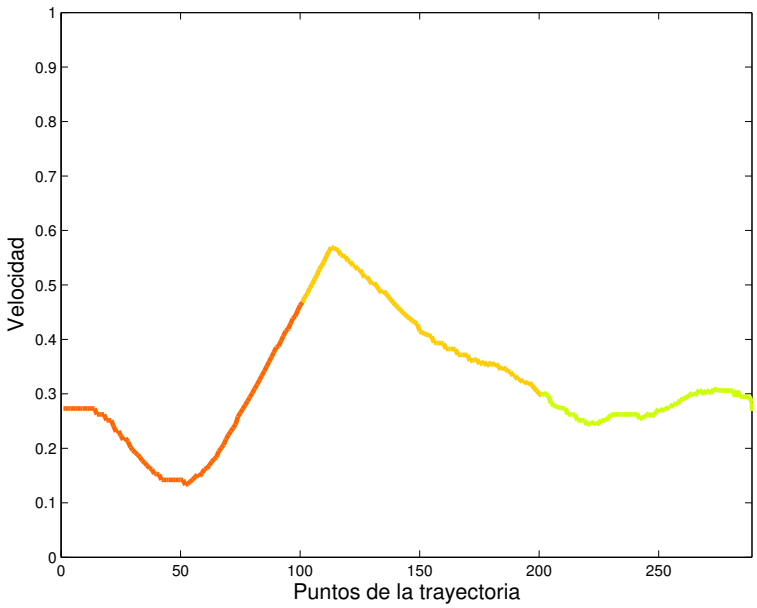
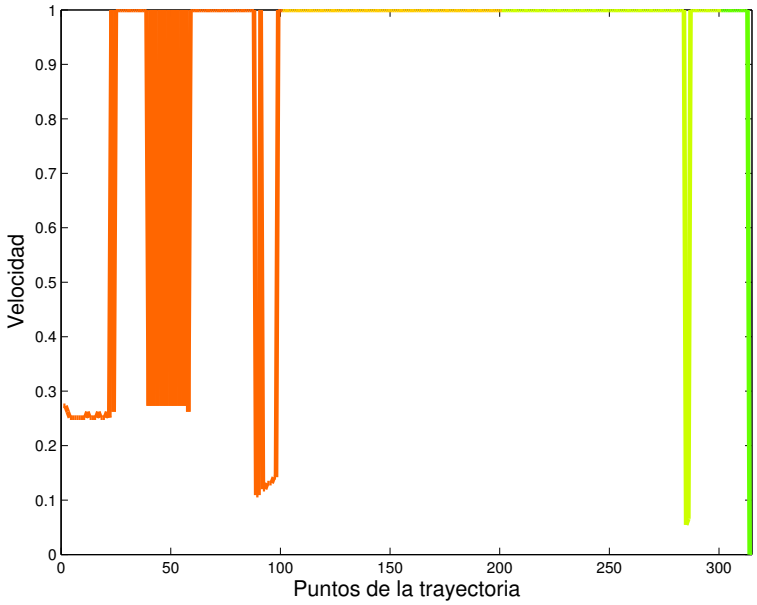


Figura 5.23: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 12.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.24: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 7.*

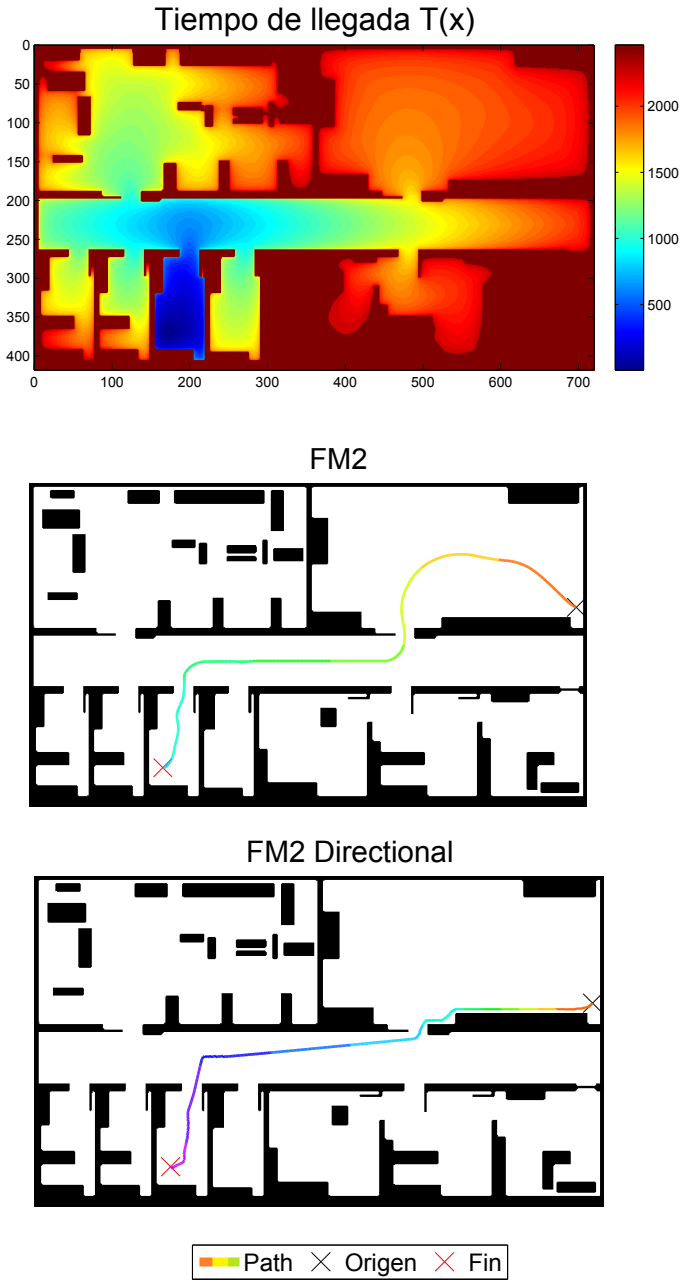
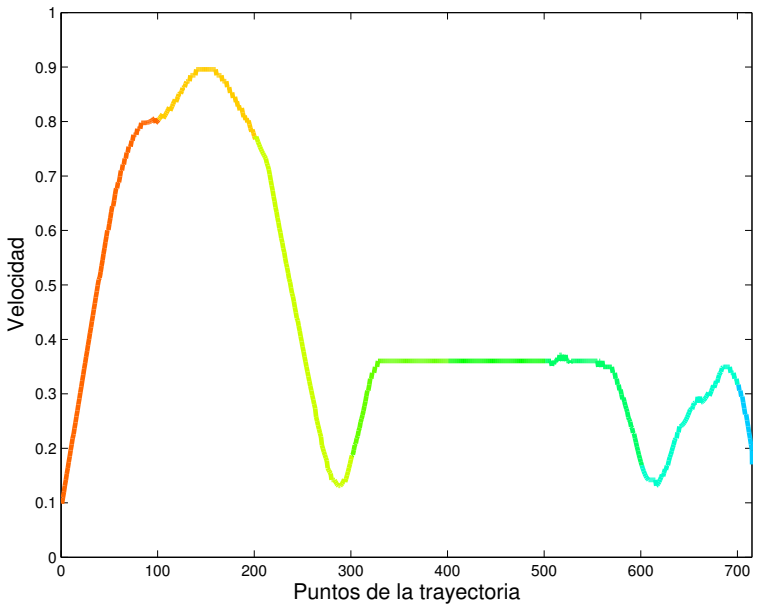
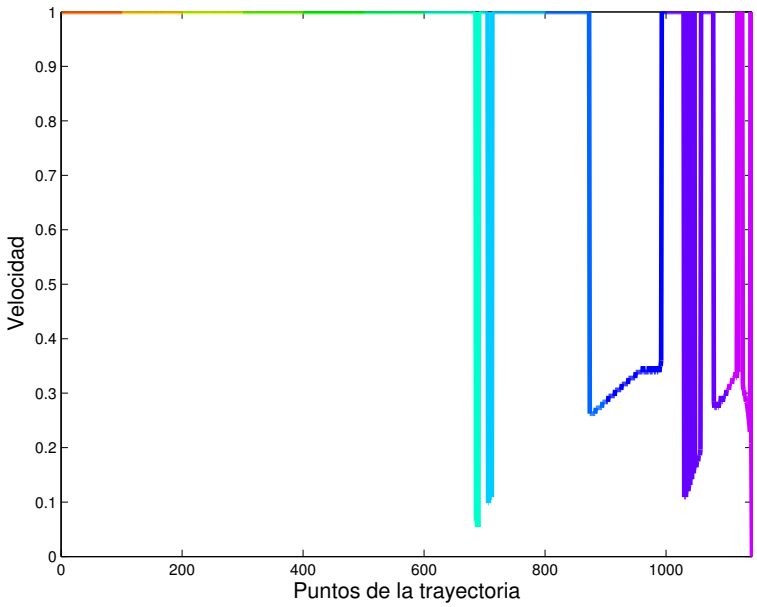


Figura 5.25: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 13.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.26: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 8.*

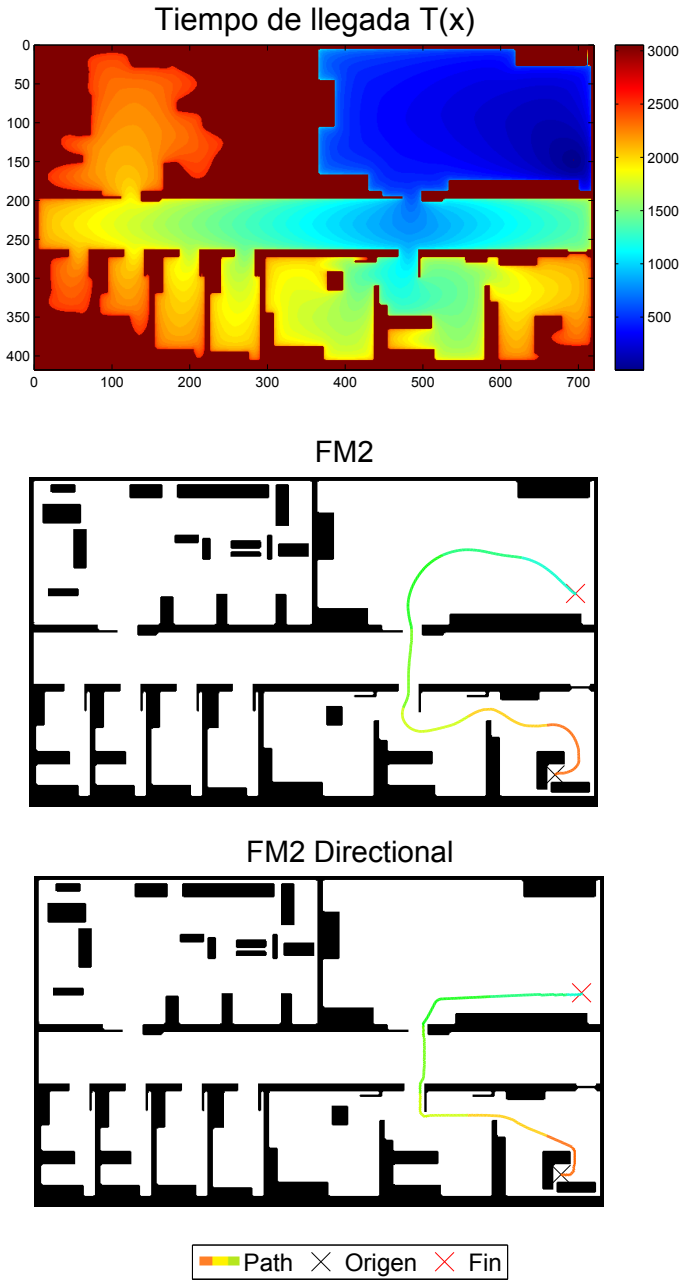
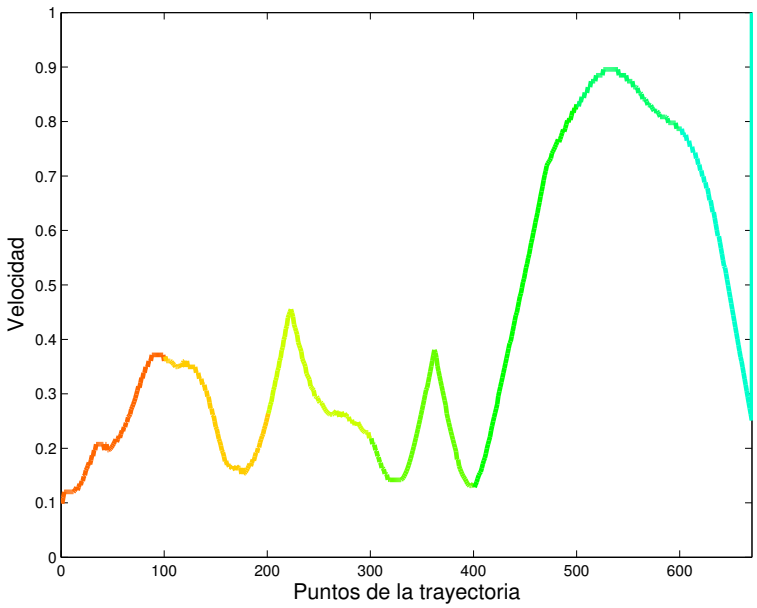
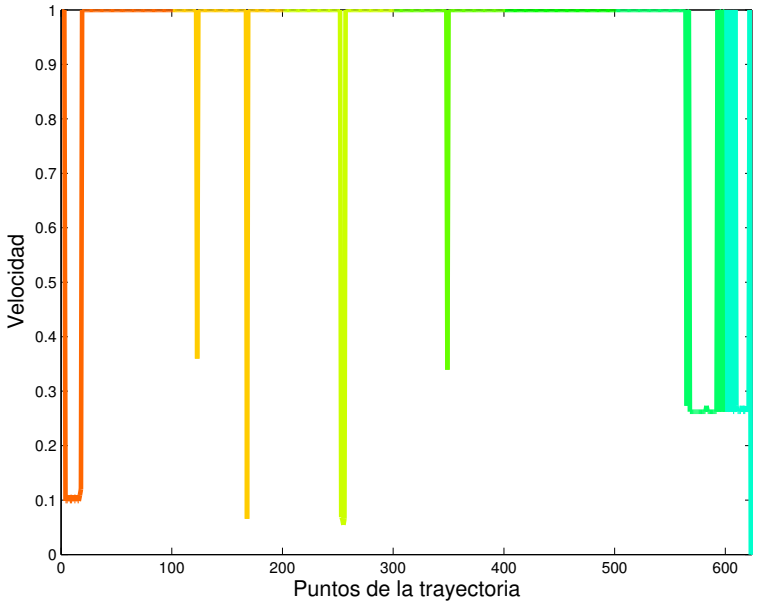


Figura 5.27: *Comparativa de trayectorias y expansión de onda de FM^2 frente a FM^2 Directional en caso 15.*



(a) *Perfil de velocidad utilizando FM^2.*



(b) *Perfil de velocidad utilizando FM^2 Directional.*

Figura 5.28: *Comparativa de perfiles de velocidad de FM^2 frente a FM^2 Directional en caso 9.*

En la figura 5.29 se muestra una comparativa de los valores de suavidad en cada ensayo, calculados acorde a la fórmula 4.4. Como se puede ver los valores son muy similares en la mayoría de los casos. En el peor de ellos el valor de la suavidad de FM^2 Directional crece aproximadamente el doble con respecto al de FM^2. Recordemos que la suavidad es mayor a medida que se acerque a 0 el valor, por lo que se considera un descenso de la misma. También se pueden observar casos en los que la suavidad de FM^2 Directional es mejor que la de FM^2. El motivo de esto es que aunque el resultado de suavidad es intuitivo no es perfecto, y la suavidad en zonas como las esquinas se obtiene como muy buena.

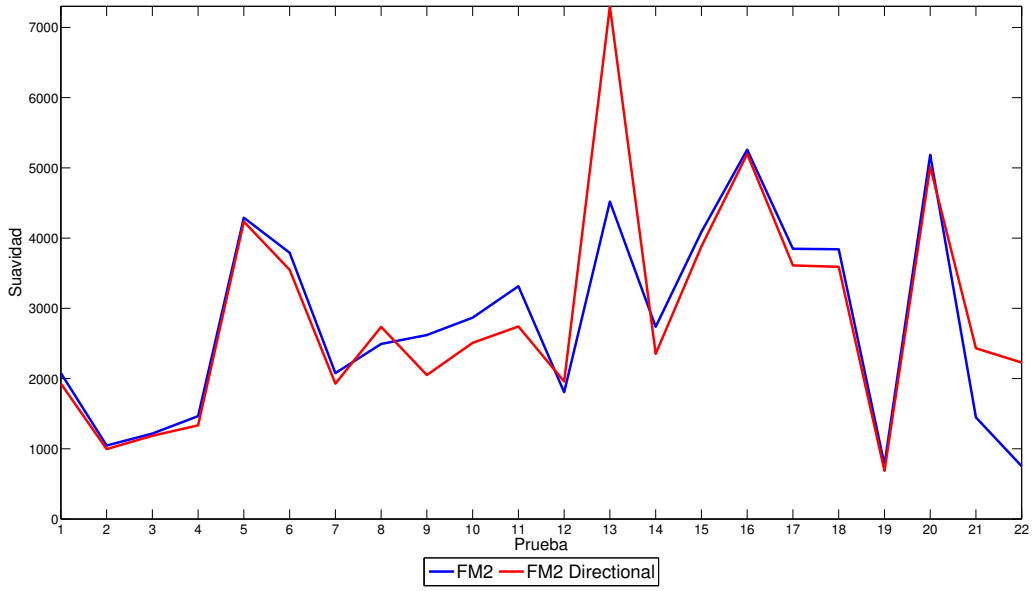


Figura 5.29: *Suavidad de la trayectoria utilizando FM^2 Directional.*

Dado que la motivación para realizar este método es disminuir el tiempo de recorrido de la trayectoria es inevitable hacer un análisis sobre el mismo. Como se puede ver en la figura 5.30, dado que los perfiles de velocidad de FM^2 Directional se mantienen la mayor parte del tiempo a velocidad máxima, los resultados resultan muy favorecedores para la nueva aproximación. Aunque se

trata de un factor de escala, como tamaño de celda se ha escogido 1 metro, y como velocidad máxima se ha utilizado 1 m/s.

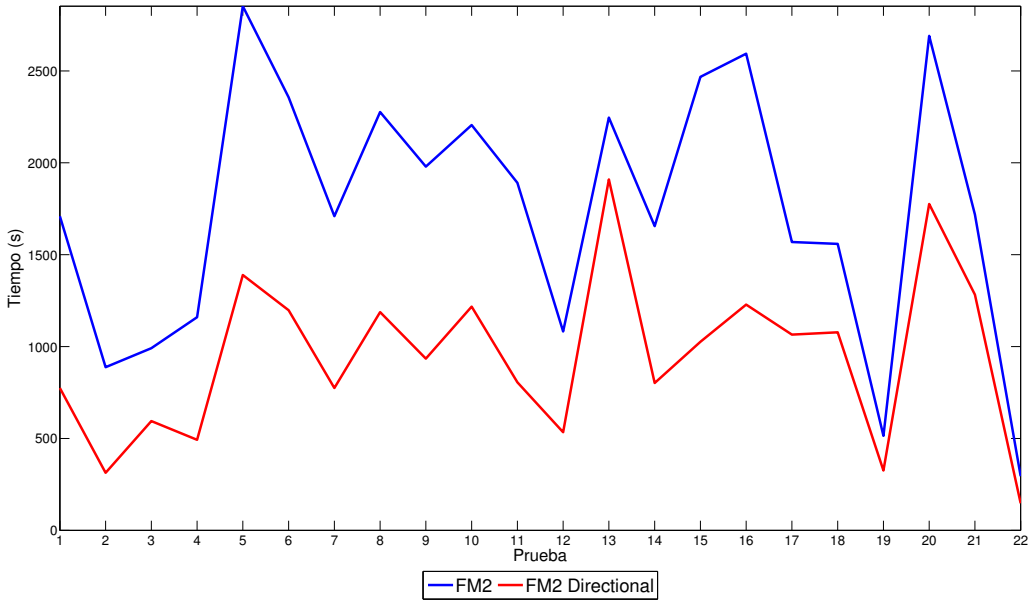


Figura 5.30: *Tiempo de recorrido de la trayectoria utilizando FM^2 Directional frente a FM^2.*

Análogamente se ha realizado un análisis de la distancia recorrida, utilizando de nuevo 1 metro como tamaño de celda. Como se puede ver en la figura 5.31 en la mayor parte de los casos la distancia recorrida por las trayectorias generadas utilizando FM^2 Directional es inferior a la distancia de las trayectorias de FM^2. En el caso en el que no lo es, la variación del perfil de velocidad hace que aún así se recorra en un tiempo menor.

Al mismo tiempo, y para comparar los resultados con FM^2 y FM^{2*} de la forma más equitativa posible, mostrando las virtudes y defectos de cada uno, se muestran también los tiempos de ejecución del algoritmo. Como se puede ver en la figura 5.32, los tiempos de ejecución del método FM^{2*} frente a FM^2 se ven disminuidos de forma visible, mientras que en el caso de FM^2 Directional se ven aumentados siempre. Esto se debe a que, aunque el algoritmo funciona igual que el método FM^2, durante la expansión de la onda se resuelve dos veces la

ecuación Eikonal por celda, por lo que inevitablemente el tiempo se ha de ver incrementado. Sin embargo este incremento de tiempo no es crítico ya que es la única solución que tiene en cuenta la direccionalidad hasta el momento.

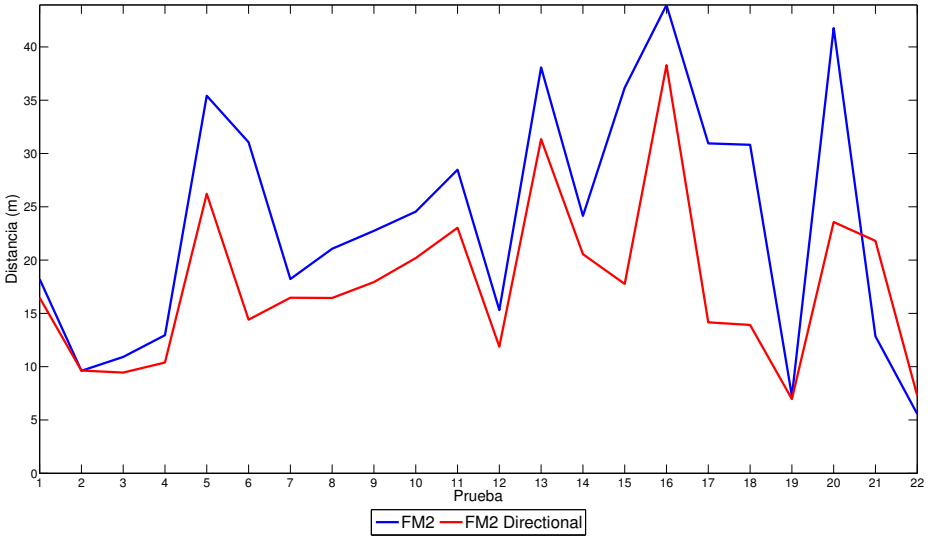


Figura 5.31: *Distancia recorrida por la trayectoria utilizando FM^2 Directional frente a FM^2.*

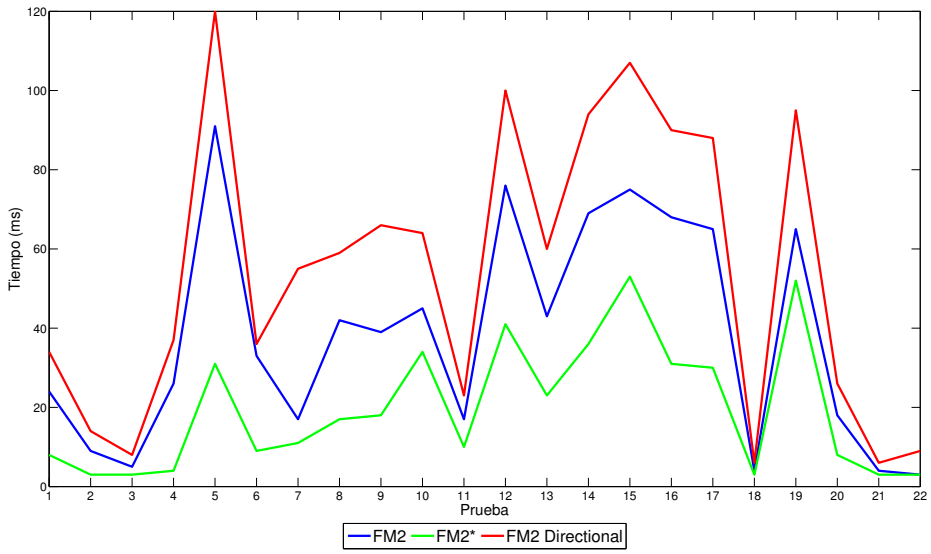


Figura 5.32: *Comparación de tiempo de ejecución de los distintos algoritmos.*

5.4. Conclusiones

En este capítulo se han presentado los fundamentos teóricos del método FM^2 Directional, descritos en la sección 5.2.

El objetivo final del desarrollo de FM^2 Directional busca mantener la suavidad de las trayectorias de FM^2 en la medida de lo posible pero al mismo tiempo generar trayectorias más óptimas dada la geometría del escenario y los puntos de inicio y fin de la trayectoria, aumentando además la velocidad del perfil de velocidad. Para ello se resuelve la ecuación Eikonal de dos formas diferentes en paralelo. Por un lado se expande la onda al igual que en FM^2, resolviendo la ecuación Eikonal con la velocidad del mapa de velocidades, al mismo tiempo que se calcula un tiempo de llegada diferente con un valor de velocidad diferente. Dicho valor puede ser la velocidad máxima en caso de que la velocidad de la celda sea mayor que la de sus vecinos (lo que implica que el robot se aleja del obstáculo) o la del mapa de velocidades en caso contrario. De este modo se obtienen trayectorias más cortas en términos de tiempo al generar perfiles de velocidad con una velocidad superior a la que propone FM^2.

Durante los comparativas realizadas en la sección 5.3 se obtienen siempre trayectorias más cortas en términos de tiempo ya que la velocidad empleada para recorrerlas es superior. En términos de distancia las trayectorias de FM^2 acostumbran también a ser más cortas, y en caso de no serlo compensan esa distancia añadida con una alta velocidad que implica que se recorran en un tiempo menor. Además la suavidad, aunque es una medida orientativa y no exacta, ronda valores similares a los que se obtiene a partir de FM^2 para las mismas condiciones. Por ese motivo, aunque existe una pérdida de suavidad en el nuevo método, esta pérdida resulta asumible. Por contra el tiempo de cómputo se ve incrementado al tener que realizar un mayor número de operaciones intermedias y resolver la ecuación Eikonal dos veces por operación.

Dados todos estos datos, se puede determinar que el método de FM^2

Directional propuesto ofrece unos resultados correctos y cumple los objetivos principales que han motivado el desarrollo de una variación de FM^2 que genere trayectorias más óptimas.

La elección de FM^2 Directional frente a FM^2 o FM^{2*} se basa principalmente en las necesidades de cada situación. Los tres métodos han resultado ser muy similares entre sí, sacrificando ligeramente alguno de los aspectos para poder mejorar otros. En caso de dar mayor importancia al tiempo de cómputo, la elección obvia es FM^2. Mientras, si la suavidad es la prioridad, FM^2 es el método más confiable. Y finalmente si se buscan los caminos más cortos en tiempo de recorrido, FM^2 Directional es el método más recomendable.

Capítulo 6

Pruebas de algoritmos en sistema real Turtlebot

6.1. Introducción

A lo largo de los capítulos 3, 4 y 5 se han detallado tres métodos diferentes para abordar el problema de la planificación de trayectorias a partir de métodos derivados de Fast Marching, concretamente FM^2, FM^{2*} y FM^2 Directional. La funcionalidad y características de los tres métodos ha sido probada en dichas secciones a nivel teórico sobre mapas ideales. No obstante el fin del desarrollo de algoritmos de planificación de trayectorias no es otro que resolver problemas de movimiento en entornos reales.

Para probar la funcionalidad se ha implementado en una plataforma real, concretamente en la primera versión del robot Turtlebot. El Turtlebot, mostrado en la figura 6.1, es un robot móvil de dos ruedas de bajo coste y basado en la filosofía de software y arquitectura de hardware libre. Este último punto quiere decir que pese a usar elementos propietarios de otras empresas, como la base robótica, el ordenador o la cámara, comprando dichos elementos por separado se puede construir un Turtlebot compatible. Es un robot especialmente indicado

para investigación y pruebas de algoritmos ya que su filosofía libre permite tener acceso al más bajo nivel.

Figura 6.1: *Robot Turtlebot.*

La base sobre la que se cimienta el robot es una base iRobot Roomba 530 (Wikipedia, 2014c), muy extendida en el campo de la robótica doméstica. La base no es diferencial, por lo que el robot precisa parar para rotar antes de continuar la trayectoria. Además de una base móvil Roomba 530, el robot dispone de una base de metacrilato sobre la que se sitúa una cámara Kinect (Wikipedia, 2014a), muy extendida en investigación por su gran compromiso entre versatilidad, calidad y precio, y un portátil de bajo consumo que funciona como estación de control del robot. Dicho portátil funciona con el sistema operativo Ubuntu (Wikipedia, 2014d) y utiliza el framework ROS (Wikipedia, 2014b) como herramienta básica de control del sistema.

El robot funciona plenamente a través de ROS, permitiendo no solo el uso de utilidades o características proporcionadas como el fabricante, tales como la generación de mapas del entorno o la localización dentro del mismo mediante

el Algoritmo de Monte Carlo (Fox, Burgard, Dellaert, y Thurn, 1999), sino además permite el desarrollo y la integración de algoritmos desarrollados por investigadores.

6.2. Descripción de la arquitectura del sistema

ROS (Robot Operating System) es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS provee los servicios estándar de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de numerosas fuentes como sensores, algoritmos de control, estados, planificaciones y actuadores. ROS es software libre que se rige bajo términos de la licencia BSD, que permite libertad para uso comercial e investigador.

ROS se compone de dos partes básicas: el sistema operativo y un conjunto de paquetes, aportados tanto por empresas como usuarios, que implementan la funcionalidades tales como localización, planificación, percepción o simulación. Sobre un mismo núcleo de ROS corriendo en un PC se pueden conectar distintas plataformas (por ejemplo, distintos ordenadores) con la posibilidad de enviar y recibir mensajes entre ellos. En la arquitectura de este sistema se utilizan dos ordenadores conectados en red con el fin de diversificar la carga computacional entre ellos. Por un lado se dispone de la estación embarcada en el Turtlebot, encargada de conectarse con el robot, y por otro lado un portátil sobre el que ejecutar los algoritmos. De este modo, la estación embarcada en el robot se utiliza únicamente como centro de control encargado de enviar y recibir mensajes, mientras que el ordenador externo recibe la carga computacional del algoritmo.

A día de hoy ROS se ha establecido como el sistema más extendido en robótica, especialmente a nivel de investigación. Una de sus principales aportaciones consiste en la estandarización bajo un mismo framework de todas las aplicaciones orientadas a robótica, facilitando así la interconexión entre módulos y la exportación de elementos a otros sistemas basados en ROS. Este motivo ha sido esencial para la elección del robot Turtlebot ya que la integración de los algoritmos FM^2, FM^{2*} y FM^2 Directional está generalizada para poder ser integrada en cualquier otro sistema.

La arquitectura básica sobre la que se cimienta ROS, como se ha comentado anteriormente, se basa en una estructura de grafos, con nodos conectados entre sí. Cada uno de esos nodos contiene un conjunto de entradas y salidas que se conectan a su vez con otros nodos. A nivel de desarrollo, la estructura de ROS funciona como paquetes que contienen al menos un nodo en su interior.

Para implementar correctamente los métodos de planificación en el Turtlebot se ha desarrollado una arquitectura basada en tres paquetes. El primero de ellos es el paquete de Fast Marching, que hace las veces de planificador de trayectorias y está diseñado para ser independiente de la plataforma o sistema sobre el que se usará. El segundo paquete, *Turtlebot fm*, hace las veces de traductor entre los elementos del robot y el planificador. Finalmente, el último paquete, *Turtlebot move*, recoge los datos que envía el paquete *Turtlebot fm* y se comunica con el robot para que realice los movimientos. A nivel esquemático se muestra en la figura 6.2.

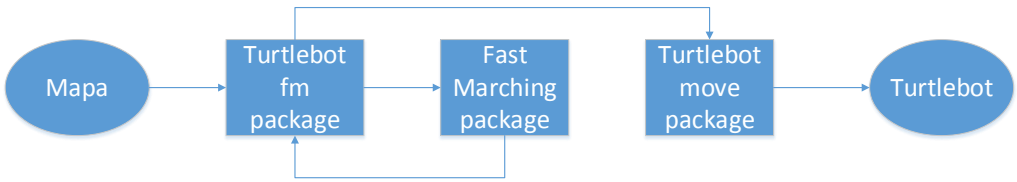


Figura 6.2: *Diagrama de flujo de la arquitectura propuesta para el Turtlebot.*

6.2.1. Paquete Turtlebot fm

En esta sección se va a detallar la arquitectura propuesta para el paquete *Turtlebot fm*. El paquete busca realizar una conexión entre el planificador, genérico, y los distintos elementos del robot. Para ello se utilizan dos nodos diferentes, como se puede ver en la figura 6.3. El primero de ellos, *load map*, se encarga de transformar un mapa obtenido previamente en un formato comprensible para el planificador, y además permite seleccionar los puntos de inicio y final, además de la rotación inicial del robot. El segundo nodo que incluye el paquete, *send path*, recibe la trayectoria y el perfil de velocidad generado por el planificador para transformarlo en incremento en posición (en metros) e incremento en rotación (en radianes), publicando además la velocidad relativa de ese punto según el perfil de velocidad. En la figura 6.3 se presenta un diagrama de flujo del paquete.

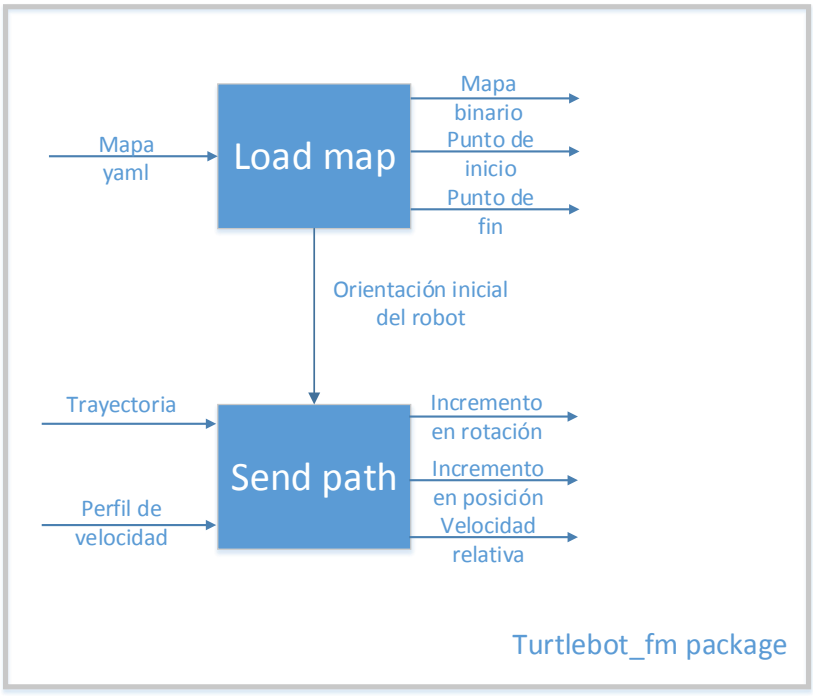


Figura 6.3: *Diagrama de flujo del paquete Turtlebot fm.*

El mapa puede ser generado de distintas formas, aunque de forma habitual se utiliza la cámara Kinect para realizar un modelado del entorno. Dicho mapa se almacena en el ordenador, utilizando un formato llamado *yaml* y posteriormente se carga desde el nodo *map server* integrado en ROS. Este nodo abre el mapa y lo publica en formato *yaml* muy extendido en ROS. Dicho formato provee la siguiente información:

- **Rejilla de ocupación**: Rejilla de ocupación compuesta por valores entre 0 y 1 (0 significa espacio ocupado y 1 espacio libre), o valor -1 (desconocido). Idealmente la rejilla sería binaria (celda ocupada o no) pero el sistema tiene en cuenta la incertidumbre de los sensores.
- **Origen del mapa**: Punto inicial desde el que se ha generado todo el mapa.
- **Resolución**: Tamaño de la celda, en metros.
- **Escalón de ocupación**: Valor a partir del cual cualquiera que lo supere en la rejilla de ocupación se considera ocupado.
- **Escalón de zona libre**: Valor a partir del cual cualquiera que se encuentre por debajo en la rejilla de ocupación se considera totalmente libre.

El nodo *load map* se encarga de leer esa información y procesarla de tal forma que se genere un mapa binario en un formato comprensible para el planificador, tal que:

- **Rejilla de ocupación**: Rejilla de ocupación discretizada a un formato binario, es decir, 0 y 1, representando el valor 0 la zona ocupada y el valor 1 la zona libre. Por simplicidad, y dado que según los valores de los escalones de zona libre y zona ocupada tienen un rango intermedio indefinido, se ha establecido que cualquier valor inferior a 0.5 será considerado 0, y cualquier valor superior será considerado 1.
- **Resolución**: Tamaño de la celda, en metros.

El nodo *load map*, además, solicita los puntos de inicio y fin, además de la orientación inicial del robot, para permitir que el sistema comience a funcionar.

Por su parte, el nodo *send path* recibe la trayectoria que ha generado el planificador, junto al perfil de velocidad y la orientación inicial del robot, para traducirlo en incrementos de posición, rotación y velocidad relativa. El planificador, como se ha comentado antes, está pensado para generar trayectorias para n dimensiones, lo que quiere decir que cada trayectoria estará definida por un conjunto de p puntos, y cada punto contendrá sus coordenadas en cada una de las n dimensiones. El seguimiento secuencial de dicho conjunto de puntos será la trayectoria generada por el planificador. Las coordenadas de posición que devuelve el planificador están transformadas a posiciones en metros, en lugar de posiciones en celdas, como se describe en la sección 6.2.2. Dada la geometría de los problemas de planificación utilizada por el Turtlebot, el número de dimensiones será únicamente 2 (coordenadas X e Y).

La trayectoria, pues, será un conjunto de puntos c_i con $i \in 0, 1, ..., p$, donde c_0 es el punto de inicio de la trayectoria y c_p el destino. Imaginemos un punto $i \in c$ y un punto consecutivo $i+1 \in c$. Para realizar el movimiento desde (x_i, y_i) hasta (x_{i+1}, y_{i+1}), el incremento de posición, ΔD se calcula acorde a la expresión 6.1.

$$\Delta D = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{6.1}$$

Mientras, el incremento de rotación, $\Delta\theta$, tiene en cuenta además de la posición de los puntos i e $i+1$, el ángulo de rotación inicial según la odometría, θ_{odom} y el ángulo de orientación inicial del robot cuando comienza el recorrido, θ_0. Se calcula según la expresión 6.2.

$$\Delta\theta = atan2((y_{i+1} - y_i), (x_{i+1} - x_i)) - \theta_{odom} + \theta_0 \tag{6.2}$$

La velocidad relativa, por su parte, se obtiene del perfil de velocidad generado. Este proceso se realiza iterativamente hasta que se recorran todos los puntos del conjunto c. Dado que la densidad de puntos de una trayectoria es

muy alta, se opta por elegir un punto de objetivo cada 5, con el fin de evitar que el robot realice movimientos excesivamente cortos.

6.2.2. Paquete Fast Marching

En esta sección se va a detallar la arquitectura propuesta para el paquete *Fast Marching*. Este paquete no es más que el propio planificador, que integra nodos para generar la trayectoria utilizando como métodos FM^2, FM^{2*} y FM^2 Directional. Este paquete está orientado a ser independiente de la plataforma o sistema en el que se va a integrar. Por motivos de simplicidad y evitar la repetición, en la figura 6.4 se muestra un nodo genérico ya que las entradas y salidas de todos los nodos son idénticas y solo varía el método utilizado para generar la trayectoria. Como se puede ver, la entrada del nodo consiste en el mapa binario y los puntos de inicio y final, todos provenientes del nodo *load map*, que forma parte del paquete *Turtlebot fm*. Las salidas son la trayectoria, en coordenadas *n* dimensionales y el perfil de velocidad, que funcionan como entrada para el nodo *send path*, del paquete *Turtlebot fm*. En la figura 6.4 se presenta un diagrama de flujo del paquete.

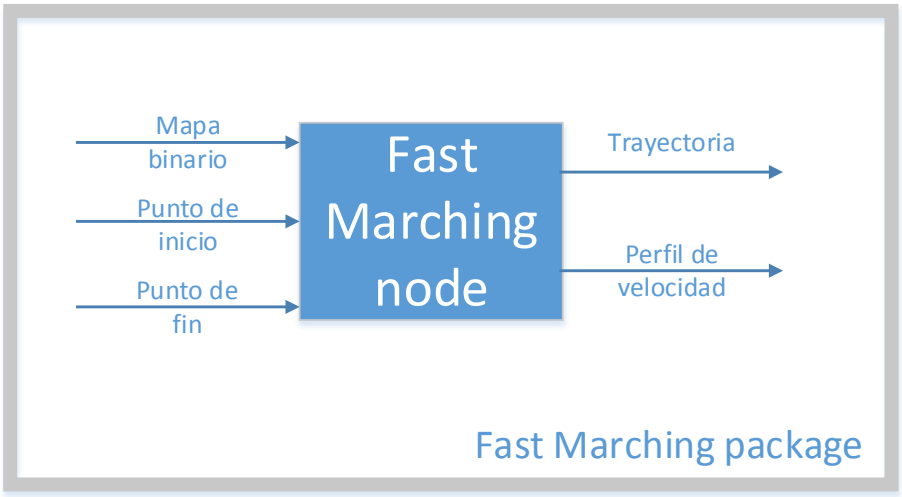


Figura 6.4: *Diagrama de flujo del paquete Fast Marching.*

El mapa de entrada, como se define en la sección 6.2.1, consta de los siguientes elementos:

- **Rejilla de ocupación**: Rejilla de ocupación discretizada a un formato binario, es decir, 0 y 1, representando el valor 0 la zona ocupada y el valor 1 la zona libre.
- **Resolución**: Tamaño de la celda, en metros.

Utilizando la rejilla de ocupación, y los puntos de inicio y final de la trayectoria se generan las trayectorias y el perfil de velocidad. El planificador, originalmente, genera las coordenadas en el sistema de coordenadas de la rejilla en lugar de metros. Por tanto es necesario un procesado posterior que transforme las coordenadas a metros. Para ello se utiliza la resolución como se propone en la ecuación 6.3

$$\begin{aligned} x_m &= x_{cell} * resolution \\ y_m &= y_{cell} * resolution \end{aligned} \tag{6.3}$$

6.2.3. Paquete Turtlebot move

En esta sección se va a detallar la arquitectura propuesta para el paquete *Turtlebot move*. El paquete se encarga de recibir el incremento en posición (en metros), el incremento de rotación (en radianes) y la velocidad relativa desde el nodo *send path* del paquete *Turtlebot fm*. Este último proceso es iterativo ya que recibirá los incrementos de posición y rotación y velocidad relativa para pasar de un punto al siguiente, por lo que recibirá información tantas veces sea necesario para recorrer toda la trayectoria. En la figura 6.5 se presenta un diagrama de flujo del paquete.

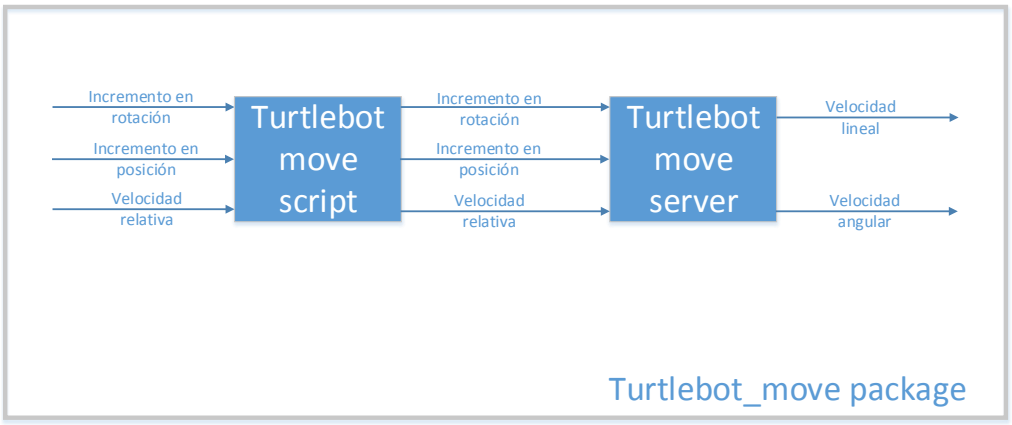


Figura 6.5: *Diagrama de flujo del paquete Turtlebot move.*

Como se puede ver en la figura 6.5, el paquete está compuesto por dos partes. La primera de ellas se trata de un script (*turtlebot move script*) que trabaja igual que un nodo, encargado de recibir el incremento de rotación (en radianes) y posición (en metros), además de la velocidad relativa, para publicarla posteriormente al servidor de movimiento (*turtlebot move server*). Este servidor se encarga de que el robot realice los movimientos generando una velocidad lineal y angular durante el tiempo necesario para que el robot realice el movimiento. La velocidad lineal se encarga de generar el incremento de posición, mientras que la angular realiza el incremento de rotación. La velocidad relativa afecta a las velocidades lineal (v) y angular (w) ya que realiza un escalado de las mismas, acorde a la ecuación 6.4.

$$\begin{aligned} v &= v_{max} * v_{relative} \\ w &= w_{max} * v_{relative} \end{aligned} \tag{6.4}$$

Dado que $v_{relative}$ se encuentra siempre entre 0 y 1, el valor final de la velocidad lineal y angular puede encontrarse entre 0 y los valores máximos de cada una. La modificación de los valores conlleva, además, que el robot necesitará de más tiempo para realizar el mismo movimiento ya que las

velocidades utilizadas serán menores.

Es necesario tener en cuenta en este punto que el robot, por su propio hardware, es incapaz de generar una velocidad lineal y angular al mismo tiempo por lo que para cada trayectoria ha de orientarse primero mediante la velocidad angular, y moverse posteriormente utilizando la lineal. De este modo cada movimiento necesita de dos pasos: giro y avance.

6.2.4. Resumen

En esta sección se realiza un resumen final de la arquitectura propuesta, mostrando en la figura 6.6 un esquema detallado de las entradas y salidas de cada nodo. En el apéndice B se detalla la información a nivel de uso sobre los nodos y se realiza un tutorial para ejemplificar.

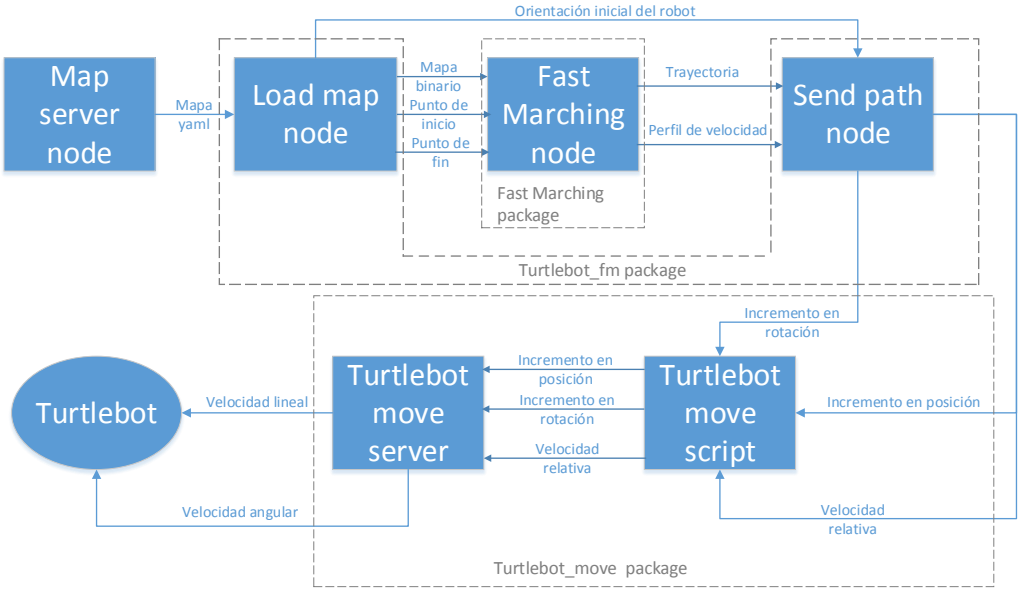


Figura 6.6: *Diagrama de flujo detallado del sistema.*

En la tabla 6.1 muestra el conjunto de nodos que conforman el sistema con sus entradas y salidas.

Nodo	Entradas	Salidas
Map server node	Ruta del mapa	Mapa yaml
Load map node	Mapa yaml	Mapa binario Punto de inicio Punto de fin Orientación inicial del robot
Fast Marching node	Mapa binario Punto de inicio Punto de fin	Trayectoria Perfil de velocidad
Send path node	Trayectoria Perfil de velocidad Orientación inicial del robot	Incremento en posición Incremento en rotación Velocidad relativa
Turtlebot move script	Incremento en posición Incremento en rotación Velocidad relativa	Incremento en posición Incremento en rotación Velocidad relativa
Turtlebot move server	Incremento en posición Incremento en rotación Velocidad relativa	Velocidad lineal Velocidad angular

Cuadro 6.1: *Resumen de entradas y salidas de los elementos del sistema.*

6.3. Pruebas realizadas sobre el robot

En esta sección se realizará un análisis de los métodos de planificación de trayectorias FM^2, explicado en la sección 3.2, FM^2*, desarrollado en la sección 4.2 y FM^2 Direccional, detallado en la sección 5.2. El método FM^2* Direccional no será analizado puesto que genera las mismas trayectorias que FM^2 Direccional, con la ventaja de calcularlas en un intervalo de tiempo menor. La comparativa consistirá en generar distintas trayectorias utilizando los tres métodos y comparar la diferencia entre el camino ideal y el recorrido por el robot. A partir de esta comparación se realizará un análisis de errores para comprobar la adaptación al campo real de los algoritmos teóricos. Los métodos serán probados en dos ámbitos diferentes, por un lado simulación y por otro sobre el robot real. En ambos ámbitos se utilizarán un conjunto de mapas de parte de los laboratorios de la Universidad Carlos III de Madrid generados por

el robot.

6.3.1. Detalles de implementación

La implementación ha sido realizada acorde a la arquitectura descrita en la sección 6.2, utilizando los códigos en C++ desarrollados para las pruebas de las secciones 3.2, 4.2 y 5.2 como planificadores.

Para las pruebas en simulación se utilizará Gazebo, un simulador integrado en ROS que permite la simulación de comportamientos de un robot modelado. La empresa encargada del desarrollo del Turtlebot integra el modelado del robot con soporte para Gazebo, permitiendo simular todos los comportamientos y las entradas y salidas reales del sistema. Gracias a esto, la implementación posterior en el sistema real será transparente ya que a nivel de estructura de ROS serán idénticos. En la figura 6.7 se muestra un ejemplo con el robot en Gazebo.

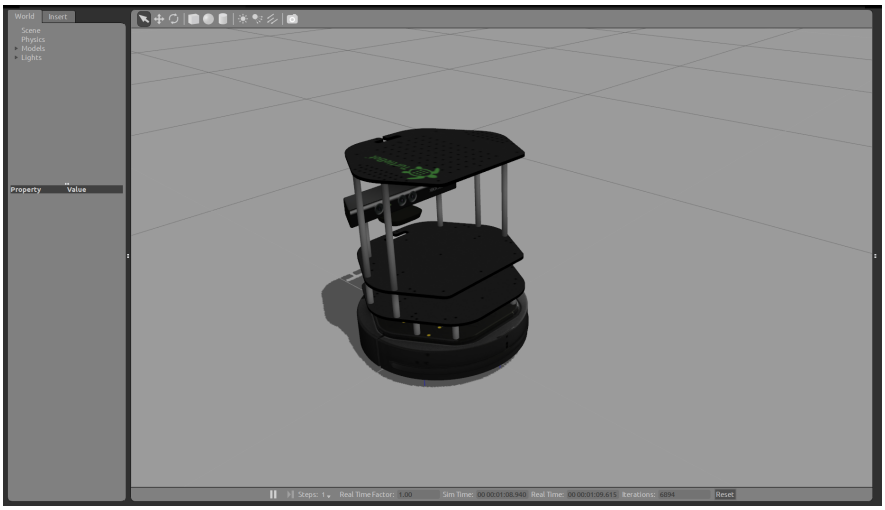


Figura 6.7: *Turtlebot en Gazebo.*

Como visualizador del recorrido se usa RViz. RViz es una herramienta que también forma parte de ROS y está orientada para mostrar de forma visual los mensajes que envía y recibe ROS, como pueden ser datos de localización o de

sensores. De esta forma se permite visualizar la posición del robot dentro del mapa, como se muestra en la figura 6.8.

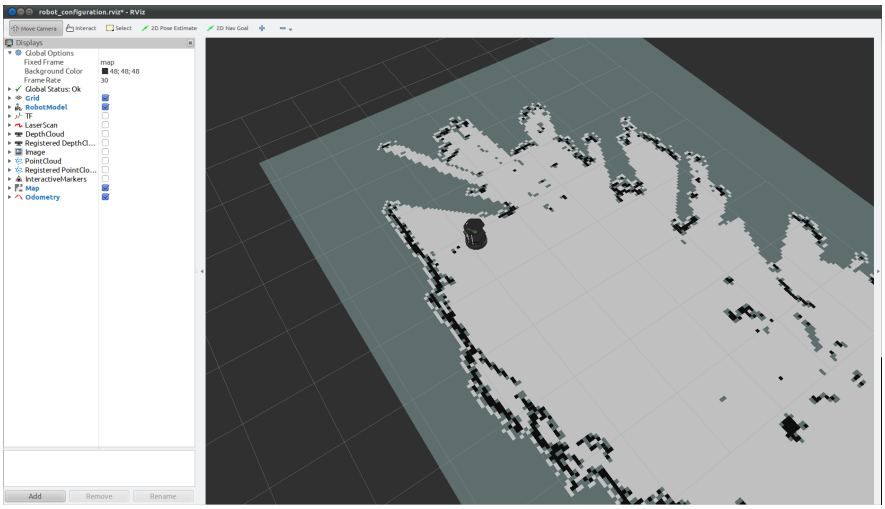
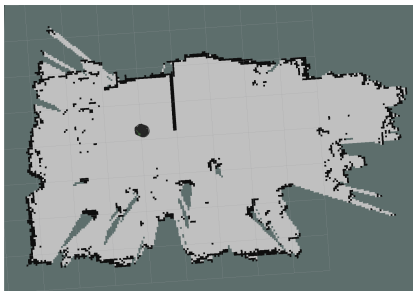


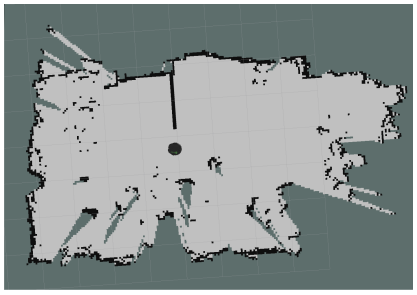
Figura 6.8: *Turtlebot en RViz.*

Con una correcta configuración de RViz se puede dibujar la localización del robot dentro del mapa tanto en simulación como con el robot real ya que utiliza los datos de odometría para mover al robot. En la figura 6.9 se muestra un ejemplo con una secuencia en la que se puede ver la correspondencia entre el robot real y RViz.



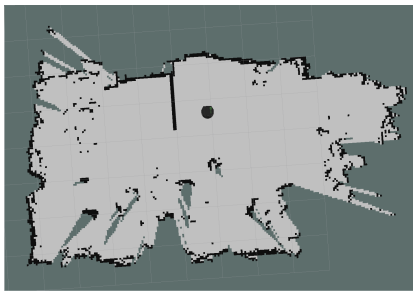
(a) *Posición 1 en RViz.*

(b) *Posición 1 en el entorno real.*



(c) *Posición 2 en RViz.*

(d) *Posición 2 en el entorno real.*



(e) *Posición 3 en RViz.*

(f) *Posición 3 en el entorno real.*

Figura 6.9: *Secuencia de movimiento del Turtlebot en la realidad y en RViz.*

El error se calculará como la distancia euclídea entre el punto de destino, (x_{goal}, y_{goal}), y el punto final alcanzado por el robot realmente, dado por la odometría, (x_{odom}, y_{odom}), tal y como se describe en la ecuación 6.5.

$$error = \sqrt{(x_{goal} - x_{odom})^2 + (y_{goal} - y_{odom})^2} \tag{6.5}$$

6.3.2. Pruebas en simulación

En las figuras 6.10, 6.11, 6.12 y 6.13 se muestran los resultados de las pruebas 1 a 4, respectivamente. En las imágenes se presentan las trayectorias generadas por cada uno de los métodos junto a la trayectoria seguida por el robot sobre un mismo mapa en entorno de simulación Gazebo. Como se puede ver en todos los casos, la trayectoria seguida por el robot comienza a desviarse desde el primer tramo por lo que aunque acabe resultando muy similar a la ideal el recorrido tiene una acumulación de errores. Estos errores se encuentran ocasionados por multitud de motivos. En primera instancia, el robot no es un sistema ideal y comete imprecisiones en las medidas. La resolución de los encoder del robot es discreta por lo que tampoco le es posible realizar los movimientos con precisión. Además, el funcionamiento del robot, basado en paradas entre cada giro y cada avance hace más propicia la aparición de errores de medida, ya que se producen acelerones bruscos difíciles de controlar. Se ha comprobado además que a mayor velocidad se producen mayores errores. En el caso de FM^2 Directional se puede observar, además, una mayor sensibilidad al ruido del mapa.

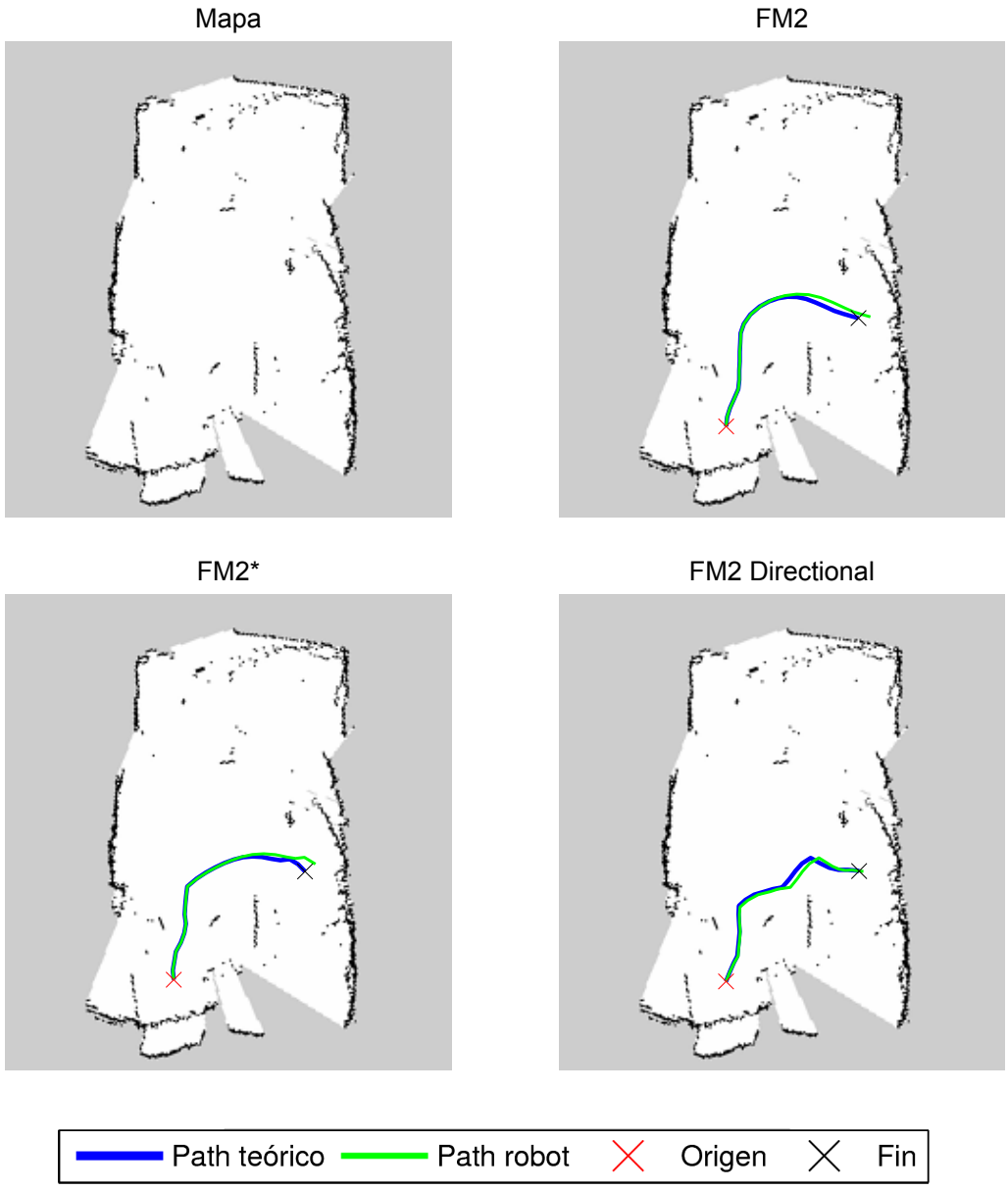


Figura 6.10: *Comparativa de trayectorias en mapa 1 en caso 1 en simulación.*

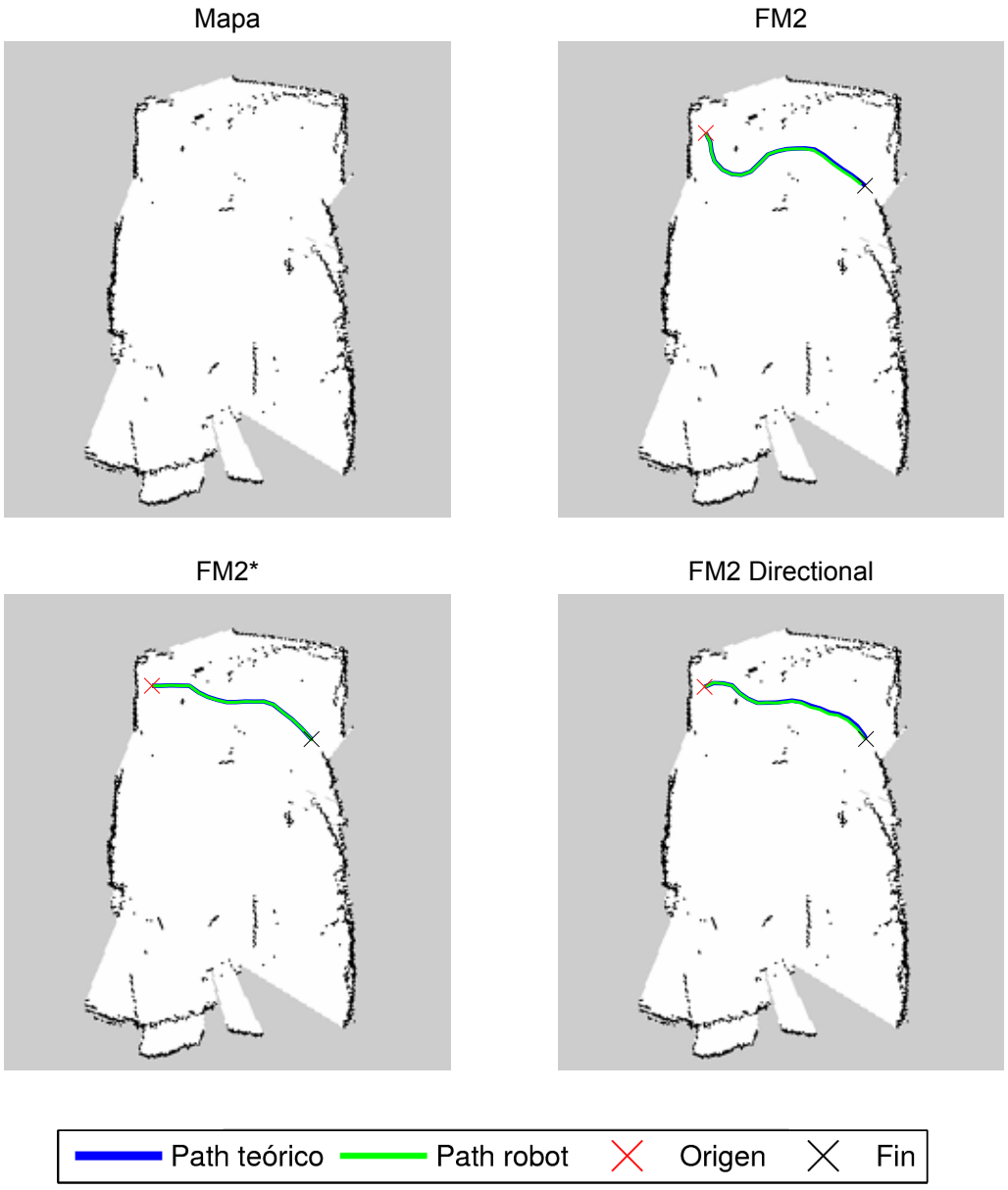


Figura 6.11: *Comparativa de trayectorias en mapa 1 en caso 2 en simulación.*

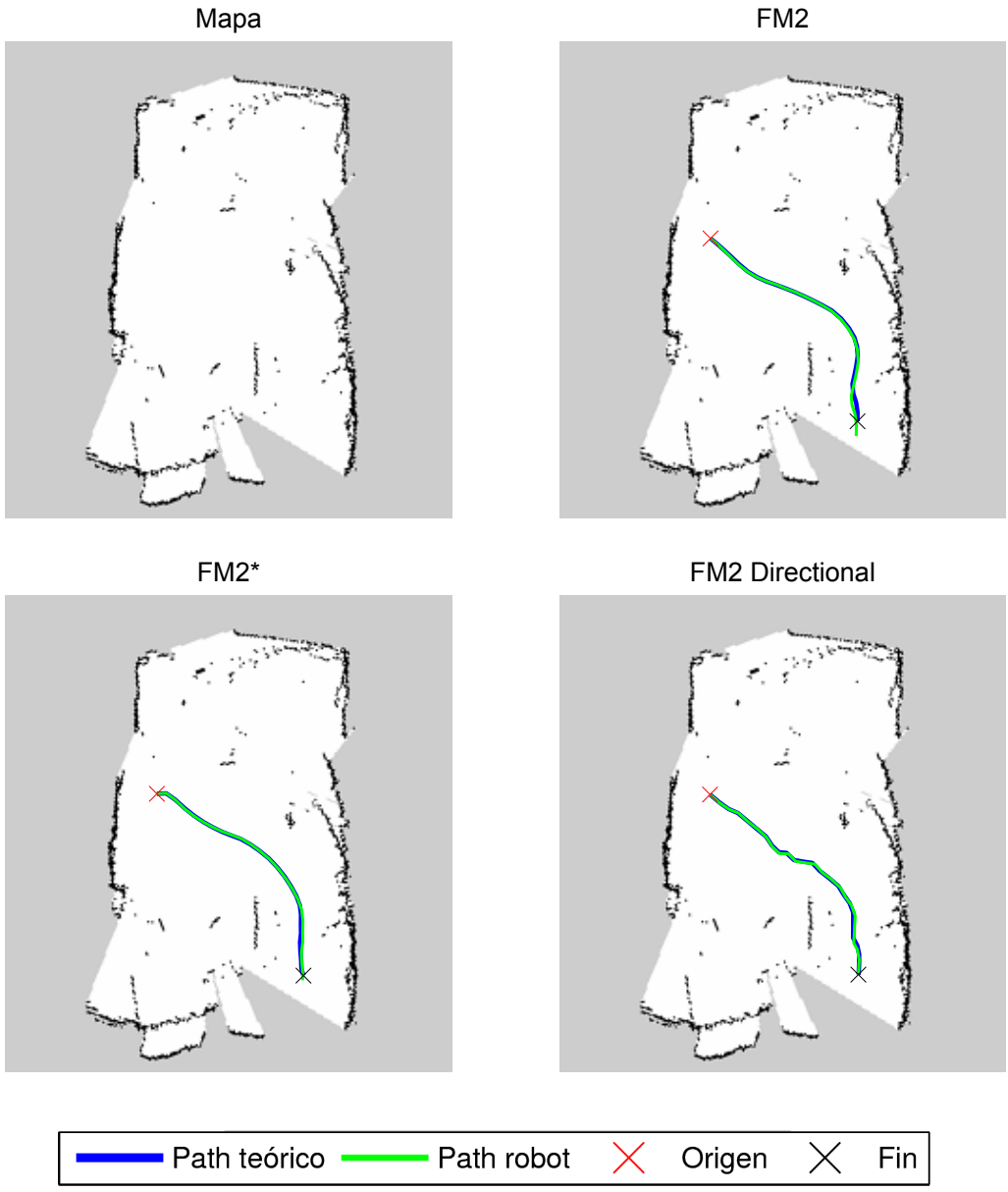


Figura 6.12: *Comparativa de trayectorias en mapa 1 en caso 3 en simulación.*

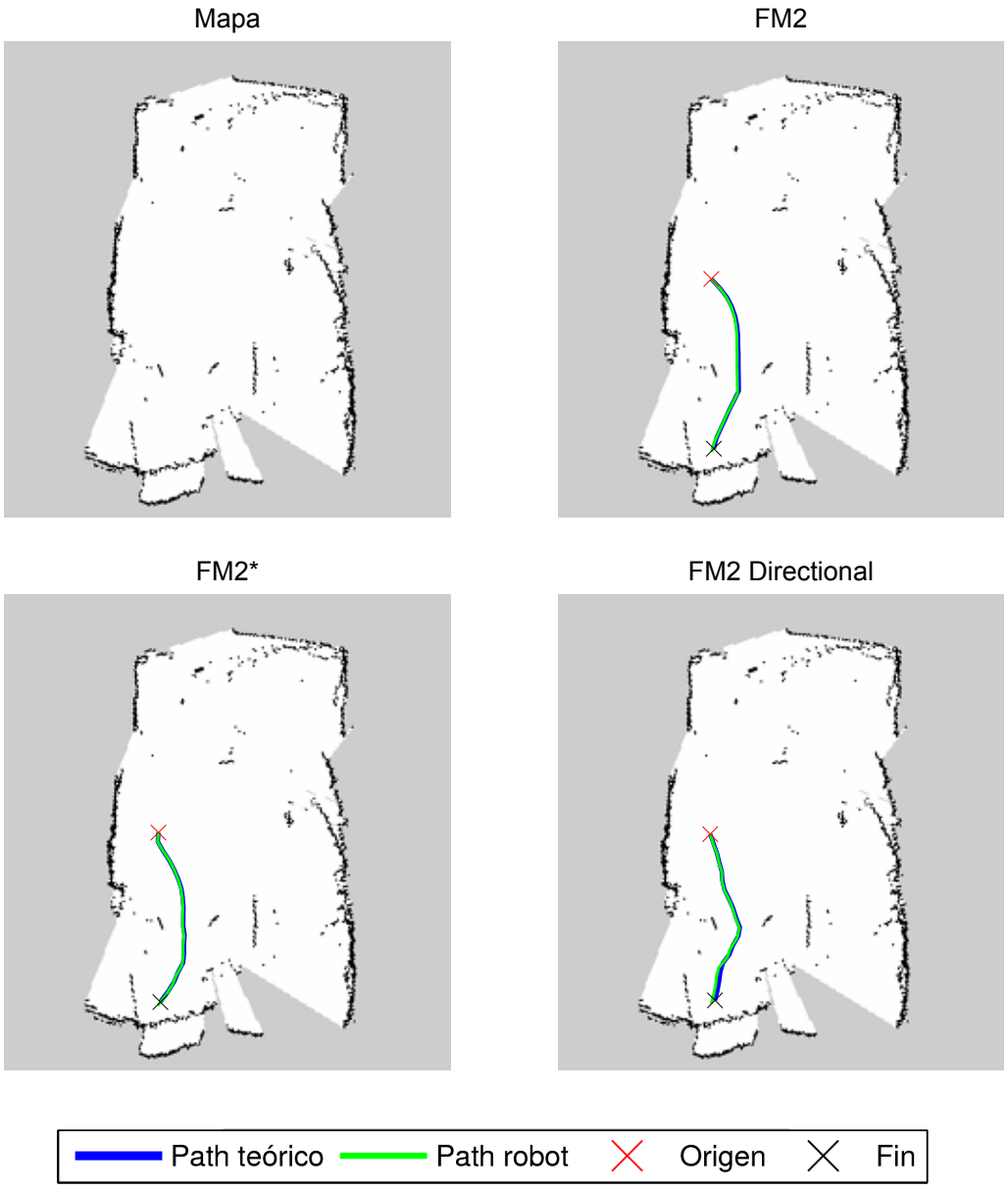


Figura 6.13: *Comparativa de trayectorias en mapa 1 en caso 4 en simulación.*

En las figuras 6.14, 6.15, 6.16 y 6.17 se representan las pruebas 5 a 8, respectivamente. En ellas se muestran las trayectorias generadas por cada método de planificación junto a la trayectoria seguida por el robot sobre un

nuevo mapa. De nuevo, la trayectoria seguida por el robot comienza a desviarse desde el primer tramo por lo que aunque acabe resultando muy similar a la ideal el recorrido tiene una acumulación de errores. En estos casos se puede observar también la sensibilidad al ruido de FM^2 Directional.

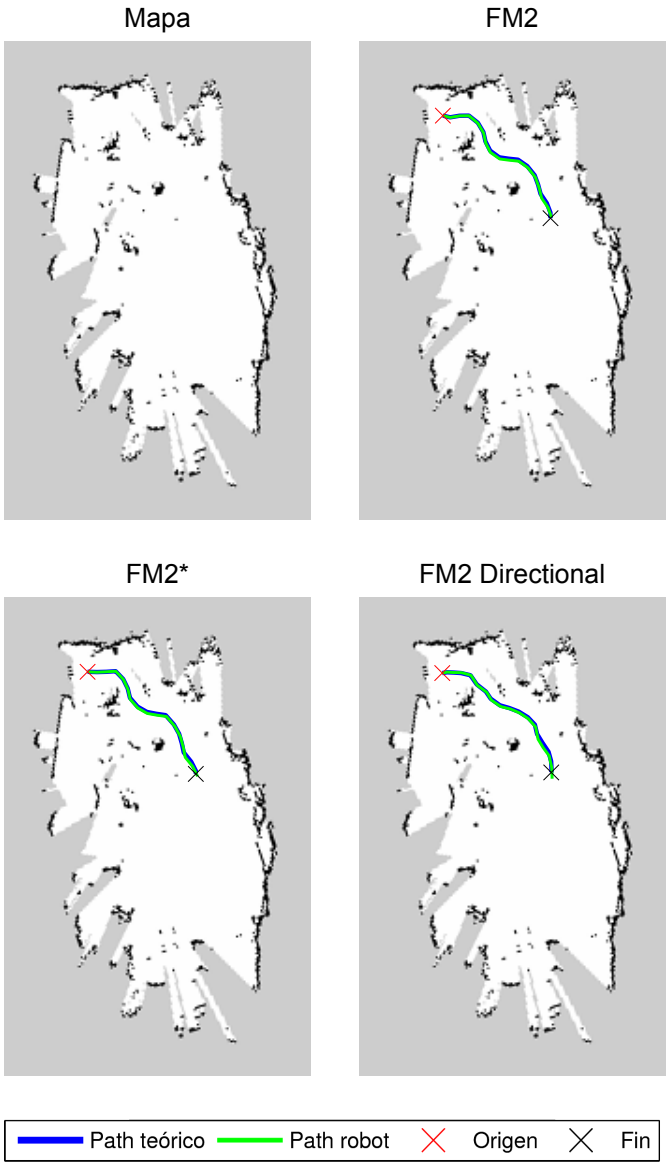


Figura 6.14: *Comparativa de trayectorias en mapa 2 en caso 1 en simulación.*

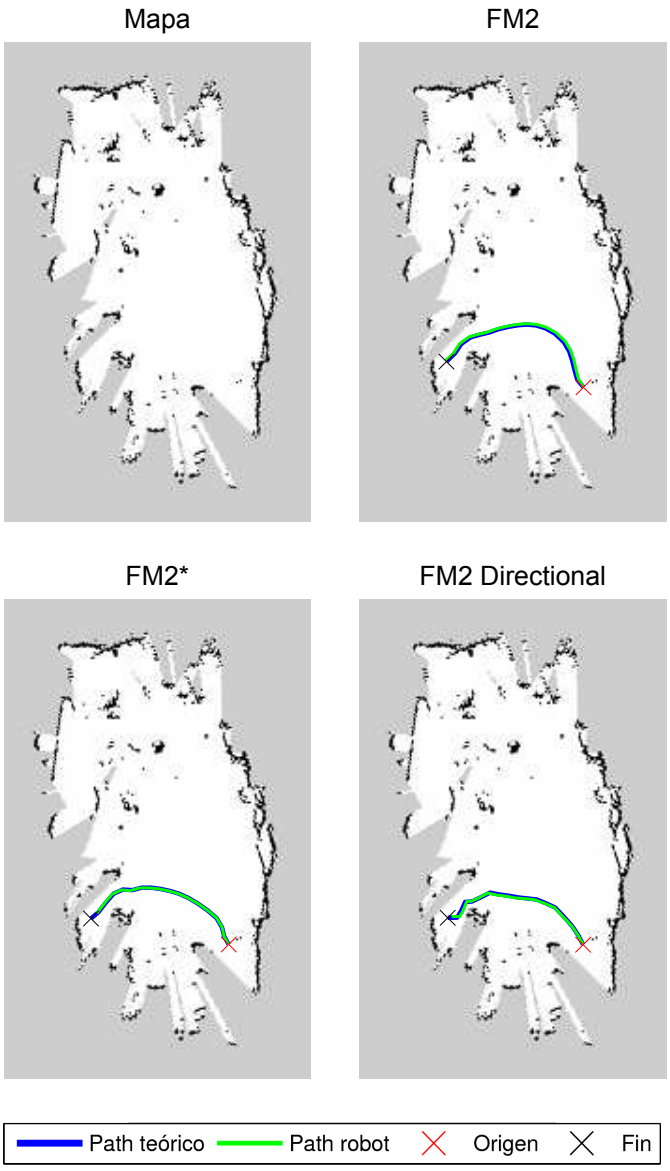


Figura 6.15: *Comparativa de trayectorias en mapa 2 en caso 2 en simulación.*

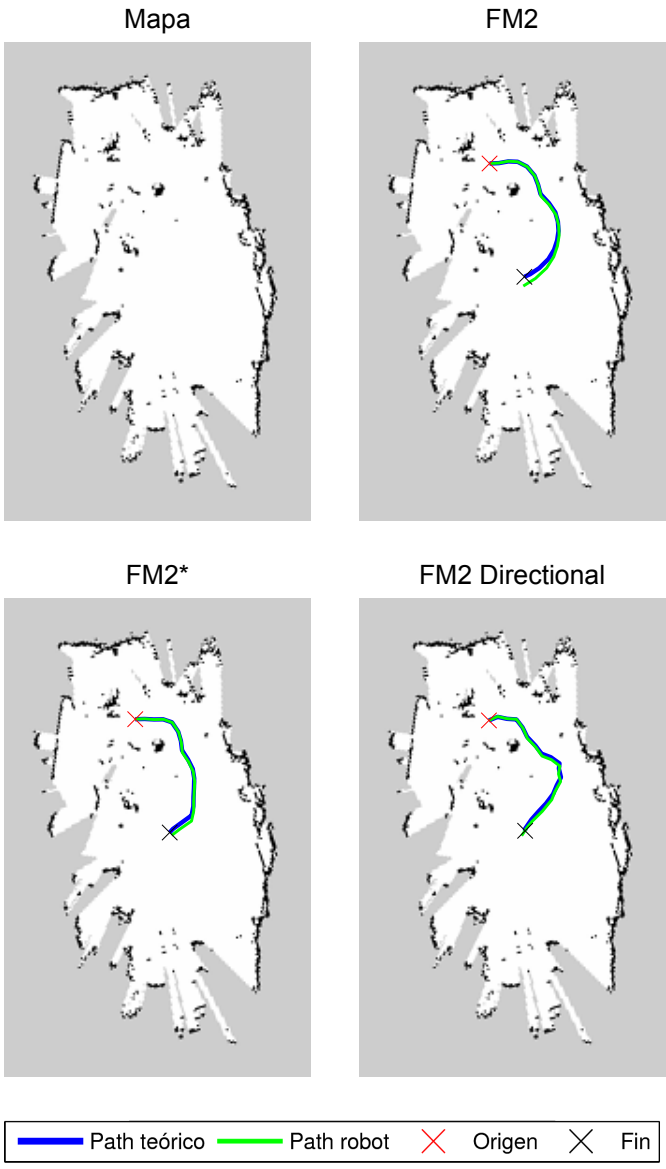


Figura 6.16: *Comparativa de trayectorias en mapa 2 en caso 3 en simulación.*

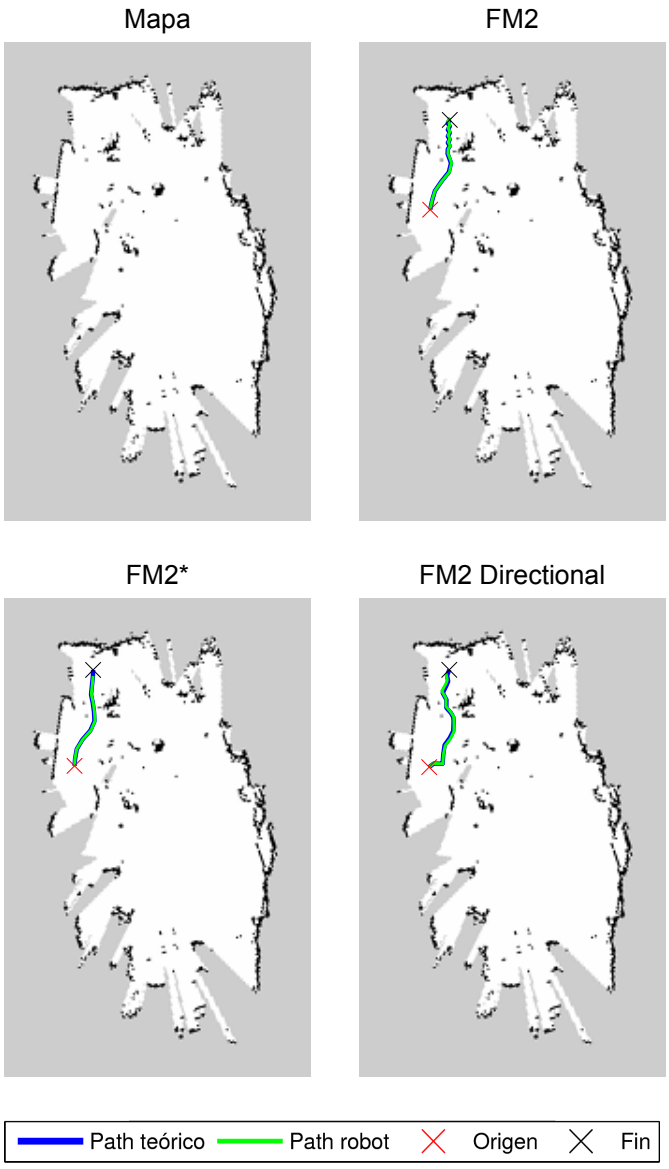


Figura 6.17: *Comparativa de trayectorias en mapa 2 en caso 4 en simulación.*

En la figura 6.18 se muestra el error de la trayectoria acorde a la fórmula presentada en la sección 6.3.1. El error medio que aparece es de 20 cm, con picos de 40 cm aproximadamente, sin diferencias muy significativas entre métodos. Aunque es un error alto, dadas las características del sistema utilizado se pueden

considerar dentro de márgenes válidos.

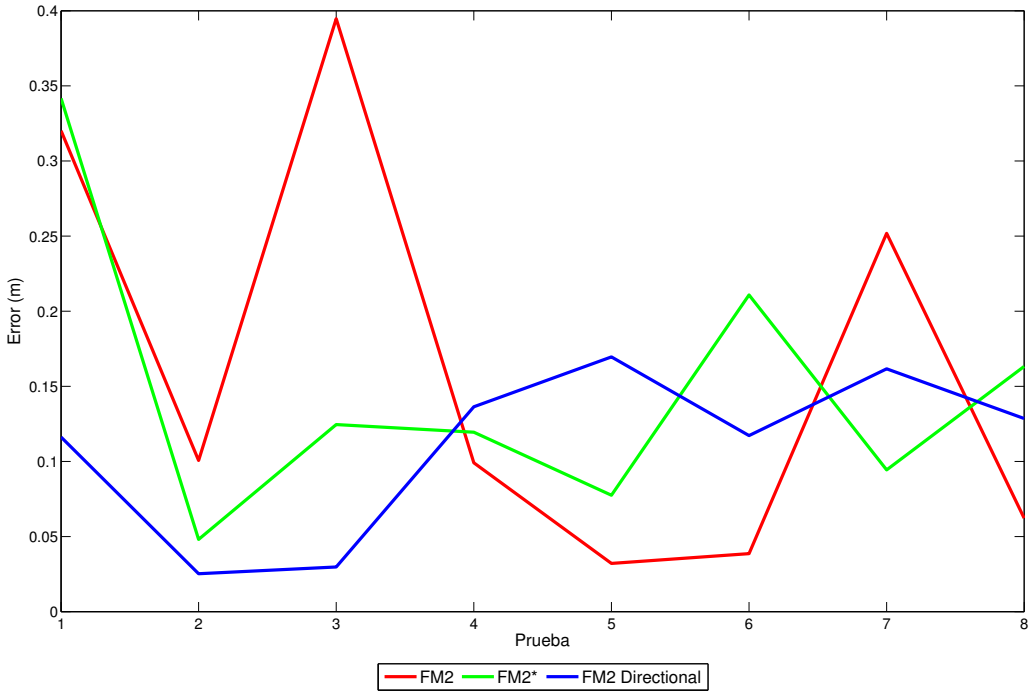


Figura 6.18: *Comparativa de error en distancia en cada prueba en simulación.*

6.3.3. Pruebas en sistema real

En las figuras 6.19 y 6.20 se muestran las trayectorias generadas por cada uno de los métodos junto a la trayectoria seguida por el robot utilizando el sistema real y la trayectoria seguida en simulación. Como se puede ver en todos los casos, la trayectoria seguida por el robot comienza a desviarse desde el primer tramo no solo por los motivos comentados durante las pruebas de simulación, sino por otras razones relacionadas con su uso en entornos reales como una calibración que no es perfecta, la textura del suelo y el propio desgaste del equipo que produce imperfecciones mayores en el funcionamiento. Todos estos errores son no sistemáticos del sistema, por lo que resultan difíciles de modelar y corregir y producen efectos no deseados sobre el

robot controlado. En todos los casos, la trayectoria seguida por el robot en simulación es más cercana a la realidad ya que se el simulador presupone un entorno ideal en el que únicamente añade un ruido blanco gaussiano.

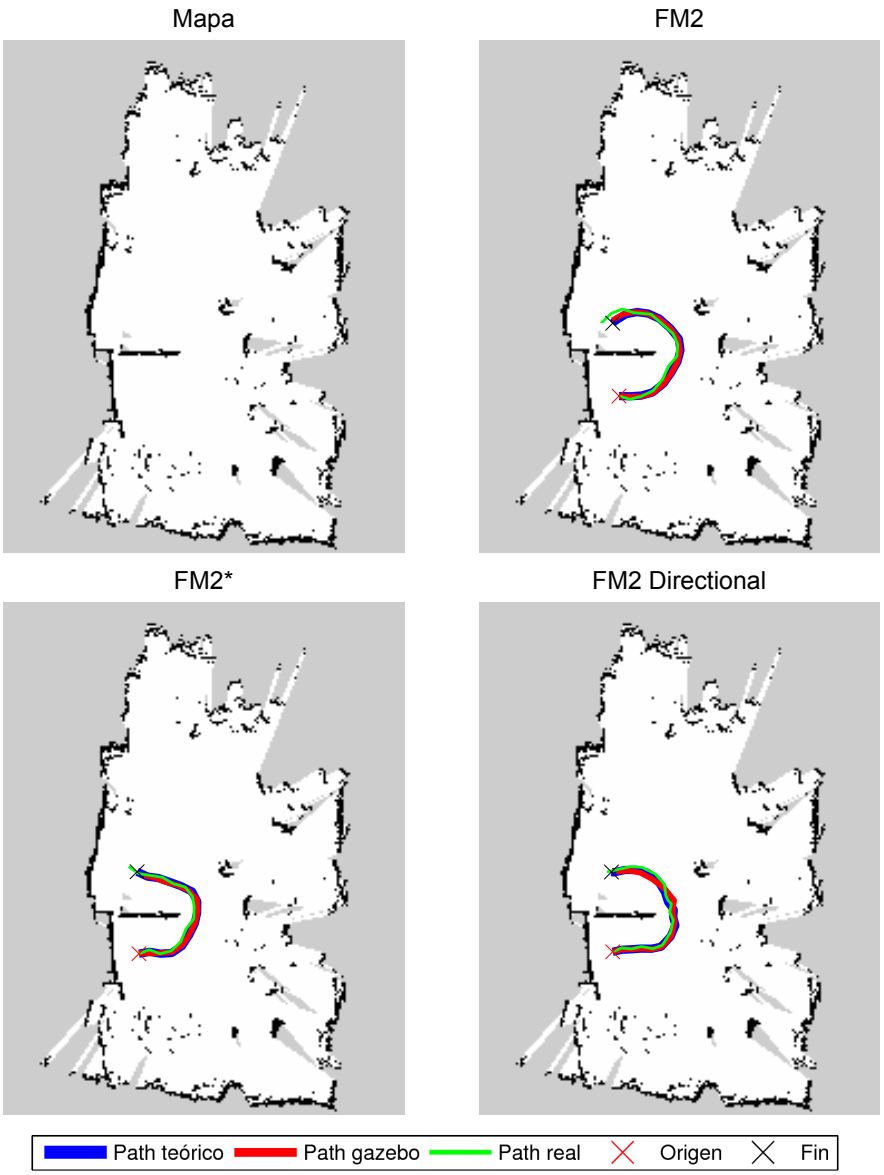


Figura 6.19: *Comparativa de trayectorias en mapa 1 en sistema real.*

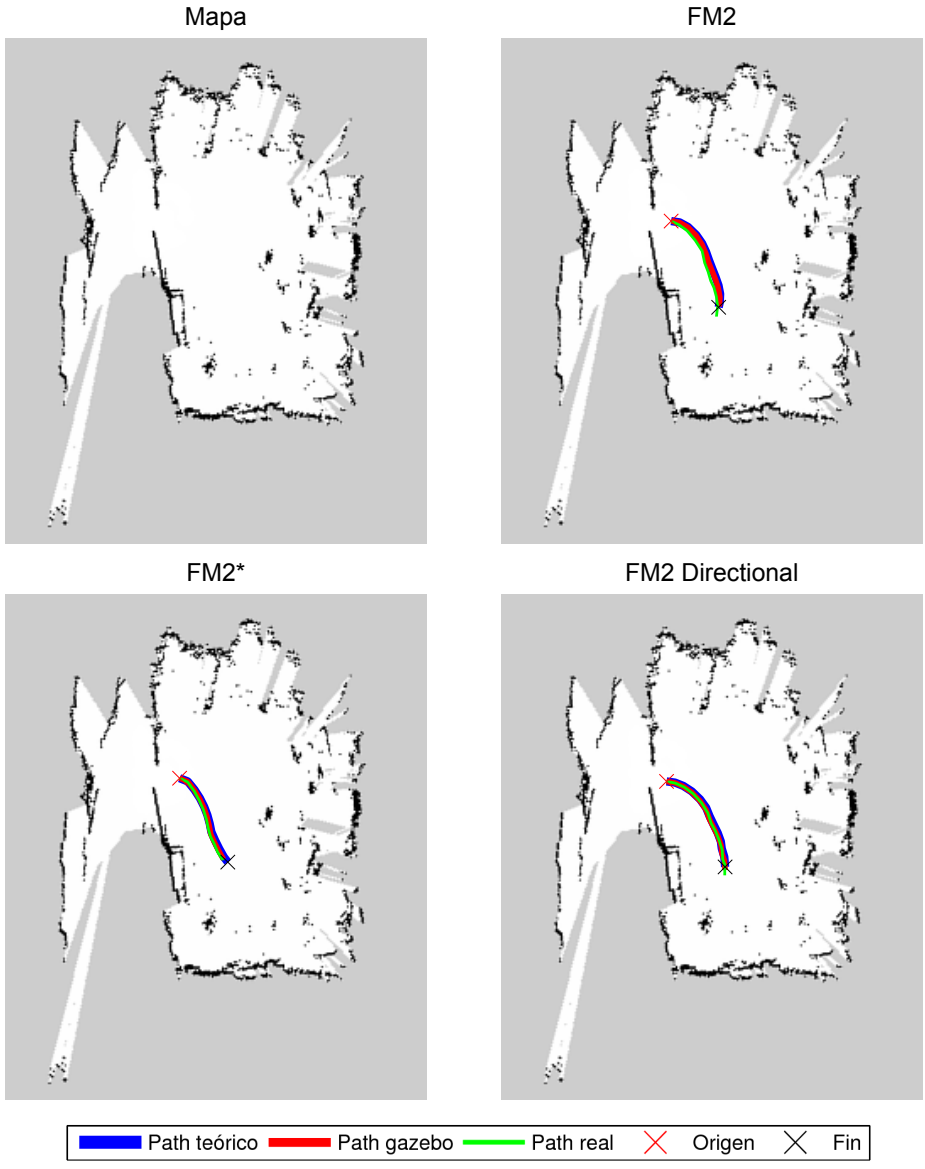


Figura 6.20: *Comparativa de trayectorias en mapa 2 en sistema real.*

En la tabla 6.2 se muestra el error de la trayectoria acorde a la fórmula presentada en la sección 6.3.1. El error en el sistema real oscila entre 10 cm y 30 cm, siempre superior al obtenido por el mismo método en simulación. Aunque es un error alto, si se tiene en cuenta que las trayectorias se han realizado sobre

	Error (m)					
	Gazebo			Real		
Prueba	FM^2	FM^{2*}	FM^2 Directional	FM^2	FM^{2*}	FM^2 Directional
1	0.06	0.17	0.09	0.26	0.23	0.1
2	0.03	0.15	0.08	0.27	0.18	0.25

Cuadro 6.2: *Comparativa de errores entre Gazebo y sistema real*

el sistema real y las características del mismo los errores se sitúan en márgenes correctos.

6.4. Conclusiones

En este capítulo se ha comprobado como resulta la implementación de los algoritmos FM^2 (capítulo 3), FM^{2*} (capítulo 4) y FM^2 Directional (capítulo 5) sobre un sistema real.

A lo largo de los capítulos que desarrollan teóricamente los resultados se plantean las hipótesis, condiciones y resultados considerando que van a ser utilizados por sistemas reales. En este capítulo se realiza dicha implementación sobre un robot de bajo coste, el Turtlebot, en entornos tanto reales como de simulación.

El hardware, por su naturaleza de bajo coste, presenta limitaciones tales como poca resolución en los encoder, que el comportamiento del robot no se puede modelar como un sistema diferencial y la necesidad de hacer los avances y los giros como acciones separadas con una pausa intermedia. Estos problemas intrínsecos del robot, unidos a los de precisión que aparecen en cualquier sistema real, hacen que el robot sea incapaz de seguir las trayectorias con precisión absoluta. Dadas las limitaciones del robot empleado no se pueden extraer unas conclusiones fiables sobre el rendimiento del algoritmo en un sistema real. Sin embargo, las simulaciones muestran que que valorando dichas limitaciones el rendimiento de los algoritmos es correcto, con errores medios de 20 cm. Además, la arquitectura software creada en ROS es válida

para otros robots similares dado que las partes relativas al planificador se han desarrollado de forma independiente al sistema en el que se usarán.

Capítulo 7

Conclusiones y trabajo futuro

Aunque a lo largo del presente documento se han realizado conclusiones sobre el contenido de cada uno de los capítulos, en este capítulo se presentarán las conclusiones y trabajo futuro planteadas de forma global.

7.1. Conclusiones

Como se ha mostrado en publicaciones y tesis anteriores, Fast Marching Method se plantea como un método versátil y útil en distintas aplicaciones independientes a la planificación de trayectorias, que es el área que nos ocupa. Dentro de este campo, Fast Marching Method se plantea como un método que se puede utilizar no solo en planificación, sino también en tareas más complejas que tengan un paso de planificación, tal y como se plantea en (Garrido, 2008), (Sanctis, 2010),(Jurewicz, 2010),(Gómez, 2012).

Fast Marching Method, por sus características, posee ciertas limitaciones para su uso directo como planificación de trayectorias. Las soluciones que proporciona FMM se corresponden con el camino más óptimo, en términos de distancia, para unir dos puntos. Por ese motivo, las trayectorias no buscan ser suaves ni tienen en cuenta la distancia con los obstáculos, por lo un robot, por

sus limitaciones físicas, no puede seguirlas. De esa limitación nace Fast Marching Square, un método basado en FMM que propone el uso de mapas de velocidad para solucionar sus inconvenientes. No obstante, el método no es perfecto y pese a que proporciona trayectorias suaves y alejadas de los obstáculos sin tener en cuenta la geometría del escenario.

El objetivo de esta tesis consiste en desarrollar variaciones del método FM^2 que palíen sus defectos o mejoren sus características. El primero de ellos es Fast Marching Square Star, un método que añade una heurística a la expansión de la onda que direccione su expansión hacia el objetivo. De ese modo se obtiene una trayectoria muy similar a la obtenida por Fast Marching Square pero empleando un menor tiempo de cómputo. Esta aproximación fue realizada por Alberto Valero en (Valero-Gomez y cols., 2013), pero en el presente documento se propone una alternativa que mejora los resultados en cuanto a tiempo de ejecución.

El segundo método propuesto es Fast Marching Square Directional, un método que busca tener en cuenta la geometría del escenario para obtener trayectorias más eficientes en cuanto al entorno, tanto por distancia como por velocidad.

A lo largo de los capítulos se han ido desarrollando métodos que cumplan distintas características, sin llegar a obtener un método que englobe todas. El método más suave continúa siendo FM^2, frente a FM^2*, método con el que se obtiene una suavidad ligeramente menor a cambio de una bajada considerable en tiempos de cómputo. Finalmente, FM^2 Directional se colocan como los método que permiten recorrer las trayectorias en un menor intervalo de tiempo a costa de ser también los más lentos y menos suaves. Por tanto, la elección de cualquiera de los métodos está condicionada por la aplicación.

Para concluir, la tesis también busca comprobar que los resultados teóricos son posibles en la práctica. Es habitual desarrollar métodos que funcionen en entornos de simulación muy controlados pero que, por sus características, no

resulten tan satisfactorios a nivel práctico. Los algoritmos comentados anteriormente se han implementado en el robot Turtlebot con el fin de validar su posible uso en sistemas reales. Aunque el sistema plantea limitaciones por su propio hardware, basado en los estándares libres y en componentes de bajo coste, los resultados obtienen un rendimiento correcto dadas las circunstancias. No obstante no se pueden extraer conclusiones fiables sobre el rendimiento del algoritmo en un sistema real.

7.2. Trabajo futuro

Leyendo el presente documento, y especialmente las conclusiones, aparece un trabajo futuro obvio. Los algoritmos han sido probados en un robot de bajo coste que si bien aportaba numerosas facilidades para la integración de los mismos, sus limitaciones de hardware han dificultado la elaboración de un análisis exhaustivo. No obstante, la arquitectura software diseñada permite su exportación a otros sistemas, por lo que aparece la necesidad de integrar los algoritmos en un sistema con menos limitaciones.

Por otro lado, durante el desarrollo de FM^2 Directional han surgido distintas posibilidades, pero en especial se ha valorado la posibilidad de utilizar el gradiente de dirección de la velocidad y de expansión de la onda para modificar las velocidades. Este trabajo ha sido desechado por problemas de base ya que no contempla la existencia de esquinas en los elementos en el criterio comparativo de gradientes y la variación de velocidad propuesta no es dinámica, sino que se sustituye directamente por la máxima. Modificando dicho criterio de forma dinámica en base al gradiente de velocidad se puede solventar el problema.

Otro método propuesto se basa en combinar el trabajo de FM^2 Directional con el de FM^{2*} para obtener trayectorias con las características de FM^2 Directional, pero reduciendo el tiempo de cómputo como propone FM^{2*}. Aunque en varias pruebas los resultados han sido favorecedores, con

resultados análogos a los obtenidos al comparar FM^{2*} frente a FM^2, en este caso aparece un problema derivado de los cambios bruscos de velocidad. Estos cambios producen saltos bruscos en el mapa de tiempos de llegada utilizado para obtener la trayectoria. Esto genera que, al aplicar el gradiente, en ocasiones no se pueda encontrar una solución y hace que el método pierda estabilidad y robustez. Este problema se podría aplacar notablemente al modificar las velocidades de forma gradual, en lugar de cambiarla por la máxima directamente. De este modo los saltos serían más graduales y el gradiente podría encontrar una solución en todos los casos.

Apéndice A

Lista de Acrónimos

Acrónimos

BFMT Bidirectional Fast Marching Tree

EVT Extended Voronoi Transform

FMM Fast Marching Method

FM^2 Fast Marching Square

FMT Fast Marching Tree

FSM Fast Sweeping Method

HCM Heap-Cell Method

HMM Group Marching Method

ML-RRT Manhattan-like Rapidly-exploring Random Trees

MLT-RRT Manhattan-like Transition-based Rapidly-exploring Random Trees

MPLB Motion Planning using Lower Bounds

PRM Probabilistic Roadmap

RRT Rapidly-exploring Random Trees

SRT Sampling-based Roadmap of Trees

T-RRT Transition-based Rapidly-exploring Random Trees

UFMM Untidy Fast Marching Method

VFM Voronoi Fast Marching

Apéndice B

Utilización de paquetes de ROS

B.1. Introducción

En este apéndice se va a realizar una descripción para utilizar los paquetes de ROS desarrollados para esta tesis. Esta descripción incluirán los enlaces de descarga de los paquetes, las dependencias de los mismos y los requisitos de ROS para poder utilizarlos.

B.2. Dependencias de paquetes

En esta sección se detallarán las dependencias necesarias para poder compilar y ejecutar los módulos. Las dependencias son las siguientes:

1. **Ubuntu**: Oficialmente ROS se encuentra soportado únicamente por el sistema operativo Ubuntu. Estos paquetes han sido desarrollados para las versiones Groovy e Hydro de ROS, que funcionan sobre la versión 12.04 de Ubuntu, la cual puede descargarse desde su página web oficial[1].

2. **ROS**: Es el núcleo sobre el que funcionan los paquetes. Las versiones soportadas por los paquetes son Groovy e Hydro, ya que son las

[1]Página web oficial: http://www.ubuntu.com/

versiones más actuales con soporte completo para el Turtlebot. Para instalarlo se necesita tener instalado el sistema operativo Ubuntu 12.04, y las instrucciones de instalación se encuentran en la página web oficial de ROS[2][3]. Se recomienda instalar la versión *full* para evitar problemas posteriores.

3. **Soporte para Turtlebot en ROS**: Se compone del conjunto de paquetes que contienen todos los elementos de bajo y alto nivel que permiten controlar el robot, incluyendo los elementos necesarios para su puesta en funcionamiento tanto en Gazebo como en la plataforma real. Las instrucciones de instalación aparecen en la página web oficial de ROS en el Turtlebot[4][5]. Para evitar posibles problemas se recomienda seguir las instrucciones correspondientes a *Debs Installation*.

4. **GCC 4.8**: Para poder compilar el paquete de Fast Marching es necesario tener una versión de GCC 4.8 o superior, con soporte para *C++11*. Existen multitud de métodos para instalar esta versión en Ubuntu 12.04, pero se recomienda introducir los siguientes comandos en un terminal:

sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-4.8 g++-4.8
sudo update-alternatives –install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50
sudo update-alternatives –install /usr/bin/g++ g++ /usr/bin/g++-4.8 50

[2]Instrucciones de instalación en Groovy: wiki.ros.org/groovy/Installation/Ubuntu
[3]Instrucciones de instalación en Hydro: wiki.ros.org/hydro/Installation/Ubuntu
[4]Instalación de Turtlebot en Groovy: wiki.ros.org/turtlebot/Tutorials/groovy/Installation
[5]Instalación de Turtlebot en Hydro: wiki.ros.org/turtlebot/Tutorials/hydro/Installation

5. **Boost**: Es una librería básica sobre la que se desarrolla el paquete Fast Marching. La versión utilizada se corresponde a la 1.55, y se puede descargar desde su página web oficial[6]. Una vez descargada se ha de descomprimir y colocar la carpeta, de nombre *boost_1_55_0*, en la carpeta personal del usuario.

B.3. Preparación de paquetes

En esta sección se detallarán los pasos para configurar el espacio de trabajo de ROS y descargar y compilar los paquetes utilizados en esta tesis. Asumiendo que se ha cumplimentado los requisitos detallados en la sección B.2, los pasos son los siguientes:

1. **Espacio de trabajo de ROS**: Para poder compilar y ejecutar los paquetes de ROS desarrollados por terceros, como es el caso, es necesario tener configurado un espacio de trabajo en el que alojar los códigos fuente. Una vez instalado instalado ROS en el ordenador, los pasos para crear un espacio de trabajo se detallan en su web oficial[7]. Se recomienda crear un espacio de trabajo *catkin* para evitar problemas con estos paquetes.

2. **Compilación de los paquetes de ROS**: Para poder realizar la compilación es necesario descargar los tres paquetes, alojados en un repositorio *GitHub*. Los paquetes son:

   - ***turtlebot_fm***. Disponible en `https://github.com/jpardeiro/turtlebot_fm`.
   - ***turtlebot_move***. Disponible en `https://github.com/jpardeiro/turtlebot_move`.

[6]Página web de Boost: http://www.boost.org/users/history/version_1_55_0.html
[7]Creación de espacio de trabajo en ROS:
http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

- ***fastmarching***. Disponible en `https://github.com/jpardeiro/fastmarching_node`.

Una vez descargados los tres paquetes se han de colocar en la carpeta *src*, dentro del espacio de trabajo de ROS creado en el paso anterior. Para poder compilar los paquetes se ha de utilizar un terminal que se encuentre en la ruta del espacio de trabajo. En dicho terminal se ha de utilizar el comando *catkin_make*, y si todos los pasos se han realizado correctamente, ROS comenzará a compilar los paquetes sin lanzar ningún error.

B.4. Instrucciones de uso

En esta sección se detallarán las instrucciones de uso de los paquetes propuestos en esta tesis, ejemplificando con un mapa adjunto con los paquetes. Para ello se asume que se han realizado correctamente los pasos de la sección B.3 y los paquetes están listos para ser utilizados. La única diferencia entre ejecutarlo sobre el Turtlebot y hacerlo sobre Gazebo se encuentra en el primer paso, por lo que se explicarán las dos posibilidades.

Los pasos necesarios se muestran a continuación, y aunque el orden entre los pasos 2 y el 6 es indiferente, por seguridad se recomienda seguir el orden establecido. Todos los pasos que requieren de introducir un comando en terminal necesitan una ventana o una pestaña nueva, ya que cada nodo no liberará el terminal hasta que haya finalizado.

1. **Arrancar el robot**: El primer paso consiste en poner a funcionar el robot. Para ello existen dos posibilidades, dependiendo de si se va a funcionar en simulación (en Gazebo) o en el sistema real:

   - **Gazebo**: En un terminal se escribe el siguiente comando:

     roslaunch turtlebot_gazebo turtlebot_empty_world.launch

- **Robot real**: En un terminal se escribe el siguiente comando:

  roslaunch turtlebot_bringup minimal.launch

2. **Paquete *turtlebot_move***: Se lanza todo el paquete utilizando un *launcher*, escribiendo en un terminal el siguiente comando:

   roslaunch turtlebot_move turtlebot_move.launch

3. **Nodo *send_path***: Se lanza el nodo *send_path* del paquete *turtlebot_fm* de forma individual, escribiendo el siguiente comando en un terminal:

   rosrun turtlebot_fm send_path

4. **Nodo *load_map***: Se lanza el nodo *load_map* del paquete *turtlebot_fm* de forma individual, escribiendo el siguiente comando en un terminal:

   rosrun turtlebot_fm load_map

   Una vez funcionando el nodo pedirá la orientación del robot inicial respecto al mundo para poder ajustar todos los movimientos posteriores.

5. **Nodo planificador**: Se lanza un nodo correspondiente con el planificador. El paquete Fast Marching incluye tres nodos diferentes, correspondientes con los métodos de planificación FM^2, FM^{2*} y FM^2 Directional. Para lanzar uno de ellos se siguen los siguientes pasos:

   - **FM^2**: Para utilizar FM^2 como planificador de trayectorias se escribe el siguiente comando en un terminal:

     rosrun fastmarching fm2

   - **FM^{2*}**: Para utilizar FM^{2*} como planificador de trayectorias se escribe el siguiente comando en un terminal:

     rosrun fastmarching fm2star

- **FM^2 Directional**: Para utilizar FM^2 como planificador de trayectorias se escribe el siguiente comando en un terminal:

  rosrun fastmarching fm2directional

6. **Cargar mapa**: Es necesario cargar un mapa para que el sistema pueda funcionar correctamente utilizando el nodo *map_server* de ROS. En el paquete *turtlebot_fm* se incluye un mapa de ejemplo en la carpeta *map*. Para cargar el mapa hay que proporcionar la ruta del mapa, por lo que en este ejemplo se supondrá que el espacio de trabajo se encuentra en la ruta que se propone en la sección B.3. El comando que se ha de introducir en un terminal es el siguiente:

   rosrun map_server map_server
   ~/catkin_ws/src/turtlebot_fm/map/map_example.yaml

7. **Visualizador**: Para poder visualizar el mapa y la posición del robot en ella se utiliza el visor RViz. Este visor tiene multitud de opciones de configuración, pero en el paquete *turtlebot_fm* se ha incluido una configuración por defecto. Para cargar el visualizador con esa configuración se escribe el siguiente comando en un terminal:

   roslaunch turtlebot_fm view_robot.launch

   Hasta que no se realice el siguiente paso, en el visualizador únicamente se mostrará el mapa cargado en el paso anterior, sin la presencia del robot en ningún punto. Un ejemplo se puede ver en la figura B.1.

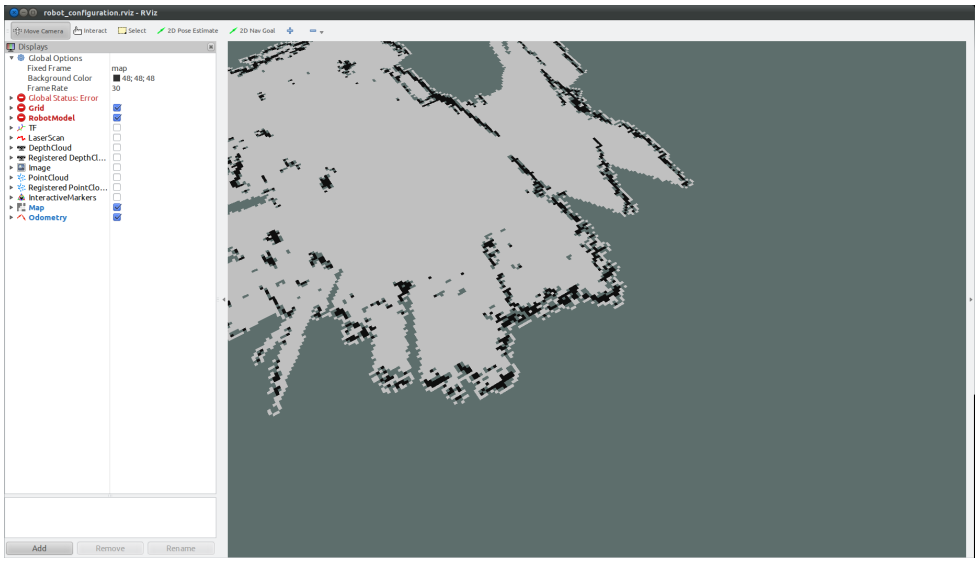
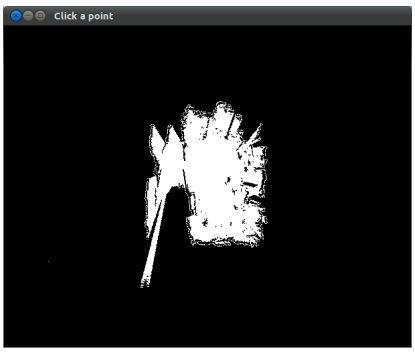


Figura B.1: *Visualización del mapa en RViz.*

8. **Seleccionar puntos de inicio y final**: Una vez realizado los pasos anteriores correctamente saltará una ventana con el mapa, mostrada en la figura B.2a. Sobre este mapa se seleccionarán los puntos de inicio y de fin de la trayectoria haciendo click sobre los puntos, tras lo cual se obtendrá la trayectoria, mostrada en otra ventana como se puede ver en la figura B.2b.



(a) *Mapa utilizado.*

(b) *Trayectoria de ejemplo sobre el mapa.*

Figura B.2: *Generar trayectorias utilizando los paquetes de ROS.*

A partir de este momento se podrá visualizar desde la ventana de RViz el movimiento del robot dentro del mapa, como se muestra en la figura B.3. La posición en RViz se irá actualizando hasta que la trayectoria finalice y por tanto el robot se quede estático en el punto final.

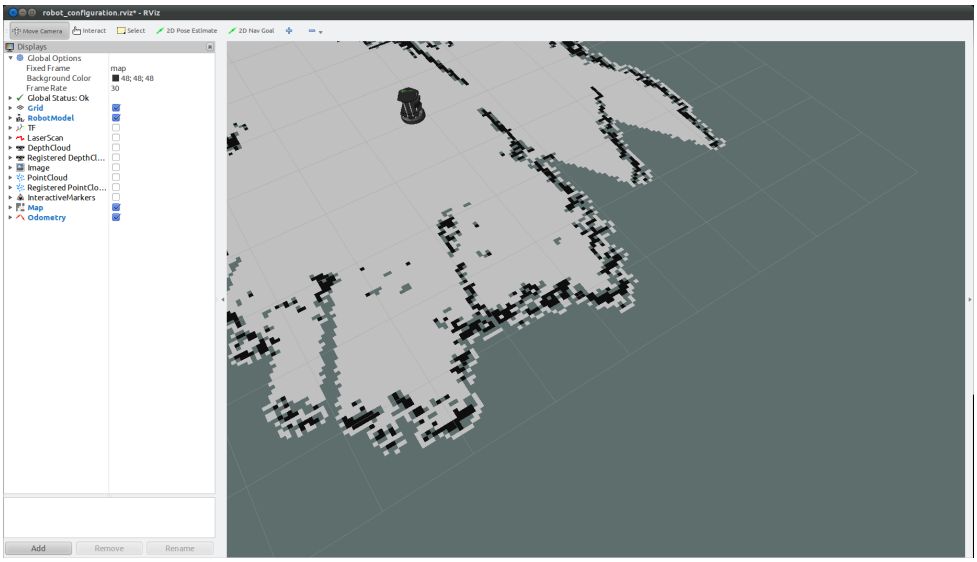


Figura B.3: *Visualización del robot y el mapa en RViz.*

Referencias

Alvarez, D., Gomez, J., Garrido, S., y Moreno, L. (2014, May). 3D robot formations planning with Fast Marching Square. En *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on* (pp. 59–64).

Amorim, D., y Ventura, R. (2014, May). Towards efficient path planning of a mobile robot on rough terrain. En *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on* (pp. 22–27).

Ansuategui, A., Arruti, A., Susperregi, L., Jauregi, E., Lazkano, E., Sierra, B., y cols. (2014). Robot Trajectories Comparison: a Statistical Approach. *The Scientific World Journal*.

Asano, T., Asano, T., Guibas, L. J., Hershberger, J., y Imai, H. (1986). Visibility of Disjoint Polygons. *Algorithmica*, *1*(1), 49–63.

Attali, D., Boissonnat, J.-D., y Lieutier, A. (2003). Complexity of the Delaunay Triangulation of Points on Surfaces: the Smooth Case. En *Symposium on Computational Geometry* (pp. 201–210).

Berg, M. de, Kreveld, M. van, Overmars, M., y Schwarzkopf, O. (2008). *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag.

Bialkowski, J., Karaman, S., Otte, M., y Frazzoli, E. (2012). Efficient collision checking in sampling-based motion planning. En *Workshop on Algorithmic Foundations of Robotics (WAFR).*

Borlaug, I. (2007). *Seismic processing using Parallel 3D FMM.* Institutt for datateknikk og informasjonsvitenskap.

Chacon, A., y Vladimirsky, A. (2013, junio). A parallel Heap-Cell Method for Eikonal equations. *ArXiv e-prints*.

Cortes, J., Jaillet, L., y Siméon, T. (2008). Disassembly Path Planning for Complex Articulated Objects. *IEEE Transactions on Robotics*, *24*(2), 475–481.

Delaunay, B. (1934). Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, *7*, 793–800.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*, 269–271.

Do, Q. H., Mita, S., y Yoneda, K. (2014, June). Narrow passage path planning using fast marching method and support vector machine. En *Intelligent Vehicles Symposium Proceedings, 2014 IEEE* (pp. 630–635).

Ettlin, A., y Bleuler, H. (2006). Randomised Rough-Terrain Robot Motion Planning. En *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 5798–5803). IEEE.

Fox, D., Burgard, W., Dellaert, F., y Thurn, S. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots Dieter Fox, Wolfram Burgard. En *In Proc. of the National Conference on Artificial Intelligence (AAAI* (pp. 343–349).

Fu, Z., Jeong, W.-K., Pan, Y., Kirby, R. M., y Whitaker, R. T. (2011, octubre). A Fast Iterative Method for Solving the Eikonal Equation on Triangulated

Surfaces. *SIAM J. Sci. Comput.*, *33*(5), 2468–2488. Descargado de `http://dx.doi.org/10.1137/100788951`

Garrido, S. (2008). *Robot Planning and Exploration Using Fast Marching*. Tesis de Master no publicada, Carlos III University of Madrid.

Garrido, S., Moreno, L., Abderrahim, M., y Blanco, D. (2009). FM2: A Real-time Sensor-based Feedback Controller for Mobile Robots. *International Journal of Robotics and Automation*, *24*(1), 3169–3192.

Garrido, S., Moreno, L., Abderrahim, M., y Martin, F. (2006). Path Planning for Mobile Robot Navigation Using Voronoi Diagram and Fast Marching. En *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2376–2381).

Garrido, S., Moreno, L., y Blanco, D. (2008). Exploration of a Cluttered Environment using Voronoi Transform and Fast Marching Method. *Robotics and Autonomous Systems*, *56*(12), 1069–1081.

Garrido, S., Moreno, L., y Blanco, D. (2009). Exploration and Mapping Using the VFM Motion Planner. *IEEE Transactions on Instrumentation and Measurement*, *58*(8), 2880–2892.

Garrido, S., Moreno, L., Blanco, D., y Martin, F. (2009). Smooth Path Planning for Non-holonomic Robots using Fast Marching. En *Proceedings of the 2009 IEEE International Conference on Mechatronics* (pp. 1–6).

Garrido, S., Moreno, L., Blanco, D., y Muñoz, M. L. (2007). Sensor-based Global Planning for Mobile Robot Navigation. *Robotica*, *25*(2), 189–199.

Ghosh, S. K., y Mount, D. M. (1991). An Output-sensitive Algorithm for Computing Visibility Graphs. *SIAM Journal on Computing*, *20*(5), 888–910.

Gómez, J. V. (2012). *Advanced Applications of the Fast Marching Square Planning Method*. Tesis de Master no publicada, Carlos III University of Madrid.

Gomez, J. V., Álvarez, D., Garrido, S., y Moreno, L. (2013). Improving Sampling-Based Path Planning Methods with Fast Marching. En *ROBOT 2013: First Iberian Robotics Conference - Advances in Robotics, Vol. 2, Madrid, Spain, 28-29 November 2013* (pp. 233–246).

González, V., Monje, C. A., y Balaguer, C. (2014). Planificación de misiones de vehículos aéreos no tripulados con Fast Marching en un entorno 3D . En *IEEE International Colloquium on Signal Processing and its Applications* (pp. 352–357).

Hart, P., Nilsson, N., y Raphael, B. (1968, july). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, *4*(2), 100–107.

Hwang, Y., y Ahuja, N. (1992, Feb). A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, *8*(1), 23–32.

Iehl, R., Cortés, J., y Siméon, T. (2012). Costmap Planning in High Dimensional Configuration Spaces . En *IEEE/ASME International Conference on Advanced Intelligent Mechatronics* (pp. 166–172).

Jaillet, L., Cortés, J., y Siméon, T. (2010). Sampling-Based Path Planning on Configuration-Space Costmaps. *IEEE Transactions on Robotics*, *26*(4), 635–646.

Janson, L., y Pavone, M. (2014). Fast Marching Trees: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions - Extended Version. *CoRR*, *abs/1306.3532*. Descargado de `http://arxiv.org/abs/1306.3532`

Javier V. Gómez, N. M., y Garrido, S. (2013). Social Path Planning: Generic Human-Robot Interaction Framework for Robotic Navigation Tasks. En *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013).*

Jeong, W.-k., Ross, y Whitaker, T. (2007). A fast eikonal equation solver for parallel systems. En *In SIAM conference on Computational Science and Engineering.*

Jeong, W.-k., y Whitaker, R. T. (2008). *A fast iterative method for Eikonal equations.*

Jurewicz, P. (2010). *Smooth Path-planning for a Robot Manipulator Using Probabilistic Methods*. Tesis de Master no publicada, Carlos III University of Madrid.

Karaman, S., y Frazzoli, E. (2011). Incremental Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, *30*(7), 846–894.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

LaValle, S. M. (2011). Motion Planning. *IEEE Robotics and Automation Magazine*, *18*(1), 79–89.

Luo, J., y Hauser, K. (2014). An Empirical Study of Optimal Motion Planning. En *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).*

Malfaz, M., Garrido, S., y Blanco, D. (2012). Application of the Fast Marching Method for Outdoor Motion Planning in Robotics. *Robotics and Autonomous Systems*(Accepted).

Masehian, D. J. . E. (2012). Car-like Robots Motion Planning by the Fast Marching Method (FMM). *International Journal of Industrial Engineering & Production Management*, *23*(2), 227–238.

Masoud, S., y Masoud, A. (2002, Nov). Motion planning in the presence of directional and regional avoidance constraints using nonlinear, anisotropic, harmonic potential fields: a physical metaphor. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *32*(6), 705–723.

Muñoz, P. (2014). *Implementación y análisis del algoritmo Fast Marching y sus distintas versiones*. Tesis de Master no publicada, Carlos III University of Madrid.

Osher, S., y Sethian, J. A. (1988). Fronts Propagating with Curvature-dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, *79*(1), 12–49.

Pan, J., Chitta, S., y Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. En *ICRA* (pp. 3859–3866). IEEE. Descargado de `http://dblp.uni-trier.de/db/conf/icra/icra2012.html#PanCM12`

Pan, J., Sucan, I. A., Chitta, S., y Manocha, D. (2013). Real-time collision detection and distance computation on point cloud sensor data. En *ICRA'13* (pp. 3593–3599).

Qureshi, A., Iqbal, K., Qamar, S., Islam, F., Ayaz, Y., y Muhammad, N. (2013, Aug). Potential guided directional-RRT* for accelerated motion planning in cluttered environments. En *Mechatronics and Automation (ICMA), 2013 IEEE International Conference on* (pp. 519–524).

Rasch, C., y Satzger, T. (2007). Remarks on the O(N) Implementation of the Fast Marching Method. *CoRR*, *abs/cs/0703082*. Descargado de `http://dblp.uni-trier.de/db/journals/corr/corr0703.html#abs-cs-0703082`

Salzman, O., y Halperin, D. (2014). Asymptotically Near-Optimal Motion Planning using Lower Bounds on Cost. *CoRR*, *abs/1403.7714*. Descargado de `http://arxiv.org/abs/1403.7714`

Sanctis, L. D. (2010). *Determinación de Trayectorias Óptimas Sobre Superficies Tridimensionales Utilizando Fast Marching*. Tesis de Master no publicada, Carlos III University of Madrid.

Seongjai Kim, D. F. (2000). *The Group Marching Method: An O(N) Level Set Method* (Inf. Téc.). University of Kentucky,.

Sethian, J. A. (1996). *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

Sethian, J. A. (1999). *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press.

Shkolnik, A. C., y Tedrake, R. (2009). Path Planning in 1000+ Dimensions Using a Task-space Voronoi Bias. En *IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2061–2067).

Starek, J. A. S. E. J. L., y Pavone, M. (2014). Bidirectional Fast Marching Trees: An Optimal Sampling-Based Algorithm for Bidirectional Motion Planning. *CoRR*.

Valero-Gomez, A., Gomez, J. V., Garrido, S., y Moreno, L. (2013). The Path to Efficiency: Fast Marching Method for Safer, More Efficient Mobile Robot Trajectories. *IEEE Robot. Automat. Mag.*, *20*(4), 111–120.

Wikipedia. (2014a). *Kinect — Wikipedia, The Free Encyclopedia.* Descargado de `http://en.wikipedia.org/wiki/Kinect` ([Online; accedido 3-Noviembre-2014])

Wikipedia. (2014b). *Robot Operating System — Wikipedia, The Free Encyclopedia.* Descargado de `http://en.wikipedia.org/wiki/Robot_Operating_System` ([Online; accedido 3-Noviembre-2014])

Wikipedia. (2014c). *Roomba — Wikipedia, The Free Encyclopedia.* Descargado de `http://en.wikipedia.org/wiki/Roomba` ([Online; accedido 3-Noviembre-2014])

Wikipedia. (2014d). *Ubuntu (operating system) — Wikipedia, The Free Encyclopedia.* Descargado de `http://en.wikipedia.org/wiki/Ubuntu_(operating_system` ([Online; accedido 3-Noviembre-2014])

Xu, B., Stilwell, D., y Kurdila, A. (2013). Fast Path Re-planning Based on Fast Marching and Level Sets. *Journal of Intelligent & Robotic Systems*, *71*(3-4), 303–317. Descargado de `http://dx.doi.org/10.1007/s10846-012-9794-2`

Yatziv, L., Bartesaghi, A., y Sapiro, G. (2005). O(N) Implementation of the Fast Marching Algorithm. *Journal of Computational Physics*, *212*, 393–399.

Yershov, D., y Frazzoli, E. (2014). Asymptotically Optimal Feedback Planning: FMM Meets Adaptive Mesh Refinement. En *Workshop on Algorithmic Foundations of Robotics (WAFR).* Istanbul, Turkey.