Ingeniería Informática

Curso académico 2015/2016

Proyecto Fin de Carrera

Docencia en robótica: Prácticas de navegación global

Autor:
Víctor Nieto Lobo
Tutor:
José María Cañas Plaza

Resumen

Este proyecto trata sobre el desarrollo, en la plataforma JdeRobot, de dos componentes académicos que servirán como material de prácticas para la enseñanza de dos técnicas de navegación global sobre un robot simulado. Las técnicas que el alumno podrá desarrollar sobre estos componentes son Grandient Path Planning y Grafo de Visibilidad. Estas técnicas son empleadas para guiar al robot desde una posición origen a una posición destino, evitando colisionar con los obstáculos que están descritos en el mapa.

El objetivo de estos componentes académicos es abstraer a los alumnos de todas las complejidades de la interfaz gráfica, conexiones con los sensores y actuadores... y que pueda centrarse únicamente en la resolución del algoritmo de navegación global. Desde el primer momento que el alumno desarrolle sus primeros pasos, podrá visualizar sus progresos gracias a estos componentes.

Además de la preparación del entorno de prácticas, se ha desarrollado una solución concreta a estas técnicas de navegación global, que sirve de ejemplo para los alumnos. Estas soluciones cubren la interpretación de un mapa del entorno, la planificación de la trayectoria sobre esta, y por último, la ejecución de esta trayectoria desarrollando el pilotaje que lleva a cabo el robot.

Este proyecto ha sido desarrollado sobre el sistema operativo Linux, utilizando diferentes herramientas proporcionadas por JdeRobot o ajenas a ella, como Gazebo. El lenguaje de programación utilizado para el desarrollo ha sido C++ utilizando como plataforma de desarrollo QT Creator.

Capítulo 1

Introducción

El afán del ser humano por fabricar máquinas capaces de realizar tareas independientes ha sido una constante de la historia. Durante siglos el ser humano ha tratado de construir máquinas que se encarguen de realizar las tareas más pesadas, o máquinas que simplemente imiten comportamientos del ser humano. El concepto de máquinas automatizadas se remonta a la antigüedad. Los autómatas, o máquinas semejantes a personas ya se manifestaban en los relojes de las iglesias medievales o en la construcción de brazos mecánicos en las estatuas de los antiguos egipcios.

Ideas como el desarrollo de herramientas especializadas, el control por realimentación o la división del trabajo en tareas más pequeñas que pudiesen realizar máquinas fueron los ingredientes esenciales en la automatización de las fábricas en la revolución industrial del siglo XVIII. En los comienzos de esta etapa, se descubrió la máquina de vapor, con el objetivo principal de poder desplazar máquinas de un lado a otro. Estas máquinas eran peligrosas, ya que producían explosiones en las calderas por lo que fue necesario regular automáticamente el suministro de vapor para poder controlar estas explosiones. En 1787, Sir James Watt introdujo el regulador centrífugo de bolas, cuya función principal consistía en la movilización de una válvula que controlaba el suministro de vapor, proporcionando en todo momento la cantidad que la máquina necesitase, evitando sobrecalentamientos y por lo tanto las explosiones.

En el primer tercio del siglo XX se inicia el desarrollo de la ingeniería en ramas muy diversas como la mecánica, la electrónica, la informática, las telecomunicaciones, la inteligencia artificial... que permitirá la construcción de robots modernos. Los acontecimientos científicos y técnicos que tienen que ver con la creación y el desarrollo de la robótica en la historia, no se limita a la ingeniería, sino que involucra a otras disciplinas, como las matemáticas, la física, la mecánica... Los avances en computación de las últimas décadas son el impulso definitivo que permite desarrollar máquinas muy cercanas al ideal de automatismo que siempre se había perseguido.

1.1. Docencia en robótica

La finalidad principal de este proyecto es la aportación de nuevos medios para el aprendizaje de la robótica, en concreto sobre el problema de la navegación global. La formación es un aspecto muy importante que favorece el desarrollo y la innovación. Es importante que la formación en robótica llegue a diferentes perfiles de usuario, ya que implica mayor iniciativa , desarrollo de nuevos productos y de nuevas líneas de mercado.

DISTRIBUCIÓN POR CURSOS

PRIMER CURSO		SEGUNDO CURSO		TERCER CURSO		CUARTO CURSO	
Semestre 1	Semestre 2	Semestre 3	Semestre 4	Semestre 5	Semestre 6	Semestre 7	Semestre 8
Fundamentos de Matemática Aplicada I 6 ECTS	Fundamentos de Matemática Aplicada II 6 ECTS	Ampliación de Matemática Aplicada 6 ECTS	Resistencia de Materiales 6 ECTS	Automatización 6 ECTS	Sistemas Inteligentes 6 ECTS	Manipuladores 6 ECTS	Sistemas Multirobot 6 ECTS
Fundamentos Físicos de la Ingeniería I 6 ECTS	Fundamentos Físicos de la Ingeniería II 6 ECTS	Ampliación de Física 6 ECTS	Fundamentos de Automática 6 ECTS	Ingeniería de Control 6 ECTS	Comunicaciones 6 ECTS	Robots Móviles 6 ECTS	Proyectos de Sistemas Robóticos 6 ECTS
Fundamentos Químicos de la Ingeniería 6 ECTS	Computadores 6 ECTS	Tecnología de Materiales 6 ECTS	Procesadores Integrados 6 ECTS	Algoritmia 6 ECTS	Programación de Robots 6 ECTS	Robótica de Servicios 6 ECTS	Trabajo Fin de Grado (1) 12 ECTS
Programación I 6 ECTS	Programación II 6 ECTS	Tecnología Eléctrica 6 ECTS	Mecanismos y Modelado de Robots 6 ECTS	Visión por Computador 6 ECTS	Control de Robots 6 ECTS	Teleoperación 6 ECTS	
Expresión Gráfica 6 ECTS	Iniciación a la Ingeniería Robótica 6 ECTS	Tecnología Electrónica 6 ECTS	Sensores e Instrumentación 6 ECTS	Sistemas Empotrados 6 ECTS	Sistemas de Percepción 6 ECTS	Empresa 6 ECTS	*A elegir entre:* -Inglés -Prácticas Externas 6 ECTS

Figura 1.1: Plan de estudios, Grado en Ingeniería Robótica, Universidad de Alicante

La docencia en robótica se puede llevar a cabo sobre estudiantes de cualquier nivel educativo. Además, la construcción de un desarrollo sobre un robot, no solo implica el desarrollo cognitivo sobre disciplinas como la geometría, trigonometría, electrónica, programación, mecánica... sino que también se desarrollan otras aptitudes como los son la ingeniería de sistemas, el diseño, conceptos de ergonomía del trabajo y la planificación.

La formación en robótica aparece de forma directa en formaciones profesionales o grados especializados. En la ESO y en el Bachiller tecnológico comienzan a aparecer contenidos relacionados indirectamente con la robótica, en asignaturas cómo "Tecnología Industrial", obligatoria en 3º y 4º de la ESO o "Tecnología 1 y 2", en la modalidad del Bachiller citado.

En educación más especializada, como grados, se está comenzando a implantar. Por ejemplo, el Grado en Ingeniería Robótica, en la Universidad de Alicante inaugurado este curso. Este grado cuenta con un plan de estudios completo mostrado en la figura 1.1.

También encontramos formación ligada directamente con la robótica en algunos Másteres, cómo el Máster en Robótica y Automatización, en la Universidad Carlos III de Madrid, o el Postgrado en Automática y Robótica en la Universidad Politécnica de Madrid.

La Universidad Rey Juan Carlos, cuenta con una asignatura en Ingeniería en Telemática, cuyo objetivo es introducir a los alumnos en el campo de la robótica. En esta asignatura, se presentan los componentes esenciales los robots móviles y los problemas de la robótica autónoma. También, cómo una formación teórica y práctica en la que los alumnos puedan afianzar todos los conceptos relacionados con la robótica. Además, esta universidad ofrece como títulos propios dos cursos, que son: Curso Superior Universitario en Robótica y Curso Superior Universitario en Programación de Robots con ROS.

En el nuevo mercado global, dominar estas tecnologías es un factor clave, ya que permiten automatizar muchos procesos de producción y es un papel decisivo al ser una alternativa de bajo

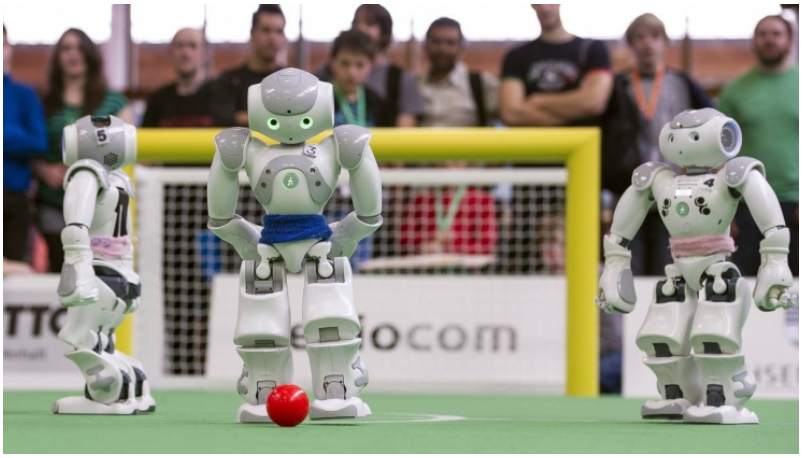

Figura 1.2: RoboCup

coste de mano de obra.

Otras actividades de divulgación dónde la docencia en robótica tiene gran repercusión, son los clubes de robótica o las competiciones de robots, como Robocup, fundado en 1997, a través de competencias integradas por robots autónomos, investigación y educación sobre la inteligencia artificial. En la figura 1.2, podemos ver una imagen de un partido de fútbol con robots autónomos «Nao».

1.2. Tipos de robots

En la actualidad, la robótica está en pleno auge, tanto es así, que el desarrollo de esta ingeniería se está abordando en diferentes ramas. La construcción de los robots es muy variada ya que la finalidad que tienen también lo es. Por ello, a continuación se detalla una clasificación de los robots creados en la actualidad según su arquitectura.En la actualidad, el desarrollo de la robótica está en pleno auge, tanto es así, que el desarrollo de esta ingeniería se está abordando en diferentes ramas. La construcción de los robots es muy variada ya que la finalidad que tienen también lo es. Por ello, a continuación se detalla una clasificación de los robots creados en la actualidad según su arquitectura.

- Brazos robóticos: En este grupo existen robots de formas y configuraciones muy diversas ya que la característica común que tienen todos ellos es estar estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número limitado de grados de libertad. Por lo general estos robots son de carácter sedentario. Los robots que forman este grupo son los manipuladores, los robots industriales, los robots cartesianos, etc. Este tipo de robots han significado un gran avance para la medicina. Se utilizan robots para uso cirujano, para terapias de rehabilitación, etc. El Sistema Quirúrgico Da Vinci es un equipo de cirugía robótica que se utiliza para múltiples procedimientos quirúrgicos. Podemos ver un ejemplo de este robot en la imagen 1.3.

- Robots móviles: Este grupo está formado por los robots con gran capacidad de desplazamiento. Las formas que adoptan generalmente son de carros o plataformas dotadas de

Figura 1.3: Brazo robótico

pequeños motores. Algunos usos de estos robots son el desplazamiento de objetos de un punto a otro, por ejemplo, el transporte de piezas en una cadena de producción. Otro uso es la autonavegación, para la exploración de mapas o la resolución de caminos posibles dentro de ellos. Estos robots pueden ser guiados a través de mapas, a través de pistas materializadas a través de la radiación electromagnética de circuitos empotrados en el suelo, o a través de los propios sensores de los que haya sido dotado detectando bandas fotoeléctricamente, señales a través de cámaras, sensores de ultrasonido con los que pueden calcular distancias a los obstáculos cercanos, etc.

- Androides: Son robots que intentan reproducir total o parcialmente la forma y el comportamiento del ser humano. Actualmente los androides son todavía dispositivos muy poco evolucionados y sin utilidad práctica. Están destinados principalmente, al estudio y experimentación en tiempo real y el mantenimiento de su equilibrio. Un ejemplo de este tipo de robots, es el robot «Nao», mostrado en la figura 1.4. Es un robot humanoide programable y autónomo, desarrollado por Aldebaran Robotics. Ofrece una unidad de medición inercial con acelerómetro, girómetro y cuatro sensores de ultrasonidos que proporcionan la estabilidad y el posicionamiento en el espacio.

- Zoomórficos: Estos robots son muy similares a los anteriores. Constituyen una clase caracterizada principalmente por sus sistemas de locomoción que imitan a los diversos seres vivos. Ejemplo de este grupo de robots es el robot «AIBO» de Sony, mostrado en la figura 1.5. Se trata de un robot autónomo, que responde a estímulos externos y es capaz de mostrar emociones y aprender.

- Híbridos: Estos robots corresponden a aquellos de difícil clasificación, cuya estructura se sitúa en combinación con alguna de las anteriores ya expuestas. Puede considerarse un robot híbridos algunos robots formados por un cuerpo formado por un carro móvil y un brazo articulado semejante al de los robots industriales.

Figura 1.4: Robot Nao

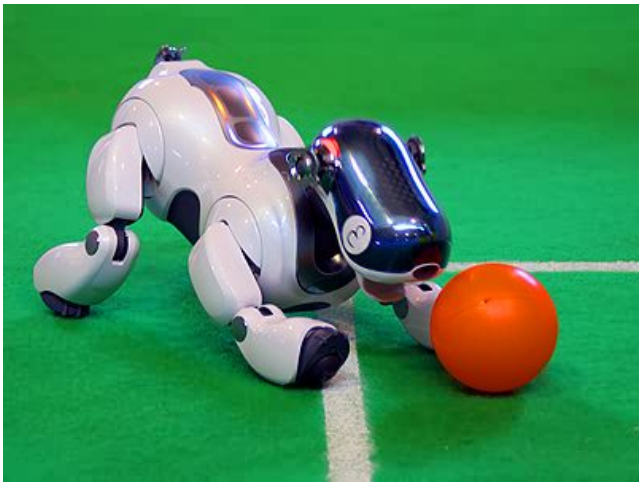

Figura 1.5: Robot AIBO

Figura 1.6: Robot RMAX

- Drones: Por definición, son aeronaves que vuelan sin tripulación. Pueden ser teleoperados o autónomos. Los drones pueden ser usado en infinidad de tareas que el ser humano no puede realizar o son peligrosas. Este tipo de robots ha evolucionado mucho en los últimos años y su aplicación es muy diversa hoy en día. Un ejemplo de sus aplicaciones es el drone «T-Hawk» que tomó fotografías del estado en que se encontraban los reactores de Fukushima. Otras aplicaciones que se dan hoy en día son los drones encargados de rescate y emergencias en las costas o en las montañas, drones que se encargan de la fumigación de grandes espacios como el robot «RMAX» de la figura 1.6, transporte y entrega de mercancías, cartografía, etc.

1.3. Navegación de robots

Como se ha comentado previamente, los robots móviles son los encargados de realizar desplazamientos dentro de un entorno de manera autónoma. Un robot móvil es un sistema que integra percepción del entorno mediante sensores o mapas, toma de decisiones y realiza acciones. Además debe disponer de un sistema de locomoción que le permita desplazarse. Existen numerosas posibilidades al respecto, ya que se pueden encontrar robots que anden, salten, vuelen, repten o sencillamente, que se mueven sobre ruedas. La mayor parte de estos mecanismos de locomoción surgen como imitación de los modos de movimiento que se pueden observar en la naturaleza. Cuando existe un desplazamiento de todo el robot, se utilizan técnicas de navegación que permiten realizar movimientos dentro de un entorno sorteando los obstáculos que puedan aparecer ante el robot cuando va desde un punto origen a un punto destino. Para realizar esta navegación entre dos puntos surgen los conceptos de la navegación global y la navegación local.

La primera trata sobre sistemas que contienen un mapa completo de un entorno y de los obstáculos presentes. Para conseguir un desplazamiento con esta metodología, además del mapa se necesita obtener en todo momento la posición que ocupa el robot dentro del mapa. En estos sistemas es posible la planificación de una ruta óptima que evite los obstáculos dado que conoce la posición exacta de inicio y la posición del mapa a la que se desea llegar. Actualmente existen

Figura 1.7: Robot Spencer

sistemas capaces de navegar en entornos conocidos como industrias, oficinas, hospitales, museos, aeropuertos... que son capaces de trasladar material entre distintas zonas de los edificios en cuestión. Como ejemplo de robots que utilizan estos sistemas de navegación global está el robot «Spencer», un robot creado por un equipo de investigadores de la Universidad de Örebro, en Suecia, que ayuda a los pasajeros de un aeropuerto a orientarse a la hora de tomar un vuelo. En la imagen 1.7 podemos ver un ejemplo de este robot.

La navegación local consiste en la realización de un desplazamiento en un entorno sorteando los obstáculos mediante información sensorial. El robot utiliza los sensores que contiene para obtener información del entorno que les rodea y utiliza esta información para decidir sus movimientos. Existen muchos tipos de sensores que se utilizan para obtener información del entorno externo. Son muy fiables a la hora de tomar decisiones sobre sus movimientos, como las cámaras o los sensores de ultrasonido, láseres, infrarrojos... Aunque no son estrictamente imprescindibles, los robots móviles suelen incorporar sensores propioceptivos, que suministran medidas relativas a su estado: velocidad, incremento de posición y aceleraciones. Los encoders situados en las ruedas del robot, por ejemplo, registran el número de revoluciones de las mismas y permiten obtener la llamada posición odométrica a partir de la estimación del movimiento relativo incremental. Aunque esta aproximación presenta una buena precisión a corto plazo, la acumulación de errores a medida que aumenta el recorrido causa grandes diferencias con la posición real. Por este motivo, hace falta disponer de sensores exteroceptivos que perciben los aspectos externos al robot.

Este tipo de navegación es más común en entornos variables, donde los obstáculos son móviles como personas, animales, vehículos, etc. Mediante sensores se puede obtener en tiempo real la posición de los obstáculos que se puede encontrar el robot. Con técnicas de navegación global, se asume que los obstáculos permanecen estáticos.

Una de las técnicas más utilizadas para la navegación local es la de los campos de fuerza virtuales «VFF». Está técnica consiste en la obtención del rumbo que ha de llevar mediante un cálculo vectorial. El robot es atraído por la posición final a la que se desea que llegue, mientras

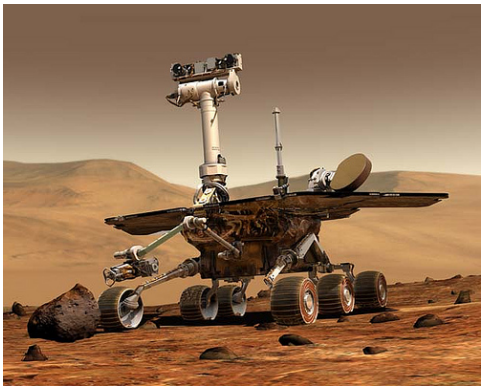

Figura 1.8: Robot Spirit

que los obstáculos que se encuentran a su alrededor ejercen una fuerza repulsiva que hace que se aleje de ellos. Con toda esta información, el robot recalcula la orientación y la velocidad que debe adoptar en sus movimientos.

El desarrollo de técnicas de navegación en robots móviles es uno de las principales problemas que se está abordando en gran medida en la actualidad. Ejemplo de ello son, por ejemplo, el coche autónomo de Google, o autopiloto Tesla, y la aspiradora Roomba.

El coche autónomo de Google combina la navegación global y la local para definir su trayectoria. Compara la vista satélital con mapas ilustrados e imágenes del mundo real. Esto hace más fácil encontrar rutas entre dos puntos. Además posee un Lidar, sobre el techo que detecta objetos y mide la distancia hasta ellos mediante rayos de luz láser. Éste se complementa con un sistema de posicionamiento GPS, y una unidad de medición inercial que mide la aceleración y la velocidad angular mediante acelerómetros, giróscopos y magnetómetros. También tiene cuatro radares que al igual que el Lidar, le sirven para detectar objetos y medir distancias. Por otro lado, cuenta con una cámara que reconoce las señales de tráfico, los semáforos y las líneas de la calzada. En cuanto a la navegación global, otro elemento imprescindible son los mapas. En la figura 1.9 se muestra el coche diseñado por Google.

Otro ejemplo actual de navegación autónoma son las aspiradoras Roomba. Utiliza un sistema de navegación llamado «IAdapt» que proporciona un método eficaz de limpieza y detección de obstáculos, incluye sensores anticaída y un sistema llamado «Dirt Detect» que detecta la suciedad utilizando los sensores ópticos y acústicos integrados en la parte inferior de la aspiradora. También es capaz de adaptar el patrón de limpieza en función de los obstáculos encontrados en la estancia y modificar la pauta de limpieza para obtener mejores resultados en el menor tiempo posible.

Además, incluye dos módulos "Virtual Wall Lighthouse", que consisten en dos dispositivos que actúan como barrera virtual para limitar o condicionar el perímetro de trabajo del dispositivo. Estos muros virtuales actúan de dos formas: como pared virtual que impide la entrada o salida del robot a determinadas zonas de la casa, y como indicador de estancias, que ayuda a la aspiradora a diferenciar una habitación de otra. Este aspirador también cuenta con una cámara que registra elementos desde el suelo hasta 45 grados. Con ésta y gracias a la ayuda de la tecnología VSLAM, es capaz de crear y localizarse dentro de un mapa de las habitaciones de la casa. Lo hace de forma dinámica, es decir, tiene en cuenta los nuevos objetos que se encuentra en su movimiento. En la

Figura 1.9: Google car

figura 1.10 podemos ver un ejemplo de trayectoria del aspirador Roomba utilizando la tecnología IAdapt.

1.4. Antecedentes

Para la realización de este proyecto me he apoyado en proyectos de alumnos que han realizado trabajos similares que me han servido de guía. Un ejemplo de ellos es el Proyecto de Fin de Carrera de [Raúl Isado, 2015]. El objetivo de su proyecto fue conseguir dotar al robot Pioneer, mostrado en la figura 1.11, de movimiento autónomo en entornos conocidos de oficina, a través de navegación local y navegación global. La elaboración de estos algoritmos de navegación los llevó a cabo en un simulador. El algoritmo utilizado en la navegación global fue *Gradient Path Planning*. Esta navegación es la que predomina en el robot hasta que el algoritmo de navegación local se activa. Este algoritmo es VFF (Virtual Force Field). El destino a alcanzar le ejerce una fuerza atractiva en la dirección destino a la que se debe dirigir, mientras que cada obstáculo percibido, ejerce una fuerza repulsiva proporcional a la distancia que se encuentra de él. Con una suma vectorial de todas estas fuerzas se obtiene la dirección que debe tomar el robot en ese momento. Estos algoritmos se ejecutan concurrentemente, aunque el algoritmo de navegación global es el que marca la dirección destino que debe de llevar el robot. Con el algoritmo de navegación local se consigue evitar obstáculos en tiempo real, obstáculos que no están descritos en el mapa.

Otro ejemplo de Proyecto de Fin de Carrera interesante es el de [Julio Vega, 2008].El Proyecto estaba basado en la navegación de un robot en un espacio determinado. Barajó la posibilidad de resolver el problema de la navegación global a través de técnicas como Grafo de Visibilidad o Diagramas de Voronoi, aunque finalmente optó por la utilizar GPP (*Gradient Path Planing*). Como técnica de navegación local, al igual que Raúl Isado, utiliza VFF (Virtual Force Field) y le añade una ventana de seguridad, que advierte en todo momento del entorno más próximo que rodea al robot. Esta ventana está dividida en dos laterales, izquierdo y derecho, frontal e interior. El control implementado con esta ventana está basado en casos. El comportamiento del robot variará dependiendo de si hay obstáculos en cualquiera de las secciones en la que está dividida la ventana.

Figura 1.10: Aspirador Roomba

Figura 1.11: Robot Pioneer

Para el desarrollo del algoritmo Grafo de Visibilidad (VG) me he apoyado también en el Proyecto de Fin de Carrera de [Alejandro López, 2005]. Su proyecto está enfocado en la creación de este algoritmo de navegación global utilizando el robot Pioneer como sujeto de movimiento. Trató de interpretar un mapa y mediante este algoritmo localizar todos los caminos posibles por los que podía moverse dicho robot. Una vez tenía el mapa VG construido, para elegir la ruta más adecuada, utilizaba los algoritmos de Dijkstra y A*. El primero trata de un algoritmo para la determinación del camino más corto entre el nodo origen y el resto de los nodos conexos en un grafo. El segundo es un algoritmo de búsqueda heurística del camino más corto desde un nodo inicial a un nodo final. La diferencia entre ambos algoritmos es que el primero calcula la ruta de un nodo al resto, y el segundo, la calcula desde un nodo origen a un nodo destino.

A continuación, en el capítulo 2 se exponen los objetivos y requisitos previos propuestos, en el capítulo 3, se describe la infraestructura utilizada en este proyecto. El capítulo 4 y 5 contienen la descripción detallada del proyecto realizado, dividido por las distintas técnicas de navegación implementadas. Finalmente, en el capítulo 6 se detallan las conclusiones y trabajos futuros propuestos.

Capítulo 2

Objetivos

Una vez presentado el contexto de éste proyecto en el capítulo previo, ahora se describirán los objetivos concretos, requisitos y la metodología empleada durante la creación del proyecto de fin de carrera.

El objetivo principal de este proyecto es la creación de prácticas de navegación global para alumnos. Se utilizará para ello la arquitectura de control JdeRobot desarrollada por el grupo de robótica de la URJC que se describirá en detalle en el capítulo 3.

2.1. Objetivos

El objetivo global que persigue el proyecto es la elaboración de prácticas de alumnos, que consisten en la creación de algoritmos de navegación global del robot Kobuki por un entorno conocido sin colisionar con ningún obstáculo. Para ello, se crearan dos componentes académicos que ofrecerán un entorno gráfico de dos dimensiones y toda la infraestructura que se comunica con el simulador Gazebo, dónde el alumno podrá ver los resultados de sus implementaciones. Además, también se ofrecerá una solución posible a los algoritmos que deberán completar los alumnos. Concretamente, se ofrece una posible solución a los algoritmos de navegación global *Gradient Path Planning* y Grafo de Visibilidad, incluyendo el pilotaje del robot utilizando estos algoritmos.

La plataforma de desarrollo consta de dos componentes que se proporcionaran al alumno para la realización de las prácticas de navegación global. Estos componentes están formados por una interfaz gráfica que proporciona un visor del mapa, donde se ejecutarán los algoritmos en dos dimensiones y un conjunto de opciones de visualización y de activación de movimiento del robot. También se facilitarán las conexiones con los sensores y actuadores de Gazebo que sean necesarios dónde el alumno podrá crear nuevas conexiones si lo requiriese. El objetivo de creación de estos componentes académicos, es la abstracción de los alumnos de toda la complejidad ajena al problema, para que puedan abordar directamente un problema de la navegación global.

El interfaz gráfico, además de proporcionarle la funcionalidad básica para la implementación las técnicas, mostrará datos relativos a la posición, velocidad y trazados del robot. Entre las utilidades que ofrecerá el componente podemos destacar la utilización de varios mapas de pruebas que irán anexos al componente, la posibilidad de cambiar de posición al robot o la descripción gráfica del trazado del robot.

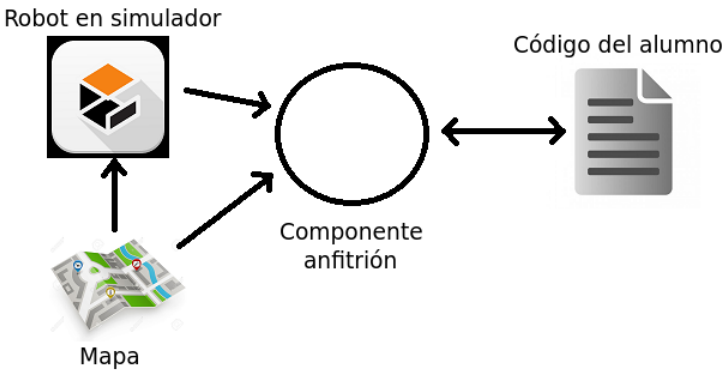

Figura 2.1: Estructura

Estos componentes anfitriones estarán dotados de otras utilidades, que se pondrán a disposición de los alumnos para la que puedan visualizar con mejor detalle los progresos obtenidos con sus prácticas. El código con la solución desarrollada por los alumnos se empotrará en estos componentes académicos.

Para facilitar el desarrollo de las prácticas se crearán los componentes con una arquitectura software para que los alumnos únicamente tengan que desarrollar la construcción, planificación y el pilotaje del robot sobre un único fichero, que proporcionará las estructuras de datos e interfaces ICE que tendrán que modificar. El componente estará preparado para interactuar con este fichero y mostrará las pruebas que realicen los alumnos en su interfaz gráfica. En la figura 2.1, se muestra la estructura que tendrá el componente enfocado a estas prácticas.

Además de estos dos componentes académicos, se desarrollará una posible solución a estas técnicas de navegación global para que sirvan como ejemplo.

2.2. Requisitos

El desarrollo del proyecto está guiado por los objetivos comentados anteriormente y deberá ajustarse a los requisitos de partida, que condicionan la solución de navegación global desarrollada en este proyecto. Estos requisitos son los siguientes:

1. El lenguaje de programación escogido para es desarrollo del proyecto será C++.
2. El software del los componentes interactuaran con la plataforma JdeRobot, que simplifica el acceso a los sensores y actuadores y ofrecerá sus interfaces al fichero que implementarán los alumnos.
3. Todas las simulaciones de este proyecto se realizarán sobre Gazebo.
4. La solución propuesta evitará tiempos de computación demasiado largos y proporcionará una interfaz sencilla al usuario.

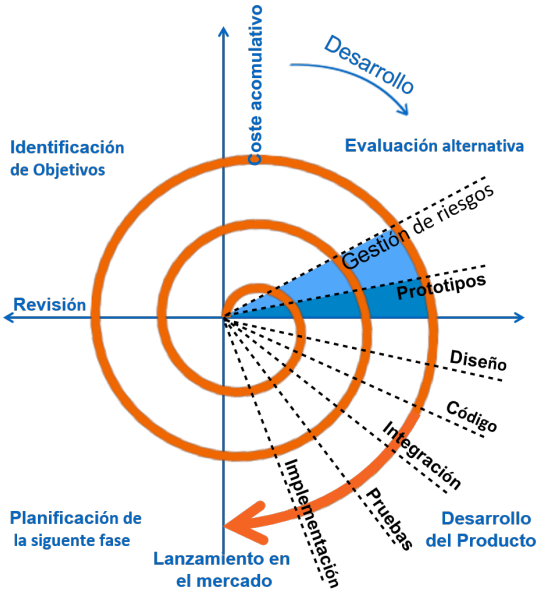

Figura 2.2: Modelo en espiral

2.3. Metodología

En esta sección se describe la metodología utilizada para la realización de este proyecto. Básicamente consiste en realizar iteraciones que se componen de: Recolección de los requisitos que se van a satisfacer, análisis, diseño, implementación, experimentación y planificación de la siguiente fase, así como reuniones periódicas con el tutor.

Para la realización del proyecto se establecieron unas tareas a realizar entre la idea del proyecto hasta la realización de la misma produciéndose el producto final. El desarrollo se ha basado en el modelo en espiral basado en prototipos. He elegido este modelo de desarrollo ya que se basa en separar el comportamiento final en un conjunto de subtareas más sencillas que reúnen la suficiente funcionalidad, como para obtener una versión estable del producto. De esta manera, cada subtarea completada aporta los requisitos e información necesaria para abordar la siguiente iteración y aumenta la funcionalidad del producto hasta la obtención del producto final.

La gran ventaja de este modelo de desarrollo es la existencia de puntos de control al finalizar cada iteración. En la fase de pruebas se comprueba la funcionalidad de la tarea realizada, así como la consistencia de esta en todo el producto. Con esto se asegura que la creación del producto es incremental y estable. Además, es muy flexible en cuanto al cambio de requisitos.

En este tipo de modelo de desarrollo, los productos son creados gracias al número de iteraciones que se da en el proceso de vida del software. En la figura 2.2 se puede observar los ciclos que forman este modelo de desarrollo dónde podemos destacar: identificación de objetivos, evaluación alternativa (Análisis), diseño, código, integración, pruebas y planificación de la siguiente fase.

Al final de cada iteración se produce un prototipo, esto es, una versión preliminar de un sistema con fines de demostración o evaluación de ciertos requisitos. Cada prototipo debe cumplir

los requisitos propuestos en la planificación de cada fase y debe de ser consistente y estable con el resto del producto.

Los prototipos creados a lo largo del proyecto han sido:

- **Prototipo 1:** Creación de los componentes anfitriones que resuelven la interfaz y la comunicación con Gazebo.
- **Prototipo 2:** Lectura de mapas «XML» en el mismo formato que recibe Gazebo.
- **Prototipo 3:** Creación de utilidades: Cambio de posición del robot y pintado del rastro del robot en toda su ejecución.
- **Prototipo 4:** Solución que resuelve el método de *Gradient Path Planning*.
- **Prototipo 5:** Solución que resuelve el método Grafo de Visibilidad.

Estos prototipos han sido las versiones obtenidas del producto, y corresponden con cada una de las iteraciones realizadas en el desarrollo del proyecto. Estas iteraciones han sido planificadas en reuniones periódicas con el tutor del proyecto. Para un mejor seguimiento del proyecto, se ha llevado de forma paralela al desarrollo del software, la elaboración de un blog[1], permanentemente accesible desde Internet, con todos los avances y problemas que han surgido en el desarrollo de cada iteración y se ha utilizado en toda la implementación el controlador de versiones SVN.

[1] http://jderobot.org/Vnieto

Capítulo 3

Plataforma de desarrollo

En este capítulo se explicará la plataforma software sobre la que se ha desarrollado este proyecto, cuya implementación se contemplará en los siguientes capítulos. También se detallará una descripción del robot al que están orientados todos los algoritmos implementados, y se comentarán brevemente algunas librerías y entornos que se han usado.

3.1. Turtlebot

El robot utilizado en mi Proyecto de Fin de Carrera ha sido el llamado "Turtlebot". Es un robot de movilidad que puede navegar por un escenario de forma autónoma o teledirigida. La constitución de este robot se creó con la idea de navegar en espacios reducidos y con terrenos sin grandes desniveles. La estructura del robot está montada sobre una base de Roomba e integra sensores de odometría y una cámara RGB-D, entre otros.

A continuación se describe la arquitectura hardware y software propias del robot y nuestro sistema.En la figura 3.1 podemos ver una imágen de este robot. Dentro del robot «Turtlebot» encontramos estas partes perfectamente diferenciadas:

- **Base Kobuki:** Utiliza dos ruedas motrices independientes que le permiten ejecutar giros de 360 grados. Incluye un sistema diferencial de movimiento con encoders y giroscopio electrónico. A través de los encoders podemos saber en todo momento en qué posición del mapa se encuentra el robot en todo momento.

- **Kinect:** Es un dispositivo, inicialmente pensado como un simple controlador de juego, que gracias a los componentes que lo integran: sensor de profundidad, cámara RGB, array de micrófonos y sensor de infrarrojos (emisor y receptor), es capaz de capturar objetos en un escenario, reconocerlos y posicionarlos en el espacio.

  Gracias a toda la información que captura este dispositivo, se utiliza para desarrollar programas cuyo objetivo principal sea recibir información sensorial de un escenario, procesarla y convertirla en respuestas de un dispositivo.

  Este robot posee un kinect para capturar la información sobre los objetos que se le presentan de frente en su trayectoria. Este dispositivo interviene directamente en la navegación local del robot, ya que es capaz de capturar obstáculos en tiempo real, e informar al programa que calcula la trayectoria del robot para evitarlos.

Figura 3.1: Robot Turtlebot

En este proyecto no se ha hecho uso de el, ya que siempre se trabaja sobre entornos estáticos con un mapa conocido.

- **Computador:** Encargado de ejecutar el software creado y controlar la interacción con los sensores y actuadores del robot.

3.2. Gazebo

Es un simulador de entornos 3D que posibilita evaluar el comportamiento de un robot en un mundo virtual. Es capaz de simular conjuntos de robots, sensores y objetos, en un mundo tridimensional. Además, reproduce la reacción de sensores y la interacción entre dispositivos de una forma muy próxima a la realidad. Permite, entre otras muchas opciones, diseñar robots de forma personalizada, crear mundos virtuales o importar modelos ya creados. Fue creado por [Nathan Koenig and Andrew Howard, 2004].

Posee una simulación realista de la física de cuerpos rígidos, los robots pueden empujar, tocar, recoger objetos, en general, interactúan con el mundo de una manera plausible. Esto se debe, a que todos los elementos tienen masa, velocidad, fricción, y otros atributos que lo hacen más realista.

Gazebo ofrece un entorno muy preparado para realizar pruebas con diferentes robots. La información del entorno es capaz de leerla a través de ficheros con formato «world», los cuales, contienen información de todos los elementos físicos que se muestran en el escenario en forma de modelos. Cada modelo, puede contener varios elementos físicos, uniones entre los mismos, sensores, etc. Además, contiene una API sencilla para añadir todos estos modelos y los enlaces necesarios para la integración con los programas. Estos enlaces, por ejemplo, pueden manejar el movimiento de los motores de un robot o las lecturas de datos sobre cámaras, láseres, etc. En la figura 3.2, se puede ver este simulador.

La versión utilizada de este producto ha sido la versión 5.1.0.

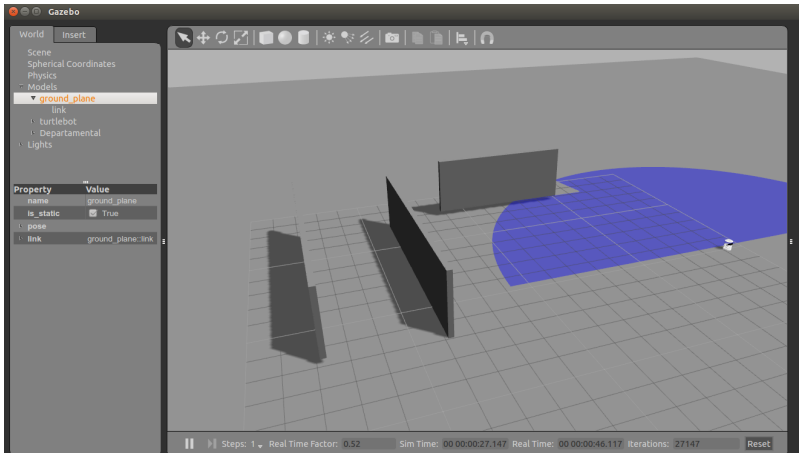

Figura 3.2: Gazebo

3.3. JDERobot

Es una herramienta de software libre, orientada al desarrollo de aplicaciones de robótica, domótica y visión artificial. Se creó en el año 2003, y desde entonces ha ido evolucionando gracias al equipo de robótica de la Universidad Rey Juan Carlos.

Esta herramienta está basada en la creación y utilización de componentes que se comunican entre sí a través de interfaces estandarizadas utilizando la librería ICE, descrita en el apartado siguiente. Con estos componentes se consigue un nivel mayor de abstracción y desde el punto de vista del programador, no tiene que tener en cuenta cómo se realicen las transmisiones de datos entre otros componentes, siempre y cuando se utilicen las mismas interfaces de comunicación. Esto permite el desarrollo de sistemas distribuidos, y proporciona flexibilidad en cuanto a la interacción con otros componentes. Con esto, es posible que dos componentes puedan comunicarse utilizando entornos o lenguajes de programación diferentes.

JdeRobot ofrece un amplio repertorio de drivers con dispositivos hardware que facilitan el envío y recepción de datos de sensores y actuadores, utilizando las mismas interfaces que las usadas entre componentes. Hay drivers para robots reales y para robots simulados en Gazebo. Dentro de esta herramienta, existen componentes creados con distintos lenguajes de programación, como C++, Java o Python.

En este proyecto, se han utilizado dos drivers para la implementación de ambos componentes didácticos. El primero, "Pose3d", es el plugin con el que los componentes se comunican tanto para obtener la posición en tiempo real, como para capacitarlos para poder cambiarle de posición. El segundo es "Motors", es el plugin con el interactúa el componente, consiguiendo la velocidad en cada instante, que modifica los datos de la interfaz de usuario, y establece las modificaciones en las velocidades lineal y angular, una vez se haya aplicado la técnica de navegación correspondiente.

Se ha utilizado la versión 5.3 de JdeRobot.

3.4. ICE

Es un middleware desarrollado por la empresa ZeroC que se encarga de realizar la comunicación entre aplicaciones distribuidas. Sigue el paradigma de la orientación a objetos para realizar llamadas remotas a procedimientos de forma sencilla, como si se tratase de llamadas locales, sin necesidad de usar el protocolo HTTP. Ofrece también otras funciones cómo el balanceo de carga, servicios de publicación y suscripción, etc. Soporta varios lenguajes como C++, C\#, Java, Visual Basic, Python... y funciona sobre la mayoría de sistemas operativos.

Para usar los servicios que ofrece, las aplicaciones se asocian a unos esquemas definidos en el lenguaje 'slice', que sirven para determinar qué interfaces se utilizaran.

En este proyecto se ha empleado esta librería para la comunicación entre el robot y nuestro componente. Las conexiones que se han utilizado han sido sobre los sensores "Pose3d", y sobre los actuadores "Motors". Gracias a estas conexiones, hemos sido capaces de actualizar la posición del robot en nuestro componente y de dotar de movimiento al robot.

3.5. OpenCV

Es una librería de software libre de visión artificial. Provee una infraestructura para aplicaciones de visión artificial. Esta librería alberga una gran cantidad de algoritmos, entre ellos, encontramos algoritmos que permiten identificar objetos, caras, realizar el seguimiento de objetos, extraer modelos 3D, seguir el movimiento de los ojos, reconocer escenarios...

OpenCV ha sido diseñado para ser eficiente en cuanto a gasto de recursos computacionales y con un enfoque hacia las aplicaciones de tiempo real. Está escrito y optimizado en C y puede tomar ventaja de los procesadores con múltiples núcleos. Uno de los objetivos de OpenCV es proveer una infraestructura de visión por computador fácil de utilizar que ayuda a los programadores a desarrollar aplicaciones "sofisticadas" de CV (*Computer Vision*) rápidamente.

Originalmente, fue desarrollada por Intel en 1999 y mantenida por Willow Garage e Itseez. Debido a que se encuentra bajo la licencia BSD, lo que permite utilizar y modificar el código. Debido a su uso generalizado, tiene una amplia documentación, foro propio y una gran comunidad de desarrolladores.

En este proyecto, se ha utilizado esta librería para realizar el pintado en pantalla de la matriz RGB que se obtiene de la construcción del algoritmo Gradient Path Planning sobre el visor del mapa. En concreto, la versión utilizada en este proyecto ha sido la versión 2.3.1.

3.6. QT Creator

Es un Entorno Integrado de Desarrollo (IDE) multiplataforma, creado por Trolltech, que integra múltiples herramientas para facilitar la edición y depuración de código en varios lenguajes de programación, principalmente en C++.

Qt Creator está totalmente integrado con QT Designer, lo que ayuda a diseñar una interfaz de usuario fácil y sencilla. Posee un entorno gráfico compuesto por varios módulos con distintas funcionalidades. En este entorno se puede editar y depurar código, configurar la manera en que

los proyectos son compilados y ejecutados, editor gráfico con varios controles personalizados diseñados para la construcción de la interfaz gráfica. También cuenta con toda la documentación registrada por QT Assistant, tales como la biblioteca Qt y la documentación de Qt Creator.

Este entorno de desarrollo ha sido utilizado para el desarrollo de todo el proyecto de fin de carrera. Con él se ha diseñado la interfaz gráfica en la que el usuario puede interactuar con todo el componente. También se ha utilizado la librería QT Designer para el pintado de líneas, puntos y demás utilidades que se han necesitado en para desarrollar el visor 2D que muestra el robot y el mapa. Por otro lado, se ha utilizado para gestionar los eventos que el usuario generará con el componente para registrar las posiciones dónde desea mover el robot o la posición destino dónde se desea calcular cada una de las técnicas de navegación global.

Ha sido empleado continuamente durante este proyecto por la muchas de sus funcionalidades como el resaltado de sintaxis, auto completado de código, herramientas para la rápida navegación, plegado de código, depurador visual, control estático de código, etc. También se ha hecho uso de la librería QT para el uso del mecanismo de *signals* y *slots* que proporcionan la comunicación entre hilos dentro del programa declarando que método se ejecutará cuando se lanza una señal determinada.

Se ha utilizado la versión 5.2.1.

Capítulo 4

Práctica sobre *Gradient Path Planning*

Sobre la plataforma descrita en el capítulo anterior, se explicará la solución desarrollada para la práctica con el algoritmo Gradient Path Planning, que satisface los objetivos y los requisitos explicados en el capítulo 2 A continuación, se detalla el algoritmo Gradient Path Planning y su implementación realizada en código C++. El código fuente de esta implementación está formado por un conjunto de ficheros que se añaden a los proporcionados por los de la plataforma JDE.

Se presenta primero el algoritmo, segundo el componente académico anfitrión elaborado, tercero la solución desarrollada y por último, unos cuantos experimentos que validan el software desarrollado.

4.1. Gradient Path Planning

Para resolver la navegación global, una de las soluciones es el método de gradiente [Konolige, 2000], que garantiza una trayectoria mínima entre los puntos a viajar. La trayectoria obtenida por este algoritmo es la obtenida entre un punto origen y un punto destino, teniendo en cuenta todos los obstáculos que existen en un mapa.

Este método consiste principalmente en la generación de un frente de onda que recorre el espacio libre del mapa. El origen de esta expansión es el punto destino, y el final de la misma es la posición del robot. La expansión del frente de onda no tiene por qué alcanzar todo el espacio libre del escenario. Cuando el frente de onda alcanza al robot, significa que éste ya tiene una trayectoria mínima entre su posición y el punto destino. Además, en un escenario también pueden existir zonas inaccesibles para la expansión del frente de onda, cómo por ejemplo habitaciones cerradas o sitios en los que por la envergadura del robot, o por razones de seguridad no se permite el acceso.

Si el frente de onda llega a un punto del espacio libre que ya tiene valor, este frente se detiene en este punto, debido a que si este punto ya ha sido visitado, su valor siempre será menor. Si el frente de onda no encuentra al robot en su expansión, se detendrá una vez se haya expandido por todas los puntos del espacio libre accesibles.

Adicionalmente, la cercanía de un punto a un obstáculo hace que este punto tenga un valor inicial mayor que cero, este valor es mayor cuanto más cerca se encuentre el punto del obstáculo. Los obstáculos generan su propio campo de penalización, con un frente de onda propio. Este frente de onda será la protección que se otorgará al robot para que tenga espacio suficiente para maniobrar ante el encuentro de un obstáculo cercano, y así se evitará que colisione con él. Sin la existencia de esta protección de obstáculo el robot colisionaría o rasparía cualquier obstáculo ubicado en el mapa.

Una vez construido el campo, el robot sólo tiene que comprobar los valores de las adyacentes a su posición y dirigirse hacia el punto cuyo valor del campo sea menor. Este algoritmo permite generar una ruta, desde el punto dónde está situado el robot a un punto destino. Además, por construcción, la ruta que sigue el robot es única y de distancia mínima, e incorpora obstáculos a ella. Esta técnica es de navegación global, por lo que todos los obstáculos que el robot será capaz de evitar, tienen que estar descritos en el mapa. Si en tiempo de ejecución, el escenario variara y apareciesen obstáculos que no están contemplados en el mapa, mediante esta técnica el robot sería incapaz de evitarlos.

Resumiendo, esta técnica propaga un campo escalar de valores crecientes desde el objetivo a viajar hasta la posición del robot. Este campo se suma a un campo asociado a los obstáculos, que alejará al robot de estos. Cuando el campo ha terminado de expandirse, se sigue el gradiente de éste para llegar al objetivo. Por construcción, el camino seguido es la ruta óptima.

4.2. Componente anfitrión TurtlebotGPP

El objetivo que aborda este proyecto, es la creación de un componente JdeRobot completo donde los alumnos podrán desarrollar la creación de técnicas de navegación global para robots. En este apartado se detalla el diseño de este componente que podrán utilizar los alumnos para la creación de su algoritmo Gradient Path Planning. A la hora de programar, se hará sobre la plataforma de desarrollo JdeRobot y en lenguaje C++, como se ha comentado en el capítulo 3.

El diseño de esta práctica se divide en tres partes perfectamente diferenciadas: el robot real o simulado por Gazebo, el componente de control y visualización TurtlebotGPP y el fichero "MyAlgorithms". Por un lado, el componente se conecta mediante interfaces ICE a los sensores y los actuadores del robot. Esta información nos la sirve el robot real Turtlebot o el simulador de Gazebo. Esta conexión es necesaria para que el componente pueda obtener la información de la posición en la que se ubica en todo momento y para enviar la velocidad lineal y angular que precise.

Por otro lado, el componente TurtlebotGPP será el encargado de interactuar con el robot y con el usuario que utiliza el componente. Cuando el componente se inicia, necesita un mapa dónde recibe toda la información acerca de los obstáculos que tiene a su alrededor y lo pinta en el visor. Otra función que realiza el componente es la conexión con el interfaz "Pose3d", el cual, proporciona la posición del robot.

Este componente posee dos hilos de ejecución: uno de visualización, que actualiza la interfaz de usuario, y otro de control, que actualiza la comunicación con los sensores y actuadores. El intervalo de tiempo de actualización de estos hilos es muy importante. En cuanto a la interfaz gráfica, es necesario que el intervalo sea pequeño, para poder garantizar al usuario que la posición del robot que está visualizando en el visor es en tiempo real. En cuanto al hilo de control,

esta velocidad de refresco también es muy importante, ya que este componente, se encarga de establecer la velocidad y la dirección del robot en todo momento. Si este tiempo de actualización fuese mayor, las decisiones adoptadas por el componente que modifican su trayectoria podrían no ser las correctas. A medida que el robot avanza, recalcula mediante el algoritmo GPP las siguientes instrucciones que tiene que tomar para llegar a su posición destino. Esto quiere decir, que el robot recorre una distancia determinada desde que recibe la información, hasta que ejecuta la corrección de su trayectoria, por lo que si el tiempo de refresco es muy grande, no será capaz de modificar su trayectoria a tiempo para evitar un obstáculo.

El hilo que actualiza la interfaz y la propia interfaz se comunican a través de señales (signals y slots). El hilo de actualización modifica toda la información que obtiene del robot en la API, que es la memoria que comparten todas las clases del componente. Cada vez que este hilo envía una señal a la clase que contiene la interfaz, ésta lee de la API los parámetros y actualiza los valores de la interfaz, y ésta muestra los datos obtenidos en forma de gráfica o en forma de texto. En forma de gráfica se visualiza la posición del robot. En forma de texto la interfaz muestra varios datos: Posición del robot y orientación obtenidos directamente del interfaz "Pose3d" y la velocidad lineal y angular que tiene en cada momento.

El componente cuenta con un fichero de configuración, en el que se especificarán las conexiones dónde los sensores y actuadores del robot están sirviendo los datos y la ruta del mapa que se desea cargar. Cuando el componente arranca crea la interfaces ICE con estos datos. Por otro lado, intentará abrir la ruta del fichero del mapa especificada, y cargará el mapa en el visor con todos los obstáculos pintados,tal y como se muestra en la figura 4.1.

El formato de los ficheros de mapas soportados son de dos tipos:

- Formato «world»: Este tipo es el mismo que utiliza el simulador Gazebo para la lectura de sus escenarios. Este fichero contiene entre otros, un conjunto de los elementos geométricos que describen el escenario, entre ellos están los posibles obstáculos y el robot. El sistema de carga de líneas geométricas se basa en la construcción de una línea pintada en el visor a escala dado el punto central y la longitud de la misma. Se recorren cada una de las líneas que alberga este fichero y se pintan en la interfaz del componente.
- Formato imagen: Los formatos de imagen compatible con la aplicación serán «JPG» y «BMP». El sistema de lectura de estos ficheros está basado en la lectura de cada uno de los píxeles de la imagen, y la comprobación del valor RGB que tiene cada uno de estos. Todos los píxeles cuyo valor RGB represente al color blanco, serán interpretados como espacio libre, el resto, como obstáculos.

Por lo tanto, cuando el componente es iniciado, crea las interfaces ICE necesarias para la interacción con el robot e interpreta el mapa descrito. La estructura del mapa inicial creada, almacena todo el contenido del mapa leído en un matriz de valores decimales. El estado inicial de todas las celdas de esta rejilla será espacio libre, salvo los obstáculos y sus alrededores, que contendrán los valores más altos para su posterior identificación. Los obstáculos emiten un frente de onda cuya expansión máxima está limitada por una constante. Cada nivel del frente de onda recorrido contendrá el valor del píxel anterior menos uno, por lo que, a medida que el frente de onda se aleja de un obstáculo el valor del campo será menor. Esta protección tiene la función de evitar que el robot colisione con cualquier obstáculo debido a que, por construcción de la técnica GPP, el robot siempre deberá moverse hacia valores más pequeños y tanto los obstáculos como sus protecciones, siempre contendrán los valores más altos del escenario.

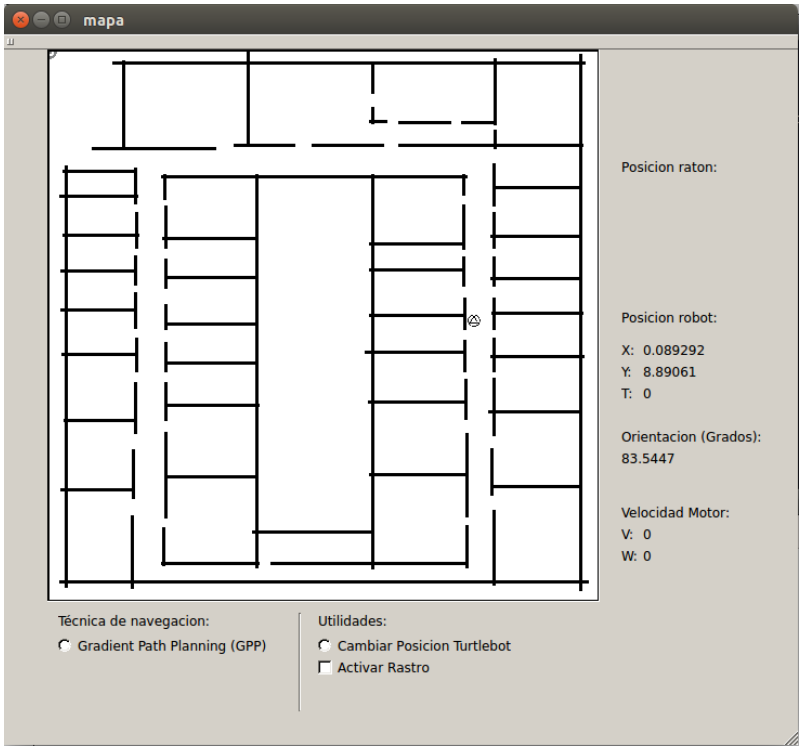

Figura 4.1: Mapa inicial

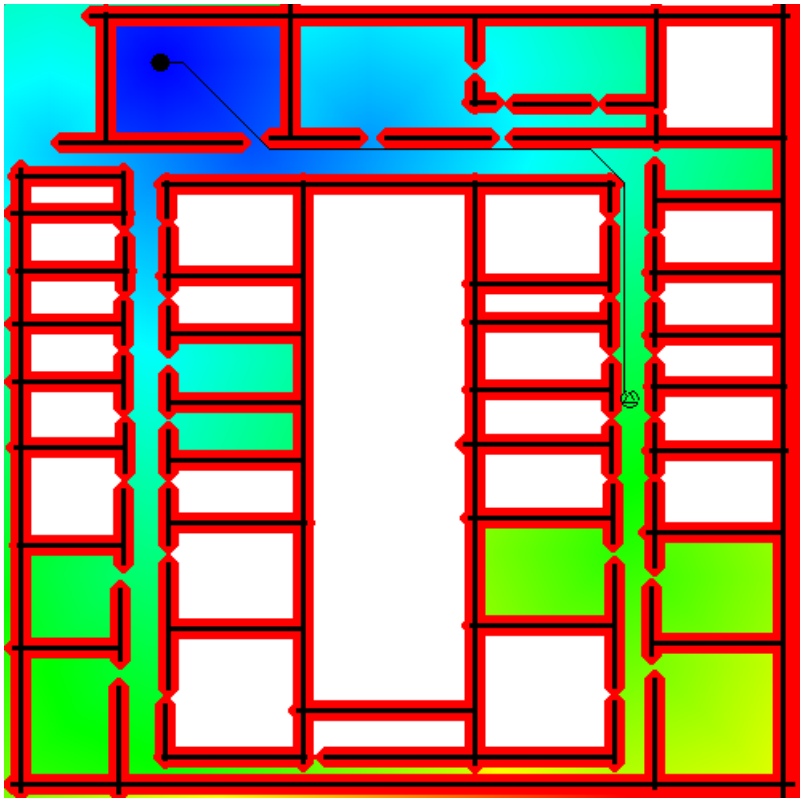

Figura 4.2: Planificación de ruta GPP

Cuando se selecciona en la interfaz que se calcule el mapa de gradiente, la interfaz ofrece dos opciones adicionales que son calcular una planificación de la ruta óptima y activar la navegación. La primera calcula la trayectoria óptima que debería seguir el robot para alcanzar el punto marcado como destino. Para ello utiliza la posición en la que se encuentra el robot y el mapa de gradiente. Por definición de la construcción de este algoritmo, cuanto menor sea el valor de una celda, más cerca nos encontraremos del destino. Por lo tanto, el cálculo de la trayectoria ideal escoge la trayectoria basándose en este principio, eligiendo entre las celdas adyacentes al robot, la de menor valor y repitiendo este paso hasta encontrar la posición destino. En cada una de las iteraciones que se realizan se almacena en una lista una línea formada por la posición que está observando el algoritmo en ese momento y la adyacente de menor valor. Este cálculo se realiza una única vez por cada posición final elegida. Los alumnos podrán mostrar u ocultar el camino según su conveniencia. En la figura 4.2, podemos ver un ejemplo del pintado de la planificación de la trayectoria en un escenario.

Tanto la opción activación de la construcción del mapa de gradiente como la opción de activación de la navegación del robot, son las opciones cuyo comportamiento tendrá que implementar el alumno en el fichero "MyAlgorithms". La creación del algoritmo que calcula el mapa de gradiente puede realizarse y probarse en el componente de manera independiente.

Por otro lado la interfaz ofrece dos utilidades, que ofrecen la posibilidad de cambiar de posición al robot y de mostrar u ocultar el rastro del robot. Para la ejecución de la primera utilidad citada,

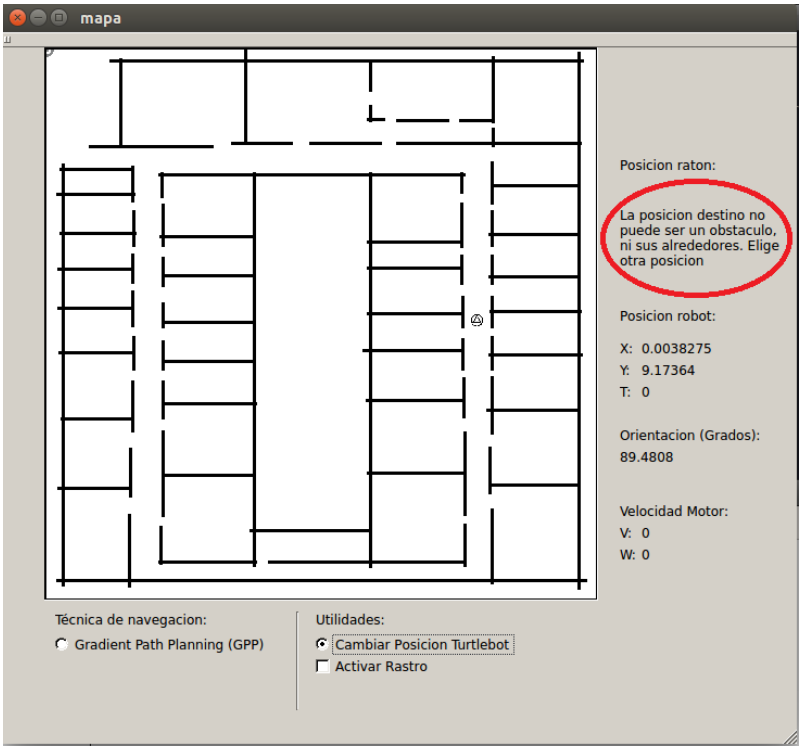

Figura 4.3: Utilidad: Cambio de posición

será necesario seleccionar dentro del visor, una posición con el ratón dentro del espacio libre de transito del robot. Si el usuario selecciona una posición cuyo valor describe la posición de un obstáculo o de un protección, el robot no cambiará de posición y se notificará por medio de un texto tal y como se muestra en la figura 4.3.

La segunda utilidad citada que ofrece la interfaz es la encargada de dibujar el trazado del robot en toda su navegación. Gracias a la planificación de la ruta GPP descrita previamente, podemos comprobar cuál es la ruta ideal que debería llevar el robot para alcanzar la posición destino. Pero en la realidad, hay varios factores en lo que refiere al pilotaje del robot, que dificultan el trazado óptimo de la ruta planificada. Estos son principalmente, la inercia que acumula y el tiempo transcurrido entre el análisis de la trayectoria a seguir en un instante y la ejecución de las correcciones realizadas en la misma. Por ello, es de gran utilidad conocer la trayectoria que realmente ha seguido el robot para alcanzar la posición destino. En la figura 4.4, se muestra un fragmento del trazado real del robot respecto a la línea que describe la trayectoria ideal que debería llevar. Esta figura se ha tomado utilizando una simulación de mi solución propuesta a esta técnica, que se explicará en la siguiente sección.

Uno de los objetivos que tendrán las prácticas sobre este componente es la construcción de un mapa de gradiente. La implementación del algoritmo capaz de resolver esto se realizará sobre el fichero “MyAlgorithms”. Los datos de entrada que el componente servirá a este fichero serán una matriz de valores de tipo “Double”, que se inicializará automáticamente, como se ha comentado, cuando el componente arranque, y un el punto destino desde el que deberá de

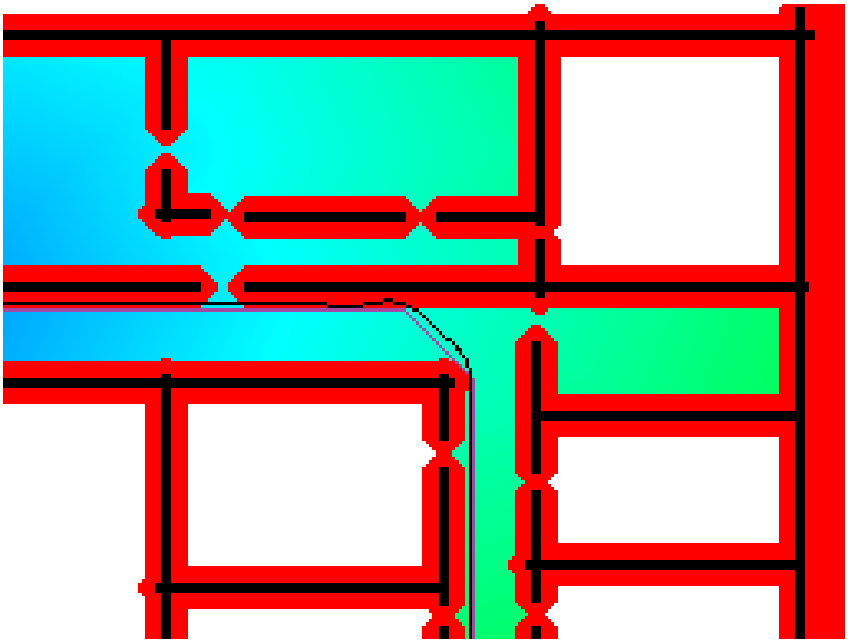

Figura 4.4: Utilidad: Rastro del robot

comenzar la expansión del frente de onda. Este método tendrá que devolver una matriz de las mismas propiedades que la de entrada, que el componente académico anfitrión copiará en su interfaz.

La forma en que se pinta la matriz de gradiente simula un mapa de temperatura, siendo los colores más fríos, los más cercanos a la posición destino, y a medida que nos vayamos alejando del mismo, serán más cálidos. El componente cuenta con un método capaz de leer la matriz que alberga el campo de cada celda del mapa, y transformar cada valor a un color en formato RGB. Por cada valor de esta matriz, en primer lugar comprobará si se trata de uno de los dos límites, que serán representados por el color blanco, para el espacio libre, y con el color negro, para los obstáculos. Con el resto de valores, se calculará el color que le corresponde por su valor, y discriminándolo por bloques de colores, siendo estos de color frío a cálido: azul, verde, amarillo y rojo. El código de la función es el siguiente:

```
int* mapa::convertirDoubleARGB(double number) {
    int *colorRGB = new int[3];
    if (number == -1) {
        colorRGB[0] = 255;
        colorRGB[1] = 255;
        colorRGB[2] = 255;
        return colorRGB;
    }
    if (number == MaxPesoCelda + DistanciaSeguridadParedes +1) {
        colorRGB[0] = 0;
        colorRGB[1] = 0;
        colorRGB[2] = 0;
```

```
        return colorRGB;
    }
    int roundNumber= round(number);
    int numeroDeBloque = roundNumber / 255;
    int numeroTonalidad = roundNumber %255;
    switch (numeroDeBloque)
    {
        case 0:
            colorRGB[0] = 0;
            colorRGB[1] = numeroTonalidad;
            colorRGB[2] = 255;
            break;
        case 1:
            colorRGB[0] = 0;
            colorRGB[1] = 255;
            colorRGB[2] = 255-numeroTonalidad;
            break;
        case 2:
            colorRGB[0] = numeroTonalidad;
            colorRGB[1] = 255;
            colorRGB[2] = 0;
            break;
        case 3:
            colorRGB[0] = 255;
            colorRGB[1] = 255-numeroTonalidad;
            colorRGB[2] = 0;
            break;
        default:
            colorRGB[0] = 255;
            colorRGB[1] = 0;
            colorRGB[2] = 0;
            break;
    }
    return colorRGB;
}
```

4.3. Solución implementada del algoritmo GPP

La implementación de la técnica de navegación global *Gradient Path Planning* puede llevarse a cabo de muchas formas diferentes. A continuación se describirá la solución creada, que se aporta a los alumnos para que sirva como ejemplo. La solución propuesta está contenida en el fichero "MyAlgorithms" y consta de dos métodos correspondientes a la construcción del mapa de gradiente y al pilotaje sobre el mapa construido.

4.3.1. Construcción del mapa de gradiente

La construcción del mapa de gradiente está basada en la propagación de un frente de onda que se expande por todo el espacio libre del escenario, deteniéndose cuando encuentra un obstáculo.

El origen de la expansión es siempre el punto destino al que queremos que se desplace el robot. Este punto toma el valor 0. El valor de campo que recibirá cada posición de la matriz incrementará a medida que se aleja de este punto, provocando que el robot pueda dirigirse siempre hacia la posición destino navegando hacia las posiciones con menor valor de campo. La expansión del frente de onda debería finalizar en el instante en que este alcanza al robot, debido a que si se da esta casuística, se habrá encontrado un camino óptimo que puede recorrer el robot para llegar a su destino. Si el frente de onda nunca alcanza al robot, el frente de onda morirá cuando haya recorrido todo el espacio libre accesible del escenario propuesto. Si se da este último caso, significará que el destino escogido por el usuario es un punto del mapa inaccesible para el robot. En la solución propuesta, la expansión del frente de onda continua siempre hasta haber recorrido todo el espacio libre accesible, sin detenerse al encontrar al robot, con el objetivo de que el alumno pueda comprobar el valor de gradiente en todo el escenario.

La expansión del frente de onda siempre se realiza por niveles, por lo tanto, el recorrido de la lista de nodos, siempre se realizará en anchura y nunca en profundidad. Esto es así, dado que si un frente de onda que se propaga por distintos caminos, tratan de modificar el valor de la misma celda de la rejilla, necesitamos saber cuál es el que ha llegado antes a dicha celda, ya que siempre se almacenará el primer valor que se reciba. Con esto podremos asegurar que el mapa de gradiente construido será óptimo.

Las estructuras de datos utilizadas para la construcción del mapa de gradiente, han sido un mapa de control, dónde se establecen los valores de campo y una lista FIFO de nodos pendientes de expandir. El comportamiento de este método consiste en extraer un nodo de la lista, comprobar si ya ha sido visitado, y en caso de no haberlo sido, establecer el valor de campo que le corresponde e introducir los nodos adyacentes a la lista. La expansión finalizará cuando la lista quede vacía. En la primera iteración se introduce el nodo que alberga la posición destino en la lista, ya que será el primer nodo que haya que expandir. Para evitar que en diferentes iteraciones se visiten varias veces la misma celda de la matriz, cada vez que se accede a un nodo, se comprueba si ya ha sido visitado, de ser así, el valor que hay depositado en él no se modifica y tampoco se expanden los de sus celdas adyacentes.

El primer problema que nos encontramos en cuanto al diseño de la solución es que la forma en la que se deberían de propagar la expansión tendría que tener forma circular. Como primera aproximación al algoritmo de construcción, realicé una expansión en la que cada celda adyacente tomaba el valor de campo correspondiente a la actual más uno. Por tanto, la forma que tenía la expansión era cuadrada. El problema más importante que desembocaba esto era la priorización de las trayectorias verticales y horizontales, de tal manera, que el robot tomaba los giros en la mayoría de las ocasiones en forma perpendicular a su trayectoria. Esto era debido a que el valor de campo obtenido en las celdas adyacentes diagonales no tenían el valor adecuado.

Como podemos comprobar en la figura 4.5, si se define una circunferencia alrededor de un punto y se coloca una rejilla representando un fragmento del escenario ubicado en la posición destino, se puede comprobar que la circunferencia nunca recorre los centros de todas las celdas adyacentes. La circunferencia descrita sólo atraviesa los centros de las celdas adyacentes horizontales y verticales, pero nunca las diagonales. Por lo tanto, el valor de campo que se emite a las celdas adyacentes es diferente dependiendo de la posición en la que está colocada cada celda. En las celdas adyacentes verticales y horizontales se sumará 1 al valor de campo mientras que en las diagonales será raíz de 2.

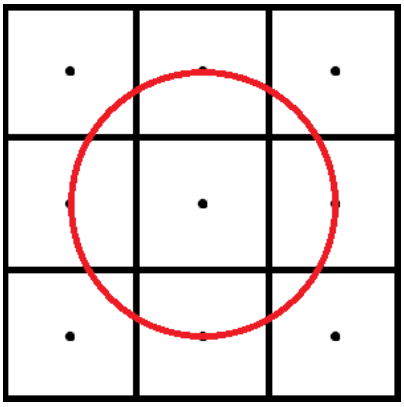

Figura 4.5: Celdas adyacentes

Este error no sólo repercutía a la hora de que el robot interpretase los giros, sino que también implicaba que la construcción del mapa de gradiente tuviese formas cuadradas. De esta manera, cada vez que el robot tuviese que entrar por un espacio reducido, como una entrada, la única manera que tenía de hacerlo era de forma completamente perpendicular a la misma.

En la construcción final, a las celdas adyacentes diagonales se les establece el valor de raíz de dos que es la distancia al centro de estas celdas. De esta manera se obtiene una aproximación más realista de la expansión del frente de onda en forma circular. Citando el ejemplo anterior en el que el robot accede a un espacio libre a través de una entrada, en este caso el robot tendrá una trayectoria arqueada para entrar, sin necesidad de colocarse en posición perpendicular a la entrada. En la figura 4.6, se muestra la propagación del frente de onda mediante las distintas implementaciones comentadas, sobre un mapa que contiene únicamente un obstáculo. A la izquierda la implementación de la primera aproximación, a la derecha, la implementación final de la expansión del frente de onda.

4.3.2. Pilotaje del robot

El pilotaje del robot que interpreta un mapa de gradiente generado está basado en la navegación hacia posiciones con valores inferiores del mismo. El pilotaje del robot se puede categorizar en tres partes que son la lectura e interpretación del mapa, el cálculo de la nueva trayectoria a seguir y la transformación de la misma en forma de velocidades que se emiten al motor del robot.

El componente anfitrión posee un hilo de control que además de actualizar la posición del robot, comprueba si se ha activado en la interfaz la opción de navegación. Si se ha activado, el componente llama a la clase «MyAlgorithms» y ejecuta el método que calcula continuamente las velocidades de los motores.

La dirección final hacia la que se tiene que dirigir el robot para completar la trayectoria, se calcula comprobando las celdas adyacentes a la posición del robot y tomando como dirección el vector que une el centro del robot con la celda adyacente de menor valor. El siguiente paso que realiza el algoritmo es el cálculo de la diferencia angular entre la orientación que tiene el robot en ese preciso momento y la dirección escogida. Con esta diferencia como dato de entrada, se ha

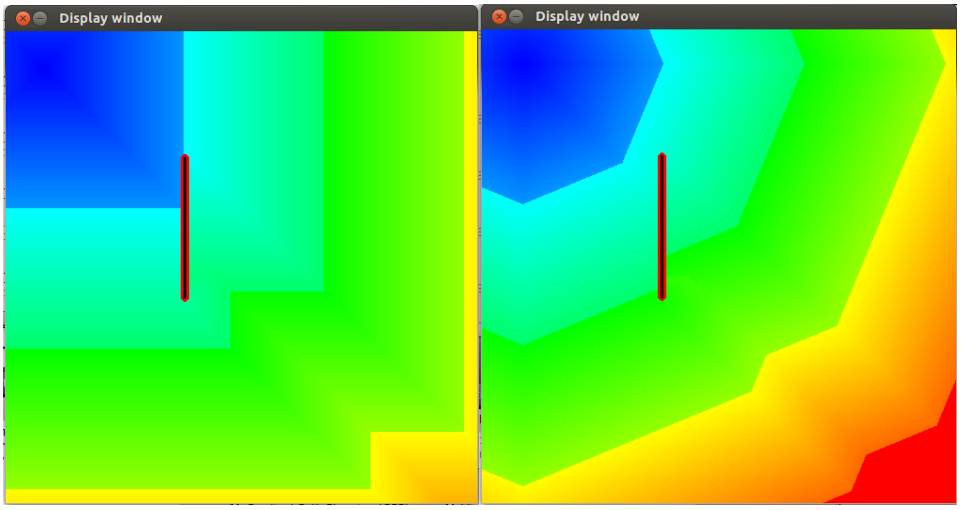

Figura 4.6: Implementaciones de la expansión del frente de onda

definido un sistema de control basado en casos, es decir, dependiendo de la diferencia angular calculada, se emitirán una velocidad linear y angular correspondientes a cada caso.

El pilotaje del robot cuenta principalmente con dos problemas que dificultan la navegación, estos son la inercia y la oscilación. La inercia se puede evitar ajustando la distancia que el robot recorre desde que comienza el procesamiento de las correcciones efectuadas en su trayectoria, hasta que las ejecuta.

Como primera aproximación de ejecución del pilotaje, el robot tomaba como trayectoria, la celda con menor valor dentro de su rango de inmediatamente adyacentes. Cuando el robot realizaba las correcciones para producir un cambio en su movimiento para ajustarse a la trayectoria idónea, éste sobrepasaba esta trayectoria, produciéndose las correcciones necesarias en el sentido contrario. Esto producía una oscilación perfecta como se puede apreciar en la figura 4.7.

Para solucionar esta oscilación se han realizado dos correcciones. En primer lugar el cálculo de la trayectoria no se produce sobre las celdas inmediatamente adyacentes. Se realiza una simulación de dónde debería estar el robot de manera idónea en un futuro próximo, determinado por una constante. Esto repercute directamente en las correcciones que tiene que realizar proponiendo siempre como subobjetivo un punto adelantado dentro de su trayectoria idónea. El subobjetivo que se fija puede ser actualizado y pintado mediante las variables «posObjetivoX» y «posObjetivoY» ubicadas en el fichero «API», utilizado como memoria compartida en todo el componente. Gracias al pintado del subobjetivo se puede comprobar en el componente anfitrión, hacia donde se dirige el robot en todo momento.

La otra medida que se ha aportado al control del robot es un controlador proporcional con correciones. Es un mecanismo de control por realimentación que calcula el error de la trayectoria, que es la desviación entre un valor medido y uno deseado. Este se adapta perfectamente al problema del pilotaje del robot, ya que el valor medido corresponde a la orientación que tiene el robot en un instante, y el valor deseado corresponde a la trayectoria que debe llevar en el mismo instante. La parte proporcional consiste en el producto entre la señal de error y la

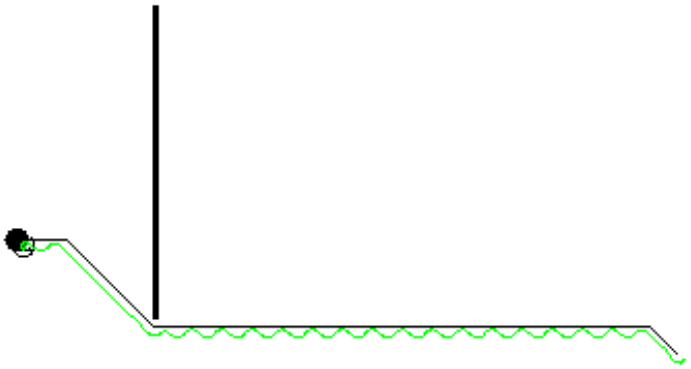

Figura 4.7: Oscilaciones GPP

constante proporcional obtenida. Con el objetivo de eliminar la oscilación que produce la parte proporcional, también se aplican correciones, que tienen como propósito disminuir y eliminar el error provocado. También tienen la función de promediar el error, para ello, se aplica un incremento a lo que dicta el controlador proporcional, consiguiendo disminuir el error, y por tanto, la oscilación.

Para agilizar el pilotaje del robot se ha incluido un sistema de aceleración que está basado en la incrementación gradual de la velocidad lineal, en proporción al número de veces que se encuentra en el mejor caso de control. Este caso consiste en que la orientación del robot está dentro del rango angular definido por la orientación de la trayectoria ideal más menos 5 grados.

Además en cada iteración se comprueba si el robot ha llegado a su destino, con el objetivo de enviarle las instrucciones de parada para que se estacione en su posición destino.

4.4. Experimentación

En este apartado se explican varias experimentaciones representativas del robot *Turtlebot* en distintos escenarios de pruebas. Para realizar una comparativa dentro de estas ejecuciones, se medirán los tiempos de los recorridos que trazará el robot hacia el mismo punto destino, con estrategias de navegación con diferente factor de riesgo.

El sistema de control basado en casos siempre obtendrá la misma proporción de velocidades, estas se ponderarán por medio varios factores. Los factores que se variarán en las ejecuciones con el objetivo de conseguir navegaciones más ágiles serán la modificación de la velocidad lineal y angular y la modificación de los parámetros relativos a la aceleración.

4.4.1. Mapa edificio departamental

En primer lugar se ha lanzado una ejecución con la navegación original de mi solución. Esta ejecución se ha comparado con dos ejecuciones posteriores en las que se ha variado la velocidad

lineal y angular. En la figura 4.8, podemos ver el mapa original y la posición del robot y destino de partida que serán siempre las mismas para poder hacer una comparativa justa.

A continuación, se muestra una tabla con los tiempos de ejecución obtenidos:

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
Velocidad Lineal	1x	1x	1.2x	1x	1x
Velocidad Angular	1x	2x	2x	2x	2x
Aceleración máxima	64	64	64	64	84
Incremento aceleración	1x	1x	1x	3x	3x
Tiempo de ejecución	3:10	2:35	2:19	2:08	1:57

Como se puede observar en la segunda ejecución, al modificar la velocidad angular se consigue que el robot se ajuste más a la trayectoria y se consiguen mejores resultados. En la tercera ejecución, se ha aumentado un 20 % la velocidad lineal en todos los casos del sistema de control. En este sistema, si se trata de aumentar aún más la velocidad, la navegación del robot puede producir oscilaciones o colisiones con objetos, ya que no cuenta con el tiempo ni el espacio suficientes para maniobrar y evitarlos.

En la figura 4.9, a la izquierda, se muestra la ejecución original y a la derecha, la navegación las alteraciones descritas previamente.

Como aumentar la velocidad lineal implica arriesgar la integridad del robot, se va a modificar esta velocidad cuando la trayectoria es estable. Por lo tanto, se puede observar cómo con un aumento en la aceleración del robot podemos mejorar los tiempos de navegación. Dentro de la aceleración, se modificarán dos factores relativos a la misma, el incremento de velocidad en cada instante, y el incremento de la velocidad máxima que se limita al robot.

En la figura 4.10, podemos comprobar la trayectoria que ha seguido el robot en la prueba 5.

4.4.2. Mapa aeropuerto

En este caso, el escenario simulado es un aeropuerto, este mapa es cargado directamente de un fichero en formato RGB. En la figura 4.11, se puede observar el estado inicial del robot en este mapa y la trayectoria que deberá seguir en todas sus navegaciones.

La siguiente tabla muestra los resultados de todas las pruebas realizadas sobre este escenario. Como la trayectoria seguida, en su mayoría es recta, la diferencia del tiempo entre el ejemplo base y los ejemplo con la mejora en la aceleración han sido los más beneficiados en la navegación del robot.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
Velocidad Lineal	1x	1x	1.2x	1x	1x
Velocidad Angular	2x	2x	2x	2x	2x
Aceleración máxima	64	64	64	64	84
Incremento aceleración	1x	1x	1x	3x	3x
Tiempo de ejecución	2:48	2:42	2:34	2:17	2:08

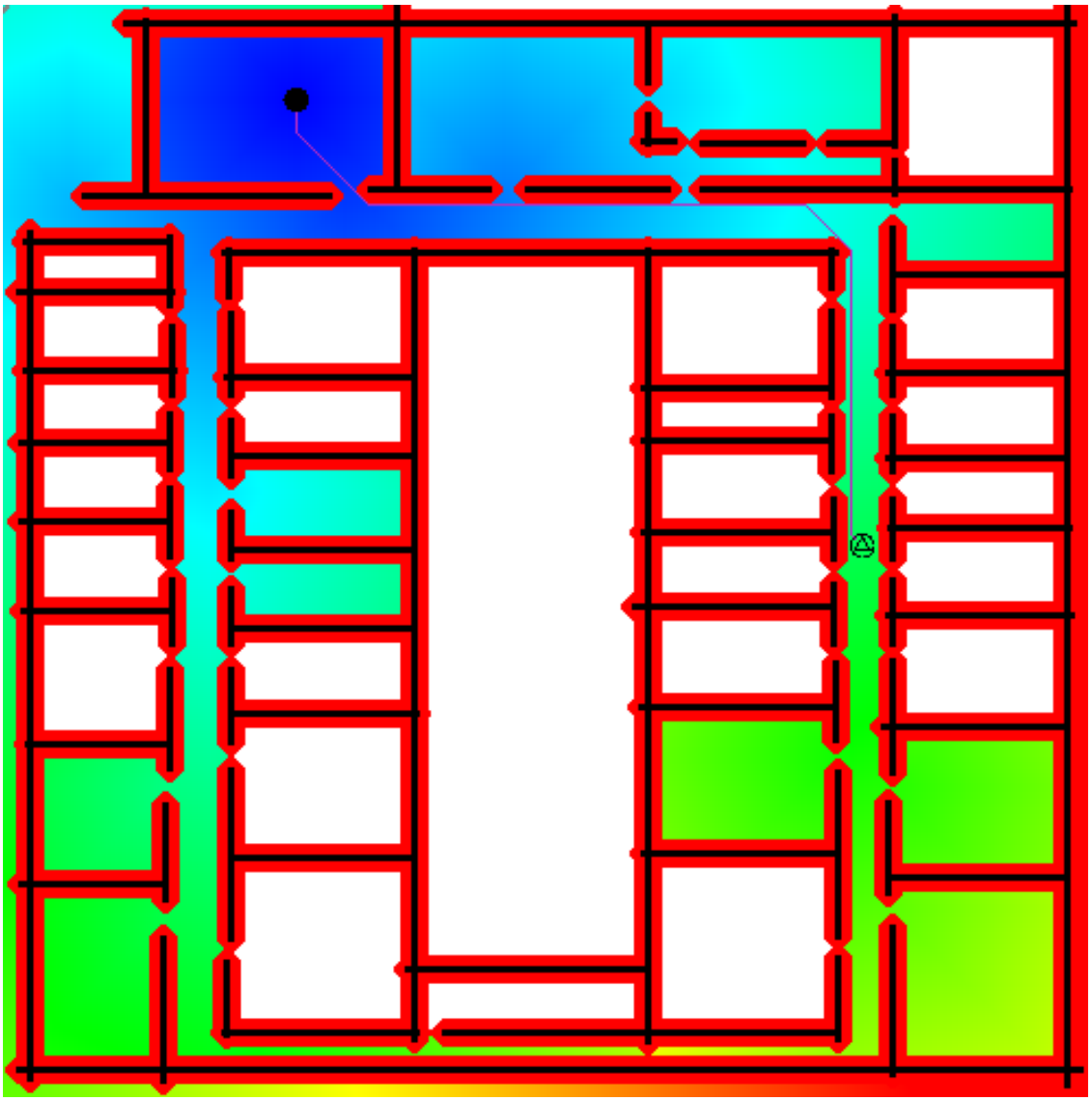

Figura 4.8: Mapa del departamental 1

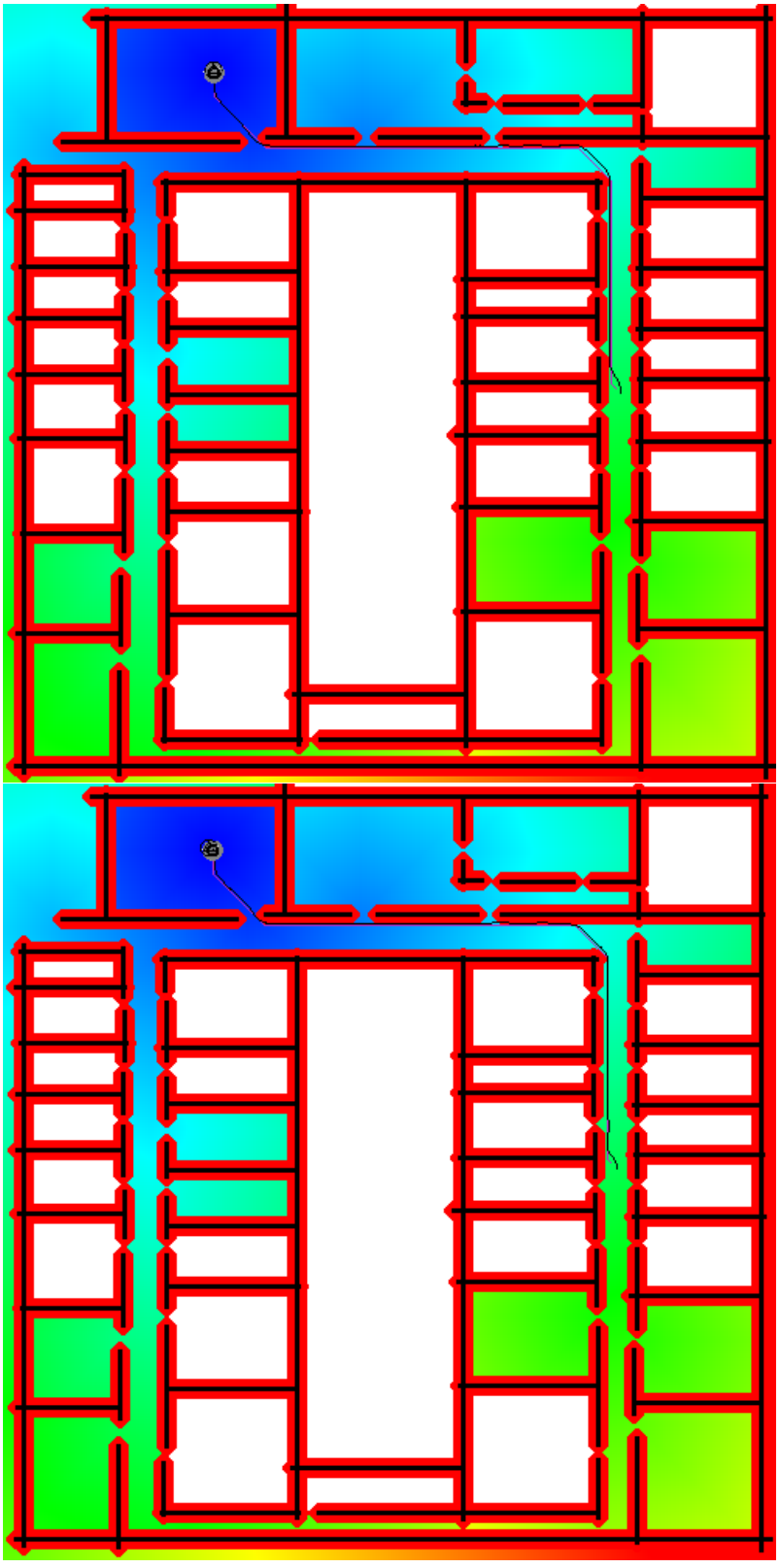

Figura 4.9: Mapa del departamental 2

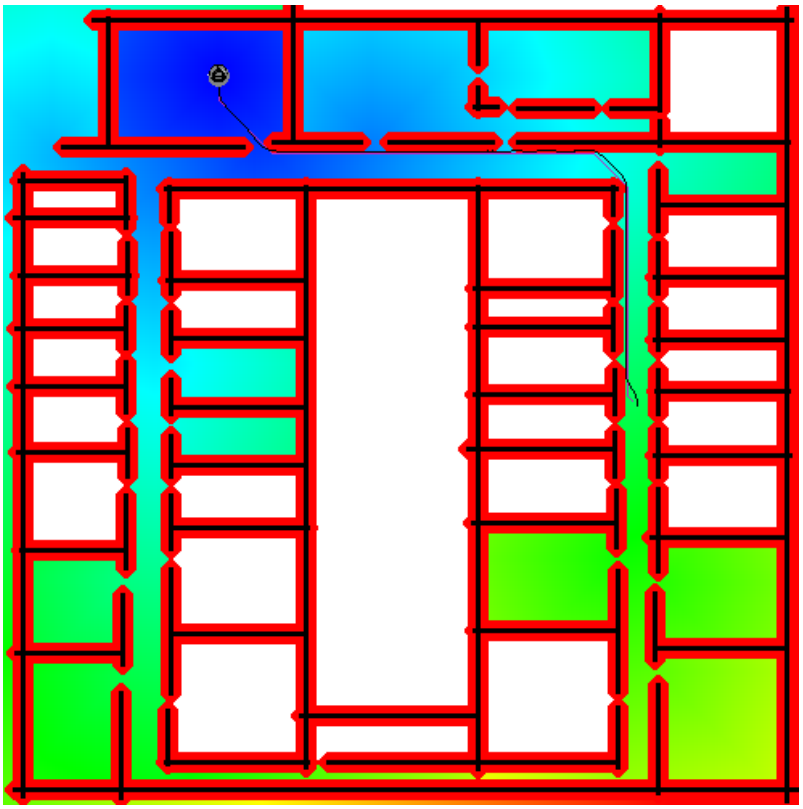

Figura 4.10: Mapa del departamental 3

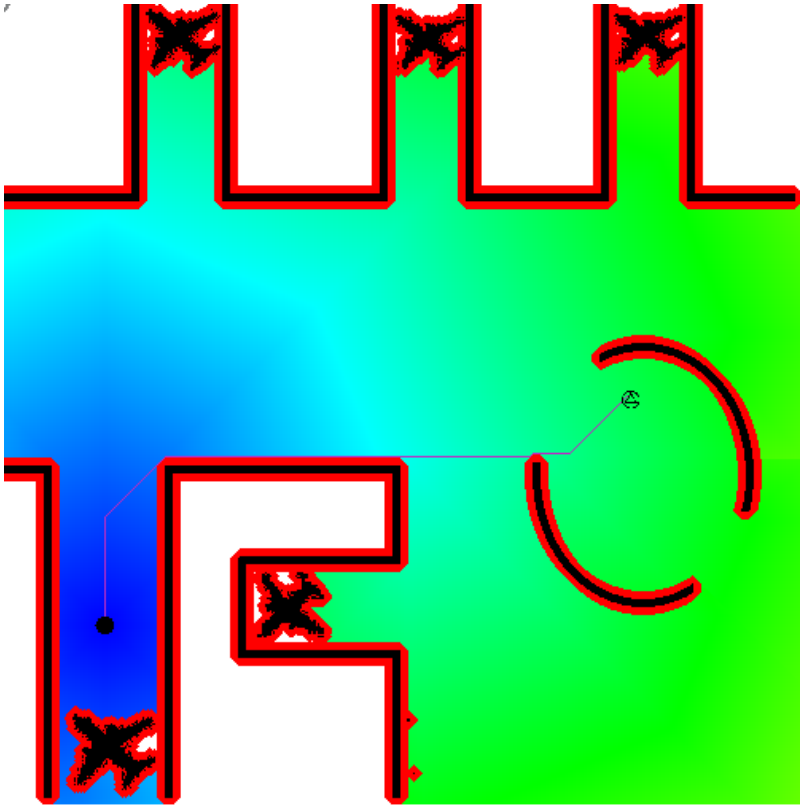

Figura 4.11: Mapa del aeropuerto 1

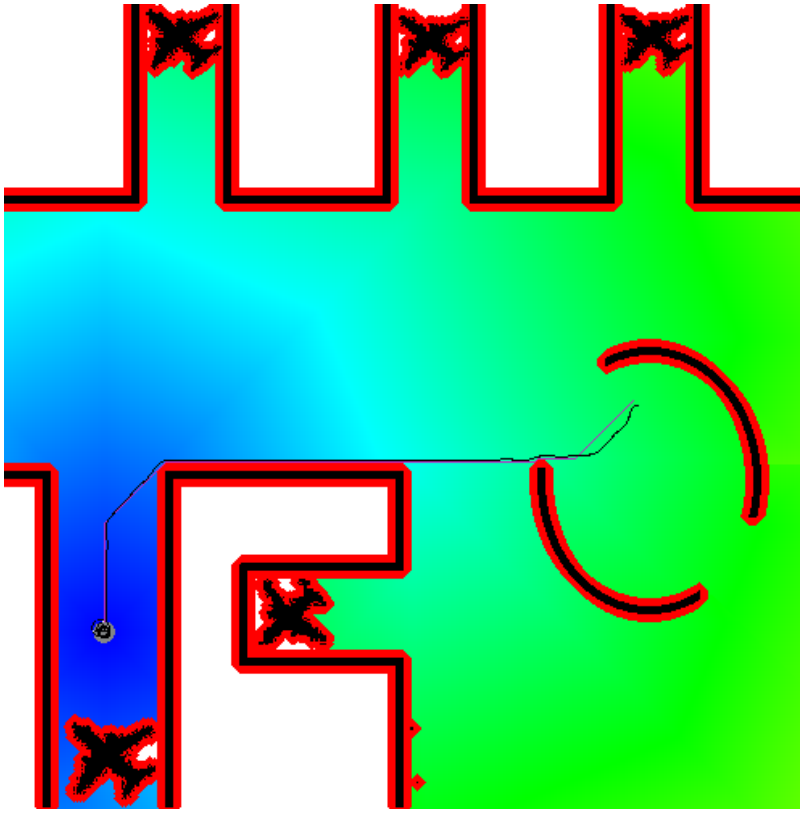

Figura 4.12: Mapa del aeropuerto 2

En este escenario el robot no tiene que realizar demasiados giros, por lo que el cambio en la velocidad angular no supone una diferencia de tiempo demasiado grande. En la figura 4.12, correspondiente al tercer caso de prueba, se puede observar cómo al salir de la primera entrada, el robot comienza a oscilar y gracias al controlador PID, se recupera y continúa con una trayectoria recta.

La figura 4.13, corresponde a la ejecución del quinto ejemplo. En su trayectoria podemos ver pequeños indicios de oscilaciones, a pesar de ello, el tiempo de ejecución es el mejor obtenido ya que al acelerar más rápido y poseer mayor velocidad máxima, subsana el tiempo perdido en las pequeñas oscilaciones de su trayectoria.

4.4.3. Análisis computacional

Se ha realizado un breve análisis de coste computacional del algoritmo *Gradient Path Planning* en cada una de las partes de su construcción. En la siguiente tabla se muestran por un lado, el tiempo que necesita el computador para ejecutar la construcción del mapa de gradiente, y por otro lado, el tiempo que tarda en ejecutar cada iteración que calcula las velocidades que modifican la trayectoria del robot.

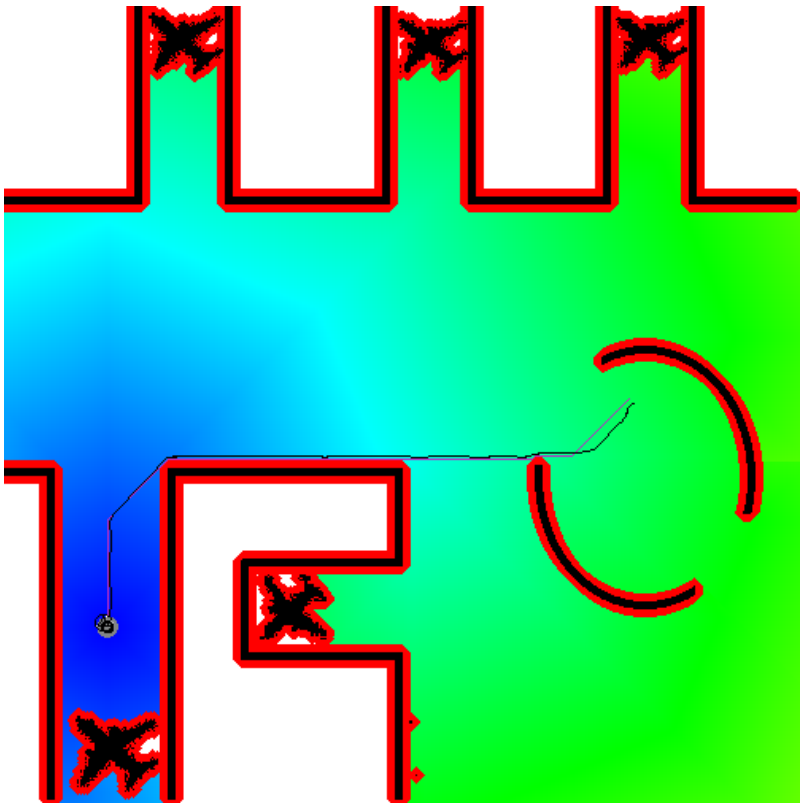

Figura 4.13: Mapa del aeropuerto 3

	Departamental	Aeropuerto
Construcción mapa de gradiente	153 ms	223 ms
Ejecución iteración de control	0.235 ms	0.238 ms

Los costes en tiempo que tarda el algoritmo en crear el mapa de gradiente está directamente influenciado por la cantidad de obstáculos que tiene cada escenario. Además, ambos datos pueden variar dependiendo del equipo donde se esté ejecutando y la carga presente en el mismo en ese momento.

En cuanto a la construcción del mapa de gradiente estos datos nos revelan que cuanto mayor es el espacio libre por el que puede transitar el robot, mayores son los tiempos de cálculo, ya que tiene más nodos que expandir.

Si nos fijamos en las iteraciones de control, los promedios obtenidos son muy similares, de hecho, el mapa de gradiente no influye en este cálculo, ya que se realiza de forma dinámica en cada iteración, por lo que el cálculo es similar.

Capítulo 5

Práctica sobre Grafo de Visibilidad

A continuación se detallará el desarrollo efectuado para la creación de la práctica sobre la técnica de *Grafo de Visibilidad*. Para su implementación se ha utilizado la infraestructura descrita en el capítulo 3.

5.1. Algoritmo Grafo de Visibilidad

Para resolver el problema de la navegación global en un mapa determinado, la otra solución propuesta en este proyecto de fin de carrera es la construcción del Grafo de Visibilidad. Es aquel grafo en el cual cada nodo representa un vértice de los polígonos y las aristas son las conexiones visibles entre tales vértices. Para cada arista en el grafo definida, el segmento de recta que conecta los vértices correspondientes en el plano no intersecta con ningún polígono.

Para la construcción de esta técnica son necesarios como datos de origen: un mapa determinado, la posición del robot y la posición destino a la que se quiere que se desplace el mismo. El primer paso de su construcción es la creación de polígonos que engloben a cada uno de los obstáculos hallados en el mapa de origen. El margen de seguridad añadido de estos polígonos actuará como protección para que el robot no colisione con los obstáculos, por lo que debe ser lo suficientemente amplia para que el robot pueda maniobrar y esquivarlos.

En segundo lugar, es necesario crear un grafo de visibilidad tratando de unir cada uno de los vértices formados por todos los polígonos que engloban los obstáculos. Para ello, hay que tener en cuenta que las aristas formadas por dos de estos vértices, nunca deben tocar ni cruzar ningún obstáculo y tampoco cruzar ninguna línea de los polígonos creados. Los puntos en los que permanecen el robot y el punto destino también serán vértices posibles para la creación de las aristas.

Una vez creadas todas estas aristas, tendremos como resultado el Grafo de Visibilidad, y cada una de estas representará un tramo posible en la obtención de la trayectoria que deberá seguir el robot para alcanzar el punto destino.

La planificación de la ruta podrá realizarse con diferentes algoritmos como Dijkstra o A*, proporcionando como dato de entrada el grafo de visibilidad y como resultado el listado de vértices que el robot tiene que visitar para llegar a su destino.

En resumen, esta técnica crea una estructura de datos que alberga todos los caminos posibles por los que el robot puede transitar sin peligro de colisionar con ningún obstáculo.

5.2. Componente anfitrión TurtlebotVG

Para la práctica en la que los alumnos podrán implementarán la técnica de navegación global *Grafo de Visibilidad*, se proporcionará del mismo modo que en la técnica explicada en el capítulo anterior, un componente anfitrión. Este resolverá toda la problemática ajena a la construcción del algoritmo de navegación global y de un fichero «MyAlgorithms», el cual deberán completar los alumnos con la implementación de la construcción del grafo, la interpretación del mapa y el pilotaje del robot.

La estructura que necesita el alumno para probar sus algoritmos es la misma que se ha utilizado para el algoritmo *Gradient Path Planning*. Por un lado, se necesitará un simulador como Gazebo o el robot real para que sirvan las interfaces ICE necesarias para que se pueda comunicar el componente anfitrión tanto para enviar como recibir datos.

El componente cuenta con dos hilos de ejecución perfectamente diferenciados. Por un lado tiene un hilo que actualiza en tiempo real la interfaz de usuario y por otro lado, tiene otro hilo de control que actualiza la información de la API con la información obtenida del plugin «Pose3d» y actualiza la velocidad del robot mediante la interfaz que se comunica con el plugin «Motors».

Cuando se arranca el componente anfitrión, éste lee la información contenida del fichero de recursos, dónde está ubicada toda la información dónde los sensores y actuadores del robot están sirviendo los datos para poder crear las interfaces ICE, y la ruta física del fichero que contiene el mapa. El único formato aceptado en esta técnica será «World», que es el mismo tipo de fichero que puede leer *Gazebo*. Una vez leída esta información el componente interpreta el mapa y crea tres estructuras de datos: una lista de obstáculos, una lista de las líneas de protección de los obstáculos que corresponden con los polígonos que los engloban y por último, una lista con los vértices creados por la intersección de las líneas que forman los polígonos.

Estas listas están a disposición de los alumnos, ya que permanecen en la memoria compartida «API», y podrán manejarlas como crean conveniente. La lista obligatoria que necesitará el alumno para poder crear el grafo de visibilidad será la lista de obstáculos. La creación de las otras dos listas mencionadas en el arranque del componente se han creado por razones de optimización y los alumnos podrán utilizarlas o no a su elección.

Los alumnos deberán completar tanto la construcción del grafo de visibilidad como el pilotaje del robot interpretando el grafo creado por dicha técnica. Para la construcción del grafo, dispondrán de las estructuras de datos comentadas previamente y de un punto destino elegido por el usuario a través de la interfaz. El componente comprobará si el destino es un objetivo válido, entendiendo por ello, que se ha elegido un punto del espacio libre del escenario. De no ser así, se notificará por medio de un mensaje de texto en la propia interfaz. En este método el alumno deberá de implementar la construcción del grafo de visibilidad y almacenarlo sobre la variable «Grafo» que pertenece a la memoria compartida del componente. De forma paralela, una vez construido el grafo de visibilidad, el alumno deberá crear un algoritmo de planificación de la ruta que será la trayectoria pintada en la interfaz.

En cuanto a la resolución del pilotaje utilizando la ruta previamente creada, se proporcionarán

el punto destino y la interfaz ICE que se comunica con los motores del robot para que se pueda dirigir su movimiento.

La interfaz de usuario es muy similar a la explicada en capítulo anterior. De la misma manera, contendrá información sensorial en forma de texto o en forma gráfica en el visor del mapa. También contendrá las utilidades explicadas de cambio de posición del robot y activación de la utilidad que muestra u oculta el rastro por donde el robot ha pasado en toda su ejecución. Además, la API contiene las mismas variables que el alumno tendrá a su disposición si necesita pintar los subobjetivos que se le van marcando al robot en su ejecución con el propósito de visualizarlo en tiempo real.

Las funciones de la interfaz diferentes son las que se despliegan una vez se ejecuta la construcción del grafo de Visibilidad. Por defecto se activan las opciones «Vista de protección de paredes» y «Vista del grafo de visibilidad». Dependiendo del mapa, si éste contiene un número de obstáculos elevado, implica que el número de vértices generados para la construcción del grafo sea mayor, y por tanto, el número de aristas que identifican los caminos posibles que puede atravesar el robot sea muy elevado. Esto origina un número del líneas en el visor demasiado grande que puede dificultar la visualización del robot o de su trazado. Por esto, se ofrece al usuario la posibilidad de mostrar u ocultar las líneas correspondientes a la protección de paredes y las líneas que forman el grafo de visibilidad.

Otra función propuesta por la interfaz es la encargada de mostrar la planificación de la ruta. Esta funcionalidad sólo estará disponible si el alumno completa la estructura de datos creada en la memoria compartida, llamada «listaCaminoOptimoVG». Se utilizará para que el alumno pueda visualizar la ruta que ha calculado sobre el visor de la interfaz.

La última funcionalidad ofrecida por la interfaz es la activación de la navegación. Esta función únicamente activará el control del robot y para ello, llamará a la función que debe implementar el alumno contenida en el fichero «MyAlgorithms.

5.3. Solución implementada del Grafo de Visibilidad

La solución desarrollada tendrá como objetivo, servir como ejemplo de orientación a los alumnos que realicen estas prácticas.

La solución propuesta está contenida en el fichero «MyAlgorithms», que inicialmente se presentará vacío a los alumnos. En los siguientes apartados, se detallará la solución que he construido para resolver el problema de la navegación global con esta técnica. Estará diferenciada en dos partes: la construcción del grafo y el pilotaje del robot.

5.3.1. Construcción del grafo de visibilidad

La creación del grafo de visibilidad consiste en la construcción de todos los caminos posibles por los que puede transitar el robot con seguridad de que no colisionará con ningún obstáculo, teniendo en cuenta una distancia de seguridad a los mismos.

Para la construcción del grafo se han utilizado las tres estructuras de datos proporcionadas por el componente anfitrión. En primer lugar, se utilizará la lista de vértices, se cruza cada vértice

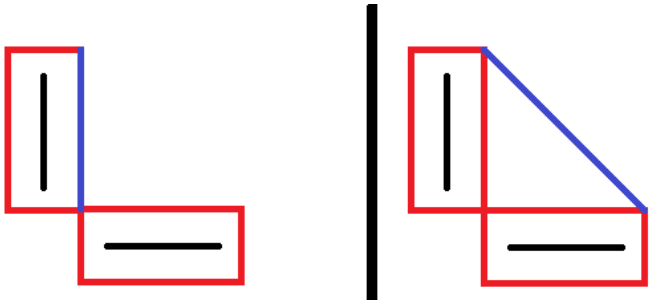

Figura 5.1: Aristas válidas

con el resto para obtener todas las aristas del mapa. Una arista creada con dos vértices es válida si cumple las siguientes condiciones:

- La arista no puede tocar, ni atravesar ningún segmento de la lista de obstáculos.
- La arista no puede atravesar ningún segmento de la lista de protecciones de los obstáculos.

En ningún caso una arista puede tocar un segmento que define un obstáculo. Esta arista habría sobrepasado alguno de los segmentos que proporcionan una distancia de seguridad a los obstáculos. Sin embargo, una arista sí puede tocar o incluso contener un segmento de protección, ya que en ningún caso proporciona peligro de colisionar con algún obstáculo. En la figura 5.1, se puede observar dos tipos de aristas que tocan un segmento de protección y ambas son válidas.

Todas las aristas que cumplan estas condiciones serán caminos posibles por los que el robot podrá transitar. Los puntos correspondientes a la posición en la que se encuentra el robot y la posición destino a la que se desea que el robot navegue tendrán que estar incluidos cómo vértices antes de ejecutar el algoritmo que crea el grafo de visibilidad, con el objetivo de que cada uno de ellos se relacione con el resto de vértices el mapa y poder crear aristas posibles con ellos. En la figura 5.2 se puede observar un ejemplo del grafo de visibilidad.

La planificación de la ruta se podrá implementar con técnicas cómo A*, Dijkstra, etc. En este caso he utilizado Dijkstra *(Edsger Dijkstra, 1959)* cómo función de planificación de la ruta dentro del grafo. Este algoritmo determina el camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su funcionamiento consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices del grafo. Cuando este algoritmo encuentra el camino más corto, se detiene.

El algoritmo Dijkstra se ejecuta a continuación de la creación del grafo, y el resultado del mismo se almacena sobre la lista de la memoria compartida «listaCaminoOptimoVG», lo cual, será necesario para la visualización en el componente anfitrión de la ruta elegida. Esta lista contiene una lista de los nodos que el robot tiene que visitar para llegar a la posición destino. Posteriormente se utilizará esta lista para determinar los subobjetivos que usará el algoritmo de pilotaje para la navegación del robot. En la figura 5.3, se puede ver un ejemplo de la planificación de ruta.

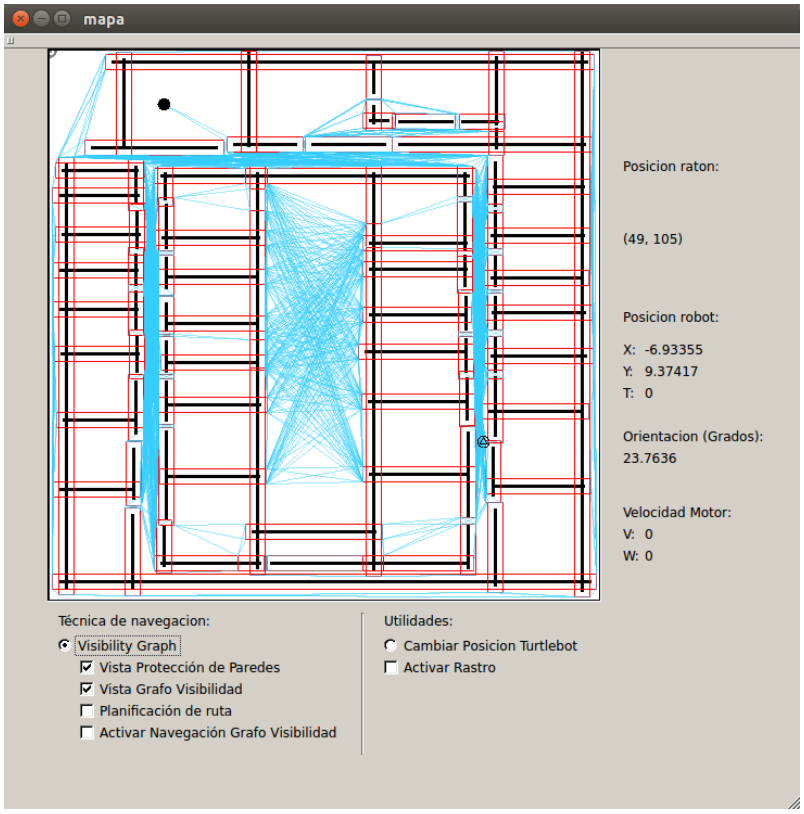

Figura 5.2: Grafo de Visibilidad

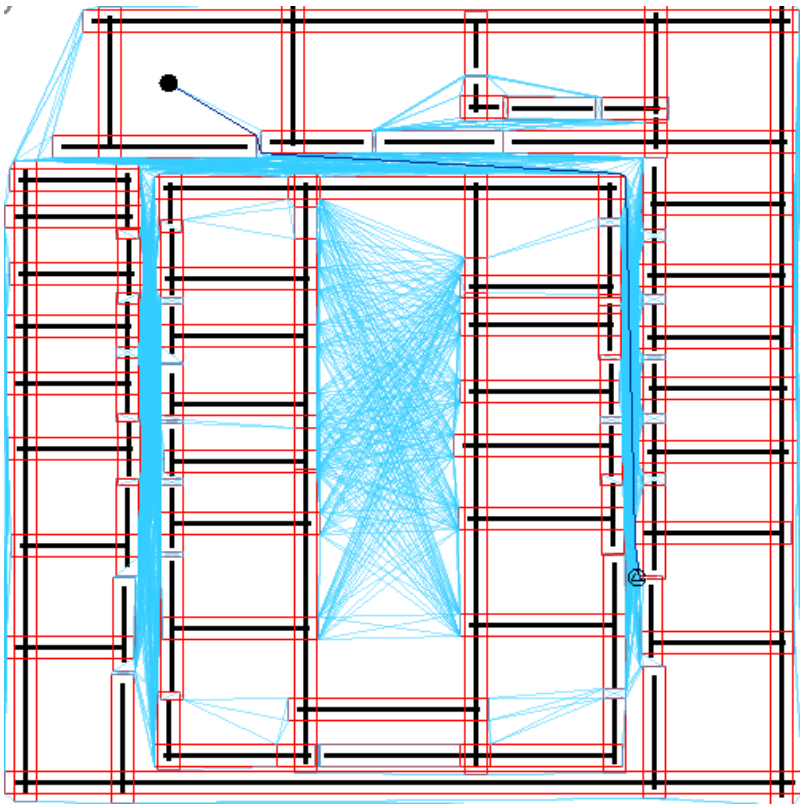

Figura 5.3: Planificación Grafo de Visibilidad

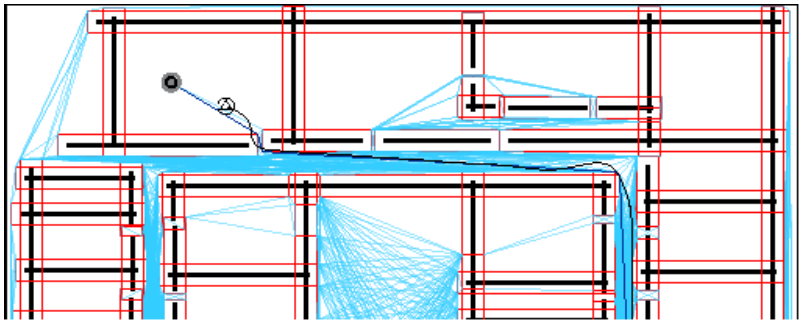

Figura 5.4: Trazado giro con Grafo de Visibilidad

5.3.2. Pilotaje del robot

El pilotaje del robot que interpreta el grafo de visibilidad está basado en la visita de una serie de nodos, elegidos por la planificación calculada con el algoritmo Dijkstra, cuyo último nodo corresponde con la posición destino. El pilotaje está dividido en el cálculo del siguiente subobjetivo al que se pretende desplazar y el cálculo de la velocidad lineal y angular del robot que modifican la trayectoria del robot hacia el subobjetivo planteado.

El hilo de control que ejecuta el componente anfitrión llama a la función de navegación en intervalos de tiempo cortos. En cada iteración, el algoritmo descrito del pilotaje del robot comprueba si se ha alcanzado el siguiente nodo que tiene que visitar. Si lo ha hecho, cambiará el subobjetivo al siguiente nodo y eliminará éste de la lista. Hasta que el robot no visite el nodo que tiene como subobjetivo, éste no cambiará. Para optimizar la navegación, en cada iteración se calcula la distancia que hay entre la posición del robot y el nodo que tiene que visitar, si éste nodo está muy cerca, el subobjetivo cambiará al siguiente vértice, sin la necesidad de que el robot pase exactamente por la posición del nodo.

Esta optimización repercute directamente en la navegación del robot, ya que cuando éste realiza un giro, cuenta con la inercia que lleva de los desplazamientos inmediatamente anteriores, lo que produce que tome los giros abiertos. Si la condición de nodo visitado se restringe a que el robot obligatoriamente tenga que pasar por la posición del nodo, se producirá un retroceso en el giro y por lo tanto. no se podrán realizar giros con soltura. En la figura 5.4, se puede observar el trazado de un giro que realiza el robot, respecto al trazado pintado por la trayectoria calculada.

A diferencia del pilotaje con *Gradient Path Planning*, cuyos subobjetivos establecidos eran muy próximos al robot y cambiaban con mucha frecuencia, en el pilotaje de Grafo de Visibilidad, el cambio de subjetivos se realiza con mucha menos frecuencia. Los subobjetivos están posicionados en la mayoría de los casos en puntos lejanos al robot. Al tenerlos prácticamente constantes, se asegura que la trayectoria que debe tener el robot es constante cada tramo establecido, lo que repercute directamente en la velocidad que se le puede establecer al robot, que es mucho mayor a la establecida en la técnica anterior. También repercute directamente en la oscilación del robot, ya que al no variar tanto, el robot tiene más espacio para decrementar la oscilación hasta hacerla cero.

El control del robot está basado en un control por casos, de la misma manera que en el algoritmo anterior, salvo que se han incrementado la velocidad lineal en todos los casos. Además,

se ha reutilizado y reajustado, el controlador PID para evitar oscilaciones.

En cada iteración se comprueba si el nodo fijado como subjetivo al que se está dirigiendo es el nodo que identifica la posición final, y acto seguido, se comprueba si la distancia si el robot ha alcanzado este subobjetivo. De ser así, el robot interpretará que ha alcanzado su meta y se detendrá.

5.4. Experimentación

En esta sección se realizarán varios casos de prueba sobre dos escenarios diferentes, el primero es el más sencillo, consta de 3 obstáculos. Este mapa tiene como objetivo mostrar la construcción del grafo de visibilidad ya que en mapas más complejos el cruce de las aristas del grafo impiden comprobar con facilidad si el grafo está bien formado.

Las pruebas realizadas se harán de la misma manera que en la técnica explicada en el capítulo anterior, con la diferencia de que en esta técnica no se modificará la velocidad angular. En el pilotaje de *Gradient Path Planning*, los subobjetivos son calculados en cada iteración, por lo que cuando el robot realiza un giro, la velocidad angular es crítica. En el pilotaje del robot mediante un grafo de visibilidad los subobjetivos marcados son constantes en varias iteraciones de control, por lo que el ajuste de este parámetro no provoca un decremento del tiempo de ejecución.

En los siguientes apartados, se mostrarán imágenes y comparativas que determinan el tiempo de navegación del robot.

5.4.1. Mapa básico

La figura 5.5 representa la situación inicial en todos los casos de prueba realizados. Contaran con el mismo escenario, posición del robot de partida y posición destino. En estas pruebas, se han modificado parámetros como la velocidad lineal la aceleración.

Tras los primeros casos de pruebas, en los que se ha modificado la velocidad lineal y el incremento de la aceleración en las trayectorias rectilíneas se han conseguido mejores tiempos de ejecución. Tal y como podemos comprobar en la tabla expuesta a continuación, la mayor diferencia de tiempos de ejecución obtenidos se produce al aumentar el incremento en la aceleración y la velocidad lineal.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4
Velocidad Lineal	1x	1x	1.2x	1x
Aceleración máxima	64	64	64	84
Incremento aceleración	1x	3x	3x	3x
Tiempo de ejecución	1:45	1:38	1:30	1:29

En la figura 5.6, la primera imagen representa el trazado de ejecución correspondiente al caso base. La segunda, corresponde al ejemplo 3 de la tabla. En estas imágenes, se puede observar en la última, que los giros efectuados por el robot son más pronunciados, aún así, la navegación del robot endereza tu trayectoria y gana tiempo mediante su aceleración.

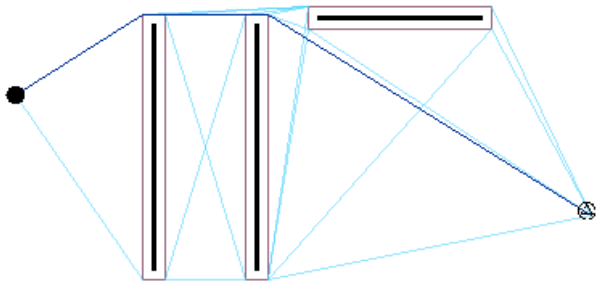

Figura 5.5: Mapa básico 1

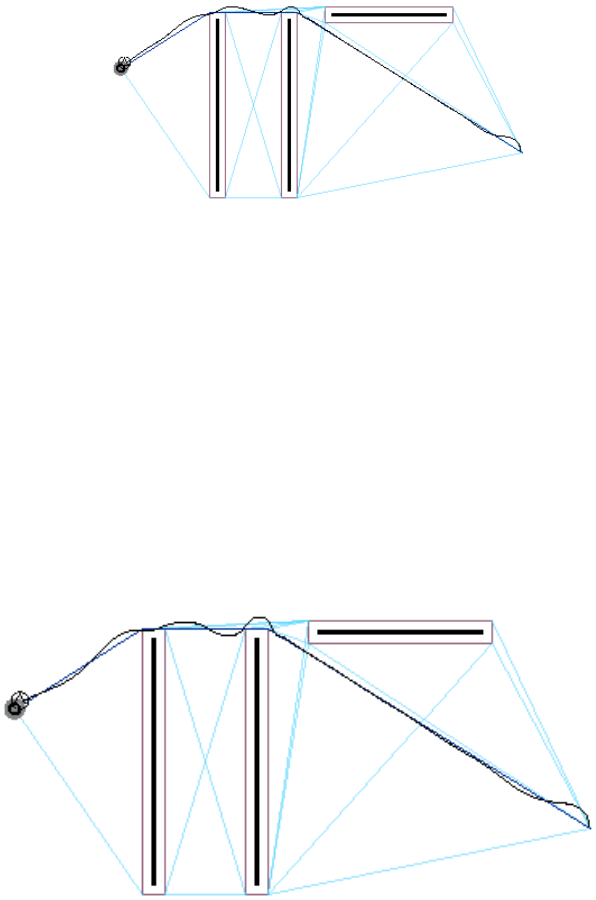

Figura 5.6: Mapa básico 2

5.4.2. Mapa edificio departamental

En el segundo caso de pruebas se ha utilizado el mismo escenario utilizado en *Gradient Path Planning*, el mapa departamental. La situación inicial está representada en la figura 5.7. Junto a esta imagen, se expone el primer caso de pruebas correspondiente a la navegación inicial del robot, sin modificaciones en los parámetros referentes a la velocidad.

El pilotaje de este algoritmo es más veloz que el de la técnica descrita en el capítulo anterior. Utilizando los mismos datos de entrada, se han obtenido tiempos mejores. Esto es debido en gran parte, a los subobjetivos fijados por cada una de las técnicas. Como se ha explicado previamente, con esta técnica los subobjetivos son constantes en muchas iteraciones de control, lo que hace que el robot se adecúe con mayor facilidad a la trayectoria que debe seguir. Las modificaciones en los parámetros de la velocidad en este pilotaje no han supuesto diferencias muy elevadas como se puede comprobar en la siguiente tabla:

	Prueba 1	Prueba 2	Prueba 3	Prueba 4
Velocidad Lineal	1x	1x	1.2x	1x
Aceleración máxima	64	64	64	84
Incremento aceleración	1x	3x	3x	3x
Tiempo de ejecución	1:27	1:23	1:15	1:12

En todos los casos de prueba efectuados, la mayor diferencia en cuanto a tiempos de navegación sucede en una mayor aceleración. Esta se produce cuando el robot tiene la como orientación la misma que la trayectoria que ha de seguir durante varias iteraciones, por lo tanto, es el caso en el que la trayectoria del robot es más estable y dónde más velocidad se le puede dar. La velocidad máxima también influye mucho en los tiempos, ya que puede recorrer una trayectoria rectilínea a mayor velocidad. Este parámetro siempre será obligatorio y habrá que adecuarlo al mapa en el que se ejecuta, ya que si este contiene espacios libres muy limitados, predeterminar una velocidad máxima demasiado grande puede producir la colisión del robot, ya que tendrá mayor inercia y menor tiempo y espacio para maniobrar y esquivar un obstáculo.

En la figura 5.8, se pueden ver las trayectorias que ha seguido el robot en las ejecuciones de las pruebas 3 y 4 correspondientemente.

5.4.3. Análisis computacional

En este apartado se analizan los tiempos de ejecución correspondientes a la creación del grafo de visibilidad y al promedio de las ejecuciones de una iteración de control del pilotaje. Midiendo estos tiempos, los alumnos podrán comprobar si las implementaciones desarrolladas son óptimas computacionalmente o no.

En la siguiente tabla se muestran los tiempo que necesita un computador para ejecutar estas dos partes del algoritmo, en los dos mapas dónde se han realizado las experimentaciones.

	Básico	Departamental
Construcción grafo de visibilidad	0.000546 ms	0.811061 ms
Ejecución iteración de control	0.000242 ms	0.000230 ms

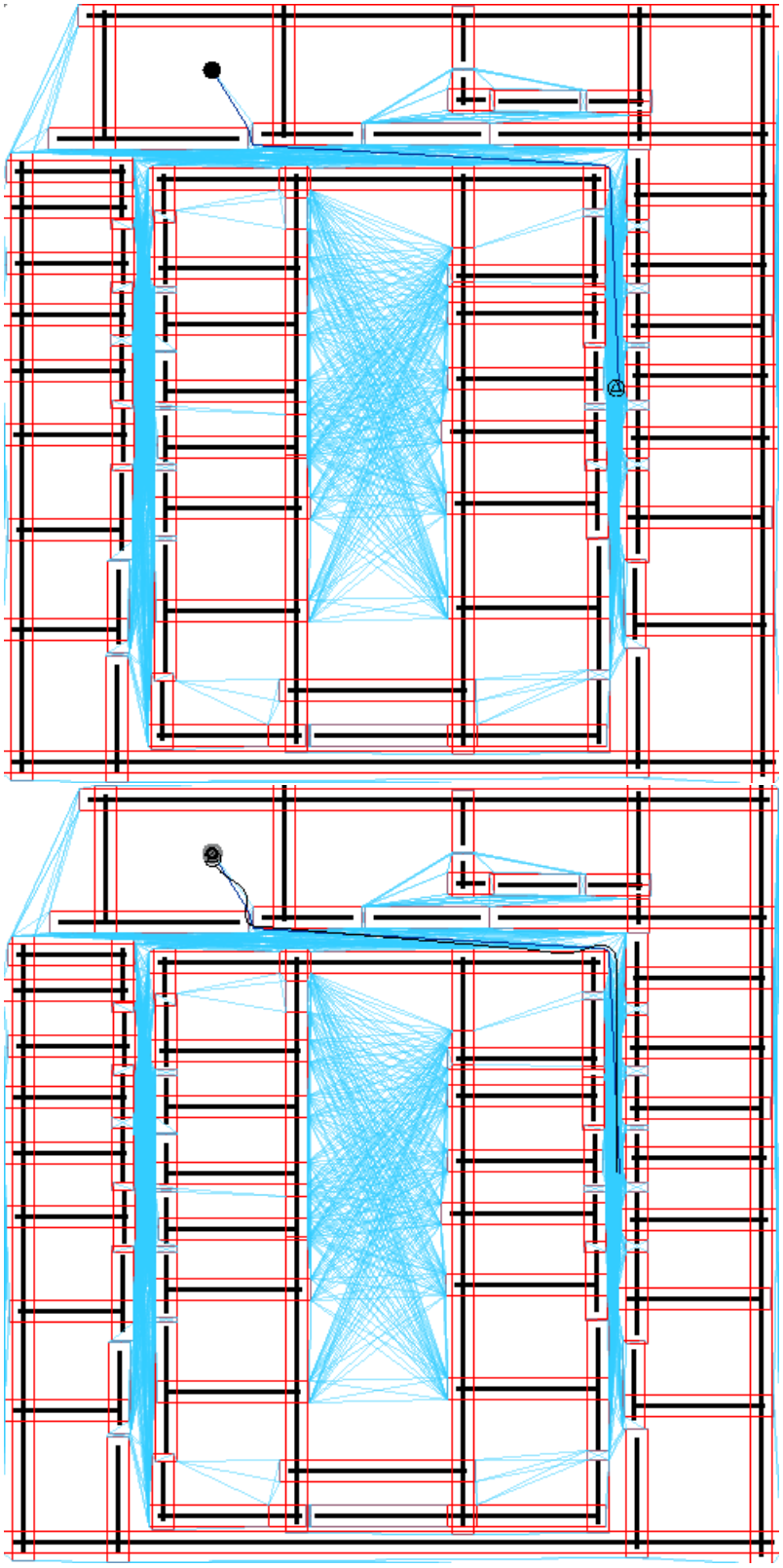

Figura 5.7: Mapa del departamental 1

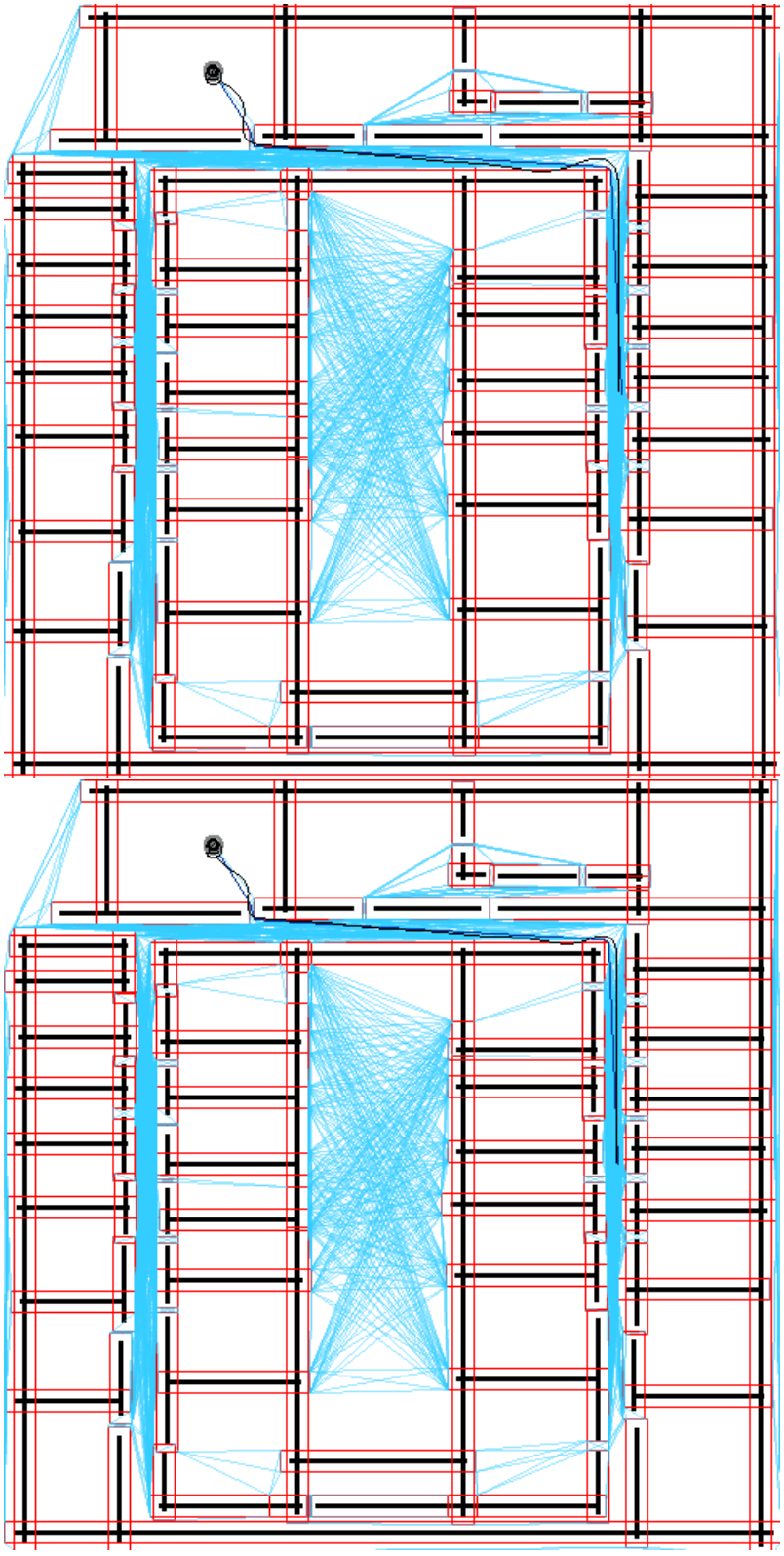

Figura 5.8: Mapa del departamental 2

Si comparamos estos resultados con los obtenidos en el capítulo anterior, observando al mapa departamental, común a ambas pruebas, podemos observar que la construcción del grafo de visibilidad es mucho más costosa computacionalmente que la construcción del mapa de gradiente. Los tiempos de ejecución de una iteración de control son muy similares en ambas técnicas, ya que se basan en los mismos principios, el cálculo del siguiente subobjetivo que marcará la trayectoria del robot, y un sistema de control basado en casos que transmitirá las velocidades lineal y angular al robot.

Con los tiempos obtenidos con esta técnica de navegación global, también podemos sacar la conclusión de que cuantos más obstáculos haya en el escenario, mayores serán los tiempos de creación del grafo de visibilidad.

Capítulo 6

Conclusiones

En este capítulo se resume en qué medida se han cumplido los objetivos propuestos y si se han satisfecho los requisitos planteados. También se hará énfasis sobre los conocimientos adquiridos en la ejecución del proyecto de fin de carrera.

6.1. Conclusiones

A lo largo de esta memoria se ha descrito la creación de los dos componentes académicos destinados a la docencia en robótica, con el propósito de que los alumnos puedan afianzar los conocimientos teóricos sobre la navegación global, poniéndolos en práctica sobre dichos componentes. Además, se ha propuesto una solución para cada una de las técnicas propuestas (*Gradient Path Planning* y Grafo de Visibilidad) y los resultados de su experimentación.

En este proyecto, me he enfrentado a la problemática de la navegación global. Se ha abordado este problema partiendo de una base en conocimientos en robótica básicos. Los componentes didácticos se han desarrollado teniendo en cuenta, en la medida de lo posible, todos los problemas ajenos que los alumnos necesitan resolver para poder ejecutar sus algoritmos de navegación global.

En cuanto a la solución propuesta, se ha comprobado que la técnica de navegación *Gradient Path Planning*, tiene gran complejidad en la ejecución del pilotaje del robot. Como este algoritmo no tiene una trayectoria premeditada, si no que en cada momento evalúa cuál es la mejor opción que tiene, dificulta el pilotaje del robot autónomo, ya que este comportamiento produce oscilaciones e influye directamente en la velocidad con que ejecuta sus trayectorias. Con esta técnica, se deben de asumir menos riesgos en su pilotaje. Esto nos ha servido para identificar los puntos fuertes dentro de su navegación y aprovecharlos para conseguir mejores resultados.

Por otro lado, hemos podido observar que la creación del grafo de visibilidad puede ser muy compleja computacionalmente, dependiendo del mapa, ya que además de enlazar cada uno de los puntos origen del grafo de visibilidad, debe de ser comprobada su validez. La complejidad de la construcción del grafo de visibilidad es proporcional al número de obstáculos del escenario propuesto. Se han obtenido buenos resultados en la navegación de esta técnica, ya que ofrece en su ejecución intervalos de tiempo mayores para la estabilización de la trayectoria del robot.

Cabe destacar también que se han realizado aproximadamente 1300 líneas de código para implementar el componente académico «TurtlebotGPP» y 1400 en el componente académico

«TurtlebotVG». Las soluciones desarrolladas que resuelven cada una de las prácticas planteadas, contienen aproximadamente 400 líneas de código cada una.

6.2. Trabajos futuros

Una de las mejoras que se podrían realizar relacionadas con el problema de la navegación de los robots, serían la incorporación de la información sensorial de los dispositivos integrados en el robot Turtlebot, como el láser, o el Kinect. Con esta información, y con técnicas como VFF (Virtual Force Field), que genera trayectorias por medio de la suma vectorial de fuerzas atractivas y repulsivas, se podría incorporar la detección de obstáculos dinámicos en el escenario del robot, y con ello, obtener una navegación híbrida más segura.

Por otra parte, se podrían crear otros componentes académicos a imagen y semejanza de estos, que satisfagan otras técnicas de navegación global, o la incorporación de librerías como OMPL (Open Motion Planning Library), que contiene multiples algoritmos de planificación de movilidad.

Respecto a los componentes propuestos, se podría optimizar la construcción del mapa de gradiente y del grafo de visibilidad e implementar otros algoritmos de navegación que obtengan mejores resultados en las ejecuciones de sus trayectorias. En cuanto a la técnica del grafo de visibilidad se pueden implementar otros algoritmos de planificación de ruta como A*.

Bibliografía

[Raúl Isado, 2015] José Raúl Isado. Navegación global utilizando método del gradiente. *Proyecto de fin de carrera, Universidad Rey Juan Carlos*, 2015.

[Julio Vega, 2008] Julio Vega Pérez. Navegación and autolocation over a guide robot. *Proyecto de fin de carrera, Universidad Rey Juan Carlos*, 2008.

[Alejandro López, 2005] Alejandro López. Navegación global utilizando grafo de visibilidad. *Proyecto de fin de carrera, Universidad Rey Juan Carlos*, 2005.

[Cañas, 2002] Jose María Cañas. Dynamic schema hierarchies for an autonomous robot. *Universidad de Sevilla*, 2002.

[Cañas, 2003] Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis Doctoral*, 2003.

[Cañas, 2004] Jose María Cañas. Manual de programación de robots con jde. *Universidad Rey Juan Carlos*, 2004.

[Crowley, 1985] James L. Crowley. Navigation for an intelligent mobile robot. IEEE Journal of Robotics and Automation. 1985

[Ford, 1984] D.R Ford, L.R. y Fulkerson. Graphs and algorithms. Wiley, 1984.

[Nathan Koenig and Andrew Howard, 2004] Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. 2004.