UNIVERSIDAD DE MÁLAGA

PhD Thesis - Tesis Doctoral

Efficient Solutions in Path Planning for Autonomous Mobile Robots

Alejandro Hidalgo Paniagua

Departamento de Tecnología Electrónica

Supervised by:

Dr. Antonio Jesús Bandera Rubio

Dr. Juan Pedro Bandera Rubio

July 2018

UNIVERSIDAD DE MÁLAGA

AUTOR: Alejandro Hidalgo Paniagua

http://orcid.org/0000-0002-1823-9592

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

UNIVERSIDAD DE MÁLAGA

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

El Dr. Antonio Jesús Bandera Rubio, Profesor Titular de Universidad, perteneciente al Departamento de Tecnología Electrónica de la E.T.S de Ingeniería de Telecomunicación de la Universidad de Málaga, y el Dr. Juan Pedro Bandera Rubio, Profesor Titular de Universidad, perteneciente al Departamento de Tecnología Electrónica de la E.T.S de Ingeniería de Telecomunicación de la Universidad de Málaga,

Certifican que D. Alejandro Hidalgo Paniagua, Ingeniero en Informática, ha realizado en el Departamento de Tecnología Electrónica de la Universidad de Málaga, bajo nuestra dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulada:

Efficient Solutions in Path Planning for Autonomous Mobile Robots

Revisado el presente trabajo, estimamos que puede ser presentado al tribunal que ha de juzgarlo, y autorizamos la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Málaga, Julio de 2018

Fdo. Antonio J. Bandera Rubio
Prof. Dpto. Tecnología Electrónica

Fdo. Juan P. Bandera Rubio
Prof. Dpto. Tecnología Electrónica

"La perseverancia y la paciencia tienen un efecto mágico ante el que las dificultades desaparecen y los obstáculos se desvanecen."
— John Quincy Adams (1767-1848)

A mi pareja, Nuria, y a mis padres, Ricardo y Carmen, por su apoyo incondicional durante todos estos años.

Contents

List of figures

List of tables

Abstract

Each day robots are more integrated into society. In the past, robots were static machines that performed very specific tasks in the factories assembly lines. These robots had no locomotion system and that's why they always stayed in the same place. They were also not equipped with decision systems, cognitive systems or with complex systems based in Artificial Intelligence (AI). In fact, these last terms were poorly known or they practically were nonexistent. Briefly, robots were mechanical machines that were programmed to perform repetitive and monotonous tasks.

Later, the technological advancements made scientists and researches to start thinking in more complex machines until the point of imagine human-like machines. In this standpoint the main challenges were to equip the old robots with a locomotion system to allow them moving along the available space and provide them a certain degree of intelligence that allow robots to take decisions to involve them into more complex and dynamic and tasks.

Taking into account that the surrounding space of the robot is full of obstacles and potential dangers. If the robot collides with one of these obstacles or dangers when moving it could be damaged. This is the Path Planning (PP) problem and it is one of the most known and researched problem in robotics. Hence, the PP has the aim of finding an obstacle-

free path that allows the robot to safely move through the surrounding environment.

In the most of cases, the motion planner needs to be intelligent in the sense of it has to take reasonable decisions to reach the target goal in the best possible way. Therefore, in these cases, the robot navigation system and robot intelligence are pretty related. Furthermore, many applications have real time requirements and it is necessary to get the PP solutions as fast as possible (into reasonable margins of time).

This doctoral dissertation studies the PP problem mainly from the real time perspective but also considering the robot intelligence. High Performance Computing (HPC), based on *multi-core* Central Processing Units (CPUs) and Graphics Processing Units (GPUs), and Multi-Objective Evolutionary Algorithms (MOEAs) have been used to accomplish the thesis goals. In addition, a novel and innovative proposal to solve the PP problem using real large scale maps in applications with real time requirements has been presented. This is the quad-RRT algorithm.

Acknowledgments

Aún recuerdo cuando era estudiante de último curso de Ingeniería Informática y me ofrecieron la oportunidad de conocer y trabajar en el campo de la robótica. Tanto en aquel momento como a día de hoy, todas y cada una de las disciplinas que convergen en la robótica, desde la electrónica fundamental hasta la inteligencia artificial, despiertan en mí una inquietud constante que me lleva a un proceso de aprendizaje continuo e incansable. Esta inquietud constante por descubrir hace que comiences a cuestionarte ciertas cosas e incluso que surjan nuevas ideas. Por ejemplo, ¿Y si yo hiciera esto así?, ¿Podría funcionar este planteamiento?, etc. Todas estas preguntas surgen o bien porque las propuestas existentes no se ajustan a tus requerimientos o bien porque aún no tienen respuesta. Cuando este tipo de cuestiones se ponen en práctica nos encontramos en una fase de investigación. A mí me invadían estas preguntas, y es por eso por lo que decidí a aventurarme a realizar una tesis doctoral. Ha sido un proceso muy duro, cargado de contratiempos e infortunios, que incluso en alguna ocasión me llegó a parecer un imposible y en otras incluso hasta plantearme abandonar. Estos pensamientos negativos se superan gracias al apoyo incondicional de las personas que te quieren y te aprecian y que, realmente, caminan junto a ti durante todo el tiempo en el que transcurre la tesis. Nada hubiera cobrado sentido sin el cariño de todas estas per-

sonas que sin duda son merecedores de estos y todos los agradecimientos posibles.

En primer lugar agradecer de manera muy especial, hoy y siempre, a D. Antonio Jesús Bandera Rubio y a D. Juan Pedro Bandera Rubio, directores de esta tesis doctoral, porque vosotros conocéis de primera mano todas las dificultades y adversidades acontecidas hasta culminar este trabajo y sin duda alguna si algo ha hecho que esta investigación esté donde está es gracias a vosotros. Por toda vuestra ayuda, por abrirme las puertas de vuestro laboratorio como si hubiera sido mi casa de toda la vida, por el trato impecable que he recibido de vosotros y, sobre todo, porque habéis sido las únicas personas que me han tendido la mano cuando todo parecía perdido. Muchas gracias.

No puedo pasar por alto agradecer a mis padres, mi familia, D. Ricardo Hidalgo Guerrero y D.ª Mª del Carmen Paniagua Álvarez todo su apoyo incondicional, en los buenos y en los malos momentos, y por los valores que me han inculcado, que de alguna u otra manera me han convertido en la persona que soy. Gracias de corazón.

A mi nueva familia, especialmente a mi pareja, D.ª Nuria Gómez Ojeda, por aguantarme durante todo este tiempo sin perder la paciencia y la esperanza en mí, por ser mi confidente y mi compañera de vida, por su cariño y por su forma de ser y que, en definitiva, ha hecho que todo esto haya sido mucho más ameno. A sus padres, D. Andrés Gómez Vizuete y D.ª Antonia Ojeda Grueso, por su acogida, por su trato hacia mi persona y por haberme abierto las puertas de su casa como si de su hijo se tratase. Mil gracias.

Agradecer a Aeorum España S.L por todo el material que ha puesto a mi disposición, de manera desinteresada, y que ha permitido que este trabajo quede culminado. Agradecer también de forma muy especial a D.ª Estefanía García Sánchez por su ayuda y su trabajo desinteresado y que ha permitido darle un aire moderno y renovado a la parte gráfica de esta tesis doctoral.

Agradecer de todo corazón el apoyo de mis amigos, especialmente a D. José Mateos Sánchez, D. José Moreno del Pozo y a D. Santiago Salamanca Miño, por toda su ayuda, por todos sus ánimos y por todas las muestras de cariño recibidas. Gracias por todo lo que me habéis enseñado.

Finalmente a mis compañeros de batalla, D. David Álvarez Carrizosa, D. José Carlos Moruno Sáez, D. Fernando Jiménez Mejías, D. Juan Cabello Castillo y D. Manuel Guisado Quintana, por interesarse por mí en los peores momentos, por convertir los malos momentos en buenos momentos y por hacer de psicólogos personales cuyas charlas me reconfortaban bastante. Amigos, gracias.

De nuevo y a todos, gracias.

PART 1

Resumen

CHAPTER

1 Resumen

"Crear un ser artificial ha sido el sueño del hombre
desde que nació la ciencia."
— Steven Spielberg, Inteligencia Artificial: I.A (2001)

1.1 Introducción

Un mundo poblado de robots. Máquinas que desempeñan un abanico de tareas de lo más variopinto. Desde realizar tareas muy específicas en cadenas de montaje o ensamblado de componentes, hasta desempeñar la mayoría de labores cotidianas que los seres humanos tenemos que afrontar cada día. Como se puede intuir, para esto se necesitan no solo máquinas, sino máquinas dotadas de cierta inteligencia, que surge de la necesidad de que las máquinas abandonen su estatismo y monotonía para comenzar a enfrentarse a un mundo dinámico y ambiguo: nuestro mundo. Esto son los robots.

Uno de los factores que ha llevado al ser humano a dotar de inteligencia y movilidad a las máquinas es su afán de dominar y, al mismo tiempo, liberarse de un entorno cada vez más estresante. Un ejemplo claro lo encontramos en los populares robots para la limpieza del hogar. Estos

robots, a pesar de tener una funcionalidad muy limitada, pueden ser programados para que, periódicamente, realicen rondas de limpieza. En un mundo cada vez más estresante, como el nuestro, se está haciendo habitual encontrar a estos pequeños *"empleados cibernéticos"*, que nos permiten liberarnos de ciertas tareas y disfrutar más de nuestro poco tiempo libre. Aunque este ejemplo es bastante simple, es lo suficientemente ilustrativo como para pensar con cierta convicción que esta será la tendencia de futuro.

Está claro que la robótica es, quizá, la revolución tecnológica más importante del siglo XX. Pero también es indudable de que esta revolución no ha hecho más que empezar, y así lo confirman los avances que se producen cada día gracias a la investigación en este campo. Estos avances son los que nos permiten pensar en máquinas cada vez más complejas, cada vez más inteligentes y cada vez más versátiles. Lo bueno es que este pensamiento se materializa de manera muy rápida.

Como ya se ha podido intuir en estas pocas líneas, hay dos aspectos irrefutables que marcan la versatilidad de una máquina: su inteligencia y su movilidad. Aunque ambos conceptos no son excluyentes, sí están estrechamente relacionados cuando hablamos de robótica. No tiene sentido (en la mayoría de los casos) estudiar profundamente la parte inteligente subestimando la parte móvil y viceversa. Los seres vivos, por ejemplo, cuando necesitan alcanzar un objetivo en un entorno desconocido es posible que las primeras veces encuentren la solución a través de un procedimiento de prueba y error. Sin embargo, lo normal es que esta solución sea mejorada constantemente por la experiencia del propio ser. Esta es la relación movilidad-inteligencia.

Hablando de robótica y movilidad surge el problema de cómo y por dónde debe moverse un robot para alcanzar un determinado objetivo sin comprometer su integridad física. Como el significado de moverse puede ser muy amplio, aquí hablaremos de desplazamiento, en el sentido literal de viajar. Y cuando viajamos a algún lugar siempre nos preguntamos lo siguiente: ¿Por dónde vamos? y ¿Cuál es el la mejor alternativa?. Esta problemática, en robótica, se conoce como el problema del PP.

Hay que tener en cuenta que cuando viajamos existe un factor que puede llegar a ser más importante que por dónde vamos o qué alternativa

es la mejor. Este factor es el tiempo. Siempre que viajamos queremos hacerlo lo más rápido posible. Esta condición puede ser más o menos estricta, pero en algunos entornos se convierte en una condición poco flexible. Es decir, el hecho de viajar está fuertemente condicionado por restricciones temporales. Por ejemplo, pensemos en el caso de un robot que forma parte de un sistema de seguridad. Ante un desplazamiento del robot causado por una situación de alerta, no importa mucho ni la solución ni la calidad de la planificación estimada, pero sí importa que reaccione y se mueva en unos márgenes de tiempo críticos.

Otro aspecto importante relacionado con la movilidad es la inteligencia. Y también es muy frecuente en los tiempos que corren oír hablar de Inteligencia Artificial (IA). En este contexto, el hecho de que un robot no solo se desplace sino que lo haga de manera inteligente nos compromete, en cierta medida, a estudiar la naturaleza. Esto es, observar cómo actúan los seres vivos, qué patrones siguen, cómo se organizan y colaboran y cómo discriminan algunas soluciones potencialmente correctas. De esta manera, si los robots pudieran reproducir estos patrones de comportamiento manifestarían, muy probablemente, comportamientos más inteligentes tanto a la hora de realizar acciones como a la hora de desplazarse (o moverse). Dejando a un lado las teorías cognitivas, uno de los campos más prometedores de la IA son los algoritmos basados en metaheurísticas. Estos algoritmos se caracterizan porque son capaces de generar soluciones a problemas, por lo general buenas, pero sin garantizar la solución óptima. Cuando estas metaheurísticas, en un determinado *espacio de búsqueda*, están diseñadas para optimizar un solo objetivo se habla de *metaheurísticas mono-objetivo*, mientras que si están diseñadas para optimizar varios objetivos al mismo tiempo se habla de *metaheurísticas multi-objetivo*. Esto es lo que conocemos por *Computación Evolutiva* (ya sea en su versión mono-objetivo o multi-objetivo), que recurre al uso de metaheurísticas para buscar soluciones a los problemas de una manera poco convencional y tomando como referencia los postulados de la evolución biológica. De su aplicación al campo de la robótica surge lo que conocemos por *Robótica Evolutiva* (en sus versiones mono-objetivo o multi-objetivo).

Esta tesis doctoral aborda, de manera innovadora y altamente para-

lela, el problema del PP sobre mapas reales extensos en un contexto de tiempo real. Este grado de paralelismo se consigue gracias al uso intensivo de las populares Unidades Gráficas de Procesamiento (GPUs) y los bien conocidos *chips multi-core*. Pero aquí no solo se aborda el problema del PP desde un punto de vista altamente paralelo sino que, de manera transversal, también se aborda desde un punto de vista inteligente aplicando *metaheurísticas*.

1.2 Motivación

La robótica constituye, probablemente, la mayor revolución tecnológica del siglo XX. Una revolución que se materializa cada día gracias al esfuerzo de la comunidad científica. Como resultado, es posible encontrar robots donde hace años creíamos imposible. En sus comienzos, la robótica solo tenía sentido en el sector industrial donde el concepto de robot estaba ligado a una máquina estática y poco inteligente que realizaba solo unos pocos trabajos de manera muy precisa. Sin embargo, hoy en día, los robots se utilizan en entornos muy variopintos, como son la medicina o el sector doméstico. Además, los robots han adquirido nuevas capacidades, entre las que cabe destacar la inteligencia y la movilidad. Estas capacidades son las claves del éxito y las que han convertido a los robots en las máquinas que hoy conocemos.

En cuanto a movilidad, los robots se enfrentan a problemas muy complejos, en terminos computacionales, la mayoría de ellos enmarcados en la categoría de problemas *NP-hard* y *NP-completos*. Este tipo de problemas tienen la particularidad de que el tiempo de cálculo necesario para encontrar la solución óptima crece de manera exponencial conforme se aumenta el tamaño de los datos de entrada. Resulta evidente que no es una buena idea abordar este tipo de problemas recurriendo a técnicas de programacion tradicionales. En su lugar se aplican técnicas de programación basadas en *metaheurísticas*.

Los algoritmos basados en metaheurísticas, siempre y cuando estén bien diseñados, generan una o varias soluciones aceptables para un problema dentro de unos márgenes de tiempo razonables pero sin garantizar

la solución óptima. Generalmente, este tipo de algoritmos se aplican a problemas de *optimización combinatoria*. La optimización combinatoria tiene como objetivo encontrar un objeto matemático que maximice o minimice una función determinada. Estos objetos matemáticos se denominan *estados*, el conjunto de todos los estados candidatos se conoce como el *espacio de búsqueda* del problema y la función a optimizar recibe el nombre de *función de fitness* o *función objetivo* (es importante tener en cuenta que estos conceptos pueden tener asociados distintos términos en función del área de aplicación). Es posible diseñar la función objetivo para optimizar más de un parámetro al mismo tiempo. Esto es lo que marca la diferencia a la hora de hablar de *metaheurística mono-objetivo* o *metaheurísticas multi-objetivo*. Por lo tanto, los algoritmos basados en metaheurśiticas, cuando son aplicados a problemas NP-hard o NP-completos, tienen como objetivo encontrar el subconjunto de los estados del problema que mejor se ajustan a una determinada función de fitness. Además, los algoritmos basados en metaheurísticas son candidatos perfectos para ser abordados utilizando técnicas de *computación paralela*.

La computación paralela tiene como objetivo reducir el tiempo de procesamiento requerido para resolver un problema determinado. Esto se consigue dividiendo un problema base en un conjunto de subproblemas que pueden ser procesados de manera concurrente. La ejecución de cada uno de estos subproblemas se denomina *hilo de ejecución*. Es posible la coexistencia simultánea de varios hilos de ejecución gracias al uso, cada vez más habitual, de las Unidades Centrales de Procesamiento (CPUs) *multi-core*. Cada uno de los cores que componen la CPU puede ejecutar uno o, en el caso de los procesadores con *HyperThreading*, dos hilos de ejecución. Por ejemplo, una CPU multi-core compuesta por 4 cores con HyperThreading podría ejecutar hasta 8 hilos de ejecución de manera simultánea. Como se puede deducir, la computación paralela nos permite explorar el espacio de búsqueda de un problema en un menor tiempo. Hoy en día también es habitual escuchar el concepto de *computación altamente paralela*. La computación altamente paralela extiende la programación paralela al utilizar un número masivo de hilos de ejecución. Dejando a un lado los rack de computación y los supercomputadores, la GPU y su creciente uso y popularidad es lo que ha hecho posible el uso de técni-

cas de computación altamente paralela embarcada directamente en los robots. Podemos hablar de la GPU como un conjunto de cores (basados principalmente en multiplicadores y no en cores completos como los de los chips multi-core) de cardinalidad considerable y que pueden ser utilizados para abordar tareas altamente paralelas desde el punto de vista de la *computación de propósito general*. Esto es la Computación de Propósito General en Unidades Gráficas de Procesamiento (GPGPU).

En robótica, podemos clasificar los robots en dos grupos bien diferenciados: los robots móviles y los robots no móviles. A diferencia de los segundos, los primeros están dotados de un mecanismo de locomoción que les permite moverse en el espacio. Y a la hora de desplazarse surge uno de los problemas de mayor interés científico, el problema del PP. Resolver este problema implica encontrar un camino válido que permita a un robot llegar de un punto a otro del espacio sin comprometer su integridad física. Para solucionar el problema del PP es necesario disponer de:

- Una representación computacional del entorno del robot.
- El punto de comienzo del camino a calcular.
- El punto destino del camino a calcular.

A la salida del algoritmo se obtendrá, siempre y cuando sea posible, el camino que une los puntos origen y destino (ver figura 1.1).

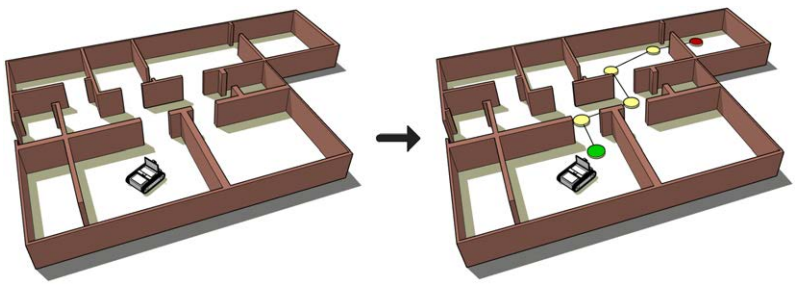


Figura 1.1: Ejemplo de Path Planning.

Más en detalle, el PP pertenece a la categoría de problemas *NP-hard*. Como se ha comentado más arriba, los problemas NP-hard implican para

su resolución un tiempo de cómputo que aumenta exponencialmente a medida que crece el tamaño del espacio de búsqueda de las soluciones. Por este motivo, y teniendo en cuenta que el PP es un problema de optimización combinatoria, es un candidato perfecto para ser solucionado mediante la aplicación de metaheurísticas.

A la hora de calcular el problema del PP mediante el uso de metaheurísticas, uno de los objetivos más frecuentemente utilizados es el correspondiente a la *longitud del camino*, siendo el mejor el más corto. Suponiendo que el camino obtenido está representado por una lista ordenada de coordenadas del espacio, *(x, y)*, y que dos puntos consecutivos forman un tramo del camino, la longitud del camino puede calcularse como la suma de las distancias de cada tramo del camino (ver ecuación 1.1).

$$Longitud_{camino} = \sum_{i=1}^{n} D(T_i) \tag{1.1}$$

En la ecuación 1.1, n es el número de tramos del camino y $D(T_i)$ se refiere a la longitud del i-ésimo tramo del camino. Este objetivo está, por lo general, directamente relacionado con el tiempo de operación del robot, ya que un camino más corto será, supestamente, recorrido por el robot en un menor tiempo.

Cuando se trabaja con metaheurísticas multi-objetivo, se optimizan algunos objetivos extra, como son la *seguridad* y la *suavidad del camino*. Contemplar la seguridad del camino consiste en minimizar el número de colisiones posibles con los obstáculos del entorno. Es posible que un camino en el que se produzca alguna colisión con algún obstáculo del entorno no sea totalmente erróneo. Si el robot choca con un objeto pequeño, lo más probable es que tras la colisión el obstáculo pase a ocupar una posición diferente en el espacio y el robot no sufra ningún daño. En cambio, si el robot choca con una pared probablemente éste resulte dañado. Por esta razón, es interesante asignar un peso (en cuanto a ocupación se refiere) a las zonas del entorno dependiendo del peligro potencial que suponen para el robot. Cuando en una zona del espacio no aparezca ningún objeto el porcentaje de ocupación asignado será del 0 %, mientras que si está totalmente ocupada por un obstáculo el porcentaje de ocupación será del 100%. Así, el porcentaje de ocupación de una zona del espacio semiocu-

pada por un obstáculo estará comprendido ente el 0% y el 100%. De esta forma, la seguridad del camino puede evaluarse como la suma de los pesos de las zonas del espacio que tiene que atravesar el robot para desplazarse hasta el punto de destino (ver ecuación 1.2). No obstante, hay que tener en cuenta que considerar la seguridad de un camino solo tiene sentido siempre y cuando se disponga de un conocimiento a priori del entorno y de sus obstáculos. Debido a que este no es ni mucho menos el tema principal de esta tesis doctoral, asumiremos que este conocimiento, propio de otra línea de investigación, será proporcionado por un agente externo.

$$Seguridad_{camino} = \sum_{i=1}^{n} W(Z_i) \tag{1.2}$$

En la ecuación 1.2, n es el número de zonas del entorno que el robot atraviesa al recorrer un camino determinado y $W(Z_i)$ es el peso asociado a la zona i-ésima del espacio.

También resulta interesante analizar la suavidad del camino. Este aspecto tiene como objetivo conseguir caminos que sean lo menos tortuosos posible, es decir, sin giros bruscos y con el mínimo número de giros. En aquellas aplicaciones donde prime el factor energético este objetivo puede resultar clave, ya que está directamente relacionado con el consumo de energía del robot. Los robots no están dotados de una fuente de energía infinita, por lo que realizar acciones minimizando el consumo de energía puede ser vital en algunos ámbitos de aplicación. Así, un robot que recorra un camino que implique pocos giros o giros menos bruscos tendrá un consumo de energía más bajo. Para calcular la suavidad basta con sumar el valor del menor ángulo formado entre cada dos tramos consecutivos del camino en cuestión (ver ecuación 1.3).

$$Suavidad_{camino} = \sum_{i=1}^{n-1} min(\alpha(Si, S_{i+1}), \beta(Si, S_{i+1})) \tag{1.3}$$

En la ecuación 1.3, n es el número total de tramos del camino y $min(\alpha(Si, S_{i+1}),\ \beta(Si, S_{i+1}))$ es el menor de los ángulos formados entre los tramos i e i+1 del camino.

Teniendo en cuenta el estado del arte, la mayoría de los autores afrontan el problema del PP desde desde una perspectiva mono-objetivo y apli-

cando algoritmos, basados o no en metaheurísticas, de carácter puramente secuencial. Últimamente también se pueden encontrar vaias propuestas que abordan el problema del PP desde una perspectiva multi-objetivo basada en técnicas propias de la *Computación Evolutiva multi-objetivo*. La computación evolutiva multi-objetivo es una rama de la IA que resuelve problemas de optimización combinatoria tomando como base los postulados de la evolución biológica y optimizando más de un objetivo al mismo tiempo. Cuando este tipo de computación se aplica al campo de estudio de la robótica surge lo que se conoce como *Robótica Evolutiva Multi-Objetivo*. Por lo tanto, La Robótica Evolutiva Multi-Objetivo es un área de la robótica autónoma en la que los controladores de los robots se desarrollan tomando como base la teoría de la evolución biológica al aplicar algoritmos evolutivos multi-objetivo.

El uso de técnicas multi-objetivo presenta la gran desventaja de que la generación y evaluación de las soluciones candidatas precisan mayores tiempos de cómputo frente a las aproximaciones mono-objetivo. Teniendo esto en cuenta, este tipo de técnicas, ya sean mono-objetivo o multi-objetivo, pueden verse beneficiadas notablemente al emplear técnicas de *computación paralela*. Así, en el campo de la robótica, la computación paralela persigue los siguientes objetivos:

- Acelerar los tiempos de cómputo mediante el aprovechamiento de las arquitecturas hardware actuales (CPU multi-core, GPU y otras).

- Obtener mejores soluciones en el mismo intervalo de tiempo mediante equipos de algoritmos paralelos para realizar búsquedas exhaustivas en el espacio de soluciones.

Obtener resultados con rapidez es un requisito crítico en cierto tipo de aplicaciones, en las que es necesario suministrar un resultado coherente en un margen de tiempo más que razonable. Un ejemplo claro lo encontramos en el sector seguridad, donde el tiempo necesario para tomar una decisión es un factor determinante en cuanto al éxito o fracaso de una determinada operación. En robótica, por lo general, estas restricciones temporales toman un papel determinante. Queremos que los robots hagan cada vez cosas más complejas, cada vez de manera más independiente

y cada vez más rápido. Un robot que tarda horas o días en planificar una determinada tarea carece de utilidad.

Por lo tanto, la principal motivación de esta tesis doctoral es la búsqueda de nuevas técnicas que permitan resolver el problema del PP sobre mapas reales extensos y con unas restricciones temporales muy severas (requerimientos de tiempo real). Para satisfacer estos requerimientos de tiempo se recurre a la aplicación combinada de dos tipos de paralelismo. Por una parte, un *paralelismo de grano grueso* que explota las capacidades de las CPU *multi-core* y, por otra parte, un *paralelismo de grano fino* que aprovecha los recursos que ponen a nuestra disposición las populares y potentes GPU usando técnicas basadas en GPGPU. Además, en este trabajo, no solo se profundiza en cómo satisfacer las restricciones temporales sino que también se estudia, desde el punto de vista de las metaheurísticas multi-objetivo, cómo obtener caminos de calidad que garanticen la seguridad y al mismo tiempo cumplan con los principios de eficiencia energética, de manera que ayuden a incrementar la durabilidad de las fuentes de energía, en este caso baterías. Desde este punto de vista, se aborda el problema del PP mediante la aplicación de Algoritmos Evolutivos (AEs). Por último, se hará una rigurosa y justa comparación, desde el punto de vista de la robótica móvil, de los resultados obtenidos de aplicar todas las técnicas estudiadas en esta tesis doctoral.

1.3 Quad-RRT

Quad-RRT es un algoritmo masivamente paralelo para el cálculo de trayectorias sobre mapas reales de gran escala, abarcando áreas del orden de varios kilómetros cuadrados (km^2). Quad-RRT implementa un procedimiento metaheurístico que combina dos niveles de paralelismo que se corresponden con un *paralelismo de grano grueso* y un *paralelismo de grano fino*.

El primer nivel de paralismo o *paralelismo de grano grueso* se realiza en un *chip* de tipo *multi-core* (CPU multi-core). Este primer nivel obtiene la información disponible del entorno (información no estructurada) y la procesa de manera paralela para, finalmente, obtener una represantación estructurada de la información del entorno en forma de mapa de dos

dimensiones (2D). El número de *hilos de ejecución* generados para completar esta tarea es configurable, siendo en este caso el máximo número de hilos que puede ejecutar de manera simultánea (sin realizar cambios de contexto) el hardware disponible. En este caso un chip compuesto por 4 cores con tecnología *HyperThreading*, donde cada core puede ejecutar 2 hilos de ejecución de manera simultánea, lo que permite un máximo de 8 hilos de ejecución simultáneos.

Una vez se dispone de información estructurada del entorno, quad-RRT aplica un *algoritmo de particionado* inteligente que, en función de las coordenadas origen y destino de la ruta deseada, divide el mapa en diferentes *sectores* (cuatro en esta versión del algoritmo) que serán procesados en una etapa posterior (ver figura 1.2). Dos de estos sectores comparten la coordenada origen y los otros dos comparten la coordenada destino de la ruta deseada. Estos sectores componen el conjunto de datos de entrada para el segundo nivel de paralelismo.

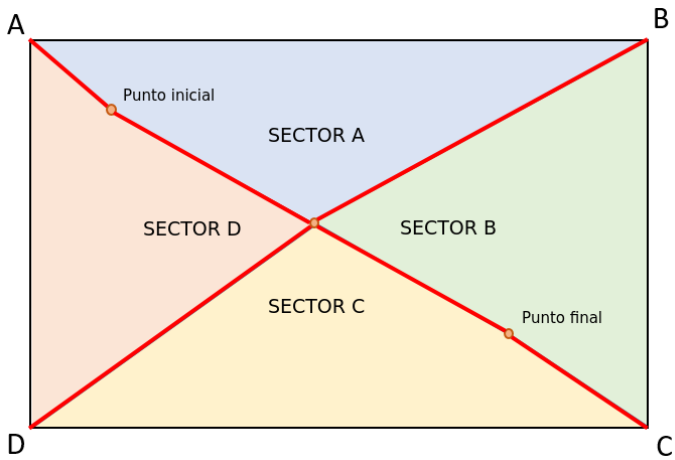


Figura 1.2: Partición del mapa.

El segundo nivel de paralelismo o *paralelismo de grano fino* implementa un software masivamente paralelo (utiliza un gran número de hilos de ejecución simultáneos) que se ejecuta sobre una GPU. A groso modo, la GPU estructura el paralelismo de datos en *bloques* e *hilos* de GPU. Un bloque de GPU puede verse como un conjunto de hilos de ejecución. Una GPU

moderna puede albergar decenas o cientos de bloques, y estos a su vez albergar cientos o miles de hilos de ejecución. De aquí el nombre de *paralelismo masivo de datos*. Tanto los hilos como los bloques de GPU pueden organizarse en *matrices* de GPU de varias dimensiones (ver figura 1.3). Pues bien, el paralelismo de grano grueso llevado a cabo por el algoritmo quad-RRT procesa cada sector de datos en un bloque de GPU, donde los cálculos se llevan a cabo con varios hilos de ejecución. quad-RRT implementa el agoritmo Rapidly-exploring Random Tree (RRT), el cual se realiza de manera masivamente paralela en cada bloque de GPU. Cada instancia del algoritmo RRT explora una sección del entorno durante un tiempo o un número de ejecuciones determinado de forma simultánea, es decir, los árboles de exploración crecen de manera muy rápida (del orden de milisegundos) sobre el mapa (ver figura 1.4a). Cuando el algoritmo RRT en cada bloque finaliza su ejecución se aprovechan de nuevo los recursos que nos brinda la GPU para realizar un proceso de matching altamente paralelo entre los árboles para, finalmente, determinar las posibles rutas entre los puntos extremos de la ruta deseada y seleccionar la más adecuada (ver figura 1.4b).

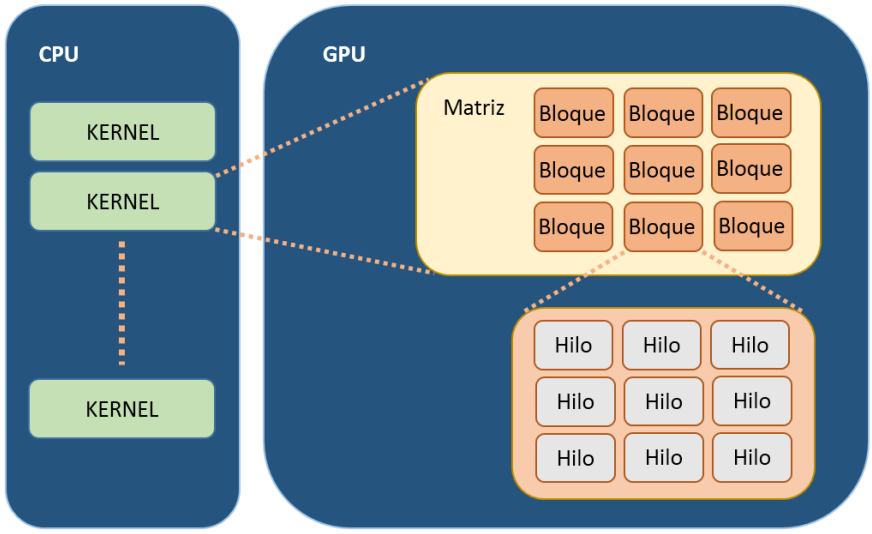
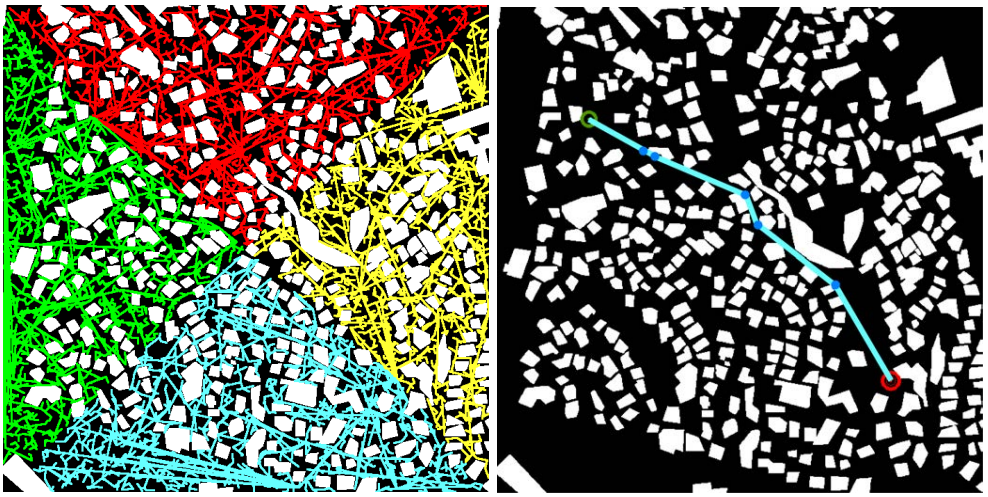


Figura 1.3: Modelo de computación típico de una GPU NVidia.



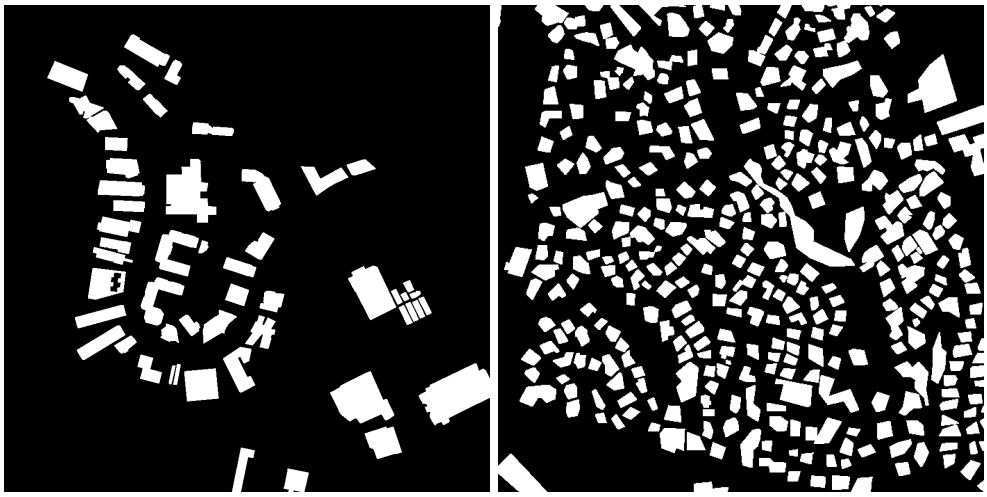
(a) Árboles de exploración RRT. (b) Trayectoria calculada.

Figura 1.4: Árboles de exploración y trayectoria calculados por el algoritmo quad-RRT.

1.4 Resultados y conclusiones

Para demostrar que el algoritmo propuesto, quad-RRT, destaca sobre las propuestas presentes en la literatura se ha sometido a una serie de pruebas exhaustivas con el objetivo de determinar su *eficiencia*, *eficacia* y *calidad de las soluciones* obtenidas cuando se aplica para resolver trayectorias sobre mapas reales de gran escala. Para estas pruebas se ha diseñado un conjunto de escenarios sobre mapas reales. Para determinar de una manera generalista el comportamiento y respuesta del algoritmo, estos escenarios contemplan el peor y el mejor de los escenarios posibles, esto es, un mapa con una densidad de objetos baja o normal y un mapa con una densidad de objetos muy alta (ver figura 1.5). En definitiva, los escenarios estan formados por estos mapas y en diferentes tamaños.

Como ya se ha explicado antes (ver sección 1.3), quad-RRT es un algoritmo masivamente paralelo que aprovecha las capacidades de la GPU. Es importante tener en cuenta que la relación entre el número de tareas simultáneas y el tiempo de ejecución del algoritmo no es siempre inver-



(a) Mapa poco denso. (b) Mapa muy denso.

Figura 1.5: Mapas utilizados para la definición de los escenarios de prueba del algoritmo quad-RRT.

samente proporcional. Es decir, aumentar el número de tareas paralelas no implica reducir el tiempo de ejecución del algoritmo. Esto es el *limite de aceleración* de los algoritmos paralelos y se debe a los retardos temporales introducidos a causa de la *sincronización* y las *colisiones* entre *hilos de ejecución*. Para determinar la mejor configuración (número de tareas paralelas) posible y de una manera generalista (que sirva para mapas de cualquier densidad), el algoritmo quad-RRT se ha ejecutado sobre los escenarios propuestos y posteriormente se han analizado los resultados obtenidos.

Estas pruebas determinan que los mejores resultados se obtienen cuando se ejecuta el algoritmo quad-RRT utilizando 25 hilos de ejecución por cada bloque de GPU. La tabla 1.1 muestra las configuraciones propuestas para el algoritmo y las tablas 1.2 y 1.3 muestran los tiempos de ejecución medios obtenidos al utilizar cada una de estas configuraciones.

Como se puede observar, la tabla 1.3 no contiene información para las configuraciones $L1$, $L2$ y $L3$, que sí aparecen en la tabla 1.2. Esto se debe a que el algoritmo quad-RRT no encuentra solución en alguno

Configuración	Bloques de GPU	Hilos por bloque de GPU
L1	4	1
L2	4	5
L3	4	10
L4	4	**25**
L5	4	50
L6	4	75

Tabla 1.1: Configuraciones propuestas para el algoritmo quad-RRT.

Configuración	Tiempo medio de ejecución medio (msec)
L1	5079.19
L2	5046.88
L3	4931.87
L4	**5143.68**
L5	5279.25
L6	5629.18

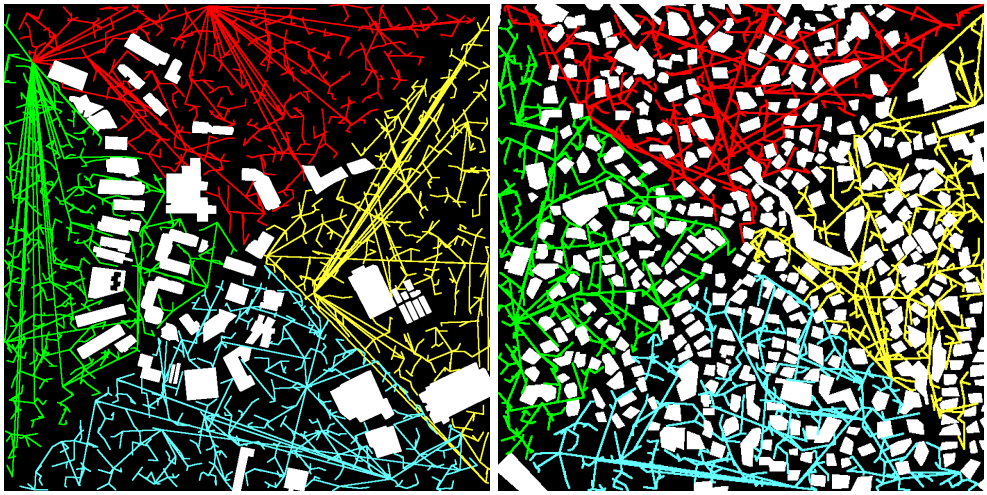
Tabla 1.2: Tiempos de ejecución medios obtenidos al ejecutar el algoritmo quad-RRT sobre un mapa con una densidad normal de objetos.

Configuración	Tiempo medio de ejecución medio (msec)
L4	**372**
L5	515.375
L6	711.375

Tabla 1.3: Tiempos de ejecución medios obtenidos al ejecutar el algoritmo quad-RRT sobre un mapa con una densidad muy alta de objetos.

o en ninguno de los escenarios propuestos cuando se utilizan estas configuraciones, por lo que no se consideran configuraciones válidas. De las configuraciones restantes ($L4$, $L5$ y $L6$), $L4$ es la configuración que obtiene mejor tiempo de ejecución medio para cualesquiera de los escenarios propuestos, por lo que queda demostrado que usar 4 bloques de GPU y 25 hilos de ejecución es la mejor configuración para el algoritmo quad-RRT.

Los árboles de expansión generados por cada una de las instancias paralelas del algoritmo RRT utilizando la configuración $L4$ se expanden de manera uniforme por el espacio libre de obstáculos, es decir, exploran cada hueco del mapa por igual (ver figura 1.6).



(a) Expansión de los árboles RRT utilizando la configuración $L4$ sobre un mapa con una densidad normal de objetos.

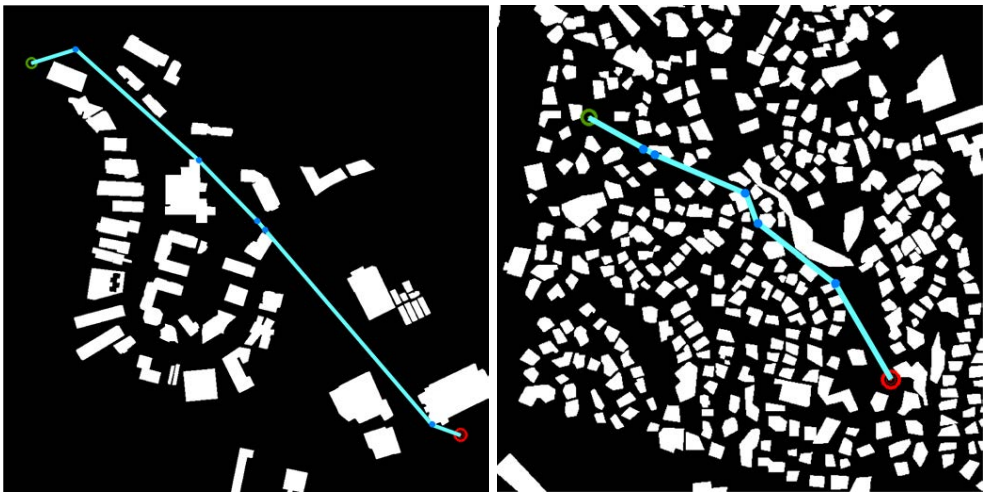
(b) Expansión de los árboles RRT utilizando la configuración $L4$ sobre un mapa con una densidad muy alta de objetos.

Figura 1.6: Expansión de los árboles RRT utilizando la configuración $L4$.

Visualmente y en cuanto a la calidad de las soluciones obtenidas, estas resultan bastante acertadas (ver figura 1.7) ya que un experto cualquiera podría catalogarlas como óptimas o como bastante exactas, por ser casi óptimas.

Finalmente, el algoritmo quad-RRT se ha comparado con otros métodos de cálculo de trayectorias existentes en la literatura con el objetivo de demostrar las ventajas que aporta su uso. Esta comparativa demuestra que usar quad-RRT produce resultados entre 9.98 y 206.86 veces más rápido (ver tabla 1.4). Además, por el número medio de nodos generados para los árboles RRT también se demuestra que el algoritmo propuesto realiza una exploración profunda del mapa, lo que aumenta significativamente la probabilidad de encontrar una ruta válida.

Todo esto demuestra que el algoritmo propuesto, quad-RRT, es un



(a) Resultado obtenido al utilizar quad-RRT con la configuración $L4$ sobre un mapa con una densidad normal de objetos.

(b) Resultado obtenido al utilizar quad-RRT con la configuración $L4$ sobre un mapa con una densidad muy alta de objetos.

Figura 1.7: Resultados obtenidos al utilizar el algoritmo quad-RRT con la configuración $L4$.

Algoritmo	Tiempo medio de ejecución (msec)	Número medio de nodos RRT
RRT	81918	5875
Biased-RRT	59702	2606
Greedy-RRT	39499	7336
BiasedGreedy-RRT	28412	3123
Topological-RRT	3956	347
quad-RRT	**396**	**2652**

Tabla 1.4: Comparativa de rendimiento según los resultados publicados en [3]

candidato excelente a la hora de resolver trayectorias sobre mapas reales de gran escala en applicaciones con requerimientos de tiempo real.

CHAPTER

2

Contribuciones

2.1 Contribuciones a la computación paralela

En esta tesis doctoral se desarrollan algoritmos paralelos que se ejecutan sobre *chips multi-core* (CPUs multi-core) y algoritmos masivamente paralelos que se ejecutan sobre la GPU. Ambos paradigmas, esto es la *computación multi-core* y la computación GPGPU, se combinan para dar solución a problemas muy complejos (problemas *NP-completos* o *NP-hard*) en aplicaciones con requerimientos de *tiempo real*. En este tipo de aplicaciones es más importante generar un resultado válido en un tiempo de computación razonable más que emplear un tiempo de computación excesivo para obtener el resultado óptimo. La aportación a la computación paralela en esta tesis va más allá que el hecho de combinar distintintas técnicas de computación paralela con el objetivo de reducir los tiempos de cómputo, sino que también contribuye en la forma en la que ciertos algoritmos clásicos se dividen en subtareas para, posteriormente, abordarlas de forma simultánea y de la forma más eficiente posible. Además, se intenta que las soluciones paralelas aportadas no deterioren la calidad de las soluciones obtenidas, sino todo lo contrario, es decir, buscando también incrementar la calidad de las soluciones finales.

2.2 Contribuciones a la robótica móvil

En cuanto a la aportación al campo de la robótica móvil, esta tesis doctoral propone un metodo innovador, eficaz y eficiente para resolver el problema del PP sobre mapas reales de gran escala. El problema del PP es un problema que se encuentra constantemente bajo estudio por tres motivos fundamentales:

1. Se trata de un problema *NP-completo*. Este tipo de problemas se caracterizan por ser los problemas más difíciles de resolver y donde el tiempo de computación necesario para encontrar una solución crece exponencialmente a medida que el tamaño de los datos de entrada crece. Resumiendo, mapas más grandes implican mayor tiempo de computación.
2. Es un problema fundamental y necesario de resolver para permitir que los robots móviles se desplazen de manera segura, es decir, sin colisionar con los obstáculos en el entorno.
3. Los algoritmos existentes no dan soluciones válidas en tiempos razonables cuando los mapas son excesivamente grandes.

Esto hace que el problema del PP suponga un reto científico constante. Por tanto, esta tesis aporta al campo de la robótica móvil un enfoque algorítmico capaz de solucionar el problema del PP de manera eficiente en aplicaciones con requerimientos de tiempo real.

2.3 Publicaciones

- **Alejandro Hidalgo-Paniagua**, Juan Pedro Bandera, Manuel Ruiz-de-Quintanilla, Antonio Bandera. **Quad-RRT: a real-time GPU-based global path planner in large-scale real environments**. Expert Systems With Applications 99: 141-154 (2018)
- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Joaquin Ferruz, Nieves Pavón: **Solving the multi-objective path planning problem in mobile robotics with a firefly-based approach**. Soft Computing 21(4): 949-964 (2017)

- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Joaquín Ferruz Melero: **Applying the MOVNS (multi-objective variable neighborhood search) algorithm to solve the path planning problem in mobile robotics**. Expert Systems With Applications 58: 20-35 (2016)
- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Joaquín Ferruz Melero, Nieves Pavón: **MOSFLA-MRPP: Multi-Objective Shuffled Frog-Leaping Algorithm applied to Mobile Robot Path Planning**. Engineering Applications of Artificial Intelligence 44: 123-136 (2015)
- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Nieves Pavón, Joaquín Ferruz Melero: **A Comparative Study of Software filters Applied as a Previous Step of the ICP Algorithm in robot Location**. Journal of Circuits, Systems, and Computers 23(8) (2014)

2.3.1 Otras publicaciones

Además de las anteriores (ver seccion 2.3), el tesitando también ha participado en publicaciones no directamente relacionadas con la robótica autónoma, aunque sí en el ámbito de la paralelización de algoritmos. Estas publicaciones son las siguientes:

- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Nieves Pavón, Joaquín Ferruz Melero: **A Comparative Study of Parallel RANSAC Implementations in 3D Space**. International Journal of Parallel Programming 43(5): 703-720 (2015)
- **Alejandro Hidalgo-Paniagua**, Miguel A. Vega-Rodríguez, Nieves Pavón, Joaquín Ferruz Melero: **A comparative study of parallel software SURF implementations**. Concurrency and Computation: Practice and Experience 26(17): 2758-2771 (2014)

2.3.2 Publicaciones en congresos

El tesitando también ha participado en varios congresos de ámbito nacional e internacional. La temática de los congresos es variopinta, sin

embargo, todos, en mayor o menor medida, tienen alguna relación directa con el mundo de la robótica, la visión por computador y la inteligencia artificial. Estas publicaciones son las siguientes:

- **Alejandro Hidalgo-Paniagua**, José Mateos Sánchez, Pedro Nuñez Trujillo, Pablo Bustos, José Moreno del Pozo: **LearnBot: A Robotics-Based Prototype for Educational Environments**. Workshop Hispano-Brasileiro in Autonomous Robots and Robotics Intelligence, Cáceres (España), (2013)

- Juan M. Muñoz Lobato, **Alejandro Hidalgo-Paniagua**, Santiago Salamanca Miño, Pilar Merchán Gracía: **Diseño de software para el control de dispositivos en aplicaciones de visión por computador**. XXXIII Jornadas de Automática, Vigo (España), (2012)

- Pablo Manzano Gómez, **Alejandro Hidalgo-Paniagua**, José Moreno del Pozo: **Augmented Reality Books: software educational tool for surgeons in training**. Congreso Anual de la Sociedad Española de Ingeniería Biomédica, Cáceres (España), (2011)

2.4 Organización de congresos

Hay que destacar el papel del tesitando en la organización de congresos de ámbito nacional e internacional. Los roles que el tesitando ha desempeñado en este aspecto son los siguientes:

- **Multicore and GPU Programming (PPMG2015)** *Comité organizador.* Cáceres (España), (2015)

- **PBio 2013: International Workshop on Parallelism in Bioinformatics (EuroMPI2013)** *Comité de programa.* Madrid (España), (2013)

- **2nd International Conference on the Theory and Practice of Natural Computing (TPNC2013)** *Comité organizador.* Cáceres (España), (2013)

- **XXXIV Jornadas de Automática** *Colaborador del Grupo de Visión por Computador del Comité Español de Automática.* Barcelona (España), (2013)

2.5 Estructura de la Tesis

Esta tesis doctoral se estructura en dos partes. La primera parte contiene un resumen en castellano de la descripción de la tesis. La segunda parte, escrita en inglés, describe la tesis. La descripción de la tesis se desarrolla a lo largo de cinco capítulos, los cuales a su vez se organizan en varias secciones y subsecciones.

El capítulo uno introduce de manera pausada tanto la temática de la tesis como la motivación que ha llevado a realizar esta investigación.

El capítulo dos explica el problema objetivo de este trabajo de investigación (el problema del PP), las técnicas y algoritmos existentes en la literatura y cómo estas técnicas y algorimos modelan el entorno y codifican las soluciones (tanto las soluciones parciales como las soluciones finales) para abordar el problema y aproximarse a la solución deseada.

En el capítulo tres se explica con detalle el algorimo propuesto para resolver el problema del PP en aplicaciones con requerimientos de tiempo real que utilizan mapas reales extensos, se demuestra matemáticamente su completitud y, finalmente, se realiza un primer análisis gráfico sobre los resultados obtenidos en distintas ejecuciones del algoritmo.

En el capítulo cuatro se presentan varios escenarios de ejecución, se analizan con detalle los resultados obtenidos y se determina de manera generalista la mejor configuración paralela del algoritmo. Además, en este capítulo, se realiza un exhaustivo análisis gráfico de los resultados obtenidos a través de la batería de pruebas diseñada y se realiza una comparativa con los resultados obtenidos por otros algoritmos existentes en la literatura.

El capítulo cinco recoge las conclusiones obtenidas de este trabajo de investigación y plantea cuestiones y propuestas futuras que podrían mejorar los resultados actuales.

Finalmente un capítulo de glosario, que recoge todos los acrónimos que aparecen a lo largo de este documento, y un capítulo bibliográfico, que

contiene todas las referencias y citas que soportan este trabajo, marcan el cierre de este trabajo de investigación.

PART 2

Thesis

CHAPTER

1 Introduction

"Creating an artificial being has been the dream of man since the science was born."
— Steven Spielberg, Artificial Intelligence: A.I (2001)

1.1 Introduction

A world populated by robots. Machines performing a variety of tasks of diverse nature, going from very specific jobs in assembly lines, to cooperation with people in daily life tasks. But in order to adapt to different tasks and contexts, and work in the dynamic, unpredictable world in which we live, these machines need to be provided with a certain level of intelligence. Hence, machines become robots.

People want machines to be intelligent, and mobile, to use them to dominate, but also to free themselves, of an environment that becomes more stressful day by day. A growing set of *"cybernetic servants"* is becoming available to free people from different tasks. A good example of these robots are the autonomous vacuum cleaners. These robots, despite having a very limited functionality, can be set up to periodically perform cleaning rounds, so the person do not need to worry about that task at

all. Although this example is too simple, it is representative enough to depict a context that will most probably be the future trend (see [4] and [5]).

Robotics is among the most important technological revolution of the 20^{th} century. But it is also unquestionable that this revolution has only just begun. New scientific advancements are accomplished each day thanks to the research activities in this area. This advancements move us to believe that more complex, more intelligent and more versatile machines will be soon available [6]. These thoughts, to date, have been quickly verified with new products, results and proposals [7].

There are two irrefutable aspects that define the versatility of a machine: its intelligence and its motion ability. These aspects are closely related and it has no sense (in most of the cases) to study deeply the machine's mobility without taking into account the intelligence aspect. A good example of the relation between intelligence and mobility is the trial and error procedure: living beings trying to accomplish a certain goal in an unknown environment, may find a certain solution by actively exploring this environment. Then, they can refine and improve the final solution in a constant process, taking into account their own experience.

In the field of robotics, mobile agents have to face a key problem: how and where a robot should move to accomplish a goal without damaging itself. As the meaning of *moving* can be very ambiguous, in this proposal we will refer to displacement, in the literal sense of traveling. Two key questions arise, regarding traveling: Where are we going? and which is the best alternative to go there? In robotics, these difficulties are known as the Path Planning (PP) problem (see [8]).

when traveling, another important question to answer is the following: how long will it take to go there? Hence, time is also a parameter to keep in mind in the PP problem: Whenever we travel we want to do it as fast as possible. This condition can be more o less strict but, in some contexts, it becomes a hard constraint. For example, a robot working as a component of a security system, should move fast in an alert event, without taking much care about the quality of the calculated path or the elegance of the solution. Here, response time becomes the most important parameter of the mission.

Another important aspect related with the robot mobility is the robot intelligence. Nowadays it is common to hear about the AI concept (see [5]). In this context, a robot should not only move, but do it in a intelligent way. Biologically inspired solutions may be helpful achieving these smart movements. The analysis of how the living beings behave, what patterns do they follow, how they are organized, how they interact each others and how they discard some potentially right solutions to subsequently mimic them can provide interesting clues to design robotic behaviours. Leaving aside the cognitive theories, one of the most promising fields of the AI is the *Evolutionary Computation*, in which *metaheuristic-based algorithms* are used. These algorithms are characterized because they are usually able of generating right solutions to problems, although they do not ensure an optimal result. In an specific *search space*, when these metaheuristics are designed to optimize only one objective, they are known as *single-objective metaheuristics*. Conversely, when these type of algorithms are designed to optimize several objectives at the same time, they are known as *multi-objective metaheuristics*. The use of the evolutionary computation in robotics leads to the concept of *Evolutionary Robotics*, that can be employed in both its single-objective or multi-objective modalities.

This doctoral dissertation addresses, in an innovative and highly parallel way, the PP problem in large-scale real maps and in a real time context. The General-Purpose Computing on Graphics Processing Units (GPGPU) and the intensive usage of the GPU, together with the well-known *multi-core chips*, lead us to achieve a high degree of parallelism. But this doctoral dissertation not only addresses the PP problem from the point of view of the parallel computation. It also analyzes and proposes solutions to the PP problem from the point of view of the AI, by applying metaheuristic-based techniques to find the path from the origin to the goal.

1.2 Motivation

Robotics probably constitutes the most important technological revolution of the 20^{th} century. A revolution that becomes stronger each day

thanks to the efforts of the scientific community. As a result, it is possible to find robots working in tasks and locations that seemed impossible just some years ago. Hence, at the beginning robotics only made sense in the industrial sector, where the concept of robot was bound to a static and not much intelligent machine. These industrial robots only carry out a few jobs in a very precise way. However, nowadays the robots are used in a large number of different areas, like medicine or in the home environment. Furthermore, robots have acquired new capabilities regarding both mobility and intelligence. These capabilities are the key to success and they have turned the robots into the machines that we currently know.

Referring to the mobility aspect, robots are dealing with highly complex problems, in terms of computation. These problems belongs, in most of the cases, to the *NP-hard* and *NP-complete* categories. These type of problems have the peculiarity that the computation time needed to optimally solve them exponentially grows, as the size of the input data increases. It is obvious that it is not a good idea to try to solve these type of problems using traditional computation techniques. Instead *metaheuristic-based techniques* are applied.

The metaheuristic-based algorithms, as long as they are well designed, generate at least one suitable solution to an specific problem and within reasonable margins of time, but without guaranteeing its optimality. These type of algorithms are usually applied to *combinatorial optimization* problems. The combinatorial optimization aims finding a mathematical object that maximizes or minimizes a specific function. These mathematical objects are known as *states*. The set of all the candidate states as the problem *search space*, and the function to be optimized as the *fitness function* or the *objective function* (it is important to take into account that these concepts can be bounded to different terms depending on the application area). It is possible to design the fitness function to optimize more than one objective at the same time. This is the main difference between *single-objective metaheuristics* and *multi-objective metaheuristics*. Therefore, metaheuristic-based algorithms applied to NP-hard or NP-complete problems try finding the set of the problem's states that better fits to a specific fitness function.

One of the advantages of metaheuristic-based algorithms is that they

are perfectly suitable to be addressed by using *parallel computing techniques*. Parallel computing aims reducing the processing time needed to solve a certain problem. This goal is achieved by splitting a base problem into a set of subproblems, that can be tackled concurrently. The parallel execution of each subproblem is known as *execution thread*. This strategy is becoming popular, thanks to the use of the *multi-core* CPU. In these processors, each core composing the CPU is able of executing one or, in the case of *HyperThreading* capable processors, two execution threads. Hence, an HyperThreading technology-based multi-core CPU composed by 4 cores could concurrently execute up to 8 execution threads. This parallelization allows exploring the problem's search space in a lesser time, if correctly employed.

Nowadays a new concept is growing over the parallel computing approach: the HPC [9]. The HPC extends the traditional parallel computing by running a massive number of execution threads. Leaving both the computation racks and supercomputers aside, the GPU chips, due to their popularity and their growing usage, have made possible the usage of highly parallel computing techniques directly running on board the robots. So, we can refer to the GPU as a big set of cores (multiplier-based cores and not full cores like the ones on the multi-core CPU chips), that can be used to address highly parallel tasks from the point of view of the *general purpose computing*. This is the GPGPU computing.

In robotics, we can classify robots into two well differentiated groups: the mobile robots and the non-mobile robots. The main difference is that the first ones are equipped with a locomotion system that allow them to move through the available space in the environment. And it is at the time to start moving when one of the most interesting problems from the point of view of the science arises, the PP problem. PP aims finding the optimal (or the closest possible) and collision-free path in an environment while taking into account kinematic constraints (including geometric, physical, and temporal constraints). The purpose of PP, unlike motion planning which must consider dynamics, is to find a kinematically optimal path as fast as possible, while modelling the environment completely [10]. The following data are required to solve this problem:

- A computational representation of the robot environment.

- The starting point of the path to be calculated.
- The target point of the path to be calculated.

The resultant path, if one exists, will be obtained at the output of the PP algorithm (see figure 1.1).

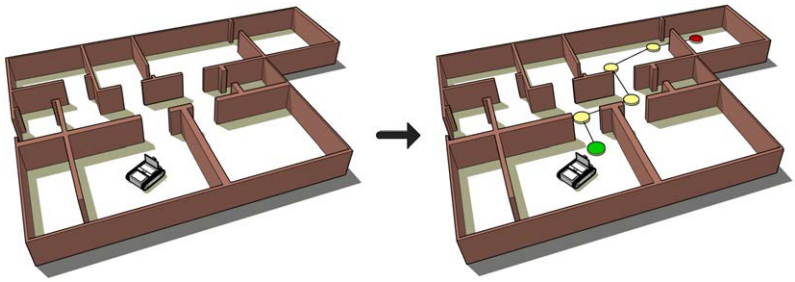


Figure 1.1: A Path Planning example.

The PP problem belongs to the *NP-hard* problems category. As discussed above, NP-hard problems imply for their resolution an exponential growing of the processing time as the size of the problem's search space grows. For this reason, and taking into account that the PP is a combinatorial optimization problem, it is a perfect candidate to be solved by applying metaheuristic-based methods.

When solving the PP problem by using metaheuristic-based methods, one of the most frequent objectives for the optimization is the *path length*, being the best path the shortest one. If the obtained path is depicted by a sorted list of space coordinates, *(x, y)*, and two consecutive points of the path form a path stretch, the path length is calculated by adding the length of each path stretch (see equation 1.1), where n refers to the number of path stretches and $D(T_i)$ refers to the length of the i^{th} path stretch. Generally, this objective is directly related to the operation time since the robot will traverse a shorter path in a lesser time.

$$Length_{path} = \sum_{i=1}^{n} D(T_i) \tag{1.1}$$

When working with multi-objective metaheuristic-based methods, some extra objectives like *path safety* and *path smoothness* are usually taken

into account to be optimized. In these situations, it is possible that some objectives are not fully completed in order to optimize the global result, as detailed below.

Path safety consists on minimizing the number of collisions with the obstacles in the robot environment. But it is possible that a path that implies a collision may not lead to a wrong solution if more objectives are considered. Hence, if the robot needs to move fast but it collides with a little obstacle, the most probable situation is that the collision won't damage the robot, and the object simply takes up a new position in the space. This event could be part of a valid solution. But of course if the robot collides with a wall it will most probably be damaged, so it is interesting to assign a weight to the space's regions depending on their danger level. Therefore, an empty region of the environment will be assigned a weight of 0% whereas if the region is fully occupied then its associated weight will be of 100%. Thus, the associated weight to a half-occupied region of the space will be between 0% and 100%. The path safety can then be evaluated by adding the weights of the regions that the robot must go through to reach the target goal, as equation 1.2 shows.

$$Safety_{path} = \sum_{i=1}^{n} W(Z_i) \tag{1.2}$$

being n the number of regions of the environment that the robot goes through when traveling a path, and $W(Z_i)$ the weight associated to the i^{th} region of the space.

It is important to take into account that considering the path safety only makes sense as long as you have a prior knowledge of the environment and its obstacles. It is also interesting to highlight in this example that this is not the main issue of this doctoral dissertation. Hence, we will understand that the information about region occupation will be provided by and external agent.

Path smoothness is another parameter that can be analyzed. This parameter aims to achieve paths as straight as possible. In addition, this parameter is directly related to the energy consumption. Thus, a robot moving through a smoother path will consume less energy. The path smoothness calculation is obtained by adding the value of the smallest

angles between each two consecutive path's stretches (see equation 1.3).

$$Smoothness_{path} = \sum_{i=1}^{n-1} min(\alpha(Si, S_{i+1}), \beta(Si, S_{i+1})) \tag{1.3}$$

In equation 1.3, n is the number of path stretches and $min(\alpha(Si, S_{i+1}), \beta(Si, S_{i+1}))$ corresponds to the smallest angle existing between the i^{th} and the $i + 1^{th}$ stretches of the path under study.

Taking into account the literature, most of the authors address the PP problem from a single-objective perspective (although in the paper titles they state that multiple objectives are used) ([11], [12], [13], [14], [15], [16] and [17]), and classical approaches always apply either metaheuristic-based or non metaheuristic-based algorithms in a purely sequential way. More recent proposals have addressed the PP problem from the *Multi-Objective Evolutionary Computation*-based metaheuristics perspective (for example the proposals presented in [18], [19] and [20]). The Multi-Objective Evolutionary Computation is a branch of the AI domain that solves combinatorial optimization problems applying the biological postulates and fitting several objectives at the same time. When this paradigm is applied to solve problems in the robotics area, the term known as *Multi-Objective Evolutionary Robotics* arises. Therefore, the Multi-Objective Evolutionary Robotics is a field of the autonomous robotics in which the robot controllers are developed taking as reference the biological evolution theory, by applying multi-objective evolutionary algorithms.

The usage of multi-objective-based techniques has the disadvantage that both the generation and the evaluation of the candidate solutions need longer computation times, compared to the single-objective-based alternatives. Taking this into account, the metaheuristic-based techniques, either in their multi-objective or single-objective versions, can notably be benefited from applying parallelism-based computation techniques. Thus, in robotics, the parallel computation pursues the following goals:

- Speed up the the execution time of the algorithms through the use of modern hardware architectures (multi-core CPUs, GPUs and others).

- Get better solutions in the same time interval, by applying sets of parallel algorithms to perform exhaustive searches in the problem search space.

The importance of speeding up computation in the robots is growing as robots move from controlled, industrial environments and predefined tasks to daily life scenarios, where they have to cooperate with people and adapt to dynamic, unpredictable environments. Temporal requirements for task execution can become strong constraints, and algorithms are required to deliver consistent solutions in reasonable margins of time. A clear example is found in the security sector, where the time to get a plan is a decisive factor in the success or failure of a certain operation. We want robots to do more complex task each time and in a faster way. A robot that spends too much time planning a certain task is useless.

This doctoral dissertation focuses on finding new techniques, based on evolutionary and parallel algorithms, that allow us solving the PP problem in large-scale real maps, and with severe temporal restrictions (real time requirements). Two levels of parallelism are applied to satisfy these time requirements. On the one hand, a *coarse-grained parallelism* that takes advantages of the multi-core-based processors and, on the other hand, a *fine-grained parallelism* that intensively uses the resources provided by the popular and powerful GPU units by using GPGPU programming techniques. Furthermore, this thesis not only provides a deep study about satisfying these temporal restrictions but, from the point of view of the multi-objective metaheuristics, it also studies how to obtain quality paths, guaranteeing robot safety and low energy consumption, thus increasing the duration of the on board energy sources (i.e. batteries). Finally, the thesis also provides a fair, complete and precise comparison of all the obtained results.

CHAPTER

2

Theoretical background

2.1 The Path Planning problem

The Path Planning (PP) problem is the problem of finding a safe path that allows a robot moving from a starting point to a target point through a given environment. It is necessary to consider both the robot and the obstacle geometry, in a 2D or 3D space, to solve this problem.

From the motion planning perspective, a robot can be defined as a set of parts. Each of these parts can occupy a space position with a certain orientation [21]. Hence, the robot can be represented as a set of geometric objects, each one characterized by a translation and a rotation. In this doctoral dissertation, however, PP just aims towards determining the accessible space positions. In this scenario, a robot can be considered as a single-body machine, that can occupy any obstacle-free space position. This model does not need to consider the cinematic nor the dynamic constraints of the robot, and so the PP problem becomes more understandable.

In any case, before solving the PP problem, it is necessary to known several common concepts to all existing planners. These concepts are described below (see [21] and [22]):

- **Robot configuration**: this is the minimum set of parameters un-

ambiguously defining the position and orientation of all the robot components. In this case, the robot configuration reduces to the robot location (Cartesian coordinates) in a certain environment.

- **Source configuration**: this refers to the initial robot configuration. In this case, this is the source point of the desired path.
- **Target configuration**: this refers to the final robot configuration. In this case, this is the target point of the desired path.
- **Configuration space *C***: this is the set of all the possible configurations of the robot. When working with constrained robots it is usual that the configuration space does not match with the physical space, but in this case both spaces are the same.
- **Collision**: a collision occurs when a robot (or some of its components) occupies a non-empty environment position.
- **Obstacle**: an obstacle is a convex set of points that occupies a space region.
- ***C*-Obstacle**: considering r as an element belonging to C (configuration space), *P(r)* referring to the physical space occupied by the robot when using the r configuration, the *C-Obstacle* (CO_i) is defined as follows (see equation 2.1).

$$CO_i = \{r \in C : P(r) \cap O_i \neq \emptyset\} \tag{2.1}$$

Taking into account equation 2.1, O_i refers to an obstacle of the environment, so the C-Obstacle associated to the obstacle O_i (CO_i) is the set of robot's configurations whose physical space collides with the obstacle O_i [21].

- ***C*-Obstacles region *CO***: this refers to the set of all the C-Obstacles. Equation 2.2 defines the C-Obstacles region mathematically.

$$CO = \bigcup_{i=1}^{n} CO_i \tag{2.2}$$

In equation 2.2, n refers to the number of C-Obstacles in the environment. In other words, n is the cardinality of the CO set.

- **Collision-free Configuration space** C_{free}: this concept refers to the subset of C (configuration space) formed by the non-colliding configurations (see equation 2.3).

$$C_{free} = C \setminus CO \tag{2.3}$$

- **Trajectory**: this is the set of configurations that the robot takes while moving from the source configuration to the target configuration. Usually, a valid trajectory is composed by configurations belonging to C_{free}. Equation 2.4 defines the trajectory concept mathematically.

$$Trajectory = \{r \in C_{free}\} \tag{2.4}$$

The PP is currently one of the most interesting problems in robotics. Thus, the scientific community is continuously trying to find new techniques to efficiently solve the problem. Although we are considering a terrestrial mobile robot in the examples provided in this thesis, the available techniques for the PP problem can be also applied when working with other types or robots, for example drones. Depending on the problem constraints and requirements, the environment structure, and the robot model, the use of certain motion planning techniques can deliver better results, while there are motion planning techniques that are not suitable to solve certain planning problems. In any case, all the existing motion planning methods can be sorted according to the Latombe's taxonomy [22], in which these categories are described:

- **Roadmaps**: the roadmap-based methods transform a multidimensional environment model into an unidimensional model composed by a network of connected curves. The network of curves expands through the collision-free configuration space (C_{free}). Thus, the resultant unidimensional model represents all the possible trajectories in the environment [23]. Some of the most representative roadmap-based techniques are the following:

  - *Visibility graphs.*
  - *Voronoi diagrams* .

  For a more detailed explanation about *Visibility graphs* and *Voronoi diagrams* see subsections 2.2.1 and 2.2.2 respectively.

- **Cell Decomposition**: these methods decompose the collision-free configuration space (C_{free}) into a set of convex regions named cells. The result of the decomposition process is an *adjacency graph* in which cells correspond with the graph's nodes. An existing edge between two nodes means the corresponding cells are adjacent, and they can be safely connected [24]. The cell decomposition-based methods can be divided into the two following categories:

  - Exact decomposition: these decomposition techniques divides the space into a set of regions whose union is identical to the C_{free} space (see section 2.1). Two examples of this subcategory are the following:

    * *Vertical decomposition.*
    * *Delaunay decomposition.*

    For a more detailed explanation about *Vertical decomposition* and *Delaunay decomposition* see subsections 2.2.3 and 2.2.4 respectively.

  - Approximated decomposition: these decomposition techniques divides the space into a set of regions of predefined shape, usually rectangular-shaped regions. For this reason, the final partition does not have to match the C_{free} space (see section 2.1), as a region can overlap with an object of the environment. Two examples of this subcategory are the following:

    * *Quadtree method.*
    * *FMM.*

    For a more detailed explanation about *Quadtree method* and *FMM* see subsections 2.2.5 and 2.2.6 respectively.

- **APF**: the aim of this technique is to lead the robot to its goal considering it as a particle moving inside an artificial potential field. This artificial potential field is formed by neutral particles (particles without electric charge) and negatively or positively charged particles. Particles with positive electrical charge attract the particles with negative electrical charge and vice versa. Particles with the same sign repel themselves. This navigation technique represents the robot and the target goal as particles with opposite electrical charges, and the obstacles in the environment as particles with the same charge sign that the particle representing the robot. In this way, the robot and the obstacles will repel themselves, while the robot and the target point will attract themselves, leading the robot to accomplish its goal [25]. For a more detailed explanation about the *APF* see subsection 2.2.7.

- **Sampling-based approaches**: these methods generate random configurations to gradually identify the environment where the robot moves [26]. In some cases the sampling-based methods iteratively build multiple *roadmap*-based graphs at the same time they check multiple randomly generated configurations for the robot. These graphs are subsequently connected in a join process. This is known as *multi-query planning*. Other sampling-based methods compute a single roadmap-based graph at the same time they check a single randomly generated configuration for the robot. This is known as *single-query planning*. Some examples of the sampling-based approaches are the following:

  – *PRM*.
  – *Rapidly-exploring Random Tree (RRT)*.

  For a more detailed explanation about *PRM* and *Rapidly-exploring Random Tree (RRT)* see subsections 2.2.8 and 2.2.9 respectively.

The Latombe's taxonomy can be extended by adding a new category known as **Combinatorial Optimization methods** (hereinafter the extended Latombe's taxonomy). Combinatorial optimization aims finding a mathematical object that maximizes or minimizes a specific function.

These mathematical objects are known as *states*, the set of all the candidate states as the problem *search space*, and the function to be optimized as the *fitness function* or the *objective function*. It is possible to design the fitness function to optimize more than one objective at the same time. This is the main difference between *single-objective optimization techniques* and *multi-objective optimization techniques*. Two of the most representative techniques of this category are the Evolutionary Algorithms (EAs) and the MOEAs, when they optimize several objectives at the same time [20]. An example of this new category is the *MOSFLA-MRPP* presented in [19]. For a more detailed explanation about *MOSFLA-MRPP* see subsection 2.2.10.

A clear explanation about how these techniques model the environment to accomplish their goals is presented in section 2.2.

2.2 Modeling the environment

This section takes into account the extended Latombe's taxonomy ([22] and [21]) presented in section 2.1. It describes how the selected representative methods of each category model the environment to subsequently try solving the PP problem in large scale maps (see figure 2.1).

The following point list summarizes the selected methods and their corresponding category according to the extended Latombe's taxonomy.

Figure 2.1: Example of large-scale real environments.

- Roadmap
  - *Visibility graphs.*

  - *Voronoi diagrams.*

- Cell Decomposition
  - *Exact decomposition.*
    - *Vertical decomposition.*
    - *Delaunay decomposition.*
  - *Approximated decomposition.*
    - *Quadtree method.*
    - *FMM.*

- *APF*

- Sampling-based approaches
  - *PRM.*
  - *RRT.*

- Combinatorial Optimization methods
  - *MOSFLA-MRPP.*

2.2.1 Visibility graphs

As other roadmap methods, the *Visibility graphs* generate a *connectivity graph* for a given environment. The connectivity graph usually is a non-directed and valued graph composed by a set of nodes and a set of edges connecting nodes. On the one hand, each node represents a feasible configuration of the robot and, on the other hand, each edge represents both a collision-free (visibility condition) connection and the connection cost between two different nodes. The resultant graph is known as the *visibility graph* [27]. Once the visibility graph has been generated, several graph-theoretic approaches, for example the *Dijkstra algorithm* [28], can be used to solve navigation planning problems.

Regarding graph creation, this type of methods need a polygonal representation of the objects in the environment. The graph creation process starts generating a list of coordinate points containing the obstacles' vertices and the source and the target points of the desired path. The

points in the list correspond with the visibility graph's nodes. When two different nodes can be safely connected (without crossing objects) a new graph edge between them is created (see figure 2.2). Each graph edge has an associated value indicating the connection cost (for example, the connection cost can refer to the edge length). The resultant graph is computationally represented as an $N \times N$ square matrix ($G(N \times N)$), where N is the number of nodes of the graph and $G(i, j)$ is the cost of the connection between i and j nodes.

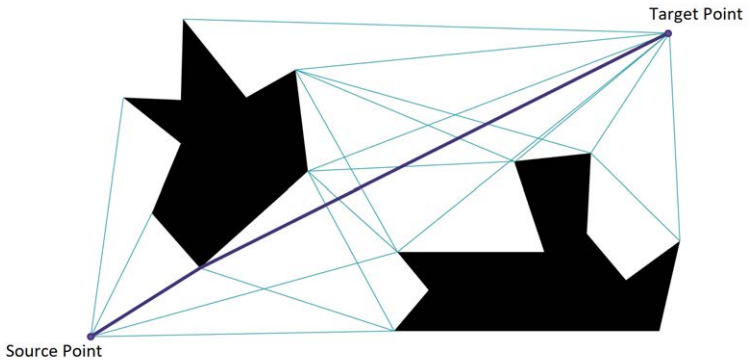


Figure 2.2: A Visibility graph example.

The main disadvantage of these methods is that when the number of objects (in this case the number of polygons) in the environment grows, the computation time needed to process the visibility graph grows excessively.

2.2.2 Voronoi diagrams

The *Voronoi diagrams* ([29] and [30]) are probably one of the most popular roadmap-based methods. These type of diagrams try to model the collision-free space (C_{free}) (see section 2.1) as a *connectivity graph*, but focusing on maximizing the distance between the robot and the objects of the environment. It is necessary a polygonal representation of the objects of the environment to create the Voronoi diagrams.

As in other roadmap-based methods, the connectivity graph generated

using Voronoi diagrams usually is a non-directed and valued graph where nodes represent valid configurations of the robot and edges represent safe transitions among nodes. The value associated to an edge indicates the cost of the transition between the connected nodes. This cost is usually a pseudo-path property, for example the euclidean distance (length of a stretch) between the related nodes.

Regarding environment modeling, the connectivity graph creation process firstly generates a set of vertices of the polygons representing the objects of the environment (see equation 2.5).

$$V = \{v_1, v_2, v_3, ..., v_n\} \mid n(V) \geq 3 \tag{2.5}$$

Once the set of vertices, V (see equation 2.5), has been generated, the graph creation process associates to each $v_i \in V$ a $r \in C_{free}$ (see section 2.1). The r point is the center of the circumference determined by the vertices v_i, v_j and v_k ($\{v_i, v_j, v_k\} \subseteq V$), where v_j and v_k are the closest vertices to v_i (see figure 2.3).

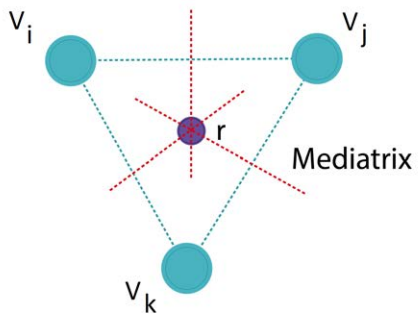


Figure 2.3: r element calculation.

Once the graph (the Voronoi diagram) has been created, nodes represent the r elements and edges represent visibility among nodes (see figure 2.4). So, in order to solve the PP problem several graph-theoretic approaches, for example the *Dijkstra algorithm* [28], or Overmars' proposals [31], can be applied for.

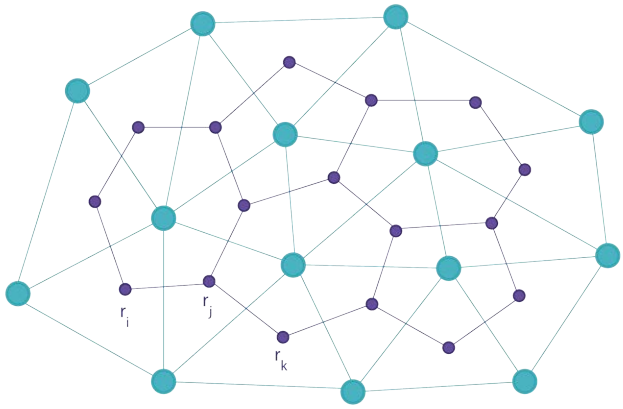


Figure 2.4: An example of the Voronoi's connectivity graph.

Computationally, the resultant graph is stored as an $N \times N$ square matrix, where N is the number of nodes of the graph, and a matrix's element at row i and column j represents the connection cost between r_i and r_j nodes.

These methods share the main disadvantage than Visibility graphs: when the number of objects (in this case the number of polygons) in the environment grows, the computation time needed to process the graph also grows excessively.

2.2.3 Vertical decomposition

The *Vertical decomposition* method (belonging to the *Exact decomposition* category), is a cell decomposition-based method. The main feature of this method is that it generates a set of regions whose union is identical to the C_{free} space (see section 2.1). Assuming that the objects of the environment have a polygonal representation, this method traces vertical lines crossing each polygon vertex in order to get a trapezoidal partition of the C_{free} space [32].

Once the partition of the C_{free} has been done, the next step is creating a *connectivity graph* to subsequently solve the planning problem. The

graph creation process takes each vertical segment (note that a traced line can be split into several segments depending on the objects intersections) and selects the middle points as the graph nodes. When two different nodes have direct vision between them, the process creates a graph edge connecting nodes (see figure 2.5). The resultant graph is usually a non-directed and valued graph.

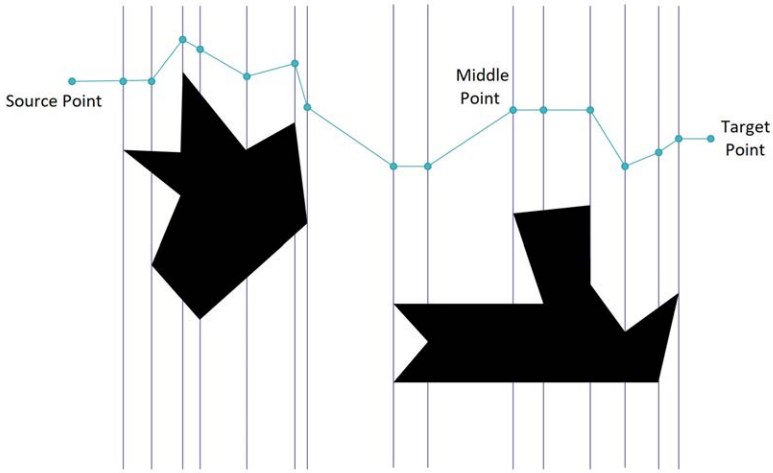


Figure 2.5: A Vertical decomposition example

Once the graph has been done, several graph-based techniques to discover the desired path can be applied for (in the case of the PP problem).

Respect to the computation time, the Vertical decomposition techniques require an extra time to sort the polygon's vertices horizontally. The complexity of this operation, in terms of computation, is $O(n \log n)$. Another important aspect is that the computation time exponentially grows as the graph expands. Respect to the storage, the final graph is represented as a square matrix where cells values indicate the transition costs among nodes.

2.2.4 Delaunay decomposition

The *Delaunay decomposition* (belonging to the *Exact decomposition* category), also named *Delaunay triangulation*, is a cell decomposition-based

method that represents a surface as a set of triangle-shaped regions that meet the *Delaunay condition*. This condition establishes that the circumscribed circumference of each triangle does not contain any vertex of another triangle of the set [33] (see figure 2.6).

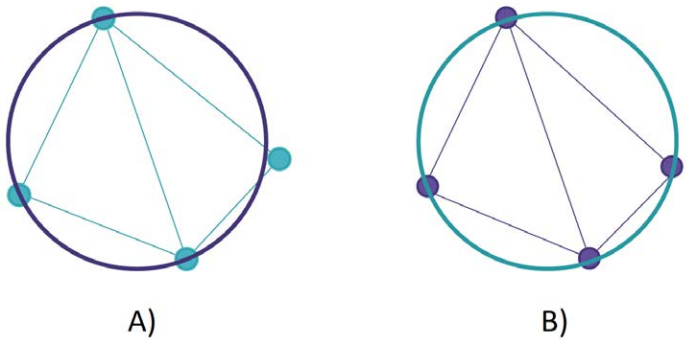


Figure 2.6: A) Mesh of triangles that meets the Delaunay condition and B) Mesh of triangles that does not meet the Delaunay condition.

The Delaunay decomposition provides very important geometric features that can be leveraged to optimize the computational calculations. These features are the following:

- The Delaunay decomposition forms the convex hull of the points at the input.
- The Delaunay decomposition maximizes the minimum angle of the triangles of the mesh. This ensures that the resultant triangles are as equilateral as possible.
- The resultant triangulation is an unambiguous triangulation if inside a circumscribed circumference there are only three vertices.

Although the Delaunay triangulation is one of the most important algorithms in *computational geometry*, it is also used to solve many other problems coming from diverse areas, for example the PP problem in robotics. Taking as reference the PP problem, in order to calculate the

Delaunay triangulation a polygonal representation of the objects of the environment is needed. Thus, the vertices of the Delaunay triangles will be the vertices of the polygons representing the objects of the environment (see figure 2.7).

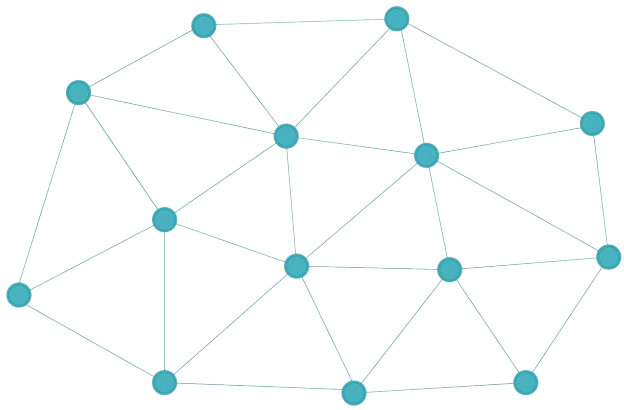


Figure 2.7: A Delaunay decomposition example.

The Delaunay decomposition is directly related with the Voronoy diagram (see section 2.2.2). The later can be obtained as the union of the centers of the circumscribed circumferences of each triangle of the Delaunay triangulation (see figures 2.8 and 2.9).

Finally, several algorithms to calculate the Delaunay decomposition exist [34]. The main advantage of these methods is that they can be easily adapted to calculate the Delaunay triangulation over the 3D space, by considering the circumscribed sphere instead of the circumscribed circumference to check the Delaunay condition. Conversely, the main disadvantage of this decomposition technique is that when the number of objects (in this case the number of polygons) in the environment grows, the computation time needed to process the visibility graph also grows excessively.

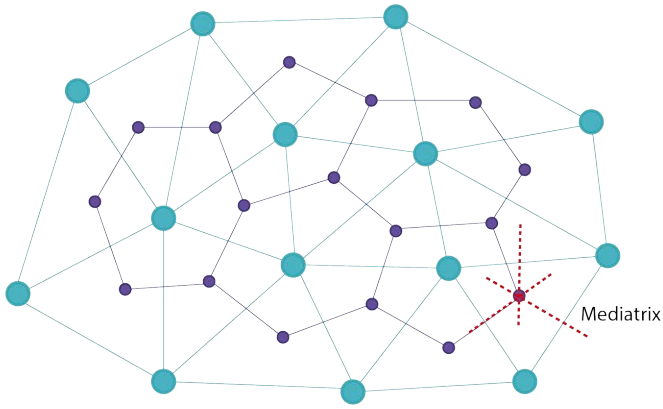


Figure 2.8: The Delaunay-Voronoi relation.

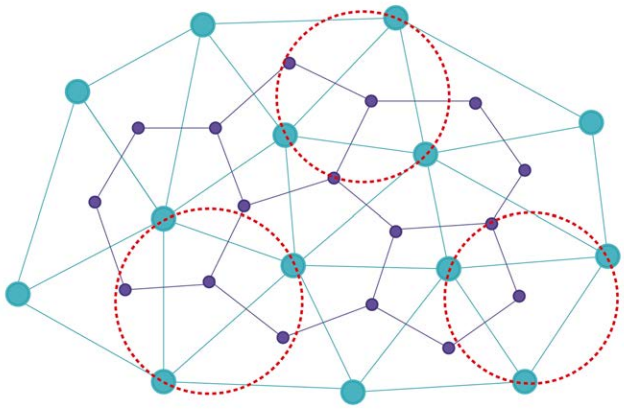


Figure 2.9: The Delaunay-Voronoi relation and the Delaunay condition.

2.2.5 Quadtree method

The *Quadtree method* (belonging to the *Approximated decomposition* category), is a cell decomposition-based method that recursively partitions

the space into rectangular regions (see figure 2.10). Once the environment is partitioned, this method generates a tree-based structure of connected quadrants to compute the environment representation quickly ([35] and [36]) (see figure 2.11).

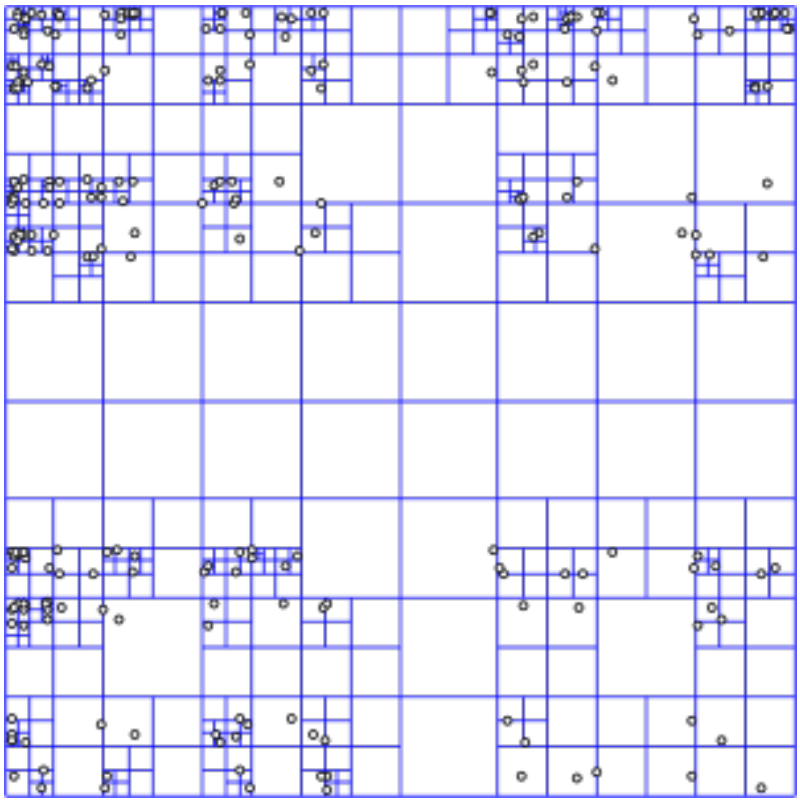


Figure 2.10: An Example of a Quadtree-based partition.

The decomposition method can be explained considering the objects of the environment as single pixels. Then, the Quadtree method recursively divides the environment representation into rectangular regions named *quadrants*. Hence, this method requires a representation of the environment (or a part of it) contained into the limits of a rectangle. The initial quadrant is then obtained as the rectangle containing the whole environment. Each time a quadrant contains one or more objects (in this case

pixels) and it is not fully-occupied, or each time a quadrant contains two or more objects and it is not fully-occupied (depending on the required precision of the final representation), then that quadrant is divided into four new quadrants. The process continues recursively until no more quadrants can be divided. Finally, the environment representation will be composed by quadrants of different sizes (see figure 2.10).

Once the environment representation has been partitioned, the Quadtree method creates a tree-based structure to compute the environment representation quickly. The main node of the tree always represents the whole environment. Each tree node has four or less children (one per sub quadrant). Each tree leaf indicates either an empty cell, a fully-occupied or a semi-occupied cell of the C_{free} (see section 2.1) space (depending on the partition precision) (see figure 2.11).

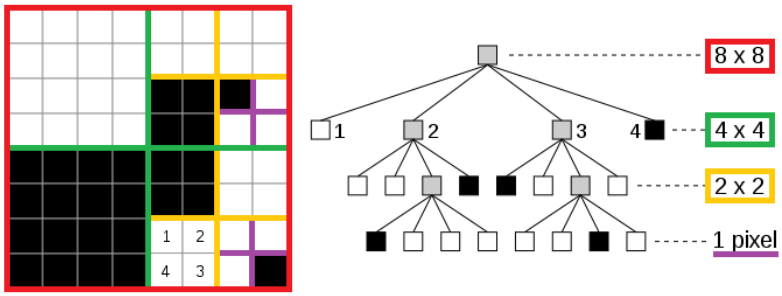


Figure 2.11: A tree-based structured generated by the Quadtree method.

The main disadvantage of this method is that when the environment is composed by a high amount of objects the method becomes very slow, in terms of computation time. The resultant tree-based structure can also consume a high amount of memory.

2.2.6 Fast Marching Method (FMM)

The FMM, which belongs to the *Approximated decomposition* category, is a cell decomposition-based method that considers a rectangular and uniform space partition. The FMM models the environment as a set of rectangular regions of the same size [37].

FMM models a wave behavior assuming that the wave only moves to the outside of the point where is was generated. In this sense, the FMM is similar to the *Dijkstra algorithm* [28]: the generated wave tends to be the shortest path, in terms of propagation time, from the generation point (in this case the initial point of the path) to the target point of the path.

The most illustrative example of an FMM arises when someone throws a stone to a pond. When the stone hits the water a new wave is generated at this point. Then, the generated wave spreads through the water with both a regular shape (circles) and a constant velocity. While the wave is propagating, if it hits anything distinct from water then it will change both its propagation velocity and its shape (see figure 2.12).

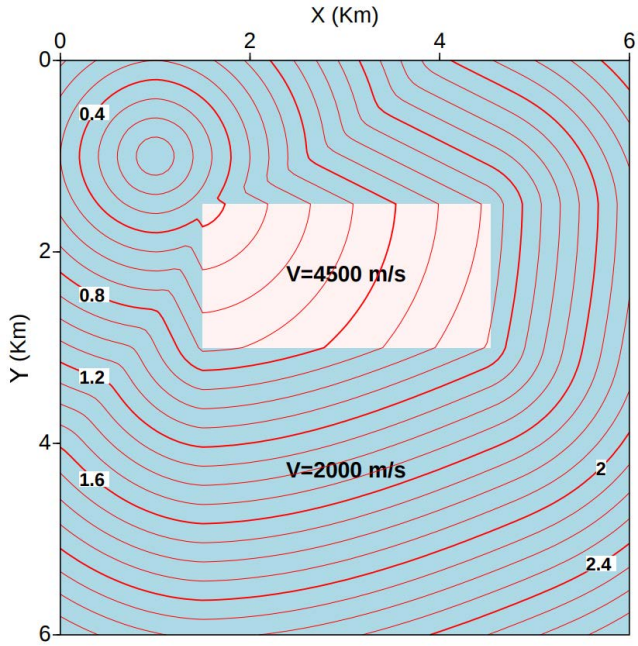


Figure 2.12: An FMM graphical example [1].

The main difficulty of this technique lies in both creating the wave computational model and simulating the wave behavior when it hits with the objects of the environment. For this reason, the FMM is sometimes a very slow method when solving the PP problem.

2.2.7 Artificial Potential Fields (APF)

The APF is a reactive navigation technique widely used when navigating in unknown environments, or for real time obstacle avoidance systems, among other applications. This navigation technique considers the robot, the target point and the objects of the environment as magnetic particles and therefore the whole environment as a magnetic model [38].

As in all magnetic models, two positive or negative particles will repel each other while two particles with different magnetic fields will attract each other. Taking this into account, the APF method considers both the robot and the objects of the environment as negative magnetic particles, and the target goal point as a positive magnetic particle (or vice versa). Subsequently, the robot and the target goal will attract each other while the objects of the environment and the robot will repel each other (see figure 2.13).

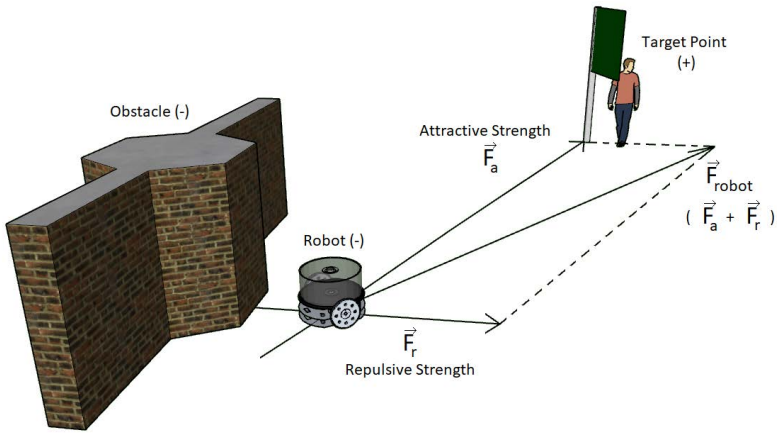


Figure 2.13: An graphical explanation of the APF method.

Figure 2.13 provides an example of these effects. In this example, the robot is exposed to a magnetic strength F_{robot} that is calculated as the sum of the attraction F_a strength and the repulsion strength F_r (see equation 2.6). The resultant strength, F_{robot}, will cause the robot to move in the same direction and with an amount of movement (magnitude) that is proportional to the associated strength vector, $\vec{F}_{robot}$ (see equation 2.7).

Respect to the motion, the robot will move towards the region of the space with the minimum value of the potential field, which will always be the problem goal.

$$F_{robot} = F_a + F_r \tag{2.6}$$

$$\|\vec{F}_{robot}\| = \sqrt{(F_a)^2 + (F_r)^2} \tag{2.7}$$

Summarizing, the execution of the APF method is based in an iterative method that follows the next sequence of actions:

1. Calculate the magnetic strength (F_{robot}) which the robot is exposed to.
2. Determine the associated strength vector $\vec{F}_{robot}$.
3. Move the robot according to the magnitude (see equation 2.7) and direction of $\vec{F}_{robot}$.

In this sense, the APF method is faster than the non reactive navigation methods, as it handles the navigation problem in a more direct way. In other words, the APF method does not spent an excessive computation time calculating complex representations of the environment, but it directly acts over the robot motion mechanism, speeding up the navigation process. But, on the other hand, the magnetic model definition implies a very slow process.

The APF is an iterative-based method. As a result, its main disadvantage is related to the estimation of the computation time. It is very difficult to estimate the computation time or the number of algorithm iterations that the robot will consume to reach the problem goal. If the number of iterations is too small, it is possible that the robot does not reach the problem goal. Conversely, if the number of iterations is too big it is possible that the robot has an excessive computation time consumption before the problem goal is reached. This number of iterations varies depending on the map density and, for this reason, it its difficult to determine the optimal configuration for the APF algorithm.

2.2.8 Probabilistic RoadMaps (PRM)

The PRM are very efficient sampling-based motion planners when it comes to robots with many degrees of freedom. PRM model the environment using *connectivity graphs*. These connectivity graphs are iteratively created at the same time they check random configurations for the robot (see figure 2.14). Taking this into account, two types of PRM exist:

- *Multi-query PRM.*
- *Single-query PRM.*

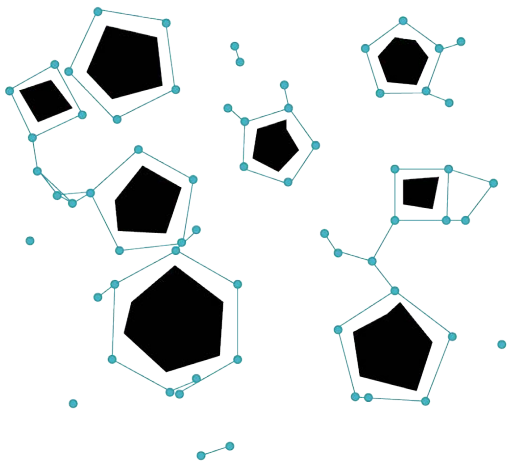


Figure 2.14: An example of a (multi-query) PRM.

On the one hand, the *multi-query* PRM are capable of checking multiple configurations for the robot simultaneously, while they build several connectivity graphs (see figure 2.14) which model the obstacles-free space, C_{free} (see section 2.1). In these approaches, a join process is responsible of connecting the isolated graphs.

On the other hand, the *single-query* PRM only check a single configuration for the robot, while they build a single connectivity graph that model the obstacles-free space, C_{free}.

The main advantage of the **PRM**-based approaches is that they do not need to calculate the limits of the obstacles of the environment. Note that this last step is more expensive (in terms of computation) even in the case of multi-dimensional environments. Accordingly, the PRM only need to use a function checking the possible collisions between a robot configuration and the objects of the environment.

2.2.9 Rapidly-exploring Random Tree (RRT)

The RRT is a sampling-based motion planner (see section 2.1). The RRT creates an *exploring tree* along the C_{free} space (see section 2.1) from a source location ($p_{ini} \in C_{free}$) with the aim of finding a safe path connecting the source location with a target location ($p_{fin} \in C_{free}$) [39] (see figure 2.15).

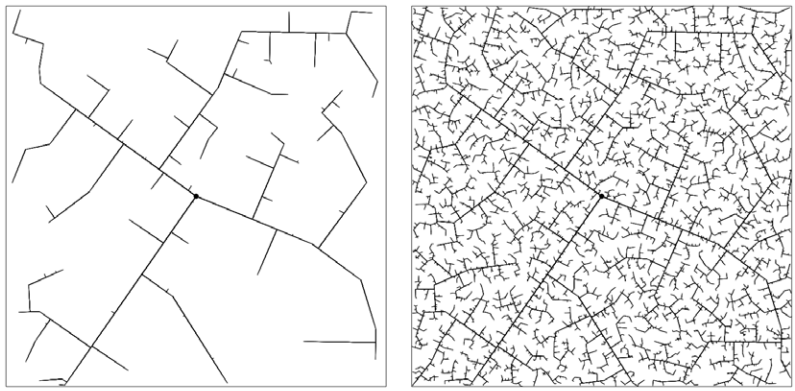


Figure 2.15: An example of the RRT algorithm [2].

The exploring tree created by the original RRT algorithm always contains the source point (p_{ini}) of the desired path at the beginning of the procedure. After that, the RRT algorithm selects a random point of the C_{free} space ($p_{random} \in C_{free}$) and expands the exploring tree if possible by adding a new point ($p_{new} \in C_{free}$). The exploring tree grows by safely connecting (without crossing objects of the environment) the p_{new} point with another point belonging to the exploring tree, p_{tree}. The selection

of the p_{tree} is done by taking into account a specific metric, for example the euclidean distance between two points (see equation 2.8). If the euclidean distance is used as the selection metric, the p_{tree} point will correspond with the nearest point from the p_{random} point in the exploring tree, so it can be referred as $p_{nearest}$.

$$D_{Euc}(p = (p_x, p_y), q = (q_x, q_y)) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{2.8}$$

Respect to the p_{new} point, if a constraint for the maximum length of each branch (denoted as ε) of the exploring tree exists, then the p_{new} point will correspond with an intermediate point in the line connecting the p_{random} and the p_{tree} points (see figure 2.16). Otherwise, p_{new} point equals p_{random} point.

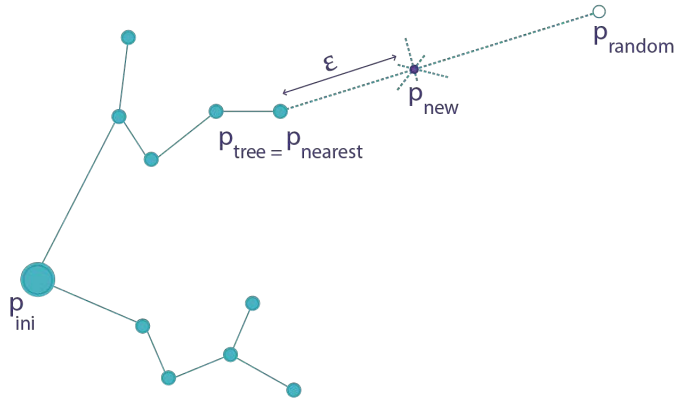


Figure 2.16: Example of calculation of p_{new} in the original RRT algorithm.

The RRT has been one of the most popular tools to solve the PP problem. Since the original algorithm was presented in 1998 by S. M. LaValle [39] several variants of the RRT algorithm and RRT-based improvements have appeared, e.g. the RRT star (RRT*) [40] or the Bidirectional-RRT* (B-RRT*) [41] among others.

Some of the main advantages of the RRT-based algorithms are the following:

- The RRT-based algorithms are probabilistically completes: the probability of finding the desired path tends toward 1 as the number of the exploring tree nodes grows.
- The exploring trees generated by the RRT-based algorithms uniformly expand throughout the available space: this makes searches over the exploring trees balanced.
- The RRT-based algorithms are generally simples: this facilitates both the behavior analysis and their implementation in any scenario.
- The RRT-based algorithms do not require preconditions to start running: they can start running from a random configuration, thus diversifying the RRT scope.
- The RRT algorithm does not need a representation of the C_{free} space: it only needs a function to check whether a path segment crosses objects of the environment.

2.2.10 MOSFLA-MRPP

The MOSFLA-MRPP is an EA that was designed to effectively solve the PP problem [19]. This algorithm belongs to the new category that extends the Latombe's taxonomy, specifically to the combinatorial optimization methods (see section 2.1).

The Shuffled Frog-Leaping Algorithm (SFLA) [42] is a memetics-based algorithm inspired by frogs' behavior in nature to tackle combinatorial optimization problems. The *memeplex* is the base concept of the SFLA. A memeplex is a puddle in which a frog population (representing potential solutions) evolves. After sometime, the best frogs (those corresponding with the best solutions) from each puddle leap to other different puddle, thus contributing to the evolution of the population.

MOSFLA-MRPP is a multi-objective approach of SFLA whose evolutionary operators were specifically designed to effectively solve the PP problem. The MOSFLA-MRPP operators optimize the safety, the length,

and the smoothness of a path at the same time in order to get a solution as close as possible to the optimal one, involving as low energy as possible.

Regarding environment representation, MOSFLA-MRPP, in its 2D version, considers the environment as a plane. This plane is split into several rows and columns to subsequently get a matrix representation of the environment (see figure 2.17).

Once the map has been split, MOSFLA-MRPP generates a matrix representation of the environment. Each cell of this matrix stores a percentage value in range [0%, 100%]. This number indicates the cell occupancy. A cell with a value of 100% indicates that it is fully occupied by an object of the environment. A cell with a value of 0% is an obstacle-free cell. Otherwise the cell will be semi-occupied by an object of the environment. Thus, the robot will be allowed to move along the obstacle-free cells and semi-occupied cells (taking into account an occupancy threshold). Graphically, empty-cells will be white colored, fully occupied cells will be black colored, and semi-occupied cells will be represented with a gray level color (see figure 2.18).

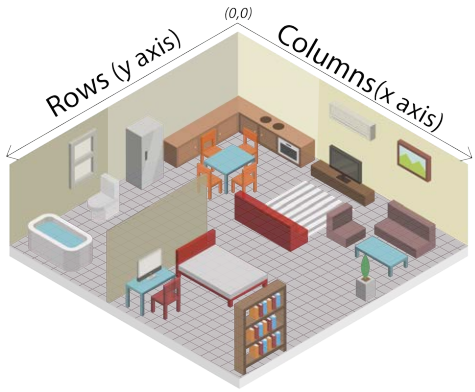


Figure 2.17: Environment splitting in the MOSFLA-MRPP proposal.

The algorithm population will evolve along the C_{free} space (see section 2.1) of the environment.

The main advantage of this proposal is that it generates pretty good solutions for the PP problem. However, MOSFLA-MRPP has several

disadvantages in comparison with other approaches in literature. These disadvantages are the following:

- MOSFLA-MRPP requires a prior knowledge of the environment and its obstacles.
- The generation of the matrix representing the environment could imply a performance reduction when dealing with large maps.
- The executions of evolutionary algorithms are generally very expensive in terms of computation (this issue implies large execution times).
- MOSFLA-MRPP, as any other EA does not guarantee a valid solution at the end of its execution.

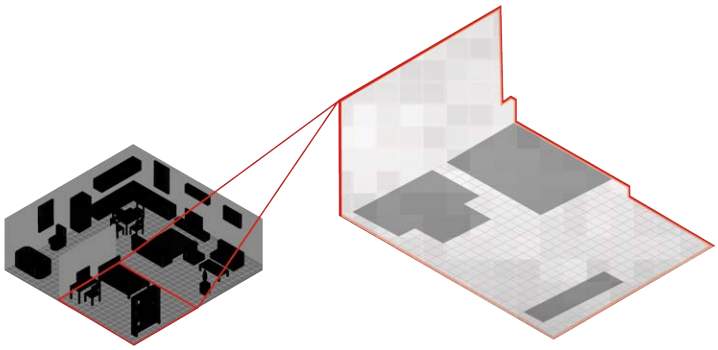


Figure 2.18: Example of environment modeling in MOSFLA-MRPP.

2.3 Path encoding

Regardless of how the robot environment has been modeled (see section 2.1), a path is always presented as a sorted list of points (representing

locations on the obstacle-free space). Each point in the sorted list could correspond with a n-dimensional location, i.e. an array of n coordinates (see figure 2.19). For example, in the 2D space a location is composed by two coordinates *(x, y)*, while in the 3D space a location will have three coordinates *(x, y, z)*. In a real scenario, using a 3D scene (a real map) with real Global Positioning System (GPS) coordinates, the x value refers to the longitude, the y refers to the latitude and the z refers to the altitude of a point of the map.

Two consecutive points in the list representing the path constitute a safe link between two different locations. The robot will safely reach its destination (i.e. without colliding with the objects of the environment), if it follows rigorously the sequence of points of the path. Otherwise, if the sequence of points is not respected, the robot could collide with the objects of the environment, and therefore it could be damaged.

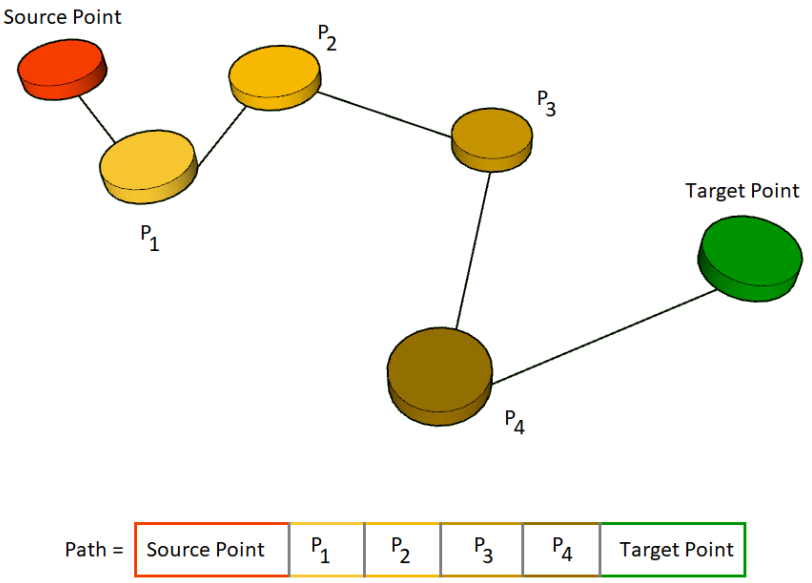


Figure 2.19: Path encoding example.

2.4 Optimizing the obtained paths

There are several techniques available to improve paths, which are independent of the chosen method. The aim of improving a path is to obtain, if possible, a more efficient variant that optimizes the robot displacement (or motion in the case of manipulators).

The existing techniques directly act over existing loops in a specific path or over the length of a path [18]. Both improvements assure that, at the output of the improving procedure, the corresponding optimal is obtained.

2.4.1 Optimizing loops

In this approach, a path is improved by identifying and erasing its loops, preventing the robot from displacing along useless path sections. Consequently, the resultant path is shorter than the original one. Firstly, this procedure searches all pairs of crossing stretches in the path (see equation 2.9).

$$\{(S_i, S_j) \in Path : i, j \in \mathbb{N} \wedge i < j\} \tag{2.9}$$

Once the crossing segments of the path have been identified, for each pair of segments, $(S_i,\ S_j)$, the procedure erases all the points between the opposite sides of S_i and S_j and adds the intersection point of the segments of the pair instead. At the end of the process a path without loops is obtained (see figure 2.20).

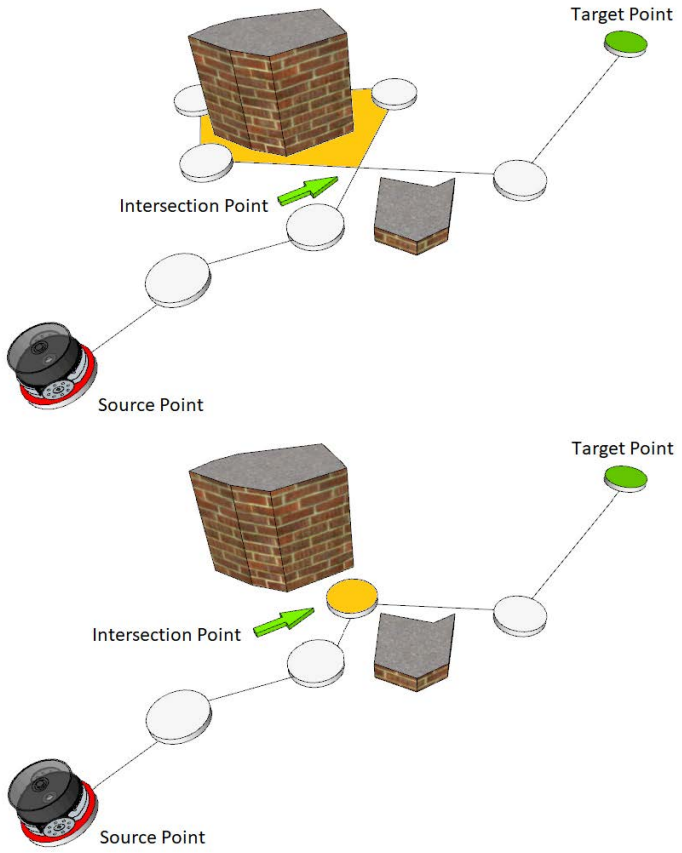


Figure 2.20: Erasing loops in path.

2.4.2 Optimizing lengths

A shorter path implies that the robot will reach the target point in a lesser time and with more balanced energy consumption. Although these are the most likely effects, the robot status at the end of the path and the travel time are not guaranteed. The length improvement procedure is an iterative process which starts improving the path length considering the target point as the origin point. Then, the process searches a safe link with the farthest point in the path (taking into account the points positions inside of the sorted list representing the path). In the next step, the

procedure will consider the last linked point as the origin point and will try to search another safe link as in the previous step. A partial length improvement will finish when the source point of the path is linked. The length improvement procedure will finish when no more partial improvements exist. The resultant path will be the optimal equivalent solution, as it will had less points and more direct stretches (see figure 2.21).

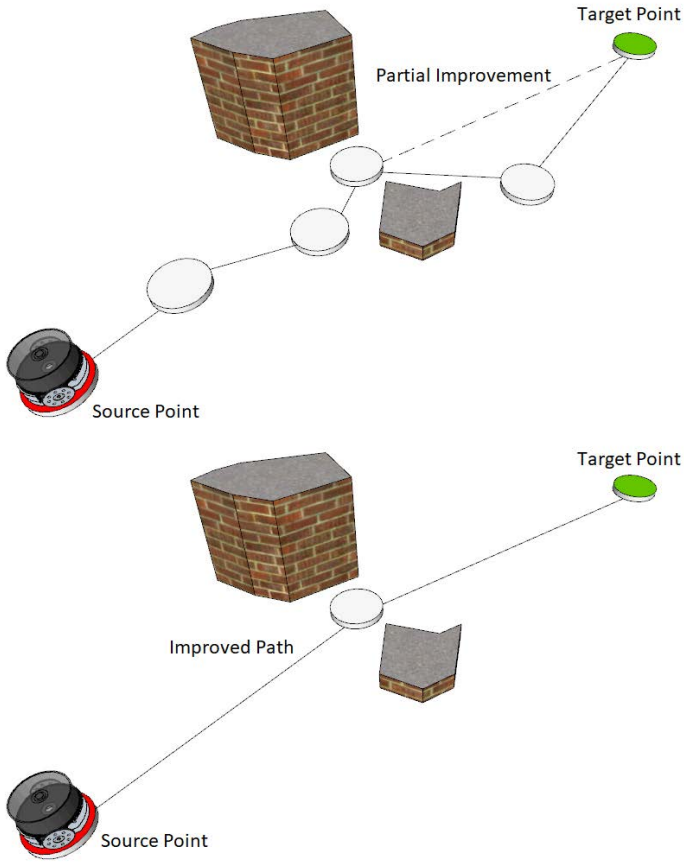


Figure 2.21: Path length improvement.

CHAPTER

3

The quad-RRT: a real-time GPU-based global path planner

3.1 Introduction

During the last decade, sampling based methods to solve the PP problem have gained more interest. Specifically, the RRT-based algorithms have become the customary technique to solve the single-query motion planning problem. However, dynamic large maps still represent a challenging scenario for these methods to produce fast enough results. Taking advantage of an NVidia Compute Unified Device Architecture (CUDA)-enabled GPU, the proposed algorithm, *quad-RRT* is an extension of the B-RRT strategy to speed up the RRT when dealing with large-scale real 2D maps. Designed for modern GPUs, quad-RRT computes four RRT trees instead of the two ones built by the bidirectional approaches. This modification aims balancing the direct searching ability of these methods with the parallel exploration of those parts of the map at both sides of the path joining the initial and goal poses. Experimental results demonstrate that the quad-RRT provides a significant speedup dealing with large-scale real maps densely populated by obstacles, when compared to other implementations of the RRT. Hence, the algorithm can have a high impact in the field of inspection PP for distributed infrastructure. It is also a

promising approach to be used in new generation robots, designed to work in unconstrained environments, and dynamically plan large-scale paths.

3.2 Related work

Taking into account that the C space (see section 2.1) contains all the possible configurations (poses) of a robot, some of these poses would result in a collision with an obstacle, and these obstacles are known before planning starts; the remainder of C space, on the other hand, is known as the C_{free} space (see section 2.1). Under these definitions, the PP algorithm is the responsible of finding a path through the C_{free} space from a starting pose $\mathbf{q_0}$ to any pose $\mathbf{q_{goal}}$ within a target set G.

Grid-based approaches provide a straightforward scheme for PP. They divide up the whole environment into an array of cells, and then apply a search technique (either classical AI techniques or more recent methods based on incremental search). These search techniques lead to a situation in which only resolution completeness can be obtained. But when the space size or the dimensionality of the problem are high, these solutions must be discarded due to their large computational cost [43]. Sampling planning approaches (the so-called PRM) were introduced to overcome this problem [44]. A PRM consists on a graph $G = (V, E)$, where each node $v_i \in V$ is a pose $\mathbf{q_i}$ in free-space. Each arc $e \in E$ between two nodes, v_i and v_j, denotes that it is possible to move from the pose $\mathbf{q_i}$ to the pose $\mathbf{q_j}$ in a straight-line movement without collision [45]. Figure 3.1 shows an example of PRM.

PRMs require the whole graph G to be built in advance. This is a computationally expensive stage that needs to unfold a large number of nodes to assure that all possible routes are considered on the graph. As a great deal of computational effort must be expended to build the graph, a PRM algorithm assumes the same topological space is used each time it is called (i.e. the graph can be reused to find new plans). This is only possible if we assume that obstacles remain unchanged. An on-line counterpart of PRMs is the RRT [39]. RRT approaches are incremental sampling-based algorithms that, as PRM, are probabilistically complete, i.e. the probability that they provide a solution, if one exists, converges to

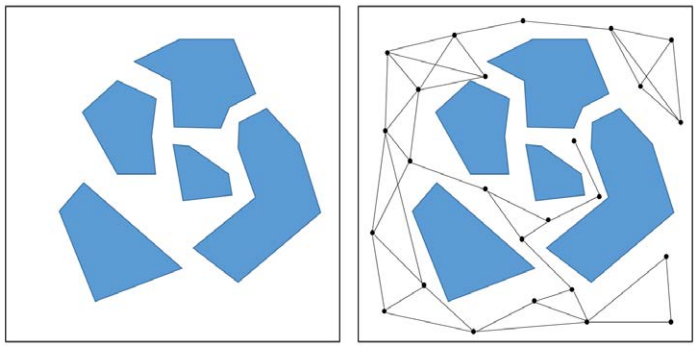


Figure 3.1: (Left) original map; and (right) a PRM, i.e. a graph consisting of poses (vertexes) in free space, with arcs connecting vertexes when a direct movement between them is collision-free

one as the number of samples converges to infinity [46]. Contrary to the multi-query PRM, RRT is single-query. A single-query planner assumes that a new state space is encountered each time it is called. Briefly, RRT samples space points and adds them to a tree, $T = (V, E)$, which grows through the whole state space. This growing process can be bounded, as the aim is to provide a single path connecting two poses. The main advantage of the RRT is its ability to look for a solution in a quick and efficient way [47]. However, RRT is not asymptotically optimal as, once an arc is set, it cannot be changed. The RRT* algorithm changed this behavior, allowing to redraw the arcs on the tree T [40], in order to optimize a given cost function estimated over the path from the initial pose to the goal one. This redefinition of the arcs makes RRT* typically slow, especially when it deals with large environments [48]. Real-time variants of the RRT* have been proposed [48, 47] to provide faster responses. However, these approaches must bound the sampling space [48] or maintain a tree that iteratively grows through the whole map [47]. In this last case the generated tree finally resembles the PRMs, and the strategy is close to the multi-query one. Other approaches make use of heuristics that speed up the original algorithm, but at the cost of computational overload caused by biasing sampling [49].

There have been more proposals to speed up the RRT: The Execution

extended RRT (RRT) [50] and the Closed Loop RRT (RRT) [51] execute the algorithm on a small portion of the environment. However, this constraint causes the trees to be tied to a specific part of the state space, which is not a desirable property. Designed for office-like environments, the *topological RRT* [3] proposed to decompose the search of the route into two separate stages. The first one is launched over a topological map. Then, the RRT is employed for continuous state-space exploration within the regions associated to the topological path. The approach provides a significant advantage on time over other RRT approaches, but the procedure to obtain the topological map is complex and off-line. Besides, it cannot be applied with single-query purposes. Taking advantage of the bidirectional search, J. J. Kuffner and S. M. LaValle proposed to speed up the RRT by building two trees instead of a single one [52]. The B-RRT creates two trees: one of them starts from the initial pose and the other starts from the goal pose. These trees grow until a single path is found, i.e. until a state that is 'common' to both trees is found [52]. Figure 3.2(top) shows an example of this process. The incremental sampling of the B-RRT can be replaced by a more aggressive heuristic to speed up the connection of both trees. The RRT-Connect [53] employs one of these heuristics (the so-called Connect one) at the end of every iteration to attempt finding a branch joining both trees. On the other hand, RRT and RRT* approaches can naturally be parallelized. Significant work has exploited this parallelism using multi-core and multi-processors CPUs, Field Programmable Gate Arrays (FPGAs) and GPUs. Some proposals rely on multiple trees: for instance, Coupled Forest of Random Engrafting Search Trees (C-FOREST) is a parallelization algorithm for single-query PP [54]. Multiple search trees grow in parallel (e.g., 1 per CPU). Each time a better path is found, it is shared between trees so all of them can benefit from its data. This approach is not a PP algorithm by itself, as it must be used on top of a predefined sampling based motion planning algorithm [54]. Other approaches, such as the Parallel RRT (RRT) and the Parallel RRT* (PRRT*) algorithms [55], use multiple threads to build a single motion planning tree. Specifically, the PRRT is designed to run on a multi-core CPU architecture. It introduces key components relevant to multi-core concurrency with the aim of creating opportunities for super-

linear speedup (i.e. computation time is sped up by a factor greater than p when p cores are used instead of one). This work shows the importance of embodying the planning approach within the specific hardware to be employed, to achieve the best results. Our focus is on designing a new, parallelized strategy to explore the state space.

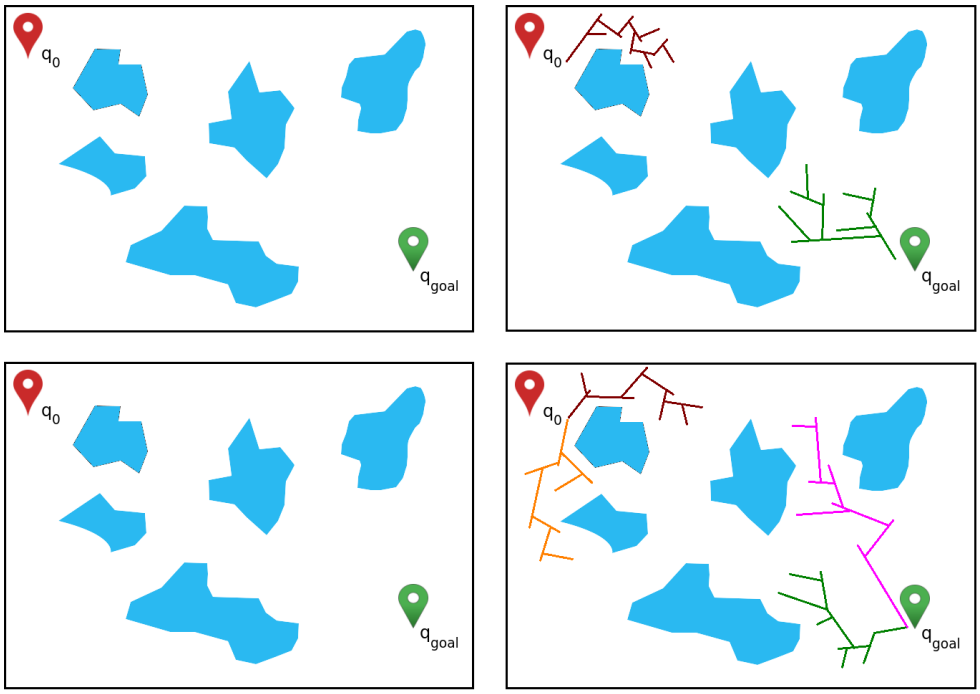


Figure 3.2: (Top) Execution of a B-RRT; and (down) execution of the proposed quad-RRT. The two additional trees increase the exploratory abilities of the algorithm, which will be able to trace a path faster than the B-RRT when the trajectory from the initial pose to the goal one is densely populated by obstacles

The quad-RRT, proposed in this thesis, resembles the data flow of the Sampling-based RoadMap of Trees (SRT) [56]. Thus, it also builds several trees, distributing the planning problem into subproblems, which are solved by the RRT. Then, these trees are connected together. As SRT, the quad-RRT proposes to distribute the workload trying to build each tree within a dedicated processing unit. However, the SRT generates

a roadmap of trees, integrating the global sampling properties of PRM with the local sampling properties of tree-based planners such as the RRT. Thus, the result is a multi-query approach. On the contrary, the quad-RRT is a tree-based, multi-query planner. Moreover, the SRT expands all trees through the whole environment. The quad-RRT creates each tree within a specific region of the map. On the other hand, the quad-RRT is implemented to be endowed within a specific hardware (NVIDIA GPU). Thus, we use the Application Programming Interface (API) of CUDA. CUDA programming model uses the GPU as a co-processor, able of executing a large number of threads in parallel. These threads are grouped into thread blocks. Each block is a collection of threads that share access to a block of device memory. In our implementation, threads within one block cooperate on building one tree. Finally, within the random sampling scheme of the RRT, we minimize the large computational cost of the collision-checking management [46] by discarding those nodes which cannot be directly linked to the tree structure.

3.3 The RRT and B-RRT algorithms

3.3.1 Problem definition

Let $X \in \Re^2$ be the bidimensional, Euclidean given state space. This space is classified into free and non-free (obstacle) regions denoted by $X_{free} \in X$ and $X_{obs} \in X$, respectively. Each pose within the state space is a node $q_i \in X_{free}$. The set of all possible nodes, V, coincides with X_{free}. When it is possible for the Autonomous Mobile Robot (AMR) to traverse from node $q_a \in X_{free}$ to node $q_b \in X_{free}$ in a straight line, an arc $e_{a,b}$ joins both nodes. The set of all arcs will be denoted by E.

Two arcs $e_{i,j}$ and $e_{m,n}$ are adjacent if they share a common node. Two poses q_1 and q_n on the free space are connected if there exists an ordered sequence of $n \geq 2$ nodes $\sigma(q_1, q_n) = \{q_i\}_1^n$ such as

$$\forall q_i, q_{i+1} \in \sigma(q_1, q_n) \quad \exists e_{i,i+1} \in E \tag{3.1}$$

The set of all arcs joining the nodes at $\sigma(q_1, q_n)$ is called a path. Then, we can also state that a path in X_{free} is a sequence of adjacent arcs. A simple path will be a path with no repeated nodes.

Let $q_0 \in X_{free}$ and $q_{goal} \in X_{free}$ represent the starting and goal poses respectively. The inherent procedure to all RRT algorithms is to explore the state space using random trees until it is possible to find a path joining q_0 and q_{goal}. A random tree $T_x = (V_x, E_x)$ is an acyclic connected graph, where V_x denotes the nodes and E_x the arcs connecting these nodes, which is formed by a stochastic process. The random tree T_x is connected because there is a path from every node $q_i \in V_x$ to every other node $q_j \in V_x$. And it is acyclic because it does not have cycles, i.e. paths whose first and last nodes are the same. Within the tree, all paths are simple paths. This simplifies the final definition of the path: when a RRT-based approach joins q_0 with q_{goal}, there will be a simple path joining both nodes.

The rest of the section presents different strategies for finding this path $\sigma(q_0, q_{goal})$. Our proposal will be described at section 3.4.

3.3.2 RRT algorithm

Algorithm 1 provides the basic implementation of the RRT algorithm [57]. Briefly, the procedure chooses a random sample q_{rand} from the search space ($Sample(\cdot)$ procedure). Then, it estimates q_{near}, the nearest neighbor to q_{rand} from the tree T (see figure 3.3).

$$NN(q_{rand}, T) := argmin_{n \in V} d(q_{rand}, n) \tag{3.2}$$

when the nearest point in T lies in an arc, then this arc is split into two arcs, and a new node is inserted into T. Within algorithm 1, this insertion is completed within the $NN(\cdot)$ procedure [57].

The procedure $New\_config(\cdot)$ allows the RRT algorithm to deal with obstacles, avoiding that the new vertex does not belong to the free space. This procedure provides the nearest free pose, q_{new}, to q_{rand}, without obstacles from q_{near}. If this new point is not the goal, the whole loop is repeated.

3.3.3 B-RRT algorithm

Algorithm 2 provides an overview of a B-RRT algorithm. The scheme is very similar to the one of the RRT algorithm presented at algorithm

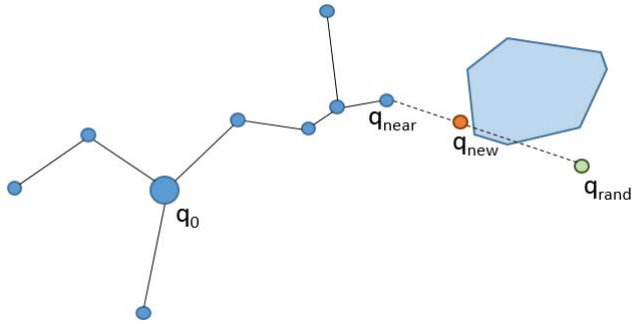


Figure 3.3: The operation for adding a new node to the tree

Algorithm 1 RRT algorithm

```
1:  procedure RRT
2:      V ← {q_0}; E ← ∅; T ← (V, E)
3:      for 0:N do
4:          q_rand ← Sample(i)
5:          q_near ← NN(q_rand, T)
6:          q_new ← New_config(q_near, q_rand)
7:          if q_new ≠ q_near then
8:              T.add_node(q_new)
9:              T.add_arc(q_near, q_new)
10:         if q_new = q_goal then
11:             return Reached
```

1. In fact, both algorithms work in exactly the same manner in the initial steps (random sampling, nearest neighbor choosing and obstacle avoiding). The proposal in the algorithm 2 also tries to ensure that the bidirectional search is balanced [58], i.e. that the sizes of both trees remain similar.

Briefly, the B-RRT algorithm decomposes the graph into two trees, T_a and T_b. These trees start from q_0 and $_{goal}$, respectively. Both trees can swap after some iterations to balance the search. As aforementioned, in each iteration, T_a grows as the RRT algorithm. The algorithm then uses q_{new} to grow T_b (lines 10-14), trying that the tree T_b grows towards T_a.

If both trees connect ($q'_{new} = q_{new}$), a solution has been found. The last step balances the search: it computes the total number of vertexes on the trees and forces new exploration to be performed from the smaller tree.

Algorithm 2 B-RRT algorithm

```
1:  procedure B-RRT
2:      V ← {q_0, q_goal}; E ← ∅; T_a ← (q_0, E); T_b ← (q_goal, E)
3:      for 0:N do
4:          q_rand ← Sample(i)
5:          q_near ← NN(q_rand, T_a)
6:          q_new ← New_config(q_near, q_rand)
7:          if q_new ≠ q_near then
8:              T_a.add_node(q_new)
9:              T_a.add_arc(q_near, q_new)
10:             q'_near ← NN(q_new, T_b)
11:             q'_new ← New_config(q'_near, q_new)
12:             if q'_new ≠ q'_near then
13:                 T_b.add_node(q'_new)
14:                 T_b.add_arc(q'_near, q'_new)
15:             if q'_new = q_new then
16:                 return Reached
17:         if |T_b| > |T_a| then
18:             Swap(T_a, T_b)
```

3.4 Proposed approach

3.4.1 Algorithm

As described in section 1.1, the possibilities of parallelizing the search using dedicated hardware should also be taken into account on the algorithms. Intimately tied to the use on 2D maps, we propose to built four trees to search the path from q_0 to q_{goal}. Two of them start from q_0 and the other two start from q_{goal}. Figure 3.4 outlines how the map is divided. The partition process considers three points: the source and goal 2D points, and the middle point on the straight segment that connects them (m).

Once these three 2D points have been calculated, the next step is to

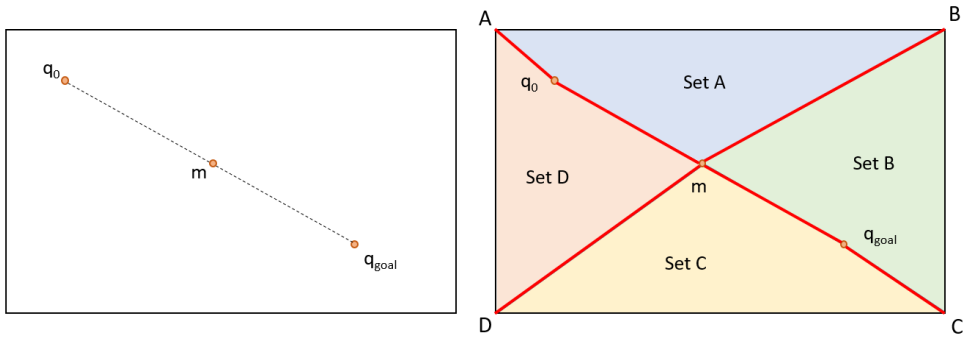


Figure 3.4: Dividing up the map to define the four submaps (see text for details)

get the partitions of the map. In order to obtain each partition, the algorithm traverses the map's corners in the clockwise direction. Set A in figure 3.4 is defined by the top left corner (A), the closest reference point to the current corner (q_0 in the figure), the middle point (m), the closest reference point to the next map corner in the clockwise direction (m again), and the next map corner (B). Boundary points of a map region cannot be repeated, so m will only appear one time into the internal structures storing data. At the end of the partition process, the map will be divided into four different parts (see figure 3.4). The quad-RRT grows four trees simultaneously. T_A and T_D start from q_0, choosing random poses from the sets A and D, respectively. T_C and T_B start from q_{goal} and they grow through sets C and B respectively. An important difference with respect to other approaches is that trees are bounded to a specific part of the map. But this constraint does not limit the searching process to a specific part of the map. The quad-tree explores all the map, as figure 3.5 shows.

Algorithm 3 describes how the quad-RRT works, while algorithm 4 details the method employed in quad-RRT to grow each tree. In each step of the quad-RRT algorithm, the four trees grow in parallel. At the beginning, each tree only contains one source point (two of the RRT trees will contain the q_0 reference point, and the other two ones will contain the q_{goal} point). In each iteration, the algorithm selects a random point of each search space ($Sample(i, A)$) and tries to increase the correspon-

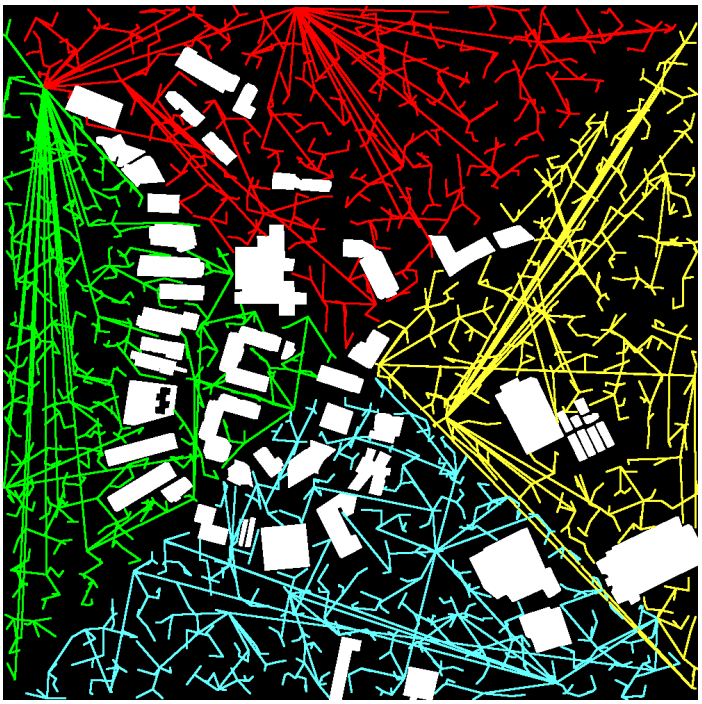


Figure 3.5: The four trees build by the quad-RRT algorithm. Each tree grows through a bounded part of the map.

ding tree. The $NN(\cdot)$ function employed in the previous algorithms is also used here, but bounded to a specific submap. The algorithm is designed to be implemented on a CUDA architecture (see section 4.1). Thus, as algorithm 4 shows, the q_{rand} is only added to the tree if it can be linked to a node q_{near} on the tree without crossing map's obstacles ($FreeLine(q_{rand}, q_{near})$). This solution provides faster response than the $New\_Config(\cdot)$ procedure employed in algorithms 1 and 2. The last step of algorithm 3 consists on a connection process between the four RRT trees, whose aim is to find the obstacle-free path between a node of the trees starting from q_0 (T_A or T_D), and a node of the trees starting from q_{goal} (T_B or T_C). This step of the algorithm is in charge, then, of finding the route from q_0 to q_{goal}. In order to avoid blockages when this connection process fails, a $max_{iterations}$ value is defined to limit the

number of times the algorithm grows the RRT trees. On the other hand, if the matching step fails, the algorithm will continue growing the RRT trees and executing the matching step $max_{attempts}$ times.

Algorithm 3 quad-RRT algorithm

```
1:  procedure QUAD-RRT
2:      V ← {q_0, q_goal}; E ← ∅;
3:      T_A ← (q_0, E); T_B ← (q_0, E); T_C ← (q_goal, E); T_D ← (q_goal, E)
4:      Initialize max_iterations
5:      Initialize max_attempts
6:      attempt ← 0
7:      reached ← false
8:      while attempt < max_attempts and reached = false do
9:          iteration ← 0
10:         while iteration < max_iterations and reached = false do
11:             grow_RRT_trees(A, B, C, D, T_A, T_B, T_C, T_D) (Algorithm 4)
12:             reached ← match(T_A, T_B)
13:             if reached = false then
14:                 reached ← match(T_A, T_C)
15:                 if reached = false then
16:                     reached ← match(T_D, T_B)
17:                     if reached = false then
18:                         reached ← match(T_D, T_C)
19:             iteration ← iteration + 1
20:         attempt ← attempt + 1
21:     return reached
```

The connection process takes each RRT tree starting from q_0 and checks if there exists a node in any of the RRT trees starting from q_{goal} which can be safely linked with that tree ($match(T_x, T_y)$ procedure). The $match(T_x, T_y)$ procedure is also carried out in a parallel way using the GPU (see algorithm 5), so this connection process is the most expensive step of the proposed algorithm. In fact, the computational cost is proportional to the number of times that the $match(T_x, T_y)$ procedure is called, and this value depends on the number of map partitions:

$$n_{matches} = (\frac{n_{map\_partitions}}{2})^2 \tag{3.3}$$

Hence, map partitions are a key parameter to reduce the computation

time of the whole process. We have heuristically set that the best performance is obtained when $n_{map\_partitions}$=4.

Algorithm 4 RRT trees growing algorithm

```
1: procedure grow_RRT_trees(A, B, C, D, T_A, T_B, T_C, T_D)
2:     q_rand ← Sample(i, A)
3:     q_near ← NN(q_rand, T_A)
4:     if FreeLine(q_rand, q_near) then
5:         T_A.add_node(q_rand)
6:         T_A.add_arc(q_near, q_rand)
7:     q_rand ← Sample(i, B)
8:     q_near ← NN(q_rand, T_B)
9:     if FreeLine(q_rand, q_near) then
10:        T_B.add_node(q_rand)
11:        T_B.add_arc(q_near, q_rand)
12:    q_rand ← Sample(i, C)
13:    q_near ← NN(q_rand, T_C)
14:    if FreeLine(q_rand, q_near) then
15:        T_C.add_node(q_rand)
16:        T_C.add_arc(q_near, q_rand)
17:    q_rand ← Sample(i, D)
18:    q_near ← NN(q_rand, T_D)
19:    if FreeLine(q_rand, q_near) then
20:        T_D.add_node(q_rand)
21:        T_D.add_arc(q_near, q_rand)
```

Algorithm 5 The quad-RRT matching procedure

```
1: procedure match(T_x, T_y)
2:     tree_node ← NULL
3:     for tree_node in T_x do
4:         if connected_gpu(tree_node, T_y) then
5:             return true
6:     return false
```

The quad-RRT is a PP algorithm that was developed as a part of a security application (a surveillance application based on Unmanned Aerial

Vehicles (UAVs) by Aeorum[1]), and in order to save computation time it has two running modes. The first one will deliver the first path found in the matching procedure among the RRT trees. This is so to reduce the computation time as much as possible. The second one alternative will deliver the shortest path found in the matching procedure among all the RRT trees. In addition, both the running mode and all parameters of parallelism used by the algorithm are fully configurable.

3.5 Analysis

3.5.1 Probabilistic completeness

As commented above, an algorithm is probabilistically complete if the probability of finding a route solution, if one exists, tends to 1 when the number of samples taken from the search space approaches infinity ($P(solution)_{samples \to \infty} = 1$). RRT is a probabilistically complete algorithm [49].

Algorithms at sections 3.3 and 3.4 show that the random sampling within the quad-RRT is addressed exactly as in the original RRT algorithm. In fact, the parallel growing of the four trees or the connection of these trees to define a route use versions of the functions employed on the RRT algorithm, which have been slightly modified to bound the growing of the trees to a specific region on the map. It can be proffered that, as RRT, the quad-RRT is a probabilistically complete algorithm.

3.5.2 Fast convergence to a solution

The main feature of the quad-RRT is its fast convergence to a solution. It is based on the simultaneous building of four trees. This Section provides a proof that our proposal provides faster convergence rates than RRT or B-RRT. A set of initial assumptions is required for this demonstration [49]:

[1] https://aeorum.com/

Assumption 1: Uniform sampling

The sampling operator provides samples from a state $C-space$ such that these samples are continuously distributed.

Assumption 2: Cluttered configuration space

The state $C-space$ is cluttered, i.e. the trees will initially grow near its starting state, and then they will incrementally grow towards the non-explored state space.

The first assumption ensures that the sampling operation will not be biased towards a goal, a procedure that has been demonstrated computationally inefficient in highly populated environments [49]. The second assumption is related to this first one: it states that there will be obstacles that will hinder the expansion of the trees in the state space.

Additionally, an important lemma for sampling-based algorithms is the one stating that they can provide almost-sure convergence to a solution [52, 40].

Lemma 1: The probability that the RRT starting at $\mathbf{q_0}$ will contain $\mathbf{q_{goal}}$ as a node approaches one when the number of nodes on the tree approaches infinity

Within the framework defined by these guidelines for the B-RRT algorithm, let $T_a = (V_a, E_a)$ be a tree starting at q_0 and $T_b = (V_b, E_b)$ a tree starting at q_{goal}. Let p_n be the probability that the n-th sampled pose v provides a final route after n-1 poses have been sampled (see figure 3.6)

$$p_n = P(v, \exists x \in V_a / e_{xv} \in E_a \wedge \exists y \in V_b / e_{yv} \in E_b) \tag{3.4}$$

Let $\Psi(N)$ be the time required to insert a node to a tree containing N nodes [59].

Lemma 2: The expected time to find a solution when inserting the n-th node for the B-RRT is

$$E_n = \sum_{n=1}^{\infty} \left[(1-p_n)^{n-1} p_n \left(\sum_{i=0}^{n} \Psi(i) \right) \right] \tag{3.5}$$

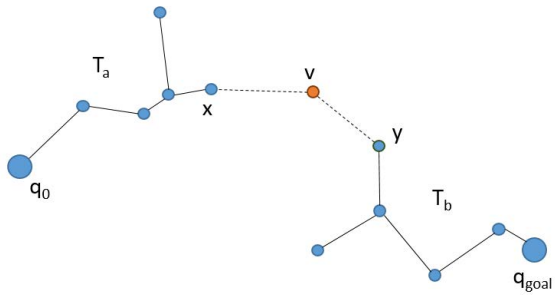


Figure 3.6: Closing a route within a B-RRT.

Proof. The time is given by the cumulative insertion of n nodes ($\sum_{i=0}^{n} \Psi(i)$), being the probability the n-th node provides a route equal to $(1-p_n)^{n-1}p_n$.

Now assuming that the quad-RRT contains four random trees, and that a route can be defined using two of these trees (one starting at q_0 and one at q_{goal}), each node insertion within a tree can lead to a solution. Let $p_{\mathbf{T},\mathbf{m}}$ be the probability that the m-th sampled pose v, inserted into tree $\mathbf{T}$, provides a final route. Let assume that these probabilities are the same for all trees. As all trees within the quad-RRT add a new node in parallel, the probability that at least one insertion provides a route is

$$p'_m = 1 - \prod_{\mathbf{T}=1}^{4} (1 - p_{\mathbf{T},m}) \tag{3.6}$$

Lemma 2 can then be extended to the case of the quad-RRT approach.

Lemma 3: The expected time to find a solution for the quad-RRT is

$$E'_m = \sum_{m=1}^{\infty} \left[(1-p'_m)^{m-1} p'_m \left(\sum_{i=0}^{m} \Psi(i) \right) \right] \tag{3.7}$$

It can be noted that we assume all proposed nodes have been inserted on the trees (i.e., there were m insertions on each tree) and that the cost of these insertions is the same for all trees. Insertions are made in parallel so we only compute the ones associated to one tree.

It can be assumed that the insertion cost is a non-decreasing function (i.e. $\Psi(i+1) \geq \Psi(i) \ \ \forall i > 0$) and that the probability a new node leads to a route is the same for all trees on the quad-RRT or on the B-RRT. Then, it can be stated that the quad-RRT, that grows more trees, will converge faster than B-RRT to a solution.

Corollary 1: Assuming $m \leq n$ and $p'_m = p_n$, the expected time required to find a route is less for quad-RRT than for B-RRT

Proof. This follows directly from lemmas 2 and 3. The term associated to the insertion cost in both expressions will be equal or lesser on the quad-RRT.

This conclusion is intuitive. As M. Otte pointed out, *drawing more samples from a random distribution increases the probability of finding what we are looking for* [59].

3.6 Graphical analysis

With the aim of analyzing the quad-RRT algorithm behavior graphically and validating the lemmas stated in section 3.5, several step by step examples of the quad-RRT have been run.

In these examples, two large-scale real maps with very different structures have been used. On the one hand, a large-scale real map populated with a normal density of objects (see figure 3.7). On the other hand, a large-scale real map containing a very high density of objects (see figure 3.8).

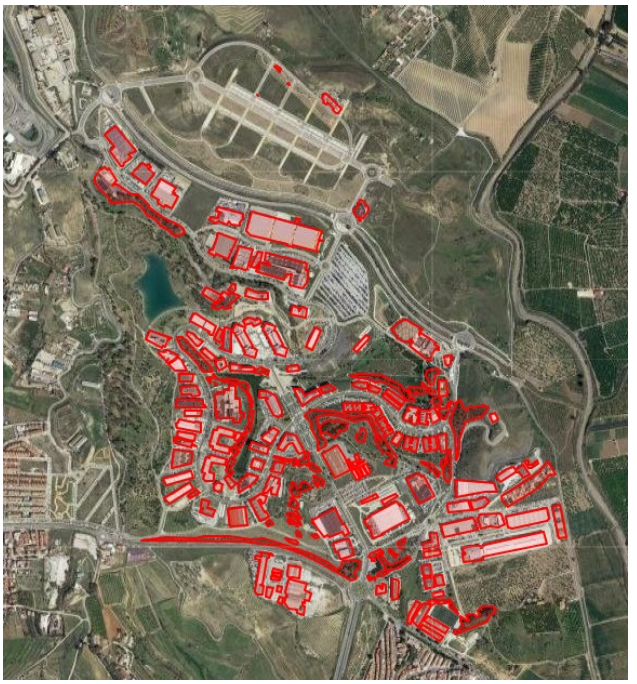


Figure 3.7: Large-scale real map of Andalusia Technology Park, Málaga, Spain.

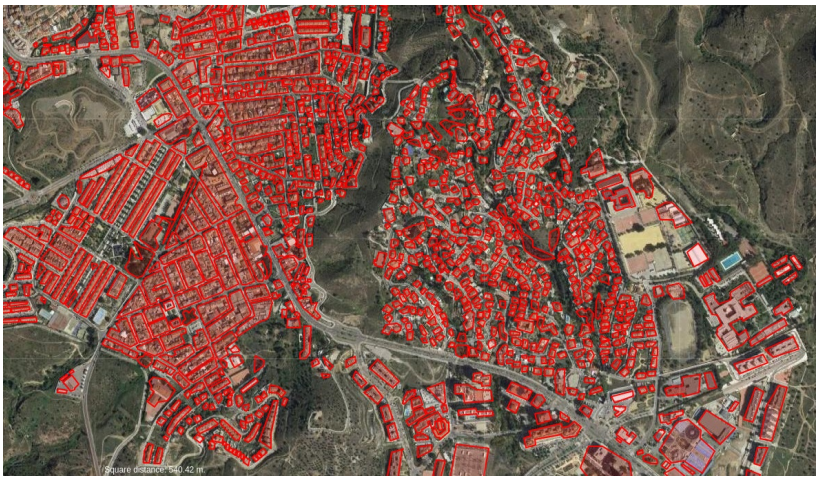


Figure 3.8: Large-scale real map of Puerto de la Torre, Málaga, Spain.

Maps in figures 3.8 and 3.7 have an approximate sizes of 5,82 km^2 and 3.51 km^2, respectively. In order to highlight the objects density in maps, figures 3.9 and 3.10 are zoomed images of sections of the maps appearing in figures 3.8 and 3.7, respectively.

Figure 3.9: Zoomed image of the map of Andalusia Technology Park, Málaga, Spain.

Figure 3.10: Zoomed image of the map of Puerto de la Torre, Málaga, Spain.

The step by step tests in this section show how the quad-RRT algorithm trees grow in each iteration of the method. Note that all the RRT trees exploring the map grow in parallel in the GPU hardware. Each RRT tree is represented using one different color and the source and target points of the desired path are represented using two orange-colored circles.

In both cases, the algorithm finishes when, after a number of iterations, a valid path connecting the source and target points is found. Hence, the obtained path is not necessarily the optimal one, although in any case it is a valid solution.

Figures from 3.11 to 3.23 show the partial RRT trees and the path obtained at the end of the process, when a section of 2.25 km^2 of size belonging to the map in figure 3.7 is used as input.

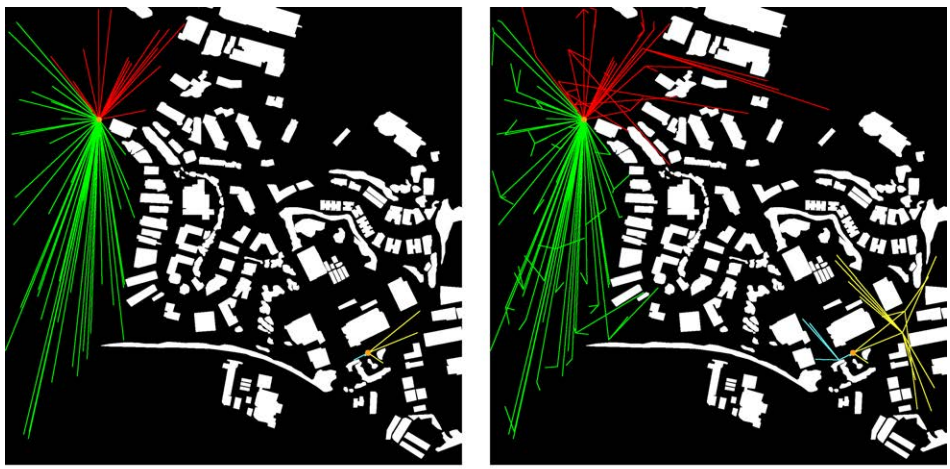


Figure 3.11: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 1 and 2.

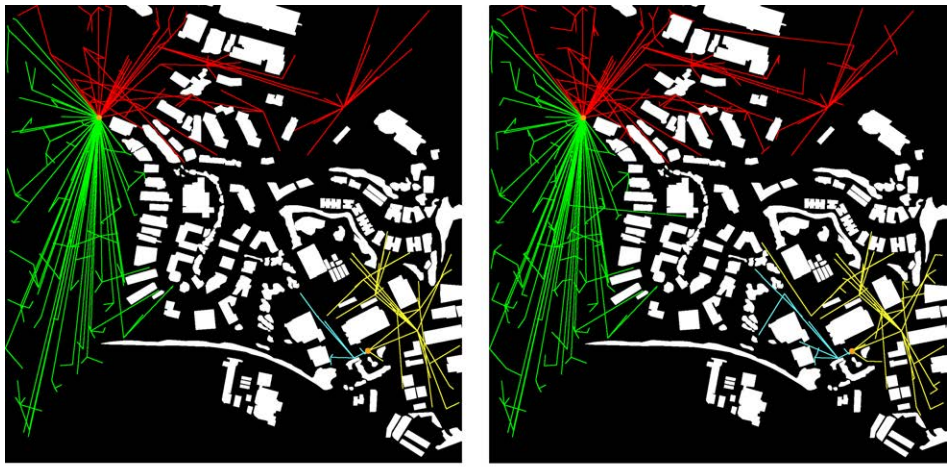


Figure 3.12: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 3 and 4.

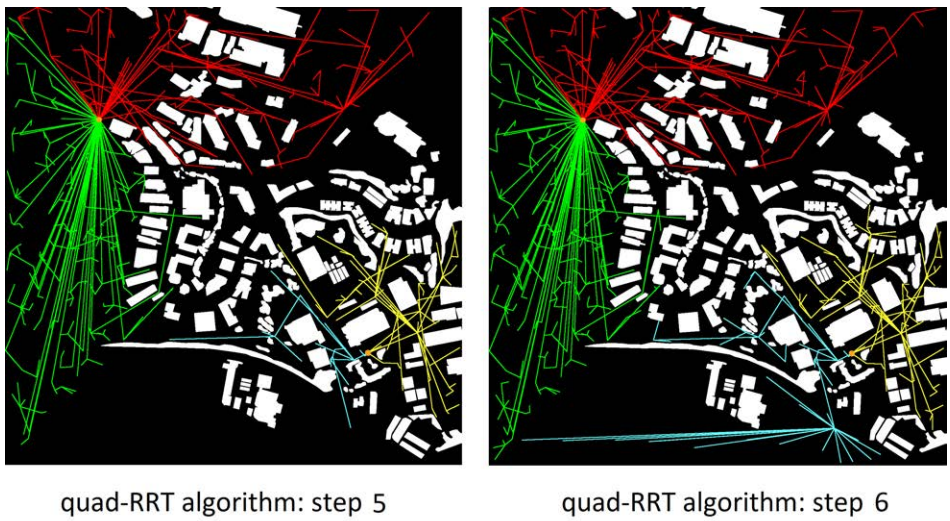


Figure 3.13: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 5 and 6.

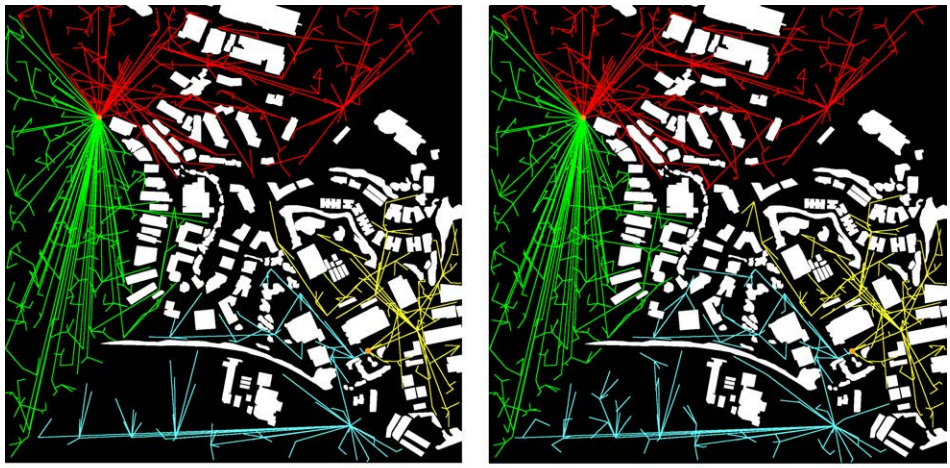


Figure 3.14: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 7 and 8.

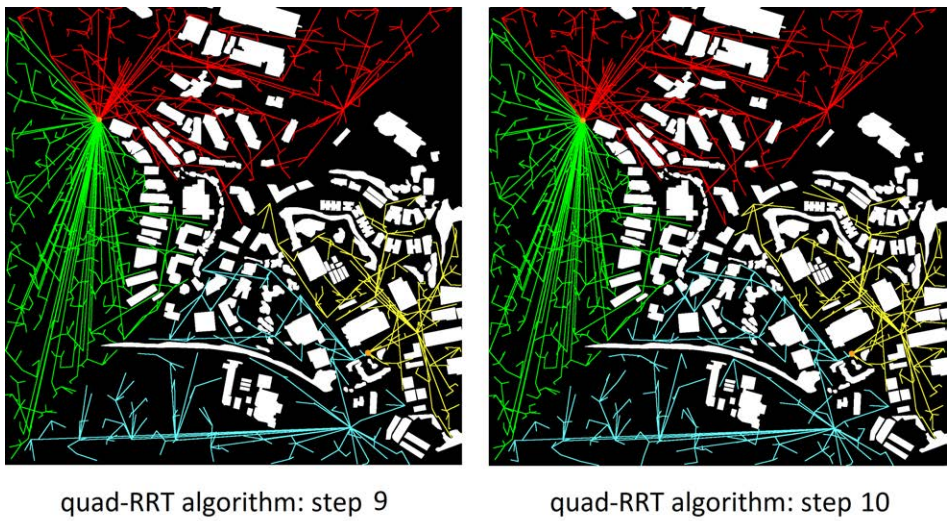


Figure 3.15: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 9 and 10.

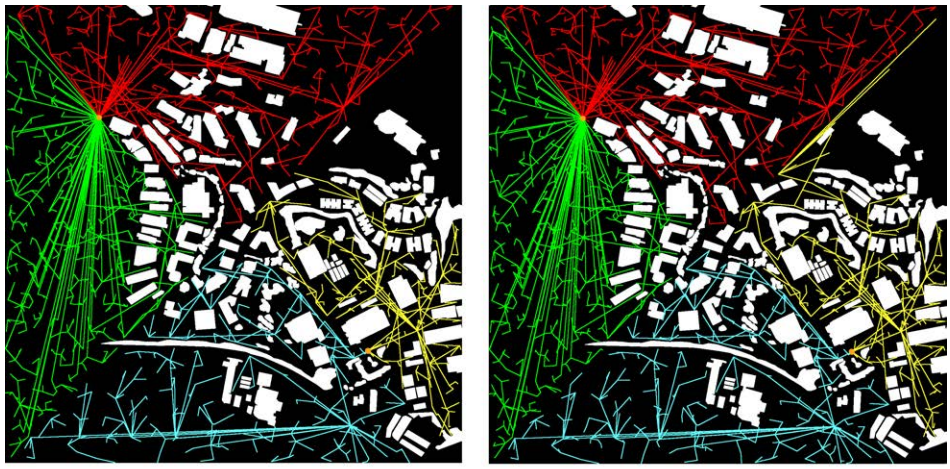


Figure 3.16: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 11 and 12.

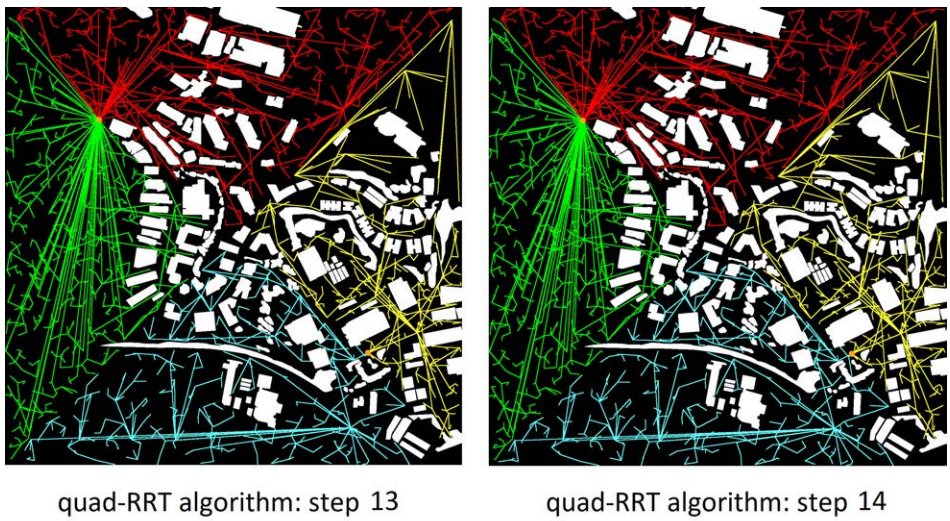


Figure 3.17: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 13 and 14.

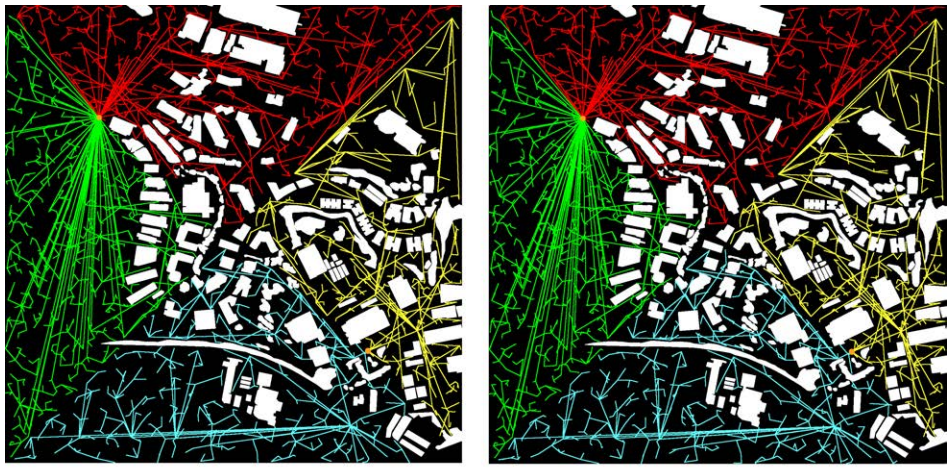


Figure 3.18: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 15 and 16.

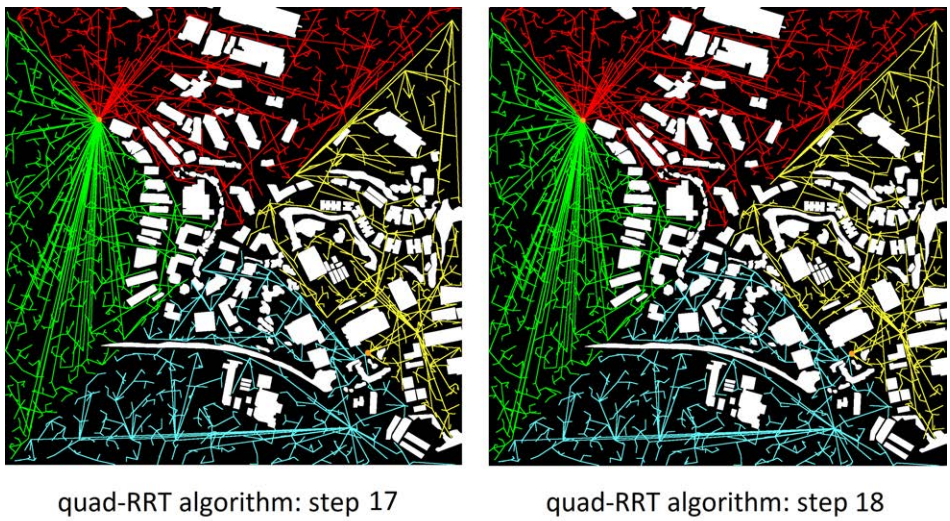


Figure 3.19: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 17 and 18.

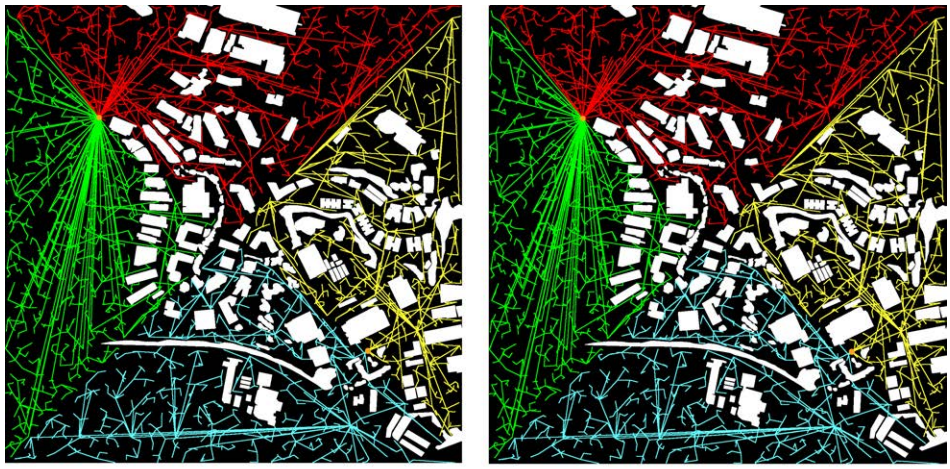


Figure 3.20: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 19 and 20.

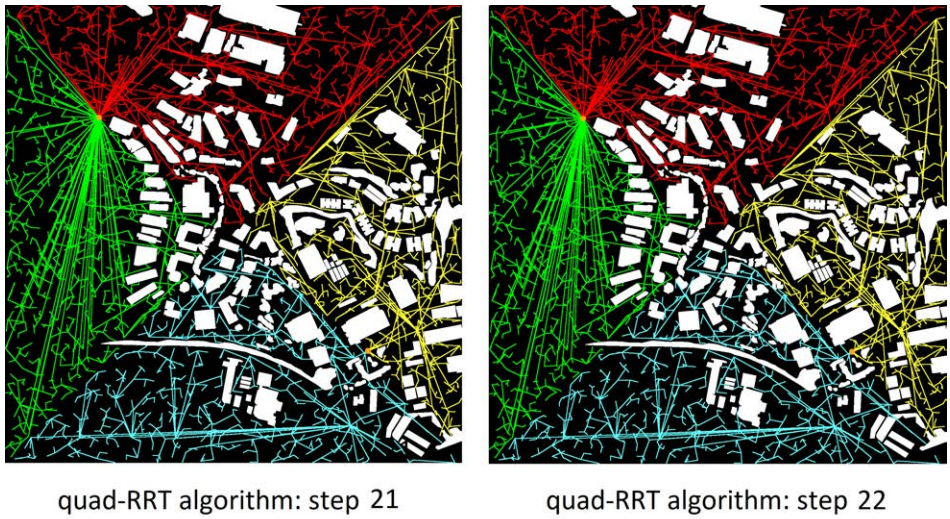


Figure 3.21: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 21 and 22.

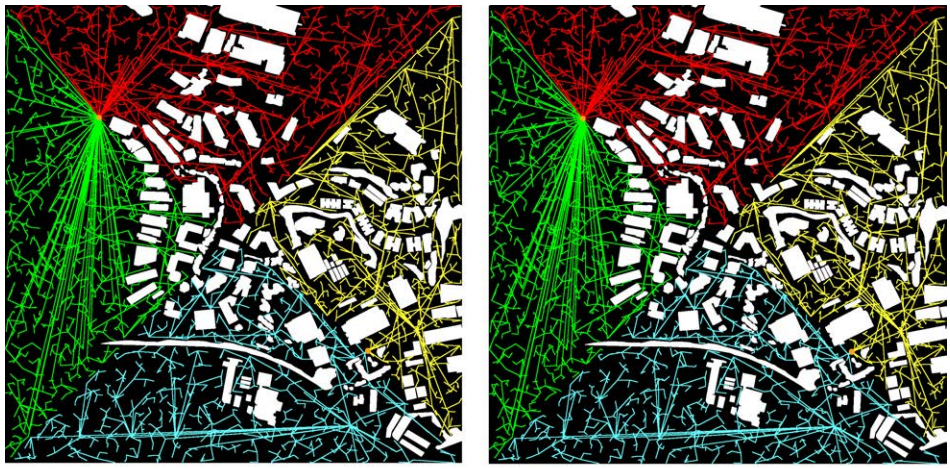


Figure 3.22: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), steps 23 and 24.

Figure 3.23: Step by step example of the quad-RRT algorithm over a large-scale real map with a normal density of objects (see figure 3.7), step 25 and the calculated path.

It can be seen that the four RRT trees that grew in parallel inside the map, by using the GPU hardware, expanded evenly through the C_{free} space (see section 2.1), exploring most of available space. Furthermore, as the trees are denser, the probability of finding a feasible path is closer to 1. As the images show, after some iterations of the algorithm a valid solution is found. Thanks to speeding up the algorithm using massive parallelization, the execution time spent to reach a feasible solution is quite reasonable, making the quad-RRT a valid algorithm for real-time applications. In this example, the execution time is less than *882 milliseconds*. This time includes some extra time, introduced by thread synchronization, to get the step by step images of the test. Furthermore, the quality of the obtained solution makes the algorithm a suitable path planner for the most demanding applications.

Another example has been computed taking as input a section of 490000 m^2 of size, belonging to the map in figure 3.8. Figures from 3.24 to 3.36 show the partial RRT trees and the path obtained at the end of the process.

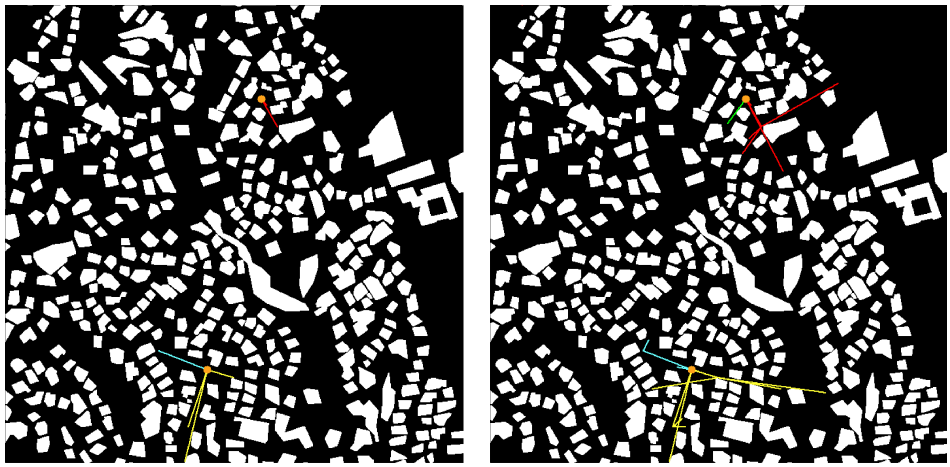


Figure 3.24: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 1 and 2.

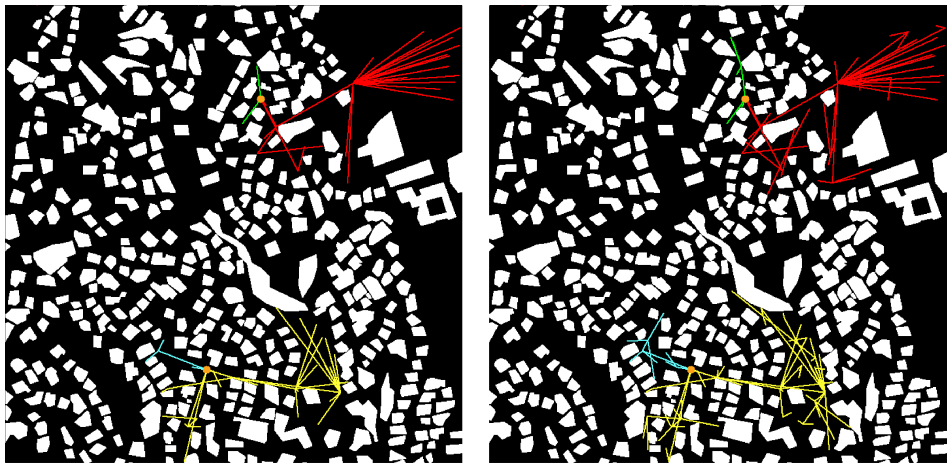


Figure 3.25: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 3 and 4.

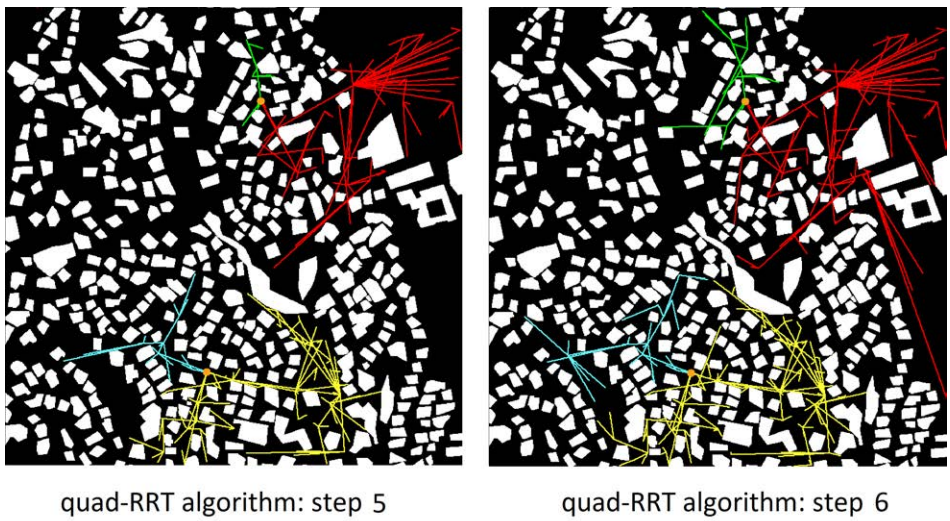


Figure 3.26: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 5 and 6.

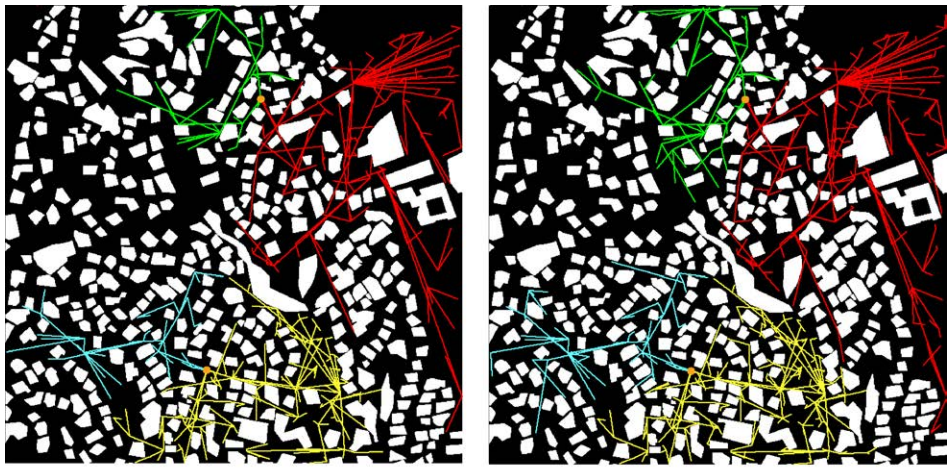


Figure 3.27: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 7 and 8.

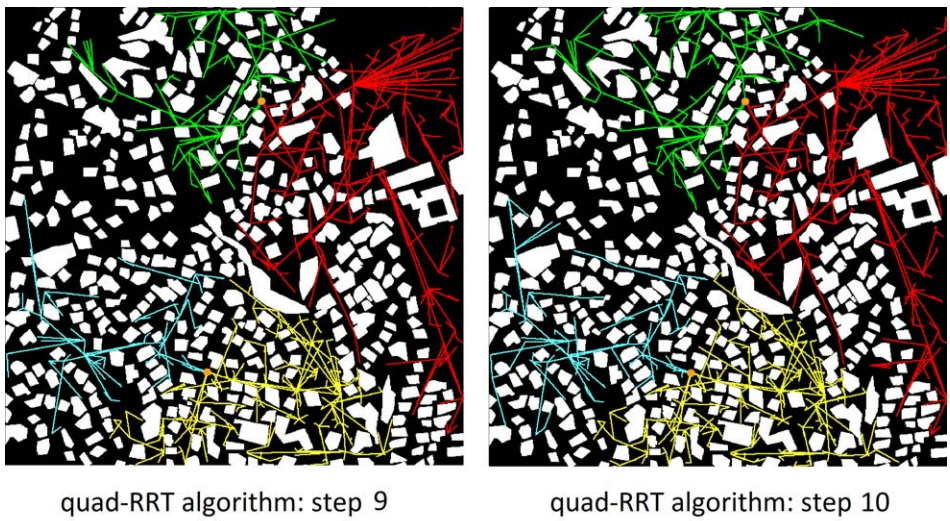


Figure 3.28: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 9 and 10.

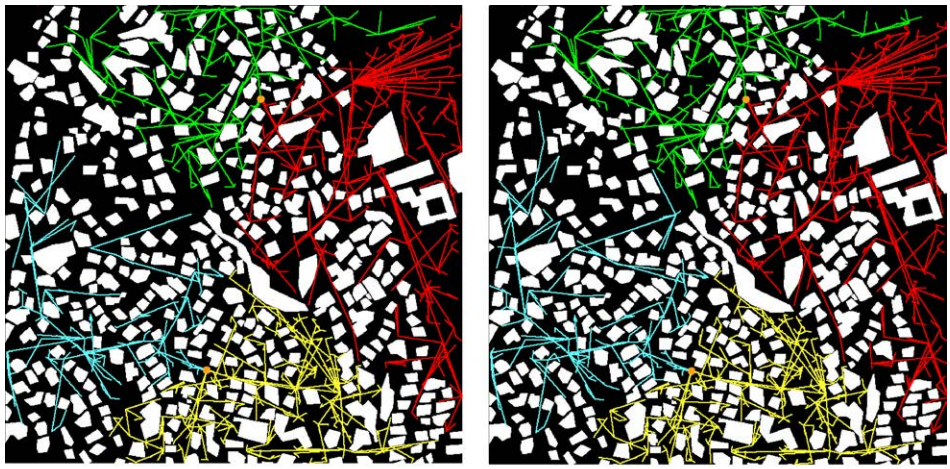


Figure 3.29: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 11 and 12.

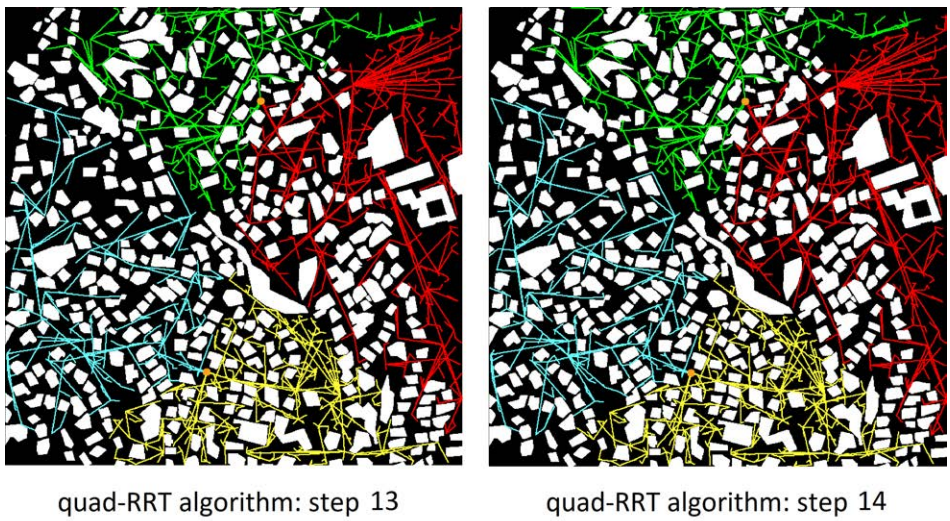


Figure 3.30: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 13 and 14.

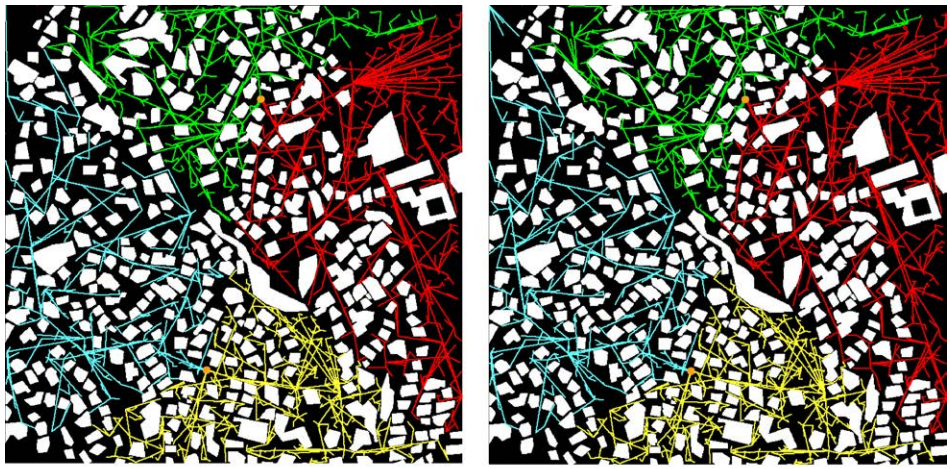


Figure 3.31: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 15 and 16.

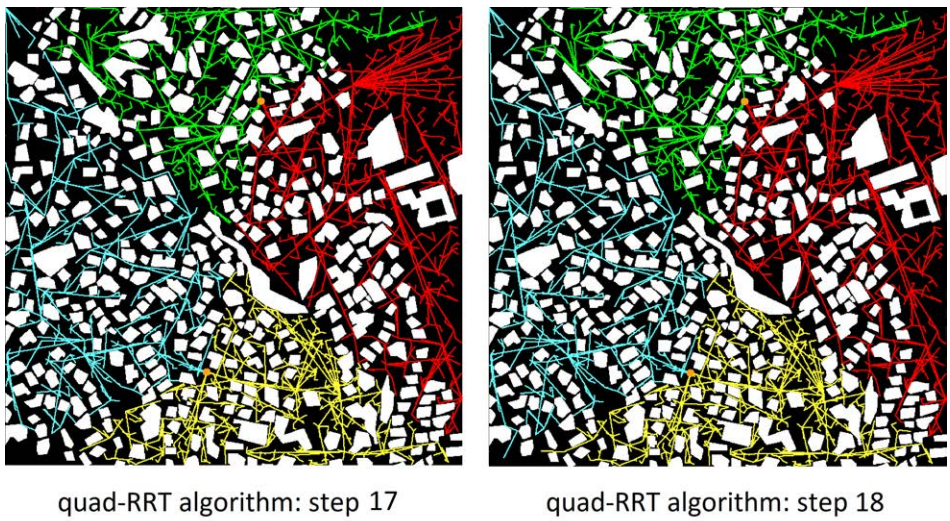


Figure 3.32: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 17 and 18.

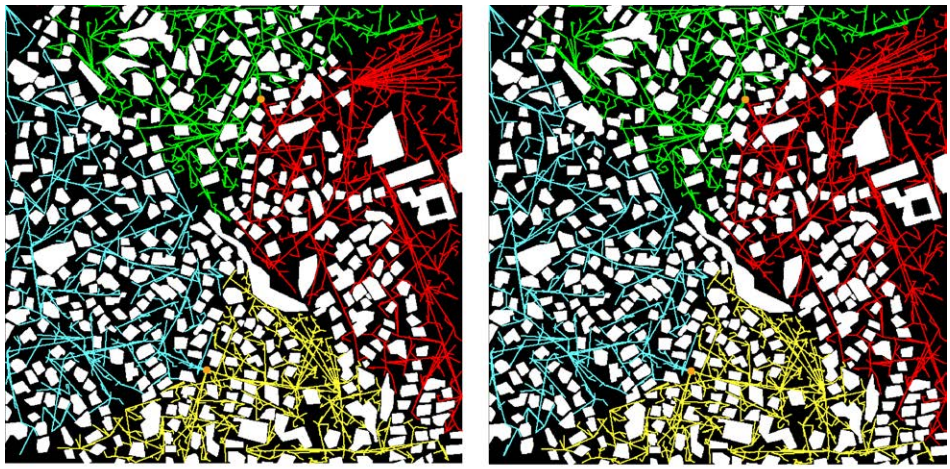


Figure 3.33: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 19 and 20.

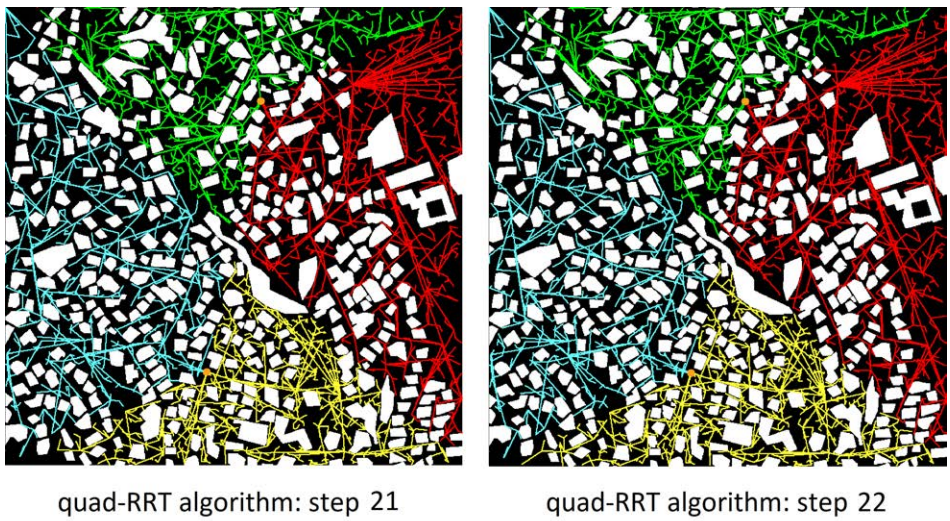


Figure 3.34: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 21 and 22.

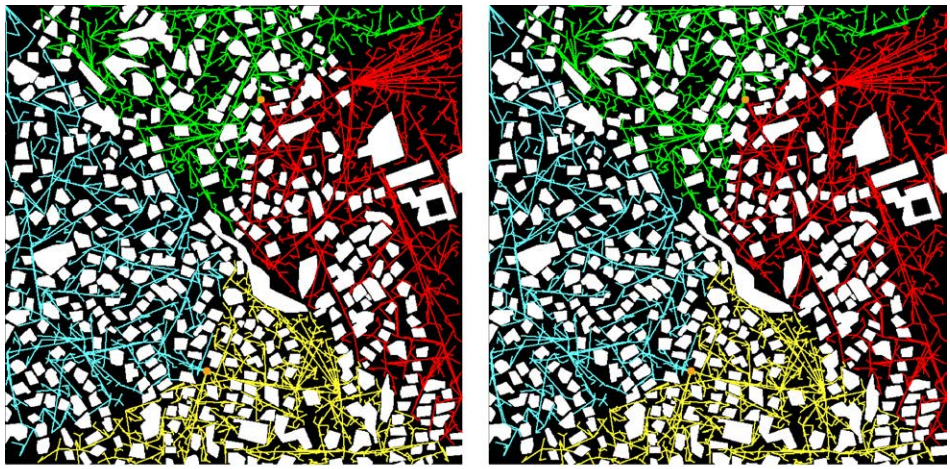


Figure 3.35: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), steps 23 and 24.

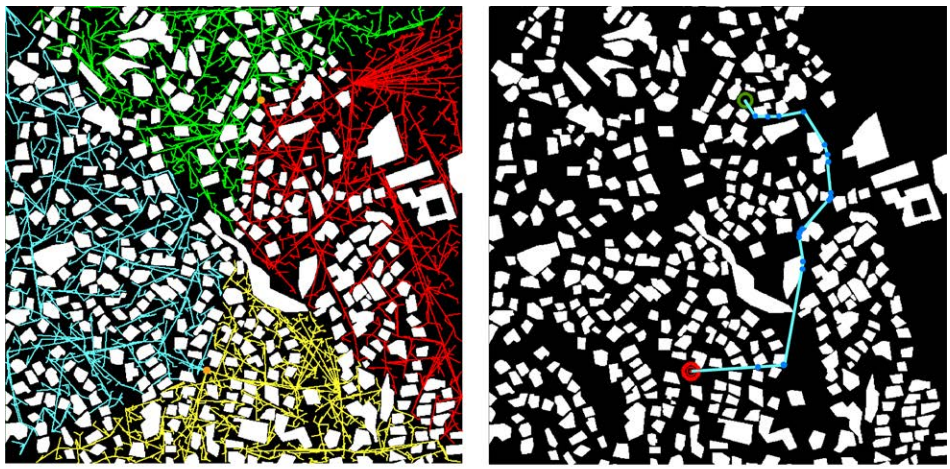


Figure 3.36: Step by step example of the quad-RRT algorithm over a large-scale real map with a very high density of objects (see figure 3.8), step 25 and the calculated path.

As in the previous test, the four RRT trees expanded evenly through the C_{free} space (see section 2.1), exploring again most of the available space. A valid solution was found, so as the RRT trees are denser, the probability of finding a feasible path is closer to 1. In these particular case, the execution time is less than *428 milliseconds* (again, this is due to an extra time introduced by thread synchronization to get the step by step images of the test). As in the previous test, the quality of the obtained solution is good enough.

When considering these results, it is possible to assert that all the lemmas in section 3.5 are fulfilled for different scenarios, from little or normal dense maps (referred to the objects in the environment) to very high dense maps.

CHAPTER

4

Experimental results

4.1 Evaluating the quad-RRT

4.1.1 Dataset and hardware specifications

This section details the results obtained by the quad-RRT algorithm when executed over eight different 2D maps. Each map corresponds with a real scenario, which is defined by its size (as rows by columns in meters) and the coordinates of its center (expressed in latitude and longitude). Half of the dataset correspond to a map with a normal density of objects and the other half of the dataset to a map with a very high density of objects. Features of the maps are summarized in tables 4.1 and 4.2 respectively. All of the maps have been generated by Aeorum[1] to test a surveillance application based on UAVs. This surveillance application is focused on security and is therefore a critical application. With the aim of avoiding memory faults, the quad-RRT has a limit respect to the map size (this size varies depending on the GPU chosen for computation). Taking into account the hardware presented in table 4.6, the quad-RRT algorithm supports areas up to $100km^2$. The maps capture real data from a large area at Campanillas (Málaga, Spain), for the maps with a normal density

[1] https://aeorum.com/

of objects, and a large area at Puerto de la Torre (Málaga, Spain), for the maps with a very high density of objects. Figures 4.1 and 4.2 show satellite views of the environment from Google Maps which refer to the normal-dense and the very high-dense maps respectively. The red flag marks each map center in latitude and longitude.

Figure 4.1: Satellite view of the normal-dense real environment mapped for testing from Google Maps.

Figure 4.2: Satellite view of the high-dense real environment mapped for testing from Google Maps.

Center (lat,lng)	S size	M size	L size	XL size
(36.738254,-4.552890)	1000×1000	2000×2000	4000×4000	8000×8000

Table 4.1: Main features of the maps for testing the proposed quad-RRT over the normal-dense map. Size is given in height per width in meters.

Center (lat,lng)	S size	M size	L size	XL size
(36.744120,-4.484123)	500×500	700×700	1000×1000	1500×1500

Table 4.2: Main features of the maps for testing the proposed quad-RRT over the high-dense map. Size is given in height per width in meters.

Within each scenario four routes were searched. The starting and target points of these routes were established in the main directions of each map. We call these directions as the main diagonal (MD), the secondary diagonal (SD), the horizontal direction (HD), and the vertical direction (VD). Tables 4.3 and 4.4 provide the real latitude and longitude of all source and target points in the normal-dense and in the high-dense maps respectively. Figures 4.3 and 4.4 draw them over real maps from Google Maps.

Direction	Source point (lat,lng)	Target point (lat,lng)
MD	(36.739458,-4.559469)	(36.732700,-4.549438)
SD	(36.737850,-4.551176)	(36.733242,-4.559512)
HD	(36.735022,-4.559169)	(36.738045,-4.550800)
VD	(36.733524,-4.556104)	(36.740212,-4.555042)

Table 4.3: Main map directions referred to the normal-dense map (see text for details).

Direction	Source point (lat,lng)	Target point (lat,lng)
MD	(36.736834,-4.474026)	(36.733497,-4.469091)
SD	(36.734518,-4.474201)	(36.737631,-4.468898)
HD	(36.736326,-4.472633)	(36.736432,-4.468482)
VD	(36.737868,-4.470588)	(36.734130,-4.471442)

Table 4.4: Main map directions referred to the high-dense map (see text for details).

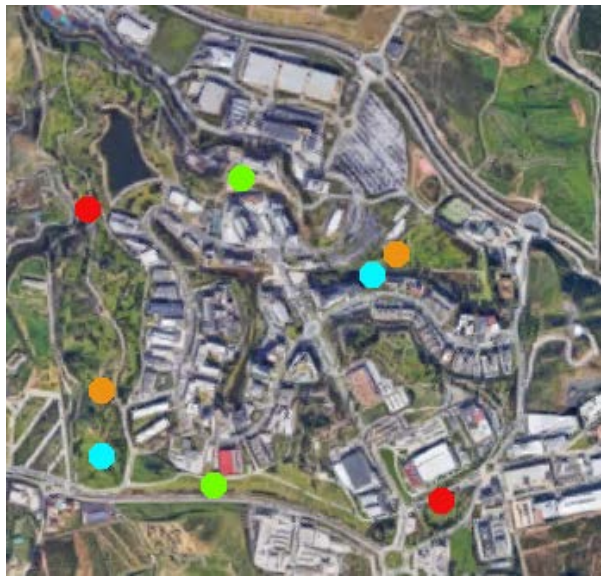


Figure 4.3: Source and target points of the routes to calculate. Points for MD, SD, HD and VD routes are drawn on red, blue, orange and green color, respectively. This image corresponds with the normal-dense map and it has been provided by Google Maps.

In order to test the proposed algorithm, several instances with different levels of parallelism have been executed. Table 4.5 summarizes the different levels of parallelism considered for this study. It can be noted that the number of thread blocks is the same than the number of map partitions (i.e. four blocks). Within our block-based implementation, each RRT tree grows efficiently inside the same thread block. Following the typical CUDA GPU architecture, each thread block can execute several

Figure 4.4: Source and target points of the routes to calculate. Points for MD, SD, HD and VD routes are drawn on red, blue, orange and green color, respectively. This image corresponds with the high-dense map and it has been provided by Google Maps.

threads at the same time.

Level of Parallelism	Thread blocks	Threads per block
L1	4	1
L2	4	5
L3	4	10
L4	4	25
L5	4	50
L6	4	75

Table 4.5: Levels of parallelism.

Finally, table 4.6 provides details about the hardware specifications of the GPU in which the proposed algorithm has been tested. The quad-RRT algorithm has been implemented using C/C++ language and NVidia CUDA v8.0.

Parameter	Value
GPU model	NVidia GeForce GTX 1070
Memory	8 GB
Frequency	1683 MHz
CUDA Cores	1920
Max. Blocks	30
Max. Threads/Block	1024

Table 4.6: NVidia GeForce GTX 1070 hardware.

4.1.2 Quad-RRT runtime analysis in the normal-dense map

In this subsection we present and analyze the results obtained by the quad-RRT over the scenarios described at section 4.1.1, specifically for the normal-dense map. The proposal has been developed to be a fast planner, so the analysis focuses on the execution times of the algorithm. Tables 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, and 4.13 show the obtained execution times. Each table is associated to a specific level of parallelism. In order to help understanding the data appearing in these tables, several graphs are also shown. The graphs at figures 4.5a-d group the obtained data taking into account the path direction (table 4.3) and the level of parallelism (table 4.5). It is also interesting to point out that there is a value of threads per block over which the execution time increases when the number of threads grows, due to collisions between threads in the blocks. Detecting this value is a key step in getting the best performance for the execution of the algorithms. Obtained results illustrate that the best performance is generally obtained when the level of parallelism $L3$ and $L4$ are used for computations.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	339	315	337	328
M size	947	1002	1037	904
L size	4171	3952	4090	3643
XL size	15531	13957	15811	14903

Table 4.7: Level of parallelism L1. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	331	335	338	349
M size	898	912	1046	1063
L size	4044	5592	3889	3772
XL size	14163	14511	15086	14421

Table 4.8: Level of parallelism L2. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	282	267	274	350
M size	1243	941	1099	1052
L size	4215	4033	4337	3940
XL size	14686	14434	13820	13937

Table 4.9: Level of parallelism L3. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	405	353	368	417
M size	999	1001	1051	1081
L size	3750	4076	3862	3936
XL size	14417	16340	15691	14552

Table 4.10: Level of parallelism L4. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	603	633	609	612
M size	1317	1336	1382	1040
L size	4094	3899	4385	4113
XL size	14972	14865	15073	15535

Table 4.11: Level of parallelism L5. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	839	795	812	865
M size	1527	1521	1552	1641
L size	4449	4889	4737	4589
XL size	15435	14929	16484	15003

Table 4.12: Level of parallelism L6. quad-RRT execution times for the MD, SD, HD and VD routes in the normal-dense map.

Level of Parallelism	S size	M size	L size	XL size	Total
L1	329.75	972.5	3964	15050.5	5079.19
L2	338.25	979.75	4324.25	14545.25	5046.88
L3	293.25	1083.75	4131.25	14219.25	4931.87
L4	385.75	1033	3906	15250	5143.68
L5	614.25	1268.75	4122.75	15111.25	5279.25
L6	827.75	1560.25	4666	15462.75	5629.18

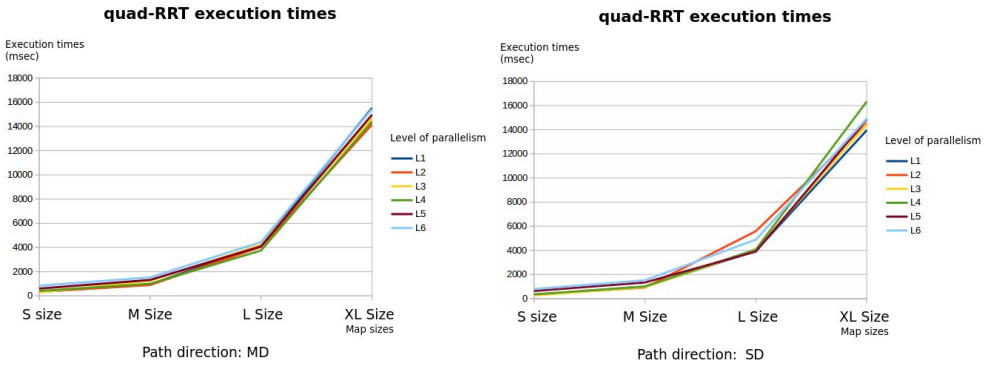
Table 4.13: Average execution times (ms), depending on the level of parallelism and the size of the maps. Test corresponding with the normal-dense map.

4.1.3 Quad-RRT graphical results obtained in the normal-dense map

With the aim of providing a graphical description of how the quad-RRT works, several images corresponding to one of the executed tests are presented in this Section. Specifically, the images correspond with the execution of the quad-RRT with a level of parallelism L4 and path direction MD. The S-sized map at table 4.1 is use din this example. The map consists of a real cartography of the area illustrated at figure 4.1. Obstacles in the map have been marked and stored in a geographical data base. The total size of the map is 1000 meters by 1000 meters. The resolution of the map (minimum size of obstacles) is 1 meter $\times$ 1 meter. Figure 4.6 illustrates the map: obstacles are drawn in the map using the white color, while free space is drawn using black color.

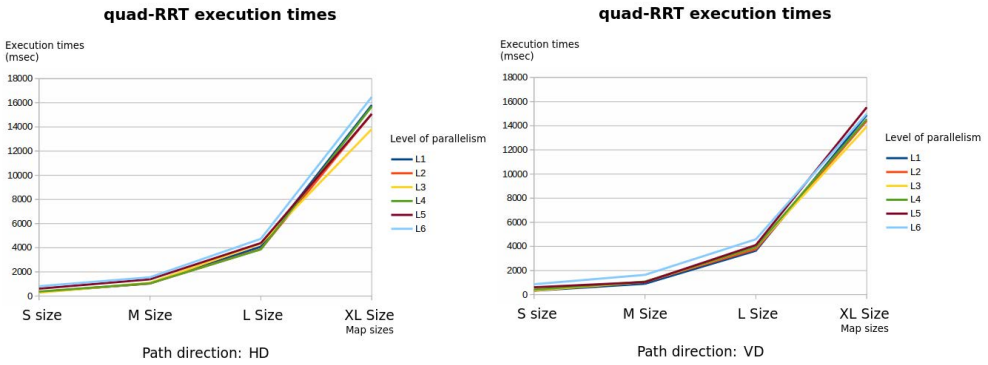
Figure 4.7 shows that the execution of the quad-RRT algorithm builds four RRT trees, which grew through the map. Each RRT tree has been drawn over the map using a different color. As depicted, the calculated RRT trees expand uniformly inside the map. This behavior indicates that the quad-RRT is correctly exploring the complete map. Figure 4.7 also shows that the final sizes (i.e. number of nodes) for the four trees are balanced. During this growing process, after adding a node to each tree, the matching process tries to find a path to travel from the source pose to the target one in a secure way. Figure 4.8 shows the finally calculated path.

Although the $L1$, $L2$, $L3$, and $L4$ parallelism levels have similar execu-



(a) MD direction

(b) SD direction



(c) HD direction

(d) VD direction

Figure 4.5: Execution times for the calculation of the path in our four directions, using different levels of parallelism and map sizes. These graphs correspond with the experimental result obtained when using the normal-dense map.

tion times, the $L4$ option, which uses 25 threads per GPU block, explores a higher map space in the same time than executions using $L1$, $L2$ or $L3$. Again, the example in figure 4.9 shows that the best results are obtained using the $L4$ level of parallelism, as the quad-RRT algorithm accomplishes a deeper exploration of the map in a similar amount of time.

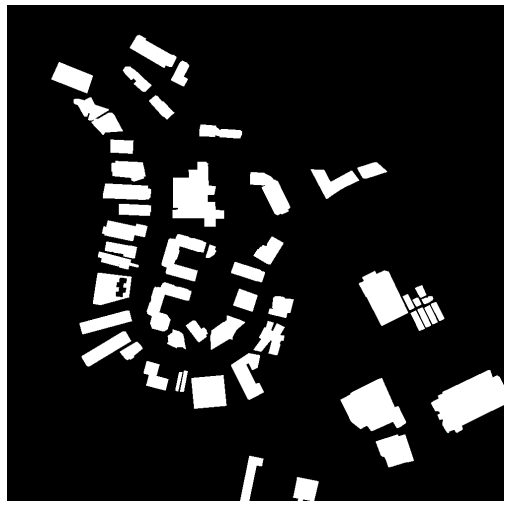


Figure 4.6: Binary representation of the real and normal-dense map of 1000 meters by 1000 meters. The area is located at Campanillas (Málaga, Spain).

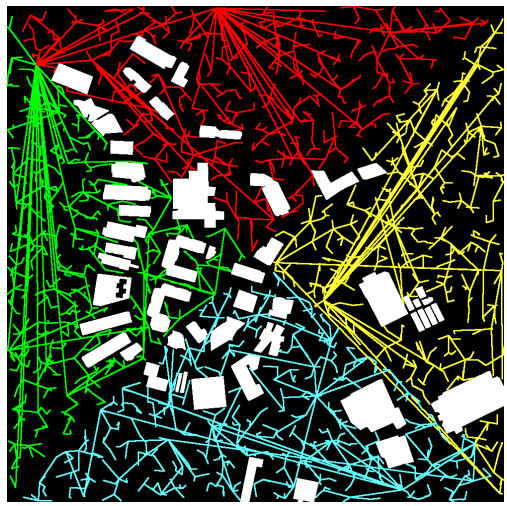


Figure 4.7: RRT trees generated by the quad-RRT algorithm in the normal-dense map.

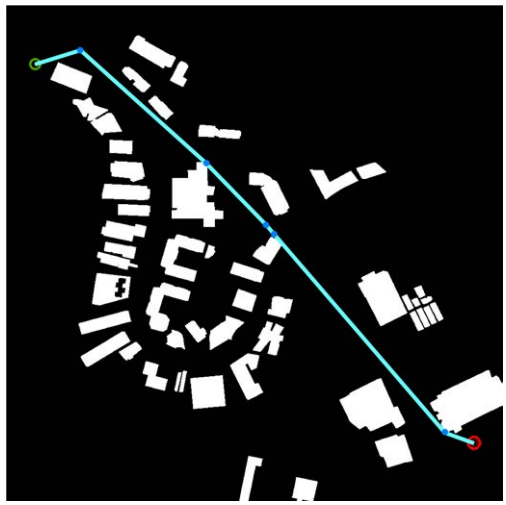
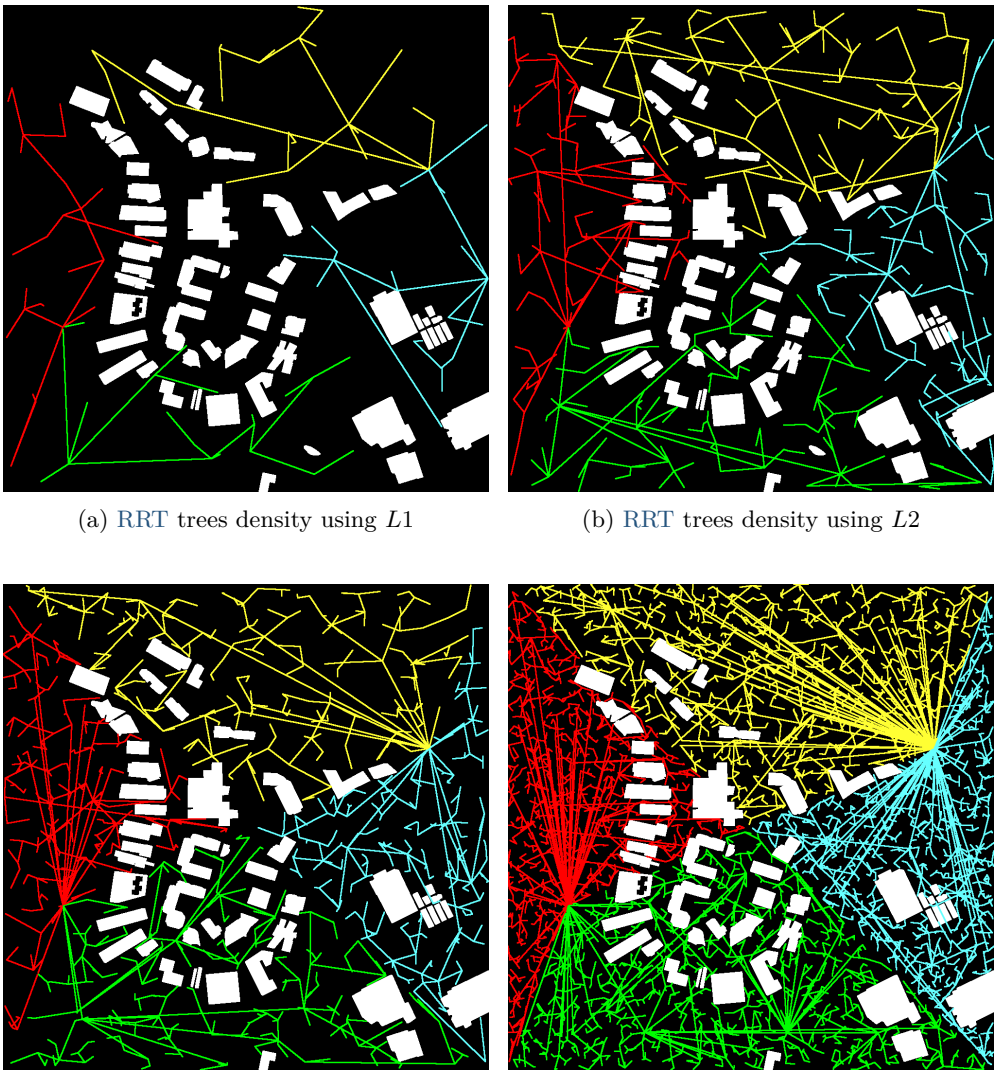


Figure 4.8: The calculated path using the quad-RRT algorithm over the normal-dense map.

4.1.4 Quad-RRT runtime analysis in the high-dense map

As in the subsection 4.1.2, in this subsection we present and analyze the results obtained by the quad-RRT over the scenarios described at Section 4.1.1, specifically for the high-dense map. Due to the proposal has been developed to be a fast planner, the runtime analysis focuses on the execution times of the algorithm, in this case over a high-dense map. Tables 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, and 4.20 show the obtained execution times. Each table is associated to a specific level of parallelism. In order to help understanding the data appearing in these tables, several graphs are also shown. The graphs at figures 4.10a-d group the obtained data taking into account the path direction (table 4.3) and the level of parallelism (table 4.5). Again, these results illustrate that the best performance is generally obtained when the level of parallelism $L4$ is used for computation. In the same way than in Subsection 4.1.2, when the number of threads grows, the execution time increases due to collisions between threads in the blocks.

Table 4.20 does not include data corresponding to the *L1*, *L2* and *L3* levels of parallelism, because these levels of parallelism do not guarantee



(a) RRT trees density using $L1$

(b) RRT trees density using $L2$

(c) RRT trees density using $L3$

(d) RRT trees density using $L4$

Figure 4.9: Example of the RRT trees density over the normal-dense map and depending on the selected level of parallelism.

finding a valid path between the source and the targets points in the map (see tables 4.14, 4.15 and 4.16).

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	171	not found	not found	not found
M size	not found	not found	not found	not found
L size	not found	not found	not found	not found
XL size	not found	not found	not found	not found

Table 4.14: Level of parallelism L1. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	202	165	212	187
M size	128	162	192	164
L size	283	242	not found	271
XL size	not found	502	not found	not found

Table 4.15: Level of parallelism L2. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	183	191	229	190
M size	150	192	208	205
L size	265	289	345	282
XL size	868	594	760	not found

Table 4.16: Level of parallelism L3. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

4.1.5 Quad-RRT graphical results obtained in the high-dense map

With the aim of providing a graphical description of how the quad-RRT works, several images corresponding to one of the executed tests are presented in this section. Specifically, the images correspond with the execution of the quad-RRT with a level of parallelism $L4$ and path direction MD. The s-sized map at table 4.2 is used in this example. The map con-

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	219	247	207	242
M size	200	220	232	203
L size	301	344	341	336
XL size	816	550	737	757

Table 4.17: Level of parallelism L4. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	408	385	398	394
M size	338	358	365	360
L size	484	458	628	378
XL size	779	821	891	801

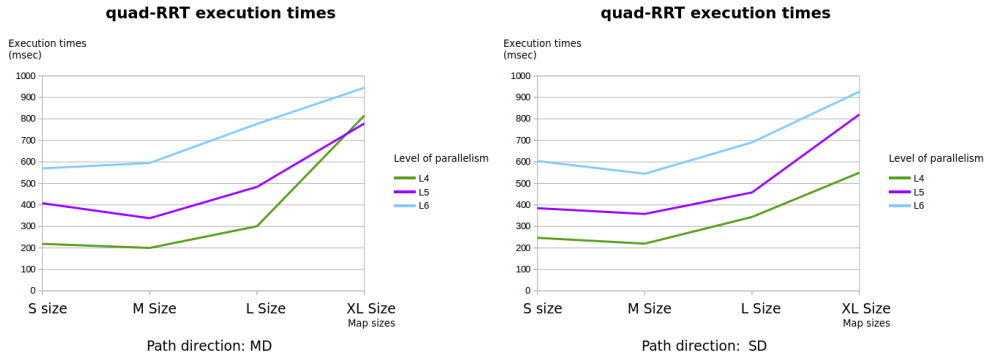
Table 4.18: Level of parallelism L5. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

Map Size	Time (MD) (msec)	Time (SD) (msec)	Time (HD) (msec)	Time (VD) (msec)
S size	570	604	605	585
M size	595	545	594	569
L size	777	691	732	635
XL size	945	927	1126	882

Table 4.19: Level of parallelism L6. quad-RRT execution times for the MD, SD, HD and VD routes in the high-dense map.

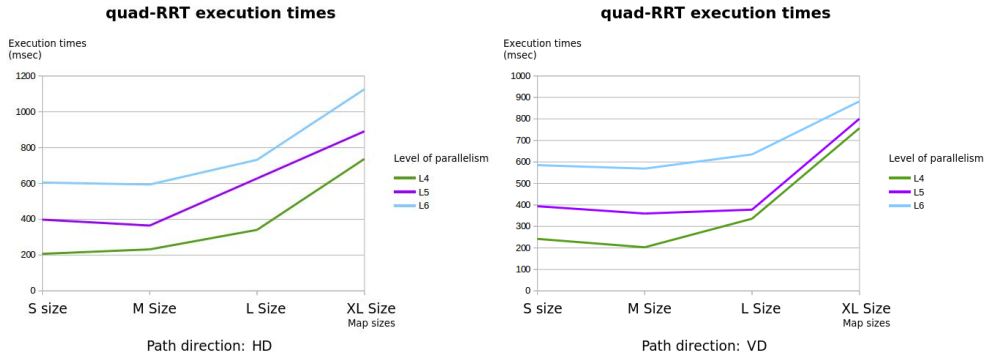
Level of Parallelism	S size	M size	L size	XL size	Total
L4	228.75	213.75	330.5	715	372
L5	396.25	355.25	487	823	515.375
L6	591	575.75	708.75	970	711.375

Table 4.20: Average execution times (ms), depending on the level of parallelism and the size of the maps. Test corresponding with the high-dense map.



(a) MD direction

(b) SD direction



(c) HD direction

(d) VD direction

Figure 4.10: Execution times for the calculation of the path in our four directions, using different levels of parallelism and map sizes. These graphs correspond with the experimental result obtained when using the high-dense map.

sists of a real cartography of the area illustrated at figure 4.2. Obstacles in the map have been marked and stored in a geographical data base. The total size of the map is 700 meters by 700 meters. The resolution of the map (minimum size of obstacles) is 1 meter $\times$ 1 meter. Figure 4.11 illustrates the map: obstacles are drawn in the map using the white color, meanwhile free space is drawn using black color.

Figure 4.12 shows that the execution of the quad-RRT algorithm builds four RRT trees, which grew through the map. Each RRT tree has been

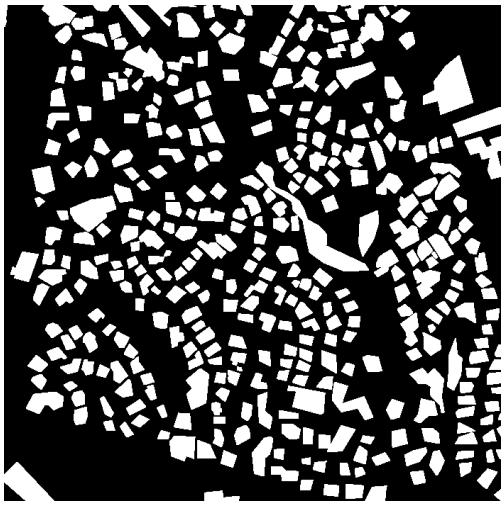


Figure 4.11: Binary representation of the real and normal-dense map of 700 meters by 700 meters. The area is located at Puerto de la Torre (Málaga, Spain).

drawn over the map using a different color. As depicted, the calculated RRT trees expand uniformly inside the map. This behavior indicates that the quad-RRT is correctly exploring the complete map. Figure 4.12 also shows that the final sizes (i.e. number of nodes) for the four trees are balanced. During this growing process, after adding a node to each tree, the matching process tries to find a path to travel from the source pose to the target one in a secure way. Figure 4.13 shows the finally calculated path.

In this case, the $L4$ parallelism level, which uses 25 threads per GPU block, is clearly the best configuration for the quad-RRT algorithm (in terms of execution time). Although the $L5$ and $L6$ configurations explore the map in a more exhaustive way than the $L4$ configuration, this last one explores the map about 100 milliseconds faster than the $L5$ configuration and between 200 milliseconds and 350 milliseconds (approximately) faster than the $L6$ configuration. Furthermore, the map exploration carried out by the algorithm when using the $L4$ configuration is uniform and pretty complete. It covers almost all the obstacle-free space and the resultant RRT trees are pretty balanced. Figure 4.14 shows the results obtained

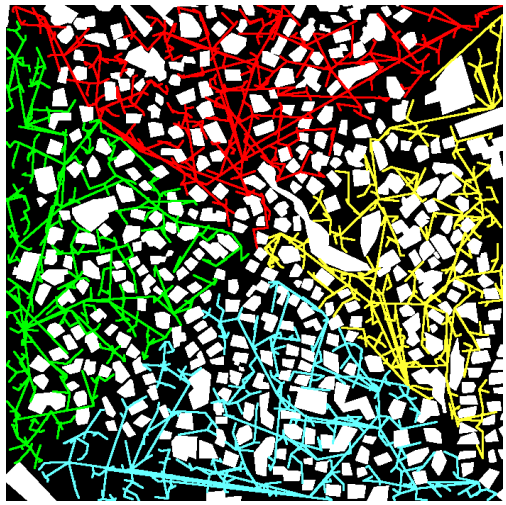


Figure 4.12: RRT trees generated by the quad-RRT algorithm in the high-dense map.

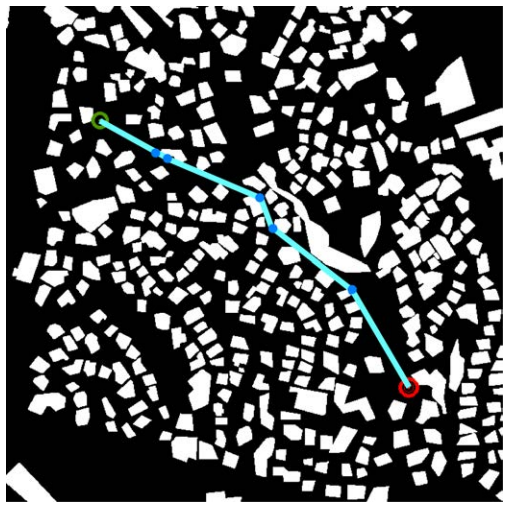
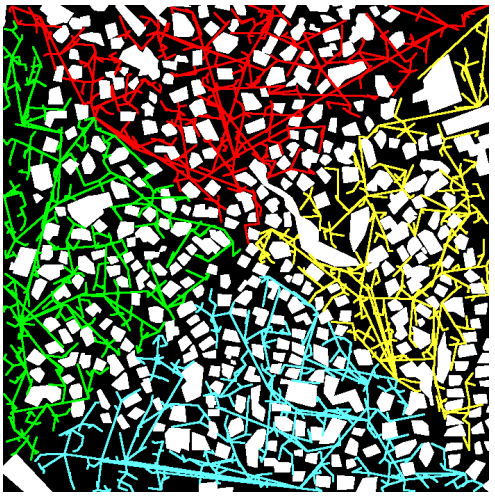
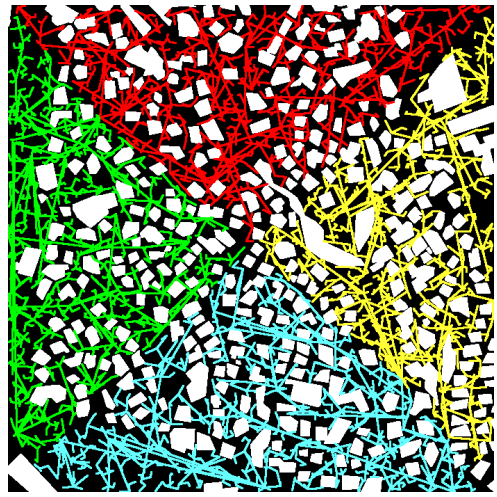


Figure 4.13: The calculated path using the quad-RRT algorithm over the high-dense map.

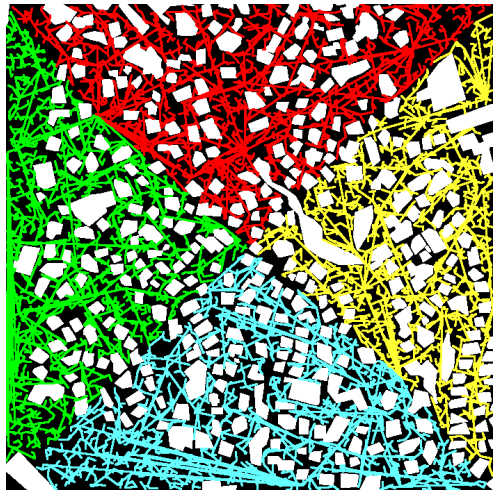
using the quad-RRT with the *L4*, *L5* and *L6* configurations respectively.



(a) RRT trees density using $L4$



(b) RRT trees density using $L5$



(c) RRT trees density using $L6$

Figure 4.14: Example of the RRT trees density over the high-dense map and depending on the selected level of parallelism.

4.1.6 Comparison to other approaches

Comparison with other RRT approaches has been conducted using the framework provided by K. Zhao and Y. Li's work [3]. In this work, the authors tested several RRT implementations over a specific map. Here, we have tested the quad-RRT approach using the same map and boundary points. Specifically, the map corresponds to an indoor environment. It has a size of 310 × 169 (Figure 4.15).

Figure 4.15: (Left) Indoor map used by K. Zhao and Y. Li [3], and (right) boundary points to plan a new path inside the map

The resulting RRT trees obtained after the execution of the quad-RRT algorithm illustrate that the algorithm correctly explores the available free space in the map (see figure 4.16). None of the RRT trees explore inaccessible map regions. Due to the location of the boundary points inside the map, and the policy followed by the quad-RRT algorithm to get map partitions (see section 3.4.1), one of the trees (the green colored

tree) grew up inside a very small map region (see figure 4.17).

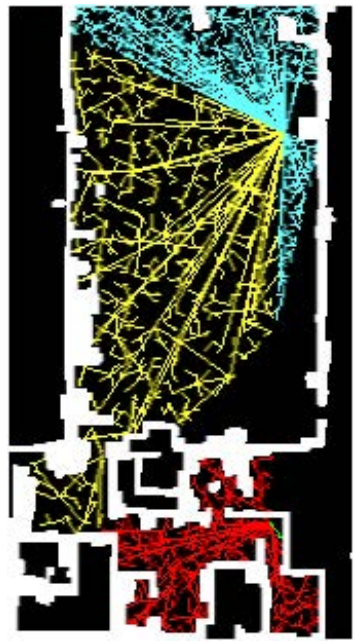


Figure 4.16: Resulting RRT trees of the quad-RRT algorithm execution on the K. Zhao and Y. Li's map

Table 4.21 resumes the execution times obtained by the RRT variants tested by K. Zhao and Y. Li [3], and the average execution time obtained by the proposed quad-RRT algorithm. These results show that the proposed algorithm is 9.98 times faster than the best alternative tested by K. Zhao and Y. Li and 206.86 times faster than the original RRT. As the BiasedGreedy-RRT and the Topological-RRT, the proposed algorithm does not produce any failure on finding a path (a failure is denoted when the number of samples exceeds 50,000). Moreover, being faster, the quad-RRT provides an average number of nodes that is greater than the one provided by the Topological-RRT. Hence, it performs a deeper exploration of the search space. Figure 4.18 shows one obtained path. It can be noted how the calculated path using the quad-RRT algorithm is always very close to the optimal solution provided by a RRT* approach.

Figure 4.17: A RRT tree inside a very small region of the map used by K. Zhao and Y. Li

Algorithm	Execution time (msec)	Fail times	Average nodes
RRT	81918	15	5875
Biased-RRT	59702	6	2606
Greedy-RRT	39499	3	7336
BiasedGreedy-RRT	28412	0	3123
Topological-RRT	3956	0	347
quad-RRT	**396**	**0**	**2652**

Table 4.21: Performance comparison taking into account the results presented in [3]

It should be noted that highly-parallelized algorithms have a better performance when they work with large datasets. In this case, the map used by K. Zhao and Y. Li is not big enough. Even so, the proposed quad-RRT algorithm is several times faster than the approaches tested by these

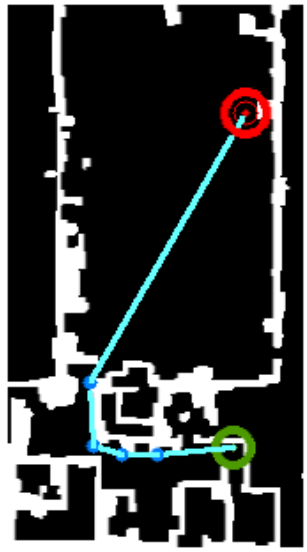


Figure 4.18: The calculated path over the map used by K. Zhao and Y. Li in [3]

authors [3]. K. Zhao and Y. Li do not provide the source code of their studied algorithms, so we cannot test these algorithms using our proposed datasets. For this reason, we cannot also compare the algorithms' memory consumption, but in our case and taking into account that each RRT tree node has 4 bytes of size, in this test the quad-RRT approximately occupies 41.4 Kbytes of memory as average. This is a reasonable memory consumption.

4.2 Discussion

The original B-RRT creates two trees, which grow until a state that is shared by both trees is found. This scheme was significantly sped up by replacing the incremental sampling by other heuristics, but also by those approaches that distribute the planning problem into subproblems. The quad-RRT is an approach designed to be embodied within a NVIDIA GPU, and aggressively employs these two premises (problem subdivision and GPU parallel processing) to provide a fast solution searching for a path within a large and populated environment. In fact, it also includes

a third premise to work even faster: each chosen node that cannot be directly linked to the tree structure is discarded. From a theoretical point-of-view, the main difference with other approaches is that the quad-RRT divides up the search space in several regions and grows a tree within each region on the map. This scheme could be extended to other n-dimensional state spaces by designing the adequate matching function. The connection process among trees is probably the bottleneck of the approach. Further work should address the design of a parallel mechanism to achieve it, specially when working on n-dimensional state spaces.

Table 4.22 summarizes the main technical advantages and disadvantages of the proposed approach. As the table shows, the proposed algorithm has only a few disadvantages. Taken into account the new hardware generation and the growing integration level in electronics, we can assert that these minor disadvantages will disappear in a very short period of time, so it is unlikely they have a significant impact in a near future.

Advantages
• Between 10 and 200 times (approximately) faster than other RRT-based approaches. • Deep and balanced map exploration. • Takes advantage of the multi-core parallelism. • Takes advantage of the GPGPU. • Feasible to be used within the most demanding applications.
Disadvantages
• More hardware requirements than other RRT-based approaches. • Slightly higher energy consumption (due to the use of the GPU computing techniques).

Table 4.22: Main advantages and disadvantages of the proposed quad-RRT algorithm

CHAPTER

5

Conclusions and future work

This doctoral dissertation presents innovative and revolutionary techniques to solve the PP problem in large-scale real maps. These techniques are materialized in the *quad-RRT* algorithm.

The main objective of the PP problem is to find valid trajectories between two different locations inside known or unknown environments. Although this is the main application of the PP problem, other frequent problems of modern robotics can be solved by applying PP algorithms. For example, PP algorithms could be applied to determine how a robotic arm should move to grab a specific object.

In this context, besides being a well-known problem in the robotics field, the PP problem is continuously under study, as it is an extremely complex algorithm to solve, which gives solutions to essential problems in an wide spectrum of applications. The importance of the PP problem and its complexity are the main motivations of this doctoral dissertation.

The PP problem belongs to the *NP-complete* problem category. Summarizing, the NP-complete problems constitute the most difficult problems of the *NP* category. The NP problems are those that cannot be solved in a polynomial computation time. In other words, the required computation time to solve a NP problem exponentially grows as its search space increases. This fact explains the complexity of the PP problem and

why the scientific community is interested in..

Nowadays robots are increasingly complex and they are able to carry out tasks that some years ago were unthinkable. Robots are even beginning to replace humans in housework. In fact, science points out that in a near future it is pretty probable that robots will be very similar to humans and, in general, to living beings. This level of similarity will begin to make sense when robots make intelligent decisions as fast and efficiently as humans do. But almost all decisions imply to move with the aim of interacting with the surrounding environment and, in this sense, the PP methods play a fundamental role.

Respect to the PP resolution, several methods exist. The first ones, which were developed using *traditional computation techniques*, are now obsolete: when they are applied to solve the PP problem in large-scale real maps they spent an amount of computation time that implies the robot start moving long after the motion order was issued. Hence, AMR using these PP methods are now useless robots.

In order to ease the effects of the traditional computation-based PP algorithms, *combinatorial optimization-based algorithms* were proposed. These methods are *metaheuristic-based algorithms*. One of the most representative metaheuristic-based algorithms are the Genetic Algorithms (GAs) and the EAs. Unlike traditional computation techniques (brute-force search or exhaustive search), evolutionary computation are probabilistic-based techniques that do not guarantee a valid solution nor the optimal solution at the end of the process. However, they have a high probability of providing a very close solution to the optimal one, in a reasonable time. Hence, these methods imply a lower computation time compared to the traditional computation-based methods. However, they are still very slow in order to satisfy the requirements of modern robots.

Overall, one of the main challenges in modern robotics is to minimize the computation time spent in calculations, offering similar or even better solutions than traditional techniques. The modern *multi-core processors*, the GPUs, and the *parallel-based computation techniques* are the key elements to achieve these objectives.

Therefore, the tendency in PP is to use parallel adaptations of the traditional computation-based techniques, parallel versions of metaheuristic-

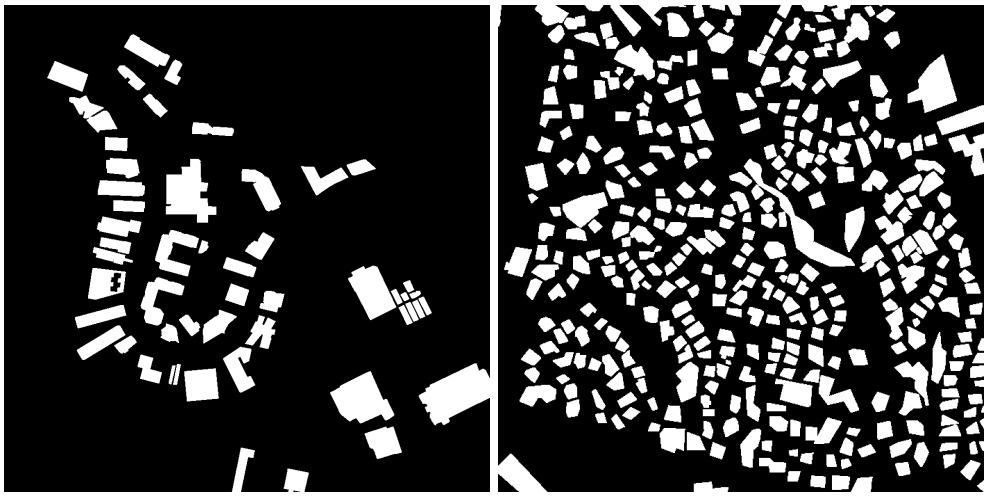
based algorithms, or parallel versions of specific heuristics-based algorithms. It is important to note that the hardware supporting parallelism is constantly evolving and so improvements of the existing parallel methods to solve the PP problem are frequently arising.

One of the most popular metaheuristic-based algorithms in PP is the RRT algorithm. The RRT algorithm iteratively generates an exploring tree (starting from the source point of the desired path), that uniformly grows along the free space of the environment with the aim of finding a valid trajectory connecting a source and a target point in a specific environment. In last years, several improvements and parallel versions based in the original RRT have been proposed. For example the RRT* or the B-RRT*.

This doctoral dissertation applies a mix of parallel programming techniques to improve the RRT existing methods. The resulting method is named *quad-RRT*. The quad-RRT applies two types of parallelism. On the one hand, a configurable layer running several threads over a multicore CPU (int this case running up to eight threads due to the CPU is composed by four cores with *HyperThreading* technology), to quickly examine the map of the environment with the aim of delivering several map partitions (specifically four partitions obtained by using an intelligent partition algorithm) and their respective structured representations to the lowest level computation layer. This lowest layer is a GPGPU-based software, that exploits parallelism in a massive way using the thousand of cores that compose the GPU subsystem. This software is responsible of simultaneously running the original RRT algorithm in each map partition, so each RRT tree grows in a parallel way.

In order to test the proposed algorithm, several scenarios of large-scale real maps with different object density (complexity of the environment) have been chosen to demonstrate the algorithm effectiveness, and deduce the best parallel configuration in a generalist approach. The easiest scenario corresponds with a normal-dense map (see figure 5.1a), in which few objects are included. The most complex scenario corresponds with a very high-dense map (see figure 5.1b), in which lots of objects are present, in comparison with the obstacle-free space.

The proposed tests measured the execution times using large-scale real



(a) Normal-dense map. (b) Very high-dense map.

Figure 5.1: (a) The normal-dense map and (b) the very high-dense map used for testing the quad-RRT algorithm.

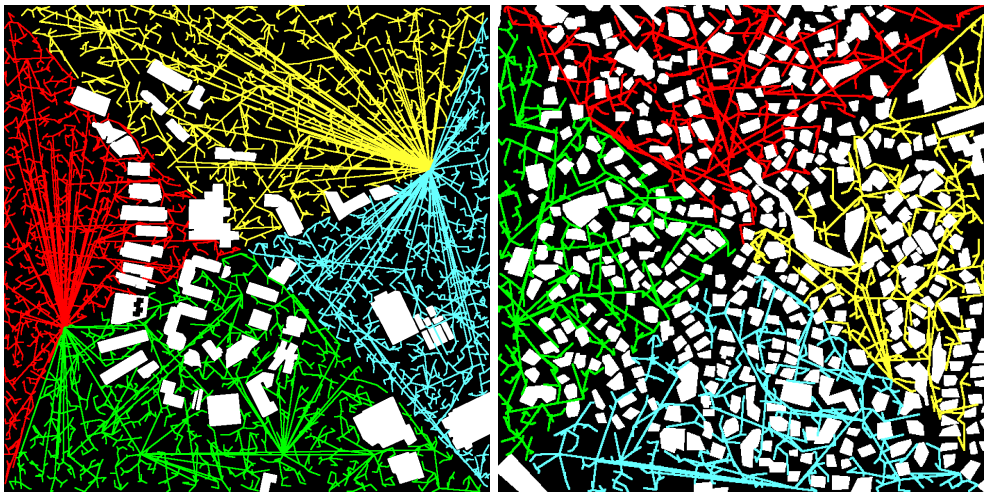
maps of different sizes combined with different parallel configurations (see table 4.5) for the GPGPU-based software (see chapter 4). The tests demonstrated that for both maps, the normal-dense one and the very high-dense one, the best configuration was the $L4$ (see sections 4.1.2, 4.1.3, 4.1.4 and 4.1.5 and Figures 4.5 and 4.10), in which 4 GPU blocks, one per map partition, and 25 threads per GPU block, are used. Parallel levels from $L1$ to $L3$ were discarded because in some cases, depending on the map size, when using the very high-dense map the desired path was not found (see tables 4.14, 4.15 and 4.16). The remainder parallel configurations ($L5$ and $L6$) always found valid and pretty good solutions. However, the obtained solutions using these parallel configurations were not better than the obtained ones using the $L4$ configuration. In all cases, the obtained solutions were visually optimal or very close to the optimal one. Besides, the average execution times using $L5$ and $L6$ configurations were increased in comparison with the obtained ones using the $L4$ configuration, due to issues regarding threads synchronization and collisions between threads in a GPU block (see tables 4.13 and 4.20). For these

reasons, $L5$ and $L6$ parallel configurations were also discarded.

In order to demonstrate the goodness, the effectiveness and the efficiency of the quad-RRT algorithm, a comparison among the proposed algorithm and other well-known RRT-based approaches was made (see section 4.1.6). This comparison is based on the results published in a previous study focused in solving the PP problem in large-scale indoor maps which was carried out by K. Zhao and Y. Li [3]. In this previous study, K. Zhao and Y. Li compared different RRT-based proposals using an specific dataset (scenario). In this Thesis, the quad-RRT was executed using the same scenario that K. Zhao and Y. Li used in their study. The presented comparison in this doctoral dissertation demonstrates that the quad-RRT is between 9.98 and 206.86 times faster than the existing RRT-based algorithms (see table 4.21).

Finally, the tests executed in this doctoral thesis demonstrate that the *RRT trees*, generated by the quad-RRT algorithm, uniformly grow through the obstacle-free space, exploring the safe areas for the robot (see figure 5.2). More precisely, the examples in section 3.6 (figures from 3.11 to 3.23, and from 3.24 to 3.36), show step by step executions that graphically demonstrate how uniform RRT trees obtained by the quad-RRT algorithm are.

For future work, several aspects of the quad-RRT could be modified to probably improve the algorithm performance. On the one hand, the number of map partitions could be increased (as many partitions as GPU blocks) in order to grow a considerable number of RRT trees at the same time. Each RRT tree will explore a little area of the environment in an exhaustive way (probably using a lower number of threads than the ones used in the parallel configuration of this proposal, this is 25 threads per GPU block). Hence, the matching process, that checks for safe connections among all the RRT trees in a massive parallel way, should be modified to focus on a more complex procedure. Besides, to increase the number of trees, the smart map partition algorithm must also be modified to split the input map into a higher number of similar partitions (balanced partitions, in terms of size), large enough to take full advantage of the GPUs resources, accomplishing thus a massively parallel processing of greater magnitude than the current one. If this modification of



(a) RRT trees in the normal-dense map. (b) RRT trees in the very high-dense map.

Figure 5.2: (a) The RRT trees expansion in the normal-dense map and (b) the RRT trees expansion in the very high-dense map. Both images correspond to the map exploration carried out by the quad-RRT algorithm when using the level of parallelism $L4$ (see table 4.5).

the algorithm could be successfully implemented, the quad-RRT could deliver valid paths in lower computation times, further increasing the performance of the algorithm.

Glossary

AE Algoritmo Evolutivo.

AI Artificial Intelligence.

AMR Autonomous Mobile Robot.

APF Artificial Potential Fields.

API Application Programming Interface.

B-RRT Bidirectional-RRT.

B-RRT* Bidirectional-RRT*.

C-FOREST Coupled Forest of Random Engrafting Search Trees.

CL-RRT Closed Loop RRT.

CPU Central Processing Unit.

CPU Unidad Central de Procesamiento.

CUDA Compute Unified Device Architecture.

EA Evolutionary Algorithm.

ERRT Execution extended RRT.

FMM Fast Marching Method.

FPGA Field Programmable Gate Array.

GA Genetic Algorithm.

GPGPU General-Purpose Computing on Graphics Processing Units.

GPGPU Computación de Propósito General en Unidades Gráficas de Procesamiento.

GPS Global Positioning System.

GPU Graphics Processing Unit.

GPU Unidad Gráfica de Procesamiento.

HPC High Performance Computing.

IA Inteligencia Artificial.

MOEA Multi-Objective Evolutionary Algorithm.

MOSFLA-MRPP Multi-Objective Shuffled Frog-Leaping Algorithm applied to Mobile Robot Path Planning.

PP Path Planning.

PRM Probabilistic RoadMaps.

PRRT Parallel RRT.

PRRT* Parallel RRT*.

RRT Rapidly-exploring Random Tree.

RRT* RRT star.

SFLA Shuffled Frog-Leaping Algorithm.

SRT Sampling-based RoadMap of Trees.

UAV Unmanned Aerial Vehicle.

Bibliography

[1] S. Kim and D. Folie, *The group marching method: An O(N) level set eikonal solver*, pp. 2297–2300. 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] K. Zhao and Y. Li, "Path planning in large-scale indoor environment using RRT," in *Proceedings of the 32nd Chinese Control Conference*, pp. 5993–5998, 2013.

[4] G. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, N. Robert, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood, "The grand challenges of Science Robotics," vol. 3, no. 14, 2018.

[5] J. Andréu, F. Deligianni, D. Ravì, and G. Yang, "Artificial Intelligence and Robotics," *CoRR*, vol. abs/1803.10813, 2018.

[6] D. Acemoglu and P. Restrepo, "Artificial Intelligence, Automation and Work," Tech. Rep. 24196, National Bureau of Economic Research, January 2018.

[7] M. Spenko, S. Buerger, and K. Iagnemma, *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Springer Tracts in Advanced Robotics, Springer International Publishing, 2018.

[8] A. Khan, I. Noreen, and Z. Habib, "On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges," *Journal of Information Science and Engineering*, vol. 33, pp. 101–121, 2017.

[9] M. Maiterth, T. Wilde, D. Lowenthal, B. Rountree, M. Schulz, J. Eastep, and D. Kranzlmiiller, "Power Aware High Performance Computing: Challenges and Opportunities for Application and System Developers - Survey Tutorial," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 3–10, July 2017.

[10] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering*, vol. 2016, p. 22, Mar. 2016.

[11] E. Masehian and D. Sedighizadeh, "Multi-Objective PSO- and NPSO-based Algorithms for Robot Path Planning," *Advances in Electrical and Computer Engineering*, vol. 10, no. 4, pp. 69–76, 2010.

[12] E. Masehian and D. Sedighizadeh, "Multi-objective robot motion planning using a particle swarm optimization model," *Journal of Zhejiang University SCIENCE C*, vol. 11, no. 8, pp. 607–619, 2010.

[13] E. Masehian and D. Sedighizadeh, "A multi-objective pso-based algorithm for robot path planning," in *Industrial Technology (ICIT), 2010 IEEE International Conference on*, pp. 465–470, March 2010.

[14] S. Geetha, G. M. Chitra, and V. Jayalakshmi, "Multi objective mobile robot path planning based on hybrid algorithm," in *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, vol. 6, pp. 251–255, April 2011.

[15] H. Wei and Q. Shiyin, "Multi-objective Path Planning for Space Exploration Robot Based on Chaos Immune Particle Swarm Op-

timization Algorithm," in *Artificial Intelligence and Computational Intelligence*, pp. 42–52, 2011.

[16] F. Guo, H. Wang, and Y. Tian, "Multi-objective path planning for unrestricted mobile," in *Automation and Logistics, 2009. ICAL '09. IEEE International Conference on*, pp. 1046–1051, Aug 2009.

[17] P. S. Krishnan, J. K. S. Paw, and T. S. Kiong, "Cognitive map approach for mobility path optimization using multiple objectives genetic algorithm," in *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*, pp. 267–272, Feb 2009.

[18] A. Hidalgo-Paniagua, M. Vega-Rodríguez, J. Ferruz, and N. Pavón, "Solving the multi-objective path planning problem in mobile robotics with a firefly-based approach," *Soft Computing*, vol. 21, pp. 949–964, Feb 2017.

[19] A. Hidalgo-Paniagua, M. A. Vega-Rodríguez, J. Ferruz, and N. Pavón, "MOSFLA-MRPP: Multi-Objective Shuffled Frog-Leaping Algorithm applied to Mobile Robot Path Planning," *Engineering Applications of Artificial Intelligence*, vol. 44, pp. 123–136, 2015.

[20] A. Hidalgo-Paniagua, M. A. Vega-Rodríguez, and J. Ferruz, "Applying the MOVNS (Multi-Objective Variable Neighborhood Search) algorithm to solve the path planning problem in mobile robotics," *Expert Systems with Applications*, vol. 58, pp. 20–35, 2016.

[21] J. C. Latombe, *Robot Motion Planning*. The Springer International Series in Engineering and Computer Science, Springer US, 2012.

[22] J. C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[23] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, *Path Planning and Trajectory Planning Algorithms: A General Overview"*, pp. 3–27. Springer International Publishing, 2015.

[24] F. Bayat, S. Najafinia, and M. Aliyari, "Mobile robots path planning: Electrostatic potential field approach," *Expert Systems with Applications*, vol. 100, pp. 68 – 78, 2018.

[25] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1398–1404, Apr 1991.

[26] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[27] M. Ben-Ari and F. Mondada, *Mapping-Based Navigation*, pp. 165–178. Springer International Publishing, 2018.

[28] E. W. Dijkstra, "A Note on Two Problems in Connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, December 1959.

[29] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pp. 38–47, July 2007.

[30] K. Ok, S. Ansari, B. Gallagher, W. Sica, F. Dellaert, and M. Stilman, "Path planning with uncertainty: Voronoi Uncertainty Fields," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 4596–4601, May 2013.

[31] M. H. Overmars and E. Welzl, "New Methods for Computing Visibility Graphs," in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, (New York, NY, USA), pp. 164–171, ACM, 1988.

[32] H. Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113–126, Oct 2001.

[33] P. J. Frey and P. L. George, *Mesh Generation: Application to Finite Elements*. ISTE, Wiley, 2010.

[34] P. J. Frey and P. L. George, *Delaunay-based Mesh Generation Methods*, pp. 235–273. ISTE, 2010.

[35] N. Hiroshi, N. Tomohide, and A. Suguru, "A quadtree-based path-planning algorithm for a mobile robot," *Journal of Robotic Systems*, vol. 7, no. 4, pp. 555–574, 1990.

[36] D. Eppstein, M. T. Goodrich, and J. Z. Sun, "The skip quadtree: a simple dynamic data structure for multidimensional data," in *In Proc. 21st ACM Symposium on Computational Geometry*, pp. 296–305, ACM, 2005.

[37] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[38] C. W. Warren, "Global path planning using artificial potential fields," in *Proceedings, 1989 International Conference on Robotics and Automation*, vol. 1, pp. 316–321, May 1989.

[39] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," tech. rep., Computer Science Department, Iowa State University, 1998.

[40] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[41] B. Akgun and M. Stilman, "Sampling Heuristics for Optimal Motion Planning in High Dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2640–2645, Sept 2011.

[42] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.

[43] I. Noreen, A. Khan, and Z. Habib, "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, pp. 97–107, 11 2016.

[44] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[45] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot Motion Planning on a Chip," in *Proceedings of Robotics: Science and Systems*, 2016.

[46] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3513–3518, 2011.

[47] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT*: A Real-time Path Planning Algorithm Based on RRT*," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pp. 113–118, 2015.

[48] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, 2014.

[49] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[50] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2383–2388, 2002.

[51] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How, "Bounds on tracking error using closed-loop rapidly-exploring random trees," in *Proceedings of the 2010 American Control Conference*, pp. 5406–5412, 2010.

[52] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[53] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 995–1001, 2000.

[54] M. Otte and N. Correll, "C-FOREST: Parallel Shortest Path Planning With Superlinear Speedup," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 798–806, 2013.

[55] J. Ichnowski and R. Alterovitz, "Scalable Multicore Motion Planning Using Lock-Free Concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.

[56] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

[57] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[58] J. J. Kuffner and S. M. LaValle, "An efficient approach to path planning using balanced bidirectional RRT search," tech. rep., Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[59] M. W. Otte, *Any-com multi-robot path planning*. PhD thesis, University of Colorado at Boulder, 2011.