



**UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA  
EXPERIMENTAL YACHAY**

**Escuela de Matemáticas y Ciencias Computacionales**

**TÍTULO: Path Planning Based on Fast Marching Method**

Trabajo de integración curricular presentado como requisito para  
la obtención del título de Ingeniero en Tecnologías de la  
Información

**Autor:**

**Jhonatan David Saguay Saguy**

**Tutor:**

**Israel Pineda, PhD.**

Urcuquí, Enero 2022

**SECRETARÍA GENERAL**  
**(Vicerrectorado Académico/Cancillería)**  
**ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**  
**CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN**  
**ACTA DE DEFENSA No. UITEY-ITE-2022-00001-AD**

A los 19 días del mes de enero de 2022, a las 11:00 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

<b>Presidente Tribunal de Defensa</b>	Dr. ANTON CASTRO , FRANCESC , Ph.D.
<b>Miembro No Tutor</b>	Dr. MANZANILLA MORILLO, RAUL , Ph.D.
<b>Tutor</b>	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

El(la) señor(ita) estudiante **SAGUAY SAGUAY, JHONATAN DAVID**, con cédula de identidad No. **0604998047**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Path Planning based on Fast Marching Method** , previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

<b>Tutor</b>	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
--------------	--

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:


Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. MANZANILLA MORILLO, RAUL , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. ANTON CASTRO , FRANCESC , Ph.D.	9,0
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.	10,0

Lo que da un promedio de: **9.7 (Nueve punto Siete)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

<p>SAGUAY SAGUAY, JHONATAN DAVID  <b>Estudiante</b></p> <p>Signé électroniquement par  FRANCESC ANTON CASTRO  cn=FRANCESC ANTON CASTRO, c= EC  Date: 2022.01.20 11:56:59 ECT  Dr. ANTON CASTRO , FRANCESC , Ph.D.  <b>Presidente Tribunal de Defensa</b></p>	<p>JHONATAN  DAVID SAGUAY  SAGUAY</p>  <p>Firmado digitalmente por  JHONATAN DAVID  SAGUAY SAGUAY  Fecha: 2022.01.26  16:16:45 -05'00'</p>
--	---

<p>Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.  <b>Tutor</b></p>	 <p>Firmado electrónicamente por:  <b>ISRAEL  GUSTAVO  PINEDA ARIAS</b></p>
---	--

<p>Dr. MANZANILLA MORILLO, RAUL , Ph.D.  <b>Miembro No Tutor</b></p>	 <p>Firmado electrónicamente por:  <b>RAUL  MANZANILLA</b></p>
--	---

DAYSY  
MARGARITA  
MEDINA BRITO

Firmado digitalmente por  
DAYSY MARGARITA  
MEDINA BRITO  
Fecha: 2022.01.20 11:52:57  
-05'00'

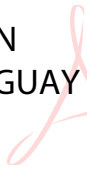
MEDINA BRITO, DAYSY MARGARITA  
**Secretario Ad-hoc**

# Autoría

Yo, **Jhonatan Saguary**, con cédula de identidad **0604998047**, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Enero del 2022.

**JHONATAN  
DAVID SAGUAY  
SAGUAY**



Firmado digitalmente  
por JHONATAN DAVID  
SAGUAY SAGUAY  
Fecha: 2022.01.26  
17:03:00 -05'00'

---

Jhonatan Saguary  
CI: 0604998047



# Autorización de publicación

Yo, **Jhonatan Saguay**, con cédula de identidad **0604998047**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urququí, Enero del 2022.

**JHONATAN  
DAVID SAGUAY  
SAGUAY**

Firmado digitalmente  
por JHONATAN DAVID  
SAGUAY SAGUAY  
Fecha: 2022.01.26  
17:03:27 -05'00'

---

Jhonatan Saguay  
CI: 0604998047



# Dedication

*“I dedicate this work to my whole family, mainly my parents and brother, my mother Luz Saguary for all the love, affection, advice and being who has largely forged the person I am now. To my father Douglas Saguary, who despite our differences advises me with his wise words, and shows me his affection in his own way. Finally, to my brother Bryan who instilled in me a taste for programming, I thank him for listening to me whenever I have needed to. Love you very much.”*





# Acknowledgments

I am deeply grateful to my family for giving me the opportunity to complete a university degree. To my friends, who became a family, who have accompanied me during this beautiful stay of my life. I will miss you very much, and I wish you all success in your lives.

I thank my tutor, Israel Pineda for all the understanding, guidance and for allowing me to work with him.



# Abstract

Path Planning has become a very active research area due to the advancement of information technology. Path Planning is very important in different areas; for example, tumor modeling, robotic navigation, seismological analysis, image processing, video games, etc. The use of these search techniques has shown that there are optimization and efficiency problems, both in their implementation and in their design. This is why many solutions and modifications of algorithms have been developed that have tried to solve these problems based on different types of storage, these alternatives have been proposed with two main objectives: to reduce its computational time and to improve its accuracy. Although several algorithms solve the route planning problem, the Fast Marching method has not been the most popular. FMM is a very popular algorithm to compute times-of-arrival maps. FMM is a special case of level set methods, it is able to provide a smooth path from one point to another. The Fast Marching method has been modified in different ways, both in its way of storing data and in its workflow to solve problems of computational complexity. Therefore, the first step is to describe all these alternatives that this method presents within a common framework and to compare the different methods, based on previously defined metrics to answer the research question: How to achieve optimal FMM performance applied to route planning? In the present work, we carry out an algorithm based on the FMM focused on Path Planning, and we reduce the execution time of the FMM algorithm.

The present work consists of six chapters. Chapter 2 presents the literature review of research related to Path Planning and the Fast Marching method. Here, the main concepts of the Eikonal equation in the road panorama and the main components of the Fast Marching method algorithm are covered. The next chapter explains the modification of the Fast Marching method and the use of a stack for the implementation of the algorithm. Chapter 4 presents the statement of the problem and the methodology of this research. Also, we explain in detail our proposal to modify the Fast Marching method in this chapter. In the last chapters, we present the results of this project, the simulation scenarios, and how the algorithms were evaluated.

The results of this research work were gratifying. For each evaluated scenario, the FMM modification presented in this work obtained the shortest execution time. Our FMM modification had an improvement of about 35% in the execution time compared with the classic FMM.

**Keywords:** Fast Marching Method, FMM, FMM modifications, OFM, Path Planning, Path Planing based on FMM.



# Resumen

Con el avance de la tecnología de la información, la planificación de rutas se ha convertido en un área de investigación muy activa. La planificación de rutas es muy importante en diferentes áreas; por ejemplo, modelado de tumores, navegación robótica, análisis sísmológico, procesamiento de imágenes, videojuegos, etc. El uso de estas técnicas de búsqueda ha demostrado que existen problemas de optimización y eficiencia, tanto en su implementación como en su diseño. Es por esto que actualmente se han desarrollado muchas soluciones y modificaciones de algoritmos que han intentado solucionar estos problemas en base a diferentes tipos de almacenamiento, estas alternativas se han propuesto con dos objetivos principales: reducir su tiempo computacional y mejorar su precisión. Aunque varios algoritmos resuelven el problema de planificación de rutas, el método *Fast Marching* no ha sido el más popular entre ellos. FMM es un algoritmo utilizado principalmente para calcular mapas de tiempos de llegada. FMM es un caso especial de métodos de ajuste de nivel, éste es capaz de proporcionar un camino suave de un punto a otro. El método *Fast Marching* se ha modificado de diferentes formas, tanto en su forma de almacenar datos como en su flujo de trabajo para resolver sus problemas de complejidad computacional. Por tanto, el primer paso es describir todas estas alternativas que presenta este método dentro de un marco común y comparar los diferentes métodos, con base en métricas previamente definidas para dar respuesta a la pregunta de investigación: ¿Cómo lograr un desempeño óptimo de FMM aplicado a la planificación de rutas?. En el presente trabajo realizamos un algoritmo basado en el FMM enfocado en planificación de rutas, y reducimos el tiempo de ejecución del algoritmo FMM.

El presente trabajo consta de seis capítulos. El Capítulo 2 presenta la revisión de la literatura de la investigación relacionada con la planificación de rutas y el método *Fast Marching*. Aquí se cubren los conceptos principales de la ecuación de Eikonal en la planificación de rutas y los componentes principales del método *Fast Marching*. El siguiente capítulo presenta información relacionada con la modificación del método *Fast Marching* y el uso de pilas en la implementación del algoritmo. El Capítulo 4 presenta el planteamiento del problema y la metodología utilizada en este proyecto de investigación. Además, nuestra propuesta para modificar el método *Fast Marching* se explica en detalle en este capítulo. En los últimos capítulos, presentamos los resultados de este proyecto, los escenarios de simulación y cómo se evaluaron todos algoritmos.

Los resultados de este trabajo de investigación fueron muy gratificantes. En todos los escenarios evaluados la modificación de FMM presentada en este trabajo obtuvo el menor tiempo de ejecución. Nuestra modificación de FMM tuvo una mejora en el tiempo de ejecución de alrededor del 35% con relación al FMM clásico.

**Palabras Clave:** *Fast Marching Method*, FMM, Modificaciones de FMM,

OFM, Planificación de rutas, Planificación de rutas basado en FMM.

# Contents

<b>Dedication</b>	<b>v</b>
<b>Acknowledgments</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>Resumen</b>	<b>xi</b>
<b>Contents</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 General Objective . . . . .	1
1.3 Specific Objectives . . . . .	1
1.4 Organization of the Work . . . . .	2
<b>2 Theoretical Framework</b>	<b>3</b>
2.1 Path Planning . . . . .	3
2.1.1 Definition . . . . .	3
2.1.2 Classification of the Path Planning algorithms . . . . .	4
2.2 Fast Marching Method . . . . .	6
2.2.1 Definition . . . . .	6
2.2.2 Eikonal Equation in Path Planning . . . . .	7
2.2.3 Phases of the Algorithm . . . . .	9
2.2.4 Pseudocode . . . . .	12
2.2.5 Computational Complexity . . . . .	13
<b>3 State of the Art</b>	<b>15</b>
3.1 Fast Marching Method Modifications . . . . .	15
3.1.1 Fibonacci Stack . . . . .	15
3.1.2 Binary Stack . . . . .	17
3.1.3 Simplified Fast Marching Method . . . . .	18



3.2	Modifications of $O(n)$ in Fast Marching Method Algorithm . . . . .	19
3.2.1	The Group Marching Method . . . . .	19
3.2.2	Fast Iterative Method . . . . .	22
<b>4</b>	<b>Methodology</b>	<b>23</b>
4.1	Phases of Problem Solving . . . . .	23
4.1.1	Description of the Problem . . . . .	23
4.1.2	Analysis of the Problem . . . . .	24
4.2	Proposed Algorithms . . . . .	24
4.2.1	Simple, Efficient, and Fast Sorting Method (SEF) . . . . .	24
4.2.2	Ordered Fast Marching Method (OFM) . . . . .	26
4.2.3	Computational Complexity of OFM . . . . .	26
4.2.4	SEF Sort Pseudocode . . . . .	27
4.3	Padding Metric . . . . .	28
4.4	Implementation of FMM . . . . .	29
<b>5</b>	<b>Results and Discussion</b>	<b>31</b>
5.1	Performance Evaluation . . . . .	31
5.1.1	Simulation Setup . . . . .	31
5.1.2	Experimental Environment . . . . .	32
5.1.3	Software Environment . . . . .	32
5.1.4	Scenarios Description . . . . .	32
5.2	Evaluation Metrics . . . . .	40
5.3	Simulation Results . . . . .	40
5.3.1	Image Size and Initial Point . . . . .	40
5.3.2	Image Obstacles and Padding . . . . .	44
<b>6</b>	<b>Conclusions</b>	<b>47</b>
6.1	Conclusions . . . . .	47
6.2	Future Works . . . . .	48
6.3	Limitations . . . . .	48
	<b>Bibliography</b>	<b>49</b>

# List of Figures

2.1	Classification of the current path planning approaches . . . . .	6
2.2	$T_1$ is the minimum between adjacent cells on the X axis . . . . .	8
2.3	$T_2$ is the minimum between adjacent cells on the Y axis . . . . .	9
2.4	Time of arrival (TT) calculation when the value of the Eikonal equation cannot be calculated . . . . .	9
2.5	Initialization stage of the FMM algorithm . . . . .	10
2.6	Iterative process of the FMM algorithm . . . . .	11
3.1	Graphical representation of the trees that form the Fibonacci stack [1] . . .	16
3.2	Binary tree representation and the equivalent array . . . . .	18
3.3	Process of inserting data ( $X = 3$ ), in a binary stack . . . . .	19
3.4	Root extraction process in a binary stack . . . . .	19
3.5	Wave expansion in GMM, white, black and gray points correspond to the states: Open, Frozen and Narrow respectively . . . . .	20
3.6	Angle between the direction of expansion of the wave (blue arrow) and the line that joins two cells (red line) . . . . .	21
4.1	Case 1, SEF sort method . . . . .	25
4.2	Case 2, SEF sort method . . . . .	25
4.3	Case 3, SEF sort method . . . . .	26
4.4	Route with and without padding metric . . . . .	29
4.5	Contour graphs for routes with and without padding metric . . . . .	29
5.1	2D Contour graph of the wave expansion for Scenario 1 . . . . .	33
5.2	3D Surface of the wave expansion for Scenario 1 . . . . .	34
5.3	2D Contour graph of the wave expansion for Scenario 2 . . . . .	34
5.4	3D Surface of the wave expansion for Scenario 2 . . . . .	35
5.5	2D Contour graph of the wave expansion for Scenario 3 . . . . .	35
5.6	3D Surface of the wave expansion for Scenario 3 . . . . .	36
5.7	Maze image used for Scenario 4 . . . . .	36
5.8	2D Contour graph of the wave expansion with respect to padding metric .	37
5.9	3D Surface of the wave expansion with respect to padding metric . . . . .	37
5.10	Maze image used for Scenario 5 . . . . .	38
5.11	2D Contour graph of the wave expansion with respect to padding metric .	39
5.12	Optimal route obtained in Scenario 5 . . . . .	39
5.13	Real simulation on the San Francisco map . . . . .	40

5.14 Execution time vs. Image size for Scenario 1 . . . . .	41
5.15 Execution time vs. Image size for Scenario 2 . . . . .	42
5.16 Execution time vs. Image size for Scenario 3 . . . . .	43
5.17 Execution time vs. Padding for Scenario 4 . . . . .	44
5.18 Execution time vs. Padding for Scenario 5 . . . . .	45

# Chapter 1

## Introduction

### 1.1 Motivation

Wave propagation algorithms are discussed in this research. This subject area is useful in many areas, such as mechanical fluid problems [2, 3] tumor modeling [4, 5], navigation robotics [6], seismic analysis [7], video games or image processing [8]. Different algorithms to find the solution to the path planning problem have been devised to have faster and more efficient results. This has sparked a wide variety of applications in different areas.

To carry out this project, a modification of the Fast Marching Method algorithm [9] has been implemented and four other existing ones have been used. The process consisted of experimenting with different configurations with each of the five algorithms and then obtaining, analyzing and comparing the results with each one of them.

The objective of this work is to implement and compare these algorithms in different circumstances. In this way, knowing which of them achieves a correct result and in the shortest time, thereby determining which is better in different situations.

The main motivation for carrying out this work is to implement an algorithm whose effectiveness and optimization is superior to the other algorithms or modifications implemented later. Consequently, this work presents a new tool for finding routes for this increasingly growing field of research.

### 1.2 General Objective

Propose an FMM optimization that achieves safe routes and reduces the computational time it takes for FMM to run.

### 1.3 Specific Objectives

- Evaluate several FMM variants with respect to how their computational time varies in the face of variables such as image size, wavefront origin, number of obstacles, and padding.

- Design a new variant of FMM that improves the performance of classic FMM.
- Compare our new variant against other variants of FMM, regarding its execution time.

## 1.4 Organization of the Work

- Chapter 2 presents the literature review of research related to Path Planning and the Fast Marching Method. Here is covered the main concepts of the Eikonal Equation in path panning, and the principal components of the Fast Marching Method algorithm.
- Chapter 3 explains the modification of the Fast Marching Method and the using of stacks for the implementation of the algorithm.
- Chapter 4 explains the methodology of this research project. Here is explained in detail our propose of Fast Marching Method modification, and its main concepts.
- Chapter 5 presents the results of this project, the simulation scenarios, and how the algorithms were evaluated.
- Chapter 6 expose the conclusions obtained in this work.

# Chapter 2

## Theoretical Framework

### 2.1 Path Planning

With the advancement in information technology, path planning has become a very active area of research. However, at present, there is still no algorithm capable of satisfying all the needs that this approach requires: reliability, completeness, low computational complexity, etc. [10, 9] The algorithms that have proven to be fast and with smooth solutions work correctly in two or three dimensions, but they give problems of complexity as the dimensionality increases. [11] Furthermore, fast algorithms for all dimensions provide paths that need a smoothing step later.

#### 2.1.1 Definition

Mathematics is a solid base for what is known today in the specific subject of path planning. Today, many solutions have been defined that converge to optimal and efficient theories that have been applied in different areas. These areas that have been benefited by the path planning algorithms are, for example, mobile robots, vehicles, video games, etc [12].

The focus of path planning has changed since the first approach was offered. The original goal was to create an algorithm that could reliably and completely find the path from the first point to the last point (i.e. the algorithm would find the path, if any). This goal is largely solved (i.e. Dijkstra's algorithm) and the computational power grows exponentially, so the goal is to find the shortest or faster path while still maintain security constraints. The new method should also provide a smooth and secure path [13].

Path planning is a purely geometric matter, since it is defined as the generation of geometric lines without reference to specific laws, but it is important to analyze the topological basis for the environment to be explored in each study case. [14]. That is, a short path is found by taking a fixed starting point to a goal target point. For example, if a point-to-point trajectory is considered (that is, only the start and final positions are determined), then two problems can be solved at the same time. The first problem is finding the shortest path and the second is optimizing any path previously passed.

The definition of a path planning problem is simple. “Find collision-free motion between the initial (start) and final (destination) configurations in a particular environment.” [9] The simplest situation is when you need to plan a route in a static and known environment. However, in general, trajectory planning problems can be built against any dynamically constrained robotic system in a dynamic and unknown environment.

## 2.1.2 Classification of the Path Planning algorithms

There are many and varied path planning algorithms. However, all these algorithms can be classified as follows:

### Geometric Methods

The environment is described as a set of polygons, and their properties calculate the path as a set of geometric primitives such as lines, arcs, and circles. The most common approach in this class is based on the visibility graph. [15, 16]. A visibility graph of a set of inseparable polygonal obstacles in a plane is an undirected graph in which the vertices are the vertices of the obstacles and the edges are pairs of vertices. For example, an open line segment between two vertices does not cut an obstacle. This approach is intuitive and provides an optimal 2D path (in terms of path length), but scaling to multiple dimensions is not easy and loses optimality; furthermore, becoming a complex and serious IT problem.

### Graph- and tree-based Methods

This is the most sought-after class in recent years. In this case, the environment is modeled by the state of the robot associated with the environment. A graph is created in which each node is the state of the robot (the states are attitude, speed, acceleration, etc). Transitions between states are modeled as costs and the selected path is the path that minimizes the total cost to reach the destination state from the current state.

There are several subclasses of the chart search method, depending on how the chart is generated and how costs are allocated. One of the possible classifications is:

- **Grid-based Methods:** It features customization of the grid space. The most common grid representation is with rectangular or triangular cells. This arbitrariness can lead to reduced accuracy, but this problem can be resolved by choosing the right cell size. Each cell in space is a node of the graph, connected to adjacent cells (or 8 cells connected in 2D, depending on the algorithm), and the cost of switching from one node to another can vary. It can be defined in the following way. If the cost is fixed, you can apply a graph search algorithm to select the allowed path to the destination at the lowest possible cost.

Within this group, the typical graph search algorithms can be found, such as Dijkstra [17] or A\* [10]. Modifications of these have been already proposed, such as D\* [18], which efficiently reuses information from previous steps to avoid redundant computations in dynamically changing scenarios or D\*-lite [19] which simplifies D\*.

Finally, wave front propagation methods, such as Fast Marching Method (FMM) [20] It is also part of this group, which is the subject of this treaty. In this case, the cost

per node is related to the time it takes for the wave to propagate and reach that node. Its formula ensures convergence towards the optimal path of the transit time when the mesh is refined. If the wave speed is constant, the path is also optimal in length.

- **Sampling-based Methods:** This type of algorithms incrementally searches in the space for a solution using a collision detection algorithm [21]. The most extended algorithms of this group are those based on rapidly exploring random trees (RRTs) [22]. The branches of these trees are randomly created from the initial point of the trajectory. Another common approach is the Probabilistic Road Maps (PRM) [23]. PRM creates a road map (set of connections) among a set of points randomly sampled. The main problem of sampling-based algorithms is their stochasticity. Most of the times the computed paths are far from the optimal one, are neither safe nor smooth. However, since RRT\* and PRM\* were proposed [24], this problem has vanished and thus this family of algorithms is the most widely used in practical applications. Finding faster approaches is a very active field nowadays, and many more asymptotically-optimal algorithms have been proposed, for instance BIT\*[25] and RRT# [26].

### Artificial Potential Field

Although these algorithms are based on grid representation, they are conceptually different from other grid-based methods. Artificial potential fields can be included in this group. Conceptually, the robot is represented as a point-like charge and the destination is represented as the opposite charge. [27]. Therefore, the robot is attracted to the target. Obstacles are simulated with the same markings as the robot, so obstacles are pushed back and collisions are avoided. Conceptually, they are simple and easy to implement, but the main drawback is that they suffer from a minimally fluctuating local path.



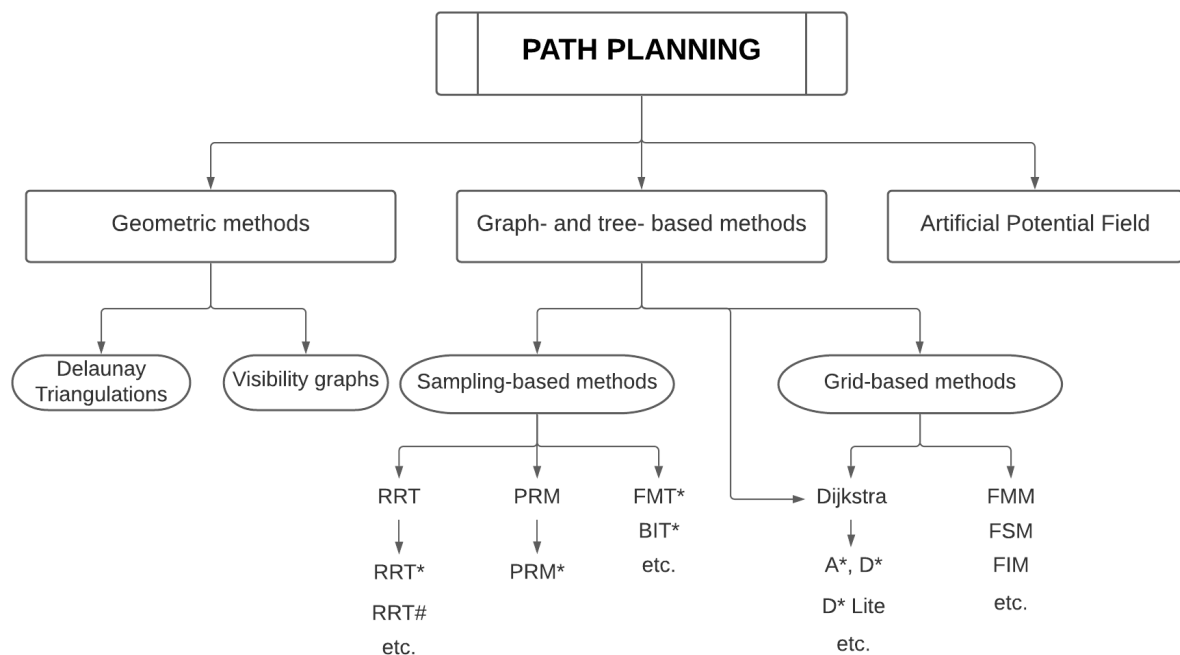


Figure 2.1: Classification of the current path planning approaches

Figure 2.1 summarizes this classification. It is important to remark that this classification might be considered not unique, as other criteria could be chosen: stochasticity, optimality, etc. For example, artificial potential methods are usually implemented using a grid discretization. A\* models the environment as a grid as well, but it is a graph search algorithm. In the same way, RRT and FMM could be in the same group considering that both work on a continuous space representation. However, this classification was chosen as it is the most common and the easiest to understand.

## 2.2 Fast Marching Method

### 2.2.1 Definition

The Fast Marching Method, FMM, is an algorithm developed by Osher and Sethian [9], which models the movement of a wave. It is a special case of the “Level Set Method”, a technique that is used to delineate interfaces and forms.

FMM is similar to the “Dijkstra Algorithm” which is based on the fact that the wave only travels out of the point where the wave started. The generated wave has the ability to be the shortest path, temporarily speaking, from where it started to the destination.

To easily understand FMM, [28] it is enough to imagine the effect of throwing a stone on a water surface at rest. When the stone is in contact with the surface of the water at a point, a circular waveform begins to expand outward from that point.

FMM calculates how long it takes for the generated wave to arrive from the initial point to any point in space, the initial point or points being those associated with a zero time [29].

The speed of propagation in the medium of movement of the wave does not have to be always the same, which means that the shortest path does not necessarily have to be the fastest.

## 2.2.2 Eikonal Equation in Path Planning

Using the Eikonal equation, it is possible to determine the movement of the wave in space, and the algorithm is used to advance the front of the generated wave [30].

$$\frac{1}{F(x)} = |\nabla T(x)|. \quad (2.1)$$

In the equation 2.1,  $x$  is the position of a point,  $F(x)$  the speed at which a wave can travel at that point, and  $T(x)$  is the time it takes for the wave to reach that position. To solve the equation, Osher and Sethian [9] propose the use of a grating to discretize the space where the wave propagates. After the discretization and subsequent simplification of the gradient  $\nabla T$ , according to Sethian, the rows and columns being named as  $i$  and  $j$  of the grid respectively, leads to:

$$\max\left(\frac{T_{ij} - \min(T_{i-1,j}, T_{i+1,j})}{\Delta x}, 0\right)^2 + \max\left(\frac{T_{ij} - \min(T_{i,j-1}, T_{i,j+1})}{\Delta y}, 0\right)^2 = \frac{1}{F_{ij}^2}.$$

Let be  $\Delta x$  and  $\Delta y$  the measure of the grating in direction  $x$  and  $y$ , assuming positive wave velocity  $F > 0$ , then  $T_{ij}$  it is always greater than  $\min(T_{i-1,j}, T_{i+1,j})$  and  $\min(T_{i,j-1}, T_{i,j+1})$ . So it can finally be solved for two dimensions:

$$\max\left(\frac{T_{ij} - \min(T_{i-1,j}, T_{i+1,j})}{\Delta x}\right)^2 + \max\left(\frac{T_{ij} - \min(T_{i,j-1}, T_{i,j+1})}{\Delta y}\right)^2 = \frac{1}{F_{ij}^2}.$$

Square grids have been established for the implementation of the algorithms:

$$\Delta x = \Delta y$$

To simplify:

$$\begin{aligned} T1 &= \min(T_{i-1,j}, T_{i+1,j}), \\ T2 &= \min(T_{i,j-1}, T_{i,j+1}). \end{aligned}$$

So:

$$\begin{aligned}(T - T_1)^2 - (T - T_2)^2 &= 1, \\ T^2 - 2TT_1 + T_1^2 + T^2 - 2TT_2 + T_2^2 &= 1, \\ 2T^2 - 2T(T_1 + T_2) + T_1^2 + T_2^2 - 1 &= 0.\end{aligned}$$

Solving, we can find the value of  $T$ , which is the value of the wave's travel time to reach that cell.

$$T = \frac{-b + \sqrt{b^2 - 4ac}}{2a},$$

being:

$$\begin{aligned}a &= 2 \quad (\text{number of dimensions}), \\ b &= -2(T_1 + T_2), \\ c &= T_1^2 + T_2^2 - \frac{1}{F_{ij}^2}.\end{aligned}$$

Figures 2.2 and 2.3 show an example of what values  $T_1$  and  $T_2$  should take, for the resolution of the arrival time of the wave for two dimensions.

$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	?	1	$\infty$	$\infty$	$\infty$
$\infty$	1	0	1	$\infty$	$\infty$
$\infty$	$\infty$	1	$\infty$	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

Figure 2.2:  $T_1$  is the minimum between adjacent cells on the X axis

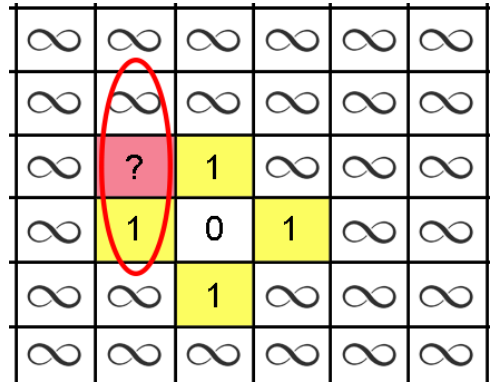


Figure 2.3:  $T_2$  is the minimum between adjacent cells on the Y axis

There is a possibility that the square root, when the time is found, will come out negative. In that case the Eikonal equation cannot be solved. To find the value of time, it is calculated by:

$$F(x) = \frac{e}{T(x)}.$$

Since, for implementations, square cells with unit size are considered:

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

That is, the time of the cell to be calculated is the time of a contiguous cell ( $T_0$ ), plus the time it takes to get from the first cell to the second, it is show in Figure 2.4

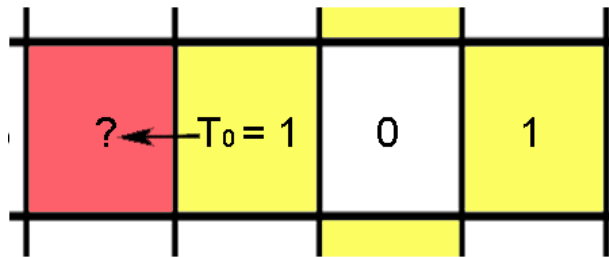


Figure 2.4: Time of arrival (TT) calculation when the value of the Eikonal equation cannot be calculated

Finally, the travel time is calculated as:

$$T(x) = \frac{1}{F(x)} + T_0.$$

### 2.2.3 Phases of the Algorithm

As described above, the space where the wave propagates has been divided into a grid.

For the implementation of the algorithm, each cell of the grid has to be labeled according to the state it is in. The possible different states in which a cell can be found can be three [18]:

- **Frozen:** When the wave has already passed through that cell at a time of arrival (TT). Its value is final, so it cannot be changed.
- **Open:** Cells without an assigned TT, since the wave has not yet arrived and has not been calculated yet.
- **Narrow:** Candidate cells to be the wavefront in the next iteration of the algorithm. They have a TT assigned, but it is not definitive, it can change in the following iterations.

Figures 2.5 and 2.6 show how the algorithm works. The black, gray, and white points correspond to the Frozen, Narrow, and Open states, respectively.

In turn, the algorithm also consists of three different phases:

1. **Initialization:** The algorithm starts by setting a TT of  $T = 0$  in all cells that are starting points, in addition to labeling them as Frozen, the algorithm starts by setting a TT of  $T = 0$  in all cells that are starting points, in addition to labeling them as Frozen, you can see it in Figure 2.5 a). Then all the cells, other than Frozen, that are in the von Neumann neighborhood, are labeled as narrow band and their TT is calculated. This is showed in Figure 2.5 b).

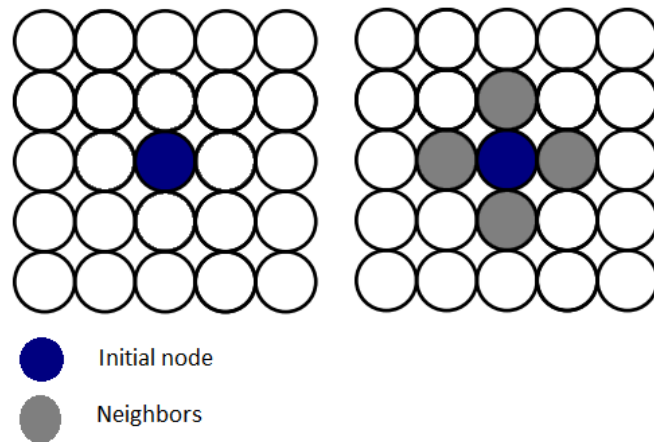


Figure 2.5: Initialization stage of the FMM algorithm

2. **Iteration:** In each repetition of this part of the algorithm, the Eikonal equation is solved for the von Neumann neighbors of the cell with the lowest TT of the narrow band. Once calculated, the neighbors are added to the narrow band, and the cell with la lowest TT is labeled as Frozen. Only neighbors which have an Open state are added to the narrow band.

Finding the cell with the lowest value TT is possible since a list of cells is kept, ordered by their TT, always knowing which is the one with the lowest value for the next iteration. It is showed in Figure 2.6.

This phase continues as long as the narrow band has any stored cells.

3. **Finalization:** This phase of the algorithm is reached when there are no cells left in the narrow band, so the algorithm has reached its end and no further iterations will be carried out. All cells have already been labeled as frozen and also have their TT calculated.

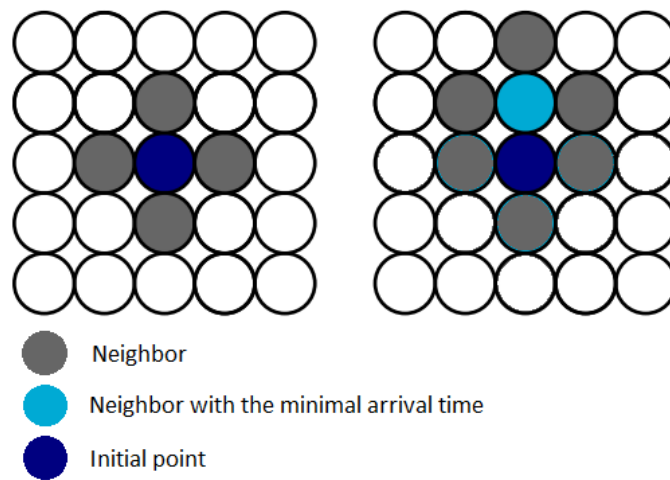


Figure 2.6: Iterative process of the FMM algorithm

## 2.2.4 Pseudocode

---

### Algorithm 1: Initialization

---

```

input :  $G$  = grid size mxn
          $g_s$  = cells where the wave originates
          $NB$  = narrow band
          $V$  = cells velocity
          $g$  = cell
          $g_n$  = von Neumann neighbor
          $T$  = travels time
          $E$  = cell label: Open | Narrow | Frozen
1 for each  $g_s$  in  $G$  do
2    $g_s.T = 0$ 
3    $g_s.E = \text{Frozen}$ 
4   for each  $g_n$  in  $g_s$  do
5     if  $g_n.E \neq \text{Frozen} \mid \text{Occupied cell} \mid g_n.V = 0$  then
6        $p = \text{Resolve Eikonal}$ 
7       if  $g_n.E = \text{Narrow}$  then
8         if  $p < g_n.T$  then
9            $g_n.T = p$ 
10      if  $g_n.E = \text{Open}$  then
11         $g_n.E = \text{Narrow}$ 
12         $g_n.T = p$ 
13        add  $g_n$  to NB
14    end
15 end

```

---

### Algorithm 2: Iterations

---

```

1 while  $NB > 0$  do
2    $g = \text{cell with lower TT in NB}$ 
3    $g.E = \text{Frozen}$ 
4   for each  $g_n$  in  $g$  do
5     if  $g_n.E \neq \text{Frozen} \mid \text{Occupied cell} \mid g_n.V = 0$  then
6        $p = \text{Resolve Eikonal}$ 
7       if  $g_n.E = \text{Narrow}$  then
8         if  $p < g_n.T$  then
9            $g_n.T = p$ 
10      if  $g_n.E = \text{Open}$  then
11         $g_n.E = \text{Narrow}$ 
12         $g_n.T = p$ 
13        add  $g_n$  to NB
14    end
15 end

```

---

## 2.2.5 Computational Complexity

When making comparisons of different algorithms, it is necessary to emphasize computational complexity ( $O$ ), since this is independent of the processing power where the algorithm is executed.

The  $O$  notation is used to express the growth orders of the algorithms in the worst case. That is, what is the performance of an algorithm, in the worst case, as the size of the data to operate increases [31].

Knowing the order of complexity that the algorithm has, it is possible to predict its behavior. The orders of complexity are:

- $O(1)$  - constant order: It does not have a growth curve, its performance is independent of the amount of data it handles.
- $O(n)$  - linear order: The performance of the algorithm is proportional to the growth of the data to be used.
- $O(\log n)$  and  $O(n \log n)$  - logarithmic order: This type of complexity order usually means that the algorithm works with data that has been divided repeatedly. The term  $\log n$  is usually  $\log_2 n$  (although not always), meaning the number of times a set can be divided into two parts without resulting in empty subsets.
- $O(n^2)$  - quadratic order: The algorithm has a performance proportional to the square of the data entered. These algorithms usually exist when, having a series of data, it is necessary to work with each of the data twice.
- $O(n^a)$  - polynomial order ( $a > 2$ ): This type of algorithms with internal loops that make you have to use each data repeatedly.
- $O(a^n)$  - exponential order ( $a > 2$ ): This type of algorithm, in the case of  $a = 2$ , doubles the time used in the calculation for each new element that is added. They are not very useful in practice, since with little data a great computational power is necessary.
- $O(n!)$  - factorial order: They are algorithms with little scalability that make them are intractable.

Assuming that you have a problem of size “ $n$ ”, each algorithm of each computational complexity solves the problem in a certain time [32]. When the problem is twice the size, now being “ $2n$ ”, you would expect them to take twice as long to resolve the problem, but they do not. For example, algorithms of logarithmic order take a little longer to solve the problem twice as long and exponential order take several times longer than twice, with respect to the problem “ $n$ ”.

Keep in mind that scalability does not mean efficiency. Thus, a less scalable complexity algorithm can yield better results than one with better scalability and worse efficiency. Ultimately, the volume of data would outweigh efficiency, but it is important to know that better scalability does not mean better calculation times, especially with little data volume.

In the case of this work, the problem size “ $n$ ” is given by the number of cells in the grid that make up the map.



Fast Marching Method is a method that has a complexity  $O(n \log n)$ , the  $\log n$  factor being introduced by the administration of the priority queue, having to obtain the cell with the lowest TT of the narrow band in each iteration.

# Chapter 3

## State of the Art

### 3.1 Fast Marching Method Modifications

In this work, the Fast Marching Method algorithm uses different priority queues. Each of these priority queues has  $O(\log n)$  computational complexity. Modifications to this algorithm have the same operation as FMM, but changing the priority queue that controls the narrow band, for another that each modification introduces.

The priority queues that have been used: a binary stack and a fibonacci stack.

The basic operations of priority queues are:

- *Get minimum:* The smallest data item is extracted from the saved data item. It is a simple operation since the priority queue orders the entered data by value.
- *Delete minimum:* The smallest data is deleted.
- *Insert:* Add a new data to the priority queue.
- *Increase / decrease key:* Increase or decrease the value of the data in the priority queue.
- *Join:* Two priority queues are joined.

#### 3.1.1 Fibonacci Stack

It is a data structure, similar to the binary stack. Its main characteristic is the amortization of processes, that is, it makes fast operations take a little longer, but in return slow operations are accelerated [33].

As stated in the previous chapter, the complexity of using this stack, to obtain the smallest TT value of the narrow band, is  $O(\log n)$ .

The Fibonacci stack [34, 1] is based on the use of several trees made up of nodes. These trees have the property that a child of a node always has the same or greater value than the parent. Due to this property, the root is always the node with the lowest value.

Process amortization is measured by the potential of the pile:

$$Potential = t + 2m,$$

where  $t$  is the number of trees and  $m$  is the number of marked nodes. To consider a node as marked, one of its children must have been removed and, in turn, be children of any other node. Therefore, root nodes cannot be marked.

For quick deletion and concatenation, the roots of all the trees are a double linked list. A graphical representation of the Fibonacci stack can be seen in Figure 3.1.

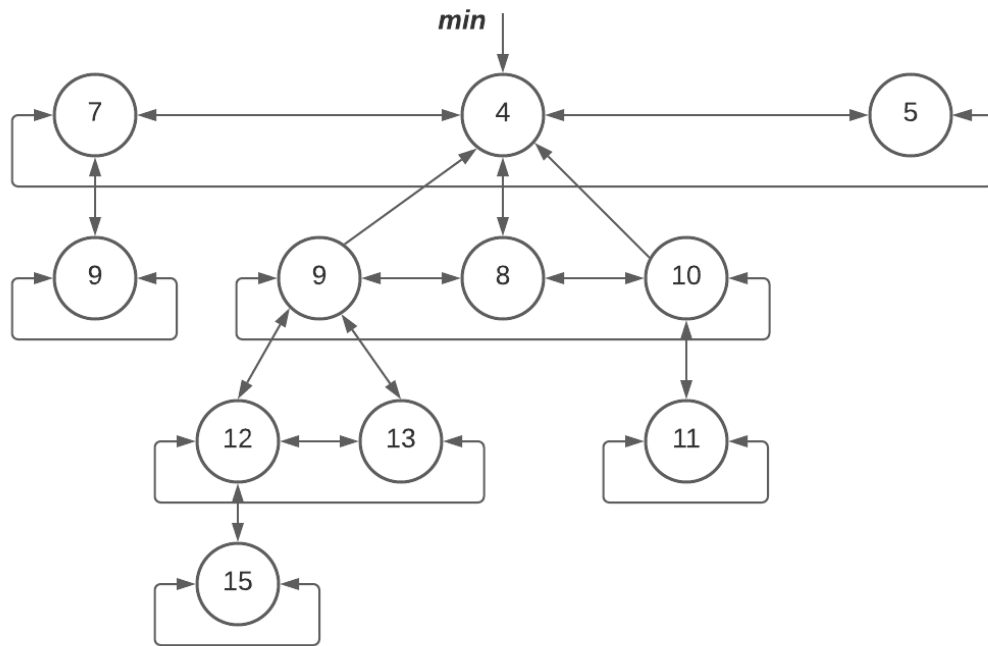


Figure 3.1: Graphical representation of the trees that form the Fibonacci stack [1]

The basic operations of a Fibonacci stack [1] are:

1. **Obtain minimum:** It is a very simple operation, since the minimum value corresponds to the data located at the root.
2. **Join:** Two Fibonacci stacks can be joined. To do this, the stack with the highest value is added to the root, as a child of the other stack.
3. **Insert:** To add new data, it is only necessary to perform the “join” operation with a new Fibonacci stack. This new stack only contains the new data to add.
4. **Remove minimum:** The minimum value is removed. For it:
  - The node with the lowest key (the root) is deleted.
  - Union of the children of the deleted node.
  - As long as there are two roots with the same degree, the “join” operation is performed. The degree of a node is equivalent to its distance from the root. Thus, the child of the root node, has degree one.

5. **Decrease key:** The value of a data in the Fibonacci stack is decreased. It is done as follows:

- Separate the node to be decremented from the tree to which it is connected.
- Reduce the value of the node.
- Add the decremented node to the tree.

### 3.1.2 Binary Stack

Like the Fibonacci stack, the binary stack [35] is a data structure that is based on trees and nodes of complexity  $O(\log n)$ .

It has the characteristic that each node has a value higher than that of its children (binary stack of maximums), or vice versa, smaller than that of its children (binary stack of minimums). Obviously, the version that interests the algorithms is the minimum version. An example of a binary maximum stack can be seen in Figure 3.2.

In order to use an array as a binary stack, as in Figure 3.2, it is necessary to take into account the positions of the root and where its children are. The root node always goes in position 0, and the position of the two children of each node can be calculated in a simple way, as:

$$\text{Position } K, \text{ child } 1 = 2K + 1.$$

$$\text{Position } K, \text{ child } 2 = 2K + 2.$$

The algorithm, used in the comparisons, has a modification of the binary stack called the d-ary stack [36].

This modification has the ability to make the priority decrements operations faster at the cost of slowing down the minimum clear operation.

This allows to obtain better times in the algorithms in which decreasing priority is used more times. These operations are more common in algorithms like Dijkstra's or, as is the case, FMM.

The structure consists of an array of  $n$  elements that has the same properties as the binary stack.

The basic operations of a binary stack [35] are:

- **Insert element:** it is done by adding an element in the position that respects the condition of the semi-complete tree, that is, that the insertions are made from left to right in order. Once placed in their position, it must be fulfilled that the children are older. Therefore, positions are exchanged from children to parents, while the child is a minor, until reaching the root if necessary. The process can be seen graphically in Figure 3.3.
- **Extract minimum:** the element that is in the root is extracted, since it is the data with the lowest value. Then it is checked which of its children has a lower value and it becomes the new root, if it is the case, they are exchanged with the children as long as the root is greater. The process can be seen in Figure 3.4.

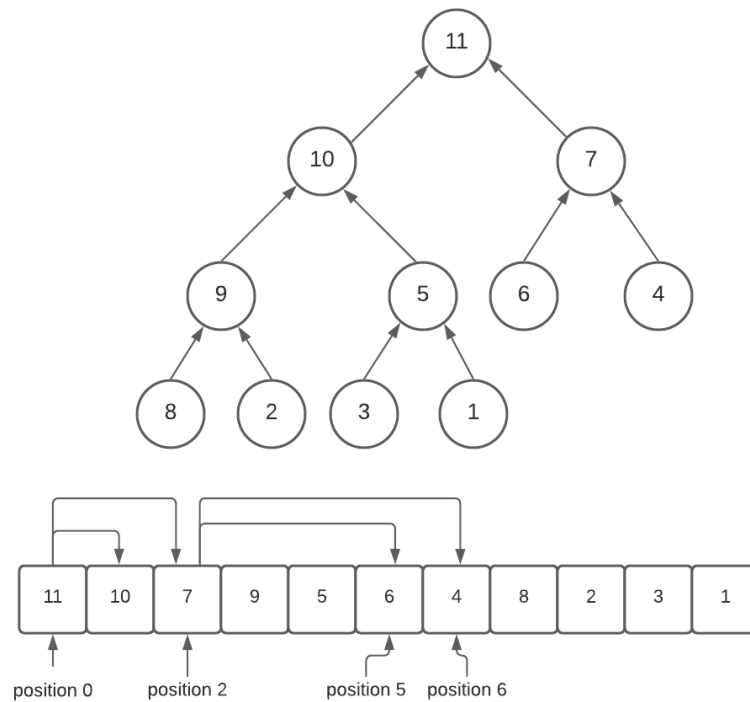


Figure 3.2: Binary tree representation and the equivalent array

### 3.1.3 Simplified Fast Marching Method

This modification of FMM avoids that the cells of the priority queue are not recalculated. If any cells are recalculated, they need to be moved into the priority queue, thus requiring drivers to handle the queue.

To avoid the use of these drivers, it is proposed to use the SFMM (Simplified Fast Marching Method) algorithm, which is obtained from the following changes [11]:

- If a value has been recalculated for a cell, it is accepted that it has two different values. When you need to know the value of the cell, the smallest value is the one you look at first and the cell goes into the Frozen state. If a second value appears, it is discarded. If this mechanism was not used, errors would appear due to the recalculation of cells.
- The errors also appear if the cells that are not Frozen are used for the calculation of the new values of the cells in the priority queue. So, it is not necessary to assign a value to the cells before they are labeled as Frozen.

These simplifications mean that a standard priority queue can be used. In the case of the experiments, the priority queue of the Boost library has been used.

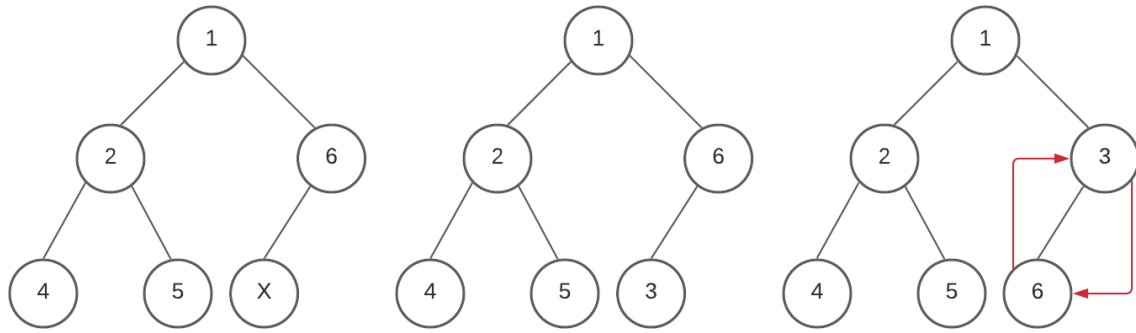


Figure 3.3: Process of inserting data ( $X = 3$ ), in a binary stack

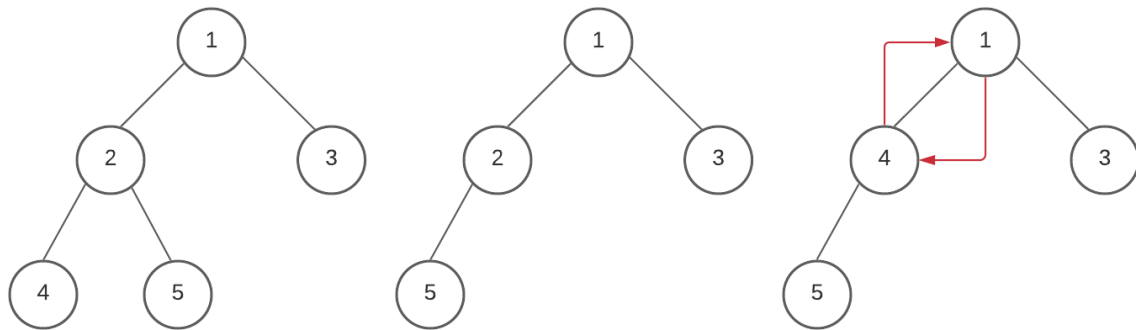


Figure 3.4: Root extraction process in a binary stack

### 3.2 Modifications of $O(n)$ in Fast Marching Method Algorithm

The order of complexity of the algorithms is very important for them to be efficient. Due to the fact that in the FMM algorithm its modifications are not of the optimal order of complexity, other variations have been developed that do comply. Three of these algorithms that have been implemented are explained below. Each of them has a computational complexity  $O(n)$ .

#### 3.2.1 The Group Marching Method

Group Marching Method [37], hereinafter GMM. It is a modification of FMM developed by Seongjai Kim and Daniel Folie, of complexity  $O(n)$  by avoiding using a complex priority queue to control the narrow band. Thus, GMM is able to improve the efficiency of FMM while maintaining the same precision.

This algorithm is based on the narrow band technique, as seen in Figure 3.5, in which the points within it form the advancing front of the generated wave. Unlike in FMM, the algorithm do not select the cell of the narrow band with the lowest value. In return, it advances multiple cells at the same time. This advance occurs by means of a double

iteration in each cell, although it does not double the time of arrival (TT) calculation, but the time used is slightly greater than a single iteration.

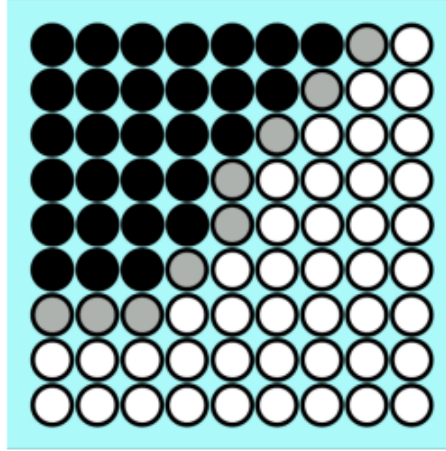


Figure 3.5: Wave expansion in GMM, white, black and gray points correspond to the states: Open, Frozen and Narrow respectively

To solve the arrival times, the algorithm uses the Eikonal equation. In addition to the Open, Narrow and Frozen label system, which allows you to know the state of each cell in the grid.

As mentioned, GMM avoids using a priority queue, but instead has to maintain a list of cells. From that list of cells called Gamma  $\Gamma$ , several of them are able to advance as if they were the cell with the lowest TT of the narrow band in FMM.

The authors have created a method to decide which cells are the ones that can advance the wavefront. Given two close cells of the narrow band, if the difference between their arrival times are less than:

$$Difference\ TT \leq \frac{1}{\sqrt{D}} \times h \times s,$$

where  $D$  is the number of dimensions,  $h$  is the length of the cell in that dimension and  $s$  is the slowness, that is, the inverse of the expansion speed of the wave  $\left(\frac{1}{F}\right)$ . If this condition is met, it means that the line that joins the two cells and the direction towards which the narrow band expands is greater than 45 degrees, as can be seen in Figure 3.6.

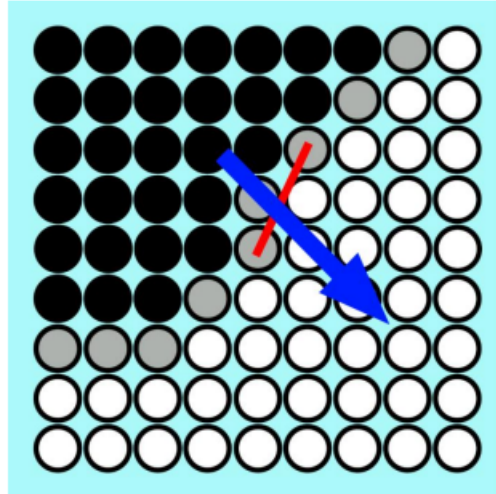


Figure 3.6: Angle between the direction of expansion of the wave (blue arrow) and the line that joins two cells (red line)

If the two cells are not together, the time of arrival (TT) update procedure, solved by the Eikonal equation, does not affect each other. In case of being adjacent, the TT of one affects the other in case the direction of expansion of the wave and the line that joins both cells, is less than 45 degrees. In conclusion, two cells do not affect each other and can be expanded at the same time in the same iteration, if:

$$G = \{x \in \Gamma : T(x) \leq T_{\Gamma, \min} + \frac{1}{\sqrt{D}F_{\Gamma, \min}}\},$$

$G$  being the set of cells that will be advanced at one time in the iteration. However, to solve the equation it is necessary to have a knowledge of the minimum arrival time,  $T_{\Gamma, \min}$ , of all  $\Gamma$  cells and of the speed  $F_{\Gamma, \min}$  in each subsequent iteration. To find these values, an  $O(n)$  algorithm is necessary. Therefore, the authors suggest using a global limit, and increasing it as the wave spreads.

This value is called TM, and it is found at the beginning of the algorithm only once. The value of TM corresponds to the value of the minimum TT of the initial narrow band, and is increased in each iteration by:

$$\text{delTAU} = \frac{1}{F_{\text{global, max}}}.$$

This simplification results in a worsening of the efficiency when the contrasts in the expansion speed of the wave are large. The low efficiency is due to the fact that if  $\text{delTAU}$  has a low value, in each iteration few cells are able to expand. Therefore, the algorithm needs more iterations to complete all cells, taking longer than with a higher value  $\text{delTAU}$ .

A double update of the  $\Gamma$  cells that expand in that iteration is performed to avoid inconsistencies when two wavefronts collide, due to the common cell neighborhood can be affected by both fronts. This algorithm is known as UFMM. It is an extension of Dijkstra's algorithm and is based on finding the cell with the shortest arrival time of the narrow band in each iteration, just like FMM.



As seen in FMM, to find this cell it was necessary to use a priority queue of order  $O(\log n)$ . To avoid the use of priority queue, the authors propose to use a strategy of “*fast sweeping algorithms*” methods. This type of structure has been named as “*Untidy Priority Queue*”, which changes the complexity of the priority queue from  $O(\log n)$  to  $O(1)$ .

### 3.2.2 Fast Iterative Method

Fast Iterative Method, FIM is an algorithm developed by Won-Ki Jeong and Ross Whitaker [38]. Its main idea is to avoid the use of a complex priority queue while maintaining the narrow band system. In this way, the algorithm goes from having computational complexity  $O(\log n)$  to  $O(n)$ .

To store the cells of the narrow band, FIM creates what the authors call the active list, equivalent to the narrow band in the previous methods. At each iteration, the cell list expands to update the arrival time (TT) values. A cell is only removed from the active list, once its solution is reached. Eikonal equation is used to find the TT of the cells as the narrow band is expanded.

The goals of the algorithm include good performance, cache consistency, and scalability for multiple processors.

To exceed these objectives, the requirements to be met [38] are:

- The algorithm should not enforce a particular narrow band update order. This requirement is necessary to maintain cache consistency, for example, the FMM algorithm requires random memory access that prevents good cache consistency.
- You should not use a complicated data structure to sort the narrow band. This criterion is necessary for SIMD (Single Instruction, Multiple Data) [39], a method to improve efficiency in applications that use the same operation on several data simultaneously. FMM maintains a priority queue with a list of cells, relatively small compared to the total number of cells. This circumstance cannot be efficiently implemented in SIMD, because it is more efficient to process a lot of data in a common operation.
- In each iteration, several points must be updated, and not just the one with the lowest value, as in FMM.

# Chapter 4

## Methodology

In this chapter, we show an extensive analysis of the problems found in the FMM algorithm, and which was the methodology used to solve these problems. This chapter consists of four sections, Section 2 and Section 3 describe the changes made in the FMM algorithm, and also how these affect its performance. In Section 4, we explain how the FMM algorithm was implemented, showing the most important developed functions.

### 4.1 Phases of Problem Solving

- The first step is to implement the FMM algorithm, in addition to defining the input parameters required for the correct operation of the Path Planning such as map, starting point, arrival point, and padding.
- The second step is to optimize the algorithm of the FMM, trying to reduce its computational complexity.
- The next step is to graph the results obtained based on the initial matrix, the size of the matrix, number of obstacles in the matrix, and the initial points.
- The final step is to compare the results with respect to previously implemented Fast Marching method variants.

#### 4.1.1 Description of the Problem

There is a wide range of algorithms that solve the Path Planning problem, however one of the biggest challenges is solving the problem of computational complexity that these algorithms require. Fast Marching Method is a complete method that can be easily replicated in 3D environments, however one of its major disadvantages is the computational cost it requires.

In the present work, a new variant of the Fast Marching Method is proposed that aims to optimize its execution time without altering its performance, which means getting a route in less time. The edges or obstacles within the map must also be considered, to get as fast as a safe route.

### 4.1.2 Analysis of the Problem

After several experiments and research, it was possible to get to the major root of the problem that FMM has. The major problem lies in the management of the narrow band. This is a list from which all the possible new wavefronts are kept. This list must also be constantly updated with each iteration of a new wavefront.

To be clear about the problem, we must understand how FMM works, specifically how to choose a new wavefront. Once an iteration of a wavefront is finished, the algorithm for verifying whether the narrow band contains any value. If true, it chooses the minimum value stored on that list. To get the minimum value, the entire list must be accessed and iterated, which entails a high computational time since the list is disordered.

This problem has been addressed frequently by creating new variants that seek to solve it by changing the narrow band for a stack. However, these methods have not had the expected success, since when increasing the size of the initial matrix they become slow. For this reason, in the present work, we approach the problem simply, ordering the list each time the narrow band is updated.

Initially, various conventional sorting methods were experimented with, such as merge [40], bucket [40], and insertion sort [40]. However, the first two methods had the worst execution times because in this case, the update of the narrow band is constant. Merge and bucket sorting methods take a long time to order a single element in the list compared with insert sort. The insertion sort method had the shortest execution time, however, with extensive lists, the algorithm took a much longer execution time than expected.

This led to the creation of a sorting method that does not take computationally long to sort a previously ordered list. The method focuses on finding the correct position where the last value will be inserted into the list, and thus we do not have to iterate through the entire list in the best case. This proposed algorithm is explained in the next subsection.

## 4.2 Proposed Algorithms

In the present work, we present a new ordering method called SEF, which is a simple to implement, efficient, and fast method. SEF sort has been applied to the FMM algorithm, getting a new variant that we called OFM. OFM comes from the initials Ordered Fast Marching, since OFM uses SEF sort to order the narrow band in each new iteration. In this section, we explain in more detail what it is and how a SEF sort performs.

### 4.2.1 Simple, Efficient, and Fast Sorting Method (SEF)

The ordering method implemented SEF is based on the binary search method[41], which comprises repeatedly dividing a set or list in half, until the probable locations are reduced to just one. SEF sort, on the other hand, does not reduce a list to a single location in all cases. For this reason, SEF avoid to waste unnecessary computational time and unnecessary recursion. One of the most outstanding characteristics of SEF sort is the reduction of the list size in each new iteration.

SEF sort comprises four key steps that make it efficient. In addition, it should be noted that the input of this method must be an ordered set or list, and the new value that we

want to enter the set or list.

- If the value is less than or equal to the beginning of the set, the initial position is returned and this value is saved in that position. To better understand it, we can see it in Figure 4.3.
- If the value is greater than or equal to the end of the set, the final position is returned and the value is saved in that position. To better understand it, we can see it in Figure 4.3.
- If the value does not meet the previous two, this divides the set or list into two smaller subsets, and one of the keys that has been used in this division was to eliminate the beginning and the end of the set. This practice obtained a better result, since for the worst case when the value was close to the beginning or the end, the algorithm required to make multiple divisions until reaching the corresponding position. To better understand it, we can see it in Figure 4.1.
- Once the set is divided, each subset is evaluated, we find the subset to which the value to be saved belongs and we repeat step 1. To better understand it, we can see it in Figure 4.2.



Figure 4.1: Case 1, SEF sort method

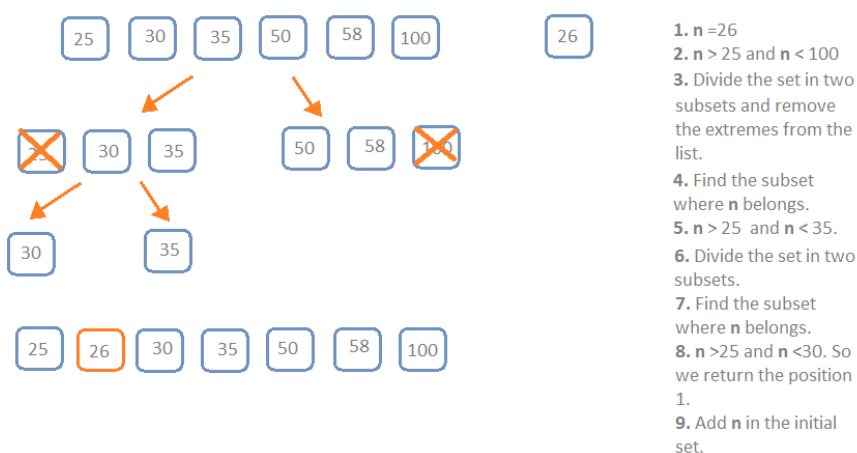


Figure 4.2: Case 2, SEF sort method



Figure 4.3: Case 3, SEF sort method

### 4.2.2 Ordered Fast Marching Method (OFM)

OFM is a new variant of FMM, which uses the SEF sort method to update the narrow band at each iteration. Narrow band is a list that FMM uses to store all possible new wavefronts. That is, narrow band stores all the cells that have already been visited, but do not yet have a constant time value. The biggest problem that we observe in the narrow band is that each time it grows, it becomes difficult to find and eliminate the minimum values.

Therefore, OFM sorts the narrow band each time a value in the list is inserted or updated, making finding or removing minimum values easy. In addition, thanks to the optimal performance that has been achieved with the SEF sort method, we have made the sort time of the narrow band quite small.

In other variants, such as Binary and Fibonacci, have replaced the narrow band with a stack to optimize the operations in the narrow band. However, these variants of FMM are complex to replicate, since each stack requires implementing a series of functions for their correct operation.

### 4.2.3 Computational Complexity of OFM

The computational complexity is necessary to explain the order of growth that the algorithm has. As OFM is in charge of handling the narrow band, it is necessary to explain the computational complexity for each operation that can be performed on this list. The operations are:

- Find minimum: For this operation, the algorithm has an  $\mathcal{O}(1)$ , since the narrow band always stays in order.
- Extract minimum: In the same way, as in the previous operation, the algorithm has an  $\mathcal{O}(1)$  for this operation.
- Insert value: For this operation, the algorithm has an  $\mathcal{O}(1)$  in the best of case. In the worst case, it has an  $\mathcal{O}(\log n)$ , since the list is divided into smaller subsets until finding the correct position. However, as explained in the algorithm description, this problem has been tackled efficiency by deleting the extremes of the list at each iteration.
- Delete value: This operation has an  $\mathcal{O}(1)$  since the narrow band is an object list.

## 4.2.4 SEF Sort Pseudocode

---

**Algorithm 3:** Sort method proposed based on Binary search

---

**input :** A set of ordered values *numberlist*, *number* is a numeric value that would be inserted inside the set, and *initialposition* is a integer value which stores the current position of the set.

**output:** Return a position of *numberlist* where we have to insert the *number*

```

1 Function Sortmethod(numberlist, number,initialposition):
2   n=length of numberlist;
3   if n > 0 then
4     if number ≤ numberlist[0] then
5       return initialposition;
6     else if number ≥ numberlist[n-1] then
7       initialposition += n;
8       return initialposition;
9     else
10      if n > 3 then
11        left = numberlist[1:n//2];
12        right = numberlist[n//2:n-1];
13        initialposition += 1;
14      else
15        left = numberlist[:n//2];
16        right = numberlist[n//2:];
17        initialposition = initialposition;
18      end
19      if ( len(left) > 0 ) then
20        indexleft = initialposition;
21        inicio = left[0];
22        fin = left[len(left)-1];
23        if number > inicio and number ≤ fin then
24          return ( Sortmethod(numberlist,number, indexleft));
25        else if number ≤ inicio then
26          return (indexleft);
27      if ( len(right) > 0 ) then
28        indexright = initialposition + len(left);
29        inicio = right[0];
30        fin = right[len(right)-1];
31        if number > inicio and number ≤ fin then
32          return ( Sortmethod(numberlist,number, indexright));
33        else if number ≤ inicio then
34          return (indexright);
35        else if number ≤ fin then
36          return (initialposition + n);
37      end
38  return initialposition;

```

---

As we can be seen in the **Algorithm 3** in the first two conditions, the algorithm asks if the value is less than the beginning of the list, or greater than the end of the list. If none of the conditions are met, the list is divided into two sub-lists.

Before dividing the list, we need to eliminate the extremes of the list. Thus, if the list is greater than three, we delete the extremes and divide the list. Finally, we get a sub-list without the beginning and another without the end of the list. On the other hand, the extremes are not deleted, and the list is simply divided.

Then we are going to verify in which of the two sub-lists the new value should be inserted. Once we recognize the sub list, we call the function again, passing the sub list as the input list. These steps are repeated until we return to a position.

Furthermore, the returned position belongs to the initial list passed, to ensure that the **initialposition** variable updates its value in each iteration.

### 4.3 Padding Metric

The construction of a path planning algorithm is usually a hard challenge, due to all the aspects that this problem covers, such as obtaining safe and optimal routes. With FMM, getting optimal routes is covered excellently. However, safe routes have not been taken into consideration, that is, routes that can be circulated without generating an accident with vehicles.

For this, in the present work, the FMM algorithm has been changed by adding a metric called padding, this metric is essential to achieve safe routes. The following graphic Figure 4.4 shows the construction of a route with and without padding. By adding padding metric, we get the object that passes through the route away from obstacles. In addition, as we can see in the contour graphs (see Figure 4.5), it is differentiated that when applying a padding to the matrix, the range of available cells is reduced, which leads to the execution of the algorithm also being reduced. After several experiments, we can conclude that this parameter helps both in the execution time, and in obtaining a safe route.

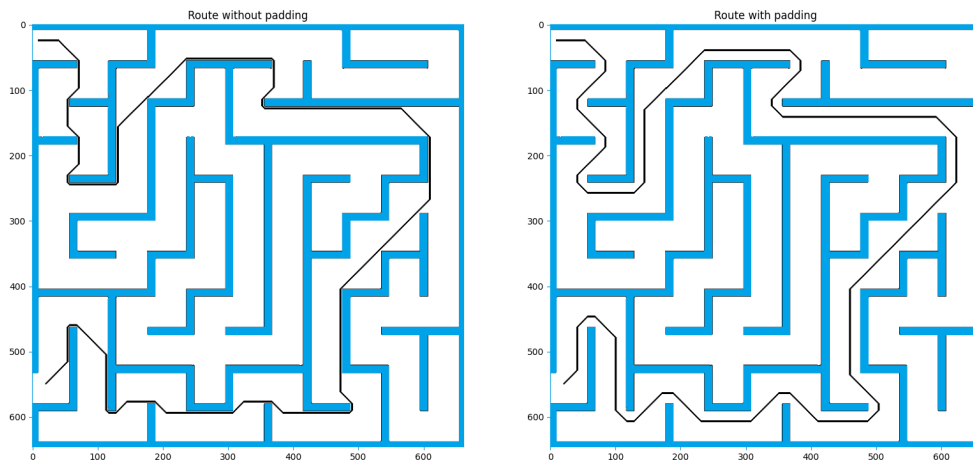


Figure 4.4: Route with and without padding metric

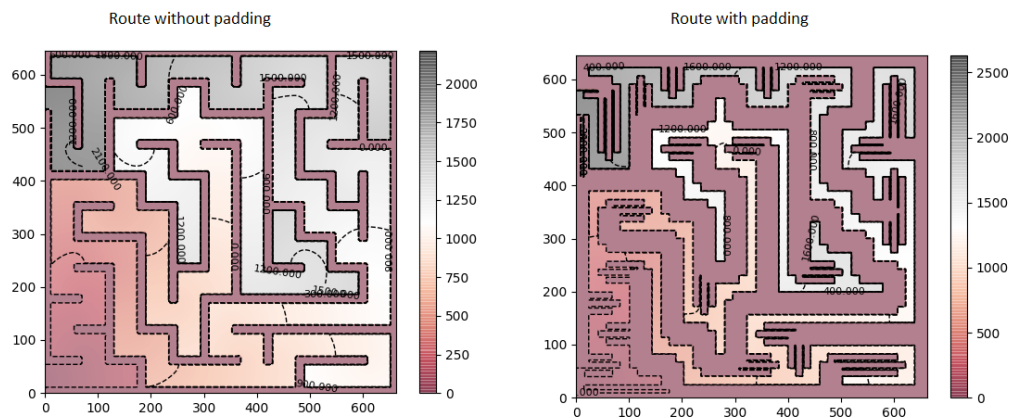


Figure 4.5: Contour graphs for routes with and without padding metric

## 4.4 Implementation of FMM

This algorithm was developed entirely in python, with the help of various libraries such as numpy, math, time, cv2, and matplotlib.

For the implementation of the Fast Marching Method (FMM), it was necessary to assign value attributes to each cell of the matrix. The inputs required to executed the algorithm are a binary image (black and white), initial points, target point, padding, and anchor. For this implementation we use the following functions:

- **Conversion Image:** This function reads a file in any image format and converts it to a matrix.



- **Construct Inicial:** In this function, we modify the matrix obtained in the conversion image function, which we add the initial points.
- **Construct Matrix:** Once we have the image matrix with its initial points, then we build an object matrix. Any cell in this matrix has the following attributes: time, speed, coordinates, type, and tail. Furthermore, in this function, it is defined that the initial points have a time equal to zero and the type equal to frozen. For the other cells, the type is narrow and the time is infinite.
  - Time: It saves the arrival time from the wavefront cell to that cell.
  - Speed: Speed at which the wave is propagated within that cell.
  - Type: State of the cell. This state tells us if the cell can be traversed or not. The states that we can find are: narrow, frozen, or open.
  - Coordinates: Position it occupies within the matrix.
  - Tail: It saves the information of the cell from where the wave originated up to that cell.
- **Main FMM:** This is the most important function, here we determine what the new wavefront is. In the case that the narrow band is empty, the wavefronts are the initial points. This function ends when all cells have a frozen type assigned, which means that they have already been visited and have an assigned time value.
- **Find Neighbors:** In this function, we find all the neighbors for each cell which is a wavefront. For each neighbor found, its travel time is calculated, its type is assigned to Narrow, and the narrow band is updated. At the end of this function, we return to the Main FMM function.
- **SEF Sort:** This function is used to sort the narrow band when a value is entered or updated.
- **Eikonal Equation:** In this function, we calculate the arrival time from one point to another. In this case, it is used to calculate the arrival time from the cell that is generating the wave to one of its neighbors.
- **Route Construction:** This function is used to build the optimal route based on the time matrix obtained in the execution of FMM. In this case, we start from the target point and evaluate the value of its eight closest cells. The cell with the lowest value is chosen and the process is repeated until we reach the inicial point.

We can find the code on my personal GitHub. The following link is the direct access to the repository in which it is hosted: <https://github.com/JhonSaguay/OFM-Ordered-Fast-Marching-Method->

# Chapter 5

## Results and Discussion

This chapter shows the results obtained after several experiments and also makes a complete analysis of the new FMM variant called OFM compared to existing FMM variants. The FMM variants chosen for this comparison were FMM, Binary, and Fibonacci FMM. These methods were chosen because they use the narrow band to carry out the control of the new wavefronts.

The experiments were carried out in five scenarios. The image on which we worked in the first three scenarios was an image with no obstacles. The biggest challenge in these scenarios was increasing the size of the image. The last two experiments were carried out on images with obstacles where the padding metric was also evaluated. In addition, each image had a different amount of obstacles. We explain each scenario with most detail in the first section.

### 5.1 Performance Evaluation

In this section, we explain how the behavior of the algorithms was evaluated and which were the parameters that were evaluated. We also detail all the scenarios that performed the simulations. We also describe the characteristics of the computer used for the simulations, as well as the software used to develop the algorithms.

#### 5.1.1 Simulation Setup

A simulation of FMM requires defining several initial parameters, which are: initial image, and starting point. So, in order to evaluate the behavior of the FMM and its variants, different scenarios have been created by changing the initial parameters described above. Furthermore, to guarantee the reliability of the results, 10 repetitions have been executed for each case. Due to the route obtained is always the same for each case, the parameter to be evaluated is the execution time of the algorithm.

### 5.1.2 Experimental Environment

The experiment was carried out on a personal laptop. This computer is equipped with 16GB of RAM, a processor Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz-1.99 GHz, a NVIDIA GeForce 1050 graphics card, and the Windows 10 Pro operating system.

### 5.1.3 Software Environment

The complete algorithm was fully developed in Python 3.6. Python [42] is a general-purpose language, it is prepared to carry out any type of program. It is an interpreted language; it is unnecessary to compile the source code to execute it. In addition, it is a multi-paradigm programming language, which supports functional programming, imperative programming, and object-oriented programming. It is characterized by being a simple, versatile programming language and also fast development. The libraries used were NumPy, Math, Matplotlib, and OpenCV.

#### NumPy

Numpy [43] is a Python library that provides the creation of large multidimensional vectors and matrices, and a variety of routines for quick operations on matrices.

#### Math

Math [44] is a Python library that provides us access to some common math functions and constants, which we can use throughout our code for more complex mathematical computations.

#### Matplotlib

Matplotlib [45] is a comprehensive library for creating static, animated, and interactive visualizations in Python.

#### OpenCV

OpenCV [46] is an open source library that has a wide range of algorithms for computer vision. This library allows us to read images and transform them into matrices.

### 5.1.4 Scenarios Description

Five scenarios have been created, which are divided as follows. For the first three scenarios, we chose an image with a white background and changed its size from 200x200 pixels to 2000x2000 pixels. Also, the starting point was changed for each scenario. The starting point is a very important parameter of FMM, since the size of the narrow band depends on it. For the last simulation scenarios, it has been showed how the padding metric affects the execution time of the FMM. For this, in scenarios 4 and 5, two images of labyrinths have been chosen as the input parameter. In each scenario, we use a maze image with a

different number of obstacles. The maze images were chosen, as these are very similar to a map of an urban area. In the last scenario, we show a simulation carried out on a real map, simulating a practical example of path planning using FMM. This scenario is used solely for the demonstration purpose of the application of FMM in path planning.

In addition, in each scenario, we show how the expansion of the wave would be within the initial image for each **initial point**.

### Scenario 1

For this scenario, we decided that the starting point is  $x = 0$  and  $y = 0$ . The possible wavefronts (cells available to propagate the wave) in the first iteration are two because the starting point is in a corner of the matrix. For this reason, the narrow band grows slowly, which implies that the execution time of the algorithms changes. Figures 5.1 and 5.2 show how the wave expands from the initial point to the last point of the matrix.

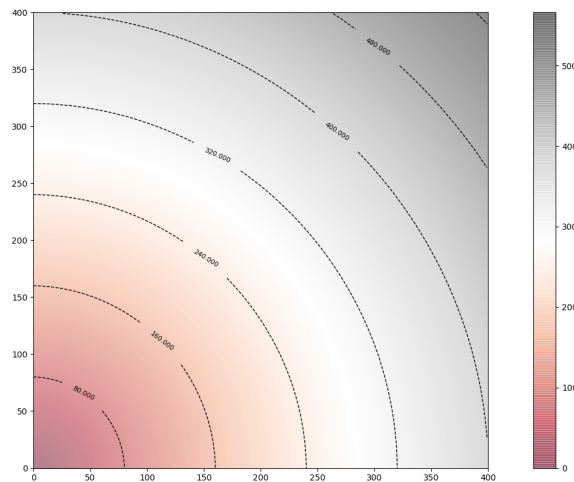


Figure 5.1: 2D Contour graph of the wave expansion for Scenario 1

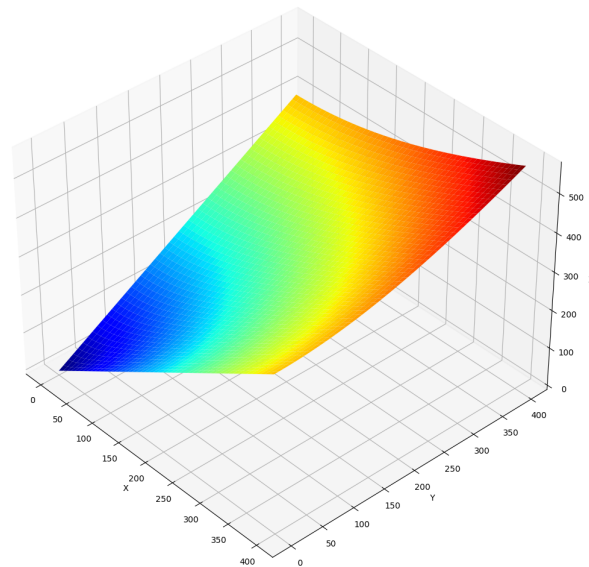


Figure 5.2: 3D Surface of the wave expansion for Scenario 1

## Scenario 2

In this scenario, we increased the difficulty by changing the starting point by  $x = \frac{n}{2}$  and  $y = 0$ , where  $n \times m$  is the size of the matrix. For this reason, the starting point had three possible wavefronts (cells available to propagate the wave) in the first iteration, which implies a faster growth of the narrow band compared to Scenario 1. Figures 5.3 and 5.4 show how the wave expands from the starting point to the last point of the matrix.

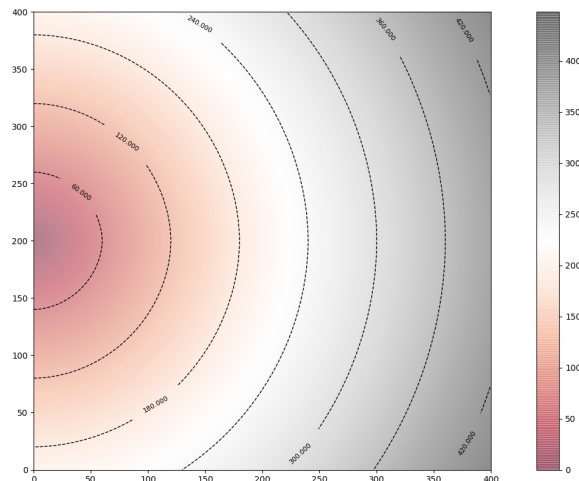


Figure 5.3: 2D Contour graph of the wave expansion for Scenario 2

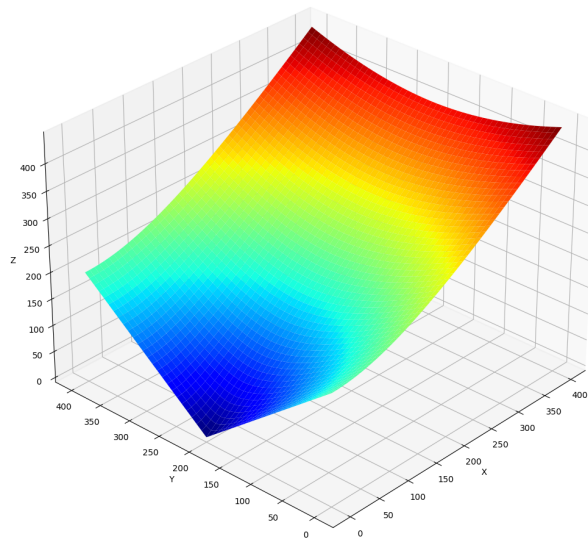


Figure 5.4: 3D Surface of the wave expansion for Scenario 2

### Scenario 3

The starting point for this scenario was the center point of the matrix, since the center point had four neighbors available to propagate the wave. So this caused that the narrow band to grow faster than in the previous scenarios, causing computational complexity to increase. Figures 5.5 and 5.6, we show how the wave expands from the starting point to the last point of the matrix.

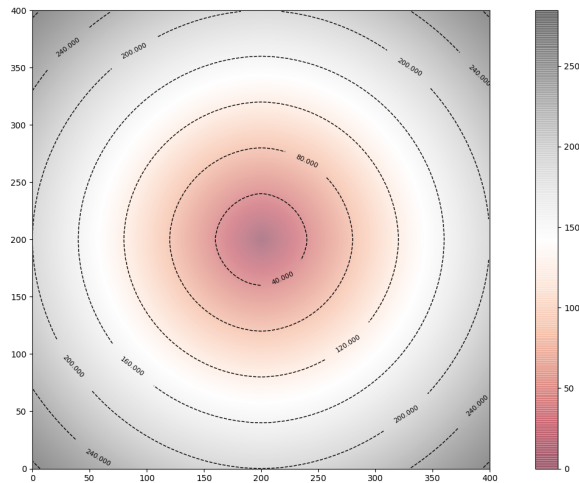


Figure 5.5: 2D Contour graph of the wave expansion for Scenario 3

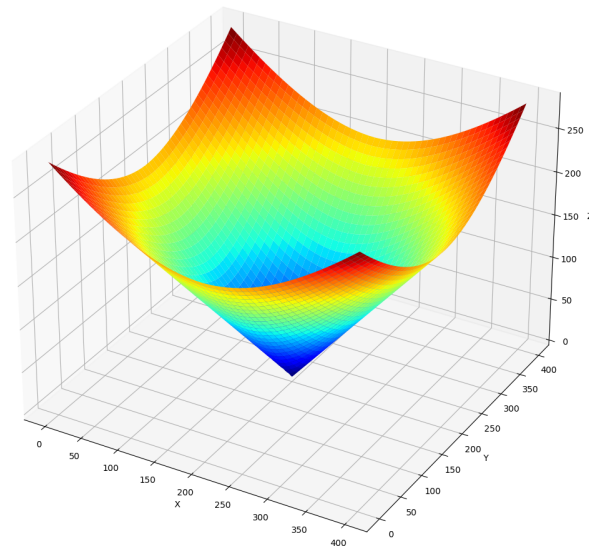


Figure 5.6: 3D Surface of the wave expansion for Scenario 3

#### Scenario 4

In this scenario, a maze was used as the input image, which can be seen in Figure 5.7.

For each simulation, the padding metric was changed, which allows the cells that propagate the wave are away from the obstacles. Figures 5.8 and 5.9 show the contour plots when we change the padding metric. These figures show that, when the padding metric increased, the cells to build the optimal path are reduced. Therefore, this also influenced the execution time of the algorithm.

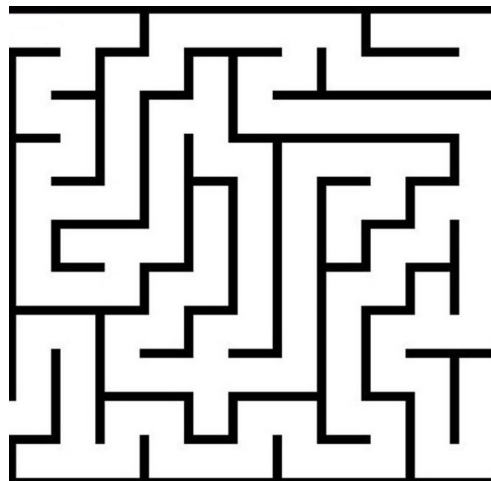


Figure 5.7: Maze image used for Scenario 4

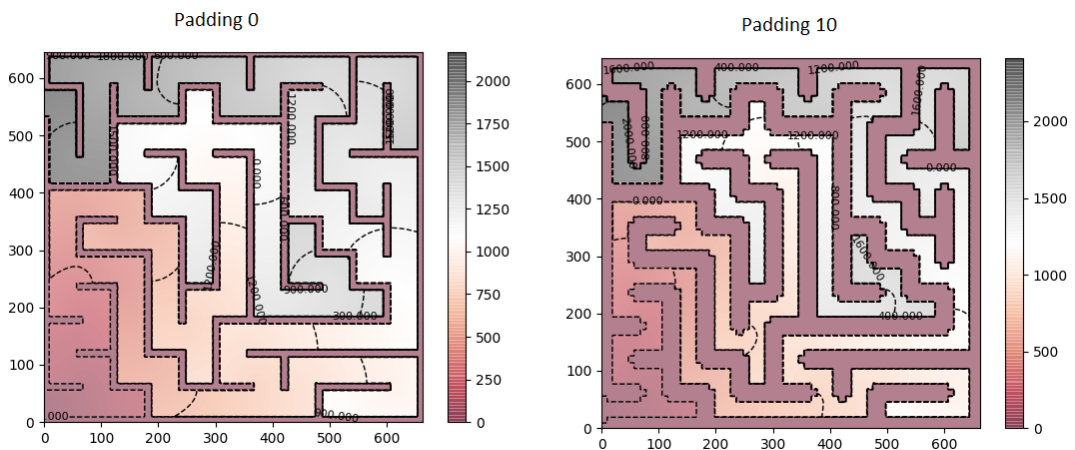


Figure 5.8: 2D Contour graph of the wave expansion with respect to padding metric

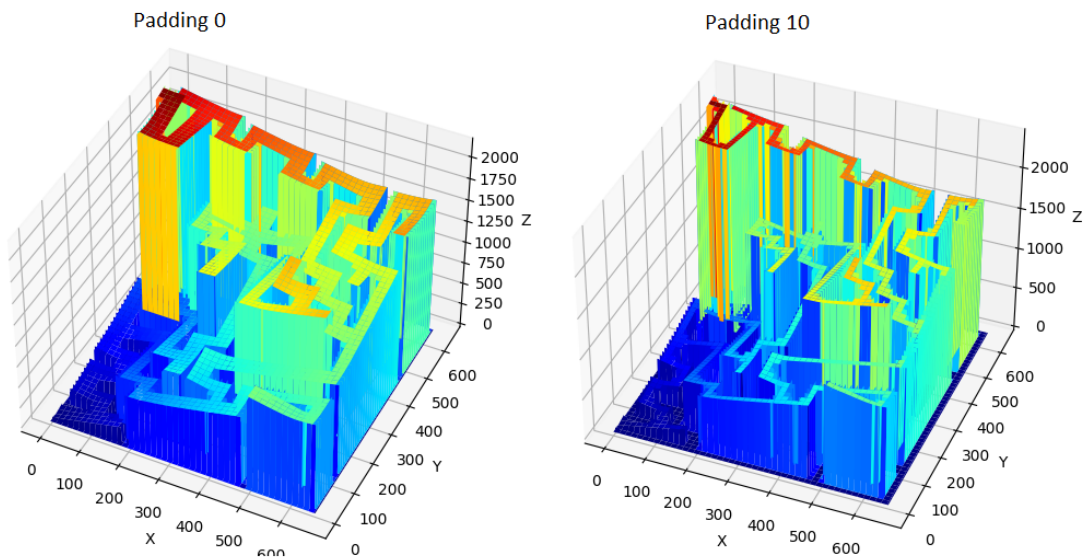


Figure 5.9: 3D Surface of the wave expansion with respect to padding metric



### Scenario 5

In the same way, as in the previous scenario, the image of a labyrinth was used, which we can see in Figure 5.10. Furthermore, it can be seen that the labyrinth used in this scenario has many more obstacles than the image in the previous scenario. This change increased the difficulty when finding the optimal route from point A to point B, since we have many more possible routes. In Figure 5.11, two contour graphs are shown where it can be clearly observed that the domain of cells available to propagate the wave decreases. Figure 5.12 shows the construction of an optimal route from point A to point B using the FMM algorithm.

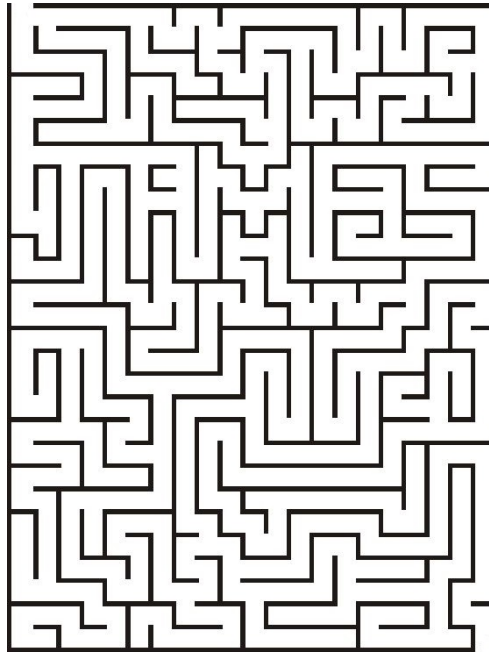


Figure 5.10: Maze image used for Scenario 5

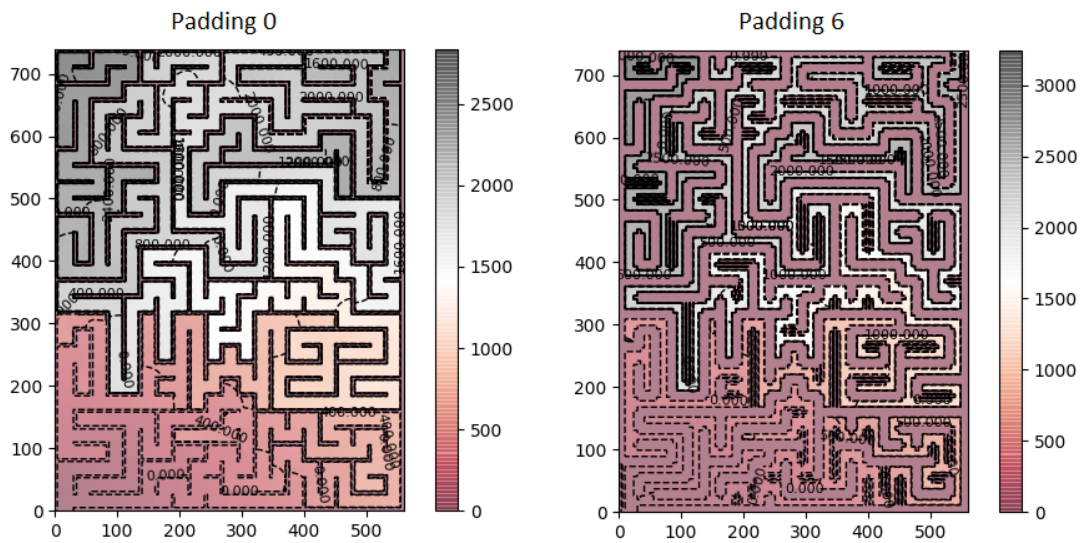


Figure 5.11: 2D Contour graph of the wave expansion with respect to padding metric

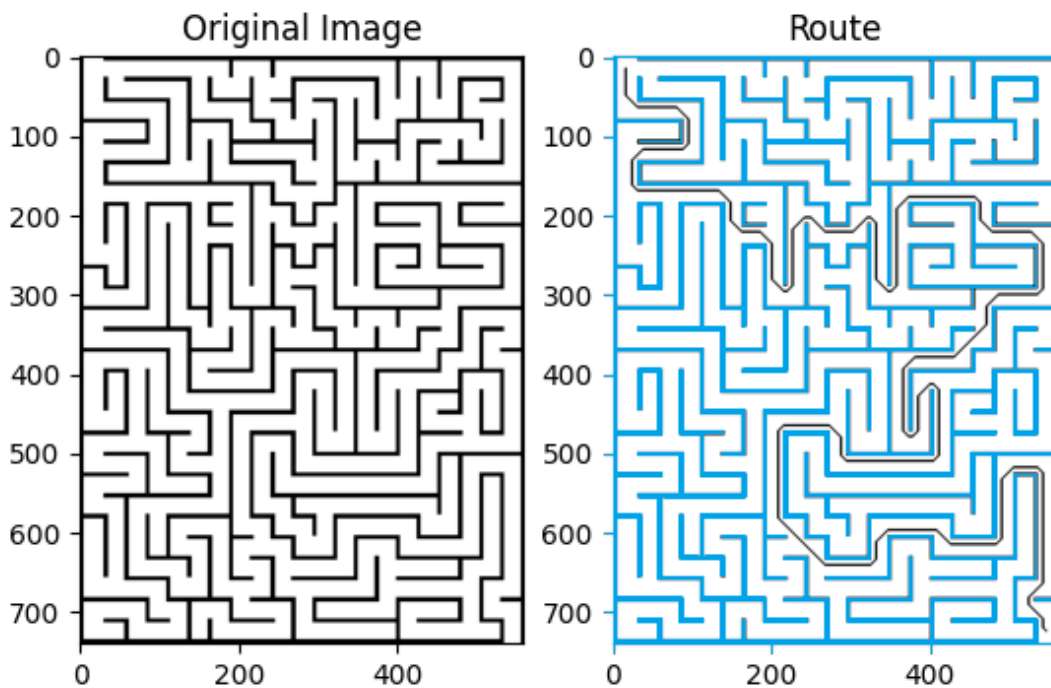


Figure 5.12: Optimal route obtained in Scenario 5

### Real Simulation

This last simulation was carried out in order to show the operation of FMM applied to Path Planning. For this, a map of the City of San Francisco, two reference points, and padding were used as initial parameters of the algorithm. The input parameters were: a starting point (215,14), arrival point (660,600), and padding (5).

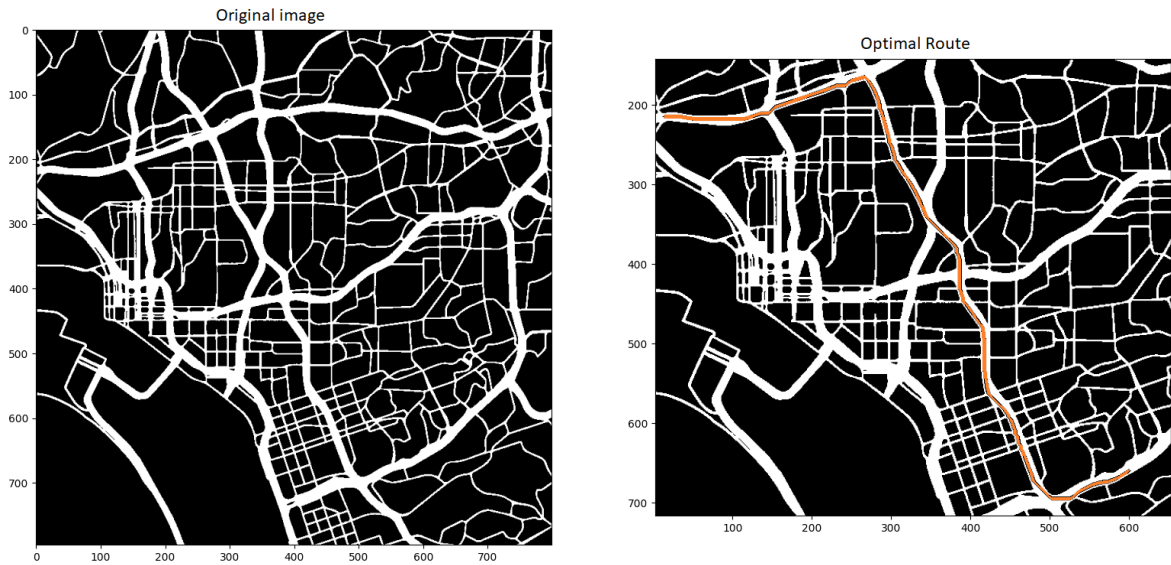


Figure 5.13: Real simulation on the San Francisco map

## 5.2 Evaluation Metrics

The evaluation of OFM (Ordered Fast Marching Method) was carried out through several scenarios, where we change the size of the image, initial point, and the number of obstacles for observing the behavior of OFM and how its execution time varies.

## 5.3 Simulation Results

For a better analysis of the results obtained in the simulations, we have segmented the scenarios into two groups, grouping them by the parameters evaluated in these. The first group covers the first three scenarios and we call it “Image size and initial point”. The second group covers scenarios four and five, we call this group “Image obstacles and padding”. In each group, we analyze the results of all scenarios, as well as a general analysis of the entire group.

### 5.3.1 Image Size and Initial Point

As we can see in Figure 5.14, the results obtained show that for small images, all the algorithms have a similar growth, and the execution time difference for all algorithms is minimal. However, when we increase the size of the image, the algorithms grow too fast compared to OFM, which obtained the shortest execution time. This uncontrolled growth is because as the size of the image increases, the number of items that are saved in the narrow band also increases. Thus, the algorithms have to interact with more data in the narrow band.

In the same way, for the following scenarios, the results are like those of the previous scenario, as we show in figures 5.15 and 5.16. According to the results obtained in the three

scenarios, we see that by changing the point of origin of the wave; the algorithms increase their execution time. This happens because each initial point has a different number of neighbors. When we increase the number of neighbors, the growth of the narrow band also alters. That is, the narrow band grow much faster when we have a greater number of neighbors.

We can conclude that, by increasing the image size and the initial point, the execution time increases rapidly for the FMM, Fibonacci, and Binary FMM algorithms.

The behavior of the algorithms is related to how operations are performed in the narrow band. Remember that for the OFM, the operation that takes the most time is the insertion of a new value. However, in most cases for these scenarios, the execution time of this operation is constant, since the new value to be entered is almost always greater than or equal to the end of the list. On the other hand, the Binary and Fibonacci algorithms depend on the size of the narrow band in different operations, such as eliminating the minimum value and updating the narrow band.

Therefore, the OFM has the shortest execution time when increasing the image size.

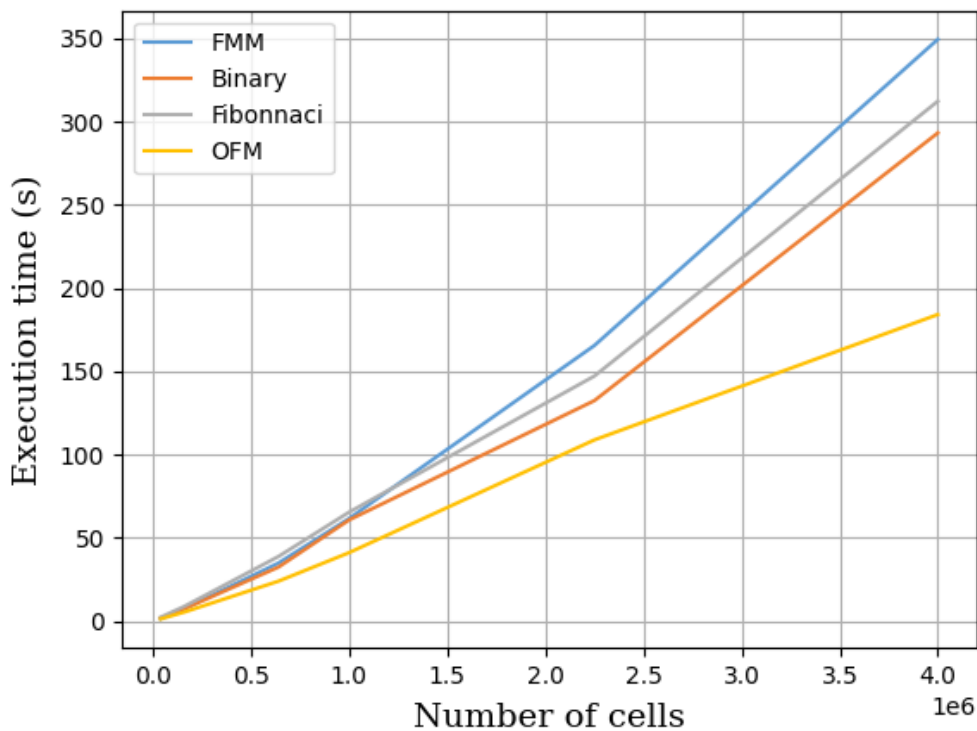


Figure 5.14: Execution time vs. Image size for Scenario 1

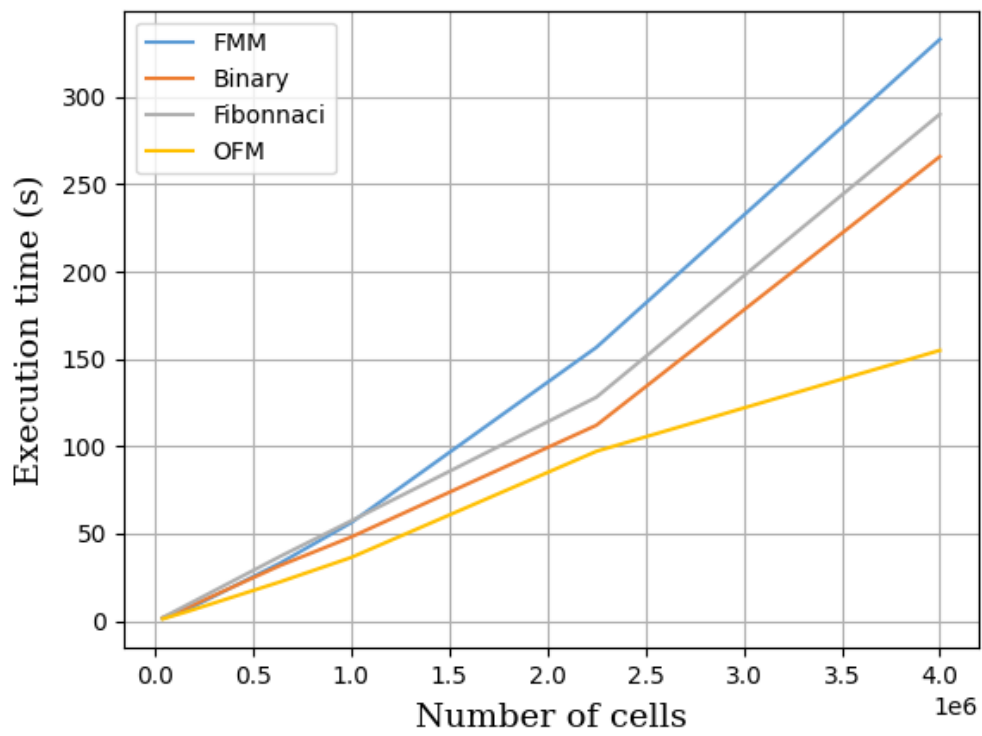


Figure 5.15: Execution time vs. Image size for Scenario 2

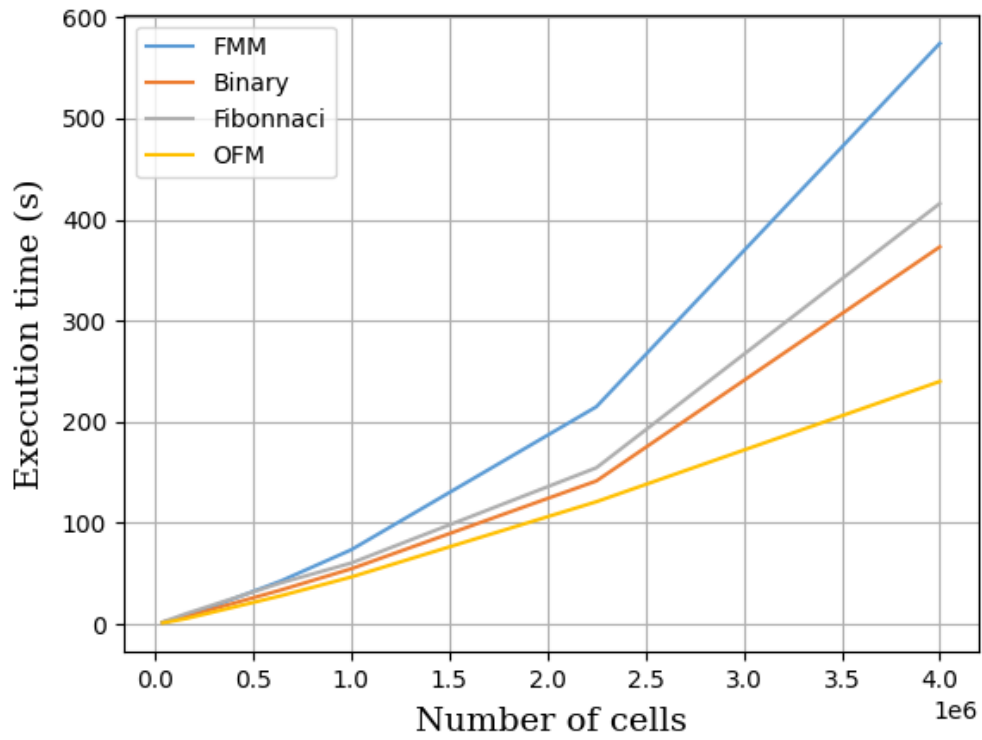


Figure 5.16: Execution time vs. Image size for Scenario 3

### 5.3.2 Image Obstacles and Padding

In this group, we evaluate the execution time regarding the padding metric and the complexity level of the image. The results obtained show that the Fibonacci and Binary algorithms are not efficient when we have images with obstacles. The Fibonacci method had the worst performance, followed by the Binary method.

Figures 5.17 and 5.18 reveal that the level of complexity of the image is not a parameter that varies the execution time of the compared algorithms.

On the other hand, for the two scenarios analyzed in this group, the padding metric influenced the performance of the algorithms, reducing the execution time when we increase this metric. This phenomenon of change happens since by increasing the padding metric, we have reduced the domain of cells to visit. This means that cells that are not far enough from the edges are not visited. In other words, the wave has to go through fewer cells in the input matrix or image.

In conclusion, the padding metric helps both to achieve safe routes and to improve the performance of the algorithms.

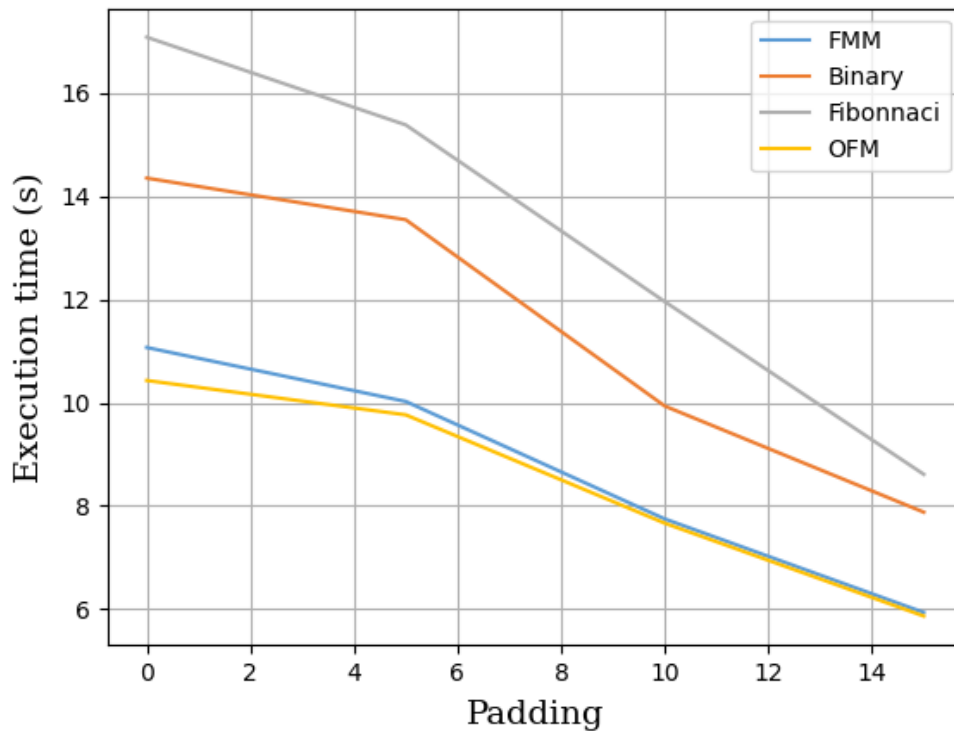


Figure 5.17: Execution time vs. Padding for Scenario 4

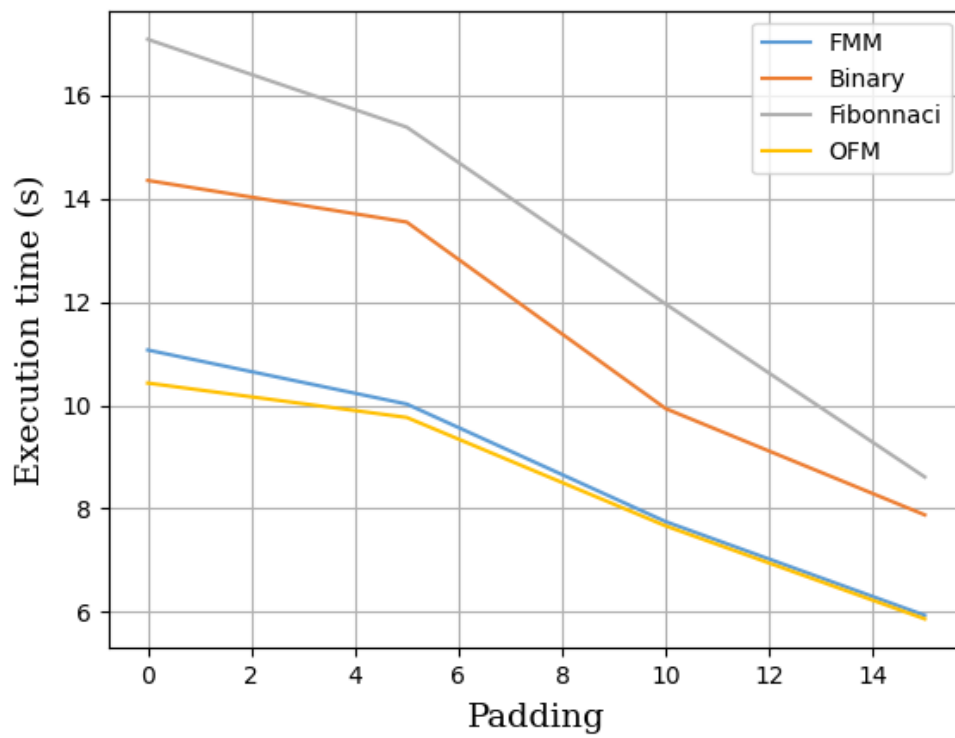


Figure 5.18: Execution time vs. Padding for Scenario 5





# Chapter 6

## Conclusions

### 6.1 Conclusions

Through this work we can conclude the following:

- We have reviewed and implemented three FMM algorithms that were developed based on narrow band handling. The selected algorithms were Binary FMM, Fibonacci FMM, and FMM. All these algorithms are based on the propagation of waves and were evaluated based on the execution time before various changes, such as the size of the image, initial points, obstacles, and padding.
- We have proposed an algorithm based on wave propagation called OFM. OFM is an FMM-based wave propagation method that takes an image, initial point, target point, and padding to avoid collisions with obstacles as input.
- Our proposed OFM method had the shortest run time in all experiments performed. OFM did not show a speed up growth when increasing the size of the images, and also when increasing the padding metric, its execution time decreased. OFM turned out to be optimal since, unlike the compared algorithms, it shows a constant growth with the increase in the size of image.
- In this research, we studied the use of FMM in path planning. Also, we examined many variants of FMM and designed a new variant to improve performance in different scenarios.

In addition, in the experiments carried out in Chapter 4, a series of conclusions have been obtained when interpreting the results. These conclusions are detailed below:

- The complexity of the image (i.e., the number of obstacles) did not affect the execution time for the compared algorithms, was not a metric that affects the execution time.
- The padding metric presented in this work shows good efficiency in all the algorithms used. The padding metric also showed good results, since, besides obtaining optimal and safe routes, this metric also helped to reduce execution time.

- The Fibonacci and Binary FMM turned out to be very good when faced with a small amount of data. However, as the image size increases, these algorithms become obsolete and become very similar to FMM. In addition, in these cases, a large narrow band causes them to lose efficiency quickly. We could also see that with the presence of obstacles, these algorithms did not have the best performance, this is because of the constant change that occurs in the narrow band. However, by increasing the padding metric, these algorithms improved their execution time, having a minimal difference with OFM and FMM.

## 6.2 Future Works

In this work, OFM is compared with wavefront algorithms based on narrow band handling. However, in future work, it would be necessary to compare OFM with other variants of FMM, which gives us another point of view about our algorithm. This would provide us with new perspectives on the same problem.

It will also be necessary for future works to do a deep analysis of the padding metric presented in this work, which will show how to influence it on FMM algorithm. It is necessary to analyze FMM and its variants with other planning algorithms, comparing them with metrics, such as execution time, short routes, and safe routes. A complete OFM implementation for 3D scenarios is also proposed for future work, since FMM, the OFM base algorithm, has a good efficiency in this type of scenario, being one of the best algorithms for this.

We also propose to work with objects such as vehicles, and with maps of large cities that have a speed or cost matrix. For working with objects, the FMM algorithm must be changed, since in that case, an object will have a width and a length, and also the entire object must be taken as our starting point. For the other, we should recreate black and white maps that enable us to have a general perspective of how the wave expands. However, later it will be necessary to add a velocity matrix to the map, which allows us to simulate real environments with greater precision. For this, it will be necessary to implement a function that transforms a color image into a speed matrix.

Parallel programming has opened new doors and gives us new tools to work with algorithms. Therefore, for future work, it is proposed to change FMM to work it in parallel programming and adapt it to our model for this type of programming.

## 6.3 Limitations

The limitations that were had when carrying out this work are:

- Little information about the application of the FMM in path planning.
- Advanced math concepts that delayed the implementation of FMM.
- Few good quality images of cities.
- One of the major limitations was the simulations, since it was very difficult to get black and white images of real maps.

# Bibliography

- [1] D. C. Kozen, *The design and analysis of algorithms*. Springer Science & Business Media, 1992.
- [2] J. Zhu and J. Sethian, “Projection methods coupled to level set interface techniques,” *J. Comput. Phys.*, vol. 102, no. 1, p. 128–138, Sep. 1992. [Online]. Available: [https://doi.org/10.1016/S0021-9991\(05\)80011-7](https://doi.org/10.1016/S0021-9991(05)80011-7)
- [3] J.-J. Xu, Z. Li, J. Lowengrub, and H. Zhao, “A level-set method for interfacial flows with surfactant,” *Journal of Computational Physics*, vol. 212, pp. 590–616, 03 2006.
- [4] C. Hoguea, B. Murray, and J. Sethian, “Computational modeling of solid tumor evolution via a general cartesian mesh/level set method.” 2005.
- [5] —, “Simulating complex tumor dynamics from a vascular to vascular growth using a general level-set method.” pp. 86–134, 2006.
- [6] J. Gómez, A. Lumbier, S. Garrido, and L. Moreno, “Planning robot formations with fast marching square including uncertainty conditions,” *Robotics and Autonomous Systems*, vol. 61, pp. 137–152, 02 2013.
- [7] M. K. Cameron, S. B. Fomel, and J. A. Sethian, “Seismic velocity estimation from time migration,” *Inverse Problems*, vol. 23, no. 4, pp. 1329–1369, may 2007. [Online]. Available: <https://doi.org/10.1088/0266-5611/23/4/001>
- [8] R. Malladi and J. A. Sethian, “Image processing via level set curvature flow,” *Proceedings of the National Academy of Sciences*, vol. 92, no. 15, pp. 7046–7050, 1995. [Online]. Available: <https://www.pnas.org/content/92/15/7046>
- [9] S. Osher and J. A. Sethian, “Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations,” *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0021999188900022>
- [10] P. E. Hart, N. J. 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.
- [11] M. Jones, J. Baerentzen, and M. Sramek, “3d distance fields: a survey of techniques and applications,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581–599, 2006.

- [12] N. Deo and C. Pang, “Shortest-path algorithms: Taxonomy and annotation,” *Networks*, vol. 14, pp. 275 – 323, 06 1984.
- [13] J. Tsitsiklis, “Efficient algorithms for globally optimal trajectories,” *Proceedings of the IEEE Conference on Decision and Control*, vol. 2, pp. 1368–1373, 01 1994.
- [14] J. Mitchell and J. Papadimitriou, “Planning shortest paths,” *SIAM Conference*, pp. 15–19, 01 1985.
- [15] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotic Research - IJRR*, vol. 30, pp. 846–894, 06 2011.
- [16] S. Wismath, “Computing the full visibility graph of a set of line segments\*\*this research was supported by n.s.e.r.c. grant og-pin 007.” *Information Processing Letters*, vol. 42, no. 5, pp. 257–261, 1992. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/002001909290033R>
- [17] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
- [18] A. Stentz, “The focussed d\* algorithm for real-time replanning,” in *IJCAI*, 1995.
- [19] S. Koenig and M. Likhachev, “D\*lite.” 01 2002, pp. 476–483.
- [20] J. Sethian, “A fast marching level set method for monotonically advancing fronts.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 93 4, pp. 1591–5, 1996.
- [21] S. LaValle, *Planning Algorithms*, 01 2006.
- [22] S. LaValle and J. Kuffner, “Randomized kinodynamic planning.” vol. 20, 01 1999, pp. 473–479.
- [23] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, “Analysis of probabilistic roadmaps for path planning,” *Robotics and Automation, IEEE Transactions on*, vol. 14, pp. 166 – 171, 03 1998.
- [24] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” 06 2010.
- [25] J. Gammell, S. Srinivasa, and T. Barfoot, “Batch informed trees (bit\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs,” in *Proceedings of (ICRA) International Conference on Robotics and Automation*, IEEE, Ed., May 2015, pp. 3067 – 3074.
- [26] O. Arslan and P. Tsiotras, “Use of relaxation methods in sampling-based algorithms for optimal motion planning,” in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2421–2428.

- [27] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.
- [28] 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, no. 4, pp. 111–120, 2013.
- [29] L. Yatziv, A. Bartsaghi, and G. Sapiro, "O(n) implementation of the fast marching algorithm," *Journal of Computational Physics*, vol. 212, pp. 393–399, 03 2006.
- [30] J. V. Gómez, D. Álvarez, S. Garrido, and L. Moreno, "Fast methods for eikonal equations: An experimental survey," *IEEE Access*, vol. 7, pp. 39 005–39 029, 2019.
- [31] S. Gayathridevi, "An abstract to calculate big o factors of time and space complexity of machine code," 07 2011.
- [32] C. A. Monje, S. Garrido, L. Moreno, and C. Balaguer, "Uavs formation approach using fast marching square methods," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 5, pp. 36–46, 2020.
- [33] M. Fredman and R. Endre, *Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms*, 2004.
- [34] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *25th Annual Symposium on Foundations of Computer Science, 1984.*, 1984, pp. 338–346.
- [35] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, 01 2001.
- [36] R. E. Tarjan, *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983. [Online]. Available: <https://epubs.siam.org/doi/abs/10.1137/1.9781611970265>
- [37] S. Kim, "The group marching method: An o(n) level set eikonal solver," vol. 19, 04 2000.
- [38] J. Yang, "An easily implemented, block-based fast marching method with superior sequential and parallel performance," *SIAM Journal on Scientific Computing*, vol. 41, pp. C446–C478, 01 2019.
- [39] W. Shuai, Z. Rong-cai, and Y. Yuan, "Recovery methodology to avoid loss for slp," *Procedia Engineering*, vol. 15, p. 204–208, 12 2011.
- [40] J. William, *Java Au Naturel*, 05 2004. [Online]. Available: <https://cs.ccsu.edu/~jones/book.htm>
- [41] T. Seidl and J. Enderle, *Binary Search*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 5–11. [Online]. Available: [https://doi.org/10.1007/978-3-642-15328-0\\_1](https://doi.org/10.1007/978-3-642-15328-0_1)

- [42] Python, <https://www.python.org/doc/essays/blurb/1>.
- [43] Numpy - Python Library, <https://numpy.org/devdocs/user/whatisnumpy.html>.
- [44] Math - Python Library, <https://docs.python.org/3/library/math.html>.
- [45] Matplotlib - Python Library, <https://matplotlib.org/stable/index.html>.
- [46] OpenCV - Python Library, <https://docs.opencv.org/4.5.2/d1/dfb/intro.html>.