Universidad
Carlos III de Madrid
www.uc3m.es

TESIS DOCTORAL

Dexterous Robotic Motion Planning using Intelligent Algorithms

Autor:

César Augusto Arismendi Gutiérrez

Director:
Luis Enrique Moreno Lorente

Codirector:
Luis Santiago Garrido Bullón

Tutor:
Luis Enrique Moreno Lorente

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
UNIVERSIDAD CARLOS III DE MADRID

Leganés, Julio 2015

Universidad
Carlos III de Madrid
www.uc3m.es

TESIS DOCTORAL

DEXTEROUS ROBOTIC MOTION PLANNING USING INTELLIGENT ALGORITHMS

Autor: *César Augusto Arismendi Gutiérrez*

Director: Luis Enrique Moreno Lorente

Codirector: Luis Santiago Garrido Bullón

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, 16 de Julio de 2015

Abstract

The fundamental purpose of robots is to help humans in a variety of difficult tasks, enabling people to increase their capabilities of strength, energy, speed, memory, and to operate in hazardous environments and many other applications. Service robots, more precisely mobile manipulators, incorporate one or two robotic arms and a mobile base, and must accomplish complex manipulations tasks, interacting with tools or objects and navigating through cluttered environments. To this end, the motion planning problem plays a key role in the ahead calculation of robot movements to interact with its world and achieve the established goals. The objective of this work is to design various motion planning methods in order to improve the autonomy of MANFRED-2, which is a mobile robot fully developed by the Robotics Lab research group of the Systems Engineering and Automation Department of the Carlos III University of Madrid.

Mobile robots need to calculate accurate paths in order to navigate and interact with objects throughout their surrounding area. In this work, we have developed motion planning algorithms for both navigation and manipulation. The presented algorithms for path planning are based on the Fast Marching Square method and include a replanner with subgoals, an anytime triangular planner, and a nonholonomic approach. The replanner with subgoals starts by generating a smooth and safe global path with the Fast Marching Square method. Then, this path is divided into multiple subpaths separated by equidistant nodes (defined by topological or metric constraints). After that, the obstacles information is progressively added and modifications are made only when the original path is unreachable. The most important advantage with respect to similar approaches is that the generated sub-paths are always efficient in terms of smoothness and safeness. Besides, the computational cost is low enough to use the algorithm in real-time. The anytime triangular planner, such as "Anytime" algorithms, quickly finds a feasible but not necessarily optimal motion plan which is incrementally improved. One important characteristic that this type of algorithms must satisfy is that the path must be generated in real-time. The planner relies on the Fast Marching Square method over a triangular mesh structure. Different variants are introduced and compared under equal circumstances that produce different paths in response time and quality, which leads us to an additional

consideration. As in the field of benchmarking it is becoming increasingly difficult to compare new planners approaches because of the lack of a general benchmarking platform, improvements to existing approaches are suggested. Finally, the nonholonomic approach is presented. It is based on the Fast Marching Square method and its application to car-like robots. In order to apply the proposed method, a three dimensional configuration space of the environment is considered. The first two dimensions are given by the position of the robot, and the third one by its orientation. This means that we operate over the configuration space instead of the bi-dimensional environment map. Besides, the trajectory is computed along the configuration space taking into account the dimensions of the vehicle. In this way, it is possible to guarantee the absence of collisions. The proposed method is consistent at local and global scale because it guarantees a motion path solution, if it exists, and does not require global replanning supervision when a local trap is detected.

Once a mobile robot has reached a goal location, it usually triggers the servomotors enclosed inside its robotic arm to manipulate a target. The manipulation algorithms presented in this work include the adaptation of trajectories, a planner with adaptive dimensionality, and an implementation of a dimensionality reduction approach inside a nuclear device. The adaptation of manipulation trajectories enables the robot to accomplish a task in different locations by using Evolution Strategies and forward kinematics. This approach avoids all the inconveniences that inverse kinematics imply, as well as the convergence problems in singular kinematic configurations. The planner with adaptive dimensionality reduces the complexity of high-dimensional path planning. First, a Rapidly-exploring Random Tree trajectory is generated using the full degrees of freedom of the robotic arm. Then, a geometry as a closed tube is built around the path line and the Fast Marching Square method is applied from start to goal using three dimensions inside the surface. The resulted three dimensional path is converted to full degrees of freedom with the inverse kinematics of the robot. The result is a smoother and safer path, visually more human friendly. Additionally, the search space is reduced, and therefore, also the planning time and the memory requirements. The application inside the nuclear device, similarly to the previous approach, reduces the degrees of freedom of the problem (but this time to two dimensions due to the mostly planar nature of the robot). The manipulation path is smooth and safe in an environment where safety must be the primarily objective. The motion planning algorithms have been tested in numerous experiments. The fast response of the methods allows its application in real-time tasks.

Resumen

El propósito fundamental de los robots es ayudar a los humanos en tareas difíciles, lo que permite a las personas incrementar sus capacidades de fuerza, energía, velocidad y memoria para trabajar en entornos peligrosos y en una inmensa variedad de aplicaciones. Los robots de servicio, puntualmente los manipuladores móviles, incorporan uno o dos brazos robóticos y una base móvil, y deben ser capaces de realizar tareas complejas de manipulación, interactuando con herramientas u objetos y navegando a través de entornos con obstáculos. Para este fin, el problema de la planificación de movimientos juega un rol clave en el cálculo anticipado de los movimientos del robot, para interactuar con su mundo y realizar las tareas establecidas. El objetivo de este trabajo es diseñar diversos métodos de planificación de movimiento con el fin de mejorar la autonomía de MANFRED-2, un robot móvil que fue desarrollado completamente en el grupo de investigación del Laboratorio de Robótica del Departamento de Ingeniería de Sistemas y Automatización de la Universidad Carlos III de Madrid.

Los robots móviles necesitan calcular de antemano trayectorias precisas para poder navegar e interactuar con objetos en su entorno. En este trabajo, hemos desarrollado algoritmos de planificación de movimiento para navegación y manipulación robótica. Los algoritmos presentados para la planificación de trayectorias de navegación se basan en el método de Fast Marching Square (FM^2) e incluyen un replanificador con sub-objetivos, un planificador triangular de tipo interrumpible (en inglés este enfoque es mejor conocido como *Anytime*), y un enfoque no holonómico. El replanificador con submetas comienza generando una trayectoria global de curvas suaves y segura con FM^2, entonces este camino es dividido en múltiples subtrayectorias separadas por nodos equidistantes (definidos por restricciones topológicas o métricas). Después de esto, se actualiza progresivamente el entorno con obstáculos detectados por los sensores; solo se realizan cambios cuando la trayectoria original resulta inalcanzable. La ventaja más importante con respecto a enfoques similares es que las sub-trayectorias generadas son siempre eficientes en términos de suavidad y seguridad. Además, el coste computacional es lo suficientemente bajo como para utilizar el algoritmo en tiempo real. El planificador triangular interrumpible, como algoritmo "Anytime", encuentra rápidamente una trayectoria de navegación válida, pero no necesariamente

óptima, a continuación, de forma incremental se va mejorando según haya tiempo hasta llegar al óptimo. La capacidad más resaltante de este tipo de algoritmos es la de generar trayectorias en tiempo real. El planificador se basa en el uso de FM^2 sobre una estructura de malla triangular. Se presentan diferentes formas de construir el mallado y se comparan en igualdad de circunstancias los diferentes caminos producidos en tiempo de respuesta y calidad, lo que generó una contribución adicional. Debido a la falta de una plataforma general de evaluación robusta, en el campo de la evaluación de trayectorias es cada vez más difícil comparar nuevos planificadores, por consiguiente se sugieren mejoras a los enfoques existentes. Finalmente, se presenta el enfoque no holónomo, que se basa en FM^2 y su aplicación en robots móviles con sistemas de dirección similares a la de los coches. Para aplicar el método propuesto, se considera un espacio de configuración tridimensional del entorno, donde las dos primeras dimensiones vienen dadas por la posición del robot y la tercera dimensión, por su orientación. Esto quiere decir, que operamos en el espacio de configuraciones en vez de en el mapa bidimensional del entorno. Además, la trayectoria se calcula en el espacio de configuraciones teniendo en cuenta las dimensiones del vehículo, de esta manera es posible garantizar la ausencia de colisiones. El método propuesto es consistente a nivel local y global, ya que si existe una solución se garantiza encontrarla, y no requiere de supervisión global para reiniciar una planificación cuando se detecta un bloqueo a nivel local.

Una vez que el robot móvil ha alcanzado la ubicación requerida, se suelen accionar los servomotores que están dentro del brazo robótico para manipular un objeto. Los algoritmos de manipulación presentados en este trabajo incluyen la adaptación de trayectorias, un planificador con dimensionalidad adaptable, y una implementación de un método de reducción de la dimensionalidad dentro de un dispositivo nuclear. La adaptación de las trayectorias de manipulación permite al robot realizar una misma tarea con diferentes ubicaciones y orientaciones haciendo uso de una Evolution Strategy y la cinemática directa del robot, este enfoque evita todos los inconvenientes que implican utilizar la cinemática inversa, así como los problemas de convergencia en configuraciones cinemáticas singulares. El planificador con dimensionalidad adaptativa reduce la complejidad de la planificación de trayectorias de manipulación con muchas dimensiones; en primer lugar, una trayectoria RRT se genera utilizando todos los grados de libertad (DOF) del brazo robótico, a continuación, una figura geométrica en forma de tubo cerrado se construye alrededor de la línea de la trayectoria y se aplica FM^2 dentro de la superficie desde el inicio hasta el objetivo utilizando tres dimensiones, la ruta 3D resultante se convierte con la cinemática inversa del robot. El resultado es un camino más suave y seguro, más amigable a la vista humana. Además, debido a que el espacio de búsqueda se reduce, también se reduce el tiempo de planificación y los requerimientos de memoria. La aplicación en el interior del dispositivo nuclear, de manera similar al enfoque anterior, reduce los DOF del problema pero

esta vez a dos dimensiones aprovechando la naturaleza mayormente plana del robot utilizado. La trayectoria de manipulación es suave y segura, lo que es conveniente en un entorno donde la seguridad debe ser el objetivo principal. Los algoritmos de planificación de movimiento resultantes han sido probados en numerosos experimentos. La respuesta rápida de los métodos permite su aplicación en tiempo real.

A mi esposa y familia.

Agradecimientos

Gracias primero a Dios por darme la vida y todas las posibilidades que ha puesto en mi camino. Un agradecimiento muy especial a mi querida familia por siempre apoyarme y darme su cariño incondicionalmente.

A mi amada esposa le dedico mi esfuerzo y mi alegría, creo que no existen palabras para expresar el agradecimiento y el amor que siento hacia ella. Siempre será el ángel que cuida de mis sueños y me da fuerzas para siempre seguir adelante.

Muchísimas gracias a mis tutores por enseñarme tantas cosas y asesorarme durante todo este tiempo. Gracias por ayudarme, escucharme y darme ánimos. Han sido más que unos tutores, unos mentores y amigos que siempre consideraré en alta estima. Este trabajo ha sido un viaje espectacular y ustedes han sido los mejores compañeros.

A mis compañeros del roboticslab muchas gracias por compartir conmigo cada día, por estar siempre dispuestos a escucharme y a echarme una mano si era necesario. Muchas gracias por ser tan buenos amigos.

A la Universidad Carlos III de Madrid por acogerme durante todo este tiempo. Gracias particularmente a todos los secretarios por ser siempre tan amables y estar dispuestos a asesorarme en cualquier trámite.

En el entorno científico quiero agradecerle a Pedro Lima por permitirme trabajar con él en el ISR en Lisboa (Institute for Systems and Robotics), donde bajo su dirección realicé una estancia de dos meses en la que aprendí y progresé bastante.

A los investigadores que desarrollan software y suben sus librerías robóticas a repositorios públicos. Gracias por compartir vuestro valioso conocimiento y facilitarnos un poquito la vida a todos.

Abbreviations

2D - two-dimensional
3D - three-dimensional
6D - six-dimensional
ATFM2 - Anytime Triangular Fast Marching Square Method
CCD - Charge-Coupled Device
DAC - Digital-to-Analog Converters
DE - Differential Evolution
DOF - Degrees of Freedom
DSP - Digital Signal Processor
EA - Evolutionary Algorithms
ES - Evolution Strategy
FK - Forward Kinematics
FM^2 - Fast Marching Square
FM^2-NH - Fast Marching Square Nonholonomic
FMM - Fast Marching Method
GA - Genetic Algorithms
GPS - Global Positioning System
IK - Inverse Kinematics
KNN - K-Nearest Neighbours
LSM - Level Set Methods
NN - Nearest Neighbor
PLC - Programmable Logic Controller
R&D - Research and Development
RRT - Rapidly-exploring Random Tree
RRT-Bidirectional or RRT-Connect - Rapidly-exploring Random Tree Bidirectional
UC3M - Carlos III University of Madrid
VFM - Voronoi Fast Marching Method
VRML - Virtual Reality Modeling Language

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the moment the word robot was introduced by the Czech brothers Čapek, the human imagination flew years ahead. A future, where robots were able to work and actuate as humans side by side was envisioned. The year was 1920 when the transition of the industrial revolution was almost completed. A rural economy primarily based on agriculture switched to an urban economy, industrialized and mechanized. The hand production methods went to new machines, chemicals manufacturing and iron production processes.

The writer Karel Čapek introduced the word robot in his play *R.U.R. (Rossum's Universal Robots)*. It was his brother, Josef Čapek, who proposed the word "roboti". The word comes from robota, which in Czech means "hard work" or, in general, "work". In the play *R.U.R.*, simplified people was created from a chemical protoplasm process. They were very efficient but emotionless and lacked of original thinking. The technology behind the robots is not described but modern concepts as androids are depicted. The issue of whether the robots are being exploited is formulated, as well as the fear of more advanced robots rising up against humans.

Even though the technologies in these stories were science fiction for those times, one can imagine how all these futuristic ideas were taken for granted. A not very far future with human-like robots capable of performing any human task was foreseen. Although these capabilities are not still a reality, many advances have been made and robotics is a extensively studied and researched field. An example of how robotics has evolve can be appreciated in Figure 1.1, where the robotics developments of Honda [1] since 1986 to date are shown.

Figure 1.1: Honda historic evolution of robot development.

One of the first difficulties that researchers had come to realize is the complexity involved in robotics. To reproduce in robots some human tasks could result immensely complex. To develop a human-like robot, or more specifically an autonomous robot, many capabilities have to be included. Primarily, the perception and interaction with the environment. In order to perceive the world, the robot must be provided with some kind of sensors such as cameras, distance range lasers, and others such as force

[1] `http://world.honda.com/ASIMO/history/`

sensors and microphones for more specific tasks. For the interaction, actuators are needed to navigate through the environment and interact with objects, *e.g.* through a robotic manipulator. Many different fields are involved in robotics, such as electronics, software development, mechanics, mathematics, physics, and even psychology and speech synthesis and recognition in the case of social robots.

Motion planning is among the most important research areas in robotics. It results essential in order to navigate, physically interact or execute any kind of movement. Path planning for robots has been studied extensively over the last three decades. Large is the amount of planners that have been developed.

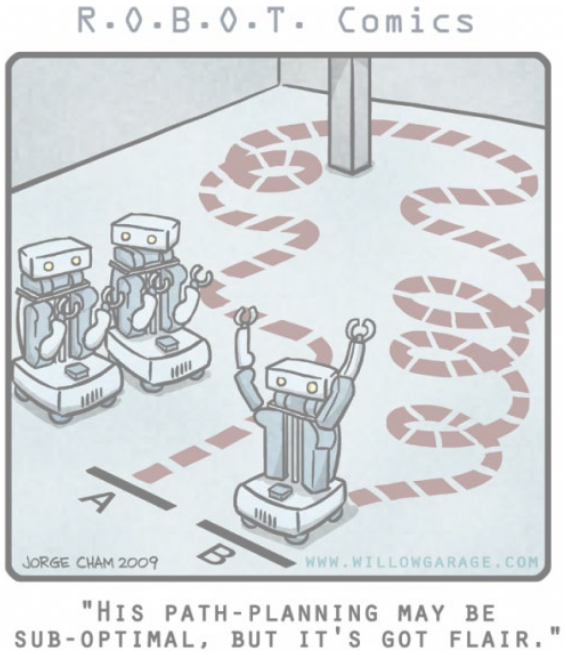


Figure 1.2: A path planning comic.

The main difficulty in motion planning is the complexity including variables such as time and control parameters. For example, finding an optimal trajectory in three dimensions with polyhedral obstacles and bounded acceleration and velocity has been proven to be Non-deterministic Polynomial-time hard or NP-hard (Donald [1]). In computational complexity theory, the NP-hard are the most complex kind of problems. For this reason, most of the planners bias to find a path instead of a trajectory (which includes time depending variables). Also, control parameters and additional restrictions are not taken into account or loosen. The complexity of the problem grows as dimensions are increased, which is better known as the curse of dimensionality. All this has contributed to popularize the development and use of randomized planning techniques. These techniques usually generate paths in short periods of time, but the

results tend to be suboptimal. A comic of path planning robots by Jorge Cham [2] can be observed in Figure 1.2. Although it is a parodic comic, it will be shown in Section 2.1.1 that it is not very far from being truth for some algorithms.

An autonomous robot is a robotic system capable of accomplishing complex tasks in uncluttered environments without human assistance. Learning and other capabilities can be included to improve the performance of its assignments. The undertaking of building an autonomous robot is very demanding, and it usually requires a group of professionals. In the Carlos III University of Madrid (UC3M), the Robotics Lab research group has built a mobile robot denominated MANFRED-2 [2] (Figure 1.3). It is an autonomous mobile manipulator robot capable of performing navigation, localization, manipulation and obstacle avoidance tasks. MANFRED-2 is bundle with a vision sensorial system, distance range lasers, a differential-type mobile base with two degrees of freedom (DOF) and an anthropomorphic light arm with six DOF and a force sensor in the wrist. The mobile base encloses a computer and all the electronic components needed to operate. The final purpose of the team working with MANFRED-2 is to develop techniques to make it more robust, reliable, accurate, and safe to work with.

Figure 1.3: Mobile manipulator robot MANFRED-2.

All the hardware and processes behind a mobile robotic system converge to achieve motion and physical interaction with the environment. After the sensorial system has committed the obtained data to the main process, the motion planners, or more specifically the path planners, determine the route that the robot should follow to execute a specific function. The plan specifies, in a sequence of space points, the

[2] http://www.willowgarage.com/blog/

pathway that the robot should follow to transfer from a start to a goal location. This usually is represented by a line that is described by the base of the robot, in navigation, or by the arm end effector, in manipulation. The planning can be made in any dimension, being common 2D for navigation and 3D for manipulation with inverse kinematics. More dimensions are used for full robotic arm DOFs that can even include the base.

1.1 Objectives

This section provides the thesis objectives that will guide the dissertation and proposed approaches.

1.1.1 General objectives

A case of study is defined for this work, a practical scenario where the mobile robot MANFRED-2 has to move to an specified location. Then, a manipulation interaction such as grasping a predefined object should be carried out. The general objective of this work is to design various specific tools related to the motion planning problem. The purpose of these developments is to improve the robot autonomy for the practical case described in this paragraph.

1.1.2 Specific objectives

The practical scenario from the general objectives section is divided into subtasks depicted in Figure 1.4. This represents the work pipeline of the mobile robot MANFRED-2 and the case of study for this research. The pipeline steps are described hereafter, and the specific objectives concerning each phase are listed. Each objective represents a development for which a description and justification is provided. Mobile robots path planning through dynamic and unstructured environments needs to be performed very quickly so the displacement of the robot can be continuous and safe. Most of the methods presented here are based on the Fast Marching Square (FM^2) method and all its good properties are inherited, generating smooth, safe, and efficient paths. The FM^2 proposed by Garrido [3], is a method based on the Fast Marching Method (FMM). First introduced by Sethian in [4], the FMM is a numerical method for solving boundary value problems of the Eikonal equation.

The first step in the diagram represents the initial path planning. The function of this step is to locate the robot in an specific location. A map of the robot environment is assumed to be known a priori. These kind of maps are usually available as building construction drawings or blueprints and they are a good reference, but they have to be

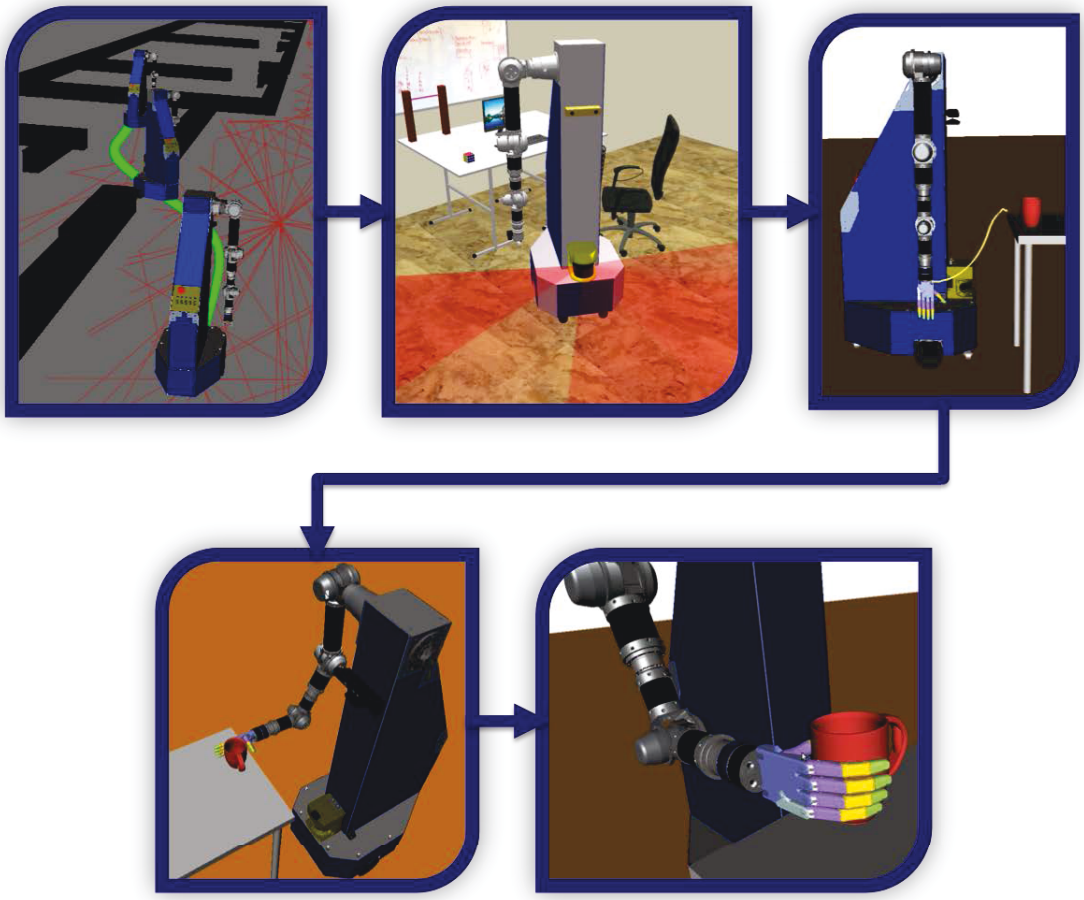


Figure 1.4: Mobile manipulator robot MANFRED-2 work pipeline.

updated as elements may change in the human environments. The specific objectives related to this phase are listed next:

- Anytime Triangular Fast Marching Square Method. The algorithms denominated "Anytime" are a time dependent approach widely used in robot motion planning. They are also called interruptible algorithms because they can be interrupted at any moment to get a result. The anytime algorithms start by quickly generating a suboptimal path with Rapidly-exploring Random Trees (RRT). RRT is a well known sampling-based planner which is widely used in robotics because of its rapid response, but the resulting paths are suboptimal and have to be improved. Then, this path is improved as time is available.

Different variants are introduced and compared under equal circumstances that produce different paths in response time and quality. This approach combines two kind of methods (sampling-based and exhaustive search methods) to benefit from both, getting a fast and more optimal planner. An additional contribution is made, as comparing planners in robotics has become increasingly difficult because of the lack of a general benchmarking platform, improvements to existing approaches are suggested. At the beginning, the study of the Anytime Triangular FM^2 was supposed to expand to more dimensions for manipulation tasks. However, a path planning with adaptive dimensionality method was proposed instead and is presented here, and the rest of the work with triangular meshes has been proposed as future developments.

- Fast Marching Square applied to Nonholonomic car-like robots. As the nature of our robot base is nonholonomic, two nonholonomic approaches named Nonholonomic Fast Marching Square (FM^2-NH) are presented. The first approach starts by pre-computing all the feasible and collision-free poses of the nonholonomic car-like robot. Then, the FM^2 and the gradient method are used to compute a smooth and reliable trajectory based on a velocity potential map. The second one relies on the vector field of velocities computed in the first step of the FM^2. These vectors are used to adapt the path planning to meet the movement constraints of a car-like robot. Both approaches calculate smooth and safe paths, while providing also with a control plan for the robot. To differentiate both methods, the first one is called C-space FM^2-NH and the second one is the control-based FM^2-NH. In addition, to generate optimal paths, the method is able to provide the control parameters of the car-like robot while maintaining quick time performance.

The second step is the navigation and replanning. This step concerns the execution of the previously generated path, here the laser scanner is used to update the map environment. In this phase, the robot might find the first path unfeasible to be completed. This could happen when a pathway has been blocked, or when the accumulated error has driven the mobile robot too far from the path. Under these scenarios, the path has to be replanned in order to adapt to the new circumstances. The next specific objective follows this phase:

- Smooth motion replanner using FM^2. Since a complete method is used, this replanner generates paths that need no additional optimization time, which represents an advantage with respect to anytime algorithms. Smooth and safe paths are guaranteed to be generated every time in real-time. A sub-goals approach is also included in order to avoid replanning entire paths, enabling its use in large environments.

The third step is the path planning for reaching the object to manipulate. Once the robot has arrived to the goal location and it is near to the object, a path is calculated to locate the arm's end effector close to this object. The following specific objectives are formulated for this phase:

- Adaptive Evolving Strategy for Dextrous Robotic Manipulation. The computer onboard of a mobile manipulator is capable of recording the movements of certain manipulation task. These movements can be reproduced by steps to repeat the complete operation. The downside of trying to execute a recorded manipulation path is that the location and orientation of a robot may vary. Navigation and localization systems are capable of getting the robot very near to a specific location and orientation. However, disruptions are commonly introduced. This approach solves this problem by adapting learned manipulation trajectories with Evolution Strategies. In this way, it is possible for the robot to accomplish a learned task from different positions. The approach avoids the use of inverse kinematics, which is known to be incomplete for some robot configurations and to generate convergence problems in singular kinematic configurations.

- Path planning with adaptive dimensionality. First, an RRT trajectory is generated using the full DOF of the robotic arm. Then a geometry as a closed tube is built around the path line and the FM^2 is applied from start to goal using three dimensions inside the surface. The resulted 3D path is converted to full DOF with the inverse kinematics of the robot. If the manipulation trajectory is not feasible due to collisions, the geometric figure is expanded around the collision space and a full DOF-based algorithm is executed in that segment. The result is a smoother and safer path, visually more human friendly. Additionally, the search space is reduced, and therefore, the planning time and the memory requirements.

- Safe Motion Planning for a Nuclear Fusion Device Arm using Fast Marching Square. The generated manipulation path is smooth and safe in an environment where safety must be the primary objective. The resulting motion planning algorithms is tested in numerous experiments. The fast response of the methods allow its application in real-time tasks.

The fourth and fifth pictures in the digram represent the manipulation action. In the present work, this specific task has been limited to grasping with a gripper, omitting the complexity behind using an anthropomorphic robotic hand. This research area is been covered by other PHD candidate, as well as all the modules needed by the robot such as those of vision, localization, hardware, control software and others are been studied by other researchers in our robotics group.

1.2 Outline

This thesis is structured in six chapters and organized as follows:

Chapter 1. Introductory chapter that describes the project problems, proposed solutions and thesis outline.

Chapter 2. The planning algorithms state of the art is reviewed in this chapter.

Chapter 3. In this chapter the path planning algorithms for moving the robot base are presented. These algorithms comprised the following: two nonholonomic approaches, these approaches have been named Nonholonomic Fast Marching Square (FM^2-NH). A replanning algorithm using FM^2. An anytime planner that relies on the FM^2 over a triangular mesh structure to improve an initial RRT path.

Chapter 4. The path planning algorithms for manipulation operations are presented in this chapter. These algorithms comprised the following: an algorithm for adaptation of manipulation trajectories that enables the robot to accomplish a task from different locations using evolution strategies. The paths are optimized in position, orientation, and energy which at the same time improve the smoothness. A planner that adaptively reduces dimensionality and, in consequence, the complexity of high-dimensional path planning problem for manipulation. An application of FM^2 with dimensionality reduction inside a nuclear fusion energy vessel. Similarly to the previous approach, the DOF of the problem are reduced (but this time to two dimensions due to the mostly planar nature of the robot).

Chapter 5. This section discusses results obtained from the experiments conducted with all the path planning algorithms for navigation and manipulation tasks.

Chapter 6. Finally, Conclusions chapter recapitulates the contributions of the thesis. Furthermore, future research directions for the individual topics as well as for the evolutionary robotics learning approach are discussed.

Additionally to the six chapters, the experimental platform MANDFRED-2 and the used software tools are presented in Appendix A.

Chapter 2

State of the Art

Autonomous robots must navigate and interact with objects and people within the environment. Therefore, the planning of motions becomes essential for the robotic system. In mobile robotics, the principle behind the motion planning problem is to guide a robot through an environment and interact with objects or tools, avoiding obstacles and taking into account any restriction such as those of mobility and energy. Motion planning has been a deeply researched area for several years. Some of the most important approaches have been addressed from the robotic field. The problem has been incrementally complicated by adding variables as obstacles, time, velocity, acceleration, nonholonomic and other constraints implicated with the environment and with the utilized mobile robot or vehicle.

In this section, the most important techniques related to the tools developed in this work are reviewed. First, motion planning and more specifically path planning are studied. Special attention is payed to the anytime algorithms and their relevance in path planning for the tasks of navigation and manipulation of a mobile robot. Then, the FMM-based techniques and their application to path planning are detailed. Finally, a comparison of motion planning methods is included.

2.1 Motion Planning

The motion planning problem involves getting a robot to determine by itself how to move while avoiding collisions with obstacles. Its original formulation, called the piano mover's problem, is imagined as determining how to move a complicated piece of furniture through a cluttered house. It has been clear for several decades that getting robots to reason geometrically about their environments and synthesize such plans is a fundamental difficulty that occurs all over robotics. The problem can be formulated as follows: let W denote the world or workspace of a robot A. The obstacle region can be denoted by O, where $O \subset W$. A configuration q is a valid transformation for the robot. The set of all rigid-body transformations that can be applied to the robot is called the configuration space or C-space. In other words, the C-space is the space comprised by all the valid dispositions of a robotic system. The C-space used in motion planning is described as a topological manifold. This description is much simpler to define and manipulate than the one considered in control theory, which is a differentiable manifold. The definition of an n-dimensional (topological) manifold C is a subset of R^m for $n \leq m$, such that every $q \in C$ is contained in at least one open subset of C that is homeomorphic. (Homeomorphic means that for an open set, say Ξ, there exists a continuous, bijective function $f : \Xi \to R^n$ for which the inverse f^{-1} is also continuous to R^n.) The intuition is that, in the local vicinity of every q, a manifold behaves like R^n. Let $A(q) \subset W$ denote a closed set of points in the world occupied by the robot A when it is transformed to configuration q. A configuration $q \in C$ places the robot into collision if and only if $A(q) \cup O \neq \emptyset$ (the

robot and obstacle are attempting to occupy at least one common point in W). The space with only valid configurations that are free of collisions, is called the free space and is defined as

$$C_{free} = \{q \in C | A(q) \cap O \neq \emptyset\}$$

Commonly, a method for detecting collisions is used to validate a configuration q, such that $q \in$ C-space and $q \in$ C-free. This means there is no need to have knowledge of C-free in advance. Although, it can be precomputed if the information of the environment is available.

The classical motion planning is conducted over the C-space and it is better known as path planning. Intuitively, a path is a sequence of states that start at a configuration, and travels from state to state along connections in space until an ending configuration. Using the C-space, the basic path-planning problem can be accurately defined: given a robot description A, an obstacle description O, a C-space C, an initial configuration $q_I \in C$, and a goal configuration q_G, compute a continuous path $\tau : [0, 1] \rightarrow C_{free}$ with $\tau(0) = q_I$ and $\tau(1) = q_G$.

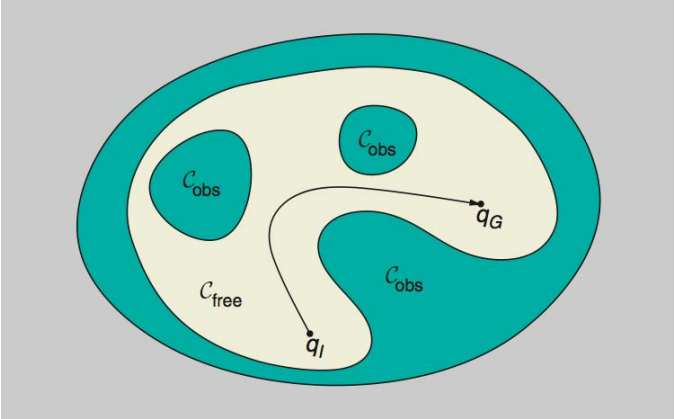


Figure 2.1: An example of the path planning problem in the C-space: connect q_I to q_G while remaining in $Cfree$ [5].

Other less common formulation is the kinodynamic planning, which is the motion planning in the state space (χ) of the robot. A state in χ is denoted by x, and it can be defined as $x = (q, \dot{q})$. A trajectory is defined as a time-parameterized continuous path $\tau : [0, T] \rightarrow \chi_{free}$. Unlike the classical approach, the kinodynamic approach involves the planning of trajectories that include differential and nonholonomic constraints that must be satisfied. These constraints arise from conservations laws, an example is the angular momentum conservation. The difference between χ and C is usually a factor of 2 in dimension. The curse of dimensionality has already contributed to

the success and popularity of randomized planning methods for C-space; therefore, it seems that there would be an even greater need to develop randomized algorithms for kinodynamic planning [6]. One reason that might account for the lack of practical, efficient planners for problems in χ-space is that attention is usually focused on obtaining optimal solutions with guaranteed deterministic convergence. Another reason why randomized kinodynamic planning approaches have not appeared is that kinodynamic planning is considerably harder owing to momentum. The unfeasibility of this requirement for generic high-dimensional systems has led many researchers to adopt a decoupled approach in which classical motion planning is performed and trajectory design is optimized around a particular motion-planning solution.

2.1.1 Motion Planning Algorithms

Early approaches of motion planners were focused on local planning, where the next action takes priority over the high level plans and motions are calculated and executed as soon as possible. This means that path planning is done while the robot is moving. Thereby, this approach is sometimes called real-time obstacle avoidance. Among these local planning approaches are the potential field-based techniques, where the robot environment maps are filled with repulsive forces which push the robot away from obstacles and attractive forces that pull it toward its goal location [7], the curvature velocity [8], and Dynamic Window [9] approaches, where planning is made in the control space to generate dynamically feasible actions. Fuzzy logic has been widely used in robotics to develop reactive controls [10, 11]. An example with a mobile robot can be found in [12]. The major disadvantage of these approaches is their susceptibility to local minima, guiding the robot to suboptimal goals [13].

Further improvements to the local minima issue were implemented in algorithms by incorporating global as well as local information in [14, 15, 16, 17]. Although these approaches improve the avoiding of local minima, their simple local planning can still cause the robot to get trapped or to execute paths very far from optimal. Subsequent approaches have focused on improving this local planning by using more sophisticated local action sets that better follow the global value function [18, 19], and by generating sequences of actions to perform more complex local maneuvers [20, 21]. The most complex of these approaches are able to perform very precise local maneuvering, but are limited by the mismatch between their powerful local planning and their approximate global planning, resulting once more in a tendency to local minima [13]. Some of the most widely known path planning methods are presented in Table 2.1.

Over the last few years, a number of methods have also been proposed based on systematic discretization of the environment and applying efficient graph searches or

using highly informative heuristics to guide the search for feasible paths. An example using fuzzy cognitive maps can be found in [34]. However, the computational complexity of generating feasible plans over large distances is high, and current approaches are restricted to either small distances, fairly simple environments or highly suboptimal solutions [13].

The techniques more related to this work will be reviewed in the next sections. The RRT method is very relevant because is used as a reference for comparative purposes, and sometimes even as part of our methods. The FM[2], core of the proposed techniques, will be also analyzed in detail.

2.2 Path-Planning Algorithms

In most solving approaches the planning problem is discretized and the state space is considered finite. When it is not finite, it will at least be countably infinite (i.e., a unique integer may be assigned to every state). Therefore, no geometric models or differential equations will be needed to characterize the discrete planning problems. The basic idea is that each distinct situation for the world is called a state, denoted by x, and the set of all possible states is called a state space, X. For discrete planning, it will be important that this set is countable; in most cases it will be finite. In a given application, the state space should be defined carefully so that irrelevant information is not encoded into a state. On the other hand, it is important that X is large enough to include all information that is relevant to solve the task.

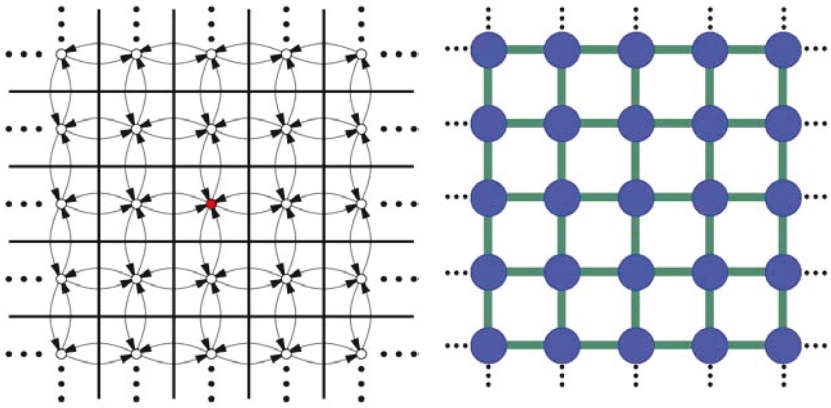


Figure 2.2: The state transition in a gridmap for an example problem that involves walking around on an infinite tile floor [35] (left). Equivalent graph representation of the gridmap on the left (right).

Space representations of all kinds can mainly be classified into two categories, which are metric maps and topological maps. Metric maps are characterized by a

representation where the position of the obstacles is indicated by coordinates in a global frame of reference, each grid point has integer coordinates of the form (i, j). Let X be the set of all integer pairs of the form (i, j), in which $i, j \in \mathbb{Z}$ ($\mathbb{Z}$ denotes the set of all integers). Let $U = (0, 1), (0, -1), (1, 0), (-1, 0)$. Let $U(x) = U$ for all $x \in X$. The state transition equation is $f(x, u) = x + u$, in which $x \in X$ and $u \in U$ are treated as two-dimensional vectors for the purpose of addition. Also, each point grid has a value that defines regions that can be occupied or not by obstacles or goals [36], [37]. In Figure 2.2 (left), an example of endless tiles gridmap is shown. In this example, the movements from one grid point to other go either up, down, left or right, but some implementations also include diagonal moves.

In Figure 2.3, an example of a gridmap with a path is shown. The black squares are the obstacles, the yellow circle is the start, the red cross is the goal and the blue line is the path.

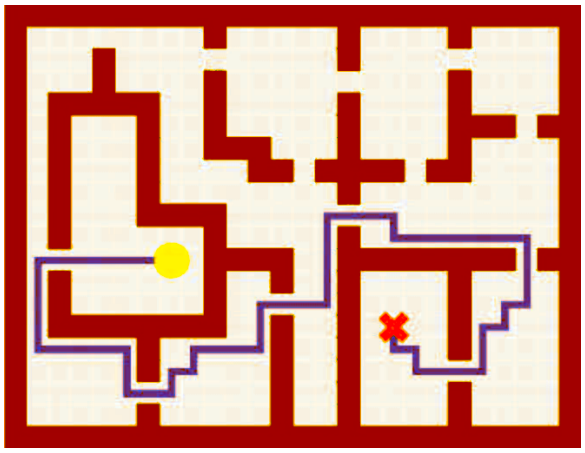


Figure 2.3: Example of a path in a gridmap. The path is depicted as a blue line.

Topological maps, also known as relational maps, represent the environment with connectivity information, typically in the form of graphs that connect landmarks or places with special features [38],[39]. The graphs are comprise of vertexes connected by edges that have numeric weights attached to them. The edges settle the valid moves between nodes. Edges are also called arcs or lines and they are not necessarily straight lines; vertexes are also called nodes or points. A graph is defined as an ordered triple $G = \{V(G), E(G), I_G\}$, where $V(G)$ is a nonempty set, $E(G)$ is a set disjoint from $V(G)$, and I_G is an incidence relation, that is, a subset of $V(G) \times E(G)$ that associates an unordered pair of elements of $V(G)$. Elements of $V(G)$ are called the vertices of G, and elements $E(G)$ are called the edges of G. The incidence relation I_G is required to be such that an edge is incident with either one vertex (in which case is a loop) or two vertices. For example, if for an edge e of G, $I_G(e) = \{v_1, v_2\}$,

then the connection is written $I_G(e) = v_1 v_2$. Graphs can be directed or undirected according to wether or not their edges have orientation. A directed graph or digraph is an ordered triple $G = \{V(G), E(G), I_G\}$, with $E(G)$ a set of ordered pairs of vertices, called directed edges or arrows. An undirected graph is one in which edges have no orientation. The edge (a, b) is identical to the edge (b, a), *i.e.*, they are not ordered pairs, but sets u, v of vertices.

Discrete planning is defined by the following elements:

1. A nonempty state space X, which is a finite or countably infinite set of states.
2. For each state $x \in X$, a finite action space $U(x)$.
3. A state transition function f that produces a state $f(x, u) \in X$ for every $x \in X$ and $u \in U(x)$. The state transition equation is derived from f as $x' = f(x, u)$.
4. An initial state $x_I \in X$.
5. A goal set $X_G \subset X$.

It is often convenient to express the above formulation as a directed state transition graph. The set of vertices is the state space X. A directed edge from $x \in X$ to $x' \in X$ exists in the graph if and only if there exists an action $u \in U(x)$ such that $x' = f(x, u)$. The initial state and goal set are designated as special vertices in the graph, which completes the representation of the above formulation in a graph form.

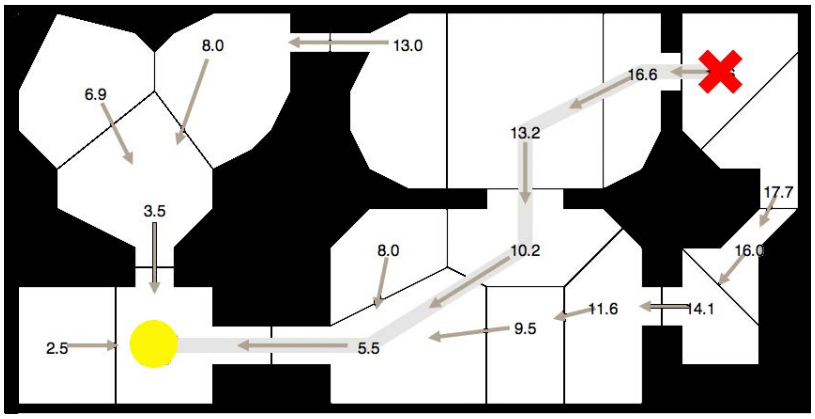


Figure 2.4: Example of a path over a metric graph map. The gray thick line represents the taken path.

A gridmap can also be viewed as a special case of a graph, where the grid points are the vertexes and the possible grid connections the edges. See Figure 2.2 (right) for a graph representation of a gridmap. Further, many actual representations use

both a topological and a metric component. In Figure 2.4, an example of the use of both approaches for path planning is shown. A graph is used over a floor blueprint to connect specific locations of the rooms. The cost between rooms coincide with the distances between vertexes.

In our work, the grid-based map has been chosen to represent the environment. Both metric and topological references have been developed for indicating subgoals. By using grids we already have a discrete environment representation, which is a great advantage because all is set to be used in the generation of the velocity potential map and in the FMM for path planning. A velocity potential map of the environment represents the admissible velocity at discrete cells in the workspace. This map of the environment gives a grey scale that is darker near the obstacles and walls (black means zero speed) and more clear far from them (white means a predefined maximum robot speed).

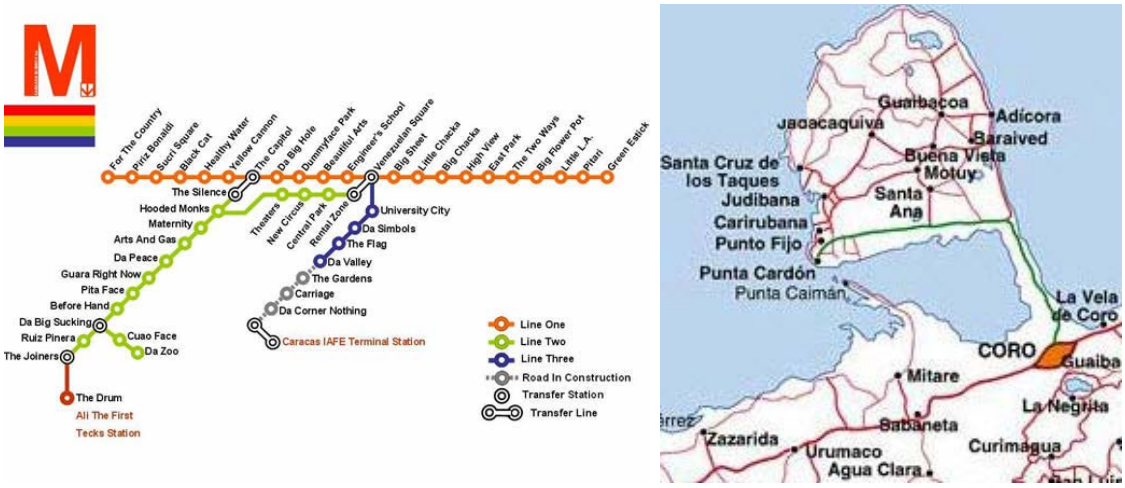


Figure 2.5: A metro topological map and a metric map.

Examples of more common topological and metric maps in our daily life are shown in Figure 2.5. The metro map on the left of the figure is composed of the underground stations (nodes) and the connections between them (vertexes). The metric map on the right of the figure contains a two-dimensional spatial model, which represents a particular portion of a land.

2.2.1 Graph Search Algorithms

This section presents several search algorithms, each of which constructs a search tree. Each search algorithm is a special case of the algorithm in Section 2.2.1. Most of these are just classical graph search algorithms [40]. An important requirement for these

or any search algorithms is to be systematic. If the graph is finite, this means that the algorithm will visit every reachable state, which enables it to correctly declare in finite time whether or not a solution exists. To be systematic, the algorithm should keep track of states already visited; otherwise, the search may run forever by cycling through the same states. Ensuring that no redundant exploration occurs is sufficient to make the search systematic [5].

General Graph-Search Algorithm

The first family of path-planning algorithms presented in this section, perform path search through a graph [5]. Graph search algorithms can be used directly on topological and metric topological maps, because these representations are essentially graphs. An important requirement for these or any search algorithms is to be systematic. If the graph is finite, this means that the algorithm will visit every reachable state, which enables it to correctly declare in finite time whether or not a solution exists. To be systematic, the algorithm should keep track of states already visited; otherwise, the search may run forever by cycling through the same states. Ensuring that no redundant exploration occurs is sufficient to make the search systematic. In other words, the requirement to be systematic is that, in the limit, as the number of iterations tends to infinity, all reachable vertices are reached, see Figure 2.6 (right).

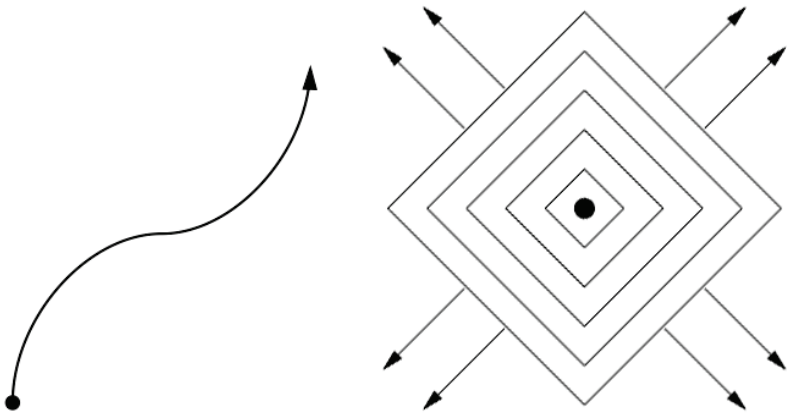


Figure 2.6: Some algorithms focus on one direction (left), which may prevent them from being systematic on infinite graphs. When the search carefully expands in wavefronts, then it becomes systematic (right).

In Graph-Search Algorithms X_G is a set of goal vertices that represent acceptable termination states for the path, and X_I is a set of possible starting locations. In the presented figures, the sets X_I and X_G contain only one vertex each for simplicity. During graph-search, an algorithm starts at X_I and attempts to find a path to X_G,

or vice versa, by exploring from node-to-node via edges. It is known as forward search when the search is conducted from X_I to X_G, and backward search when the search is conducted from X_G to X_I. Bidirectional search starts at both X_G and X_I and attempts to connect the two searches somewhere in the middle. Multi-directional search starts at X_G and X_I as well as other random or intuitive locations and attempts to link the searches together in such a way that a path is found between X_I and X_G. Algorithm 1 gives a general template of search algorithms, expressed using the state-space representation.

Algorithm 1 Graph-Search Algorithm

Input: Q, x_I, x_G
Output: SUCCESS or FAILURE.

```
1: add x_I to the open-list Q and mark as visited
2: while Q not empty do
3:    x ← remove a node from the open-list Q
4:    if x ∈ x_G then
5:       return SUCCESS
6:       for all u ∈ U(x) do
7:          x' ← f(x, u)
8:          if (x' not visited) then
9:             Mark x' visited
10:            add x' to the open-list Q
11:            Set back-pointer from x' to x
12:         else
13:            Resolve Duplicate x'
14:         end if
15:      end for
16:   end if
17: end while
18: return FAILURE
```

At any point during the search, there will be three kinds of states:

1. Unvisited: States that have not been visited yet. Initially, this is every state except x_I.

2. Dead or closed: States that have been visited, and for which every possible next state has also been visited. A next state of x is a state x' for which there exists a $u \in U(x)$ such that $x' = f(x, u)$. In a sense, these states are dead because there is nothing more that they can contribute to the search; there are no new leads that could help in finding a feasible plan. Some approaches use a variant in which dead states can become alive again in an effort to obtain optimal plans.

3. Alive or open: States that have been encountered (visited), but possibly have unvisited next states, are considered alive. Initially, the only alive state is x_I.

The set of alive states is stored in a list Q, which is called the open-list. Initially, Q contains the initial state x_I. A while loop is then executed, which terminates only when Q is empty. This will only occur when the entire graph has been explored without finding any goal states, which results in a FAILURE. In each while iteration, the highest ranked element, x, of Q is removed. If x lies in x_G, then it reports SUCCESS and terminates; otherwise, the algorithm tries applying every possible action, $u \in U(x)$. For each next state, $x' = f(x, u)$, it must determine whether x' is being encountered for the first time. If it is unvisited, then it is inserted into Q; otherwise, there is no need to consider it because it must be either dead or already in Q.

In order to produce a plan, which is a sequence of actions that achieves the goal. In line 8, an statement that associates x with x' its parent is executed. This is performed each time, thereby, one can simply trace the pointers from the final state to the initial state to recover the plan. For convenience, one might also store which action was taken, in addition to the pointer from x' to x.

In lines 8 and 9 is determined whether x' has been visited and the state value is updated to visited. For some problems the state transition graph might actually be a tree, which means that there are no repeated states. In this case, there is no need to check whether states have been visited. If the states in X all lie on a grid, one can simply make a lookup table that can be accessed in constant time to determine whether a state has been visited.

One final detail is that many search algorithms will require a cost to be computed and associated with every state. If the same state is reached multiple times, the cost may have to be updated, which is performed in line 13, if the particular search algorithm requires it. Such costs may be used in some way to sort the priority queue, or they may enable the recovery of the plan on completion of the algorithm. Instead of storing pointers, as mentioned previously, the optimal cost to return to the initial state could be stored with each state. This cost alone is sufficient to determine the action sequence that leads to any visited state. Starting at a visited state, the path back to x_I can be obtained by traversing the state transition graph backward in a way that decreases the cost as quickly as possible in each step. For this to succeed, the costs must have a certain monotonicity property, which will be introduced later in this section. Note that the method of choosing x from the open-list on line 3 has not been specified. This is the only significant difference between various search algorithms and it is defined by the particular function used to sort Q. The next two subsections describe naive methods of choosing x that are well known. Subsequent subsections cover more complicated techniques.

Breadth first

Breath-first search attempts to make the search-tree as broad as possible, as quickly as possible [40].

Algorithm 2 Breadth-First Search

Input: Q, X_I, X_G
Output: SUCCESS or FAILURE.

```
1: for all x ∈ X_I do
2:    Q.push_back(x)
3: end for
4: while Q not empty do
5:    x ← Q.pop_top()
6:    if x ∈ X_G then
7:       return SUCCESS
8:       for all u ∈ U(x) do
9:          x' ← f(x, u)
10:         if (x' not visited) then
11:            Mark x' visited
12:            Q.push_back(x')
13:            Set back-pointer from x' to x
14:         end if
15:      end for
16:   end if
17: end while
18: return FAILURE
```

The algorithm specifies the open-list Q as a First-In First-Out (FIFO) queue which selects states using the first-come, first-serve principle. This structure adds open nodes to the back of the queue, and visits nodes off the front of the queue. This way, nodes are expanded in the same order they are discovered. This causes the search frontier to grow uniformly and is therefore referred to as breadth-first search. All plans that have k steps are exhausted before plans with $k + 1$ steps are investigated. Therefore, breadth first guarantees that the first solution found will use the smallest number of steps. As regards to the Algorithm 1 in line 13, when a state has been revisited there is nothing to do. Since the search progresses in a series of wavefronts, breadth-first search is systematic. In fact, it even remains systematic if it does not keep track of repeated states (however, it will waste time considering irrelevant cycles). Pseudo-code for breadth-first search is displayed in Algorithm 2. The function push-back() in line 12, adds a discovered node to the back of the queue. In figure 2.7, a sequence example of the Breadth-First algorithm and the resulting

path is illustrated. Although it is not considered by this algorithm, the green squares are points with higher cost to traverse.

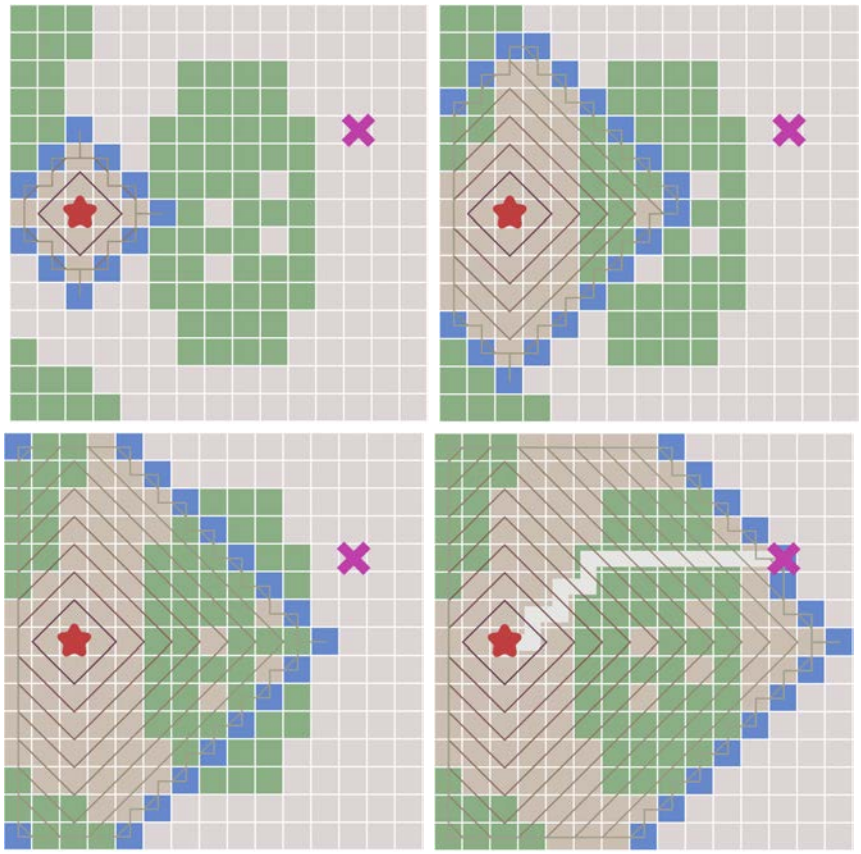


Figure 2.7: Breadth-First path planning sequence. The blue squares are alive nodes. The red star is X_I and the purple cross is X_G. Wavefronts are drawn through same level nodes. The obtained path is the light color line in the bottom right image.

Breadth-First Search is complete in a finite graph. It is incomplete in a countably infinite graph; however, it will find a solution if one exists (*i.e.* is resolution complete). This method is usually more useful when X_I only includes a few nodes. However, breadth-first search may be inefficient in large graphs and/or high dimensional spaces. The asymptotic running time of breadth-first search is $O(|V|+|E|)$, in which $|V|$ and $|E|$ are the numbers of vertices and edges, respectively, in the state transition graph. This assumes that all basic operations, such as determining whether a state has been visited, are performed in constant time. In practice, these operations will typically require more time and must be counted as part of the algorithm's complexity.

Depth first

By making the open-list Q a stack (Last-In, First-Out; or LIFO), aggressive exploration of the state transition graph occurs, as opposed to the uniform expansion of breadth-first search. The resulting variant is called depth-first search because the search dives quickly into the graph. Breadth-first search, the method described in previous subsection, is more methodical than depth-first search. The pseudo-code for depth-first search is displayed in Algorithm 3. The functions *push-top()* and *pop-top()* place a node on the top of the stack and remove a node from the top of the stack, respectively

Algorithm 3 Depth-First Search

Input: Q, X_I, X_G
Output: SUCCESS or FAILURE.

```
1: for all x ∈ X_I do
2:    Q.push_top(x)
3: end for
4: while Q not empty do
5:    x ← Q.pop_top()
6:    if x ∈ X_G then
7:       return SUCCESS
8:       for all u ∈ U(x) do
9:          x' ← f(x, u)
10:         if (x' not visited) then
11:            Mark x' visited
12:            Q.push_top(x')
13:            Set back-pointer from x' to x
14:         end if
15:      end for
16:   end if
17: end while
18: return FAILURE
```

The preference of the Depth-First search is toward investigating longer plans very early. Although this aggressive behavior might seem desirable, note that the particular choice of longer plans is arbitrary. Actions are applied in the *forall* loop in whatever order they happen to be defined. Once again, there is no work to do if a state is revisited and *else* command is omitted from line 14. Given a finite graph, depth-first search is complete. However, in a countably infinite graph may behave like Figure 2.6 (left), under these circumstances depth-first search is not complete and may not even find a solution when one exists. The search could easily focus on

one direction and completely miss large portions of the search space as the number of iterations tends to infinity. The running time of depth first search is also $O(|V|+|E|)$.

Dijkstra

Dijkstra's algorithm [22] is one of the earliest discovered optimal best-first search algorithms. It is a method for finding single-source shortest paths in a graph, and it is also a special form of dynamic programming. In figure 2.8, a sequence example of Dijkstra's algorithm is illustrated. In contrast to breadth first and depth first algorithms, the higher cost of green cells is considered.

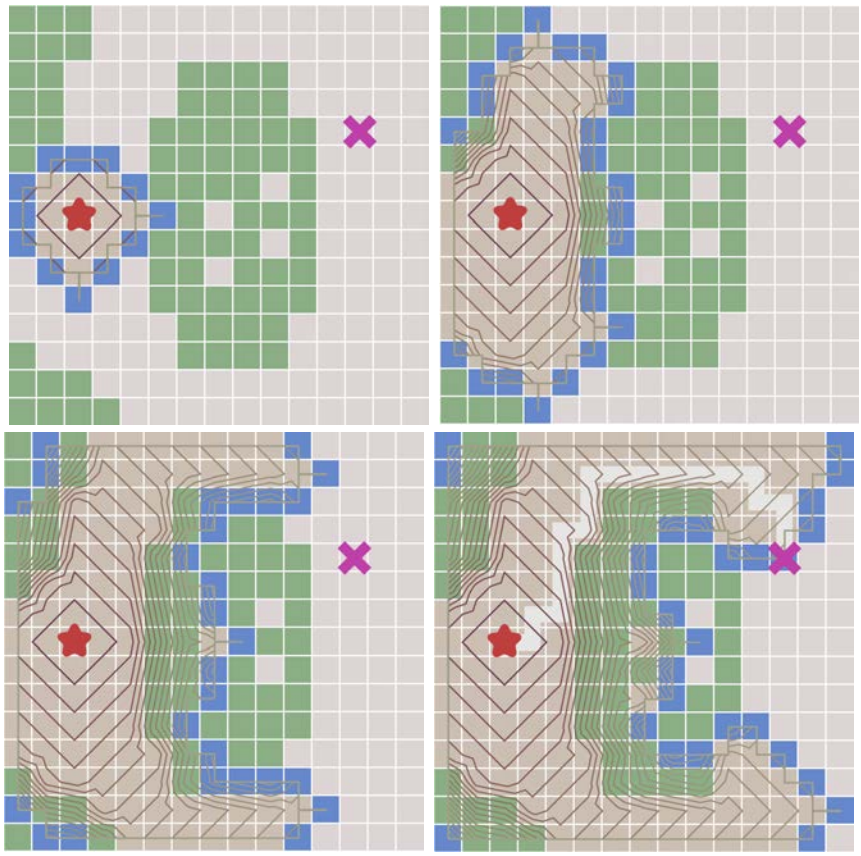


Figure 2.8: Sequence of Dijkstra's algorithm. The blue squares represent the alive nodes, the green grid points are slower points,

In the algorithms described up to this point, there has been no reason to prefer one action over any other in the search. Dijkstra's algorithm assumes that distances $l(e)$ (nonnegative cost) are known for all edges $e \in E$ in the graph representation of a discrete planning problem. The cost $l(e)$ could be written using the state-space

representation as $l(x, u)$, indicating that it costs $l(x, u)$ to apply action u from state x. The total cost of a plan is just the sum of the edge costs over the path from the initial state to a goal state. The pseudo-code for Dijkstra's algorithm is displayed in Algorithm 4. The priority queue Q, will be sorted according to a function $C : X \rightarrow [0, \infty]$, called the cost-to-come. For each state x, the value $C^*(x)$ is called the optimal cost-to-come from the initial state x_I. This optimal cost is obtained by summing edge costs, $l(e)$, over all possible paths from x_I to x and using the path that produces the least cumulative cost. If the cost is not known to be optimal, then it is written as $C(x)$.

Algorithm 4 Dijkstra's

Input: Q, x_I, X_G
Output: SUCCESS or FAILURE.

```
1:  Q.insert(x_I, 0)
2:  while Q not empty do
3:     x ← Q.get_best()
4:     if x ∈ X_G then
5:        return SUCCESS
6:        for all u ∈ U(x) do
7:           x' ← f(x, u)
8:           if (x' not visited) or C(x') > C*(x) + l(x, u) then
9:              Mark x' visited
10:             C(x') = C*(x) + l(x, u)
11:             Q.insert(x', C(x'))
12:             Set back-pointer from x' to x
13:          end if
14:       end for
15:    end if
16: end while
17: return FAILURE
```

The cost-to-come is computed incrementally during the execution of the search algorithm in line 10. Initially, $C^*(x_I) = 0$. Each time the state x' is generated, a cost is computed as $C(x') = C^*(x) + l(e)$, in which e is the edge from x to x' (equivalently, we may write $C(x') = C^*(x) + l(x, u)$). Here, $C(x')$ represents the best cost-to-come that is known so far, but we do not write C^* because it is not yet known whether x' was reached optimally. Due to this, some work is required a vertex is revisited. If x' already exists in Q, then it is possible that the newly discovered path to x' is more efficient. If so, then the cost-to-come value $C(x')$ must be lowered for x', and Q must be reordered accordingly.

Once x is removed x_I from Q using $Q.get\_best()$ in line 3, the state becomes dead, and it is known that x cannot be reached with a lower cost. Thereby, the $C(x)$ finally becomes $C^*(x)$ for some state x. This can be intuitively argued by induction as follows. For the initial state, $C^*(x_I)$ is known, and this serves as the base case. Now assume that every dead state has its optimal cost-to-come correctly determined. This means that their cost-to-come values can no longer change. For the first element, x, of Q, the value must be optimal because any path that has a lower total cost would have to travel through another state in Q, but these states already have higher costs. All paths that pass only through dead states were already considered in producing $C(x)$. Once all edges leaving x are explored, then x can be declared as dead, and the induction continues.

The running time of the algorithm is $O(|V| \log |V| + |E|)$, in which $|V|$ and $|E|$ are the numbers of edges and vertices, respectively, in the graph representation of the discrete planning problem. This assumes that the priority queue is implemented with a Fibonacci heap, and that all other operations, such as determining whether a state has been visited, are performed in constant time. If other data structures are used to implement the priority queue, then higher running times may be obtained. Dijkstra's Algorithm is complete for finite graphs and resolution complete for countably infinite graphs. Note that if $l(e) = 0$ for all $e \in E$ then Dijkstra's algorithm degenerates into breadth-first search.

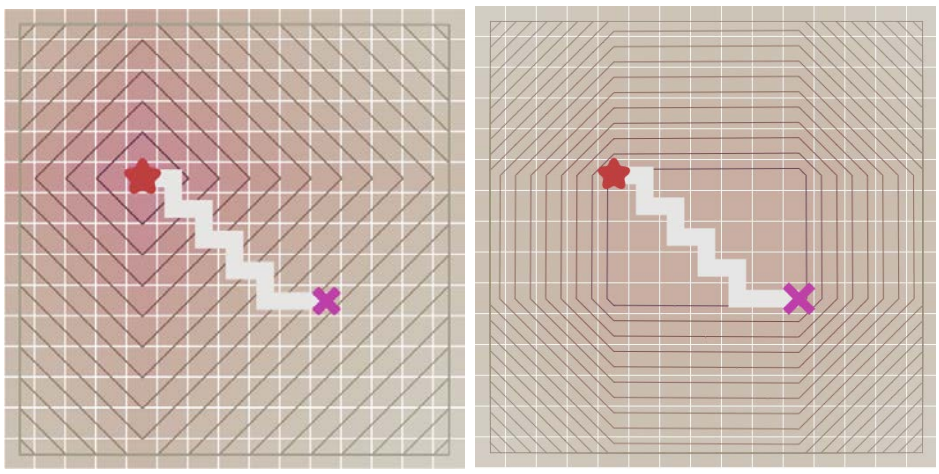


Figure 2.9: Dijkstra's path-planning wavefront (left). A* path-planning wavefront (right).

A-star

The A* (pronounced ay star) search algorithm [23] is an extension of Dijkstra's algorithm that tries to reduce the total number of states explored by incorporating a heuristic estimate of the cost to get to the goal from a given state. Let $C(x)$ denote the cost-to-come from x_I to x, and let $G(x)$ denote the cost-to-go from x to some state in x_G. It is convenient that $C^*(x)$ can be computed incrementally by dynamic programming; however, there is no way to know the true optimal cost-to-go G^*, in advance. Fortunately, in many applications it is possible to construct a reasonable underestimate of this cost.

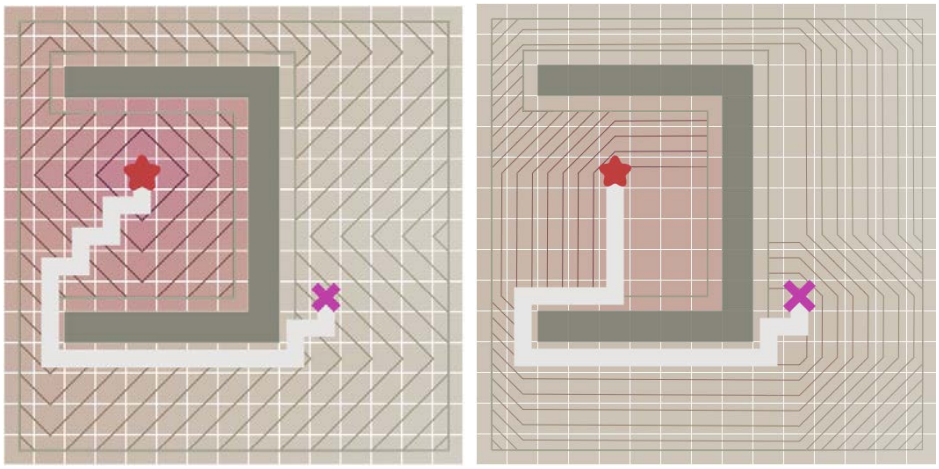


Figure 2.10: Dijkstra's path-planning wavefront (left). A* path-planning wavefront (right).

Suppose that the cost is the total number of steps in the plan. If one state has coordinates (i, j) and another has (i', j'), then $|i' - i| + |j' - j|$ is an underestimate because this is the length of a straightforward plan that ignores obstacles. Once obstacles are included, the cost can only increase as the robot tries to get around them (which may not even be possible). Of course, zero could also serve as an underestimate, but that would not provide any helpful information to the algorithm. The aim is to compute an estimate that is as close as possible to the optimal cost-to-go and is also guaranteed to be no greater. Let $\hat{G}^*(x)$ denote such an estimate. The pseudo-code for the A* algorithm is displayed in Algorithm 5.

The A* search algorithm works in exactly the same way as Dijkstra's algorithm. The only difference is the function used to sort Q. In the A* algorithm, the sum $C^*(x') + \hat{G}^*(x')$ is used, implying that the priority queue is sorted by estimates of the optimal cost from x_I to x_G. The A* search algorithm performs search where cost is

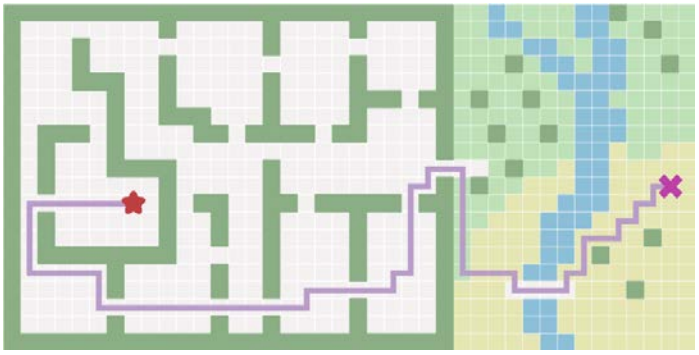


Figure 2.11: A* path planning example in a map with different costs.

defined as:

$$C(x') = C^*(x') + \hat{G}^*(x') = \min_{(u \in U(x)) \to x'} \{C(x) + l(x,u)\} + \hat{G}^*(x') \tag{2.1}$$

In Figure 2.9, the wavefronts for both Dijkstra and A* can be appreciated in a path planning example. If $\hat{G}^*(x)$ is an underestimate of the true optimal cost-to-go for all $x \in X$, the A* algorithm is guaranteed to find optimal plans [41, 42]. As $\hat{G}^*$ becomes closer to G^*, fewer vertices tend to be explored in comparison with Dijkstra's algorithm. This would always seem advantageous, but in some problems it is difficult or impossible to find a heuristic that is both efficient to evaluate and provides good search guidance.

In Figure 2.10, a path planning example where Dijkstra generates a better path than A* (the way down from start is considered as a diagonal for Dijkstra's path). Note that when $\hat{G}^*(x) = 0$ for all $x \in X$, then A* degenerates to Dijkstra's algorithm. In any case, the search will always be systematic. An example of path planning in map with different cost is presented in Figure 2.11.

2.2.2 Sampling-based Planning Algorithms

The sampling-based planning algorithms presented in this section are very similar to the family of search algorithms summarized in Section 2.2.1. The main difference lies in the local planning method step, in which applying an action, u, is replaced by generating a path segment, $\mathcal{T}_s$. Another difference is that the search graph, $\mathcal{G}$, is undirected, with edges that represent paths, as opposed to a directed graph in which edges represent actions. Although it is possible to make these look similar by defining an action space for motion planning that consists of a collection of paths.

Algorithm 5 A*

Input: Q, x_I, X_G
Output: SUCCESS or FAILURE.

```
1: Q.insert(x_I, 0)
2: while Q not empty do
3:    x ← Q.get_best()
4:    if x ∈ X_G then
5:       return SUCCESS
6:       for all u ∈ U(x) do
7:          x' ← f(x, u)
8:          if (x' not visited) or C(x') > C*(x) + Ĝ*(x') then
9:             Mark x' visited
10:            C(x') = C*(x') + Ĝ*(x')
11:            Q.update(x', C(x'))
12:            Set back-pointer from x' to x
13:         end if
14:      end for
15:   end if
16: end while
17: return FAILURE
```

PRM

The Probabilistic Roadmaps (PRMs) method addresses the motion planning as a multiple-query problem [5]. Numerous initial-goal queries are fed to the algorithm, while keeping the robot model and obstacles fixed. This changes the basic motion planning approach, where it was assumed that a single initial-goal pair was given to the planning algorithm. The original paradigm of the method was introduced in [25]. It is intended to solve the motion planning problem between any given pair of points. In this context, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently.

The main and most critical task of the planner is to build a topological graph called roadmap. Once this sampling-based graph is constructed, the paths from multiple initial-goal queries can be quickly generated on the roadmap from each of q_I and q_G.

In the basic method, a topological graph is represented by a pair $G(V, E)$ in which V is a set of vertices and E is the set of paths that map into $Cfree$. Under the multiple-query philosophy, motion planning is divided into a preprocessing and a query phase, which are explained as follows:

Preprocessing Phase: During the preprocessing phase, substantial effort is invested to build G in a way that is useful for quickly answering future queries. For

this reason, it is called a roadmap, which in some sense should be accessible from every part of $Cfree$.

Algorithm 6 Roadmap Construction

Input: N, K
Output: $\mathcal{G}$.

```
1: G.init(); i ← (0)
2: while i < N do
3:   if α(i) ∈ Cfree then
4:     G.add_vertex(α(i)); i ← i + 1;
5:     for all q ∈ Neighborhood(α(i), G, K) do
6:       if (not G.same_component(α(i), q)) and connect(α(i), q) then
7:         G.add_edge(α(i), q)
8:       end if
9:     end for
10:  end if
11: end while
```

In Algorithm 6, an outline of the basic preprocessing phase is presented. In Figure 2.12, the iterative algorithm for creating the roadmap is illustrated.

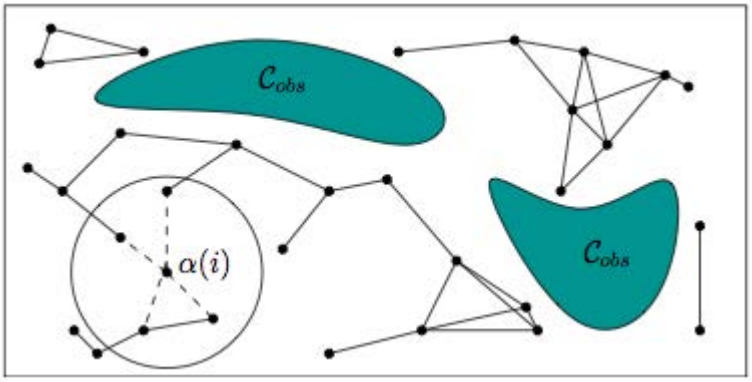


Figure 2.12: Illustration of the sampling-based roadmap algorithm. The roadmap is build by connecting new samples, $\alpha(i)$ to neighbor vertices [5].

The algorithm utilizes a uniform, dense sequence α. A sequence, as opposed to a set, is called dense if its underlying set is dense. K is the number of closest neighbors to examine for each configuration. In each iteration, the algorithm must check whether $\alpha(i) \in Cfree$. If $\alpha(i) \in Cobs$, then it must continue to iterate until a collision-free sample is obtained. Once $\alpha(i) \in Cfree$, then in line 4 it is inserted as a

vertex of $\mathcal{G}$. The next step is to try to connect $\alpha(i)$ to K nearby vertices, q, of $\mathcal{G}$. Each connection is attempted by the connect function, which is a typical local planning method explained later in Algorithm 10. In most implementations, this simply tests the shortest path between $\alpha(i)$ and q. If the path is collision-free, then connect returns true. As the algorithm iterates through the loop, more and more vertices are added to the graph $\mathcal{G}$ and components may be created. A component is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph. Observe that in Figure 2.14 the roadmap is divided in two components, one with yellow lines and other very small with light blue lines. The same component condition in line 6 checks to make sure $\alpha(i)$ and q are in different components of $\mathcal{G}$ before wasting time on collision checking. This ensures that every time a connection is made, the number of connected components of G is decreased.

Several possible implementations of line 5 for selecting neighboring samples can be made. In all of these, it seems best to sort the vertices that will be considered for connection in order of increasing distance from $\alpha(i)$. This makes sense because shorter paths are usually less costly to check for collision, and they also have a higher likelihood of being collision-free. If a connection is made, this avoids costly collision checking of longer paths to configurations that would eventually belong to the same connected component.

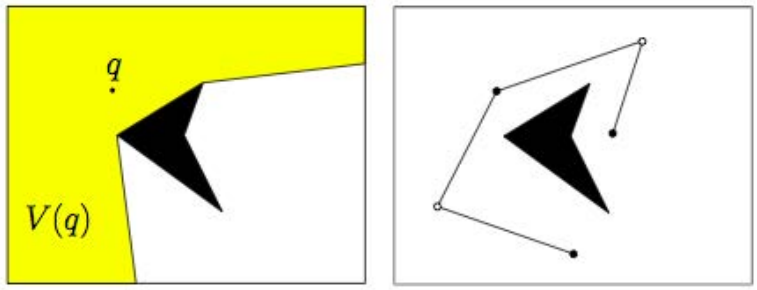


Figure 2.13: The visibility of the q configuration, $V(q)$, is the set of points reachable from q (left). An example of visibility roadmap: guard vertices are shown in black, and connectors are shown in white [5].

Several useful implementations of neighborhood are:

1. Nearest K: The K closest points to $\alpha(i)$ are considered. This requires setting the parameter K.

2. Component K: Try to obtain up to K nearest samples from each connected component of G. A reasonable value is K = 1; otherwise, too many connections would be tried.

3. Radius: Take all points within a ball of radius r centered at $\alpha(i)$. An upper limit, K, may be set to prevent too many connections from being attempted. Typically, $K = 20$. A radius can be determined adaptively by shrinking the ball as the number of points increases. This reduction can be based on dispersion or discrepancy, if either of these is available for α. Note that if the samples are highly regular (e.g., a grid), then choosing the nearest K and taking points within a ball become essentially equivalent. If the point set is highly irregular, as in the case of random samples, then taking the nearest K seems preferable.

4. Visibility: A visibility roadmap is based on the principles o visibility between nodes, and it defines two kinds of vertices: guards and connectors, see Figure 2.13. Guards are not allowed to see other guards. Connectors must see at least two guards. The roadmap construction phase proceeds similarly to the Algorithm 6. The difference is in how it determines whether to keep $\alpha(i)$ and its associated edges in $\mathcal{G}$. If the new sample, $\alpha(i)$, is not able to connect to any guards, then $\alpha(i)$ earns the privilege of becoming a guard itself and is inserted into $\mathcal{G}$. When a new sample can connect to guards from at least two different connected components of $\mathcal{G}$, it becomes a connector. If neither of the previous two conditions were satisfied, the case $\alpha(i)$ is discarded. This approach works very hard to ensure that the roadmap representation is small yet covers $Cfree$ well.

Query Phase: During the query phase, a pair, q_I and q_G, is given. Each configuration must be connected easily to G using a local planner. Following this, a discrete search is performed using any local planner, as it is for instance Dijkstra, to obtain a sequence of edges that forms a path from q_I to q_G.

In the query phase, it is assumed that G is sufficiently complete to answer many queries, each of which gives an initial configuration, q_I , and a goal configuration, q_G. First, the query phase pretends as if q_I and q_G were chosen from α for connection to G. This requires running two more iterations of the algorithm in Figure 5.25. If q_I and q_G are successfully connected to other vertices in G, then a search is performed for a path that connects the vertex q_I to the vertex q_G. The path in the graph corresponds directly to a path in $Cfree$, which is a solution to the query. Unfortunately, if this method fails, it cannot be determined conclusively whether a solution exists. If the dispersion is known for a sample sequence, α, then it is at least possible to conclude that no solution exists for the resolution of the planner.

The success rate for PRM algorithm is reduced as the problem precision is increased. Many planning problems involve moving a robot through an area with tight clearance. This generally causes narrow channels to form in $Cfree$, which leads to a challenging planning problem for the sampling-based roadmap algorithm. Finding the escape of a bug trap is also challenging, but for the roadmap methods, even

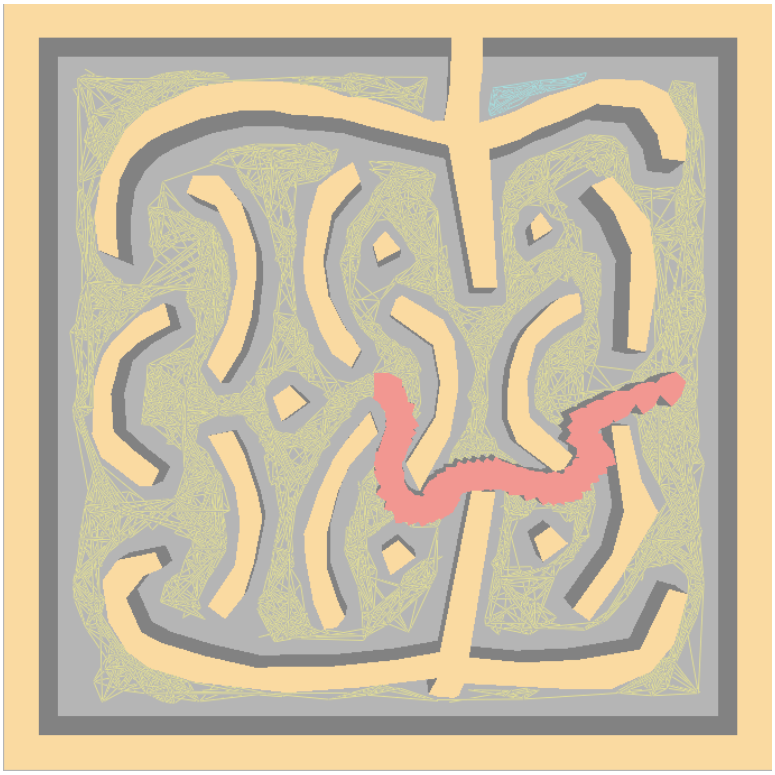


Figure 2.14: Example of an PRM search and generated path.

traveling through a single corridor is hard. For Figure 2.14, consider the scenario of taking x_I inside the blue component and x_G inside the yellow component, then the algorithm would not find a path.

RRT

The RRT is a sampling-based method that was first proposed by LaValle in [26]. It is a data structured algorithm characterized by accomplishing quick searches in high-dimensional spaces. The technique is approached as a search in the C-space C of the robot. Commonly, it is 2D or 3D for path planning, and 6D or 7D for robotics arms depending on their DOF. It can be n-dimensional. The space free of obstacles, C_{free} is represented directly in the environment. Normally, it can not be known in advance if a configuration belongs to C_{free}. However, there are collision detection algorithms that given a valid configuration q in C can determine wether $q \in C_{free}$.

The objective of the algorithm is to expand the tree $\mathcal{T}$ towards the unexplored environment. The construction of the RRT is depicted in Algorithms 7 and 8. The method receives the initial configuration q_{init}, the number of vertices K to limit the

grow of the RRT nodes, and the incremental distance Δq. The process starts in Algorithm 7, the initial vertex is added to the tree $\mathcal{T}$ in line 1. The loop generates a random configuration with a random number of vertex limited by K in line 3. This result is passed to Algorithm 8 in the next line.

Algorithm 7 BUILD_RRT

Input: q_{init}, K, Δq
Output: $\mathcal{T}$
1: $\mathcal{T}.init(q_{init})$
2: **for all** $k \in K$ **do**
3: $q_{rand} \leftarrow RAND\_CONF()$
4: $EXTEND(\mathcal{T}, q_{rand})$;
5: **end for**
6: *return* $\mathcal{T}$

Algorithm 8 EXTEND

Input: $\mathcal{T}$, q
Output: Reached, Advanced or Trapped.
1: $q_{near} \leftarrow NEAREST\_NEIGHBOR(q, \mathcal{T})$
2: **if** $NEW\_CONF(q, q_{near}, q_{rand})$ **then**
3: $\mathcal{T}.add\_vertex(q_{new})$
4: $\mathcal{T}.add\_edge(q_{near}, q_{new})$
5: **if** $q_{new} - q$ **then**
6: *return Reached*
7: **else**
8: *return Advanced*
9: **end if**
10: **else**
11: *return Trapped*
12: **end if**

In line 1, the nearest vertex to the added configuration q is found. Then, the function NEW_CONF moves towards q a predefined step of Δq. The collisions are checked and, if the step is in C_{free}, the configuration q and its edge are added to the RRT tree. Algorithm 8 returns *Reached* when q_{new} is equal to q. *Advanced*, is returned when $q_{new} \neq q$. *Trapped*, is returned when q_{new} is not free of collisions. Finally, the output of Algorithm 7 is the RRT tree $\mathcal{T}$ data structured. The basic RRT is designed only to explore state space, but with a little modification it can be tailored into a path planning method. Specifically, the loop in Algorithm 7 can be

stopped when a q_{goal} is reached. The tree can be traversed from q_{goal} to q_{init} in order to find a path.

In Figure 2.15, it can be observed an example of an RRT planner in 2D. The edges and vertexes are presented as thin red lines. A rectangular robot is placed at each node of the path. The generated path is the red thick line described by the shape of the robot.

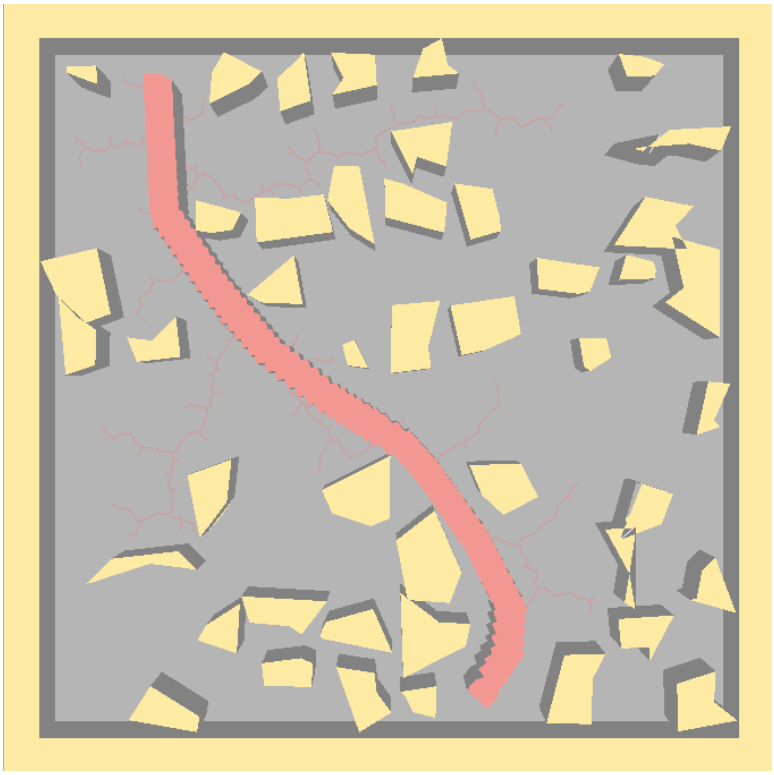


Figure 2.15: Example of an RRT search and generated path.

RRT-Bidirectional

The bidirectional version of the RRT was developed to directly suit path planning problems (LaValle [28]). The RRT-Bidirectional, also called RRT-Connect, builds two RRT trees, one starting in q_{init} and other from q_{goal}. The goal of the method is to find a common state to both trees. Given two configurations q and q' and a metric distance between them $\rho(q, q')$, they are considered to be common when $\rho(q, q') < \epsilon$ for a metric ρ and $\epsilon > 0$.

The bidirectional RRT is described in Algorithm 9. It incorporates an additional greedy approach to the basic RRT. Half of the times trees are grown to explore the

state space, and the other times they are grown towards each other. The $CONNECT$ function is a greedy heuristic. This function is presented in Algorithm 10.

Algorithm 9 RRT_BIDIRECTIONAL

Input: q_{init}, q_{goal}, K, Δq
Output: T

```
1: Ta.init(q_init); Tb.init(q_goal);
2: for all k ∈ K do
3:    q_rand ← RAND_CONF()
4:    if not(EXTEND(T_a, q_rand) = Trapped) then
5:       if (CONNECT(T_b, q_new) = Reached) then
6:          Return PATH(T_a, T_b);
7:       end if
8:       SWAP(T_a, T_b);
9:    end if
10: end for
11: Return Failure
```

Algorithm 10 CONNECT

Input: $\mathcal{T}$, q
Output: S

```
1: while not(S-Advanced) do
2:    S ← Extend(T, q)
3: end while
4: Return S
```

For the bidirectional RRT method (Algorithm 9), the iteration loop starts in line 2. A random configuration q_{rand} is generated in line 3. In the following line, the tree $\mathcal{T}_a$ is grown with the basic RRT function $EXTEND$ (see Algorithm 8). If the result is different from $Trapped$, the new configuration q_{new} is passed to the function $CONNECT$. This function extends the tree towards the new configuration of the other tree q_{new}. The connect loop iterates until the result is either *Reached* or $Trapped$. If the result is *Reached*, the algorithm returns the path. If the result is $Trapped$, the trees are swapped and the loop is repeated.

In Figure 2.16, an example of an RRT-Bidirectional planner inside a 2D maze can be observed. The edges and vertexes are represented as thin lines. The green lines belong to the edges of the q_{init} tree that starts at the bottom left corner. The blue lines belong to the edges of the q_{goal} tree that starts at the top right corner. A human-like robot is placed at each node of the path. The generated path is the red thick

line described by the shape of the robot. The RRT-Bidirectional generally performs better than the basic RRT(for reference see Cohen[43]). However, they had similar performance for this particular experiment of a maze with a unique solution. The basic RRT had a little advantage in time response over the RRT-Connect. Although, the RRT generated more states than the RRT-Bidirectional to obtain the solution.

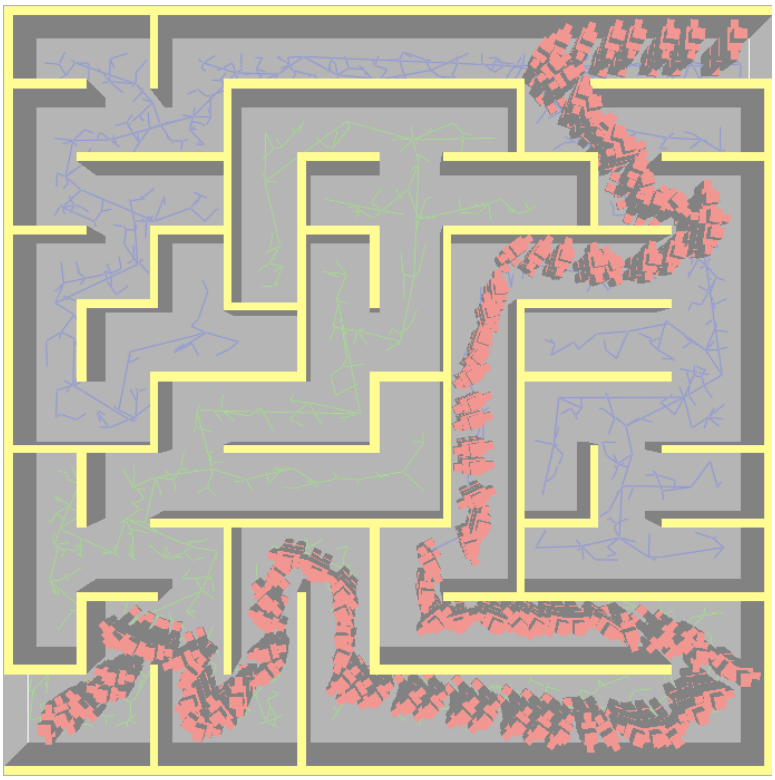


Figure 2.16: Example of an RRT-Connect generated path trough a maze.

On the downside, the bidirectional approach generates a discontinuity in the union between the two trees. This abrupt change in the direction of the path disfavors the use of the method with nonholonomic vehicles.

Nonholonomic RRT

The nonholonomic variant of the RRT (RRT-NH), was introduced by LaValle and Kuffner in the article titled "Randomized Kinodynamic Planning" [44]. It was the original approach for the RRT, and it contemplated the steering constraints of a car-like robot. In other words, a kinodynamic vehicle for which control parameters have to be calculated. This usually adds two more dimensions to the path planning problem, even though the calculations are often loosen. The curse of dimensionality

has brought researchers to focus on planning methods in the C-space. The additional parameters are built over the resulted path of these kind of methods. The C-space motion planning, so-called classic, reduces complexity and usually offers guaranties of convergence. In Figure 2.17, the nonholonomic problem is illustrated.

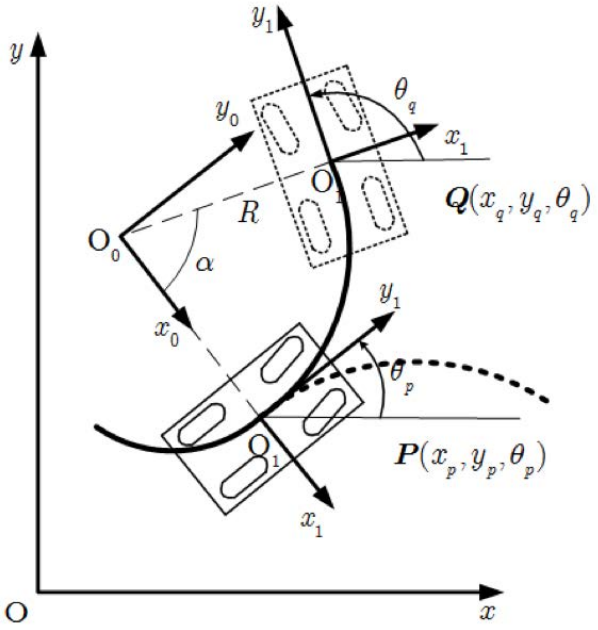


Figure 2.17: Constraints in a path of a non-holonomic vehicle [45].

As mentioned in the RRT-Bidirectional section, the bidirectional approach of the RRT is not directly suited for nonholonomic vehicles. This is because the bidirectional approach will normally generate a discontinuity in the trajectory at the place where the two trees are connected. The authors of the method proposed some techniques to perturb the generated path and make it continuous. Figure 2.18 (right) presents a bidirectional run of the algorithm. The RRT-NH will need additional time to make a coherent connection between trees regarding the nonholonomic constraints. This fact could worsen the algorithm performance. Anyhow, in [45] they propose an approach to nonholonomic planning with an RRT-Bidirectional. The method works as Algorithm 9, but when the $CONNECT$ function is called, the tree structure advances until it is trapped or the new node is within the reach of the nearest neighbor. In the case of being in reach of the other tree, an attempt to connect both trees is made. The connection is only possible is the nonholonomic constraints are met. If a solution is found, the trees are combined and a solution path is returned. Otherwise, the new configuration q_{new} is removed from the tree structure and the algorithm continues.

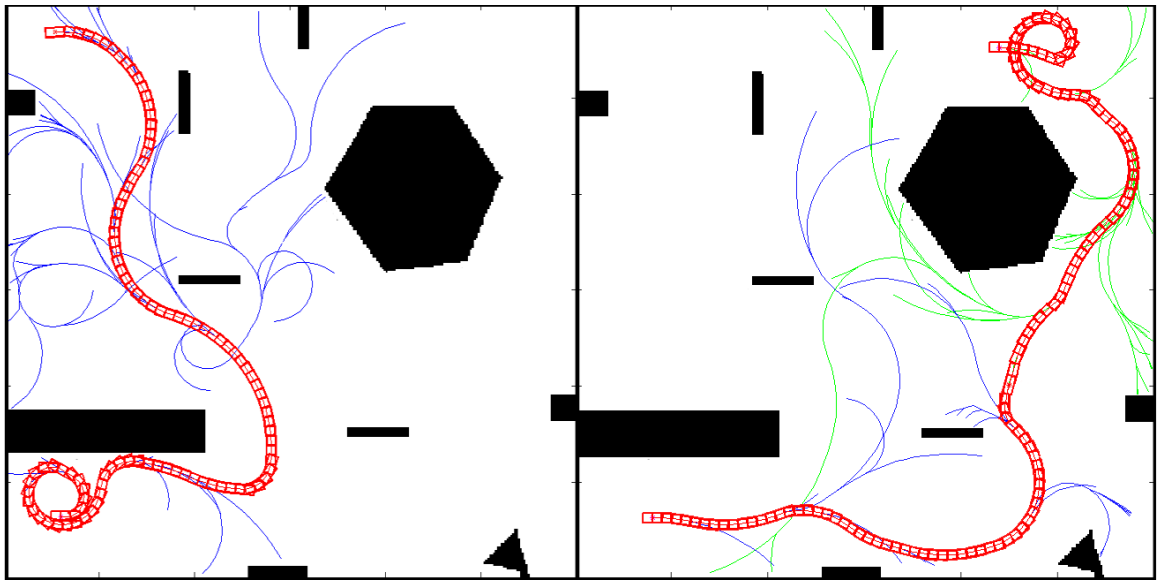


Figure 2.18: Example of RRT-NH searches and generated paths. On the left the basic RRT-NH and on the right the bidirectional variant.

In Figure 2.18 (left), it can be observed an example of an RRT-NH planner in 2D. The edges and vertexes are presented as thin blue lines. A rectangular robot, which can only move in one direction, is placed at each node of the path. The generated path is the red line and the contour of the robot. The method took 37.55 seconds to generate the path and generated 695 states.

2.2.3 Anytime Algorithms

The anytime motion planning algorithms are methods that depend on time. First, they must quickly find some motion plan that is feasible but not necessarily optimal. Then, the plan is incrementally improved over time toward optimality. An important aspect of the time-dependent algorithms is to determine the best instant to make available the needed results. Certainly, the time consumed by an algorithm depends on the complexity of the algorithm and other variables, such as dynamic inputs that may or may not be available at a certain moment. In some applications, it may be crucial to have a solution before an specific time threshold, no matter the quality of the results. These kind of conditions are fulfilled by robotics navigation, where a robot is usually moved slowly by safety reasons. Thus, to start the navigation as soon as possible can earn valuable time, no matter the imperfections of the initial path. Besides, to start the navigation with an optimal path does not guarantee the complete

execution of the path as new obstacles could be introduced in the environment and the path would have to be recalculated. Furthermore, the computational time for calculating an optimal large path would be long, and all the time the robot would be standing still. Under these circumstances, it is very likely for a robot navigating with suboptimal paths to make it to the goal before one that calculates the optimal route. Therefore, the requirement for the algorithm is to compute the best solution possible in the available time. There should be a response whether this time is very reduced or is greater than average. In Figure 2.19, a simulation of an anytime RRT algorithm (left) and an a variant called RRT* are presented. The different colors represent the time of the simulation when a better path is found.

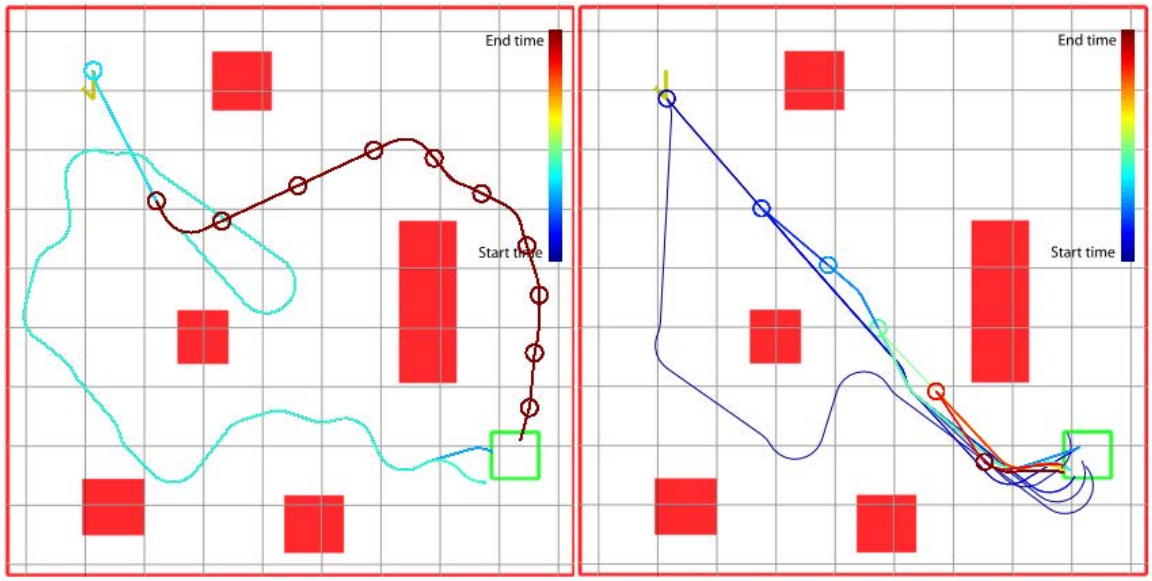


Figure 2.19: Example of RRT anytime algorithms. On the left the basic approach and on the right a more advanced RRT variant (RRT*). Figure taken from [32].

Any robotic motion planning algorithm intended for practical use must operate within limited real-time computational resources and incomplete and imperfect knowledge of the environment. Such settings favor anytime algorithms.

Anytime algorithms, also called interruptible algorithms, were first mentioned in 1988 by Thomas Dean *et al* [46]. They can be interrupted at any moment and return an answer. This approach favors robotic applications where having an answer too late is equivalent to not having an answer at all. For instance, a robot that receives objects from a conveyor with irregular frequency has to recognize an object coming with enough time in order to pick it, or else it would be too late and the object would already have passed by. Planning long paths can be benefited from these kind

of algorithms because a mobile robot or car is able to gain time by start moving right away and improving the path on the fly, specially due to the fact that speed is often reduced for these applications. Another suitable area for anytime algorithms is video games development [47]. For example, Artificial Intelligent (AI) characters should calculate long paths, and standing still for long time while an optimal solution is calculated would make the character look less intelligent and perhaps the player would get bored waiting for the AI character to take action.

Anytime algorithms have been successfully applied to robotics races with real cars and other robotic tasks in order to quickly generate a first path and to improve it in an incremental way when time is available. In [48], an algorithm based on RRT denominated Anytime Dynamic RRTs is introduced, but its generated paths are suboptimal, not soft, and bring the robot dangerously near obstacles. In [32], an anytime motion planner was proposed using RRT*. This algorithm achieves asymptotic optimality without incurring in substantial computational overhead, but they assume that the environment is known, which is unrealistic for environments where people are constantly moving and objects location may change. Also the algorithm could lead the robot through a suboptimal path if optimization time is too reduced, and even though presented examples do not appear to be unsafe, safety is not mentioned. Some of the feasible solutions produced by other planners tend to be far from optimal, as mentioned in [32]

2.3 Evolutionary Strategies

In the chapter concerning manipulation (Section 4.1), a method to adapt already learned manipulations paths is presented. The proposed approach is based on the evolutionary strategies, which are iterative optimization methods which try to find an optimal solution from stochastic small variations of the design variables [49]. Evolutionary strategies are based on the principles of natural selection: the individuals of a species mutate from generation to generation by small variation of their genes, and only the best fitted to their environment will survive and be selected for further reproduction. The analogy with the optimization problem is quite straightforward: a set of design variables can be considered as the genes of an individual and the value of the objective function for this set of design variables represents the fitness for survival of the corresponding individual.

The basic implementation of the evolutionary strategy is composed of the following steps:

1. An initial population of μ parent individuals is selected at random and uniformly in the feasible range of each design variable. Ideally, each initial parent should hold the constraints.

2. A population of λ offsprings is created from mutation of the parents. For each offspring, a parent is randomly selected, and each of its design variable b_i is mutated by adding a Gaussian random variable with zero mean and preselected standard deviation σ_i

$$b_i(offspring) = b_i(parent) + N(0, \sigma_i), i = 1, n_b \tag{2.2}$$

   The standard deviation σ_i may be chosen for the whole population or may be linked to the individual, in a similar manner as the design variables. A Gaussian distribution insures that, like in the nature, small changes occur more frequently than large ones.

3. The new parents are selected as the μ individuals with the best fitness, thats to say with the lowest cost function. The new parents may be chosen either from the set of parents and offsprings (plus strategy $\mu + \lambda$) or only from the offsprings (comma strategy μ, λ).

4. The process of mutation-selection continues until a satisfactory solution is reached. The convergence criterion will be explained later.

Some practical rules in the choice of the most important parameters are described in the following subsections.

2.3.1 Size of the population

There is so far no accurate rule to determine the size of the parent population. A good indication is to have as many or a few times as many members as the number of design variables. On the other hand, some theoretical studies have been realized on $(1, \lambda)$ strategies to estimate the optimal ratio λ/μ [50]. It has been shown that this optimal ratio depends on the objective function and increases with its complexity. A ratio λ/μ equal to 5 can be considered as a good starting point.

2.3.2 Step Length Control

The standard deviation σ_i plays an important role as it permits to control the speed of convergence. As it also corresponds to the mean variation of the corresponding parameter, it is often called the step length. Like in all optimization procedures, the control of the step length is the most important part of the algorithm after the recursion formula. The theoretical study of the two membered evolutionary strategy [50] leads to the so-called success rule of Rechenberg, which stated that after every n_b (number of design variables) iterations, check how many successes have occurred

over the preceding $10 \times n_b$ mutations. If this number is less than $2 \times n_b$, multiply the step lengths by the factor 0.85; divide them by 0.85 if more than $2 \times n_b$ successes occurred. For the initial value of the step lengths, Schwefel proposes to use the following estimation

$$\sigma_i^0 = \frac{\Delta b_i}{\sqrt{N_b}} \tag{2.3}$$

where Δb_i is the expected distance from the optimum for the corresponding design variable. The accuracy on this initial value is not critical as the success law seems to rapidly adapt the step length to a suitable value, at least when it is too large.

2.3.3 Recombinations

The basic evolutionary strategies can be enriched by adding a supplementary step of recombination between the parents before mutation. Pairs of parents are randomly selected and are recombined to yield a new set of μ parents for mutation. According to the chosen type of recombination, the design variables or the standard deviations are determined as:

- The mean arithmetic or geometric value of the corresponding parameters of the two selected parents (arithmetic or geometric recombination).
- The corresponding parameter of one or the other parent, selected at random with equal probablity (discrete recombination).

Recombinations introduce the principle of sexual propagation which is expected to be very favorable for evolution as only few primitive organisms do without it. As mentioned before, it also offers the possibility to independently vary the step lengths of each design variable.

2.3.4 Convergence Criteria

In the two membered strategy, the convergence criterion is based on the evolution of the best value of the objective function along generations. The optimum is assumed to be reached as far as the best value has not significantly changed in the last generations. With the multimembered strategy, the criterion still becomes simpler. The optimum is assumed to be reached as soon as the best individuals of a generation do not differ too much with respect to their objective function values. If we denote 0,min and 0,max the minimal and maximal values of the objective functions inside a given parent generation, the iterative process will be stopped if

$$|\psi_{0,min}\psi_{0,max}| < \varepsilon, \tag{2.4}$$

A relative error criterion can also be used as

$$|\psi_{0,min}\psi_{0,max}| < \varepsilon\psi_0, \tag{2.5}$$

where ψ_0 is the mean value of the objective function in the considered generation. To avoid endless processes, it is also safe to impose a maximum number of iterations after which the optimization is automatically stopped, and eventually adapted for further investigations.

2.3.5 Constraints

Ideally, the initial parent population should hold the constraints. However, it can be difficult to find a uniformly distributed initial population which respects all the constraints. This condition can then be dropped, the constraints being taken into account through a penalty of the objective function as soon as a constraint is violated. In this way, the selection generally makes the parent population licit in a few generations. Although simple, this approach wastes computation time in the first generations and introduces the risk of having a population coming only from a small number of licit initial parents. The problem of finding an initial licit population is therefore often treated as a particular optimization problem whose cost function ψ'_0 corresponds to the sum of the violated constraints

$$\psi'_0 = \sum_{n_c}^{j=1} \psi_j(b).\,\delta(\psi_j(b)) \quad with\,\delta(x) = \begin{cases} 1, & if \quad x < 0 \\ 0, & otherwise. \end{cases} \tag{2.6}$$

If an illicit starting point arises during the construction of the initial population, the process is performed from that point, until all the constraints are satisfied. This initialization step is easy to implement, as it uses only available tools, and yields a well distributed population.

2.4 Robotic Manipulation

The action of manipulation can make reference to a modification or change of something by any mean. In robotics, it mostly refers to physically move an object, usually in pick and place tasks. The process include many phases. The first one is reaching, which involves planning a trajectory to place the end-effector of the robot in front or at a distance where the robotic hand, clamp or other robotic tool can seize the object. After it is reached, the robot has to grasp the object. This task takes into account the form, weight, and the stiffness of the object. A trajectory has to be calculated in order to close the robotic fingers with the adequate strength to hold the object

avoiding to drop or smash it. This is often achieved with force sensors included in the end-effector of the robot. When the object is correctly gripped by the robot, the planner has to generate a trajectory to take the object to another location. Then, a plan for opening the grasp is executed, releasing the object on the target location. This plan has to consider the shape of the object in order to stand safely, as it could be fragile (for instance a glass). In Figure 2.20, a mobile manipulator in a simulation of space tasks can be observed. The figure was taken from the IEEE Spectrum[3].

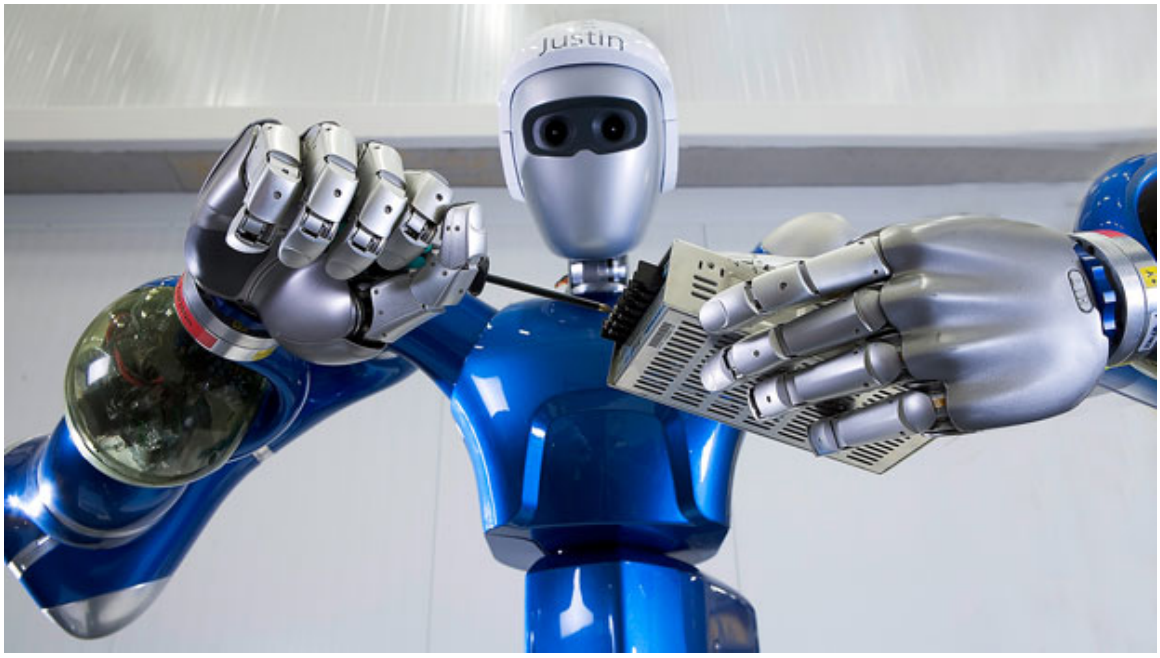


Figure 2.20: Mobile manipulator robot Justin from the German Aerospace Center (DLR). Figure taken from IEEE Spectrum, reference at the footnote.

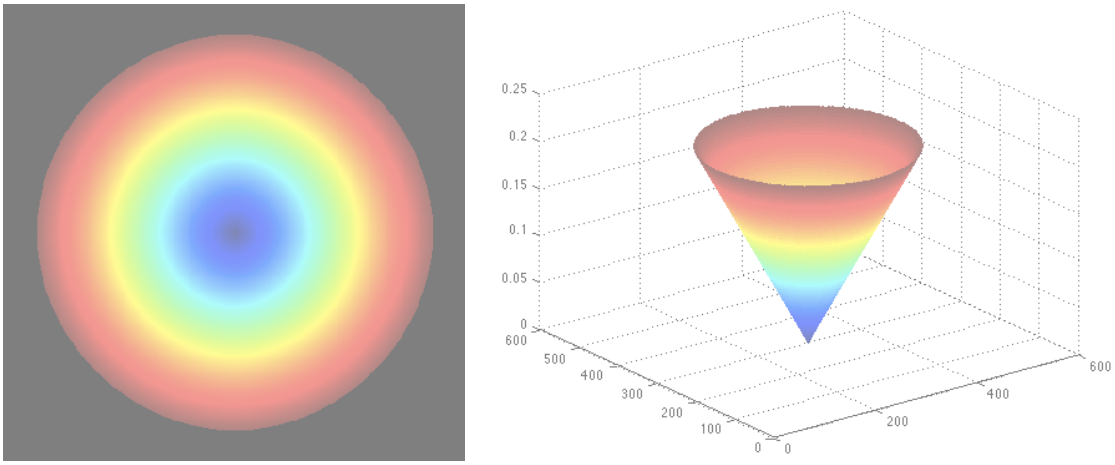
As can be observed, the stages involve in the manipulation task need the generation of many motion plans.

There are other forms of robotic manipulation, as pushing, throwing and other specific applications, where the robot makes use of a tool. In this work, the robotic manipulation presented in Chapter 4 will be mainly focused on the reaching process.

[3] `http://spectrum.ieee.org/automaton/robotics/humanoids/space-justin`

2.5 Fast Marching Method

This section discusses the main concepts and techniques related to the FMM for path planning in homogeneous and inhomogeneous environment maps, and how the refraction map is calculated for the FM^2 method, which it is also explained. Instead of presenting these search-based methods in Section 2.2.1, this entire section is devoted to them as they are of great relevance to this work developments. The disadvantages of using the basic FMM version and the benefits of using it with a velocities map are also mentioned. Both methods, the FMM and the FM^2, will be described in the next sections. The FMM on triangular meshes and the FM^2 algorithm will be detailed in Subsections 2.5.3 and 2.5.4 respectively.



(a) FMM expansion wave in 2D.

(b) FMM expansion wave in 3D.

Figure 2.21: Representation of FMM expansion waves, where the third axis is the Eikonal value calculated by the algorithm.

2.5.1 The Basic Method

The principle behind the FMM is the expansion of a wave. In two dimensions, the method intuitively simulates the spreading of a thick liquid as it is pour into a board, obtaining the time in which the front achieves every point of the grid. Similar formulations have been used in other study areas like Fluids Mechanics, Molecular Dynamics in relation to Electrostatic, Thermal Analysis, and more. Notwithstanding, it is crucial to highlight that the most important and peculiar feature of the method is how the wave expansion is calculated in reaching time for every cell in a grid. As a consequence of its particular mathematical formulation, the outputted potential map

of the method presents only a global minimum and no local minimum whatsoever. There are many preceding graph search algorithms based on similar approaches like Dijkstra and A^* (see [22] and [23] respectively). These search methods have been widely used and demonstrated. Conversely, they have been proven to be inconsistent in the continuous space [51]. In Figure 2.21(a), the expansion of a wave is presented in 2D. Figure 2.21(b) presents the expansion in 3D where the third axis is the Eikonal distance from the source.

The FMM in an homogeneous environment generates, at same levels of the wave, front interface points in circular form and centered around the source location. In such case, all the points in the interface are reached at a given time homogeneously, and the minimal path between two points in the space is always composed of straight lines.

The method foundation is the same as the one behind Fermat principle in optics, which states that a ray of light that goes through a prismatic glass always takes the fastest path between any two points. In other words, it takes the minimum or optimal path in time. The interface or wavefront can be a flat curve in 2D, a surface in 3D, or even although it may not be possible to represent it graphically, the model can be mathematically generalized to any number of dimensions. The time T is calculated for every point as the wave advances and covers the gridmap. The front, denominated Γ, advances always moving in the normal direction. The FMM allows to receive even more than one source point as input, then the front wave is generated from each source point. The interface origin points are initialized with $T = 0$ and frozen state, according to the names of the algorithm states [4]. For obtaining the geodesic path over a map the source point must be only one, which stands for an only global minimum $T = 0$, implying that the rest of the values will be always greater than zero. The speed F is established by the velocity potential map, and may vary from point to point but it is always positive, or equal to zero in obstacles grid points. The values of the front are described by the Eikonal equation, as given by Sethian [27]:

$$1 = F(x)|\nabla T(x)| \tag{2.7}$$

where x is a point in space, $F(x)$ is the speed of the wave for that position, and $T(x)$ is the time required by the wave interface to reach x. Then, the velocity is inversely proportional to the gradient magnitude of the arrival time function $T(x)$:

$$\frac{1}{F} = |\nabla T| \tag{2.8}$$

Algorithm 11 Algorithm of the FMM

Input: A grid map G of size $m \times n$, source point x_0
Output: The grid map G with the T value set for all cells

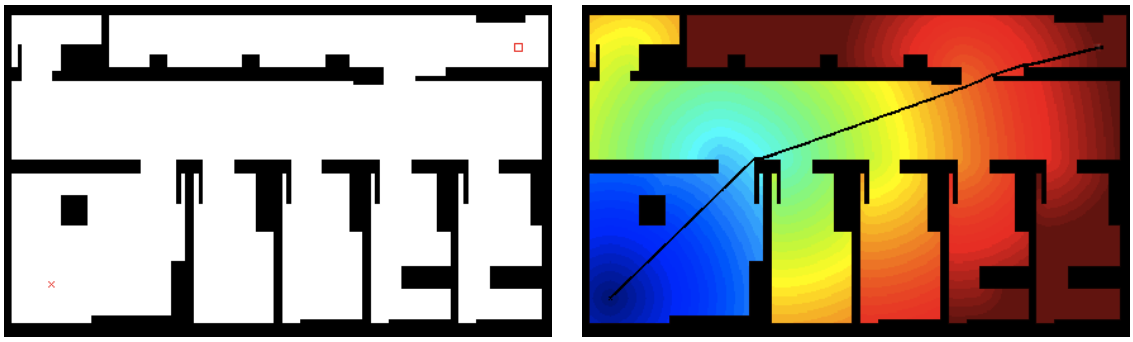
```
    {Initialization}
 1: for all g_ij ∈ x_0 do
 2:   g_ij.T ← 0;
 3:   g_ij.state ← FROZEN;
 4:   for all g_kl ∈ g_ij.neighbours do
 5:     if g_kl = FROZEN then
 6:       skip;
 7:     else
 8:       g_kl.T ← solveEikonal(g_kl);
 9:       if g_kl.state = NARROWBAND then
10:         narrow_band.update_position(g_kl);
11:       end if
12:       if gkl.state = UNKNOWN then
13:         g_kl.state ← NARROWBAND;
14:         narrow_band.insert_in_position(g_kl);
15:       end if
16:     end if
17:   end for
      {Loop}
18:   while narrow_band NOT EMPTY do
19:     g_ij ← narrow_band.pop_first()
20:     for all g_kl ∈ gij.neighbours do
21:       if g_kl = FROZEN then
22:         skip;
23:       else
24:         g_kl.T ← solveEikonal(g_kl);
25:       end if
26:       if g_kl.state = NARROWBAND then
27:         narrow_band.update_position(g_kl);
28:       end if
29:       if g_kl.state = UNKNOWN then
30:         g_kl.state ← NARROWBAND;
31:         narrow_band.insert_in_position(g_kl);
32:       end if
33:     end for
34:   end while
35: end for
```

2.5.2 Implementation and Path Planning

The solution for the Eikonal equation can be computed iteratively over a gridmap. The pseudo-code algorithm for the method is shown in Algorithm 11. Before going into the details, we explain the different labels the cells of the gridmap can take:

- Unknown: cells whose T value is not yet known because the wavefront has not reached them.
- Narrowband: cells that may be part of the front wave in the next iteration. They already have a T value assigned but it can change in future iterations of the algorithm.
- Frozen: cells whose T value is fixed because they have been passed over by the wave.

The algorithm has three stages: initialization, loop and finalization. In the initialization $T = 0$ is set in the cell in which the wave originates and this cell is labelled as frozen. Afterwards, all its Manhattan neighbours are labelled as narrow band and T is computed for each of them. The iterator g_{ij} is used to cover the grid wave. The variable g_{kl} is used to visit the Manhattan neighbours and calculate their T values.



(a) Environment map. (b) FMM wave expansion and calculated path.

Figure 2.22: Example of FMM with the calculation of the path.

In each iteration of the loop in Algorithm 11, the Eikonal equation is solved for the Manhattan neighbours (which are not labelled as frozen) of the cell in the narrow band which has a lesser T value, and this cell is then labelled as frozen. The narrow band consists of an ordered list, from lowest to highest T value, of its cells. The finalization is reached when all the cells are labelled as frozen. The output is a potential map with a reach time value (T) for every cell. If the descent gradient is applied, it leads to the shortest path in time, which in a map with homogeneous velocity is the same as the

shortest path in distance and it is called the geodesic. Figure 2.22 shows an example of the resulted geodesic path after applying the FMM between two given points in a 2D environment. The darker red color represents the unknown or unexplored areas.

In path planning, the FMM expansion of the wave is applied directly over a plain map. Albeit the result correspond to the minimum length geometrical path, the paths are neither safe nor smooth.

2.5.3 Fast Marching Method on Triangular Meshes

The basic idea of this approach is related to the Fermat principle in optics, which states that a ray of light that goes through a prismatic glass always takes the fastest path between any two points. In other words, it takes the minimum or optimal path in time. The interface or wavefront can be a flat curve in 2D, a surface in 3D, or even although it may not be possible to represent it graphically, the model can be mathematically generalized to any number of dimensions. The time T is calculated for every point as the wave advances and covers the gridmap. The front denominated Γ always advances in the normal direction. The FMM allows receiving even more than one source point as input, then the front wave is generated from each source point. The interface origin points are initialized with $T = 0$ and frozen state, according to the names of the algorithm states. For obtaining the geodesic path over a map the initial point must be unique, which stands for an only global minimum $T = 0$, implying that the rest of values will be always greater than zero. The speed F is established by the velocity potential map, and may vary from point to point but it is always positive or equal to zero in obstacles. The first versions of the FMM were based on regular orthogonal grids [52, 4]. Later, these algorithms were extended to general triangular meshes [53]. Since triangular meshes are more flexible when describing shapes, this version of the FMM was chosen.

In this section we introduce the FMM that is used in the proposed approach. Let X be the surface defined by a triangle mesh and x a coordinate parameterization of $X, x : U \to X$. A distance map $d(x) = d_X(x_0, x)$ is build with the FMM by solving the Eikonal equation:

$$\|\nabla_X d(x)\|_2 = 1 \tag{2.9}$$

where ∇_X represents the intrinsic gradient with the boundary condition $d(x_0) = 0$.

As said before, the FMM simulates a wavefront propagation calculating the time or distance of arrival d(x) for every point of the map when the wave propagates with constant, non-negative velocity. Let us suppose that a wave starts propagating at x_0 with $d(x_0) = 0$. This point is already *frozen* (its value will never change). By *open* points we call those points of the mesh which have not been visited yet by the wave ($d(x) = \infty$). Finally, only the wave front points remain in the *narrow* band, which

separates the frozen and open points. Algorithm 12 describes the procedure followed by the FMM to generate the distance map d.

Algorithm 12 FMM over a triangular mesh

Input: non-obtuse triangular mesh (X, T), source point x_0.
Output: distance map $d : X \to \mathbb{R}$ from the source point.

Initialization
1: **for all** $x \in X$ **do**
2: $d(x) \leftarrow \infty$;
3: **end for**
4: $d(x) \leftarrow 0$;
5: $frozen \leftarrow x_0$;
6: $narrow \leftarrow N(x_0)$;{Neighbours of x_0}
7: $open \leftarrow X \setminus (frozen \cup narrow)$;
Loop
8: **while** $frozen \neq X$ **do**
9: $x_1 \leftarrow \arg\min_{x \in narrow} d(x)$;
10: **for all** $t(x_1, x_2, x_3) \in \{(x_1, x_2, x_3) \in T : x_2 \in frozen \cup narrow, x_3 \in narrow \cup open\}$ **do**
11: $narrow \leftarrow narrow \cup \{x_3\}$;
12: $Update(x_1, x_2, x_3)$;
13: **end for**
Remove x1 from the unprocessed set
14: $narrow \leftarrow narrow \setminus \{x_1\}$;
15: $frozen \leftarrow frozen \cup \{x_1\}$;
16: **end while**

The update function (line 12 of Algorithm 12) is a key component of the algorithm because it is what differentiates the method from Dijsktra's, making possible to use it in a continuous surface [54]. The update process, which is shown in Algorithm 13, is perform over two vertices of a triangle to calculate the time of arrival of the third vertex. This enables the geodesics to follow the gradient of the distance map $d(x)$, and reach any vertex within the mesh.

The update process only works with non-obtuse triangular meshes. If there is any non-obtuse triangle in the mesh, a possible solution is to connect the vertex x_3 to another point of the mesh [53].

Algorithm 13 FMM update function

Input: non-obtuse triangle with x_1,x_2,x_3, and the corresponding arrival times d_1,d_2,d_3.

Output: Updated distance d_3.

$V = (x_1 - x_3, x_2 - x_3)$;
$d = (d_1, d_2)^T$;
$Q = (V^T V)^{-1}$;
$p = \frac{1_{2\times1}^T Q_d + \sqrt{(1_{2\times1}^T Q_d)^2 - 1_{2\times1}^T Q 1_{2\times1} \cdot (d^T Q_d - 1)}}{1_{2\times1}^T Q 1_{2\times1}}$;
$n = V^{-T}(d - p \cdot 1_{2\times1})$;
if $QV^T n < 0$ **then**
    $d_3 \leftarrow \min\{d_3, p\}$;
else
    $d_3 \leftarrow \min\{d_3, d_1 + \|x_1\|, d_2 + \|x_2\|\}$;
end if

2.5.4 The Fast Marching Square Method

This section presents the fundamentals of the method used in the great majority of the presented approaches. The FM^2 was first introduced by Garrido *et al.* in [29]. The method was presented as concerns for the FMM safety in path planning were raised. An approach using Voronoi diagrams was proposed before by the same authors in [55]. In spite of the safety offered by the Voronoi Diagrams, the calculated paths often result unnecessarily long. Differently, the FM^2 is able to adjust the range of the security distance or clearance and reduce the length of the paths.

The FM^2 takes advantage of the ability of the FMM to be calculated over an anisotropic map environment. This means that not only the method is valid for maps of zeros (obstacles) and ones (free space), but gray levels are also admitted in the calculation. The intermediate values in the map function represent velocities. These values influence over the Eikonal distance values. When the geodesic is determined, the fastest map areas are drawn in the final path. Essentially, the FM^2 applies FMM to an input environment map in order to create a velocities potential map. Then, over this map, the FMM is applied a second time to expand a wave from a starting point and until reaching the goal. The descend of the geodesic is calculated to determine the pathway. In Figure 2.23, two examples of the FM^2 are shown.

The 2D maps on the left of Figure 2.23 represent the velocity potential maps. Figures on the right are their respective expansion waves from a given start to goal points. The latter velocities potential map was saturated to reduce the clearance. The obtained paths are depicted in black for both cases.

The original FMM approach assumes the isotropy of the free space or, stated in

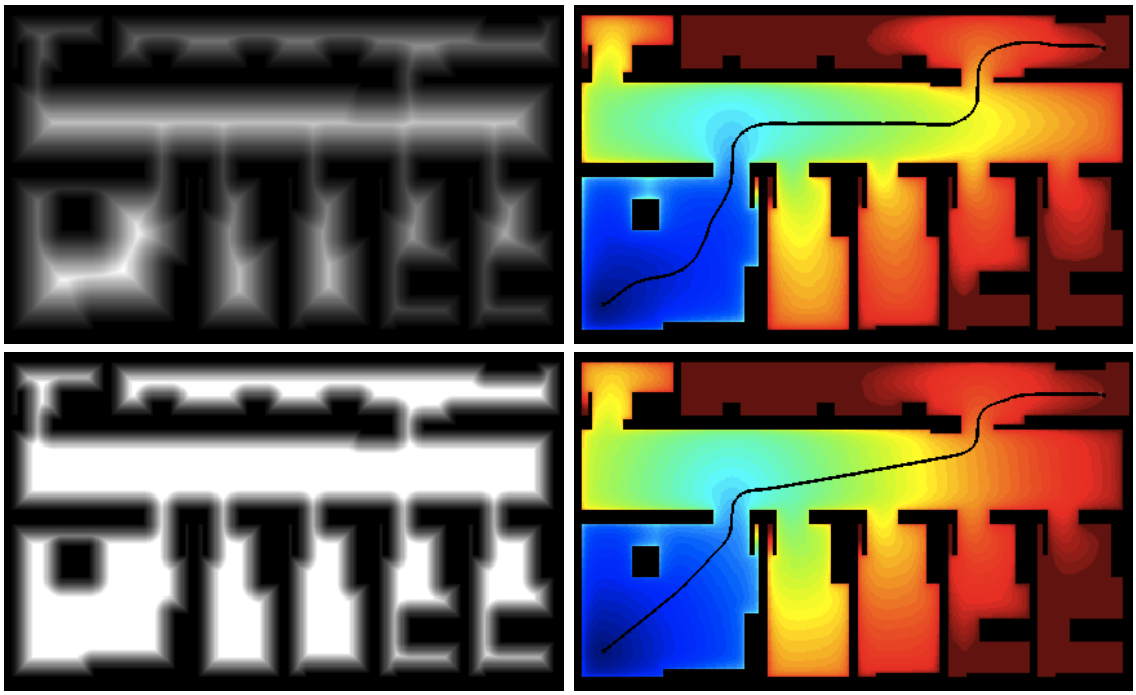
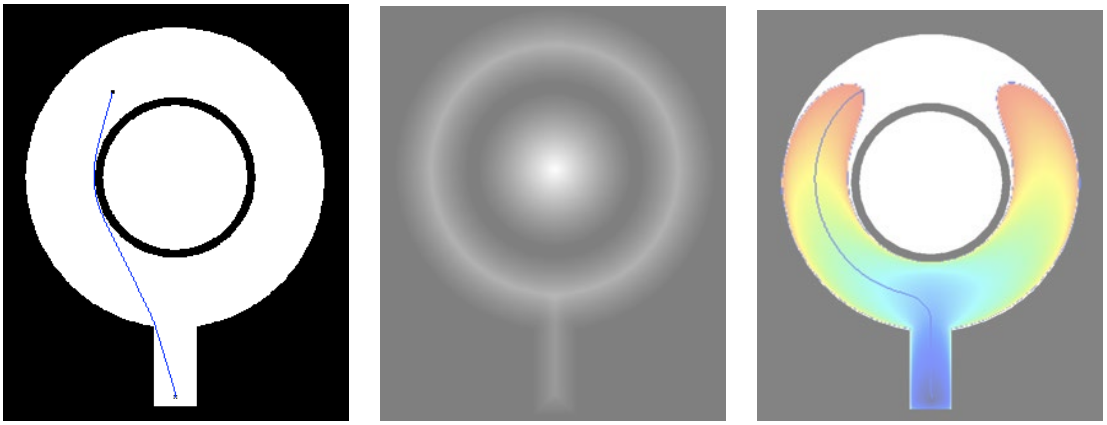


Figure 2.23: Motion trajectories obtained by the FM^2. Example of FM^2 with both stages: the generation of the velocities map (left) and the calculation of the path (right). The velocity potential map is saturated on the bottom example.

another way, the environment map has a uniform value for the entire environment where obstacles are not present. Nonetheless, the FMM time arrival values can be obtained over an uneven map. This concept was first demonstrated by Sethian *et al.*[56]. This fact has been taken into account in the FM^2 method to define a velocity potential map. This approach, produces paths with a safety distance from obstacles that can be adjusted. In Figure 2.24(c), a path planning example between two points in a 2D environment is presented. In (a), the resulted path of applying the FMM is shown. In (c), the FM^2 path for the same experiment is shown, and (b) represents its corresponding velocity potential map.

In the next paragraphs, the basic steps that comprise the FM^2 algorithm are depicted:

1. *FM^2-1st step.* A first run of the FMM is carried out with the map resulted from the previous steps. The gridmap is passed as an input parameter. In this particular case, the sources are all the obstacles present in the gridmap. The output will be another gridmap with the arriving values, as indicated in Algorithm 12, where the potential interfaces start from walls and obstacles. This map is better known as velocity potential map, and as its name suggests it



(a) Geodesic path with FM. (b) Velocity potential map. (c) Path with the FM^2 method.

Figure 2.24: Example of FM^2 with both stages: the generation of the velocity potential map and the calculation of the path.

establishes a maximum speed for every point in the map that should be taken into account when moving the real robot. Furthermore, each cell value in this potential map gives the distance to the nearest obstacle in time, data that can be employed as a clearance metric because it is related to the geometric distance. Besides, in the case of a breakdown, it can be understood as the time available to stop the robot before colliding. Therefore, the velocity potential map also provides a safety framework, not only by considering the distance to the closest obstacle but also by providing a secure reference speed for each point of the path. Figure 2.24(b) shows the FMM wavefront emerging from walls and obstacles, which corresponds to the example environment velocity potential map.

2. *Saturation of velocity potential map.* As an additional step, the saturation of the potential map was proposed in [57] to avoid getting too far from walls and other obstacles. A safety distance is established and used to saturate the map. The potential is the same for all the cells whose distance is greater than this safety threshold. The reasoning here is that maintaining a clearance greater than a predefined safety distance, would only increase the path length unnecessarily. This mechanism has not been included here, but it is an option to consider depending on the planner purpose.

3. *FM^2-2nd step.* The last step of the FM^2 method is to generate an additional FMM expansion over the potential map. The result of this step is a 3D surface in the case of 2D planning, and for n dimensions the FMM potential will give an additional axis $n + 1$.

4. *Gradient Descent.* After calculating the potential map, in the case of a 2D environment, the surface is used to obtain an optimal minimal path in expansion time of the wave. This is done by following the gradient descent direction of the wavefront potential from the target to the initial point. The obtained path corresponds to the geodesic path of the surface, and the velocity for each point is defined in the velocity potential map reference frame.

If the expansion of the wave were applied directly over a homogenous map without the velocity potential map generated in the FM^2-1st step, the results would then correspond to that of the shortest path, which is the basic FMM path and is neither safe nor smooth (Figure 2.24(a)).

Summarizing, the FM^2 method guarantees to find a solution path if it exist, and the result is optimal in length, safety distance and smoothness.

2.6 Comparison of Path Planning Methods

Regarding the methods explained in the last sections, some comparisons between them are provided here to give the reader an idea about their capabilities.

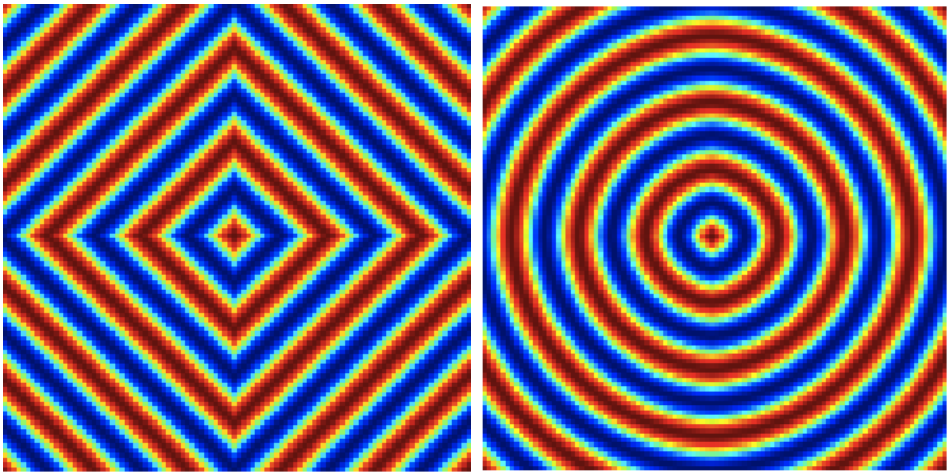


Figure 2.25: Geodesic distance maps using a cosine modulation to denote the level set. The Dijkstra algorithm generated map on the left, and the FMM on the right.

The RRT algorithm complexity is $O(n \log n)$ and presents no asymptotic optimality. In practice, this algorithm generates far from optimal paths and does not handle environments well when they have too many obstacles, which is very frequent in navigation planning. In particular, it presents problems when trying to pass a robot through a narrow passage. As was found in [43], where for this specific experiment, the RRT obtained the lowest success rate, and the computational time arose

to many seconds. In the RRT and its variations, safety is not considered beyond collision checking. However, the advantage of the method is the ability to handle high dimensional problems. This favors manipulation motion planning problems where the arms usually have seven DOF. Additionally, the environment for manipulation often presents very few obstacles, which quickens the convergence of the algorithm.

For search-based planners like Dijkstra, or A*, the problem is that they are not consistent in the continuos space [54]. In Figure 2.25, the geodesic distance maps for the Dijkstra and the FMM algorithms are presented. A cosine modulation was used to denote the level set of the expansion wave. When comparing the A* method to the FMM, the latter overcomes the first in path quality as exposed in [58]. Furthermore, if the FM^2 is used for path planning the smoothness and safety parameters are improved. The FM^2 method which is a newer and improved version of the FMM.

For path planning in continuos space with the Dijkstra algorithm, the first step is to generate the geodesic map. Then, the metric incongruence triggers when calculating the geodesic path. In Figure 2.26, a path planning example is shown. The initial point is the bottom left and the goal is the upper right. The green an red lines are paths that can be found following the Dijkstra geodesic descent. It is clear that the dashed black line is the optimal path for this example.

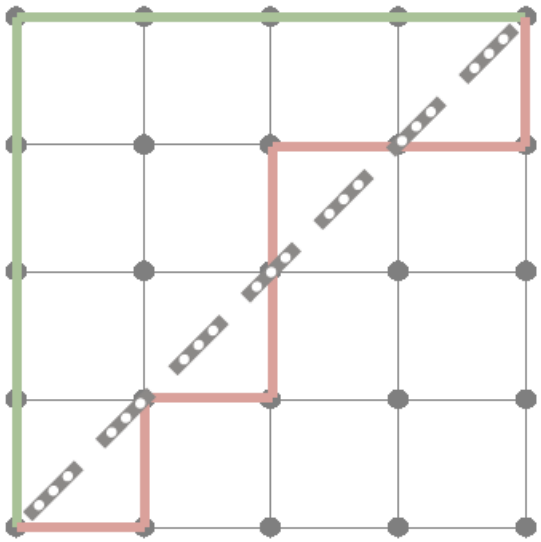


Figure 2.26: Geodesic valid paths for Dijkstra's algorithm. The green and red lines described the Dijkstra's valid paths. The dotted black diagonal is the shortest path that is not found by Dijkstra's method.

The FMM solves the metric incongruence of the Dijkstra algorithm. The computational complexity of this algorithm is $O(n)$ as shown in [59]. Since the FM^2 is

based directly upon FMM, it is also $O(n)$. Both methods, being deterministic, generate consisting paths results each time. The FM^2 offers optimality for safety, softness and path length parameters.

Table 2.1: Well-known path planning algorithms.

Author (Year)	Name	Characteristics
Dijkstra, E. W. (1959) [22]	Dijkstra	Finds the shortest paths in a graph from a source to all vertices.
Hart, P. et al. (1968) [23]	A*	Extends Dijkstra algorithm to improve time performance by using heuristics.
Stentz, A. (1995) [24]	D*	Is a Dynamic A* that plans in real-time and incrementally repairs paths as environment information is updated.
Kavraki, L. E. *et al.* (1996) [25]	PRM	Takes random samples from space and attempts to connect these to other nearby making paths.
LaValle, S. M. (1998) [26]	RRT	Is a tree data structure and algorithm that efficiently explores space. See Section 2.1.1.
Sethian, J. A. (1999) [27]	FMM	An efficient search-based planner that expands a front. See Section 2.5.
Kuffner, J.J. and LaValle, S.M. (2000) [28]	RRT-Connect	Also known as RRT-Bidirectional. Grows two RRT trees, the first from the start and the second from goal. See Section 2.2.2.
Garrido, S. *et al.* (2007) [29]	FM^2	A safe sensor-based planner based on the Fast Marching Method. See Section 2.5.4.
Sucan, I. and Kavraki, L. (2009) [30]	KPIECE	Kinodynamic Motion Planning by Interior-Exterior Cell Exploration
Karaman, S. and Frazzoli, E. (2010) [31]	RRT*	An asymptotically optimal version of RRT.
Karaman, S. and Frazzoli, E. (2011) [32]	PRM*	Based on PRM, it gradually increases the number of connection attempts.
Janson, L. *et al.* (2013) [33]	FMT*	A sampling-based motion planner based on RRT* principle and the Fast Marching Method.

Chapter 3

Path Planning for Robot Navigation

Mobile robots are often provided of wheels connected to servomotors, the mobile base can be arranged so that the robot can move in any direction or in restricted directions as automobiles, these configurations are called holonomic and non-holonomic, respectively. The main purpose of this kind of robots is to navigate through an environment and execute multiple tasks. For this reason, the robot is often equipped with sensors and a robotic arm, enabling to acquire information of the environment in order to avoid obstacles and interact with objects. Therefore, it is necessary to calculate accurate paths in order to safely maneuver in cluttered environments.

The first and second objective of this work are covered in this chapter. These objectives correspond to the path planning and replanning for moving the robot to an specific location. These were presented with a work pipeline for the robot in the introduction chapter. Some path planing approaches for achieving the safe navigation of a mobile robot are presented hereafter, all of these make use of the FM^2 method. The latter, is a path planning method known for generating safe and reliable paths with good clearance. A correct implementation of the FM^2 method enables its use in real-time applications with few dimensions. It is the case of path planning for navigation where with two dimensions the execution time can be much faster than any sampling-based method, specially in complex environments where high precision is needed. These reasons justify the use of the FM^2 over other more popular methods as the sampling-based that generate paths with random safety and may not converge to a solution even when ones exists.

3.1 Smooth Path Replanning using FM^2

The problem of path planning can be stated as finding a sequence of state transitions through a map from some initial state to a goal state, or determining that no such sequence exists. If during the traverse of the path, one or more transitions in the map are discovered to be incorrect, the remaining portion of the path may need to be replanned to preserve feasibility and optimality. It is clear that there is a necessity of replanning efficiently as daily environments where mobile robots should work are highly dynamic.

An important application for this problem, and the one that will serve as the central example throughout this section, is the task of path planning for a mobile robot equipped with a sensor, operating in a changing, unknown or partially-known environment. The robot begins with an initial estimate of the path, but since the environment is only partially-known or changing, some segments of the path are likely to be unfeasible. As the robot acquires sensor data, it can update its map and replan the optimal path from its current state to the goal. It is important that the replanning works fast, since during this time the robot must either stop or continue to move along a suboptimal path. Many replanning approaches for a mobile

robot with a sensor have been proposed, but in every case the underlying planning methods generate lower quality paths when compare to FM^2 in terms of safety and smoothness. In the method proposed by Stentz [24], the A* is used to calculate paths and as discussed in the state of the art section, there has been studies that proved the FMM to be superior [58]. Because the FM^2 is based on the FMM, it inherits all of the good properties and also improves the quality of the paths in smoothness, clearance and thereby safety. An RRT based replanner is proposed by Fergurson in [60], again as it has been and will be discussed later through this document, the sampling based methods generate random paths with unpredictable safety properties. In the past, planning was considered optimal when the minimum distance path was found [24]. Nowadays, the safety is a crucial factor as a mobile robot must operate within human environments where the integrity of people as well as that of the often very expensive robot must be guaranteed. For these reasons, the FM^2 method conciliates all of the requirements that a modern path planner must have.

Algorithm 14 Anytime Fast Marching Square

```
1: map ← load(free_map)
2: vmap ← velocities_map(map)
3: full_path ← fast_marching(map, vmap, start, goal)
4: nodes ← path_nodes(full_path)
5: while (global goal ≠ true) do
6:    map ← laser_scan()
7:    vmap ← velocities_map(map)
8:    update_next_subgoal(subgoal)
9:    if (obstacle_free(full_path, nodes.next()) then
10:      move_forward(full_path)
11:   else
12:      full_path ← fast_marching_next_node(...)
13:      move_forward(full_path)
14:   end if
15: end while
```

A path replanning algorithm in unstructured environments using the FM^2 method is proposed in this section. The proposed strategy enables the robot to safely navigate and by using subgoals reduces unnecessary calculations when replanning, which leads the mobile robot to reach its goal in less time. Also, the robot's onboard computer is released more frequently in such a way that any other important processes are able to execute.

The proposed method works in two steps. First, a smooth and safe global path is generated. This path is divided in multiple subpaths separated by equidistant nodes

(defined by topological or metric constraints). After that, the obstacles information is added and a different path is calculated only when the original path is unreachable. The recalculation of the path is made up to the next node, which reduces the computational time. Different tests have been carried out in an indoor environment. The most important advantage with respect to similar approaches is that sub-paths are always efficiently generated in one execution cycle in terms of smoothness and safeness. Besides, the computational cost is low enough to use the algorithm in real-time.

3.1.1 Anytime Fast Marching Square Method

We have developed an approach using the FM^2 algorithm that solves the path replanning problem in a robust and efficient way generating smooth and safe paths.

In order to successfully plan a trajectory, the main idea is that the robot performs updates of the environment using a sensor laser scanner. New obstacles are added to the obstacles map with every scan. The velocities map is calculated and the FM^2 is performed over this map. The FM^2 path replanning method is depicted in Algorithm 14.

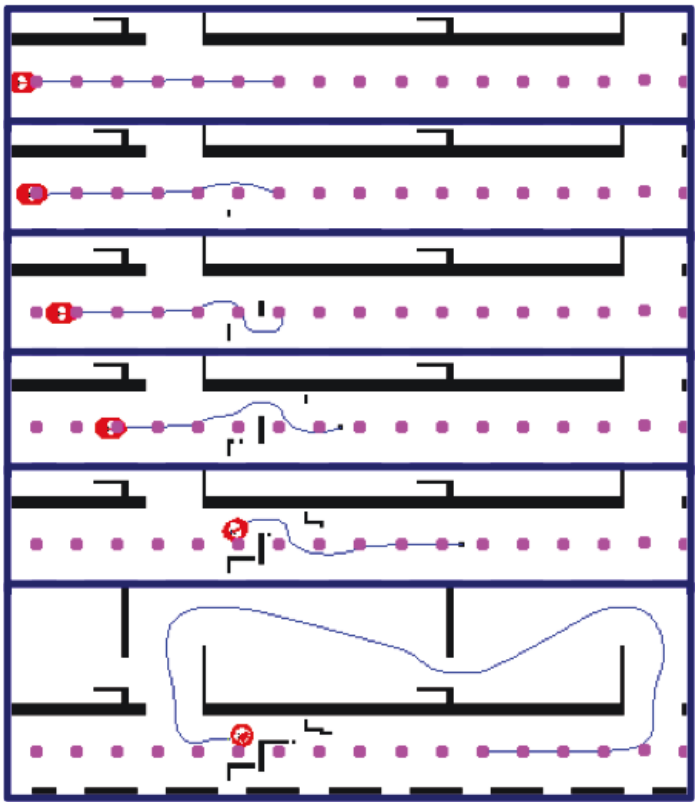


Figure 3.1: Sequence for path replanning to subgoals.

First of all, it is necessary to comment some details about the environment and the robot's characteristics. The environment has been modeled geometrically as an occupancy grid map in two dimensions and the robot's pose (the robot's pose is defined as the robot's position and orientation: x, y, and yaw) is represented with the cartesian coordinates and the horizontal orientation.

Algorithm 14 starts in line 1 by loading an environment map with the most unchanging obstacles, such as walls and fixed objects like those screwed to the floor. This map is usually equivalent to the building plans. The previous loaded map is used in line 2 to generate its corresponding viscosity map. In line 3, the method uses the previously generated viscosity map as the velocities map to generate the initial global trajectory. This initial global path is segmented in equal portions by means of Geometric or topologic specifications in line 4. The geometric strategy receives the initial global path, which is a sequence of points in the map, and defines subgoals nodes by selecting one point every predefined constant number of steps. The topology-based strategy, on the other hand, uses reference points in the environment map, where the subgoals are set to the closest initial global path points to these reference points.

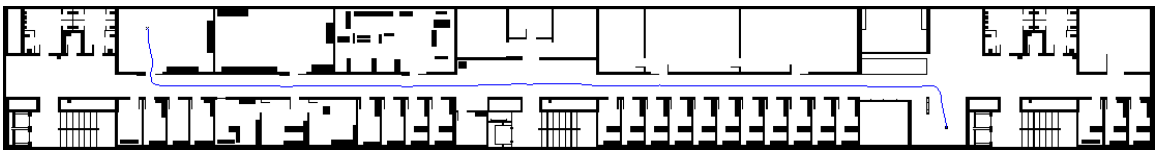


Figure 3.2: Initial Path calculated for the first proposed experiment.

In line 5, the robot is located at the starting point in the environment and the iteration loop starts. The obstacles and velocities maps are updated in line 7 and 8. When advancing through the global path, it may occur that the next node is not the closest node, which would generate an inefficient path unless this case is detected and corrected. This event can be figured out with the example illustrated in Figure 3.1. In each frame, the magenta dots are samples of the initial path and the blue lines correspond to the local plans to next subgoals. In the first image it can be appreciated how the local plan coincides exactly with the global initial path. Then, divergence between paths increases as new obstacles are discovered with the laser sensors. In the last frame, the path is recalculated as consequence of an obstructed corridor. The entire initial path for this example is presented in Figure 3.2. The new trajectory leads the robot through adjacent rooms that are connected by an open door.

In Figure 3.3, a path planning sequence that resumes the one in Figure 3.1 is presented. It illustrates the local planning when the robot passes through rooms and goes back to the main corridor. In the first frame, the local plan to next subgoal (blue path) would suppose to go south in the map and then to return east over its

track. Thereby, in order to calculate a logical shorter path, the local panning goal is updated to the nearest subgoal with the function *update_next_node*(). This is done in line 8, the euclidean distance from the location of the robot, p, to the current and to the next subgoal, g_k and g_{k+1}, are calculated

$$Q = \min_{k,k+1} \sqrt{(p_1 - g_k x)^2 + (p_2 - g_k y)^2} \tag{3.1}$$

the node with the minimum of the distances is taken as the new subgoal Q. In the sequence of Figure 3.3, it can be appreciated from the third frame up to the last one how the path is modified by updating the replanning objective to subgoals nearer the final goal.

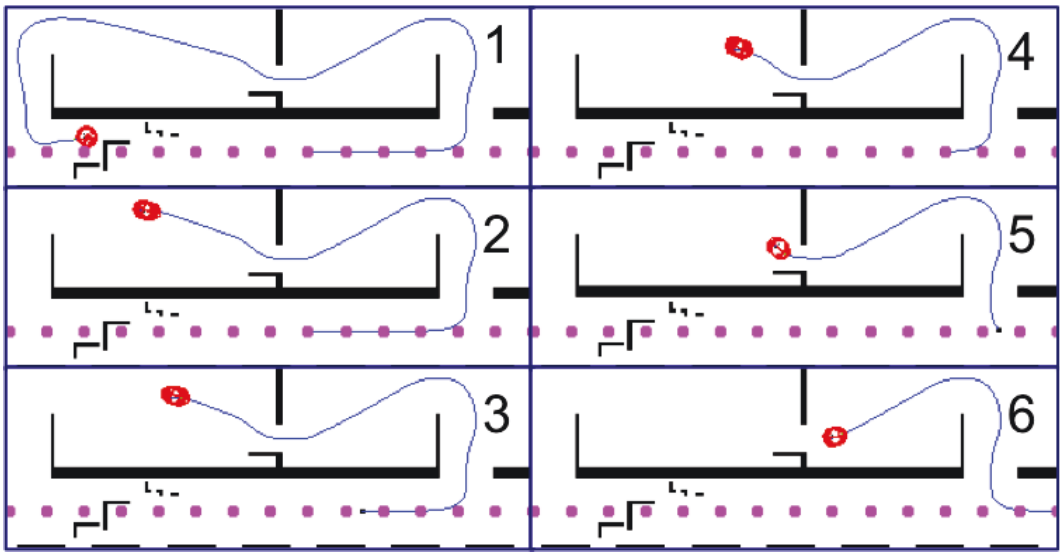


Figure 3.3: Sequence for path replanning to subgoals.

If the robot detects that the map has changed and that it is not possible to execute the initial global path, then the path is recalculated up to the next subgoal node and the global path is updated only in the affected segment. The robot moves forward through the global path to the next free obstacles node (line 9 to line 14). The robot displacement is computed considering its motion error. Thereby the executed trajectory could differ from the first calculated, even in the case when the obstacles map is the same as the pre-loaded map. This is taken into account in the experiments made in the corresponding section. In this way, our method is capable of calculating a smooth path while avoiding the obstacles at the same time.

3.2 Anytime Triangular Fast Marching Square Method

The anytime approaches in robotics are based in two fundamental observations or principles. The first is that it makes no sense to invest a lot of time in calculating a path to then be discarded because the environment has changed. The second and most important is that mobile robots move parsimoniously, which gives time to improve initial paths.When the mobile robot is in an unstructured environment, it is necessary to include real-time information about its surroundings to obtain safer paths. This information is acquired by the robot by using a laser range finder. One important characteristic that this type of algorithms must satisfy is that the path must be generated in real-time. The anytime algorithms have been successfully applied in robotics races with real cars and other robotic tasks in order to quickly generate a first path and then improve it in an incremental way when time is available.

An anytime motion planner on triangular meshes is presented in this section, the original approach was presented by Gómez *et al.* in [61]. Here, an anytime version and many variations of the method are presented and deeper analysis and experiments are carried out. Also, while developing this work, some flaws on the general formulation for evaluating path planning methods were encountered. Thereby, a discussion is conducted and some improvements are proposed in Section 3.4.

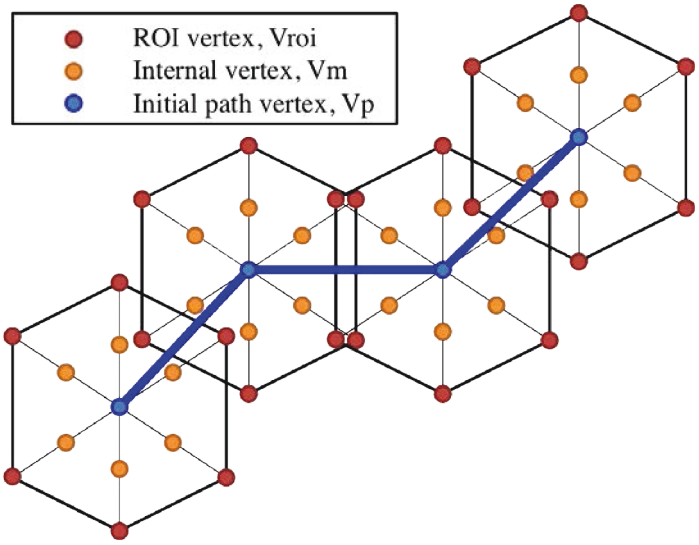
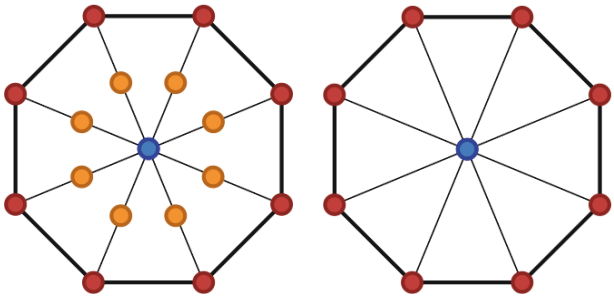


Figure 3.4: Example of mesh generation using hexagons.

A correct implementation of the FM^2 method produces excellent computational performance that enables its use in realtime applications. This assertion is true under condition of few dimensions, for instance two and three degrees of freedom. However when the dimensions are increased, the search space grows exponentially.

For example, when a degree of freedom with 1000 configuration states is increased to two the configurations are raised to 10^6 and to 10^{18} when increased to six degrees of freedom. The latter one is a very frequent scenario in robotic manipulation that increases the computational time to unfeasible rates. For this reason, the use of a triangular mesh grid is proposed. This approach would reduce the size of the search space and therefore the computational time.



(a) External and internal vertexes. (b) Only external vertexes.

Figure 3.5: Octagons used for mesh generation.

Even though the method presented in this section was intended for robotic manipulation, it was first implemented for 2D path planning for simplicity reasons. Latter, due to technical difficulties with the implementation of triangular mesh grids in higher dimensional spaces, a different approach for manipulation path planning was proposed and it is presented in Section 4.2. This section is structured as follows. First, the mesh generation process is detailed in Section 3.2.1. After that, the whole algorithm is described in Section 3.2.2. The objective of the ATFM^2 algorithm is to improve the scalability of the FM^2 planner, which is a very good method for generating smooth reliable optimal paths. Even though the best solution in 2D path planning would be to directly apply the FM^2 method, the approach presented here is addressed in 2D to reduce the demonstrations complexity and get a better understanding of the algorithm. The method real target is high-dimensional problems as path planning for six or higher Degrees of Freedom (DOF) robotic arms.

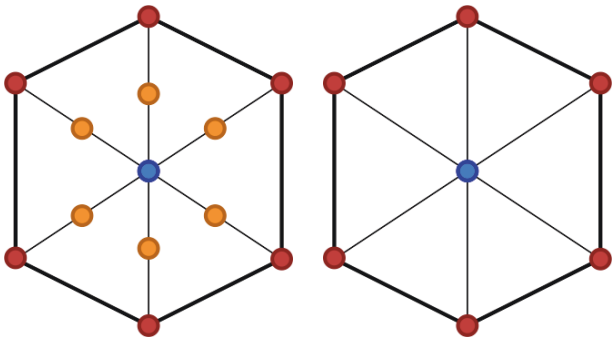
In the first step (Section 3.2.1), the RRT method is used to obtain an initial path. Although the RRT planning results tend to be suboptimal, the method is highly scalable in dimensions. In the following steps, the initial RRT path is used as a base to reduce the exploration area for the triangular FM^2 algorithm. The steps of this technique are explained in more detail below.

The basic idea behind an anytime motion planner is that the robot's path could be iteratively improved. This is a useful concept when there is time enough to compute a new path because the robot's movement is not too fast. An illustrative example of

the advantages of using an anytime planner approach is presented in Figure 3.8.

3.2.1 Mesh generation

One of the algorithm main steps is to create a structure formed by triangles around the outputted pathway nodes generated by the base planner, which in this case is the RRT method. The structure with triangles, called mesh or more specifically triangle mesh, is created over the original pathway to define what it is known as the Region of Interest (ROI). Regular polygons are arranged on the original pathway, one polygon per node and every polygon centered at each node. The distance between adjacent pathway points has to be less than two times the polygons apothem, where the apothem is defined as the distance from the center of a polygon to the midpoint of one of its sides. This condition must be satisfied in order to guarantee the connection among contiguous polygons. Since sampling-based algorithms as RRT already incorporate a step length, which is the maximum distance between nodes, this condition is easily achieved by defining an apothem greater than half the step length. Polygons of equal size are then placed in every node as shown in Figure 3.4. The ROI is computed as the area confined by the outline of the intersection between polygons. This stage will reduce the FM^2 method exploration area and thus the computational time.



(a) External and internal vertices. (b) Only External vertices.

Figure 3.6: Hexagons used for mesh generation.

The approach proposed for improving paths in [61] uses octagons as regular polygons, adding eight external and eight internal vertices as shown in Figure 3.5(a). These vertices are used to generate the triangle mesh. The external vertices coincide with those of the octagon, and the internal ones are located at the middle point between the center of the polygon and each external vertex. The octagons edges and the intersection with other polygons serve as the outline to define the ROI for the

FM^2 method and then determine the gradient descent path. The algorithm achieves to improve an initial pathway as benchmarking results will establish.

Some options that may speed-up the process are proposed for the anytime planner. One possibility is to consider different types of polygons in this stage. It is advisable to start with faster functions and then incrementally improve the path as far as possible while the robot is moving. The list of polygons that have been used here is:

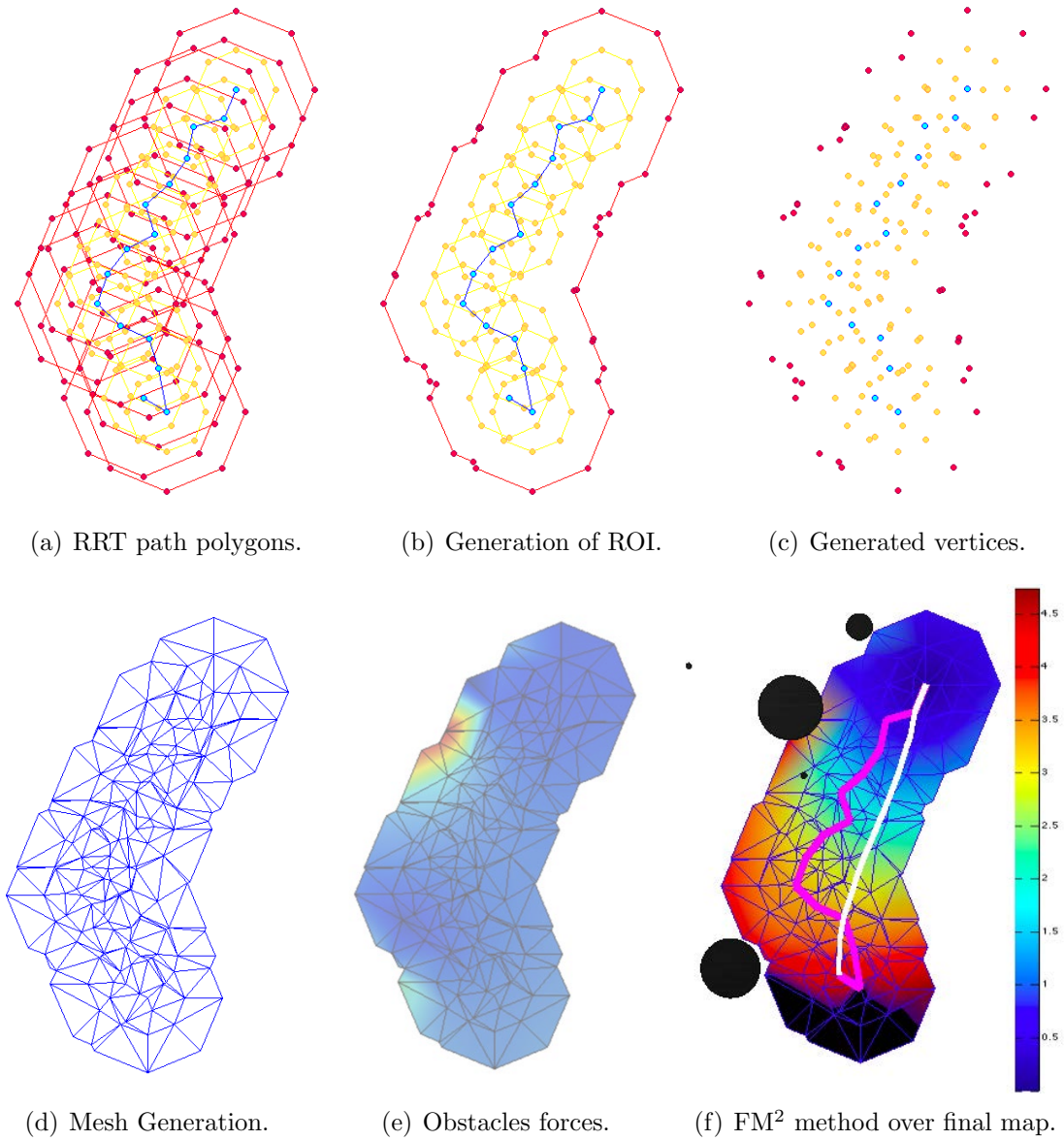
- **Eight external and internal vertices**, approach used in the original version. Different options could be combined with this one, as will be described later in this document. This polygon is shown in Figure 3.5(a).
- **Eight external vertices**, shown in Figure 3.5(b).
- **Six external and internal vertices**, shown in Figure 3.6(a).
- **Six external vertices**, shown in Figure 3.6(b).

Algorithm 15 Anytime Triangular FM^2 Method

```
1: rrt_path ← generate_rrt_path(Init, Goal)
2: internal_vertices ← generate_polygons(rrt_path)
3: external_vertices ← generate_polygons(rrt_path)
4: ROI ← polygons_union(external_vertices)
5: mesh_model ← triangle_represent(rrt_path, internal_vertices, ROI)
6: mesh_map ← combine_mesh_and_obstacles(mesh_model, obstacles)
7: velocieties_map ← fast_marching(mesh_model)
8: fmm_path ← calculate_geodesic(velocieties_map)
9: while (global goal ≠ true) do
10:    fmm_path ← validate_path(fmm_path)
11:    internal_vertices ← generate_polygons(fmm_path)
12:    external_vertices ← generate_polygons(fmm_path)
13:    ROI ← polygons_union(external_vertices)
14:    mesh_model ← triangle_representation(rrt_path, internal_vertices, ROI)
15:    mesh_map ← combine_mesh_and_obstacles(mesh_model, obstacles)
16:    velocieties_map ← fast_marching(mesh_model)
17:    fmm_path ← calculate_geodesic(velocity_potential_map)
18: end while
```

3.2.2 Anytime triangular motion planning algorithm

The ATFM^2 path planning method is depicted in Algorithm 15. The Figure 3.7 shows some stages of this procedure. In the initialization stage, the output of the RRT method is used to generate the initial path. After that, this path is continuously updated (and improved) by the anytime planner.



(a) RRT path polygons. (b) Generation of ROI. (c) Generated vertices.

(d) Mesh Generation. (e) Obstacles forces. (f) FM^2 method over final map.

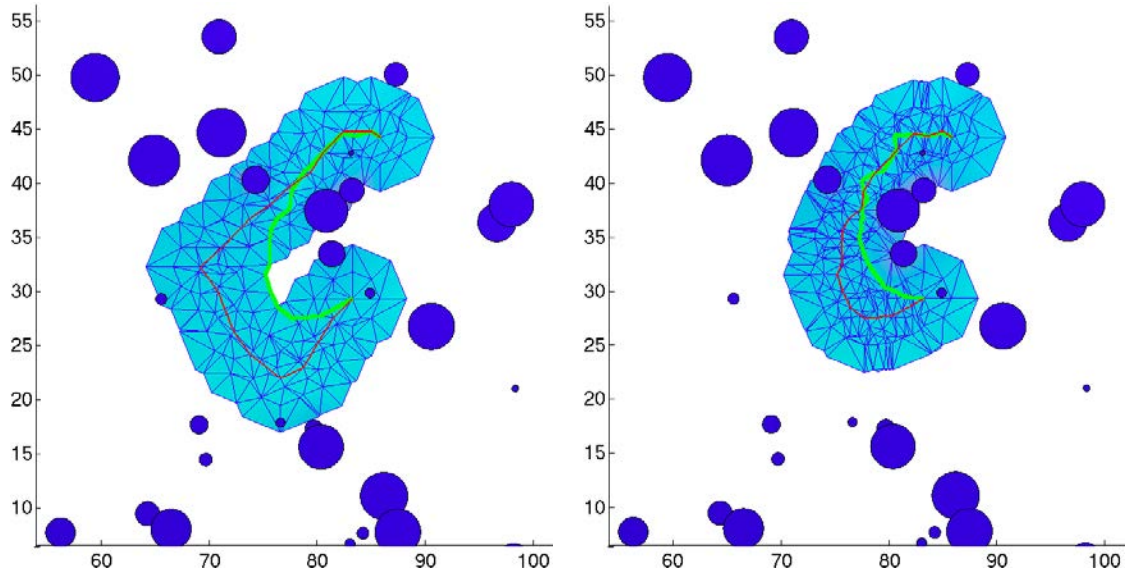
Figure 3.7: ATFM2 algorithm steps.

In line 1, the sampling-based method generates the base path. As said before, the sampling-based method is the RRT in this study, but any planner could also be used. The vertices are generated in lines 2 and 3. These points belong to the external and internal polygon vertices. In Figure 3.7(a), the RRT path points are depicted in blue, the internal vertices in yellow, and the external ones in red. The area of the external polygons is combined to compute the ROI in line 4. The red silhouette in Figure 3.7(b) is the union result which corresponds to the ROI. In line 5, the Constrained Delaunay Triangulation (CDT) method is applied over the generated vertices and inside the ROI in order to obtain a triangle mesh. Figure 3.7(c) shows the points that the triangulation function receives, which is a set that is composed of the nodes of the RRT path, the internal vertices, and the external ones if they belong to the border of the ROI. The triangle mesh presented in Figure 3.7(d) is the result of applying the CDT function. In line 6, the environment map with obstacles is combined with the mesh model obtained in the previous phase. To finish the initialization stage, the FM^2 method is applied inside the ROI by generating a velocity potential map in line 7, and finding the geodesic path that improves the original path in line 8. The velocity potential map is displayed in Figure 3.7(e).

The path obtained in line 8 is shown in Figure 3.7(f), where the black circles represent the environment obstacles, the colors inside the ROI depict the FMM wave expansion from the goal location to the start location, the black region inside the ROI depicts the area that has not been reached by the expansive wave, the magenta line is the RRT path, and the white line is the FM^2 path.

The anytime planning starts inside the while loop in line 9, when the computed path is taken as an input that will be iteratively improved. Because the distance between consecutive points in the FM^2 paths is not uniform but narrow in curves and wider in straighter parts, the function *validate_path* (line 10) verifies that the path fulfills the polygons connectivity requirement. Whenever the distance between path nodes is greater than two times the polygons apothem, additional nodes are added to the path by interpolating between nodes as many points as necessary. As before, the internal and external polygons vertices are generated in lines 11 and 12, the external polygons are combined to form the ROI, and the CDT method is applied in the next line. Hereafter, the FM^2 method is applied inside the ROI and, in the next line; the path that follows the geodesic is computed. The loop iteratively improves the path until the global goal is achieved. This stopping condition is accomplished when the path improvement between iterations is less than a significant proportion according to the benchmarking parameters.

All the proposed polygons variations were tested several times and the experimental results are presented in Section 5.2. The best results are obtained for the six vertices variations in terms of computational time and path length, smoothness and clearance; more details are given in the experimental section. Furthermore, as



(a) First iteration. Red: RRT initial path. Green: FM^2 method computed path.

(b) Second iteration. Red: FM^2 method initial path. Green: FM^2 method computed path.

Figure 3.8: Improvement of the original path in the anytime motion planner.

discussed in the following paragraphs, the anytime approach produces better results than the original method.

Anytime motion planning advantages:

As can be seen in Figure 3.8(a), the path is improved in length. Also, the benchmarks will show improvement in smoothness and safety with respect to the initially generated RRT path, and in the second iteration with respect to the path generated in the first iteration with the FM^2 triangular approach. It should be noticed that the FM^2 path goes through the outline of the light blue area around the middle of the path, meaning that in this case, the length improvement is limited by the ROI. In a second iteration, the algorithm receives the FM^2 path obtained in the first iteration and, as can be seen in Figure 3.8(b), the ROI is rebuilt with the FM^2 pathway points, letting the algorithm to further improve the path length. It can also be noticed that the second iteration creates a more dense mesh because the FM^2 method produces more points than the RRT. This figure intuitively exemplifies the fact that by confining the FM^2 action area over an non optimal method such as the RRT, many optimized paths are left out of reach. For this reason, the use of an anytime algorithm is suitable to further improve the motion planning method.

3.3 Fast Marching Square applied to Nonholonomic car-like robots

A very relevant type of robot are the nonholonomic ones, which cannot move freely in any desired direction. Mathematically this means that their movements have to accomplish a set of constraints which are not imposed by the environment. A typical case is the car-like robots or otherwise the robots based in commercial cars. In this section, the FM^2-NH approaches are described.

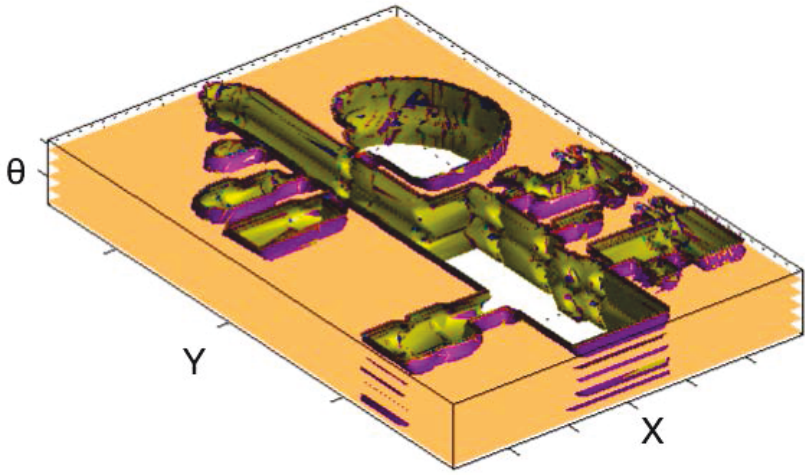


Figure 3.9: Three dimensional C-space of a car-like robot, where the third dimension is the orientation.

The basic approach was published by Garrido *et al.* in [62]. Here, the original approach is divided in two methods and studied in greater depth with further experiments. Further contributions are made by presenting pseudo-code algorithms for both methods and benchmarking results. Also the formulation for calculating the control actions of the car-like robot is presented. We discuss the details on how to apply the methods to car-like robots, and how safety and physics considerations are taken into account to accomplish the robot path planning problem.

3.3.1 Nonholonomic Fast Marching Square in C-space

The first approach models the environment, in which the car-like robot has to move, as a C-space. In this space, the two first dimensions consist on the position of the robot and the third one is given by the orientation of the vehicle. If we compute a

trajectory along this C-Space, it is possible to guarantee the absence of collisions. In order to achieve this, we need to take into consideration two aspects: first, the possible orientations of the vehicle at every position in the map; second, for each orientation we need to take into account the dimensions of the vehicle in order to know when collisions occur.

Algorithm 16 Algorithm of the C-space FM^2-NH

Input: A grid map G, starting point x_{init}, goal point x_{goal}, dimensions of the car d, an obstacle grid-point λ

Output: The calculated path ρ and the control actions U

{Initialization}

1: $c\_space\_map \leftarrow Create\_c\_space\_map(G, d)$

{FM2-NH 1st step: velocity potential map}

2: $vp\_map \leftarrow FMM(c\_space\_map, \lambda)$

{FM2-NH 2st step}

3: $fm2\_map \leftarrow FMM(vp\_map, x_{init}, x_{goal})$

{Geodesic path and control actions}

4: $\rho \leftarrow Geodesic\_path(fm2\_map, x_{init}, x_{goal})$

5: $U \leftarrow Control\_actions(fm2\_map, path)$

In order to consider the aforementioned aspects, some changes are introduced in the computation of the map on which the FM^2-NH algorithm is applied. The necessary steps are:

1. *Create configuration space map.* In the first step, the poses that are not feasible, depending of the orientation of the robot, are eliminated. Since in the path computation step, the robot is intrinsically considered as a one cell body, we need to enlarge the obstacles to assure non-collision paths. This enlargement depends on the shape of the robot. The obstacles growth is performed by adding a rectangular shape whose size is half the size of the car in both X and Y dimensions. Besides, the rectangular shape to be added is turned with respect to the orientation for which the configuration space is being calculated. Thus, the safe navigation of the robot is ensured, and the expansion of the wave is shrank due to the reduction of the free space. Furthermore, in the case of narrow entrances that are inaccessible to the robot, the dilation of walls closes those entrances, diminishes the wave expansion area, and in consequence reduces the computation time. In this way, the remaining poses define the three dimensional free configuration space. This step is computed n times, being n the amount of different orientations of the robot for which the C-space is created. The bigger n is chosen, the smoother trajectory will be computed, since the step

in between orientations will be smaller. At the same time, the bigger is n, the more computation time is needed for this step. An example of the output of this step can be seen in figure 3.9, in which the third dimension is the orientation of the robot and $n = 20$. The orientations are repeated above and below the calculated values in order to permit manoeuvres.

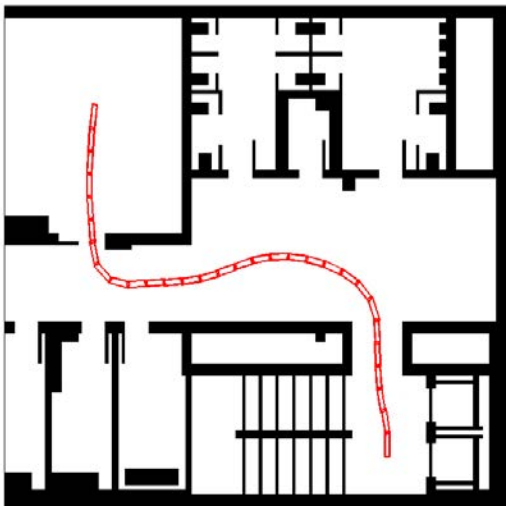


Figure 3.10: C-space FM^2-NH applied to the car-like robot in the university environment.

2. *FM^2-NH 1st step.* A first run of the FMM is carried out with the C-space map resulted from previous step. In this particular case the sources are all the obstacles points present in the C-space map. The output will be another grid map with the arriving values T, as indicated in Algorithm 11. This map is better known as velocity potential map, and as its name suggests, it establishes a maximum speed for every point in the map that should be taken into account when moving the real robot.

   As an additional step, a safety distance M from which the obstacles are not taken into account can be established. This can be easily done since each cell value in the velocity potential map gives the distance to the nearest obstacle in time, which can be employed as a clearance metric because it is proportional to the geometric distance [57]. Therefore, the velocity potential map can be saturated, and all the grid point potential values greater than the predefined safety distance M, are set to M (which also can be interpreted as the maximum velocity allowed at that point). This enables the planer to maintain a prudential distance from obstacles, while at the same time, the path length is shortened. The reasoning here, is that maintaining a clearance greater than a predefined

safety distance, would only increase the path length unnecessarily.

3. *FM^2-NH 2nd step.* The last step of FM^2-NH is to generate an additional FMM wavefront over the velocity potential map. The obtained surface is used to obtain an optimal minimal path in time expansion of the wave, by following the minimum gradient direction of the wavefront potential from the target to the initial point. Because we have modeled the environment taking into account the orientations of the vehicle, the obtained trajectory corresponds to the geodesic path of the surface along the car-like robot orientations.

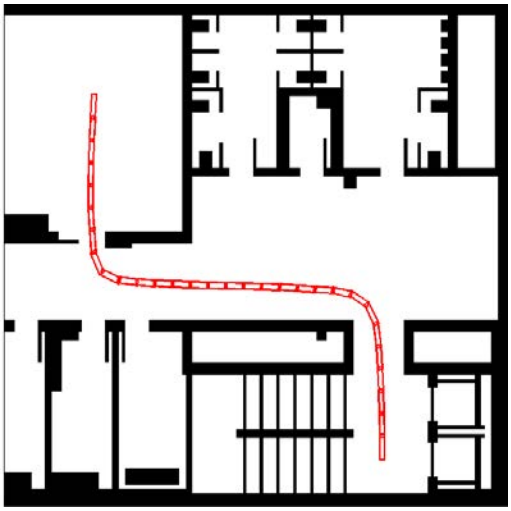


Figure 3.11: C-space FM^2-NH with saturated velocity potential map applied to the car-like robot in the university environment.

In Algorithm 16, the C-space FM^2-NH approach is presented in a pseudocode language.

A result of a path obtained using the aforementioned steps is shown in Figure 3.10. The corresponding C-space is represented in Figure 3.9. The top and the bottom configuration values are connected because the angle wraps around 2π radians. It can be seen that the resulting path respects the kinematics constraints imposed by the vehicle, while trying to move as far as possible from obstacles. Since the C-space is built iteratively placing the vehicle in every position and with many different possible orientations, it is a slow task. However, it can be precomputed off-line and only has to be done once per map.

Most of the times, it is not necessary to get as far as possible of obstacles like in Voronoi diagrams. In Figure 3.11, the same rooms as in Figure 3.10 were set as initial and goal locations. For this example, the velocity potential map was saturated with a

sufficiently safe distance from obstacles. Therefore, the obtained path in Figure 3.11 is shorter than the one in Figure 3.10, and also maintains a distance to obstacles that is safe enough.

In Figure 3.12, an example of the FM^2-NH in C-Space is presented. The environment map is a representation of an Intel Research Center located in Seattle, the map of this robotics laboratory is a common used benchmark dataset. For this example, the velocity potential map was saturated in the execution of the method.

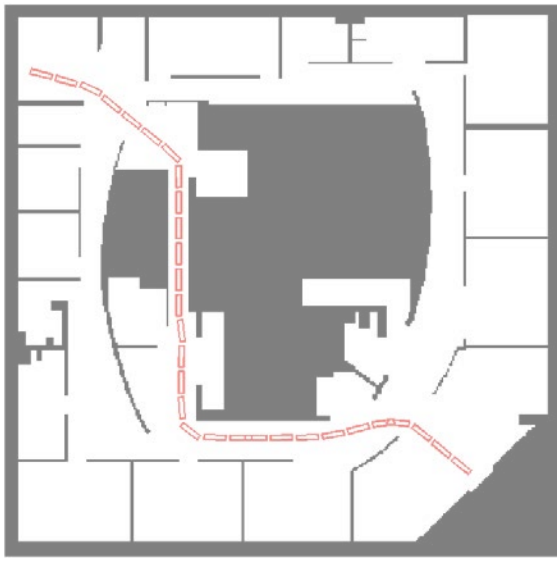


Figure 3.12: Execution of a path obtained with the C-space FM^2-NH method. The environment map is an intel lab located in Seattle, and the robot is a car-like robot.

3.3.2 Control-based Nonholonomic Fast Marching Square

A very relevant feature that has not been sufficiently highlighted in Section 2.5, is that by using the gradient over the second potential, it is possible to calculate a vector field whose field lines are the paths that go from each point to the target, moving away from obstacles and walls in the map environment.

The velocity potential map is then used by the FM^2 to create a second potential $T(\mathbf{x})$. This new potential represents the arrival time of the wavefront, and in this way the method gives the arrival time as the third axis. The wave is originated from the goal point and continues to propagate until reaching the starting point, *i.e.*, the current position of the robot.

In the Control-based Nonholonomic Fast Marching Square the FM^2 second potential is used to calculate the gradient values OX and OY associated to each grid

point. The result of this operation is the OXY vector field of the motion plan. The directions of the gradient vectors point away from obstacles. They follow paths across the different environment points to converge to the goal point. The magnitude of the vectors can be used to determined the velocity of the car-like robot, and so, the gradient vectors can be used to move the robot forward. Figures 3.13 and 3.14 show a car-like robot drawn over the vector field of the gradient vectors.

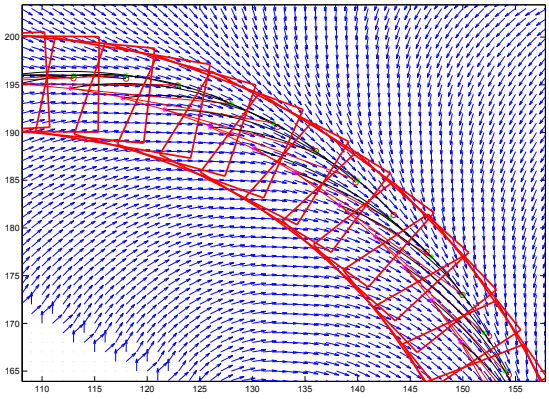


Figure 3.13: Movement of the vehicle on the vector field.

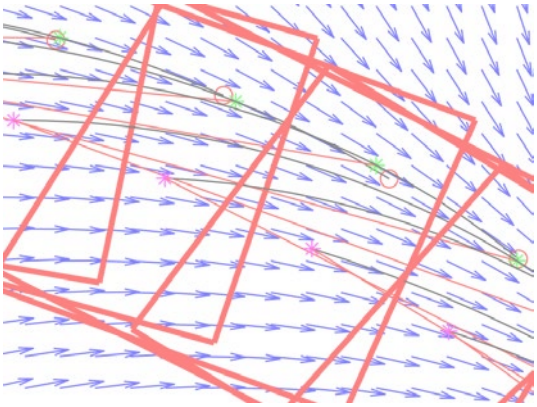


Figure 3.14: Detailed representation of the vector field.

Car-like robots have a limited steering angle causing them to move along paths of bounded curvature. In Figure 3.15 a car-like robot is presented, where R is the center of the rear axis and is represented by its (x, y) coordinates. The angle θ is the car orientation respect to the OX axis. For the specification of the motion problem, it is a necessary to consider the following nonholonomic constraint

$$\dot{y}\ cos\theta - \dot{x}\ sin\theta = 0$$

and the car-like movement can be modeled, with unit length between the front and rear axes of the wheels, as

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} v\ \cos\phi\ \cos\theta \\ v\ \cos\phi\ \sin\theta \\ v\ \sin\phi \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} v_1 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} v_2 \tag{3.2}$$

where the front wheels orientation is expressed by ϕ, the car velocity by $\dot{v}$, and the two control inputs are v_1, v_2 : the acceleration of the robot and the front wheels angular velocity.

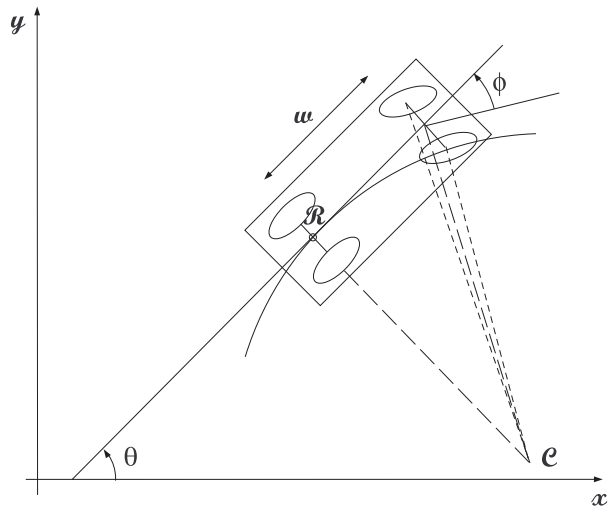


Figure 3.15: A car-like robot.

This model can be expressed as a constraint on the curvature radius of the path. This constraint can be directly included in the algorithm using the vector field, in form of limits during the path calculation. An interesting remark is that the variables in equation 3.2 are given by the vector field, except for the control inputs v_1, v_2. This means that these control inputs can be easily deduced and this way the method not only gives the trajectory but also the control inputs to follow that trajectory. In Figure 3.16, an example of the ϕ variable over a trajectory is presented. It can be appreciated how the angle in radians fluctuates between 0 and 2π over time. The dynamic of the car-like robot, as in the majority of planning methods deal with

driftless control affine systems, which have the form

$$\dot{x} = \sum_{i=1}^{m} h_i(x) u_i \tag{3.3}$$

where the state $\dot{x} \in X$ consists of the configuration variables and their derivatives, h_i is a vector field on the state space of X, and $u = [u_1 ... u_m]^T \in U$ is the vector of the control variables. If the configuration is treated as the state, then $x = q$. This equation is used to find the variable u necessary to make the control of the robot. Since the method provides the positions, velocities and the vector field, the control can be calculated. The Equation 3.3 can be solved for the control ui. In Figure 3.17, the resulted control signal for a motion plan example is shown.

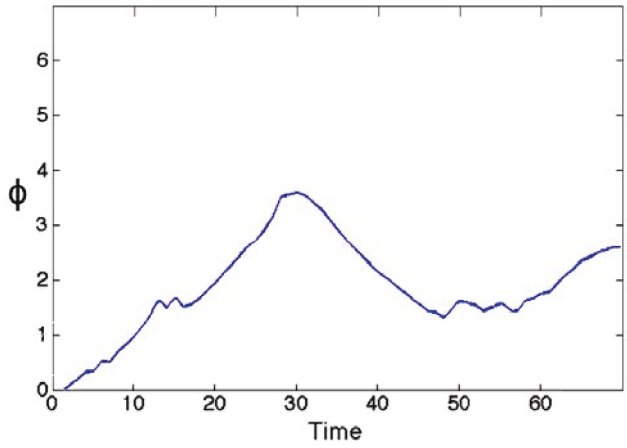


Figure 3.16: Front wheels orientation ϕ vs. time for an example of the car-like robot path planning.

In order to compute the complete path from the start to the goal position and orientation, the path is incrementally generated, beginning from the initial pose, and according to the following order:

- The front wheels are aligned with the vector field in the midpoint of the front axis.
- The perpendicular lines to the front and rear wheels are considered and their intersection is taken as center of the step movement.
- With the previously calculated center C, the vehicle is moved a circumference arc of length proportional to the vector modulus correspondent to that point.

Algorithm 17 Algorithm of the Control-based FM^2-NH

Input: A grid map G, starting point x_{init}, goal point x_{goal}, dimensions of the car d, an obstacle grid-point λ

Output: The calculated path ρ and the control actions U

{FM2-NH 1st step: velocity potential map}

1: $vp\_map \leftarrow FMM(c\_space\_map, \lambda)$

{FM2-NH 2st step}

2: $fm2\_map \leftarrow FMM((vp\_map, x_{init}, x_{goal}))$

{Geodesic path}

3: $path \leftarrow Geodesic\_path((fm2\_map, x_{init}, x_{goal}))$

{Initialiazation}

4: $position \leftarrow x_{init}$

5: $\rho.add(x_{init})$

{Loop}

6: **while** $(global\_goal \neq true)$ **do**

7: $\phi \leftarrow Caculate\_action(G, path, position, d)$

8: $position \leftarrow Advance(\phi)$

9: $\rho.add(position)$

10: $U \leftarrow Caculate\_control(fm2\_map, position)$

11: **end while**

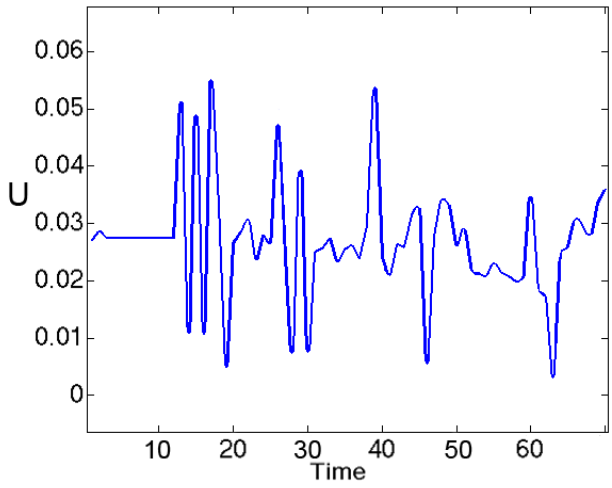


Figure 3.17: Control signal u vs. time for an example of the car-like robot path planning.

The previous process is repeated from the new point until the destination point is reached. The final point and orientation is always reached because the funnel potential end at this point and orientation. Finally, the control inputs for the robot can be computed for the whole trajectory. Figure 3.18 shows the result of applying the algorithm for a parking manoeuvre.

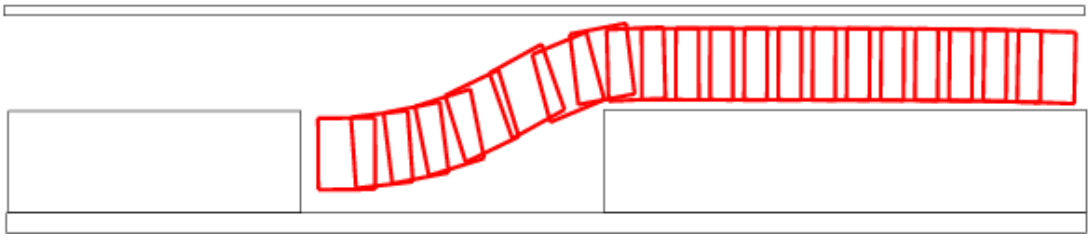


Figure 3.18: Parking manoeuvre using Control-based FM^2-NH.

In Algorithm 17, the Control-based FM2-NH approach is presented in a pseudocode language.

The result of four examples generated with the control-based FM^2-NH can be observed in Figure 5.23. The location and orientation for the start and goal points were randomly chosen. The presented map represents a cluttered environment frequently used to test RRT algorithms. The results of the experiments are discussed in Section 5.3. The presented nonholonomic methods can be mixed with the anytime approach

presented in Section 3.1.

3.4 Performance Parameters and Benchmarking Improvements

This section presents the different parameters used for the evaluation of the quality of the paths generated by the motion planning algorithms. The computational complexity of algorithms is a common used metric in computer science, but in this case it is not enough since the convergence performance of path planning algorithms varies. In sampling-based methods, the convergence-time changes randomly and it is influenced by the conglomeration of obstacles and the complexity of the task. For the search-based methods, the convergence-time is increased according to the grow of the exploration space size and the dimensionality of the problem. As flaws have been found in particular cases when using the traditional formulas to calculate the benchmarking parameters [43], some improvements are proposed in the next paragraphs. The considered metrics are listed below:

- *Computational times:* The execution time is computed for each stage of the planning method, whether the algorithm is calculating a path or optimizing in some way an already generated path. The development of the algorithm under similar architectures is taken for granted, since this metric depends greatly on the programming language and the computational platform used for its execution.

- *Path length:* This parameter is the sum of the distances from one waypoint to the next one in the planner state space. For instance, the results presented here correspond to a 2D space where the sum of the Euclidean distances between consecutive waypoints is an appropriate metric. Regarding benchmarking, the path length is usually connected to the method performance since a shorter path could represent a gain in energy and time. Nevertheless, this is not always necessarily true because the clearance and hence the safety can be notoriously affected when the path is the shortest possible. Besides, there are other factors that could hinder the path such as the inclination of the terrain, the presence of wind in certain zones, the roughness of the surface, water currents in aquatic appliances, and other factors that could produce difficulties when moving. All these factors could make a shorter path inefficient in energy and time. Fortunately, in the FM^2 method, this kind of factors can be taken into account in the velocity potential map.

- *Path smoothness:* The smoothness of a path refers to the amplitude of the angles that are described while the robot follows the path. Concerns arise when robots

turning control is limited to a certain angle. Even when the mobile robot's base is holonomic omnidirectional, small angles require less energy for its execution. Furthermore, when the path is performed in the real robot, smooth trajectories are more human friendly since they are more predictable, and rough paths seem unpredictable and violent when turning. In [43], a "generic infrastructure for benchmarking motion planners" is presented and, in particular, an equation to measure the smoothness (denoted by κ') is proposed:

$$\kappa' = \frac{1}{n}\sum_{i=2}^{n}\alpha_i^2, \tag{3.4}$$

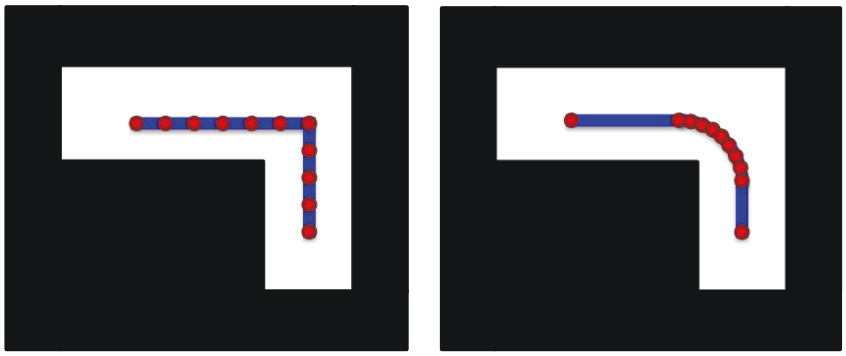
where n is the number of internal angles and α_i are the internal angles formed by each pair of consecutive segments in the path (α_i and the turning angle are supplementary angles). The above equation also can be formulated as

$$\kappa' = \sqrt{\frac{1}{n}\sum_{i=2}^{n}\alpha_i^2}, \tag{3.5}$$

which represents the quadratic mean of the angles α_i. Equation 3.4 is presented by the authors as a general frame that can be adapted to specific scenarios or robots. It is not easy to find weaknesses in this formula when using sampling-based methods because they produce similar results. However, when using other approaches such as the FMM-based planner, which is a search-based method, some problems have been found. We will illustrate it with an example.

Consider a simple scenario where the robot has to advance through a corridor and then turn right. Let us imagine that an algorithm generates a path with 11 points where all points advance in a straight line except one of them that is a right turn of 1.57 radians (Figure 3.19(a)). The smoothness is $\kappa'_{path_a} = 3.01$ radians according to Equation 3.5. Now let us consider that another method generates a path that describes a straight line and then smoothly turns nine times with $\alpha_i = 2.97$ radians in every turn until reaching the goal (Figure 3.19(b)). For this second path, the smoothness is $\kappa'_{path_b} = 2.97$ radians. According to the formulation above, the first path is the smoothest, but it becomes clear that the smoothest path is the right one in Figure 3.19.

A modification is proposed to obtain a more robust parameter. The idea is that those angles greater than ψ_s radians will be saturated, taking into account that these angles do not correspond to "real" turns. Moreover, when the robot is navigating in a straight line, small angles caused by different sources of error will not be considered in the equation. Therefore, the turning angles that actually



(a) Single turn of 1.57 radians. (b) Small smooth turns.

Figure 3.19: Example to distinguish between smooth and non-smooth paths.

define how smooth is the path will have a greater influence on the formula. The new equation can be formulated as

$$\alpha'_i := \begin{cases} \alpha_i, \forall i : \alpha_i \leq \psi_s, \\ \\ \psi_s, \forall i : \alpha_i > \psi_s, \end{cases} \tag{3.6}$$

$$\vartheta = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \alpha'^2_i}. \tag{3.7}$$

where ϑ is the parameter that is used to measure the smoothness and ψ_s is a threshold that has to be fixed. This formulation improves the benchmarking results because it obtains a more refined value that enables a better comparison between paths. Let us set ψ_s to 2.967 radians for the previous example. The smoothness is now $\vartheta_{path_a} = 2.85$ radians for the left path and $\vartheta_{path_b} = 2.97$ radians for the right one, which makes more sense according to the visual appearance of the paths.

Even though the new variable gives a better idea of the path smoothness, it has been found that it can be necessary to define another parameter to measure the presence of critical angles. For this reason, an additional parameter τ_s has been defined:

$$\tau_s = min(\alpha_i) - \omega_s, \tag{3.8}$$

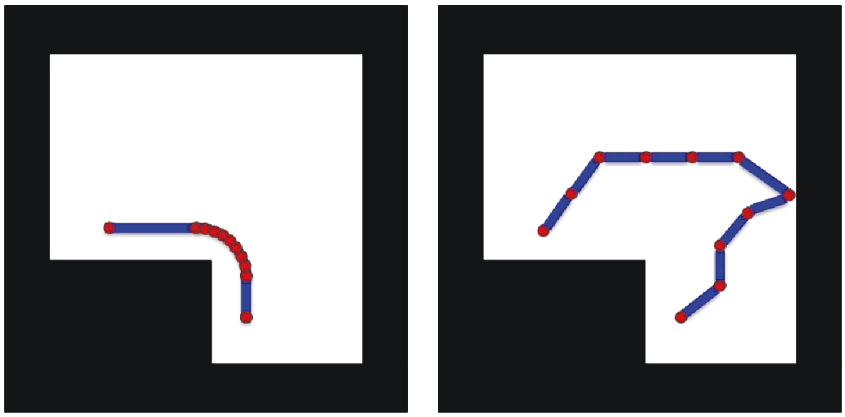
where ω_s is the control limit angle of the robot or, in other words, the maximum turn that the robot is able to perform. In the case of having a robot with a holonomic omnidirectional base, this parameter could represent a high energy

consumption or an inefficient angle. Returning to our example, $\tau_{s,path_a} = 0$ if ω_s is set to 1.57 radians. It means that at some point the robot reaches the maximum turning angle. In this way the "reliability range" τ_s would give us an idea of the probability of the robot to execute the path. The larger this unit, the more likely it will follow the path.

- *Clearance:* This metric is related to the distance from the path points to the closest obstacles, and it is determined by the average of the path points clearances. It has also been defined in [43]:

$$\mu_c = \frac{1}{n}\sum_{i=1}^{n}\delta_i, \tag{3.9}$$

where μ_c is the clearance and δ_i is the distance from the point i to the closest obstacle. Consequently, this parameter is supposed to give the safety of the path to some extent, but the manner in which it is calculated can lead to unreliable results. For example, consider the path planning example shown in Figure 3.20, where the minimum safety distance to obstacles is equal to 1.4 m.



(a) Safe path. (b) Unsafe random path.

Figure 3.20: Example to distinguish between safe an unsafe paths (clearance).

In Figure 3.20(a), all points are located at a distance of 1.5 m from the closest obstacle. The path clearance is $\mu_{c,path_a} = 1.5$ m. Another planner generates the path in Figure 3.20(b) with distances from 1.5 m up to 5 m for all points, but one of them is located at 0.1 m from a wall. The clearance for this path is $\mu_{c,path_b} = 3.15$ m.

According to the general formulation proposed in [43], the path in Figure 3.20(b) is more reliable and safe than the road presented in Figure 3.20(a), whereas

the Figure 3.20(b) path is more likely to collide against a wall. Observing the smoothness parameter, a similar procedure has been followed to fix this problem. A maximum safety distance ψ_c has been defined to represent the maximum distance from which the safety is not improved. Different thresholds can be used depending on the constrains of the specific problem. For example, it can be adjusted around a ten percent greater than the safety distance.

The new equation to obtain the clearance is defined as follows:

$$\delta'_i = \begin{cases} \delta_i, \forall i : \delta_i \leq \psi_c, \\ \\ \psi_c, \forall i : \delta_i > \psi_c, \end{cases} \tag{3.10}$$

$$\zeta = \frac{1}{n} \sum_{i=1}^{n} \delta'_i, \tag{3.11}$$

where ζ is the new parameter to measure the clearance.

Analyzing the previous example, when ψ_c is fixed to 1.54 m (ten percent greater than the safety distance), we obtain $\zeta_{path_a} = 1.5$ m and $\zeta_{path_b} = 1.4$ m.

This variable gives general information about the average safety of the path, but it cannot be used to find particular points where the mobile robot could collide against a wall. For example, in Figure 3.20(b), the path will be safe according to ζ_{path_b}, but there is one point where there will be a collision. For this reason, an additional parameter τ_c has been defined:

$$\tau_c = min(\delta_i) - \omega_c, \tag{3.12}$$

where ω_c is the navigation safety distance of the robot or, in other words, the minimum distance from the obstacles when the navigation is safe. In this way the "safety range" τ_c would give us an idea of the probability of the robot to execute the path without colliding. The larger this unit, the more likely it will follow the path without colliding.

- *Success rate:* It is equal to the percentage of times an algorithm is able to find a valid solution. Since the FM^2 planner is a complete and deterministic algorithm, it will always find a solution as long as it exists.

Ideally, the paths to be compared should have the same distance between consecutive waypoints. In this way, the benchmarks could generate more reliable results, since calculations would not be influenced by a significative difference in the number of points. However, it is not always possible to satisfy this condition.

All these metrics have been used to compare the planning algorithms in the experiments chapter. In particular, different ratios have been computed to quantify the relation between the RRT and the FM^2 approaches.

Chapter 4

Manipulation Planning

The robotic manipulation is generally the final operation of the autonomous mobile robots in order to move objects. The process often starts by navigating to locate the robot in reach of the objects to manipulate; once the location and orientation of the mobile base are proper for the task, additional systems are triggered to identify objects and calculate a trajectory to grasp the target avoiding obstacles.

The third objective of this work is covered in this chapter. Once the robot is located in a specific goal, the next step consists in reaching the object for its manipulation. This action corresponds to the third objective presented in the introduction chapter with a work flowchart diagram for the robot. Some approaches to obtain safe and smooth manipulation trajectories for a mobile robot are presented hereafter. The method proposed in next section includes manipulation of objects, which represents the fourth and fifth frames in the proposed work pipeline from introduction. In Section 4.2, a path planning algorithm with adaptive dimensionality is proposed. This method employs both the RRT and the FM^2 to calculate and improve paths with reduced time and good properties. In Section 4.3, a simpler approach of the adaptation of dimensions is presented in an implementation for a nuclear fusion device.

4.1 Evolving Strategy for Adapting Learned Manipulation Paths

A robot task can be represented as a set of trajectories conformed by a sequence of poses. In this way it is possible to teach a mobile robot to accomplish a manipulation task, and also to reproduce it. Nevertheless robot navigation may normally introduce inaccuracies in localization due to natural events as wheel-slides, causing a mismatch between the end-effector and the objects or tools the robot is supposed to interact with. Inverse kinematics could be used to calculate the learned path new configurations, but very often the calculations are difficult or not possible to obtained as convergence problems may arise in singular kinematic configurations.

We propose an algorithm for adapting manipulation paths to different locations. The adaptation is achieved by optimizing in position, orientation and energy consumption. The approach is built over the basis of Evolution Strategies, and only uses forward kinematics permitting to avoid all the inconveniences that inverse kinematics imply, as well as convergence problems in singular kinematic configurations. Manipulation paths generated with this algorithm can achieve optimal performance, sometimes even improving original path smoothness. Experimental results are presented to verify the algorithm.

4.1.1 Evolutionary Path Adaptive Problem

For a mobile manipulator a task may be defined as a sequence of points in the Cartesian space defining a path, [63]. This sequence of joint configurations is defined as:

$$\Omega_l = \{\vec{q}_k\}, \qquad k = 1, 2, \ldots, N, \tag{4.1}$$

where $\vec{q}_k \in \Re^n$ is a vector of joint variables $q_{k,j}$,

$$\vec{q}_{k,j} = (q_{k,1}, \ldots, q_{k,j}, \ldots, q_{k,n})^T. \tag{4.2}$$

The given path is denoted by Ω_l. This path describes the robot's goal or task, and may be obtained by learning methods through imitation, teleoperation, teaching techniques or telemetrics systems. Figure 4.1, shows an example path of Ω_l for the MANFRED-2 robot in the implemented 3D simulation environment.

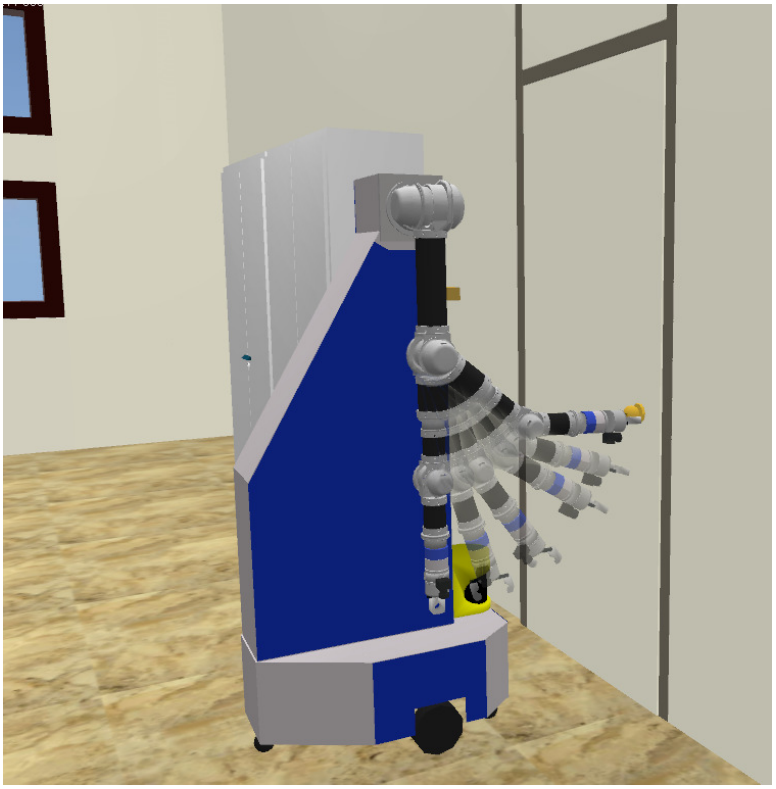


Figure 4.1: Manipulation learned path Ω_l.

By optimizing the end effector's position and orientations errors a new path is obtained for a predefined task. The trajectory is smoothened by considering the energy

consumption, and it is determined by the sum of the manipulator joints displacements as

$$\zeta(\Omega) = \frac{1}{2\pi} \sum_{k=1}^{N} |\vec{q}_k - \vec{q}_{k-1}| \tag{4.3}$$

In evolving methods, the step length is the disturbance introduced to design variables in order to change and evolve them from initial to optimum positions. This variation parameter is denoted by σ. In manipulation planning, the end-effector's position can be changed in only one axis with a movement, while orientation axes are hard coupled, varying all or at least two axes simultaneously when rotation over an axis is executed. This makes orientation optimization harder and more computational time consuming than that of position, as small changes in position could generate large variations in orientation. To overcome this problem, position minimization is first made with a large step length σ_{hi} approximated to that proposed by [50] and after position optimization target is reached, orientation minimization is added to optimization with step length σ_{low}, that is ten times smaller. This strategy results in improved convergence time of the algorithm.

Rechenberg's success rule is used for controlling the size of σ, [50]. After every N_b (number of design variables) iterations, the number of successes occurred over the preceding $10N_b$ mutations are revised. If this number is less than $2N_b$, step size is multiplied by a factor of 0.85, or divided by 0.85 if more than $2N_b$ successes occurred.

For the initial values of σ, Schwefel proposes to use the following estimation:

$$\sigma_i^0 = \frac{\Delta b_i}{\sqrt{N_b}} \tag{4.4}$$

where Δb_i is the expected distance from the optimum for the corresponding design variable. Notwithstanding as the accuracy for this initial σ value is not critical because the law of success seems to quickly adapt the step size, a generalized form is deduced to approximate initial Schwefel values

$$\sigma_i^0 = \frac{\Delta d_{q_N}}{10(N_b + 1)} \tag{4.5}$$

where Δd_{q_N} is the last node distance error in millimeters. The value obtained here approximates experimental results average of Schwefel estimations that made no significant differences on convergence times with respect to that of the exact estimation. When optimizing orientation, step length in (4.5) is reduced by a factor of ten.

There is no accurate rule for determining an appropriate parent population size μ. A good indicator is to have as many or a few times as many members as the number of design variables; parent population size used here is $N_b = 6$.

On the contrary, some theoretical studies have been realized on $(1, \lambda)$ strategies [50], where λ is the offspring population size, to estimate the optimal ratio between λ/μ. It has been shown that this ratio depends on the objective function and increases with its complexity. A ratio λ/μ equal to 5 can be considered as a good starting point, therefore an offspring population size of $\lambda = 30$ is chosen here.

Total error is the sum of the position error at each path point $k = 2, \ldots, N$, and the orientation error in the last two nodes $k = (N-1), N$, with weights $W_1 = 0.5$ and $W_2 = 1$ respectively, attaches greater importance to the last point where the robot is meant to perform the manipulation. Thereby, the joint configuration path must be transformed into end-effector position and orientation coordinates through the robot manipulator kinematic model. Position and orientation errors, denoted as E_P and E_0, are defined as [63]

$$E_P(\Omega) = \frac{1}{2R_{max}} \sum_{k=2}^{N} |p_{l_k} - p_k| \tag{4.6}$$

and

$$E_O(\Omega) = \frac{1}{2\pi} \sum_{k=N-1}^{N} |\varphi_{l_k} - \varphi_k|, \tag{4.7}$$

where $((p_l), (\varphi_l))$ are the desired position and orientation coordinates calculated by Ω_l forward kinematics, and R_{max} is the robot manipulator's maximum reach suggested by [64] as a normalization value. The resulting optimal joint path $\Omega*$ minimizes the total deviation with respect to Ω_l, and the optimization problem is realised by the minimization of:

$$f(\Omega) = w_1 E_P(\Omega) + w_2 E_O(\Omega) + w_3 \zeta(\Omega). \tag{4.8}$$

subjetc to:

$$C = \{\Omega \mid g(\Omega) \leq 0 \wedge h(\Omega) \geq 0\},$$

where g and h are restrictions imposed by the mechanical joint limits of the robot manipulator, and w_1, w_2 and w_3 are weighting factors used according to task priorities.

4.1.2 Evolution Strategies Adaptation Algorithm

The path evolutionary adaptation is accomplished with an implementation of the ES method denominated in [49]. The algorithm used to adapt the manipulation path is illustrated in Algorithm 18, where P_g, and Po_g are parent and offspring populations respectively, in generation g, and n_b is the number of design variables, which corresponds to the number of manipulator joints.

Algorithm 18 Evolution Strategies

```
initialization Pg = 0
evaluation Pg
while termination criterion ≠ true do
    Po_g ← Evolutionary mutation
    evaluation Po_g
    P_{g+1} ← selection(Po_g ⋃ P_g)
    if optimal position = true then
        reduce σ
    end if
    if g module(10n_b) = 0 then
        step length control
    end if
    g ← g + 1
end while
```

The $(\mu + \lambda)$-EE presented in [49] is used with some modifications to address the optimal path adaptation problem. A known initial manipulator configuration vector $\overrightarrow{q_1}$ is assumed, as well as a robot base location and orientation at learned path p_l.

Consider an initial population of μ parent individuals defined as in (4.1)

$$P_g = \{\Omega_{1,g}, \ldots, \Omega_{i,g}, \ldots, \Omega_{\mu,g}\},$$

where Ω_i represents a floating point vector with size $T = N.n$ and $g = 0, \ldots, g_{max}$ the generation number. In the scheme $(\mu + \lambda)$-EE the initialization process generates a population of μ random individuals distributed within the vector parameter bounds. If the initial location is unknown, then the use of an uniform distribution would be advisable to ensure the diversity of the population. Furthermore, if the initial joint configuration $\overrightarrow{q_1}$ is considered close enough to the learnt path initial node ($k = 1$), then we can intuitively assume that the optimal solution must be near the learned path Ω_l. This first estimation is included in the initialization process $P_{g=0}$, as the learnt path perturbation with a Gaussian probability distribution at the configuration nodes $K = 2, \ldots, N$, reducing the convergence time. Therefore, the initialization process can be expressed as

$$\Omega_{i,g} = \begin{cases} \overrightarrow{q_1}, & \text{if } k = 1, \\ \overrightarrow{q_{l_k}} + rand_G(0, \sigma^2), & \text{if } k = 2, \ldots, N \end{cases} \tag{4.9}$$

where $rand_G(0, \sigma)$ is a Gaussian distribution random number generator with zero mean and standard deviation σ, and $i = 1, \ldots, \mu$.

Once the population has been initialized, mutation is used to build a λ size offspring population. For each offspring, a parent is randomly selected, and each of its design variables b_i is mutated by adding a Gaussian random variable with zero mean and a standard deviation σ.

$$\Omega_{j,g} = \begin{cases} \overrightarrow{q_1}, & \text{if } k = 1, \\ \overrightarrow{q_{l_k}} + rand_G(0, \sigma^2), & \text{if } k = 2, \ldots, N \end{cases} \tag{4.10}$$

Where $j = 1, \ldots, \lambda$, the step length $\sigma = \sigma_{hi}$ during the position optimization phase and $\sigma = \sigma_{low}$ during the orientation optimization phase.

The objective evaluation function is used to assign a cost value to each member from parent and offspring populations: P_g and Po_g. The new parents P_{g+1} are selected from both populations as μ individuals with the best fitness, i.e. individuals with the lowest cost function.

$$\Omega_{i,g+1} = \begin{cases} \Omega_{j,g}, & if\ f(\Omega_{j,g}) \leq f(\Omega_{i,g}) \\ \Omega_{i,g}, & otherwise \end{cases} \tag{4.11}$$

The manipulator forward kinematics defined by an homogeneous transformation matrix is calculated to obtain total cost function on (4.8). Homogeneous transform is a four by four elements matrix that contains end-effector location used for (4.6) and a rotation sub-matrix that is used to determine orientation error for (4.7), in this way reduced computational time is achieved by avoiding exact angles calculation.

Optimization loop begins by minimizing the position error until a fitness value is reached, and then orientation error is added to the cost function. During the next few iterations the total error is incremented due to the influence of newly added criterion, but then both criterion errors are gently improved as the generations evolve.

The only drawback in obtaining E_O from within the homogeneous transform is that the calculated error in (4.7) is not directly proportional to the angle error since it is the result of mathematical functions applied to the end-effector orientation angles; therefore it can't be used as a termination criterion. This issue is overcome by checking angles after position fitness is reached. Termination criterion takes into account only the position error until a fitness value is reached, then exact angles error is evaluated, if orientation fitness is not reached the position fitness value is reduced and minimization process continues. As both errors evolve together, orientation fitness is found eventually.

To ensure generation of a feasible path, joint upper and lower limits need to be revised during optimization process. Joint limits are mechanical constraints that define the manipulator workspace, but also represent configuration values of reduced dexterity and hence should be avoided in the execution of the task. In the case of a boundary constraint violation, there are many solutions to replace values that have

exceeded their limits, [65]. Here a simple strategy is used, resetting the out-of-bound parameters with the exceeded bound value.

Finally mutation-selection process continues until convergence criterion is achieved or until the maximum number of generations is reached.

4.2 Path Planning with Adaptive Dimensionality

Until now, many motion planning algorithms have been proposed in the world of science, ranging from sampling-based and probabilistic planners [25, 66, 28, 6, 30, 32] to search-based methods [22, 23, 67, 29]. The most frequently used approaches for path planning rely on sampling-based methods. In Figure 4.2, an example of an RRT path with a parabolic smoother can be appreciated. These planners are usually fast, computationally inexpensive and dimensionally scalable. However, there are two key disadvantages when using sampling-based techniques instead of search-based approaches. First, the basis of the criterions used by the sampling-based methods does not bear any kind of optimization of the solution by itself. The obtained solutions randomly vary in path length, and the waypoints often get dangerously close to obstacles. There are some smoothing techniques that improve the path to some extend, but they are less effective in cluttered environments. Second, sampling-based methods are not complete. Conversely, the search-based methods guarantee to find a solution (they are complete) if it exist, and the algorithms are based on optimization paradigms. At the least, the search-based algorithms give suboptimality bounds and consistency in the solutions.

The search-based planners have just one disadvantage, which is that they become computationally expensive when dealing with too many dimensions. Such is the case of path planning with mobile manipulators, where the DOF of the robotic arm usually ascend to seven dimensions. In [68], the authors presented an adaptive dimensionality approach for path planning that uses the A* algorithm to calculate paths. The results obtained by the authors achieved to reduce the dimensional complexity of path planning with all the robot's DOF. Hence, in order to generate a path, the adaptive approach takes less time than the full-DOF planner. They conducted experiments with the RRT-Bidirectional method in order to make a comparison between both approaches. Even though the RRT-based algorithm performed faster than the adaptive approach in less complex task scenarios, it took longer to compute and the success rate dropped to 20% when the complexity of the manipulation problem was increased. It has to be said that the presented experiments involved a very specific and complex scenario rarely found in a human environment (a robot had to pass a long stick through a fixed hole). The adaptive method, as it is based in a complete search-based algorithm (A*), obtained a 100% success rate. When comparing the A* method to the FMM, the latter overcomes the first in path quality as exposed

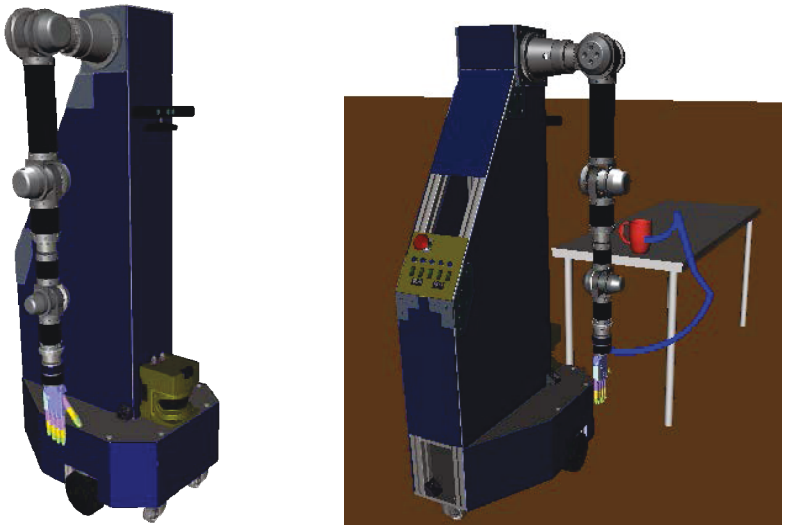


Figure 4.2: The mobile manipulator MANFRED2 in a simulation environment (left). Example of initial RRT-Bidirectional path (right).

in [58]. Furthermore, when using the FM^2 method (a newer and improved version of the FMM), the smoothness and safety parameters are improved. In this work, two adaptive algorithms that use FM^2 and RRT-Bidirectional are proposed. These approaches are designed to perform quickly and combine the two planning methods paradigms in human environments.

As corroborated by Pêtrès in [69], the curvature radius r of the FMM paths is influenced by its cost function f. According to Caselles *et al.* [70], the curvature radius r along the geodesic minimizing the functional $\int_D f((s))ds$ is bounded by:

$$r \geq \frac{inf_D\{f\}}{sup_D\{\|\nabla f\|\}} \tag{4.12}$$

where the supreme of the subset $\|\nabla f\|$ of the set D is the least element of D that is greater than or equal to all elements of $\|\nabla f\|$, and the infimum is the least element contained in D. The above equation states the relation between the cost function f and the smoothness of the optimal path obtained with the method. The bound, or limit of the curvature radius r_{lim} of the path, can be incremented by smoothing the f cost function. This result enables the FM^2 method to guarantee the smoothness of the calculated paths. In the case of robot manipulation, the path line described by the end-effector when executing a trajectory will not present peaks when using FM^2.

4.2.1 Problem definition

In this section, the representation of the path planning problem corresponds to a state-space S of dimensionality d. A state X in the robotic arm configuration is defined in 4.2, and in 3D space by the euclidean coordinates. A set of transitions $T = (X_i, X_j)|X_i, X_j \in S$, where (X_i, X_j) represents a valid transition between the states Xi and Xj. Each transition incurs in a cost $c(X_i, X_j)$ with $c(X_i, X_j) > 0$. A complete path is conformed by a series of transitions and is denoted by $\pi(X_i, X_j)$, where X_i and X_j are the initial and goal state configurations respectively. The incurring cost of executing a path will be $c(\pi(X_i, X_j))$ and the path with minimum cost will be denoted by $\pi^*(X_i, X_j)$. The presented definitions are translated into a search-graph problem with a graph G, composed of weighted edges T and vertexes S. The objective of the path planning algorithm is to calculate a minimum cost path $\pi(X_o, X_f)$ from a given initial state X_o to a goal X_f. Additionally, a suboptimality constraint ϵ can be defined to establish an optimality goal of $c(\pi(X_o, X_f)) \leq \epsilon.c(\pi^*(X_o, X_f))$

4.2.2 Adaptive Dimensionality Planner using FM^2

The path planning problem will generally require to be calculated using all available dimensions, and this is computationally expensive as dimensions increase. However, most of the paths have long sections where it is possible to reduce dimensions due to the problem structure, especially in uncluttered environments. In the case of manipulators, the dimensionality of the problem can be reduced to 3D end-effector paths and then the IK of the robot can be used to transform the solution to a full-dimensional path. The planar manipulators can be reduced to 2D in most cases. Whenever the dimensional reduced paths are not feasible to be converted into full dimensions, the path has to be planed with all of the robot's DOF. This may happen due to configurations of the robot in collision or out of reach. Taking advantage of these facts, the planning algorithm proposed in this section builds a state-space S^{ad} with transitions T^{ad}. Where the dimensionality of the problem is reduced and the full dimensional states are used as least as possible, but exclusively when a feasible path is not found in the dimensional reduced space.

As in [68], a high-dimensional space S^{hd} and a low-dimensional space S^{hd} are considered for path planning. The super indexes are used from now on to express a particular characteristic of the variable. S^{hd} and S^{ld} are considered to have h and l dimensions respectively, $(h > l, |S^{hd}| > |S^{hd}|)$. The S^{ld} space corresponds to a projection of the S^{hd} space into a space with less dimensions. In manipulation planning, the DOF of the robot are projected onto 3D as mentioned before. A mapping function between S^{hd} and S^{ld} is defined as follows

$$\lambda : S^{hd} \rightarrow S^{ld} \tag{4.13}$$

to pass from the high to the low dimensional space. In the robot, this corresponds to the forward kinematics of the manipulator's end effector. Also, the inverse operation is defined as $\lambda^{-1} : S^{ld} \rightarrow S^{hd}$ for mapping from the low dimensional states into the equivalent high dimensional states:

$$\lambda^{-1}(X^{ld}) = X \in S^{hd} | \lambda(X) = X^{ld} \tag{4.14}$$

λ^{-1} corresponds to the inverse kinematics. A tube-like surface τ of radius γ and thickness δ is built around the initial high-dimensional RRT-Bidirectional path π_o. The surface τ is defined in G^{ld} and thus consists of low dimensional states. A low dimensional state $X^{ld} \in \tau$ if there exist a point $X_i \in \lambda(\pi_o)$ such that the distance between X^{ld} and X_i is equal or greater than δ, and equal or smaller than γ. The euclidean metric distance in S^{ld} is used for these measurements.

Algorithm 19 presents the proposed method using FM^2 with Adaptive Dimensionality for manipulation. First, G^{ld} is assigned to G^{lad} (line 1), as initially all the adaptive state-space is low dimensional. An initial path is generated with RRT-Bidirectional and saved in the variable π_o (line 1). This initial path is full-DOF and executable, but as mentioned before it is far from optimal. The optimization process starts by converting π_o to a low-dimensional space $\lambda(\pi_o)$ (line 2). At the same line, a tube-like surface τ is created around π_o in the low-dimensional space. The surface τ is stored in G^{ad} which in that moment is entirely low-dimensional and includes the environment obstacles around the robot. In the next lines (lines 5 - 7), the FM^2 method is used in the low-dimensional space to improve the previously generated path π_o. First, a velocity potential map G^{vpm} is generated over the G^{ad} with the FMM. Then, a second FMM potential is generated from the end to the initial configuration point inside G^{vpm}. Finally, the gradient descent method is applied from the starting point X_o to the goal point X_f in order to determine the optimal low-dimensional path $\pi^*_{ad}(X_o, X_f)$. In the loop (line 8), the adaptive path $\pi^*_{ad}(X_o, X_f)$ is tracked to obtain a high-dimensional and executable path $\pi^*_{\tau}(X_o, X_f)$ (line 9). If it is not possible to track the entire adaptive path $\pi^*_{ad}(X_o, X_f)$, then a high-dimensional region is added to G^{ad} at X_{end}, or the space is grown if it already was in a high-dimensional region (lines 10 - 16). In the case of being able to track the entire $\pi^*_{ad}(X_o, X_f)$, a high-dimensional and executable path $\pi^*_{\tau}(X_o, X_f)$ is returned and the algorithm is finished (lines 17 - 19). A new path $\pi^*_{ad}(X_o, X_f)$ is calculated in every iteration of the algorithm (line 20).

In Algorithm 20 a variant to the proposed approach in Algorithm 19 is presented. The difference between the algorithms is that the initial FM^2 path is generated directly over the low-dimensional space without the tube-like surface τ. This approach generates a better path than the first approach in terms of path length. In contrast, being based on low-dimensional data, it is more likely to fail to be tracked in high dimensional spaces. The following is a detailed description of this algorithm. First,

Algorithm 19 FM^2 with Adaptive Dimensionality for Manipulation

$G^{ad} \leftarrow G^{ld}$
$\pi_o \leftarrow RRT\_Bidirectional(X_o, X_f)$
$\tau \leftarrow tubelike\_surface(\lambda(\pi_o))$
$G^{ad} \leftarrow \tau + G^{ad}$
An FM^2 path is calculated in next lines
$G^{vpm} \leftarrow FMM(G^{ad})$
$G^{FM^2} \leftarrow FMM(G^{vpm}, X_o, X_f)$
$\pi^*_{ad}(X_o, X_f) \leftarrow gradient\_descent(GFM^2, X_o)$
loop
  track $\pi^*_{ad}(X_o, X_f)$ for executable path $\pi^*_\tau(X_o, X_f)$
  if $\pi^*_\tau(X_o, X_f)$ is not found **then**
    let $\pi(X_o, X_{end})$ be the returned path
    if X_{end} is already within FullDOFRegion in G^{ad} **then**
      GrowFullDimRegion($G^{ad}, \lambda(X_{end})$)
    else
      AddFullDimRegion($G^{ad}, \lambda(X_{end})$)
    end if
  else
    return $\pi^*_\tau(X_o, X_f)$
  end if
  search G^{ad} for path $\pi^*_\tau(X_o, X_f)$
end loop

G^{ad} is assigned with G^{ld} (line 1), as initially all the adaptive state-space is low dimensional. An initial path is generated with the FM^2 method in the next lines (lines 2 - 4). First, a velocity potential map G^{vpm} is generated over the G^{ad} with the FMM. Then, a second FMM potential is generated from the end to the initial configuration point inside G^{vpm}. Finally, the gradient descent method is applied from the starting point X_o to the goal point X_f in order to determine the optimal low-dimensional path $\pi^*_{ad}(X_o, X_f)$. In the loop (line 5), the adaptive path $\pi^*_{ad}(X_o, X_f)$ is tracked to obtain a high-dimensional and executable path $\pi^*_{\tau}(X_o, X_f)$ (line 6). If it is not possible to track the entire adaptive path $\pi^*_{ad}(X_o, X_f)$, then a high-dimensional region is added to G^{ad} at X_{end}, or it is grown if it already was in a high-dimensional region (lines 7 - 13). In the case of being able to track the entire $\pi^*_{ad}(X_o, X_f)$, a high-dimensional and executable path $\pi^*_{\tau}(X_o, X_f)$ is returned and the algorithm is finished (lines 14 - 16). A new path $\pi^*_{ad}(X_o, X_f)$ is calculated in every iteration of the algorithm (line 17).

Algorithm 20 FM^2 with Adaptive Dimensionality for Manipulation (2^{nd} approach)

1: $G^{ad} \leftarrow G^{ld}$
An FM^2 path is calculated in next lines
2: $G^{vpm} \leftarrow FMM(G^{ad})$
3: $G^{FM^2} \leftarrow FMM(G^{vpm}, X_o, X_f)$
4: $\pi^*_{ad} \leftarrow gradient\_descent(GFM^2, X_o)$
5: **loop**
6: track π^*_{ad} for executable path $\pi^*_{\tau}(X_o, X_f)$
7: **if** $\pi^*_{\tau}(X_o, X_f)$ is not found **then**
8: let $\pi(X_o, X_{end})$ be the returned path
9: **if** X_{end} is already within FullDOFRegion in G^{ad} **then**
10: GrowFullDimRegion($G^{ad}, \lambda(X_{end})$)
11: **else**
12: AddFullDimRegion($G^{ad}, \lambda(X_{end})$)
13: **end if**
14: **else**
15: **return** $\pi^*_{\tau}(X_o, X_f)$
16: **end if**
17: search G^{ad} for path $\pi^*_{\tau}(X_o, X_f)$
18: **end loop**

A third variant of the algorithm where the low and high dimensional path calculations are made using the FM^2 method can be envisioned. In this case, the algorithm would be very similar to that presented by Gochev *et al.* [68]. The main difference would be in the used search method, which in the algorithm presented by Gochev is A* meanwhile in our approach would be FM^2. As mentioned before, although the

FM^2 takes more time to compute paths, it overcomes the quality of paths generated by A*, which would lead to a planner moderatly slower but better in quality of paths. This last approach is not evaluated here, and it is left as a future work.

4.3 Safe Motion Planning for a Nuclear Fusion Device Arm using Fast Marching Square

Inside the Joint European Torus (JET) there are some heavy components that must be replaced periodically, for which a robotic manipulator specifically designed to enter the vessel is used. It is teleoperated from a safe control room due to the radioactivity inside the vessel. As all maintenance is nowadays performed by remote control, a highly trained team specialized in teleoperation is needed, and operations inside the torus generally involve many work hours.

A strategy to generate the robotic manipulator-reaching path, reducing its dimensionality is proposed. It provides a smooth and safe path for the end-effector positioning, increasing safety and efficiency. Further, a velocities map based on obstacles proximity is generated in the path planning process, which could be used to limit the robot velocity as an additional security measure.

4.3.1 Dimensional Reduced Fast Marching Square Method

In this section an algorithm that solves the path planning problem for the articulated boom inside the torus of a tokamak fusion energy reactor, in a robust and efficient way, generating smooth and safe paths, is explained.

The velocities map is calculated and the FM^2 is performed over this map. The path planning method is depicted in Algorithm 21.

Algorithm 21 FM^2 method with dimensionality reduction for the JET environment.

1: $map \leftarrow load(free\_map)$
2: $vmap \leftarrow velocities\_map(map)$
3: $full\_path \leftarrow fast\_marching(map, vmap, start, goal)$
4: **while** $(global\ goal \neq true)$ **do**
5: $\quad move\_forward(full\_path)$
6: $\quad update\_forward\_kinematic(FK\_path)$
7: **end while**

First of all, it is necessary to comment some details about the environment and the robot's characteristics. The fundamental objective of the path planner presented here,

is to generate a path for reaching the goal location with the articulated boom end-effector. Considering the planar nature of the articulated boom, with the exception of the last link which is curved but is not considered for the reaching method, it is possible to reduce the six degrees of freedom to two dimensions. This reduction is achieved by crosscutting the JET torus at the height of the articulated boom. A 2D slice map is obtained for use of the algorithm as the environment, this has been modeled geometrically as an occupancy grid map in 2D and the robot's end-effector pose is represented with cartesian coordinates.

The algorithm starts by loading the obstacles map (line1). The velocities map is generated (line 2). This map is based on obstacles proximity with a value for every grid cell of the environment, and can be used to limit the robot's joints velocity as an additional security measure in the tokamak. The FM^2 is applied and the path is generated (line 3). Then, the forwards kinematics path is generated by incrementally moving forward the slider link of the articulated boom, and calculating the rest of links position so each link follows the line (line 4 and 5). Algorithm finishes when the end-effector reaches the goal position. The forward kinematics of the boom is calculated geometrically by using trigonometry starting at the tip of the slider link and moving forward to the rest of links until the end-effector is reached.

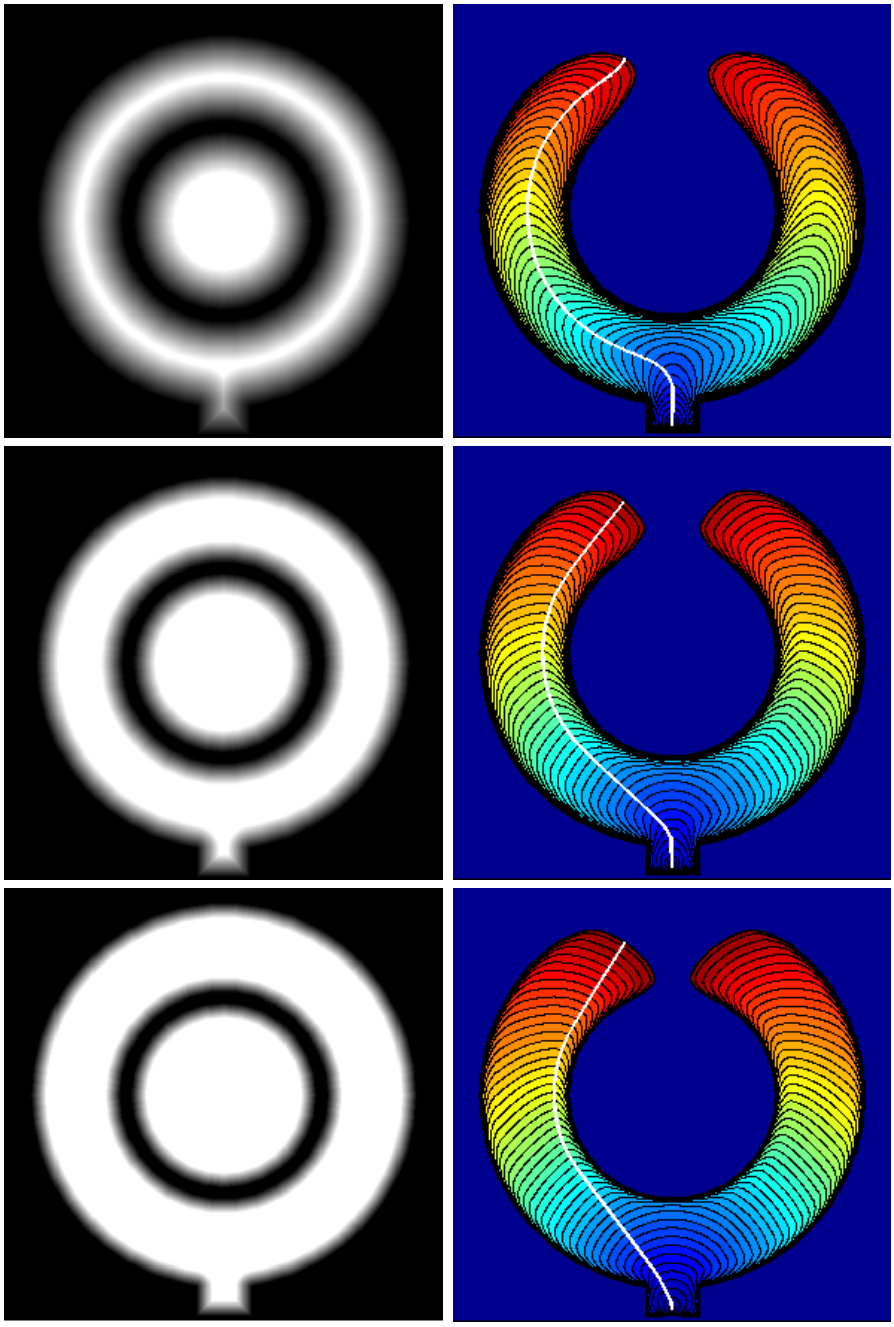


Figure 4.3: Examples of FM^2 paths generation with different saturation values for the velocity potential map.

Chapter 5

Experimental Results

The sections of this chapter present the simulations, experiments and benchmarkings that were conducted for the proposed path planning algorithms, each approach is exposed in a section.

5.1 Anytime Fast Marching Square

We have conducted experiments in a simulated environment taking into account the physics of MANFRED-2. It has a rectangular shape base of 1.15 x 0.86 m, and a mass of 129.55 kg. A SICK laser range finder is used for self localization and detection of obstacles.

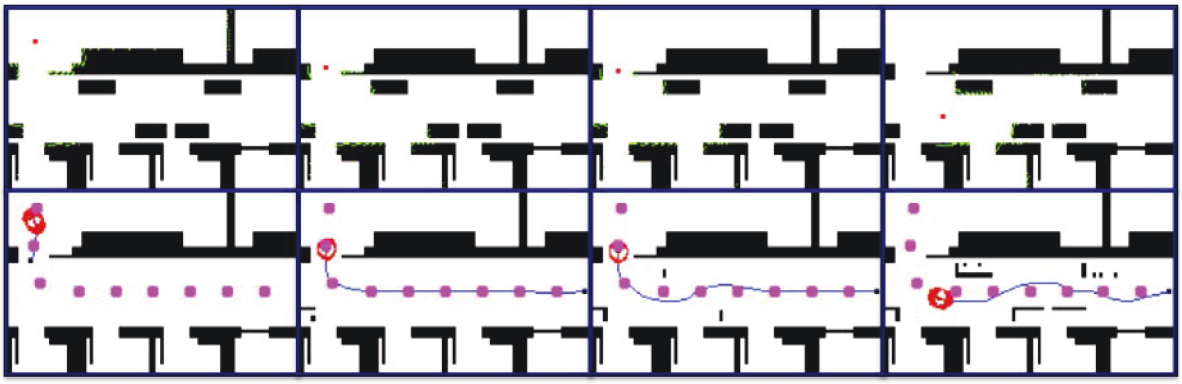
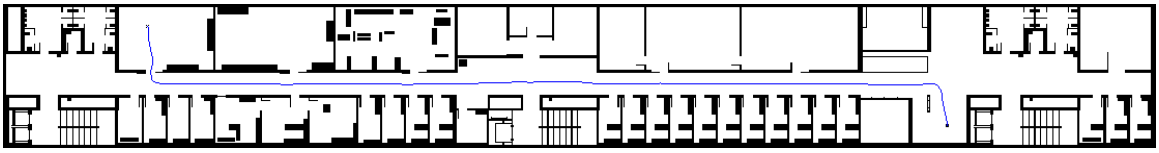


Figure 5.1: Sequence for distance measurement at every pointing direction within the laser range scanner (top). Replanning and navigation sequence, the magenta points represent the initial path (bottom).

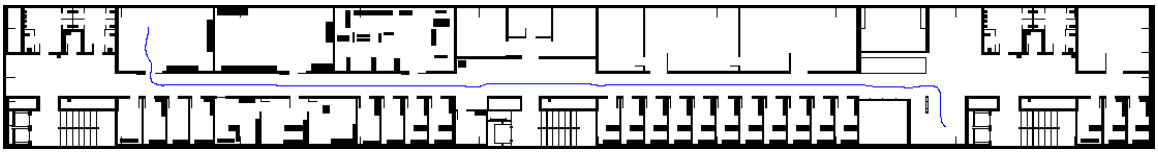
In Figure 5.1 (top), a sequence of images illustrates how a laser scan is performed, all the little green dots represent the laser ray measurements. The figures on the top sequence represent the real environment with all obstacles. The sequence of figures on the bottom shows how the incomplete initial map is updated with the new obstacles. Figure 5.1 (bottom sequence) shows a sequence with the generated paths obtained from replanning with AFM^2. The sequence on the right shows how the map is updated with new detected obstacles. The blue lines represent the dynamic plans to next sub-goals, and the magenta dots are samples of the initial path. The control cycle is made every 0.05 secs and every 0.5 secs a laser swept is made. All experiments have been developed in a simulated indoor environment: laboratories, corridors, and offices of the Carlos III University of Madrid. The dimensions of the environment are 116x14 meters (the cell resolution is 12 cm).

A first experiment is proposed starting from the upper left side of the map and ending in the lower right side, see Figure 5.2(b). The complete obstacles map is shown in Figure 5.5. Figure 5.4 presents the velocities map for the prior map, the velocity reaches its highest values in the light areas and its minimum values in the darker zones. The global path generated for the prior map is presented in Figures 5.2(a) and 5.2(b) with and without the wave propagation potential values respectively.

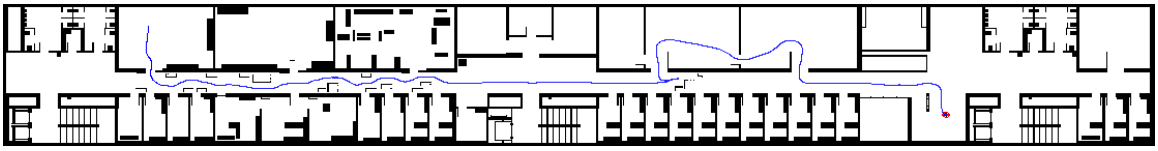
(a) Fast Marching Square Potential Map.



(b) Initial Global Planning.



(c) AFM^2 Planning with equal preloaded and real maps.

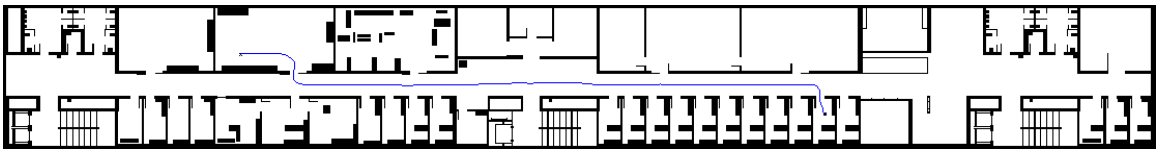


(d) AFM^2 Planning with obstacles.

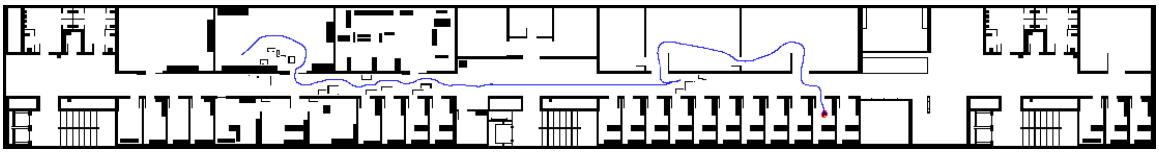
Figure 5.2: Planned Path for the first proposed experiment.

Figures 5.2(b) and 5.2(c) are presented to demonstrate that it is possible to obtain an executed global path different from the initial planned path. An experiment is carried out to prove the veracity of this sentence, where the AFM^2 receives identical prior map and obstacles map. First, the initial global path is generated in Figure 5.2(b), and then the robot starts to execute this path, updating the map with the laser scan information. Given that the obstacles map is the same as the prior map, the initial global path can be executed without additional recalculations. Even though, the resulted executed global path in Figure 5.2(c) is slightly different from the initial global planned path.This is due to the consideration of the robot physics and errors

during the execution of the path. Unlike the initial global path calculation where the robot is considered to be a point.



(a) Initial Global Planning.



(b) AFM^2 Planning with obstacles.

Figure 5.3: Planned Path for the second proposed experiment.

The initial global path is executed until the robot discovers obstacles on the corridor, then recalculation of the path to the next sub-goal is accomplished. The robot advances until it realizes that it is not possible to traverse the corridor and thereby it is blocked. The next generated path goes through adjacent rooms as shown in Figure 5.2(c). This is the final executed path.

Figure 5.4: High contrast map.

The experiment is repeated with different number of sub-goals, the obtained data is presented on Table 5.1. The subgoals are nodes from the initial path taken for the local replanning. In this way the replanning is calculated to a point near the location of the robot instead of the goal. The number of subgoals defines the interval of nodes in the initial path to which subgoals are stablished. For one subgoal node, that is replanning to the goal, the average time is 0.0787 secs and the sum of all recalculations is 37.29 secs; against an average time 0.0237 and sum of times 11.471 secs when segmenting and using 55 subgoals. Results show that an improvement is made in time when using subgoals. Greater differences could be appreciated with

larger maps. Times are also quick enough for real-time. In regards with the number of recalculations, there is not substantial differences in the table, but intuitively it is related to the density of new obstacles.

Table 5.1: Performances obtained when planning with different number of subgoals (All times in seconds).

Subgoal Nodes	Replanning	Av. Time	Sum times
1	474	0.0787	37.29
2	477	0.0807	38.486
3	474	0.0791	37.504
4	486	0.0777	37.752
5	484	0.0785	38.007
6	475	0.0778	36.962
7	479	0.0430	20.599
8	480	0.0762	36.594
10	550	0.0346	19.02
11	482	0.0779	37.541
14	478	0.0354	16.934
19	547	0.0248	13.539
28	483	0.0300	14.467
55	485	0.0237	11.471

A second experiment has been carried out. The obtained paths are presented in Figure 5.3. The method provides smooth trajectories that can be used at low control levels without any additional smooth interpolation processing.

5.2 Anytime Triangular Fast Marching Square and Benchmarking Parameters

Several experiments have been conducted in a simulated environment to make an adequate study of the algorithm and the benchmarking parameters. All the results are presented in this section, but first it is important to remark the computational complexity of the used algorithms. In the case of the FMM algorithm the computational complexity is $O(n)$ as shown in [59]. Since the FM^2 is based directly upon FMM, it is

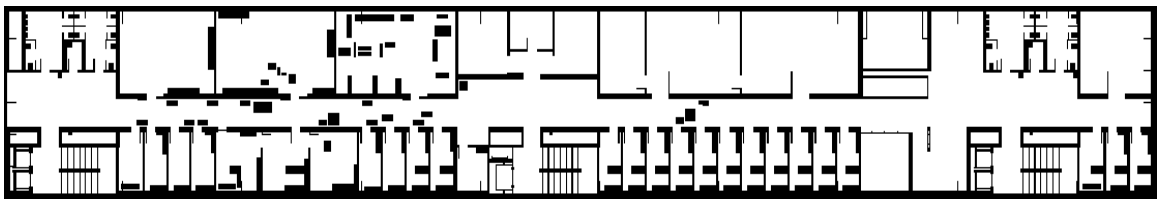


Figure 5.5: Obstacles map.

also $O(n)$. The RRT algorithm complexity is $O(n \log n)$ and presents no asymptotic optimality. As mentioned before the RRT is a probabilistic sampling-based algorithm, while the FMM and the FM^2 methods are search-based.

In the experiments, different options have been compared according to the combination of polygons proposed in Section 3.2.1. The ROI is constructed with hexagons or octagons, and it can contain only external vertices or both external and internal vertices. In our experiments, the configuration with external polygons (only external vertices) is called "external" and the configuration that contains internal and external polygons is referred to as "complete". Ten different paths have been tested for each configuration. The combinations that have been tested are listed in Table 5.2. The results will be discussed after one iteration and two iterations. Complete polygons are used in the second iteration because the objective of the planner is to refine the path.

Table 5.2: Combinations of polygons for ROI construction.

First Iteration	**Second Iteration**
◇ External Hexagon	◇ Complete Hexagon
	◇ Complete Octagon
◇ External Octagon	◇ Complete Octagon
◇ Complete Hexagon	◇ Complete Hexagon
	◇ Complete Octagon
◇ Complete Octagon	◇ Complete Octagon

First, the ROI is constructed. After that, the first iteration of the FM^2-based planner is executed over the triangular mesh and, at the end of this iteration, an improved path is obtained. The anytime approach comes into action when additional iterations are applied to further improve the path. This process continues until the optimality constraints are met, the goal point is reached, or the improvement of the current iteration is not significant. For simplicity, two iterations of the anytime approach are used in this experiments. Figure 5.6 shows some trajectories after one

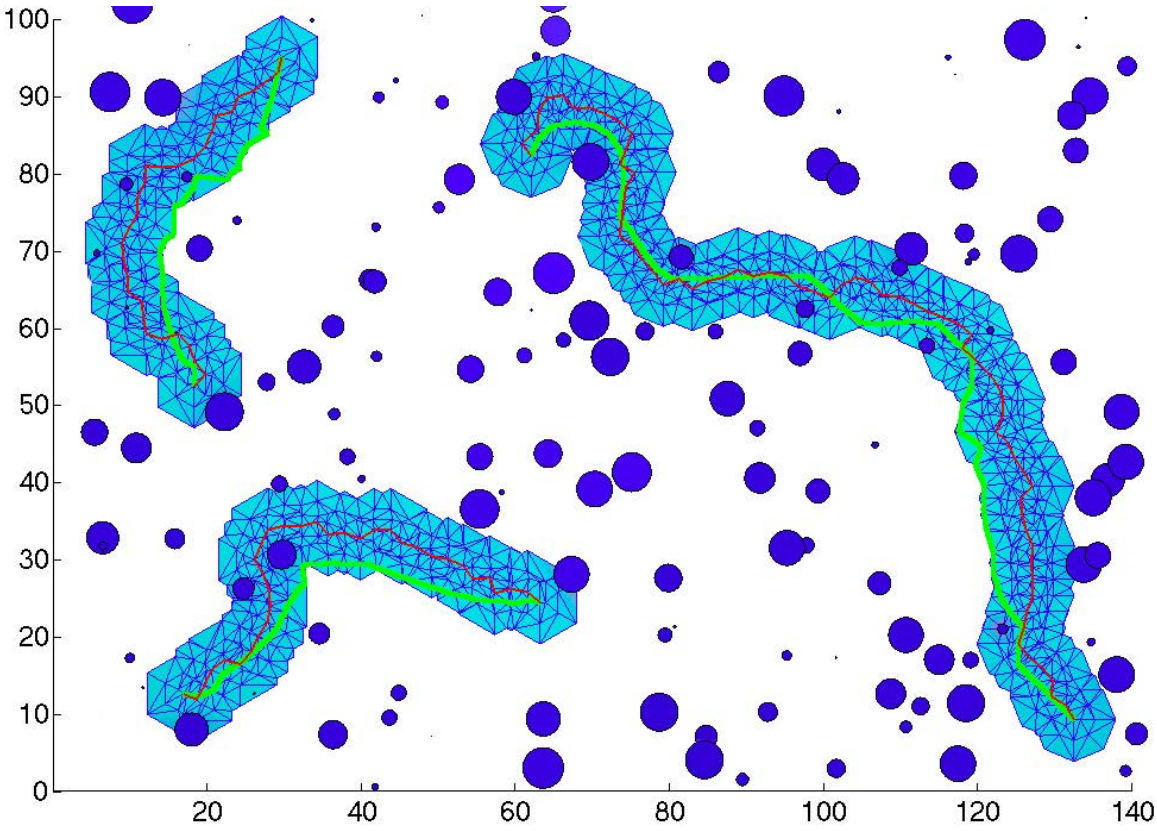


Figure 5.6: Three samples of the ten different trajectories after one iteration. Configuration: external hexagons in the left paths and external octagons in the right one. The red lines are the RRT paths, and the green lines are the ATFM^2 results.

iteration of the algorithm.

In Figure 5.7, some trajectories after two iterations are displayed. The first iteration is executed according to Figure 5.6 and, then, the paths are computed again in the second iteration. It can be easily observed that the RRT paths are improved in the first iteration. The algorithm performance after two iterations will be measured in this section. In order to do that, the benchmarking parameters introduced in Section 3.4 have been used. In addition, a comparison between the traditional parameters and the new ones is given.

The box plots of Figure 5.8 show a comparison between the computational time needed by the RRT planner to calculate the initial path and the computational time needed by the ATFM^2 approach to refine the RRT path (ROI construction and FM^2 path generation in one iteration). It can be appreciated that the time needed by

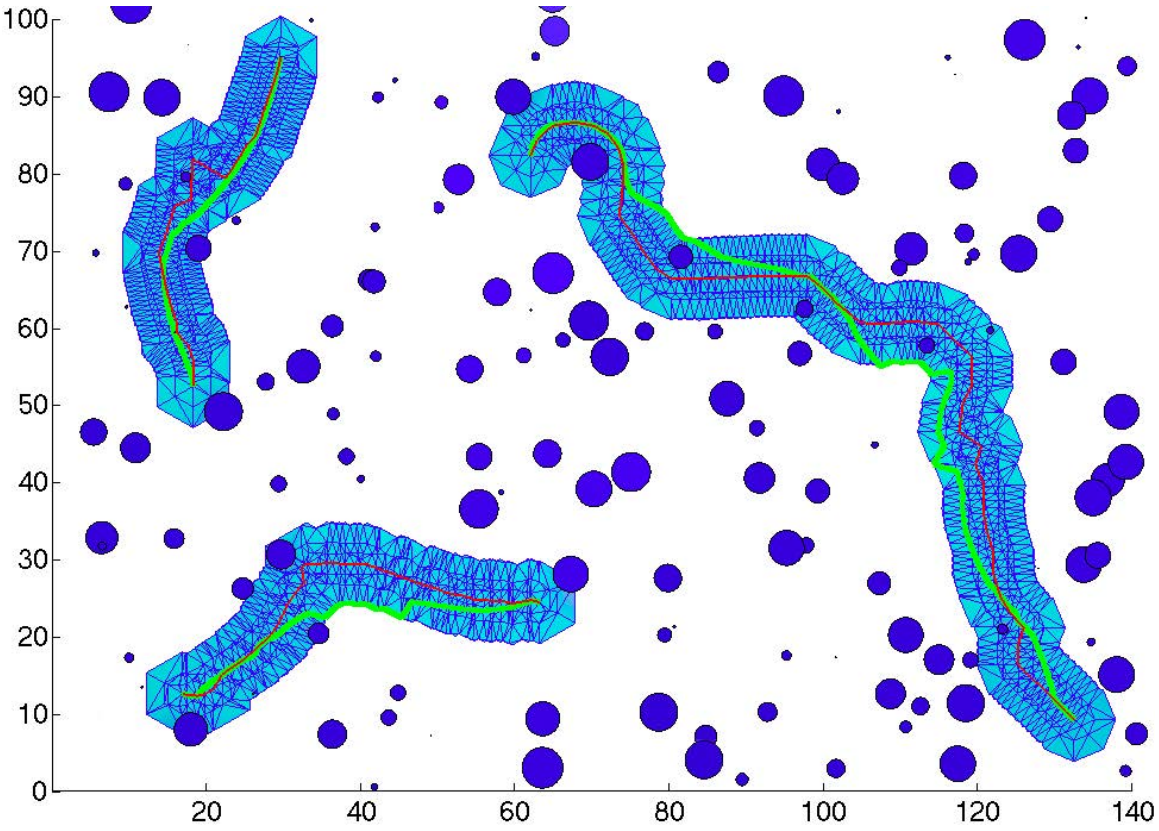


Figure 5.7: Three samples of the ten different trajectories after two iterations. Configuration of the second iteration: complete hexagons in the left paths and complete octagons in the right one. The red lines are the paths obtained after one iteration of the ATFM^2 method and the green lines are the paths after two iterations. First iteration according to Figure 5.6.

the ATFM^2 method is much lower than the time required by the RRT technique to generate the initial path, which is a logical result because the ROI drastically reduces the search space. The ATFM^2 method needs milliseconds to obtain the results, while the median time of the RRT algorithm is around 2.5 seconds. As a consequence, the anytime algorithm can be executed in real-time while the robot navigates through the environment.

Box plots are used to make a comparison between the ATFM^2 algorithm and the RRT planner. The central band inside the box is the median. The bottom and top of the box define the first and third quartiles. The ends of the whiskers represent the most extreme points not considered outliers. The outliers are plotted with red crosses

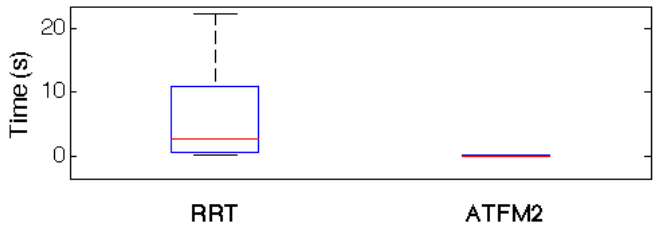


Figure 5.8: Computational times. Comparison between RRT initial path and ATFM^2 refinement in one iteration.

when necessary. The benchmarking parameters are calculated for each method and the ratio between these values is the variable that is used to compare both methods. Bar graphs with the averages are drawn to study the reliability and safety ranges. The following acronyms are used for the polygons:

- $Hex(E)$: external hexagon (Figure 3.6(b)).
- Hex: complete hexagon (Figure 3.6(a)).
- $Oct(E)$: external octagon (Figure 3.5(b)).
- Oct: complete octagon (Figure 3.5(a)).

The dash separates the configuration of the first iteration from the setup of the second one. For example, $Hex(E) - Oct$ means that an external hexagon is used in the first iteration and a complete octagon is utilized in the second one.

The computational times after one iteration of the ATFM^2 method are presented in Figure 5.9. The ratio ATFM^2/RRT is equal to the computational time of the first iteration of the ATFM^2 algorithm (ROI creation and FM^2 path generation) divided by the computational time of the RRT planner (initial path). The fastest configuration is $Hex(E)$, which is a logical result because this configuration corresponds to the lowest number of vertices.

For the second iteration, the computational times of the studied configurations are shown in Figure 5.10. The computational time of the ATFM^2 method is now equal to the sum of both iterations, including ROI creation and path generation using the triangular meshes. As expected, the computational time is closely related to the number of vertices. A higher number of vertices has a negative influence on the computational cost. The fastest results are obtained with hexagons ($Hex(E) - Hex$). Nevertheless, the median values of the proposed configurations are similar in all cases. The computational times are fast enough to use the planner in real-time applications.

Next, the other benchmarking parameters (path length, smoothness ϑ, and clearance ζ) are analyzed for the same configurations. As before, the quotients between

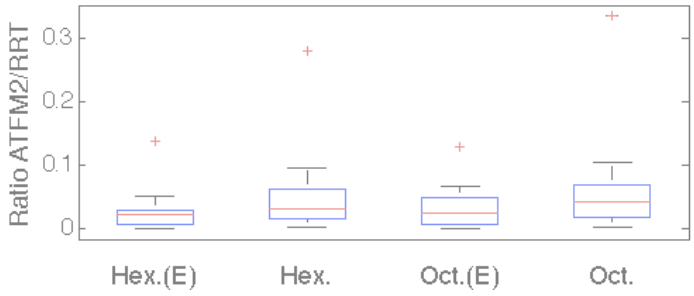


Figure 5.9: Computational time ratios for the first iteration of the ATFM^2 method.

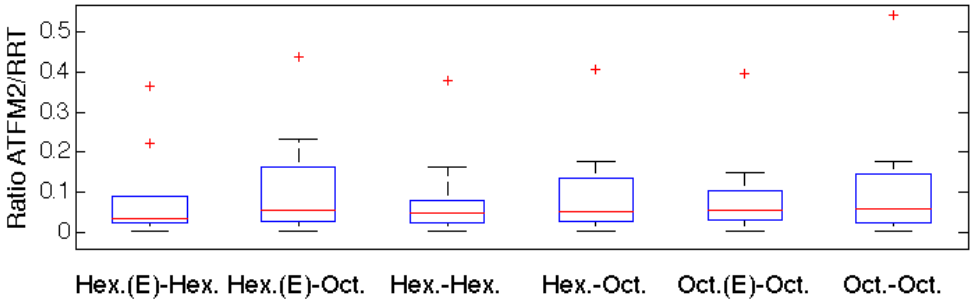


Figure 5.10: Computational time ratios for the second iteration of the ATFM^2 method.

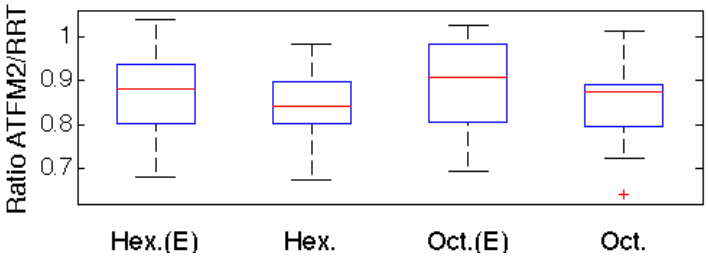


Figure 5.11: Path length ratios after one iteration of the ATFM^2 method.

the ATFM^2 and the RRT parameters have been represented in box plots (Ratio ATFM^2/RRT). In the case of the path length, ratios smaller than one indicate an improvement of the path. For the smoothness and the clearance, higher ratios mean better trajectories. Figure 5.11 shows the path length ratios after one iteration. The best results are obtained with the complete configurations (*Hex* and *Oct*), but all of them present similar values. It can be observed that the new path outperforms the RRT initial path in all cases.

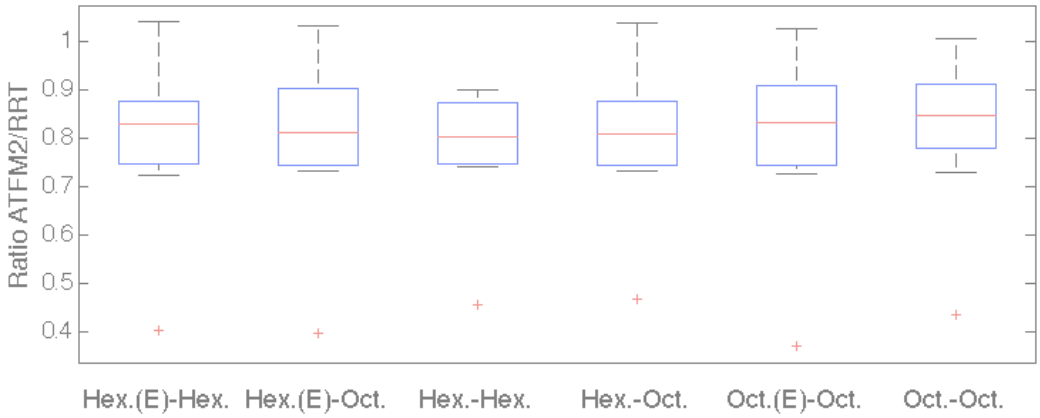
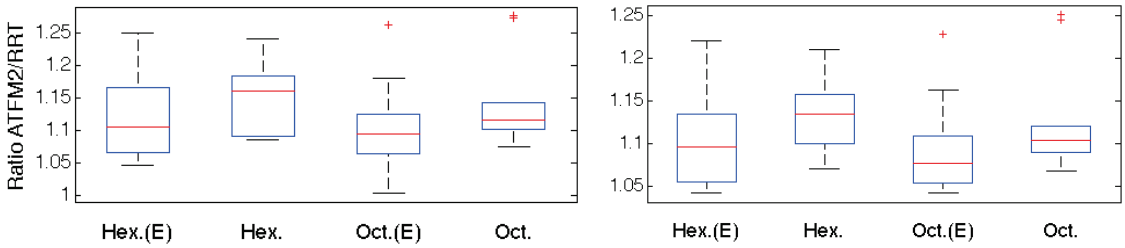


Figure 5.12: Path length ratios after two iterations of the ATFM^2 method.

In Figure 5.12, the path length ratios after two iterations are detailed for different configurations. As can be observed, all ratios present similar values. The best setup is $Hex - Hex$. The median values are slightly reduced when compared to the results after one iteration. As can be seen in the figure, there are outliers (red crosses) that represent a great reduction of the path length in all configurations. These outliers correspond to the path shown in Figure 3.8, where the ROI limits the optimization of the path length in the first iteration. It must be emphasized that an increment of the path length does not necessarily mean a deterioration of the path because there are other properties that could be improved.

The smoothness after one iteration is analyzed in Figure 5.13. The traditional formulation (κ') [43] has been utilized in Figure 5.13(a) and the new parameter proposed in this paper (ϑ, with $\psi_s = 2.97$ radians) has been computed in Figure 5.13(b). The results indicate that Hex is the best configuration according to the smoothness parameter. All ratios are higher than one, thus the smoothness of the RRT initial path is improved after one iteration. This result was expected because the RRT method does not consider the path smoothness and the FMM algorithm simulates a wave expansion following a gradient descent through a velocity potential map.

The reliability ranges (τ_s) were calculated with $\omega_s = 1.57$ radians in Figure 5.14. Positive values mean that the critical turning angle (ω_s) is not reached, and higher values correspond to a better performance. In this case, a bar graph is drawn to compare different types of polygons. The best value is obtained with the Oct configuration. The best medians of the worst turning angles are obtained with complete polygons. The RRT path and the $Oct(E)$ configuration present negative values. It means that the median of the worst tuning angle is worse than the critical turning angle, which is not a desirable situation.



(a) Traditional benchmarking.

(b) Proposed benchmarking.

Figure 5.13: Smoothness ratios after one iteration of the ATFM^2 method.

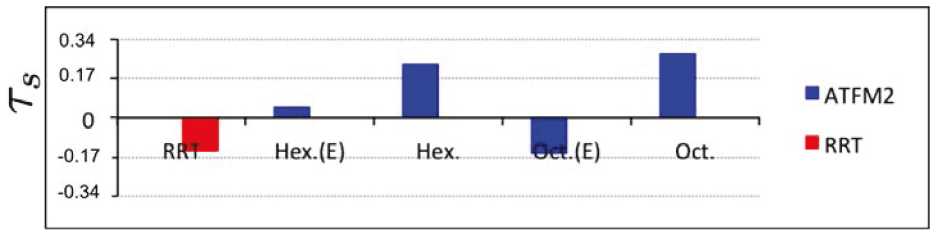


Figure 5.14: Reliability range. Comparison between the RRT initial path and the ATFM^2 method after one iteration. Units in radians.

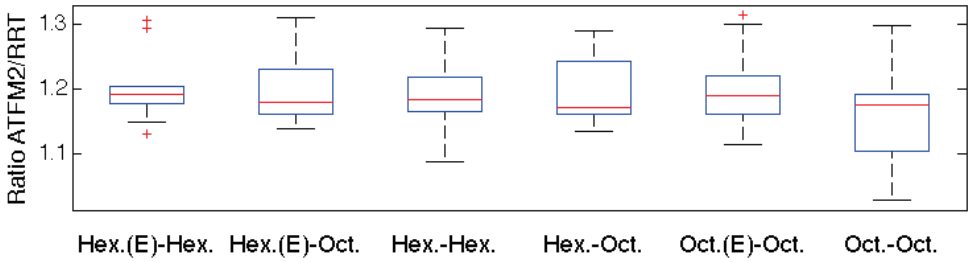


Figure 5.15: Smoothness ratios after two iterations of the ATFM^2 method. Traditional approach (κ').

The smoothness after two iterations is checked in Figure 5.15 (traditional approach) and Figure 5.16 (new formulation). As can be seen in both figures, all ratios are greater than one, which means that the smoothness is improved for all paths. Besides, the smoothness is also improved when compared to the results after one iteration. The best ratio is obtained with the $Hex(E) - Hex$ setup (1.19 for κ' and 1.15 for ϑ), but no big differences are found when comparing different configurations.

When comparing the new formulation to the traditional one, the edges of the

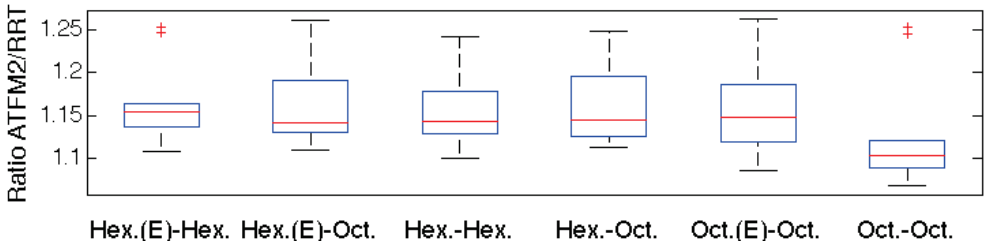


Figure 5.16: Smoothness ratios after two iterations of the ATFM^2 method. Proposed formula (ϑ, with $\psi_s = 2.97$ radians).

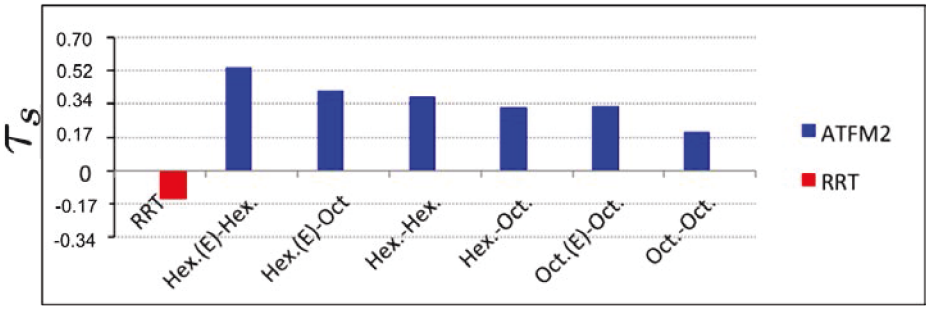
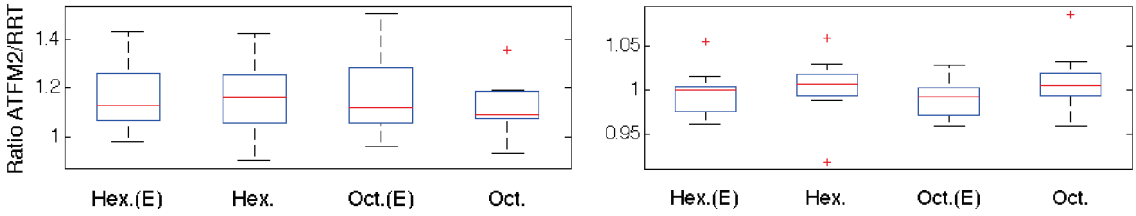


Figure 5.17: Reliability range. Comparison between the RRT initial path and the ATFM^2 method after two iterations. Units in radians.

boxes and the whiskers are closer to the median when the new formula is used. This result was expected and gives more critical information about the parameter. An interesting fact that can be appreciated in the figure is that the lowest values are slightly higher with the new formulation.

The reliability ranges after two iterations are shown in Figure 5.17. Once again, the best result is obtained with the $Hex(E) - Hex$ configuration. It can be concluded that the reliability range is significantly improved after two iterations. The ranges are greater than zero for all configurations, which means that the limit turning angle is not reached.

The last metric that has been measured in these experiments is the clearance. The results after one iteration are given in Figure 5.18(a) for μ_c and Figure 5.18(b) for ζ and $\psi_c = 1.4$ m. In both cases, the configuration with the highest median is Hex. An interesting fact is that the Oct configuration presents a better ratio (close to the best setup) with the new equation. This is a direct effect of the saturation included in the proposed formulation. The best ratios are obtained with the traditional benchmarking parameter. However, as discussed in Section 3.4, the new formula produces a more reliable parameter. The best values of ζ are slightly higher than one (complete



(a) Traditional benchmarking. (b) Proposed benchmarking.

Figure 5.18: Clearance ratios after one iteration of the ATFM^2 method.

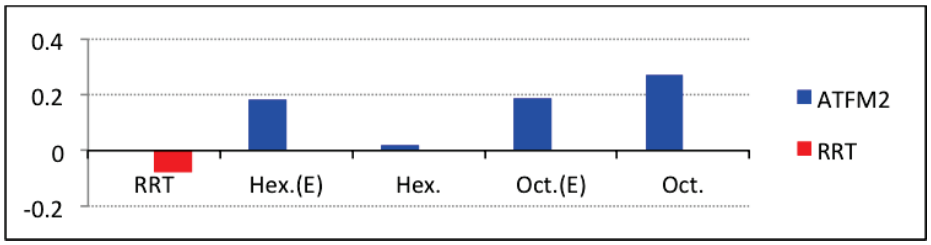


Figure 5.19: Safety range. Comparison between the RRT initial path and the ATFM^2 method after one iteration. Units in meters.

polygons).

The safety ranges were calculated with $\omega_c = 1.1$ m. In Figure 5.19, the results after one iteration are shown. The best value is obtained for the Oct configuration, followed closely by the $Hex(E)$ and the $Oct(E)$ structures. In all cases, the safety ranges are positive and better than the safety range of the RRT initial path.

In Figure 5.20, the clearance ratios after two iterations using the traditional formulation are presented. As can be seen, the $Hex - Hex$ configuration achieves the greatest value. In Figure 5.21, the same results are given for the new equation. As in the traditional formulation, the $Hex - Hex$ setup achieves the best clearance. Higher ratios are obtained with the traditional parameter.

The safety ranges (τ_c) after two iterations are displayed in Figure 5.22. The best result is computed for the $Hex - Hex$ setting. The median is better than the median of the RRT initial path in all cases, but there is one case in which the median of the safety ranges is negative ($Hex(E) - Hex$).

Summarizing, this experiments show that the configuration with hexagons tends to produce the best benchmarking results. Since the FMM needs to calculate the potential for every vertex, the $Hex(E)$ setup is the best option regarding the computational cost. However, all configurations are fast enough to be used in real-time applications. The $Hex - Hex$ setting produces the shortest paths, but the difference

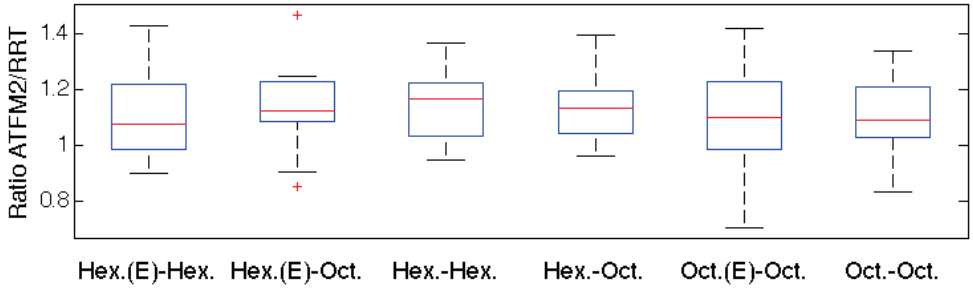


Figure 5.20: Clearance ratios after two iterations of the ATFM^2 method. Traditional approach (μ_c).

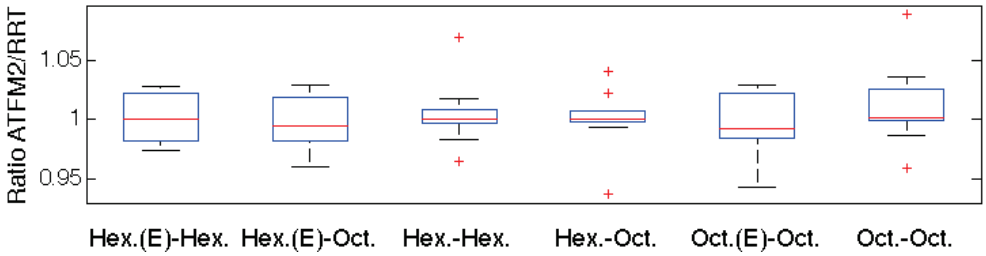


Figure 5.21: Clearance ratios after two iterations of the ATFM^2 method. Proposed formula (ζ, with $\psi_c = 1.4$ m).

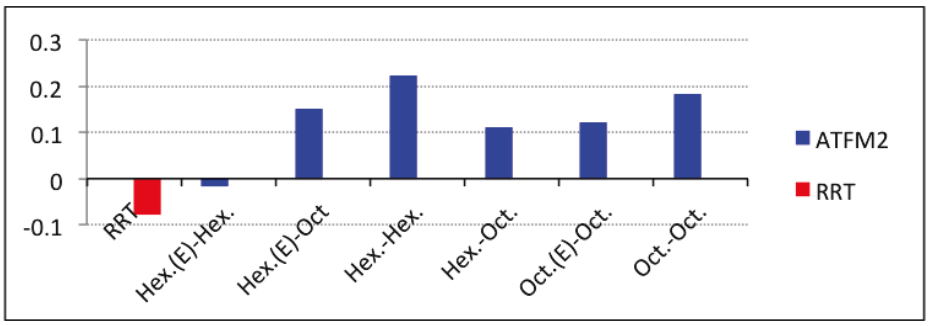


Figure 5.22: Safety range. Comparison between the RRT initial path and the ATFM^2 method after two iterations. Units in meters.

is not significant to conclude that this option is better than the other ones. Considering the path smoothness and the reliability range, the best options are Hex and $Hex(E) - Hex$. Regarding the clearance, the best choices are Hex and $Hex - Hex$

for the traditional formulation. Nevertheless, the best results are obtained with the $Hex-Oct$ structure when the new formula is applied. This fact suggests that including more points may improve the path clearance. Therefore, using a polygon with more vertices could be an interesting test to carry out in future experiments. The highest safety ranges are computed for Oct and $Hex-Hex$.

5.3 Nonholonomic Motion Planning

In order to test the performance of the method and show its versatility, this section presents several experiments and benchmarking results. In Figure 5.23, four paths are generated using the Control-based FM^2-NH with different initial and goal positions and orientations. It can be appreciated from the four scenarios in Figure 5.23, that the obtained trajectories are smooth and safe from start to goal location. The requirements of a good trajectory are fulfilled. An example of the velocity potential field is shown in Figure 3.13. For a better perception of the details, the image has been enlarged and vectors have been normalized. A close-up of the figure is presented in Figure 3.14. The methodology exhibits desirable features and versatility, generating trajectories that work properly for car-like robots.

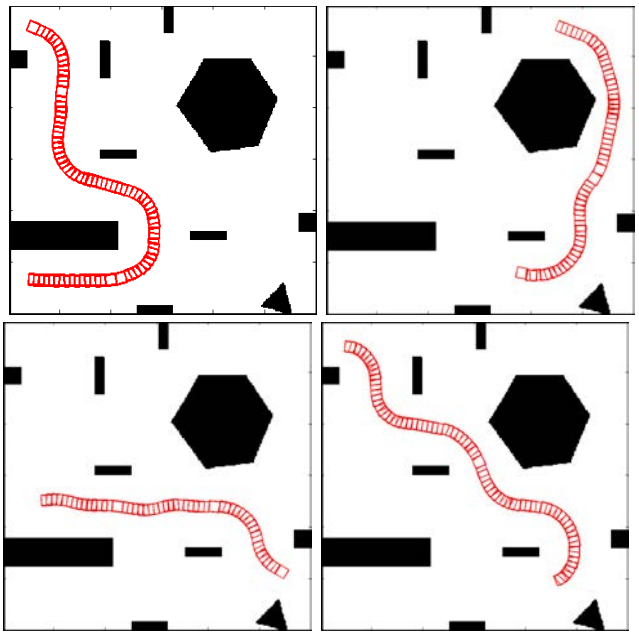


Figure 5.23: Different motion trajectories obtained with Control-based FM^2-NH.

In Figure 5.24 the same four scenarios of Figure 5.23 are taken. On this occasion, the paths are generated using the C-space FM^2-NH. The obtained trajectories are

smooth and safe from start to goal location. The results are very similar to those ones obtained using the Control-based FM^2-NH method.

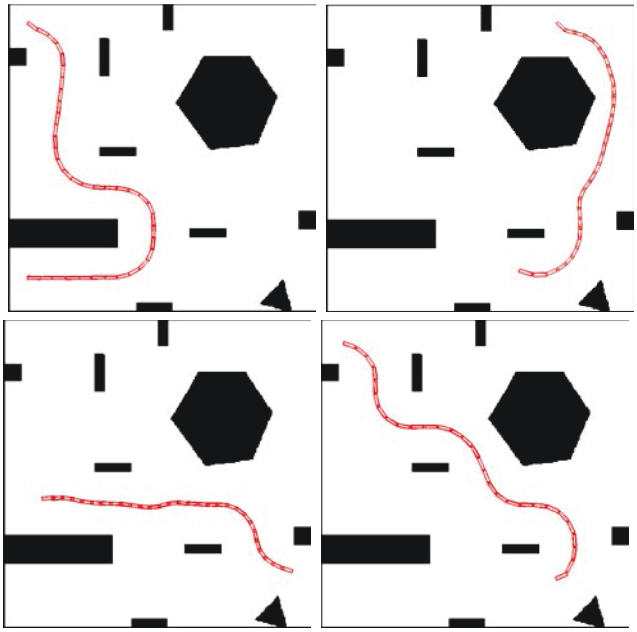


Figure 5.24: Different motion trajectories obtained with C-space FM^2-NH.

Different RRT paths are presented in Figures 5.26, 5.27, 5.28, and 5.29. All of the RRT paths in these figures match the different objectives in Figures 5.23 and 5.24, regarding to initial and goal points (including orientations). Figure 5.26 matches its objectives with Figure 5.23 upper left FM^2-NH path. Figure 5.27 has the same objectives as Figure 5.23 upper right FM^2-NH path. Figure 5.28 matches those objectives of Figure 5.23 lower left FM^2-NH path. Finally, Figure 5.29 has the same objectives as Figure 5.23 lower right FM^2-NH path. As appreciated in the Figures, the FM^2-NH methodology outperforms the quality of paths generated with RRT-NH method. From a visual inspection, the FM^2-NH generated paths seen safer, smoother and shorter. These suspicions will be discussed in the next section, where benchmarking parameters for path planners will be introduced.

5.3.1 Comparison of methods and benchmarkings

The common limitation of all the reactive navigation methods is that they cannot guarantee global convergence to the goal location because they use only a fraction of the available information (the local sensory information). Some researchers have worked on introducing global information into the reactive collision avoidance methods to avoid local trap situations. This approach has been adopted by Ulrich [71],

which uses a look-ahead verification to analyze the consequences of a given motion a few steps in advance to avoid trap situations. Other authors exploit the information about global environment connectivity to avoid trap situations (Minguez [72]). Those solutions still maintain the classical two level approach, and require additional complexity at obstacle avoidance level to improve the reliability. The proposed method is consistent at local and global scale because it guarantees a motion path (if it exists), and does not require global replanning supervision to restart a planning when a local trap is detected or a path is blocked. Furthermore, the path calculated has good safety and smoothness characteristics.

Most of the other methods give paths that are not smooth, even though they only provide a few loose points united by segments of straight lines. The only methods that give comparable results are based on harmonic functions (the solutions of the Laplace equation) but they have the problem of slowness.

The RRT is suited for high degrees of freedom. It works well with six or seven DOF in regards to computational time because it generates paths with quick response, but the additional complexity supplied by the nonholonomic approach makes the RRT to function less effectively. Our method has not been tested with higher degrees of freedom, but for the problem addressed in this work (3 dimensions) good results are obtained, as is set out in this section.

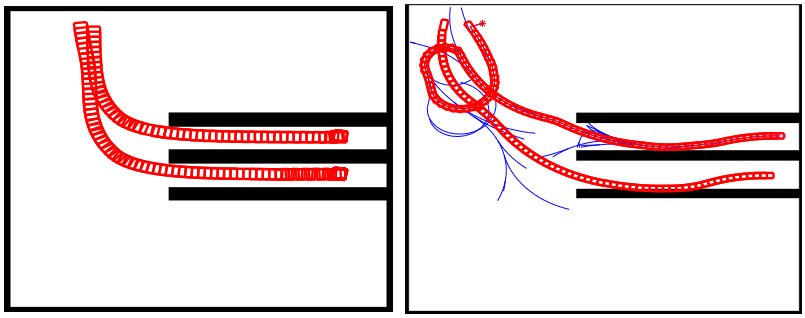


Figure 5.25: Trajectories with Control-base FM^2-NH (left) and RRT-NH (right).

Figure 5.25 has two different simulations. On the left side the trajectory generated by the FM^2-NH is clearly shorter than the one calculated by the RRT on the right hand side. The limitations of the RRT are specially important in this example, where the results obtained are very poor. We can illustrate that with further comparisons between the FM^2-NH and the RRT paths in figures previously presented.

In order to provide metrics of the quality of the methods, the performance parameters introduced in Section 3.4 are employed [43]. The computational time, the path length, the smoothness and the clearance parameters are briefly described below:

- *Computational times:* The execution time is computed for each stage of the

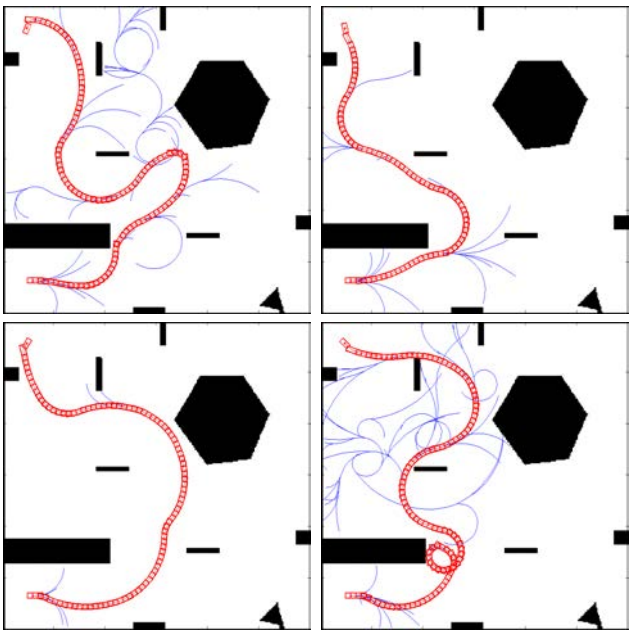


Figure 5.26: Motion trajectories obtained with the RRT-NH for the first experiment.

planning method, whether the algorithm is calculating a path or optimizing in some way an already generated path.

- *Path length:* This parameter is the sum of the distances from one way point to the next one in the planner state space.

- *Path smoothness:* The smoothness of a path refers to the amplitude of the angles that are described while the robot follows the trajectory.

$$\kappa' = \frac{1}{n}\sum_{i=2}^{n}\alpha_i^2, \tag{5.1}$$

where α_i represents the angle between two consecutive segments of a path with n segments.

- *Clearance:* This metric is related to the distance from the trajectory points to the closest obstacle, and it is defined as

$$\mu_c = \frac{1}{n}\sum_{i=1}^{n}\delta_i, \tag{5.2}$$

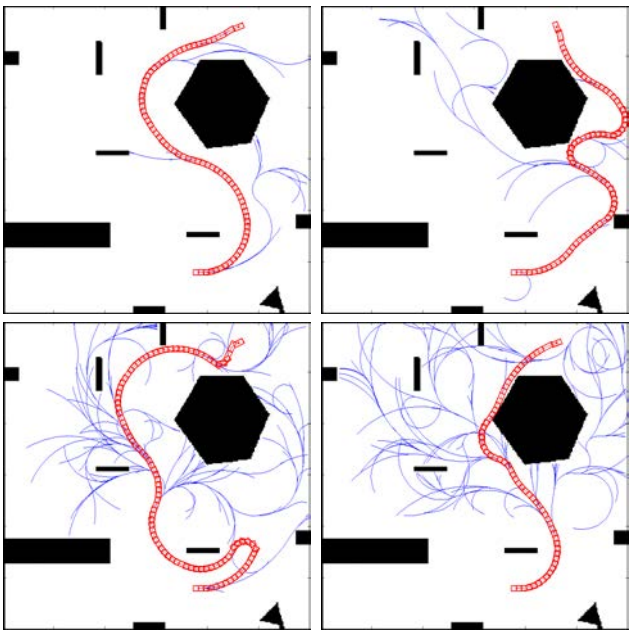


Figure 5.27: Motion trajectories obtained with the RRT-NH for the second experiment.

where δ_i represents the euclidean distance from the point i of the path to the closest obstacle and n is the number of points of the path.

- *Success rate:* It is equal to the percentage of times an algorithm is able to find a valid solution. Since the FM^2-NH planner is a deterministic algorithm, it will always find a solution as long as it exists.

Box plots are used to present the benchmarking results in order to make a comparison between the FM^2-NH and the RRT-NH planners. In the plots, the central band inside the box is the median. The bottom and top of the box define the first and third quartiles. The ends of the whiskers represent the most extreme points not considered outliers. The outliers are plotted with red crosses when necessary.

Over twenty five experiments were conducted for each of the presented methods. The benchmarking parameters were calculated for all of them. The box plots in Figure 5.30 show the computational time required by both the the RRT-NH and the Control-based FM^2-NH methods in order to calculate a path. It can be appreciated that the time needed by the Control-based FM^2-NH is much lower than the time required by the RRT technique to generate a path. The FM^2-NH needs milliseconds to obtain the results, while the median time of the RRT algorithm is around 15.1 seconds. As a consequence, our approach is able to recompute new trajectories very fast, so that changes in the goal point could be addressed.

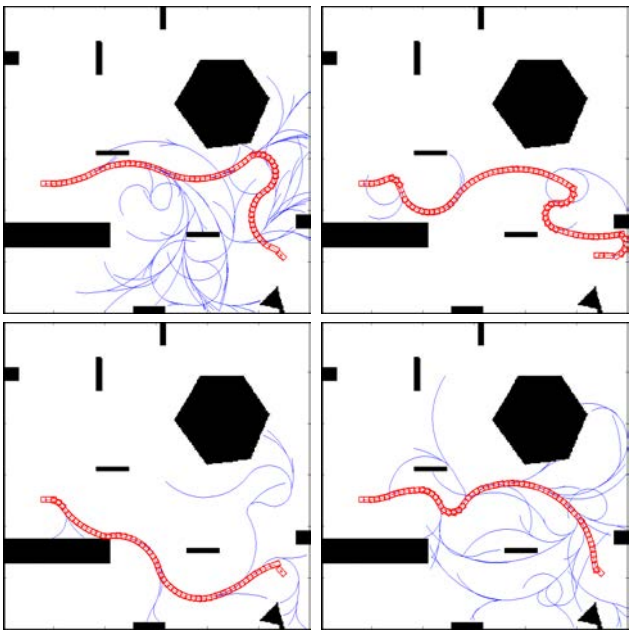


Figure 5.28: Motion trajectories obtained with the RRT-NH for the third experiment.

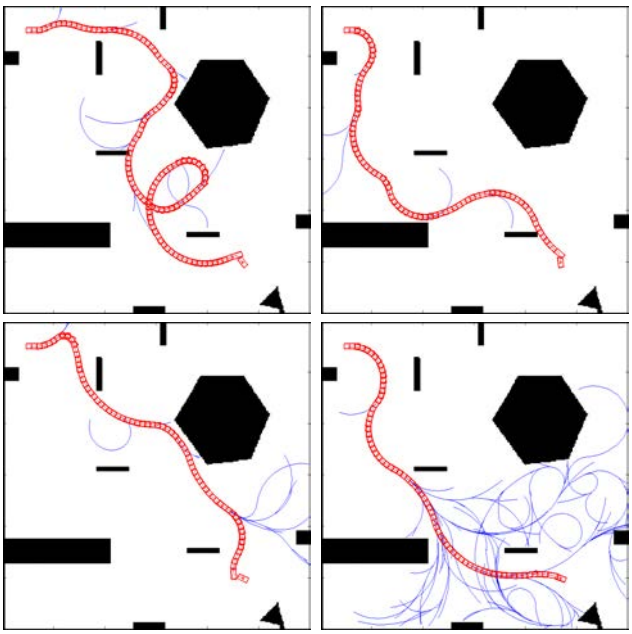


Figure 5.29: Motion trajectories obtained with the RRT-NH for the fourth experiment.

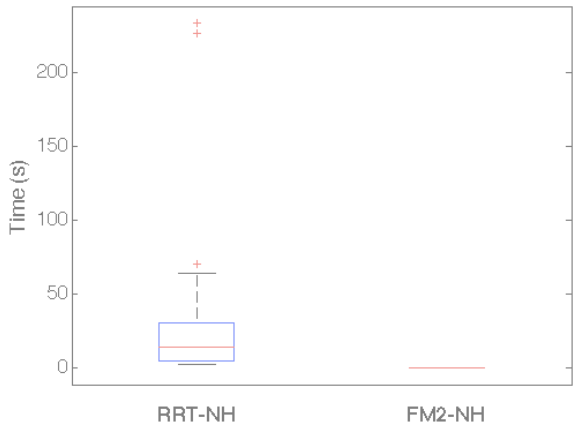


Figure 5.30: Computational time benchmarks for RRT-NH and the Control-based FM^2-NH method.

In Figure 5.31, the computational time required by the RRT-NH is now compared with the C-space FM^2-NH approach. In this benchmark the time of the FM^2 based method is rose to seconds. However, the time of the C-space FM^2-NH is much lower than that of the RRT-NH. The C-space FM^2-NH needs in median 3.63283 seconds of computational time.

In Figures 5.32 and 5.33, the ratio FM^2-NH/RRT-NH is the variable used to compare both methods. In this figure, the rest of benchmarking parameters (path length, smoothness ϑ, and clearance ζ) are analyzed. The same configurations, initial and goal locations and orientations, were taken. In the case of the path length, ratios smaller than one would indicate that the FM^2-NH is better. For the smoothness and the clearance, higher ratios mean that the FM^2-NH is superior.

In Figure 5.32, it can be observed that the path length ratio is smaller than one. This means that the length of the paths generated with the Control-based FM^2-NH are smaller in median than those of the RRT-NH. It should be noticed, that for these examples the velocity potential map was not saturated, which means that the length of the FM^2-NH approaches can be further reduced.

The smoothness ratio in Figure 5.32, approximates to one. Therefore, the paths are similar in smoothness. This result was expected because the nonholonomic restrictions prevent the planners from taking pronounced turns. In the regular RRT method the angle of the turns tend to be more violent. Finally, the clearance ratio is greater than one, showing a significant advantage for the Control-based FM^2-NH over the RRT-NH. Naturally, this result was expected since the RRT methods do not consider safety measures in their implementation, meanwhile the FM^2 based methods intrinsically include this parameter through the velocity potential map. In Figure 5.33, very similar benchmarking results were obtained for the C-space FM^2-NH. Both

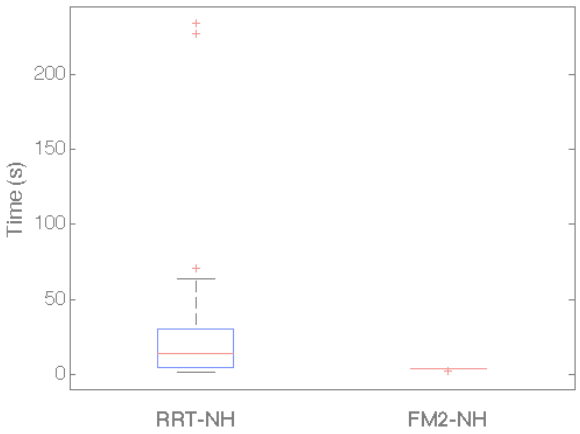


Figure 5.31: Computational time benchmarks for RRT-NH and the C-space FM^2-NH method.

approaches generate consistent trajectories with desirable properties.

An additional advantage of the obtained implementation is the ease of including, through the velocity potential map, other constraints such as uneven terrains, slopes, friction, winds or currents in the case of underwater applications.

5.4 Adaptive Evolving Strategy

The proposed methodology is tested in a simulation environment with a non-redundant mobile manipulator robot denominated MANFRED-2 [2], which consists of a six degrees of freedom (n = 6) anthropomorphic arm mounted over a two degrees of freedom mobile base (n=2).

This robot was built at the Carlos III University of Madrid. Figure 5.34 shows the MANFRED-2 robot in the implemented 3D simulation environment; our laboratory was modeled with elements such as doors and small tools to test grasping and manipulation tasks, simulations include body dynamics to increase realism and assure veracity.

The Denavit-Hartenberg parameters and joint limits for mobile manipulator MANFRED-2's robotic arm, are presented in Table 5.3.

The simulation environment software is used in order to obtain a convenient manipulation path Ω_l, this is accomplished by actioning servomotors separately until a desired pose is found. When a desired pose is reached, our software enables us to save the robot's configuration and move to the next point; each pose is saved to form a manipulation path that can be then executed as a sequence of poses that lead to the the goal reaching point.

Figure 5.32: Path length, smoothness and clearance benchmarks for the ratio of RRT-NH divided by the Control-based FM^2-NH.

Table 5.3: MANFRED-2 robot Denavit-Hartenberg parameters and joint limits.

Art.	α_j	$a_j(m)$	θ_j	d_j	q_j^{ba}	q_j^{al}
1	90	0	0	0.25	-90	90
2	-90	0.4	0	0	0	180
3	-90	0	-90	0	-90	90
4	90	0	0	0.35	-90	90
5	-90	0	0	0	-90	90
6	0	0	0	0.25	-90	90

The Ω_l path describes our known task with $N = 6$ points, as shown in Figure 4.1. Forward kinematics is calculated using [73], obtaining the end-effectors position and orientation $\{(x_k, y_k, z_k), (\phi_k, \theta_k, \psi_k)\}$ for each point $k = 1, 2, \ldots, 6$.

Table 5.4 shows the end-effector points coordinates for Ω_l. The robot location is given by the parameters (x_b, y_b, θ_b) referenced to a point predefined on the simulation map, where (x_b, y_b) determine the position coordinates in meters and θ_b the robot base orientation in degrees, position in z_b is not included since the robot base keeps the robotic arm at the same height all the time. For Ω_l, the location is $p_l = (-2.319, -2.138, 180°)$.

Learned path Ω_l is tested on the 3D simulation environment, results show how the robot reaches an experimental tool. Subsequently, the door knob is grabbed when the robotic hand is closed, and robot is capable of opening the door by moving its base backwards.

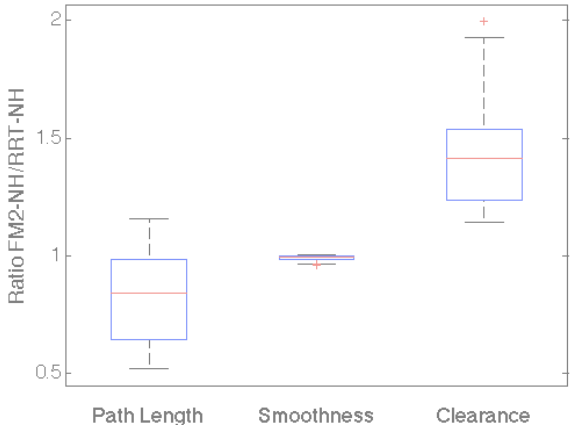


Figure 5.33: Path length, smoothness and clearance benchmarks for the ratio RRT-NH divided by the C-space FM^2-NH.

Table 5.4: End-effector learned path in Cartesian space.

k	X_k	Y_k	Z_k	ϕ_k	θ_k	ψ_k
1	250	147.63	-1000	$-180°$	$0°$	$-90°$
2	250	147.63	-980.62	$174.27°$	$17.09°$	$-108.86°$
3	250	284.83	-923.95	$156.88°$	$28.39°$	$-131.93°$
4	250	402.01	-834.35	$131.93°$	$28.39°$	$-156.88°$
5	250	491.23	-718.62	$108.86°$	$17.09°$	$-174.27°$
6	250	546.82	-585.47	$90°$	$0°$	$180°$

Two new random locations are used to verify the algorithm's effectiveness: p_1 and p_2, these locations are different in position and orientation from that of the known path. An initial robot arm configuration $\vec{q_1} = \{0°, 0°, 0°, 0°, 0°, 0°\}$ is assumed for all cases, and the offspring population size is $\lambda = 30$ as proposed in Section 4.1.1. The algorithm is executed 20 times for each location. Table 5.5 shows robot base locations for Ω_l, Ω_1 and Ω_2.

Once initial population members are generated using (4.9), the algorithm starts executing iteratively to minimize (4.8).

The mutation process of the candidate population is made via software, verifying that the generated poses are collision free, as evolving candidates in real robots is dangerous.

Position is optimized first with configuration parameters: $F = 0.012$ and σ in accordance to (4.5). After fitness is reached, orientation is added to the objective function with σ reduced by a factor of ten; termination criterion of end effector error to be less than 2.5 mm is being set. Test results are shown in Table 5.6.

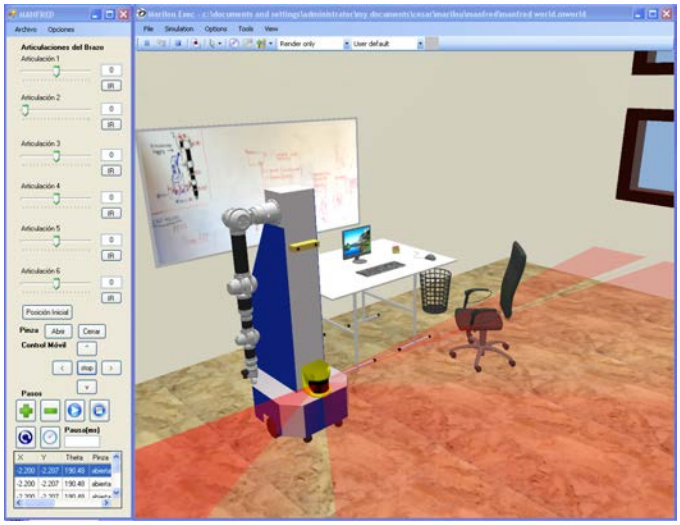


Figure 5.34: The MANFRED-2 robot in simulation environment.

Table 5.5: Position and orientation coordinates of robot base for Ω_l, Ω_1 and Ω_2

Path	x(m)	y(m)	θ
Ω_l	-2.319	-2.138	180.00°
Ω_1	-2.294	-2.104	181.48°
Ω_2	-2.200	-2.207	190.48°

.

For an execution of the algorithm at p_1, a solution Ω_1 is found after $g = 120$ generations in 1.5 seconds. In orientation terms an error of 2.98° is obtained on the $N-1$ point, and 0.29° on the last one. This makes sense when we recall the weighting factors for orientation optimization: $W_5 = 0.5$ and $W_6 = 1$. Lines described by the learned and evolutionary algorithm adapted path are shown on Figure 5.35, it can be seen that the adapted path fits position closely with a soften adaptation curve when approaching to the known path; a position error of 1.78 mm is obtained in last node for this test execution. Results showed that intermediate points presented lower position errors and greater orientation errors than others because they are only optimized in position, while last two nodes presented minimal orientation error.

All obtained manipulation paths are tested in the three-dimensional dynamic simulation environment as well as in the real robot where paths are executed and the sequences are reached correctly. The door is opened when additional steps are executed. The door knob had to be taped to increase the friction with the robot gripper and enable turning, but beyond this detail, the simulation represented precisely the real environment. Figure 5.37 shows a picture of the robot reaching the door knob.

Table 5.6: Path generarion statistics for Ω_1 and Ω_2.

	Generated paths:	
	Ω_1	Ω_2
Number of tests	20	20
Mean convergence generation	186.2	261.8
Best convergence generation	31	40
Worst convergence generation	500	500
Mean convergence time	2.18s	2.71s
Best convergence time	0.59s	3.92s
Worst convergence time	4.87s	4.90s
Mean position error	1.96mm	1.73mm
Min. position error	0.76mm	0.28mm
Max. position error	2.72mm	1.22mm
Mean orientation error	0.7831°	0.3703°
Min. orientation error	0.5679°	0.0059°
Max. orientation error	1.1180°	0.8448°

The same perspective of Figure 4.1 could not be presented because of a wall next to the robot's arm.

Experimental results show that errors can be minimized so the robot can carry out defined tasks. Time in worst-case scenario rose up to 4.9 seconds when reaching maximum generation $g_{max} = 500$, which stays within the proposed real-time threshold. Further investigation over our previous work [74] found that the forward kinematics calculus library consumed most of the algorithm computational time. Therefore, a more computationally optimal forward kinematics function was implemented reducing execution time significantly. When g_{max} is reached the fitness distance between the best and the worst individuals in population is taken as the convergence criterion, observe closely Figure 5.36 and 5.38.

On account of a reduced parent population size $\mu = 6$, lines on Figures 5.36 and 5.38 followed closely. Also an error peak when orientation optimization begins at around generation 20 can be observed on Figure 5.38. This peak is less obvious on Figure 5.36 because robot base orientation error is smaller in that case.

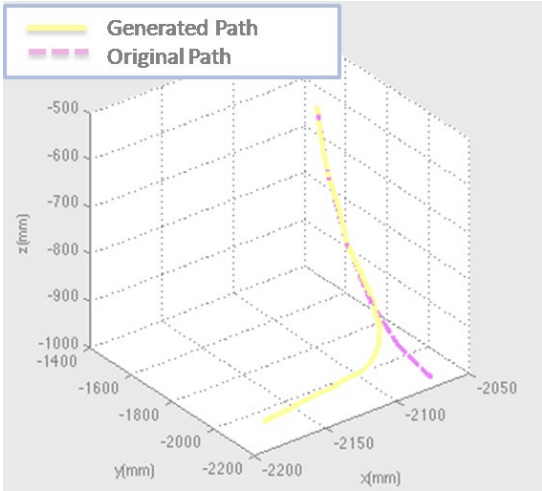


Figure 5.35: Adapted and learned manipulation paths, Ω_1 and Ω_l.

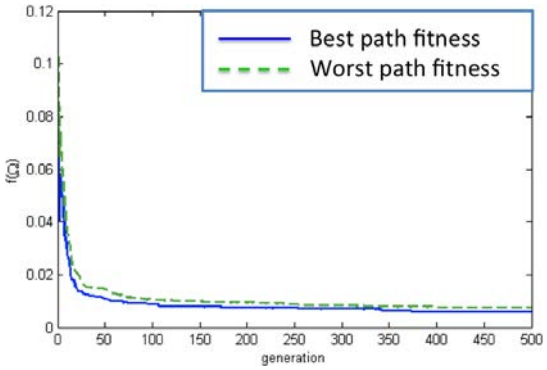


Figure 5.36: Path fitness evolution for Ω_1 through g_{max} generations.

5.5 Adaptive Dimensionality Motion Planning

This section presents the experiments that have been carried out to test the method performance. The robot manipulator MANFRED2 has been modeled in a simulated environment which takes into account the robot geometry and its dynamics.

The first experiment is implemented using Algorithm 19. In Figure 5.39 (left), the initial path π_o generated with RRT-Bidirectional is presented. The algorithm that generates this path includes a smoother that makes the path shorter and a little smoother. As it can be appreciated in the figure, there are peaks and rough changes of direction in the path. π_o is taken as the input to the next phase of the method, where a tube-like surface τ is built around the path with closed endings. The FM^2 method is used to generate a soft and secure path inside τ. The obstacles are taken

Figure 5.37: Mobile Robot MANFRED-2 reaching a door knob.

into account though the velocity potential map G^{vpm}. In Figure 5.39 (right), the surface of τ and the resulted FM^2 path (inside τ) can be appreciated.

The algorithm continues its execution within the loop and begins tracking the path to obtain a fully executable path. For this particular example, the inverse kinematic solver fails to find a solution for the first points. These misses of the inverse kinematic solver are due to the initial configuration of the arm, which is totally extended and hence in limited maneuverability. Then, the full dimensional space is grown by adding regions to G^{ad} around $\lambda(X_{end})$. The loop is executed three times until a solution is found. In Figure 5.40 (right), the result of the algorithm for the first experiment in the simulation environment can be appreciated. The red line represents the part that was recalculated within S^{hd}. The resulted path would comprise the red path until its end. After that point, the path continues the blue line until the goal posture. π_o is drawn in the left side of Figure 5.40. The FM^2 path is in the middle and the final

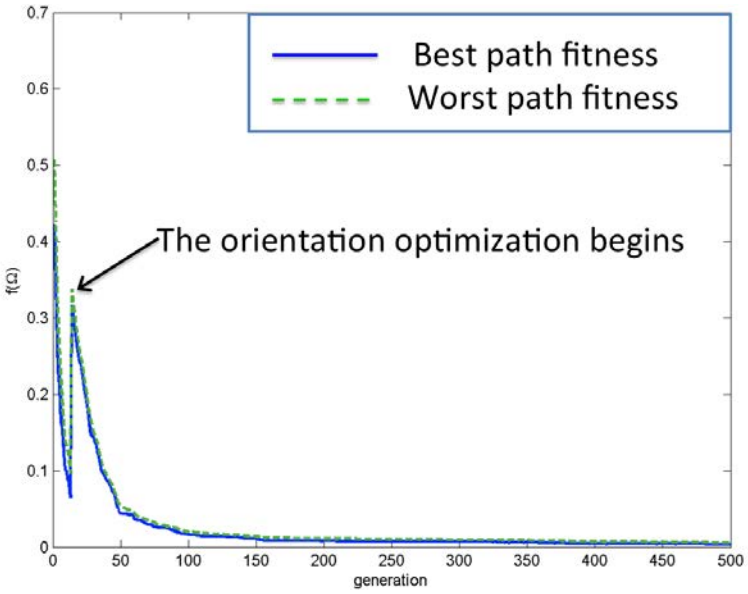


Figure 5.38: Path fitness evolution for Ω_2 through g_{max} generations.

executed path is on the right side. The obtained path is 27.52% shorter and 7.34% smoother than the initial π_o.

A second experiment is carried out with Algorithm 19. A retracted initial configuration for the arm was chosen in order to avoid the limitations of the reach and the inverse kinematics solver. The initial path π_o is generated using RRT-Bidirectional, as can be observed in the simulation environment in Figure 5.41 (left). Then, a tube-like surface τ is built surrounding π_o. The FM^2 technique is used to generate a soft and secure path inside τ (middle of Figure 5.41). As expected, the inverse kinematic solver is able to track the path and convert it to a high-dimensional space (S^{hd}). The executed path is drawn in blue in the simulation environment in Figure 5.41 (right). The obtained path for this experiment is 5.18% shorter and 11.82% smoother than the initial π_o.

For the second approach (Algorithm 20), two experiments have been carried out. The conditions of the conducted experiments (X_o,X_f) are the same tested in the first approach. First, the adaptive low dimensional path π^*_{adap} is calculated from X_o with the extended arm configuration. The loop is started and the path is tracked. For this experiment, π^*_{adap} is entirely tracked and converted into a high dimensional space in one execution of the loop. In Figure 5.42 (right), the result of the algorithm for this experiment in the simulation environment can be appreciated. The obtained path for this experiment is 31.04% shorter and 2.65% smoother than the path obtained with Algorithm 19 under the same conditions.

Finally, a second experiment for Algorithm 20 is performed. The adaptive low

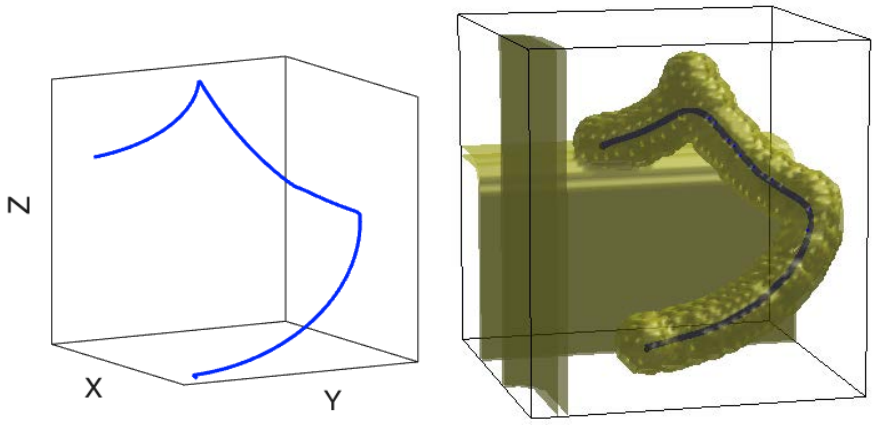


Figure 5.39: The initial RRT-Bidirectional full-DOF manipulation path (left). The 3D manipulation path smoothed with FM^2 inside a tube-like surface (right).

dimensional path π^*_{adap} is calculated from X_o with the retradted arm configuration. The loop is started and the path is tracked. For this experiment, π^*_{adap} is entirely tracked and converted into a high dimensional space in one execution of the loop. In Figure 5.42, the result of the algorithm for this experiment in the simulation environment can be appreciated. The obtained path for this experiment is 146.72% shorter and 14.22% smoother than the path obtained with Algorithm 19 under the same conditions.

Two adaptive dimensionality algorithms for path planning were presented in Section 4.2. In order to test their performance, some experiments were conducted for both algorithms in this section. The first method (Algorithm 19) achieves to improve paths generated with RRT-Bidirectional in terms of smoothness and length. The second one (Algorithm 20) generates shorter smooth paths by starting with a low-dimensional FM^2 path. The approaches are able to iteratively convert the low-dimensional calculations into high-dimensional executable paths. The clearance of paths is considered in both approaches through the velocity potential map used by the FM^2 method. The proposed algorithms, while working back and forth between high and low-dimensional spaces, manage to quickly calculate feasible manipulation paths.

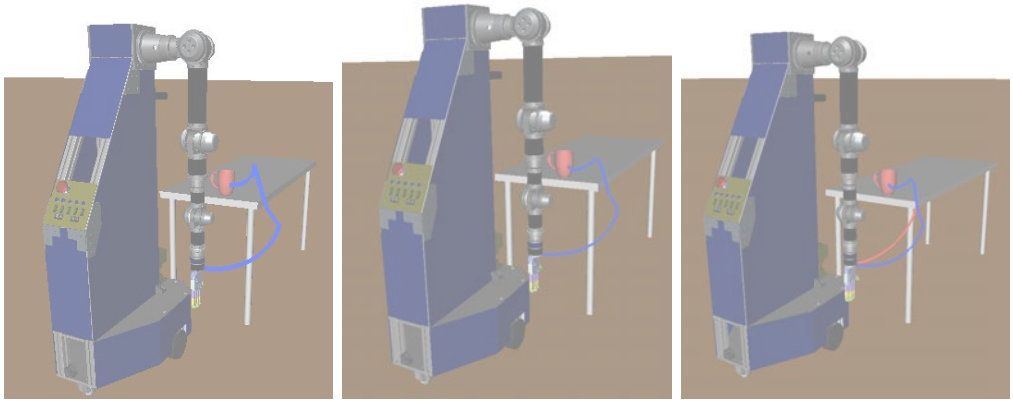


Figure 5.40: First experiment with Algorithm 19. The initial RRT-Bidirectional path (left), the adaptive FM^2 approach (middle), and the executed path(middle).

5.6 Application of the FM^2 Method in a Nuclear Fusion Device

We have conducted experiments in a simulated environment taking into account the physics of articulated boom and the JET torus.

A first experiment is proposed starting from initial position (all links at 0) and ending in the upper left side, Figure 4.3. The global path generated for the 2D torus map with the wave propagation potential values, is as presented in Figures 5.44.

It takes 0.2943 s for the algorithm to generate the path shown in Figure 4.3. In addition it takes 1.742 s to compute the forward kinematics for the boom. Further experiments are carried out using an RRT-connect algorithm to compare with the proposed method. Results are presented in the Table 5.7.

Table 5.7: Performances obtained when planning with RRT.

Time	Points Number	Opt. time	Opts. points
63.32	4226	8.38	69
56.84	4226	6.47	69
52.35	4226	6.83	69
54.30	4226	6.78	69
52.27	4226	6.82	69

The path generation average time for the RRT planner for this task was 55.82

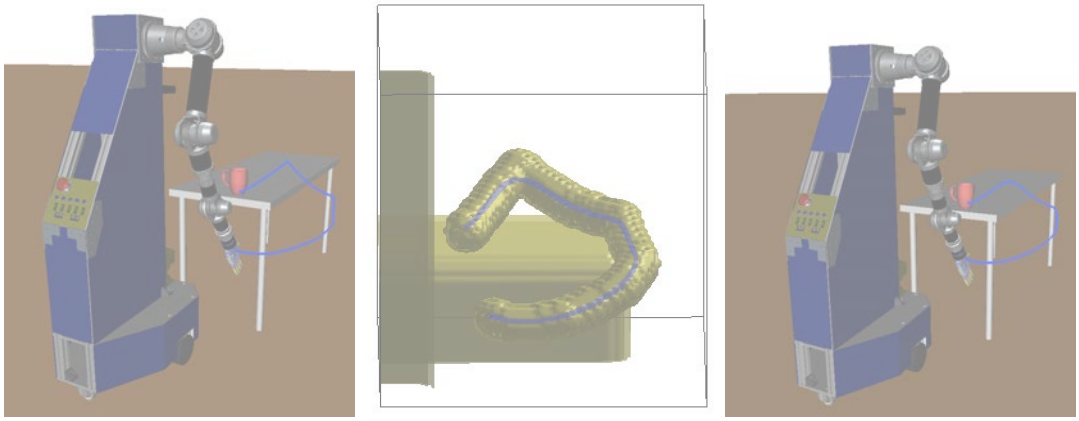


Figure 5.41: Second experiment with Algorithm 19. The initial RRT-Bidirectional path (left), the tube-like surface with the FM^2 path inside (middle), and the executed path (right).

s and the optimization average time was 7.06 s. The RRT takes considerably more time for generating a path than the FM^2 method, additionally the generated paths get dangerously close to obstacles even after optimization is performed. Furthermore it was found that when the size of the entrance of the torus was reduced to its actual width, RRT either took a very long time to generate a path (around 361s) or could not find a solution.

The approach presented here provides smooth trajectories that can be used at low control levels without any additional smooth interpolation processing, that can be generated in real-time, and that can maintain distance as far as possible from obstacles. Figure 5.45 presents a screen capture of the simulation environment where the generated path is correctly executed.

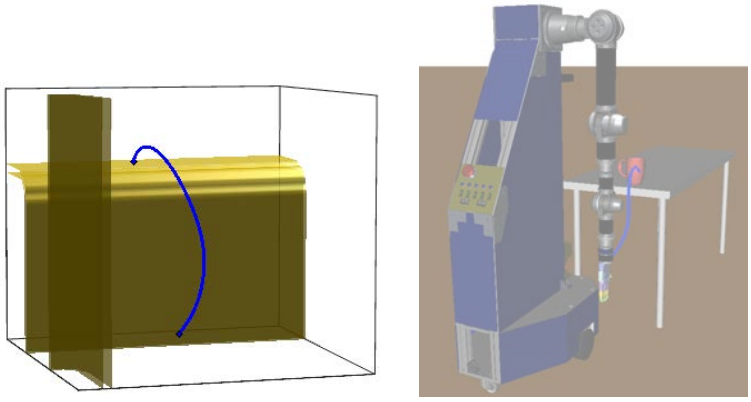


Figure 5.42: First experiment with Algorithm 20. The adaptive FM^2 path (left), and the executed path in the simulation environment (right).

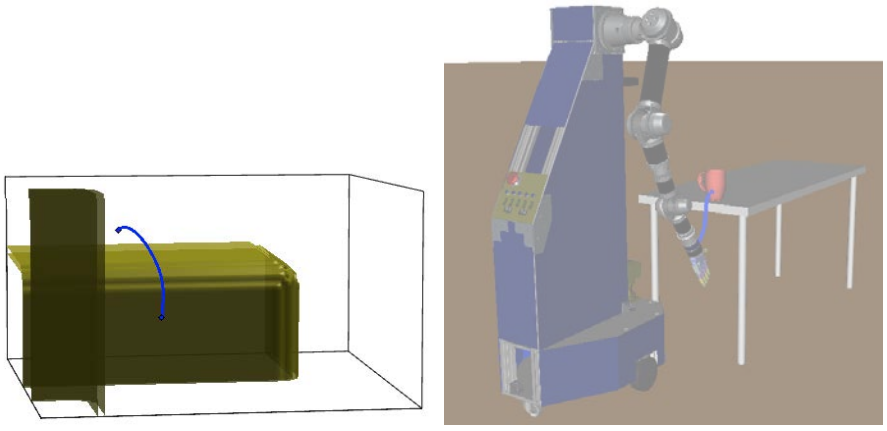


Figure 5.43: Second experiment with Algorithm 20. The adaptive FM^2 path (left), and the executed path in the simulation environment (right).

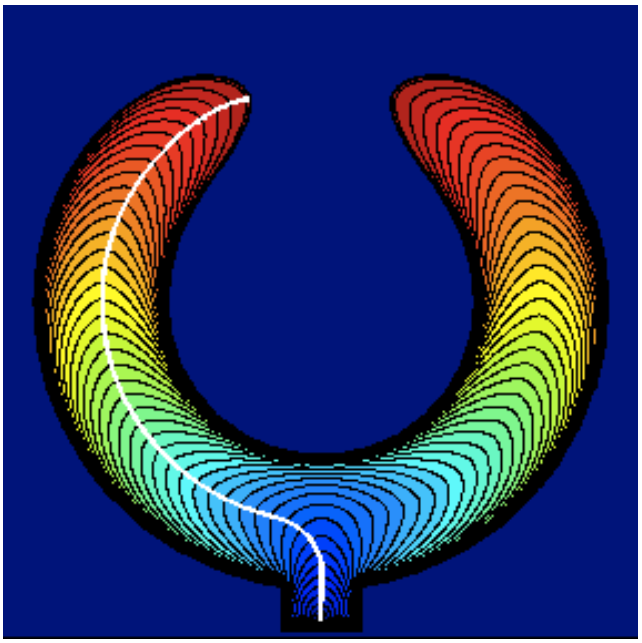


Figure 5.44: FM^2 map wave generation.

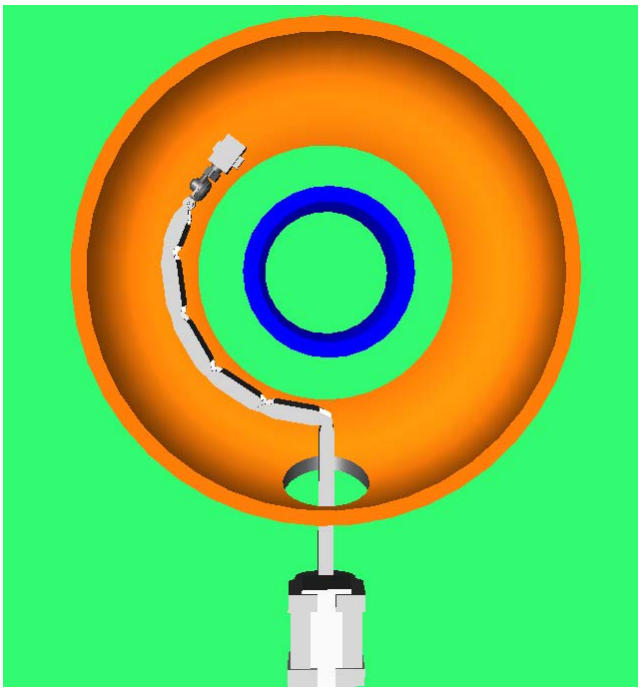


Figure 5.45: Goal position for JET's articulated boom on the simulation environment.

Chapter 6

Conclusions and Future Developments

The objectives presented at the beginning of this work were accomplished with the proposed approaches. The different algorithms for path planning in mobile robots are the main contributions of this work. These algorithms solve the path planning problem for robotic navigation and manipulation. The most important conclusions regarding each method as well as the future works are presented in this chapter.

6.1 Robot Path Planning and Navigation

In this section, the most important conclusions related to the presented path planning methods for navigation are discussed.

6.1.1 Smooth Motion Replanning using Fast Marching Square

A replanning strategy with subgoals that relies on the FM^2 method was presented in this document. The indoor robot environment was considered to be partially known because a prior map, equivalent to the building plans. The essential mechanisms used in the method included the calculation of a velocity potential map with a saturation to reduce planner distances and the FM^2 method to plan the trajectory towards the goal. First a global path is generated using a prior map, then this full path is divided into equal parts by means of topological or geometric sub-goals constraints. The robot starts traversing the global path and updating the map with sensors data. If executing the global path is not possible due to new obstacles, the path is calculated to the next node and only the implicated chunk of global path is updated. The essence of the algorithm is similar to the anytime algorithms logic, but unlike common anytime planners, the proposed algorithm always generates optimal paths in terms of safety and smoothness until reaching the target location.

The obtained results show that the FM^2 method can be used as an replanning algorithm to obtain smooth and safe trajectories in unstructured environments. An strategy to improve computational time is proposed, the trajectory is recalculated to subgoals instead of to the last node every time. The low complexity of the algorithm allows to use it in real time. Furthermore, the algorithm can directly be used with raw sensor data to implement a sensor-based local-path-planning exploratory module.

6.1.2 Anytime Triangular Fast Marching Square Method and Paths Benchmarking

An anytime motion planner that relies on the FM^2 method over a triangular mesh has been presented in this document. Different types of polygons are used to generate the ROI that is built from the initial path calculated with the RRT algorithm. After

that, the initial trajectory can be iteratively refined with the ATFM2 technique. The experimental results show that the anytime approach achieves to improve the quality of the path in length, smoothness, and clearance.

In the field of benchmarking, it is becoming increasingly difficult to compare new planners because of the lack of a general benchmarking platform. A discussion of the most used benchmarking parameters and their corresponding calculation procedures is given. Different weaknesses were found in the traditional formulation of the benchmarking parameters when the method performance was analyzed. Therefore, some improvements to existing approaches are suggested here. The new equations have been tested in the experiments and the proposed formulation enables a better comparison and evaluation of the paths quality.

The introduction of the ROI drastically reduces the search space. Thus, the anytime algorithm can be run in real-time while the robot navigates through the environment.

Analyzing the path lengths, the new trajectories outperform the RRT initial paths in all cases. The shortest routes are obtained after two iterations, which lead us to conclude that the anytime algorithm is an appropriate technique to be applied when the objective is to optimize the robot's path.

Several conclusions can be drawn when the smoothness is studied. The smoothness of the RRT initial path is improved after one iteration. Besides, this property is also enhanced after two iterations when compared to the results after one iteration. Although the reliability range is negative after one iteration in one case, this parameter is significantly improved after two iterations.

Regarding the clearance, the best ratios are obtained with the traditional benchmarking parameter. However, the new equation produces a more reliable parameter. The best values for the new equation are slightly higher than one. In general, the safety range is better when the new method is used, which has a positive influence on the navigation safety.

Although the type of polygon has to be chosen depending on the objective of the planner, the configurations with hexagons are the most promising ones according to the experiments.

6.1.3 Fast Marching Square applied to Nonholonomic car-like robots

Two different nonholonomic path planning approaches based on the FM^2 are presented in this document. They are both able to generate trajectories for nonholonomic mobile robots with high quality, i.e., with smoothness and clearance properties.

Simulations and experiments evidenced how the C-space and the Control-based FM^2-NH outperforms the RRT-NH planner results. In the current study, comparing

the FM^2-NH methods showed that they generate considerable shorter paths in length, and trajectories are more secure and smooth. Due to its random nature, the RRT-NH planner exhibits several loops in trajectories which produce longer paths; on the other hand, the deterministic quality of the FM^2 method (inherited by our methods) not only produce more coherent paths without loops, but also guarantees the computation of a solution if it exists, a criteria that is not met by the RRT-NH and all probabilistic planners.

The computational complexity for the Fast Marching Method, as well as for its successor, the FM^2, is defined as linear O(n), where n is the number of grid points in the environment map. Since the proposed method is based on the later method, the FM^2-NH methods are also highly efficient with a linear run-time complexity of O(n).

The FM^2-NH make several noteworthy contributions to path planning. Nevertheless, the most remarkable is that the algorithms calculate not only good paths, but also they provide the control variables needed to execute these trajectories.

6.2 Manipulation Planning

In this section, the most important conclusions related to the presented manipulation planning methods are discussed.

6.2.1 Adaptive Evolving Strategy for Dextrous Robotic Manipulation

An adaptive robotic manipulation methodology built over ES has been presented. The adaptation of manipulation paths for mobile manipulators is possible in real-time, achieving optimal manipulation relative to position, orientation and energy consumption. Given a learned manipulation path a new one is calculated when robot base is in a different location from that of the learned path, minimal position and orientation end-effector errors are obtained within the evolutionary process. Calculus are simplified to reduce computational time. A forward kinematics function was implemented for reaching optimal computational time, it implied a significant time reduction for the execution of the algorithm. Granted that the algorithm needs no inverse kinematics, singularities are avoided and convergence is guaranteed.

The experimental results showed the ability to apply the algorithm in real-time for obtaining adapted manipulation paths, proving to be a feasible solution for mobile robots manipulation problems. A computational time improvement was obtained by first optimizing position until position error is minimized, and then orientation error is added to the objective function with a reduced mutation step length until termination criterion is fulfilled at the end of the process. In addition, the self-adjusting step length

parameter was shown to perform efficiently compared to that of the DE algorithm. It is advisable to prioritize minimizing factors according to tasks because there is a proportional relationship between time and optimization parameters.

6.2.2 Path Planning with Adaptive Dimensionality

A manipulation path planning method with adaptive dimensionality is presented in this work. The proposed algorithm generates an initial full-DOF path using the RRT-Bidirectional method, and builds a tube-like surface surrounding the end-effector path points. Then the FM^2 is applied to find a 3D path inside the generated surface. The robot's inverse kinematic is used to follow the previously generated FM^2 path. In case of not been able to find a free of collision full-DOF path with the IK, the method enlarges the surface around the conflicted path points. Then, a full-DOF planning is executed only in the problem segment. The approach improves the initial manipulation path in terms of smoothness, safety, and path length. Furthermore, by suppressing the initial path peaks, the generated paths are visually more human friendly.

6.2.3 Safe Motion Planning for a Nuclear Fusion Device Arm using Fast Marching Square

A planning strategy using a dimensionality reduction and the FM^2 method is presented in this work. The algorithm generates a velocity potential map, which is saturated with different values to control the relation between path length and clearance. A path is generated from the robot's initial posture to the goal position. First a 2D map is obtained by crosscutting the JET torus at the height of the articulated boom, which reduces the dimensionality of the problem. Then, the velocity potential map is generated and a 2D path is calculated using FM^2. Finally this full path is followed closely by all the articulated boom links by means of the inverse kinematics. The output is a fully executable trajectory which leads the robot towards its goal keeping a desired safety distance from obstacles. Unlike common planners, the proposed algorithm always generates safe and smooth paths until reaching the target location.

The obtained results show that the FM^2 can be used as a planning algorithm to obtain smooth and safe trajectories in an unstructured environment. A strategy to improve computational time is proposed, the dimensionality of the problem is reduced.

6.3 Future Developments

We are presently working on a number of extensions to our current work. The points suggested as future work include:

- Different polygons could be considered to experiment with the ATFM2.
- Regarding the C-space and the Control-based FM2-NH, future work includes combining both solutions and introducing some changes to make it possible to use it in dynamic environments.
- Other planners could be implemented to generate the initial path in the ATFM2. More properties could be analyzed and more experiments in different environments could be conducted. It would also be interesting to study the method after three or more iterations, testing the convergence of the algorithm.
- For the adaptive manipulation method. First, the generation of paths with the specification of only one goal configuration. Second, the use of efficient collision detection to avoid obstacles. This can be implemented as another optimization parameter, which when getting closer to obstacles will increase the error value.

Exploring these implementations, and conducting further analysis forms the basis of our future work.

Appendix A

Experimental Platform

The experimental platform MANFRED-2, fully developed at the Robotics Lab of the Carlos III University of Madrid, is presented in this appendix. Most of this information and figures have been taken from the work by Álvarez [75].

The mobile robot MANFRED-2 is a mobile manipulator whose purpose is to serve as experimental platform for R&D in the mobile robots area.

One of the main objectives of this research is to build an autonomous robot for an indoor office area. In other words, MANFRED-2 must be able to navigate autonomously in an environment typically composed of a corridor and offices. For example, one specific task that the robot must perform is to move from one room to another by opening a door.

This robot has been built because it is necessary to have an experimental platform with a robust and reliable hardware that allows researchers to focus on the real problem: the implementation of an artificial intelligence that allows the robot to be autonomous and perform multiple tasks.

The robot design is inspired by planetary rovers and communications satellites. These systems are composed of several subsystems that need to be interconnected to make the whole system work. These subsystems are: onboard computer, power distribution system, sensors, drive system, etc. More instruments to explore the surroundings, such as articulated arms, can also be implemented depending on its application, but it is necessary to distinguish between the mobile platform and the inserted accessories. One important characteristic is that the subsystems are designed as independent units or boxes that are interconnected to each other by an internal wiring.

Summarizing, the design of MANFRED-2 is based on independent units that are interconnected to each other by using electric and mechanical interfaces. This modular concept facilitates the integration, repair, and future expansion of the robot.

MANFRED-2 is presented in Figure A.1. It was also shown in Figure 1.3. It has at most eight DOF. It is composed of a differential-type mobile base with two DOF and an anthropomorphic light arm with six DOF. It can execute multiple tasks. The most typical ones are opening and passing through doors, obstacle avoidance, and picking up and manipulating objects. In order to do that, the robot needs all the basic capabilities to move safely and independently around the environment, motor coordination between the base and manipulator, and sensory coordination to manipulate objects.

As was previously said in this appendix, any robotic system consists of a set of subsystems that enable (through networking) meeting the objectives for which it was designed. These modules use the environment information to generate data that are used to develop the movement skills in the robot's base and the robotic arm. The main components of the systems that constitute the mobile manipulator are described in the following sections.

Figure A.1: MANFRED-2, mobile manipulator with robotic arm.

A.1 Mechanical Design - Robot Structure

The design of the mobile robot must meet the following specifications: high mobility, mechanical and electrical robustness, high repeatability in its movements, and easy integration and repairing (modular concept).

A brief description of the mechanical design of MANFRED-2 and a breakdown of the most important elements are given in this section. The mechanical design of the robot's base is also based on the robustness and reliability that must satisfy the robot when it is performing a task. It is crucial that the the robot movement does not cause instability or inaccuracy.

The base has also been designed following a modular philosophy which has two

important advantages: it is easy to access to all elements of the mobile robot, and the change of elements due to repairs or improvements is immediate.

The general design also focuses on the improvement of the structure rigidity. The force distribution is more balanced than the distribution of the previous version (MANFRED). The location of the base elements has been optimized in order to counterbalance when the robotic arm is executing critical tasks, which means that the distribution of the elements in the robot's base gives stability to the mobile robot.

Some mechanical characteristics and their associated advantages are given below. Some of them are compared to the previous version of the mobile robot.

- When the arm is at rest, it does not collide neither interfere with the base. If the system runs out of power, the arm can fall freely without damage to itself or to the base.
- The gravity center of the base has been moved closer to the ground. This implies an improvement in the stability.
- The main mast has been extended to the bottom plate and more columns have been placed between the plates. These changes give more rigidity to the system.
- It has independent carcasses that are easy to remove and place. It is easier to access to any component of the mobile robot.
- An internal communication system from the mast to the bottom plate has been designed. This system is simple and facilitates the changes or incorporation of new elements.
- All switches, buttons, and safety mushrooms are located in a single panel. This allows an easy and fast access to each element of the control and security systems.
- A second robotic arm that will be added to the robot has been taken into account, trying to make its future implementation as simple as possible.
- A height adjustment system for the drive wheels has been designed. This allows an accurate calibration.

The robot's weight and the weight of each one of its components are shown in Table A.1. It is important to remark that most of the weight is concentrated in the bottom part, which benefits the stability.

MANFRED-2 is formed by a metal structure that can integrate all the components needed for operation (Figure A.2). It can be divided into three parts:

Table A.1: MANFRED-2's weight.

Element	Unit weight (kg)	Total weight (kg)
Batteries	15.40	61.60
Aluminum structure	29.00	29.00
Drivers	0.68	5.44
Computer	5.00	5.00
Electronic devices	2.75	2.75
DC-DC Converters	2.00	2.00
Carcasses	2.50	2.50
Caster wheels	0.42	1.26
Drive wheels	7.00	14.00
Wiring	6.00	6.00
Total		129.55

Figure A.2: MANFRED-2, lateral view.

- Mobile base:

  The robot's base is composed of two steel platforms with a diameter of 61 cm

Figure A.3: Power supply system.

and a height around 65 cm. It is equipped with wheels that allow movement. The battery system that generates the power to operate autonomously is also stored in the base.

The motion system is included in the base. It has five wheels: three of them are support wheels to improve the stability and facilitate the movement, and the other two are drive wheels with brushless motors and their corresponding servo-amplifiers. The drive wheels generate a differential displacement that allows the robot to turn around its axis.

The power supply system that gives autonomy to the robot consists of batteries that are located in the base. There are four batteries of 12 V connected in series that provide a voltage of 48 V. The selected batteries are Power-Sonic PS-12450 B (Figure A.3), which provide an output voltage of 12 V and a capacity of 45 Ah.

In addition, as a security system, the robot has a monitoring system through a PIC16F818 microcontroller that measures the voltage provided by the batteries and the current flowing through them. This system can continuously communicate the power status to the control computer, as well as stopping the motors in a controlled way in case of low voltage or too high current.

- Body:

An structure that forms the robot body and holds multiple components has been mounted on the base. The body contains all the wiring for connecting several subsystems: arm to computer, power from battery to motors, and external sensors. It has also the servo amplifiers associated with the arm.

This structure serves as dock for the robotic arm, the laser sensor, and the computer vision cameras. The onboard computer that is responsible for add intelligence to the robot is also inside this part of the robot. This computer has the PMAC2-PCI card installed, which is a controller card that can control jointly the eight DOF corresponding to the base and the manipulator arm.

- Robotic arm:

  The manipulator arm LWR-UC3M-1 is an essential element of the robot. It is composed of rigid elements connected by revolution joints. Each joint gives an additional DOF to the robot. The total number of DOF is six for the arm. It has been designed to provide a remarkable flexibility to perform manipulation tasks (grasping and and movement of objects) by combining the available DOF.

  The robotic arm that is presented in Figure A.4 has been fully developed by the Robotics Lab of the Carlos III University of Madrid. Its main characteristics are:

  1. Kinematic redundancy similar to the human arm.
  2. Weight: 18 kg.
  3. Maximum load capacity: 4.5 kg at the end of the arm.
  4. Load/weight ratio: between 1 : 3 and 1 : 4.
  5. Range: around 955 mm.

  The developed arm is mounted on the lateral side of the mobile robot in such a way that the computer vision and the laser telemetry systems are not obstructed by the arm. The arm joints are composed of DC brushless motors and Harmonic Drives that reduce the speed and increase the torque.

  Since the installed encoders obtain relative information (they provide information about the motor current position with respect to an initial or home position), an initial *home* function must be executed in order to fix the robotic arm initial position. This facilitates the conversion between relative and absolute positions. This function has been designed using the programming language of the PMAC2-PCI. It establishes that the initial position of the robotic arm is that one in which it is pointing straight to the ground. This position has been chosen because it requires a low energy consumption because most of the engines are not doing any work.

Figure A.4: LWR-UC3M-1(robotic arm).

Figure A.5: Hokuyo UTM-30LX (laser range finder).

A.2 Sensory System

The sensory system can transform the physical variables that characterize the environment into a data set that will be processed by other modules, such as the localization system, the security system, and the motion planner, in order to increase the robot intelligence and be able to execute certain tasks. This information will be provided by the robot's sensory system, which consists of the following elements:

- Laser telemetry subsystem:

  Its aim is to provide the robot with information about its surrounding environment by measuring the distance to objects. This information is primarily used in navigation and localization in order to model the workspace.

  It is possible to use 2D or 3D data depending on the task characteristics and the complexity and degree of occupancy of the workspace.

  This subsystem is composed of the following laser range finders:

  1. Hokuyo UTM-30LX with 270 opening degrees (Figure A.5) located in the rear of the vehicle. It has a detection range that varies from 100 mm to 30 m and a 25 ms period. Its angular resolution is equal to 0.25°. It is connected to the computer through a USB2.0 interface. Its power consumption is 700 mA and 12 V, which makes it suitable for battery-powered systems such as MANFRED-2.
  2. SICK PLS with 180 opening degrees (Figure A.6). The original measurements are 2D, but we have added a motor that lets it rotate up and down

Figure A.6: SICK PLS (laser range finder).

Table A.2: SICK PLS technical characteristics.

Maximum range	80 m
Angular resolution	0.25° - 0.5° - 1° (variable)
Time response	26 ms
Distance resolution	10 mm
Transfer rate	500 kbaud
Power requirements	24 V - 6 A

(±45°), being able to obtain 3D measurements (it can also be observed in the figure). The technical characteristics are summarized in Table A.2.

The 2D telemetry (horizontal plane parallel to the ground) can be used during navigation around environments with few obstacles to safe computational time.

This sensor records 361 measurements in a planar sweep with medium resolution (separation between measurements equal to 0.5°). The SICK PLS measurement error is lower than 20 mm. This error is influenced by two parameters: the measuring distance and the angle of the laser beam shot (from 0° to 180°).

- Computer vision subsystem:

  This subsystem helps in the manipulation of objects in 3D environments, which is one of the abilities of MANFRED-2. In order to do this, it is necessary to recognize the object to be manipulated, estimate its position and orientation

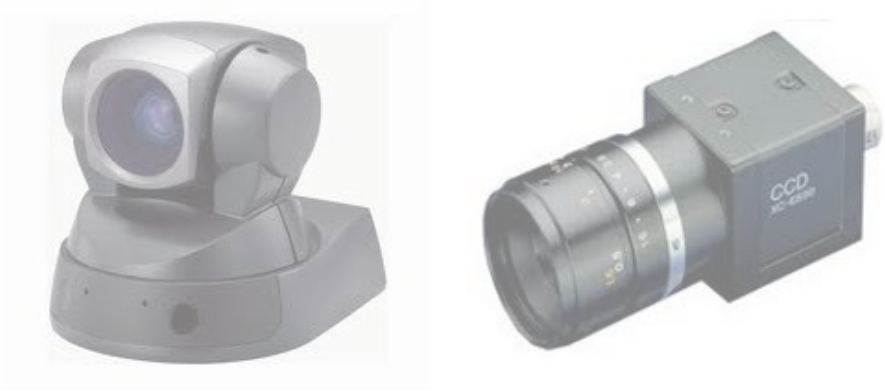


Figure A.7: Color cameras. Left: SONY EVI-D100. Right: SONY B/N XC-ES50CE.

relative to the mobile manipulator, and determine the grasping point. It also facilitates other tasks, such as opening doors, navigation, and localization.

The computer vision subsystem is composed of the following elements:

1. Color camera: SONY EVI-D100 (Figure A.7). This camera is employed to recognize objects and estimate their positions relative to the robot. It is located in the front of the mobile robot body.
2. Color camera: SONY B/N XC-ES50CE (Figure A.7). This is a mini-camera that is situated on the wrist of the robotic arm. It is used in manipulation tasks when the extreme of the arm is close to the object to be manipulated and the field of vision of the other camera is obstructed by the arm.
3. Time-of-flight camera (Kinect): the robot also incorporates a camera with time-of-flight technology (Figure A.8) that obtains a 3D image composed of an array of distances to different objects and color information. This information can be fused with the data of the other cameras in order to improve the manipulation capabilities.

- Force/torque sensor:

  MANFRED-2 has a JR3 force/torque sensor (model 67M25A-U560, Figure A.9) at the end of the robotic arm. Its purpose is to interact with the environment in manipulation tasks. This sensor is situated between the end of the arm and the clamp or terminal element.

  This device has the following features:

Figure A.8: Time-of-flight camera: Kinect.

Figure A.9: JR3 67M25A-U560 (force/torque sensor).

- Maximum load capacity: 11 kg.
- Weight: 175 gr.
- Maximum operating frequency: 8 kHz.

The JR3 sensor provides force and torque data in three axes that can be used in the force control loop of the mobile manipulator. It is based on a strain gauge system and a Digital Signal Processor (DSP) acquisition system that allow measurements with high bandwidth and signal-noise ratio. The main purpose of this sensor is to perform manipulation tasks based on force or torque control, such as opening doors, pulsation of switches, manipulating objects, etc.

- Motion sensors:

  The main function of these sensors is to obtain information about the robot location and the arm posture. This information is obtained by encoders that are mainly coupled to the rotation axes of the motors. The relative or absolute position of each motor is computed by using this information. The motion sensors are high-resolution optical encoders of the HP company with reference HEDS550.

  These motion sensors are complemented by inductive sensors that perform an

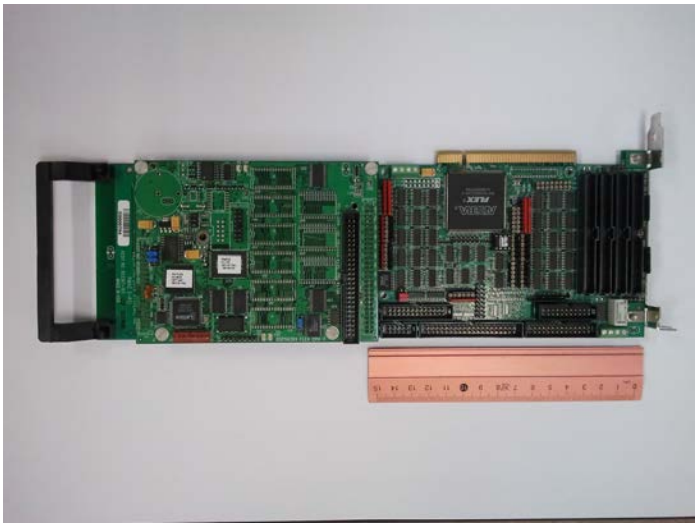


Figure A.10: PMAC2-PCI (controller card).

initial routine that is usually named as *home* in order to establish the absolute position of each joint of the arm. This routine improves the safety and minimizes the power consumption. The inductive sensors have a diameter equal to 3 mm and a detection distance equal to 1 mm. Their basic principle is based on the inductive detection of ferromagnetic materials by flux variation caused by their presence near the sensor's detection area.

A.3 Control System

MANFRED-2 has eight different motors to move its base (2) and its robotic arm (6). It is necessary to have a continuous control of these engines when the robot is navigating or it is moving its arm. This control is carried out by the PMAC2-PCI controller card (Figure A.10).

The PMAC2-PCI is a Programmable Multi-Axis Controller card developed by Delta Tau Data Systems[4]. It is a high performance device that can simultaneously control up to eight axes with high precision. It has a high performance/price ratio, with more than 1000 configuration variables and the high computing capacity of its DSP. The DSP that is incorporated in the PMAC2-PCI is the DSP56002 of 24 bits and operation frequency of 40 MHz.

[4] `http://www.deltatau.com`

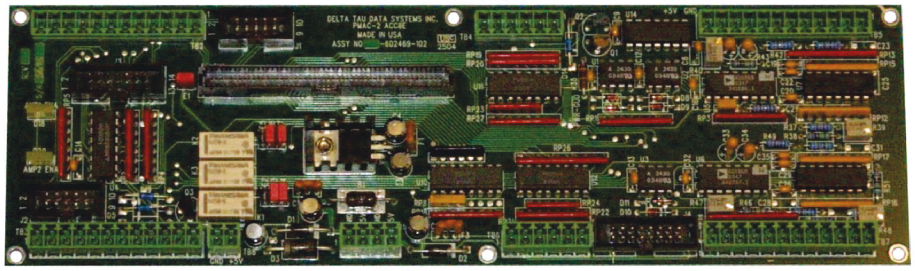


Figure A.11: ACC-8E. Interface between the PMAC2-PCI and the devices.

This card offers multiple ways to control the motors. However, it has not been designed to be connected directly to the devices. There is a set of additional cards that can be used as interfaces . These cards are also offered by Delta Tau Data Systems.

In the case of MANFRED-2, the additional card is the ACC-8E (Figure A.11). Since each card can interact with two motors, it is necessary to implement four of them. Each ACC-8E card is connected to the PMAC2-PCI through a 100-pin bus that is called JMACH. Each ACC-8E card has four 18-bit Digital-to-Analog Converters (DAC) that command two analog input drivers and must be fed with 15 V. It has also two inputs to read the encoders and five inputs per axis that capture different types of events: error signal, *home* signal (starting position), motor limits (two signals), and user-defined signal (external events for a specific application).

The configuration of the PMAC2-PCI is a very laborious and tough task. There are two available manuals, the "Software reference manual" and the "PMAC2 user manual", together with a program provided by the manufacturer, the "PEWIN32 PRO", which runs under Windows. This software offers a set of tools to modify all the configuration parameters of the PMAC2-PCI. Some of these tools are:

1. Terminal: it sends commands to the card in ASCII coding.

2. Watch window: it is a window where it is possible to view the variable values in real time.

3. Tunning Pro: it configures the PMAC2-PCI parameters, such as: PID controllers, filters, DAC calibration, and so on.

4. Position window: it displays the position of the motors, in counts of encoder, and also their speed and tracking errors.

Finally, the controller card allows different types of programs:

- Motion programs: the most common task of the controller card is to move the motors according to a particular sequence of commands. These programs are executed line by line by the controller card. They are called by a specific command and, after that, they are executed once. It is possible to make a call to another program or terminal commands. The controller card can store and execute up to 256 motion programs.

- Programmable Logic Controller (PLC): the PLC programs exist because there are some programs that must be executed continuously. For example, there is a PLC program that computes the robot's position given the encoders information. These programs are written in the same way that the motion programs, except that they are defined as PLC in their title. They are called and executed in each cycle of the controller card.

- Motion commands: it is possible to send motion commands to the PMAC2-PCI through the terminal. They are simple commands that allow the motion of each motor. These commands were initially implemented to test the controller card, but they can perform simple movements in a motion program.

A.4 Software

A.4.1 MATLAB

MATLAB (abbreviation of MATrix LABoratory)[5] is a numerical computing environment developed by MathWorks. It is oriented to projects that imply high computation resources and graphical display. It allows multiple actions, such as: manipulation of matrix and vectors, handling and plotting of functions and data, implementation of algorithms, creation of graphical interfaces, and interfacing with programs in other languages (C, C++, Java, and Fortran).

One additional advantage of this tool is that it is very easy to learn, not being necessary to study a new language because the solutions are expressed by an easy syntax (similar to C).

MATLAB includes a wide range of pre-built functions called "toolboxes". These toolboxes perform multiple operations of multiple areas of engineering and simulation, such as: signal processing, control, statistics, financial analysis, symbolic mathematics, neural networks, fuzzy logic, system identification, dynamic systems simulation, and so on. An additional package called "Simulink" offers a graphical interface for these toolboxes. It allows the simulation of dynamic models.

[5]More information can be found in `http://www.mathworks.es/products/matlab/`.

This tool is widespread in engineering, science, and economics. It has been reported that it had around one million users in 2004. It is also widely used in academic and research institutions.

All these features make MATLAB a suitable tool to be used for our purposes. All the algorithms developed in this work have been implemented in MATLAB.

A.4.2 OpenRAVE

[6]

OpenRAVE was founded by Rosen Diankov at the Quality of Life Technology Center in the Carnegie Mellon University Robotics Institute. It was inspired from the RAVE simulator James Kuffner had started developing in 1995 and used for his experiments ever since. The OpenRAVE project was started in 2006 and is a complete rewrite of RAVE. It is actively being maintained at the JSK Lab at University of Tokyo.

OpenRAVE provides an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications. The main focus is on simulation and analysis of kinematic and geometric information related to motion planning. OpenRAVE's stand-alone nature allows is to be easily integrated into existing robotics systems. It provides many command line tools to work with robots and planners, and the run-time core is small enough to be used inside controllers and bigger frameworks. An important target application is industrial robotics automation. OpenRAVE includes a seamless integration of simulation, visualization, planning, scripting and control. The plugin architecture allows users to easily write custom controllers or extend functionality. Using OpenRAVE plugins, any planning algorithm, robot control, or sensing-based subsystem can be distributed and dynamically loaded at run-time; this distributed nature frees developers from struggling with monolithic code-bases. Users of OpenRAVE can concentrate on the development of planning and scripting aspects of a problem without having to explicitly manage the details of robot kinematics and dynamics, collision detection, world updates, and robot control. OpenRAVE provides a powerful Python API for scripting demos, which makes it simple to control and monitor the demo and environment state. There are also interfaces for Octave and Matlab.

OpenRAVE's major design goals and features are:

Have a plugin-based architecture that allows users to expand its functionality without having to recompile the base code. Most functionality should be offered as plugins, thus keeping the core as simple as possible. Offer many motion planning algorithm implementations that can be easily extended to new tasks. Make it easy to debug components during run-time without having to recompile or restart the entire

[6] More information can be found in `http://openrave.org/docs/latest_stable/`.

system in order to prevent flushing of the in-memory environment state. Allow the OpenRAVE core to be used as a simulation environment, as a high-level scripting environment, as a kinematics and dynamics backend module for robot con- trollers, or as a manipulation planning black box in a distributed robotics environment. Allow simple planning knowledgebases to be generated, stored, and retried. Support a multi-threaded environment and allow easy parallelization of planners and other functions with minimal synchronization required on the user side. One of OpenRAVE's strongest points when compared with other planning packages is the idea of being able to apply algorithms to any scenario with very little modification when robots or target objects change. Users of OpenRAVE can concentrate on the development of planning and scripting aspects of a problem without having to explicitly manage the details of robot kinematics, dynamics, collision detection, world updates, sensor modeling, and robot control.

OpenRAVE has been used for planning on many real robotics systems. Its architcture makes it possible for planning-enabled robots to work consistently in a continuously changing and unpredictable environment. Many new layer of functionality have been developed that go beyond the basic kinematics, collision detection, and graphics interface requirements of classic robotics libraries. It provides a set of interfaces that let users modify existing functions and expand OpenRAVE-enabled modules without having to recompile OpenRAVE or deal with messy monolithic code-bases.

A.4.3 Marilou Robotics Studio

Marilou is a simulation software that includes dynamics and it's capable of modeling almost any kind of robot and environment. A variety of hardware, like sensors, actuators, cameras and lasers, are included. The simulator's physical entity editor is entirely graphical, facilitating robots creation. Robots are then easy to place in one or more simulations in order to test embedded algorithms. The key features of this software are presented next by areas:

Modeling

- Totally graphical handling of robots and environments models (physics parts and 3D models).
- Modeling helpers, Refactoring tools, several document viewpoints.
- Rigid bodies and n-axis joints.
- Mechanical constraints.

- Surface properties (reflection, shock, friction, incidence, rebound, behavior with infra-red or ultrasound and more).
- Hierarchy and complex assemblies.

Libraries

- Embedded robotic components: motors, servo motors, odometers, force/torque sensors, distance sensors (US, IR, Laser), laser range finders, Lidar, bumpers, actuating cylinders/jack, air pressure forces, cameras, panoramic spherical cameras, GPS, accelerometers/gyroscope, absolute compass and more.
- Off-the-shelf robotic equipment.
- Existing and virtual robots.
- Worlds.

Programming

- Robot programming using various languages (C/C++, VB#, J#, C#, C++ CLI and URBI) under Windows and Linux.
- Compatible with Matlab, Java and Intempora RT-Maps.
- Real/simulated compatibility layer on supported robots (allowing you to work with the same language and software tools as on real robots).
- English and French documentation.

Simulation

- Real-time or accelerated simulation (RT-Multiplier).
- Multi-robots.
- Multiple embedded applications, centralized or distributed.
- Acquisition/measurement cycles as low as 1 ms.
- Gravitational forces.
- 3D spatial sound.
- Interactions with running simulation.

A.4.4 ROS

ROS (Robot Operating System)[7] is an open code operating system for robots developed by Willow Garage. As it is said in its website, "it provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license (Berkeley Software Distribution, family of permissive free software licenses)".

ROS is based on a set of processes or nodes that are individually executed and linked by a communication infrastructure provided by ROS. This communication can be synchronous (client-server) or asynchronous (continuous data sending). The different data can be grouped into packages that are shared allowing a distributed collaboration.

The most remarkable characteristics are the following: light and easy to export (it has been exported to OpenRAVE, Orocos, and Player), programming language independent (it can be implemented in the most common languages, such as C++ and Python), easy error correction (because it has a testing unit), and appropriate in big systems with multiple modules.

It currently only works with Unix-based platforms. It has been extensively tested on Ubuntu (operating system of MANFRED-2).

ROS has been implemented in MANFRED-2. All modules developed for the robot must follow its guidelines.

[7] More information can be found in `http://www.ros.org/wiki/`.

Bibliography

[1] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, 1993.

[2] D. Blanco, S. A. Ansari, C. Castejón, B. López Boada, and L. E. Moreno, "MANFRED: Robot Antropomórfico De Servicio Fiable Y Seguro para operar En Entornos Humanos," *Revista Iberoamericana de Ingeniería Mecánica*, vol. 9, no. 3, pp. 33–48, 2005.

[3] S. Garrido, L. Moreno, M. Abderrahim, and D. Blanco, "FM2: A Real-time Sensor-based Feedback Controller for Mobile Robots," *International Journal of Robotics and Automation*, vol. 24, no. 1, pp. 48–65, 2009.

[4] J. A. 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.

[5] S. M. LaValle, "Motion Planning," *IEEE Robotics & Automation Magazine*, vol. 18, pp. 79–89, Mar. 2011.

[6] S. M. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, Institute of Electrical and Electronics Engineers, 1985.

[8] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3375–3382, IEEE, 1996.

[9] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, pp. 23–33, Mar. 1997.

[10] G. Feng, "A Survey on Analysis and Design of Model-Based Fuzzy Control Systems," *IEEE Transactions on Fuzzy Systems*, vol. 14, pp. 676–697, Oct. 2006.

[11] R.-E. Precup and H. Hellendoorn, "A survey on industrial applications of fuzzy control," *Computers in Industry*, vol. 62, pp. 213–226, Apr. 2011.

[12] H. Hagras, "A Hierarchical Type-2 Fuzzy Logic Control Architecture for Autonomous Mobile Robots," *IEEE Transactions on Fuzzy Systems*, vol. 12, pp. 524–539, Aug. 2004.

[13] M. Likhachev and D. Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," *The International Journal of Robotics Research*, vol. 28, pp. 933–945, June 2009.

[14] A. Kelly, "An Intelligent , Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem," *Engineering*, 1995.

[15] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt, "Map Learning and High-Speed Navigation in RHINO," 1998.

[16] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 341 – 346, IEEE, 1999.

[17] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 446–451, IEEE, 2003.

[18] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge: Research Articles," *Journal of Robotic Systems*, vol. 23, pp. 661–692, Sept. 2006.

[19] T. M. Howard and A. Kelly, "Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots," *The International Journal of Robotics Research*, vol. 26, pp. 141–166, Feb. 2007.

[20] C. Urmson, C. Ragusa, D. Ray, J. Anhalt, D. Bartz, T. Galatali, E. Gutierrez, J. Johnston, M. Clark, P. Koon, A. Mosher, and J. Struble, "A robust

approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics*, vol. 23, pp. 467 – 508, 2006.

[21] D. Braid, A. Broggi, and G. Schmiedel, "The TerraMax autonomous vehicle," *Journal of Field Robotics*, vol. 23, pp. 693–708, Sept. 2006.

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

[23] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[24] A. Stentz, "The Focussed D* Algorithm for Real-time Replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Aug. 1995.

[25] L. Kavraki, P. Svestka, J.-C. Latombe, and M. 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.

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

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

[28] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, IEEE, 2000.

[29] S. Garrido, L. Moreno, D. Blanco, and F. Martin, "FM2: a real Fast Marching sensor-based Motion Planner," in *2007 IEEE/ASME international conference on Advanced intelligent mechatronics*, pp. 1–6, 2007.

[30] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII* (G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey, eds.), pp. 449–464, Springer Berlin Heidelberg, 2009.

[31] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," in *Robotics: Science and Systems (RSS)*, (Zaragoza - Spain), 2010.

[32] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[33] L. Janson, A. Clark, and M. Pavone, "Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions," *arXiv preprint arXiv:1306.3532*, 2013.

[34] J. Vascak and M. Rutrich, "Path planning in dynamic environment using Fuzzy Cognitive Maps," in *2008 6th International Symposium on Applied Machine Intelligence and Informatics*, pp. 5–9, IEEE, Jan. 2008.

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

[36] J. Borenstein and Y. Koren, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 535–539, 1991.

[37] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 116–121, Institute of Electrical and Electronics Engineers, 1985.

[38] B. J. Kuipers and T. S. Levitt, "Navigation and mapping in large-scale space," *AI Magazine*, vol. 9, pp. 25 – 43, 1988.

[39] M. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 304–312, June 1992.

[40] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to algorithms*. The MIT Press, 3rd editio ed., 2009.

[41] R. Fikes and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, 1972.

[42] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc.,Reading, MA, 1984.

[43] B. Cohen, I. A. Sucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 589–595, IEEE, Oct. 2012.

[44] S. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 473–479, IEEE, 1999.

[45] Y. Li, "A multi-RRT based hierarchical path planning method," in *2012 IEEE 14th International Conference on Communication Technology*, pp. 971–975, IEEE, Nov. 2012.

[46] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," in *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988)*, pp. 49–54, 1988.

[47] I. Millington and J. Funge, *Artificial Intelligence for Games*. CRC Press, 2009.

[48] D. Ferguson and A. Stentz, "Anytime, dynamic planning in high-dimensional search spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1310–1315, 2007.

[49] S. Datoussaid, O. Verlinden, and C. Conti, "Application of Evolutionary Strategies to Optimal Design of Multibody Systems," *Multibody System Dynamics*, vol. 8, pp. 393–408, Nov. 2002.

[50] H.-P. Schwefel, *Numerical Optimization of Computer Models*. John Wiley & Sons Ltd, Aug. 1981.

[51] L. D. Cohen and R. Kimmel, "Global Minimum for Active Contour Models: A Minimal Path Approach," *International Journal of Computer Vision*, vol. 24, pp. 57–78, Aug. 1997.

[52] J. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1528–1538, 1995.

[53] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences*, vol. 95, pp. 8431–8435, July 1998.

[54] A. M. Bronstein, M. M. Bronstein, R. Kimmel, D. Gries, and F. B. Schneider, *Numerical Geometry of Non-Rigid Shapes*. Springer, 2007.

[55] S. Garrido and L. Moreno, "Voronoi diagram and fast marching applied to path planning," *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, no. May, pp. 3049–3054, 2006.

[56] J. Sethian and A. Vladimirsky, "Ordered upwind methods for static Hamilton–Jacobi equations: Theory and algorithms," *SIAM Journal on Numerical Analysis*, 2003.

[57] A. Valero-Gomez, J. V. Gomez, S. Garrido, and L. Moreno, "The Path to Efficiency: Fast Marching Method for Safer, More Efficient Mobile Robot Trajectories," *IEEE Robotics & Automation Magazine*, vol. 20, pp. 111–120, Dec. 2013.

[58] Chia Hsun Chiang, Po Jui Chiang, J.-C. Fei, and Jin Sin Liu, "A comparative study of implementing Fast Marching Method and A* SEARCH for mobile robot path planning in grid environment: Effect of map resolution," in *2007 IEEE Workshop on Advanced Robotics and Its Social Impacts*, pp. 1–6, IEEE, Dec. 2007.

[59] L. Yatziv, A. Bartesaghi, and G. Sapiro, "O(N) Implementation of the Fast Marching Algorithm," *Journal of Computational Physics*, vol. 212, no. 2, pp. 393–399, 2005.

[60] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1243–1248, IEEE, 2006.

[61] J. V. Gómez, D. Álvarez, S. Garrido, and L. Moreno, "Improving Sampling-based Path Planning Methods with Fast Marching," in *ROBOT2013: First Iberian Robotics Conference* (M. A. Armada, A. Sanfeliu, and M. Ferre, eds.), pp. 233–246, Springer International Publishing, 2014.

[62] S. Garrido, L. Moreno, and D. Blanco, "Smooth Path Planning for non-holonomic robots using Fast Marching," *Mechatronics, 2009. ICM*, vol. 00, no. April, 2009.

[63] C. González, D. Blanco, and L. Moreno, *Optimum robot manipulator path generation using Differential Evolution.* IEEE Congress on Evolutionary Computation, CEC, 2009.

[64] S. Tabandeh, C. Clark, and W. Melek, "A Genetic Algorithm Approach to solve for Multiple Solutions of Inverse Kinematics using Adaptive Niching and Clustering," *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, pp. 1815 – 1822, 2006.

[65] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series).* Springer, 2005.

[66] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3020–3025, IEEE, 1996.

[67] A. Stentz, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments," tech. rep., Robotics Institute, Carnegie Mellon University, Aug. 1993.

[68] K. Gochev, A. Safonova, and M. Likhachev, "Planning with adaptive dimensionality for mobile manipulation," *2012 IEEE International Conference on Robotics and Automation*, pp. 2944–2951, May 2012.

[69] C. Petres, Y. Pailhas, Y. Petillot, and D. Lane, "Underwater path planing using fast marching algorithms," in *Europe Oceans 2005*, vol. 2, pp. 814 – 819, IEEE, 2005.

[70] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International journal of computer vision*, 1997.

[71] I. Ulrich and J. Borenstein, "VFH*: local obstacle avoidance with look-ahead verification," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, pp. 2505–2511, IEEE, 2000.

[72] J. Minguez, L. Montano, T. Simeon, and R. Alami, "Global nearness diagram navigation (GND)," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 1, pp. 33–39, IEEE, 2001.

[73] J. Denavit and R. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 23, no. June, pp. 215 – 221, 1955.

[74] C. Arismendi, J. V. Gómez, S. Garrido, and L. Moreno, "Adaptive Evolution Strategy for Robotic Manipulation," in *2012 IEEE Conference on Evolving and Adaptive Intelligent Systems*, pp. 29–34, IEEE, May 2012.

[75] D. Álvarez, "Controlador cartesiano para el brazo LWR-UC3M-1 del robot MANFRED con detección de contacto," Master's thesis, Carlos III University of Madrid, 2011.