A COMPARATIVE STUDY OF QUADTREE DECOMPOSITION AND CONSTRAINED DELAUNAY TRIANGULATION USING MDP AND ARTIFICIAL POTENTIAL FIELD BASED PATH PLANNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BAŞER KANDEHİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JULY 2019

Approval of the thesis:

A COMPARATIVE STUDY OF QUADTREE DECOMPOSITION AND CONSTRAINED DELAUNAY TRIANGULATION USING MDP AND ARTIFICIAL POTENTIAL FIELD BASED PATH PLANNING

submitted by **BAŞER KANDEHİR** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Afşar Saranlı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU _____

Prof. Dr. Ömer Morgül
Electrical and Electronics Engineering, Bilkent University _____

Assist. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Ahmet Buğra Koku
Mechanical Engineering, METU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Başer Kandehir

Signature :

ABSTRACT

A COMPARATIVE STUDY OF QUADTREE DECOMPOSITION AND CONSTRAINED DELAUNAY TRIANGULATION USING MDP AND ARTIFICIAL POTENTIAL FIELD BASED PATH PLANNING

Kandehir, Başer
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Afşar Saranlı

July 2019, 60 pages

The general purpose of a mobile robot is moving from one point to another and perform certain tasks. To do so, it first computes a motion strategy and then tries to execute it. This is not feasible most of the time unless uncertainties of the real world is taken into account. Markov decision processes (MDPs) provide a mathematical system to deal with uncertainties of planning and execution stages. MDPs require finite set of states. Therefore, continuous space of the real world must be discretized. In this study, two widely used space discretization methods, namely quadtree decomposition (QD) and constrained Delaunay triangulation (CDT), are compared in terms of path length, travel time, two safety measures, planning time, number of iterations, and number of states to find out which one of these discretization methods is better in the context of MDP and planar motion planning. MDP framework is used as high-level planner, and value iteration is used to obtain the optimal policy. Then, artificial potential field (APF) method is used for low-level execution. Results showed that QD and CDT are both suitable in the context of MDP and planar path planning with APF. QD results in longer paths but requires less travel time whereas CDT results in

shorter paths but requires more travel time. QD and CDT perform almost equally in terms of safety. QD has clear disadvantages compared to CDT in terms of planning time, number of iterations, and number of states. QD and CDT might be preferable for different applications. Thus, it is best to optimize parameters for preferred metrics on a specific problem.

Keywords: Quadtree decomposition, constrained Delaunay triangulation, Markov decision process, artificial potential field, space discretization, motion planning

ÖZ

DÖRDÜN AĞAÇ AYRIŞTIRMASI VE KISITLANDIRILMIŞ DELAUNAY ÜÇGENLEŞTİRMESİNİN MARKOV KARAR SÜRECİ VE YAPAY POTANSİYEL ALAN TABANLI YOL PLANLAMASI KULLANILARAK KARŞILAŞTIRMALI BİR ÇALIŞMASI

Kandehir, Başer
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Afşar Saranlı

Temmuz 2019 , 60 sayfa

Bir mobil robotun genel amacı bir noktadan diğerine hareket etmek ve belli görevleri yerine getirmektir. Bunu yapmak için, önce bir hareket planı hesaplar ve daha sonra bunu uygulamaya çalışır. Gerçek dünyanın belirsizlikleri dikkate alınmadıkça bu çoğu zaman uygulanabilir değildir. Markov karar süreçleri (MKSler) hem planlama hem uygulama aşamasında bu belirsizliklerin üstesinden gelmek için matematiksel bir sistem sağlar. MKSler sonlu sayıda durum gerektirir. Bundan dolayı, gerçek dünyanın devamlı uzayı ayrıklaştırılmalıdır. Bu çalışmada, dördün ağaç ayrıştırması (DAA) ve kısıtlandırılmış Delaunay üçgenleştirmesi (KDÜ) olarak adlandırılan iki sık kullanılan uzay ayrıklaştırma metodu, MKS ve düzlemsel hareket planlama bağlamında bu ayrıklaştırma metodlarından hangisinin daha iyi olduğunu anlamak için, yol uzunluğu, yolculuk zamanı, iki güvenlik ölçüsü, planlama zamanı, iterasyon sayısı, ve durum sayısı cinsinden karşılaştırılıyor. MKS sistemi yüksek düzey planlayıcı olarak kullanılıyor, ve değer iterasyonu en iyi planı elde etmek için kullanılıyor. Daha sonra, yapay potansiyel alan (YPA) metodu alt düzey uygulama için kulllanılıyor. So-

nuçlar gösterdi ki, hem DAA hem KDÜ, MKS ve YPA ile düzlemsel hareket planlama bağlamında uygundur. DAA daha uzun yollarla sonuçlanıyor ama daha az yolculuk zamanı gerektiriyorken KDÜ daha kısa yollarla sonuçlanıyor ama daha fazla yolculuk zamanı gerektiriyor. DAA ve KDÜ güvenlik açısından neredeyse aynı davranıyor. DAA, KDÜ ile karşılaştırıldığında planlama zamanı, iterasyon sayısı, ve durum sayısı cinsinden belirgin dezavantajlara sahip. DAA ve KDÜ farklı uygulamalar için tercih edilebilir. Bundan dolayı, en iyisi parametreleri belli bir problem üzerinde tercih edilen ölçüler için optimize etmektir.

Anahtar Kelimeler: Dördün ağaç ayrıştırması, kısıtlandırılmış Delaunay üçgenleştirmesi, Markov karar süreci, yapay potansiyel alan, uzay ayrıklaştırması, hareket planlaması

To my cat friend Charlie

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Afşar Saranlı for his guidance, support and valuable feedback whenever I needed. Then, I would like to thank Uluç Saranlı for giving me the opportunity to work in his project and being an example of diligence and persistence. I also would like to thank Mustafa Mert Ankaralı, Yiğit Yazıcıoğlu and Gökhan Gültekin.

I am grateful that I had the privilege to work with intelligent, kind and interesting people in ATLAS and RoLab. Many hours spent together with these people taught me a lot. I thank Osman Kaan Karagöz for always challenging my beliefs, being a great seatmate, and his critical suggestions about optimization. I thank Ferhat Gölbol for being who he is (extremely clever, patient, great listener, easygoing person) and providing his suggestions, and help whenever I needed. I thank Jean Piere Demir for being a great and interesting friend, labmate and introducing me to the various .io games. I thank Görkem Seçer for enlightening me with his knowledge, friendship, sincerity. I thank Halil İbrahim Uğurlu for all the .io games we played together and his helpful suggestions. Although we did not spent much time during MS, I thank Seyit Yiğit Sızlayan for being a great friend, brother and a mentor to me. I thank my friend Alpay Ünal for always being ready to join whenever I eat, play, and work, and fighting side by side with me in .io games. I also thank my valuable labmates and friends, Sait Sovukluk, Abdullah Cem Önem, Sinan Şahin Candan, Nurullah Gülmüş, Lütfullah Tomak, Ayşe Deniz Duyul, and Merve Özen.

Last but not least, I would like to thank my family, my cat friend Charlie who literally has always been there for me, and my favorite band MUSE for making great music. I thank the researchers who found *Pomodoro technique* that helped me during hard times when writing this thesis, my computer and its peripherals for doing their job, the nature or the creator for creating me, the cumulative knowledge of humanity, otherwise I would be hunting and foraging, and the people I never had the chance to meet.

This thesis was also supported by TÜBİTAK project 117E106.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

QD	Quadtree Decomposition
DT	Delaunay Triangulation
CDT	Constrained Delaunay Triangulation
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
APF	Artificial Potential Field
DFS	Depth First Search
PSO	Particle Swarm Optimization

CHAPTER 1

INTRODUCTION

Motion planning is one of the main topics in the field of robotics. The purpose is to find a motion strategy that would move the robot from one point to another under certain constraints. To build a motion strategy that can safely be executed by the robot, it is crucial to consider uncertainties of the real world. Markov decision processes (MDPs) provide a mathematical system to deal with uncertainties of planning and execution stages. MDPs require finite set of states thus, continuous space needs to discretized. Quadtree decomposition (QD) and constrained Delaunay triangulation (CDT) are two of the widely used space discretization methods. However, in the context of MDP and planar motion planning, which of these methods perform better, remains an unanswered question. In this study, we consider a point mass with holonomic capabilities in an environment with various polygonal obstacles and using MDP framework as high-level planner and artificial potential field (APF) method for low-level execution, we compare QD and CDT with least possible states.

1.1 Literature Survey

In this section, we give related work about QD, CDT, APF and MDP.

Discretization of continuous space has been studied in robotics for many decades [34], [35]. In this study, we focus on QD and CDT. Finkel and Bentley named quadtrees in 1974 [3]. QD has been used in various areas including image processing [4], databases [5], and geographic information systems [6]. Delaunay triangulation (DT) is named after Boris Delaunay for his work in 1934 [7]. There are various Delaunay refinement algorithms e.g. Ruppert's algorithm [8], [9]. DT can be forced to satisfy

certain constraints, then it is called constrained Delaunay triangulation (CDT) [10], [11].

The APF method was originally proposed by Oussama Khatib [12]. APF offers a fast and effective way to solve the motion planning problem. APF and its modified versions has been commonly used: [13] describes a path planning method for robotic manipulators and mobile robots using APF around configration-space obstacles, [14] integrates APF with simulated annealing to mobile robots, [16] uses APF for collision-free path planning in 3D environment for unmanned aerial vehicles (UAVs). However, there is a local minima problem associated with APFs. Various methods have been suggested for solving the local minima problem: [15] uses virtual obstacles to escape local minima,[18] uses dynamic state agents to escape local minima, [19] uses input to state stability property of multistable system to avoid local minima.

MDPs have been used in various fields including robotics, economics, automatic control and manufacturing since 1950s [21], [22]. MDPs maximize an objective function to obtain a control policy over all states rather than finding a single trajectory solution [23], [24]. The partially observable Markov decision process (POMDP), is a generalization of MDP. Their complexity restricts their usage to relatively simple problems [25], [26]. Computation of an approximate solution is one of the methods for reducing the complexity [27]. However, in practice, MDP is generally used instead of POMDP in more complex planning problems [28]. Still, realistic problems require large number of states, thus it is necessary to reduce the number of states considering complexity of MDP [29]. That is why we need efficient space discretization methods like QD and CDT.

[30] uses MDP-based planning where QD is used to discretize robot's state space. Actions namely, Dubins actions, are chosen considering kinematic constraints of a mobile robot with wheels. QD and Dubins actions results in an efficent and robuts motion planner. In [31], they extend [30], and describe a navigation approach based on MDP and Markov localization. This paper also focuses on experimental aspects about Markov techniques by using learning to obtain transition function and the sensor model. Experimental work is done on a real robot for realization and demonstra-

tion of the approach.

[32] uses bounded-parameter Markov decision processes (BMDPs) and DT for fast reconfiguration of local policies. Thus, the policy can be updated if the reward function changes. This framework is tested on various online tasks, and shown that policy reconfiguration can be done in a few seconds. In [33], they again use BMDPs and DT, and consider action uncertainty in offline stage, and upon change or discovery of environment, optimal policy can be recomputed within seconds.

1.2 Contribution of the Thesis

In the context of MDP and motion planning, QD has been used in [30], [31], whereas DT has been used in [32], [33]. However, the reason for using these methods has not been justified, thus we do not know if using one or the other would change the outcome of these works dramatically. In this thesis, we try to answer if QD or CDT performs better in the context of MDP and planar motion planning.

1.3 Organization of the Thesis

The orginization of thesis is as follows: in Sec. 2, we give the background on QD, CDT, MDP, and APF; in Sec. 3, we describe our approach towards comparing QD and CDT; in Sec. 4, we explain parameter selection then give results, and discussion; in Sec. 5, we conclude the thesis with a summary and description of possible future work.

CHAPTER 2

BACKGROUND

In this section, the background on following topics will be given: discretization of continuous space using Quadtree decomposition (QD) and Delaunay triangulation(DT); Markov decision process (MDP) with its formal definition, policies, value functions and computing the optimal policy; artificial potential field (APF) with attractive potential, repulsive potential and local minima problem.

2.1 Discretization of Continuous Space

MDP framework requires finite number of states. Using uniform decomposition methods are computationally expensive because they require large number states thus not feasible for maps having large sizes. We prefer more computationally efficient methods like QD and DT which will be described in this section.

2.1.1 Quadtree Decomposition

In a quadtree structure each node has 4 sub-nodes. QD is a hierarchical state discretization method used to partition 2D space. It recursively divides the environment into four regions. Regions which include both obstacle and free space (partially occupied) are subdivided until a specified resolution is reached. By doing so, a map with QD has a finer resolution around obstacles and coarser resolution in the free space. Thus, the number of states and computation time in motion planning is much less compared to those of uniform decomposition methods. QD can be extended to 3D space, then it is called octree decomposition.

Fig. 2.1 shows a binary image and its QD.

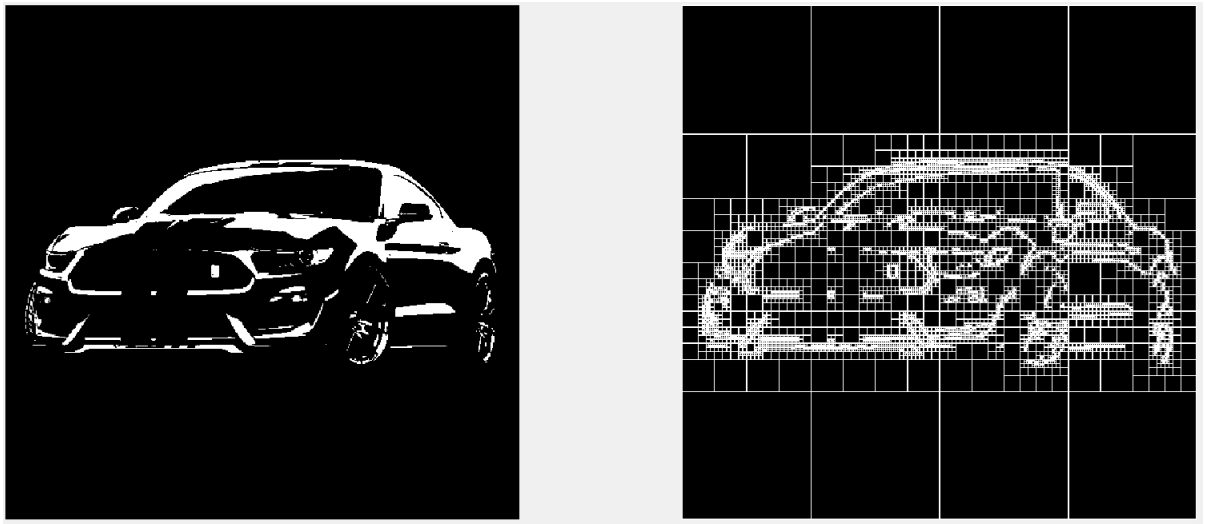


Figure 2.1: QD as applied to binary image quantization

2.1.2 Delaunay Triangulation

Delaunay triangles satisfy the empty circumcircle criterion which ensures circumcircle[1] of each triangle does not include any other point than the vertices of that triangle. Because of this property, Delaunay triangles are not sliver[2] triangles. Also, points are connected in nearest-neighbor manner in Delaunay triangulation (DT). Because of these advantageous geometric properties, they have been used widely in various applications.

DTs do not exist if the points are on the same line. And they are not unique if there are four or more points on the same circumcircle. DTs can be extended to three and higher dimensions.

Certain constraints can be defined and DT can be forced to satisfy these constraints. In this case, it is called constrained Delaunay triangulation (CDT) [10] [11]. Since empty circumcircle criterion cannot always be satisfied with the defined constraints, a CDT is not a DT. In this study, we need to specify constraints for obstacles since

[1] Circumcircle of a polygon is the circle passing through vertices of that polygon.

[2] A sliver triangle covers much less area compared to its circumcircle. Thus, some of its internal angles are very small.

triangulations inside obstacles are unnecessary and triangles should only contain the free space. Therefore, we use CDT in this study.

Fig. 2.2 shows DT of 100 random points.

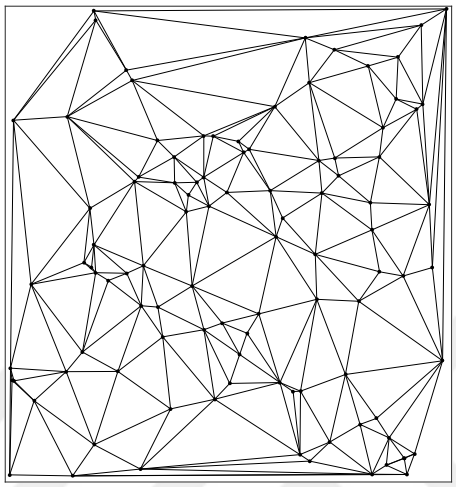


Figure 2.2: DT of 100 random points.

2.2 Markov Decision Processes

In an ideal world, the effect of control actions are deterministic. However, this is not the case in practice, since there is uncertainty in action effects, as well as in perception. Considering the uncertainty in action effects, Markov decision processes (MDPs) provide a framework for optimal decision making considering uncertainty.

There are two assumptions to be made:

- *Markov assumption*: Transition probability to the next state only depends on the current state and current action, therefore independent of all the past states and actions.

- *Full observability assumption*: Unlike partially observable Markov decision processes (POMDPs), MDPs assume full observability of states at all times.

2.2.1 Formal MDP Definition

As described in [32] and [30] an MDP is defined by 4 elements $\{S, A, P, R\}$ where

- S represents a finite set of states. In this work, states are obtained after discretization of continous space (detailed in Sec. 3.2.1).
- A represents a finite set of actions. Actions allow transitioning between states.
- $P : S \times A \times S \rightarrow [0, 1]$ represents the transition probability function where $P(s_i, a, s_j)$ gives the transition probability of going from state s_i to state s_j when action a is executed.
- $R : S \rightarrow \mathbb{R}$ represents the rewards where $R(s)$ gives the reward for staying in state s.

As given in [1], the graphical representation of MDP is shown in Fig. 2.3.

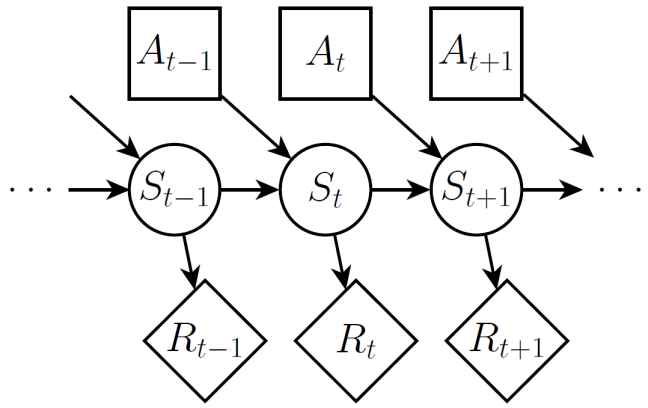


Figure 2.3: Graphical model representation of MDP from [1]. S_t, A_t, and R_t represents the state, action and reward at time t respectively.

2.2.2 Policies and Value Functions

There are a few definitions to be made to understand how MDPs provide optimal decisions under uncertainty. For these definitions we refer to [23] and [1].

Control policy, or simply *policy* is a mapping from all the past observations and actions to current action. It can be denoted as follows:

$$\pi : z_{1:t-1}, a_{1:t-1} \rightarrow a_t \tag{21}$$

where π is policy, $z_{1:t-1}$ is all the past observations, $a_{1:t-1}$ is all the past actions, and a_t is the current action.

Considering the full observability assumption, *policy* becomes a mapping from states to actions as follows:

$$\pi : s_t \rightarrow a_t \tag{22}$$

where π is the policy, s_t is the current state, a_t is the current action.

As described in [1], *value function* associated with a policy, gives the expected sum of discounted rewards for that policy. It can be written as:

$$V^{\pi}(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, a_t = \pi(s_t), s_{t+1} | s_t, a_t \sim P\right] \tag{23}$$

where $\gamma < 1$ represents the *discount factor*.

The equation defining the value function recursively, which is called the *Bellman equation* which is written as follows:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s' | s, \pi(s)) V^{\pi}(s') \tag{24}$$

Optimal policy is the policy which gives the maximum value for each state. As described in [1], the optimal value function can be found using *Bellman optimality equation* as follows:

$$V^{*}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' | s, a) V^{*}(s') \tag{25}$$

And optimal policy becomes the actions that achieve this value function:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s,a)V^*(s') \tag{26}$$

2.2.3 Computing the Optimal Policy

There are two approaches to compute the optimal policy. These are value iteration and policy iteration.

2.2.3.1 Value Iteration

The algorithm for value iteration is given as follows [1]:

1. First, the value function is initialized as zeros for all states.

$$\hat{V}(s) \leftarrow 0, \forall s \in S \tag{27}$$

2. Then, the values are updated according to Bellman optimality equation:

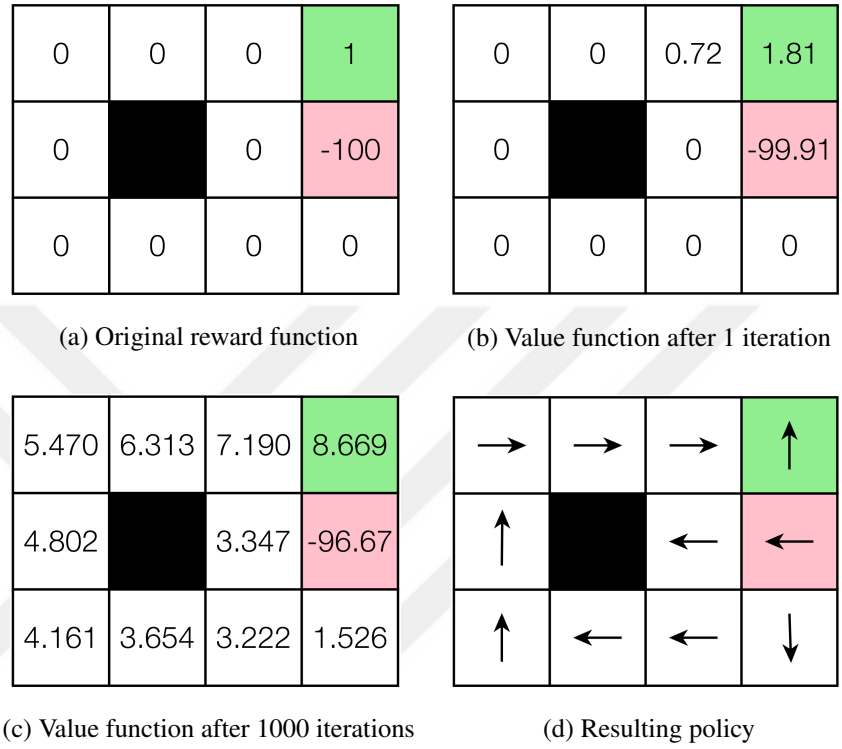
$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\hat{V}(s'), \forall s \in S \tag{28}$$

Value iteration is guaranteed to converge to its optimal value. The proof can be found in [1].

Value iteration is illustrated with a simple example in Fig. 2.4. Fig. 2.4a shows the original reward function. In this reward function, goal state is represented with value 1, and shown as green, whereas "a bad state" where robot would not want to be is represented with value -100, and shown as red. All the other states has zero value. This kind of simple reward function is good enough for value iteration. Transition probability function is chosen so that when the robot moves in a direction it reaches that state with 0.8 probability and perpendicular states with 0.1 probability. If robot takes an action that would make it crash into a wall, it stays where it is.

Using the reward function and $\gamma = 0.9$, value iteration is run for 1000 iterations. Fig. 2.4b shows the value iteration after 1 iteration, and Fig. 2.4c shows the value function

after 1000 iterations. As can be seen, the goal state has the highest value and the "bad state" has the lowest value. Using this value function and Eq. 26, the optimal policy can be found as shown in Fig. 2.4d.



(a) Original reward function

(b) Value function after 1 iteration

(c) Value function after 1000 iterations

(d) Resulting policy

Figure 2.4: Illustration of value iteration from [1]. (a) Original reward function. (b) Value function after 1 iteration. (c) Value function after 1000 iterations. (d) Resulting policy.

2.2.3.2 Policy Iteration

The algorithm for policy iteration is as follows [1]:

1. Initialize the policy $\hat{\pi}$ randomly.
2. Compute the value of the policy V^{π} as follows:

$$V^{\pi} = (I - \gamma P^{\pi})^{-1} r \tag{29}$$

where γ is the discounting factor, P^{π} is the transition probability matrix, r is the reward vector.

3. Compute policy π with respect to its value V^{π} as follows:

$$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \tag{210}$$

4. If policy π does not change in the last iteration, optimal policy is said to be found. Otherwise, return to step 2.

Policy iteration is illustrated in Fig. 2.5 with the same example given in value iteration. Fig. 2.5a shows the original reward function. After 1 iteration, resulting policy value becomes as in Fig. 2.5b. And after 3 iterations, policy value becomes as given in Fig. 2.5c. Then resulting policy is obtained as in Fig. 2.5d. An important remark here is that even at the first iteration, policy value is quite close to exact value function and after 3 iterations it reaches the exact value function. Thus, it can be said that policy iteration requires much less iterations than value iteration, but each iteration takes more time since it requires solving a linear system.

(a) Original reward function

(b) Policy value after 1 iteration

(c) Policy value after 3 iterations

(d) Resulting policy

Figure 2.5: Illustration of policy iteration from [1]. (a) Original reward function. (b) Policy value after 1 iteration. (c) Policy value after 3 iterations. (d) Resulting policy.

2.3 Artificial Potential Field Method

The artificial potential field (APF) method was originally proposed by Oussama Khatib [12] for avoiding obstacles real-time using mobile robots and manipulators. It is based on a simple idea that obstacles repel the robot and the goal attracts it, thus combination of repulsive forces with attractive forces move the robot to the goal while preventing a collision with obstacles.

As described in [2], combining the attractive and repulsive potentials, total energy of the robot can be written as:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{211}$$

where $U_{att}(q)$ is the attractive potential, and $U_{rep}(q)$ is the repulsive potential.

The robot minimizes this potential energy by moving towards the negative gradient of it, $-\nabla U(q)$.

2.3.1 The Attractive Potential

Attractive potential should be increasing as the distance from the goal configuration increases [2]. The simplest attractive potential is the conic potential. Its energy and gradient of this energy is as follows:

$$U_{conic}(q) = \zeta d(q, q_{goal}) \tag{212}$$

$$\nabla U_{conic}(q) = \frac{\zeta}{d(q, q_{goal})}(q - q_{goal}) \tag{213}$$

Implementing conic potential with gradient descent can cause "chattering" problems, thus quadratic potential is preferable. Its energy and gradient of this energy is as

follows:

$$U_{quad}(q) = \frac{1}{2}\zeta d^2(q, q_{goal}) \tag{214}$$

$$\nabla U_{quad}(q) = \zeta(q - q_{goal}) \tag{215}$$

The problem with the quadratic potential is that if initial configuration is far away from the goal, it can produce a very large velocity. Because of this, combining conical and quadratic potential gives the best results. Quadratic potential attracts the robot when it is close to the goal, and conic potential attracts it when it is far away from the goal. The resulting attractive potential and its gradient becomes [2]:

$$U_{att}(q) = \begin{cases} \frac{1}{2}\zeta d^2(q, q_{goal}), & d(q, q_{goal}) \leq d^*_{goal} \\ d^*_{goal}\zeta d(q, q_{goal}) - \frac{1}{2}\zeta(d^*_{goal})^2, & d(q, q_{goal}) > d^*_{goal} \end{cases} \tag{216}$$

$$\nabla U_{att}(q) = \begin{cases} \zeta(q - q_{goal}), & d(q, q_{goal}) \leq d^*_{goal} \\ \frac{d^*_{goal}\zeta(q - q_{goal})}{d(q, q_{goal})}, & d(q, q_{goal}) > d^*_{goal} \end{cases} \tag{217}$$

where d^*_{goal} is the distance from the goal where potential function changes between conic and quadtratic potentials.

2.3.2 The Repulsive Potential

Repulsive potential should increase as the robot gets closer to the obstacle in order to avoid colliding with obstacles. We choose to define repulsive potential functions for individual obstacles and combine them in order to prevent oscillatory paths and ease the detection of local minima. The repulsive potential for each obstacle and its gradient is defined as follows [2]:

$$U_{rep_i}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{d_i(q)} - \frac{1}{Q^*_i})^2, & d_i(q) \leq Q^*_i \\ 0, & d_i(q) > Q^*_i \end{cases} \tag{218}$$

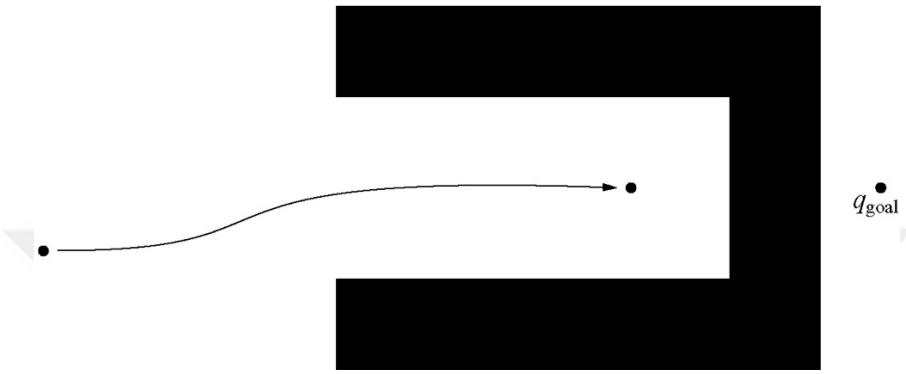
$$\nabla U_{rep_i}(q) = \begin{cases} \eta(\frac{1}{Q_i^*} - \frac{1}{d_i(q)})\frac{1}{d_i^2(q)}\nabla d_i(q), & d_i(q) \leq Q_i^* \\ 0, & d_i(q) > Q_i^* \end{cases} \tag{219}$$

where $d_i(q)$ is the distance to the individual obstacle, Q_i^* is the threshold distance from the robot where obstacles are ignored after, η is the gain of the repulsive gradient. We choose $Q_i^* = Q^*$ for all the obstacles. The total repulsive potential can be computed as follows:

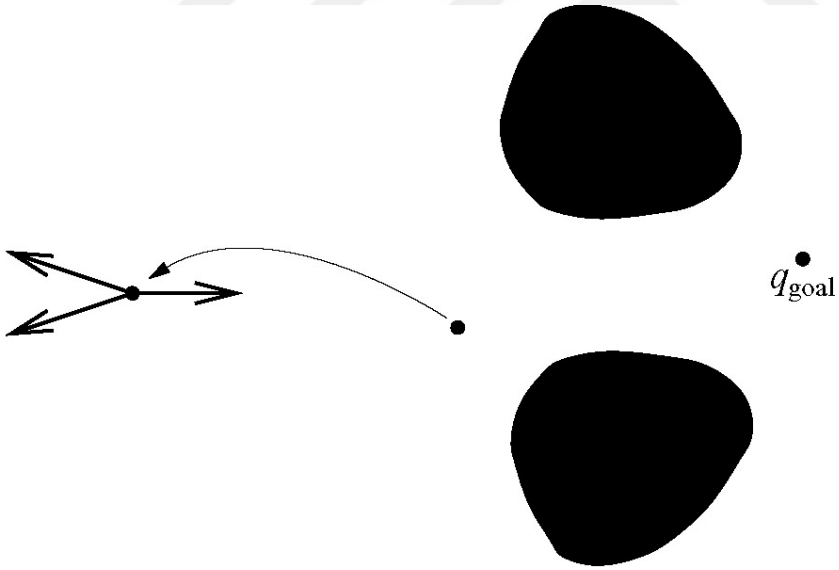
$$U_{rep}(q) = \sum_{i=1}^{n} U_{rep_i}(q) \tag{220}$$

2.3.3 Local Minima Problem

The standard APF method is not complete because of the local minima problem. The robot might get stuck when attractive and repulsive forces cancel each other. There are various methods to solve the local minima problem [17], [18], [19], [20]. Fig. 2.6 shows examples for local minima problem for both concave and convex obstacles from [2].



(a) Local minima problem for a concave obstacle



(b) Local minima problem for convex obstacles

Figure 2.6: Examples for local minima problem for both concave and convex obstacles from [2].

CHAPTER 3

APPROACH

In Sec. 2, we have given background on Quadtree decomposition (QD), Delaunay triangulation (DT), Markov decision process (MDP) and artificial potetial field (APF). In this section, we describe our approach towards comparing QD and constrained Delaunay triangulation (CDT) in the context of MDP and planar motion planning. Firstly, we discretize the maps with polygonal obstacles using QD and CDT. Secondly, we define the MDP framework to be used by defining states, actions, transition function and reward function. Then, we compute the optimal policy to obtain a high-level plan. Then, we use APF method as a low-level controller.

3.1 Discretization of Continuous Space: Quadtree Decomposition and Constrained Delaunay Triangulation

We discretize maps with polygonal obstacles using QD and CDT. We use *qtdecomp* function of MATLAB for QD. However, this function requires a grayscale image as input considering the non-exact nature of QD. Therefore, we need to approximate polygons as group of pixels to apply QD. We approximate the polygonal obstacles by considering each pixel which has at least some part of the obstacle as obstacle. However, this approximation can cause some of the paths to the goal to be blocked. To guarantee reaching to the goal state from any other state in the free space, we apply the following strategy: We start from the smallest quadtree resolution and check for this blockage using a DFS algorithm[1] to see if all the other non-obstacle states are

[1]We first construct a graph (using *digraph* function of MATLAB) with directed edges where nodes represent the states, and directed edges represent the policy. Then, we make a depth-first graph search on this graph (using *dfsearch* function of MATLAB) starting from goal state. *dfsearch* returns the vector of node IDs in the order of

reachable from the goal state. If not, we increase the quadtree resolution until there is a path from every other cell in free space to the goal cell. Thus, we choose the coarsest possible quadtree resolution which does not cause blockage towards reaching the goal.

Fig. 3.1 shows an example map with different resolutions for QD. In Fig. 3.1a and 3.1b, the goal state stays inside the obstacle, so robot cannot reach there and the result is the same with lower resolutions. As the resolution increases, the goal is now in the free space as can be seen in Fig. 3.1c. But still there is a blockage preventing reaching to the goal state from some states in the free space. In Fig. 3.1d, finally the blockage between the goal state and other cells disappear and every state in the free space can reach the goal state. This is where our algorithm stops increasing the resolution for QD.

We use *delaunayTriangulation* function of MATLAB for CDT. Having defined the vertices and the edge constraints of polygonal obstacles, receiving resulting triangles is straightforward.

Fig. 3.2 shows QD and CDT on a sample map. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white.

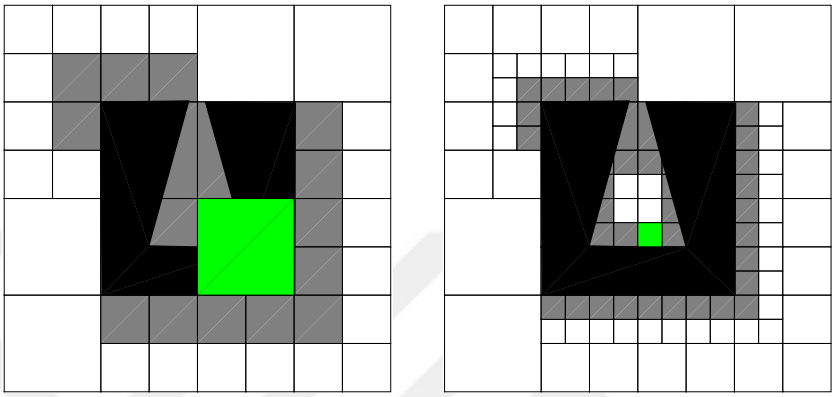
3.2 Definition of MDP

In this section, we define the 4 elements of MDP used in this study.

3.2.1 Definition of States

The states are defined based on the discretization method. In the case of QD, resulting states are squares whereas in CDT, resulting states are triangles. Each of these states can be defined by Cartesian coordinates of their vertices. Orientation is not included in state definition as done in [30] and [31], since we consider a point mass with

their discovery. We compare size of this vector with the number of non-obstacle cells in the map. If they are same, we say every other state in free space is reachable from the goal.



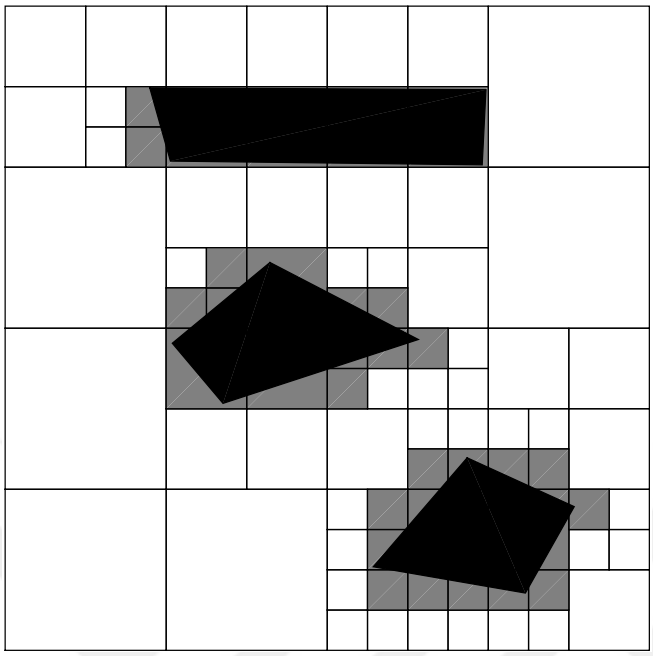
(a) QD of the sample map with resolution of 8x8

(b) QD of the sample map with resolution of 16x16

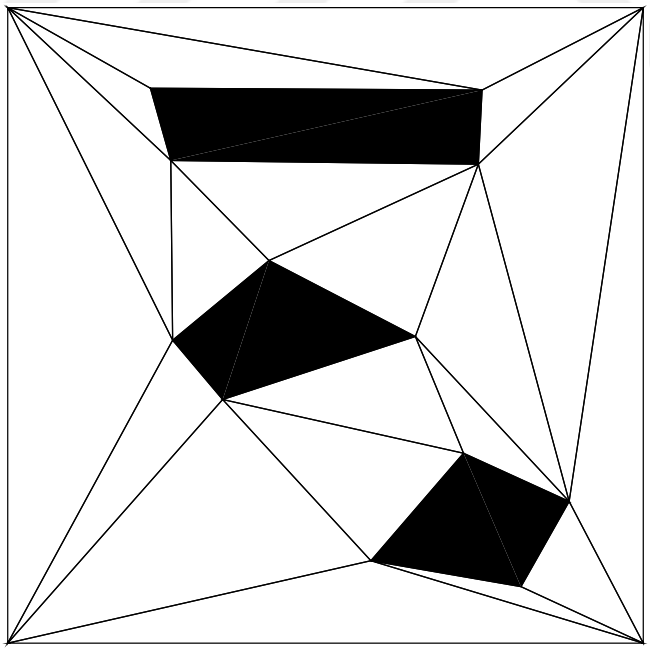
(c) QD of the sample map with resolution of 32x32

(d) QD of the sample map with resolution of 64x64

Figure 3.1: An example map with different resolutions for QD. In (a) and (b), the goal state is inside the obstacle. In (c), the goal state is in the free space but there is still a blockage. In (d), the blockage disappears.



(a) QD example.



(b) CDT example.

Figure 3.2: QD and CDT on a sample map. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in quadtree decomposition, all the other cells are in the free space and shown as white.

holonomic capabilities in this study. The states for QD and CDT is defined in Eq. 31a and Eq. 31b respectively.

$$S_{qd_i} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}_i \tag{31a}$$

$$S_{cdt_i} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}_i \tag{31b}$$

where x_n, y_n are Cartesian coordinates of nth vertice.

3.2.2 Definition of Actions

Actions allow passing from one state to another. In [30] and [31], they choose to use Dubins actions [36], whereas in [28] and [37] somewhat abstract actions are used. In the planning phase, we chose to define simple actions such that at any point agent chooses to go towards the center of the next cell determined by the optimal policy except if it is in the goal state or in a state whose policy directly points to goal and if that state is neighbor to goal state, then it directly goes towards the goal position. Sample actions from 5 different starting points to the goal are shown in Fig. 3.3. Obstacle is shown as black, s_i are starting points, and green dot shows the goal.

The real action driving the robot is being decided during execution stage by APF method and actuator limitations. Thus, all of the simple actions shown in Fig. 3.3 actually corresponds to F_{att}, the attractive force, used in APF method shown in Fig. 3.6. In Fig. 3.3, it looks as if the action starting from s_2 would collide with the obstacle, however this is not the case because of F_{rep}, the repulsive force in Fig. 3.6. Details about APF method will be given in Sec. 3.4.

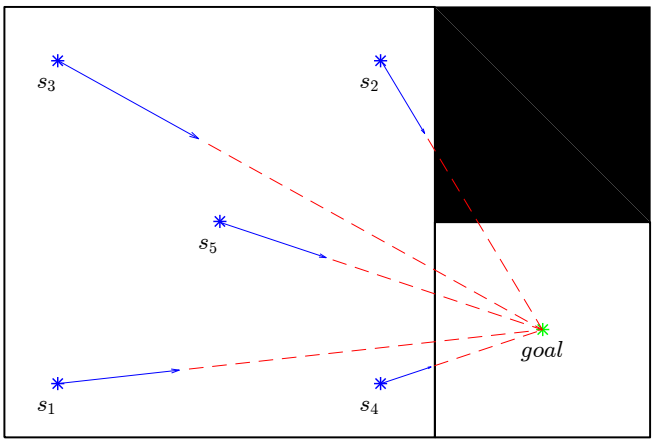


Figure 3.3: Sample actions starting from 5 different starting points to the goal. Obstacle is shown as black, s_i are starting points, and green dot shows the goal.

3.2.3 Definition of Transition Function

The transition function encodes the stochastic effects of actions in MDP. In [30] and [31], they model the transition function as multivariate normal distribution, then compute the transition functions for primitive Dubin actions and combine them for complex actions.

In our study, we do not know the exact action robot will take in advance, because it is defined by APF method during execution. Thus, we use only the attractive component of the action to construct the uncertainty model. We model the transition function as multivariate normal distribution, $X \sim \mathcal{N}(\mu, \Sigma)$ where

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \tag{32a}$$

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} \tag{32b}$$

where μ_x and μ_y are the cartesian coordinates of the center of the current cell. σ^2 is proportional to the distance between center of the current cell and center of the neighbor cell the robot wants to go to:

$$\sigma^2 \propto K \left\| \mu_{cur} - \mu_{neigh} \right\| \tag{33}$$

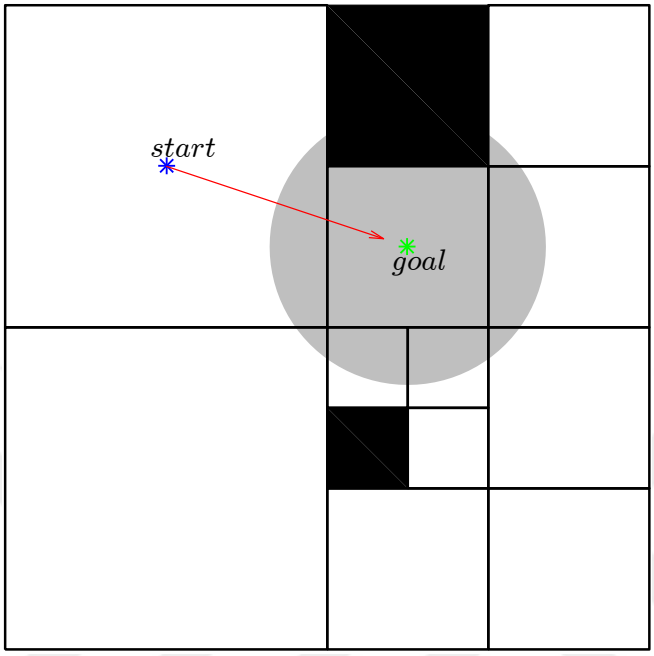
Fig. 3.4a shows a sample map discretized with QD. Black cells are obstacles, blue dot shows the center of the start cell, green dot shows the center of the goal cell. Gray area represents the high probability region for location of the robot, as a result of travelling from start point to goal point. The probability is highest at the center of the gray area, because if there was no noise, that's where the robot would be. Considering the real world scenarios and using the defined transition function, we predict where robot might be after passing from one cell to another. In Fig. 3.4a most of the gray area is in the goal cell, and rest of the area is distributed to other cells in an uneven fashion. Fig. 3.4b shows the uncertainty model used, which is a multivariate normal distribution. Its mean is at the center of the goal cell and its standard deviation is proportional to the distance between start cell and goal cell. It is reasonable to model standard deviation like this because as the distance between the cells increase, the distribution becomes more broad around the neighboring cells.

The probability values for transition function gets smaller and smaller as the position moves away from the mean. And it gets close to zero in cells far away from the goal cell. Thus, we limit the extend of the multivariate normal distribution for computational savings in the QD. Here are the limits for transition function of QD and CDT:

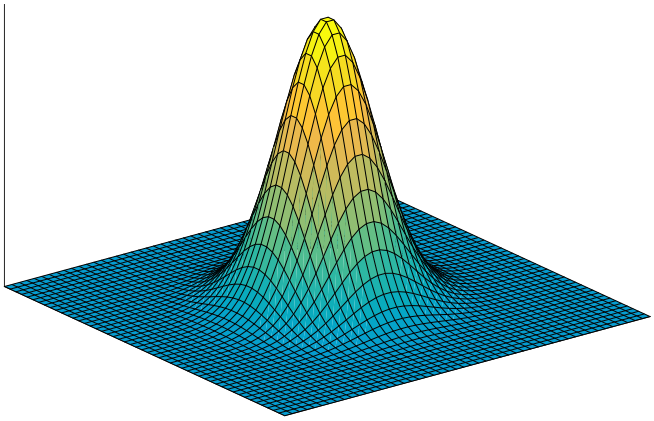
$$\begin{aligned} max(\mu_x - s, 0) \leq x_{qd} \leq min(\mu_x + s, size\_x) \\ max(\mu_y - s, 0) \leq y_{qd} \leq min(\mu_y + s, size\_y) \end{aligned} \tag{34}$$

$$\begin{aligned} 0 \leq x_{cdt} \leq size\_x \\ 0 \leq y_{cdt} \leq size\_y \end{aligned} \tag{35}$$

where x_{qd} and y_{qd} are the suitable coordinates for multivariate normal distribution for QD, x_{cdt} and y_{cdt} are the suitable coordinates for multivariate normal distribution for CDT, μ_x and μ_y are the coordinates of the center of the cell, s is the size of the cell, $size\_x$ and $size\_y$ are size of the map in x and y respectively. We do not limit the extend of the distribution in CDT, because triangulations are dispersedly distributed, and there are very few states to consider in the whole map.



(a) Demonstration of transition function on a sample map. Black cells are obstacles, blue dot shows the center of the start cell, green dot shows the center of the goal cell. Gray area represents the high probability region for location of the robot, as a result of travelling from start point to goal point.



(b) Demonstration of uncertainty model, multivariate normal distribution where its mean is at the center of the goal cell and its standard deviation is proportional to the distance between start cell and goal cell.

Figure 3.4: Transition function

3.2.4 Definition of Reward Function

Defining a suitable reward function is necessary for obtaining a policy that makes the robot go towards the goal and avoid obstacles. Defining a simple reward function is shown to be sufficient in [38], [39], [31] and [30]. We define the reward function as follows:

$$R(s) = \begin{cases} 1, & \text{if } s \text{ is the goal state} \\ -1, & \text{if } s \text{ is an obstacle state} \\ 0, & \text{otherwise} \end{cases} \tag{36}$$

This reward function is sufficient to obtain the optimal policy.

3.3 High-level Planning: Computing the Optimal Policy

As described in Sec. 2.2.2, optimal policy is the policy which gives the highest value for each state. In other words, the optimal policy gives the best actions to take in each state given the rewards and transition function. And its simply a mapping from states to actions. We use this optimal policy for high level decision making.

QD and CDT decompose the map into finite number of states. Using the defined transition function and rewards, we use value iteration to compute the optimal policy. When checking for convergence, we compare last few policies and if they remain unchanged, we say the convergence is reached. Although computing policies and checking for them increases the execution time, it is better than checking for some predefined value or percentage change because it reduces the number of iterations, is independent of other parameters and gives faster results.

Using defined transition function and reward function, value iteration is applied as in Eq. 28. The resulting optimal policy is shown on the sample map in Fig. 3.5 for QD and CDT. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state.

Blue dot indicates the goal position. The cell which has the goal is shown as green.

3.4 Low-level Execution: Artificial Potential Field Method

We obtained the optimal policy using value iteration. For any state, the optimal policy gives which state to transition to. Still, moving to these states requires a low-level execution method. We use APF method for this purpose.

Fig. 3.6 shows the attractive force $\overrightarrow{F_{att}}$, repulsive force $\overrightarrow{F_{rep}}$, and the resulting desired force $\overrightarrow{F_{des}}$. Eq. 37 gives the relationship between these forces. Attractive force simply follows the policy by going towards the center of the next cell, and repulsive force avoids the obstacles within certain region.

$$\overrightarrow{F_{des}} = \alpha \overrightarrow{F_{att}} + (1 - \alpha) \overrightarrow{F_{rep}} \tag{37}$$

Directly following $\overrightarrow{F_{des}}$ results in sharp and unrealistic paths. In reality, a robot would follow a smoother path because of its dynamics and actuator limitations. Therefore, to make the simulation more realistic, we incorporate dynamics, actuator and velocity limitations, and noise. Fig. 3.7 shows the velocities for clarity.

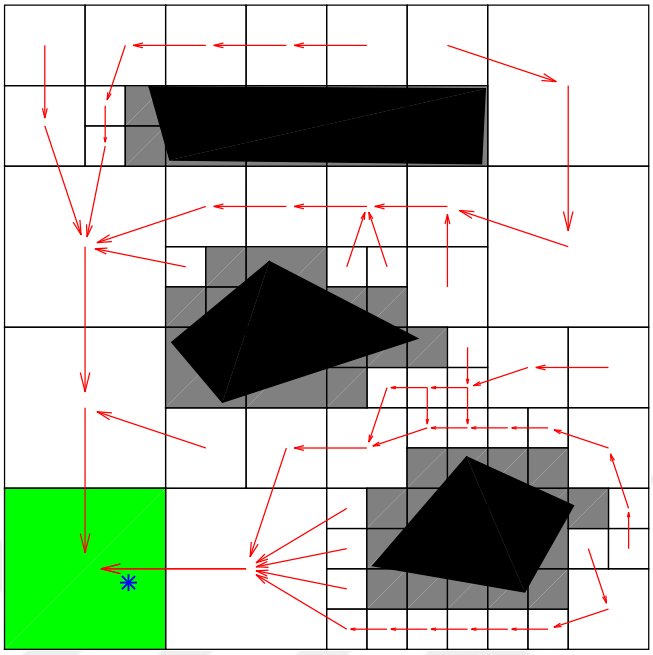
$\overrightarrow{v_{des}}$ is the desired velocity robot is trying to attain. However, this velocity may not always be attainable due to dynamics, actuator and velocity limitations. It is obtained from $\overrightarrow{F_{des}}$ as follows:

$$\overrightarrow{v_{des}} = \frac{\overrightarrow{F_{des}}}{m} \Delta t \tag{38}$$

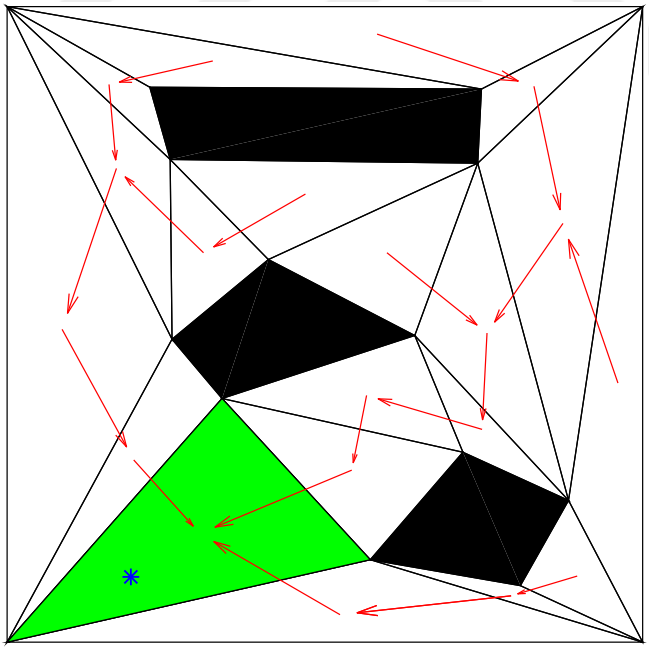
where m is the mass of the particle/robot, Δt is the elapsed time.

$\overrightarrow{v_{prev}}$ is the previous velocity of the robot. A force is required to change the velocity from $\overrightarrow{v_{prev}}$ to $\overrightarrow{v_{des}}$. This force is $\overrightarrow{F_{req}}$ and can be written as:

$$\overrightarrow{F_{req}} = m \frac{\overrightarrow{v_{des}} - \overrightarrow{v_{prev}}}{\Delta t} \tag{39}$$



(a) Optimal policy for QD.



(b) Optimal policy for CDT.

Figure 3.5: Optimal policy on the sample map. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

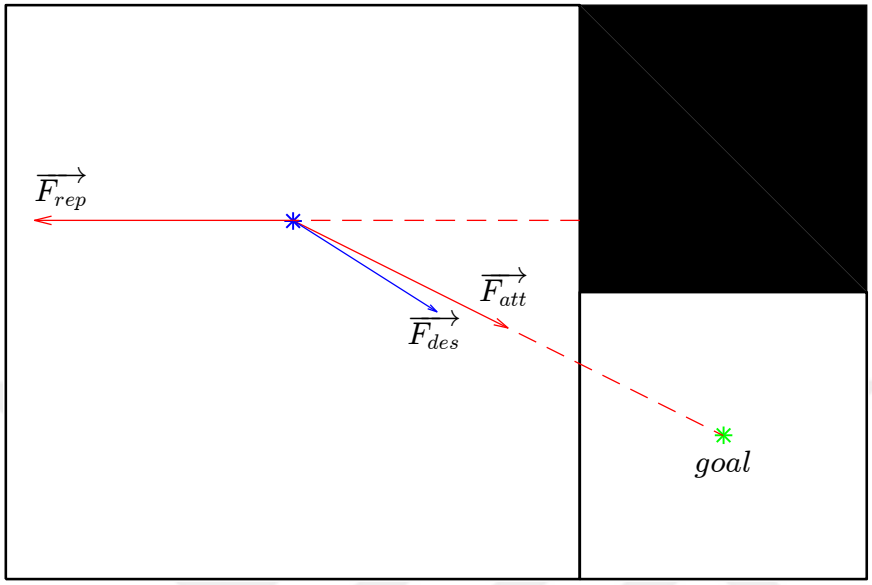


Figure 3.6: Attractive force, repulsive force and resulting desired force in APF.

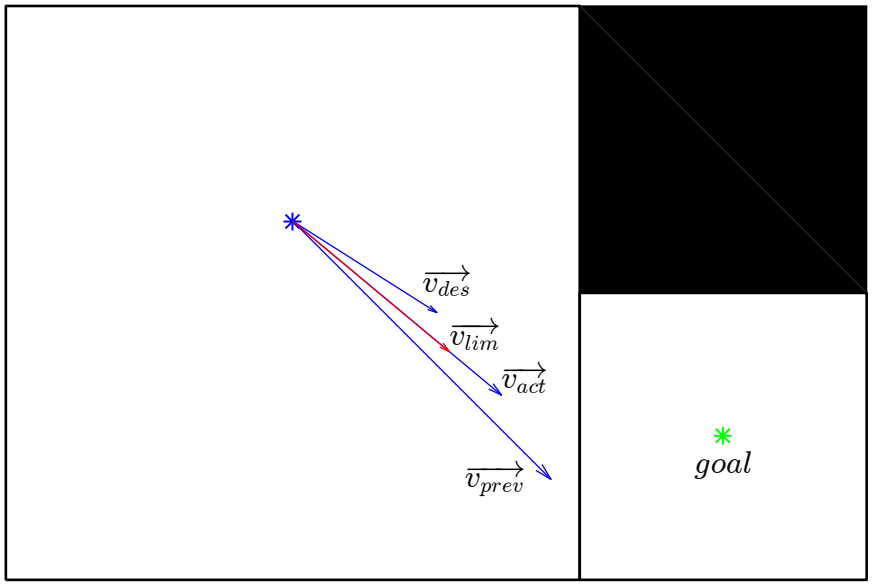


Figure 3.7: Velocities in APF.

Considering actuator limitations, we define F_{max}, the maximum force that can be applied by the robot. This force can be applied in any direction since the robot is holonomic. We also define v_{max} to limit the maximum speed of the robot.

If $\left\|\overrightarrow{F_{req}}\right\| > F_{max}$, then it is limited as:

$$\overrightarrow{F_{lim}} = \frac{\overrightarrow{F_{req}}}{\left\|\overrightarrow{F_{req}}\right\|} F_{max} \tag{310}$$

The applied force, $\overrightarrow{F_{app}}$ can be written as:

$$\overrightarrow{F_{app}} = \begin{cases} \overrightarrow{F_{req}}, & \left\|\overrightarrow{F_{req}}\right\| \leq F_{max} \\ \overrightarrow{F_{lim}}, & \left\|\overrightarrow{F_{req}}\right\| > F_{max} \end{cases} \tag{311}$$

This force is applied to the robot and the resulting actual velocity becomes:

$$\overrightarrow{v_{act}} = \overrightarrow{v_{prev}} + \frac{\overrightarrow{F_{app}}}{m} \Delta t \tag{312}$$

If $\|\overrightarrow{v_{act}}\| > v_{max}$, then it is limited as:

$$\overrightarrow{v_{lim}} = \frac{\overrightarrow{v_{act}}}{\|\overrightarrow{v_{act}}\|} v_{max} \tag{313}$$

The real velocity can be written as:

$$\overrightarrow{v_{real}} = \begin{cases} \overrightarrow{v_{act}}, & \|\overrightarrow{v_{act}}\| \leq v_{max} \\ \overrightarrow{v_{lim}}, & \|\overrightarrow{v_{act}}\| > v_{max} \end{cases} \tag{314}$$

If $\left\|\overrightarrow{F_{req}}\right\| \leq F_{max}$ and $\|\overrightarrow{v_{act}}\| \leq v_{max}$ then $\overrightarrow{v_{real}} = \overrightarrow{v_{des}}$. Robot could always go with $\overrightarrow{v_{des}}$ if there were no actuator and velocity limitations.

In the simulation, robot periodically applies the force $\overrightarrow{F_{app}}$, to reach the velocity $\overrightarrow{v_{real}}$.

3.4.1 Incorporating Noise

There is also noise component in the simulation. If there was no noise component, change in robot position would be as follows:

$$\begin{aligned} v_{mag} &= \|\overrightarrow{v_{real}}\| \\ \theta &= atan2(\overrightarrow{v_{real_y}}, \overrightarrow{v_{real_x}}) \\ \Delta x &= v_{mag} cos(\theta) \Delta t \\ \Delta y &= v_{mag} sin(\theta) \Delta t \end{aligned} \tag{315}$$

where Δx, Δy are change in x and y coordinates respectively, Δt is the elapsed time, $\overrightarrow{v_{real_x}}$ and $\overrightarrow{v_{real_y}}$ are the real velocities in x and y coordinates respectively.

After adding noise component, change in robot position becomes:

$$\begin{aligned} v_{new} &= normrnd(v_{mag}, k_1 v_{mag}) \\ \theta_{new} &= normrnd(\theta, k_2 v_{mag}) \\ \Delta x &= v_{new} cos(\theta_{new}) \Delta t \\ \Delta y &= v_{new} sin(\theta_{new}) \Delta t \end{aligned} \tag{316}$$

where k_1, k_2 are noise parameters, $normrnd$ is a MATLAB function which samples from normal distribution, its first parameter is mean, and second one is the standard deviation. We choose means as v_{mag} and θ which is straightforward, whereas we choose standard deviations as proportional to the magnitude of the real velocity. Thus, as the magnitude of the real velocity increases, the noise becomes more scattered around the mean. And by playing with k_1 and k_2, different noise distributions can be obtained. Fig. 3.8 shows a sample noise distribution of 500 points where $v_{mag} = 0.05$, $\theta = 0$, $k_1 = 1/30$, $k_2 = 2/3$. Red dot shows the initial position of the robot, green dot shows the ideal final position, and red path shows the ideal path followed by the robot. Blue dots show the positions after noise is incorporated.

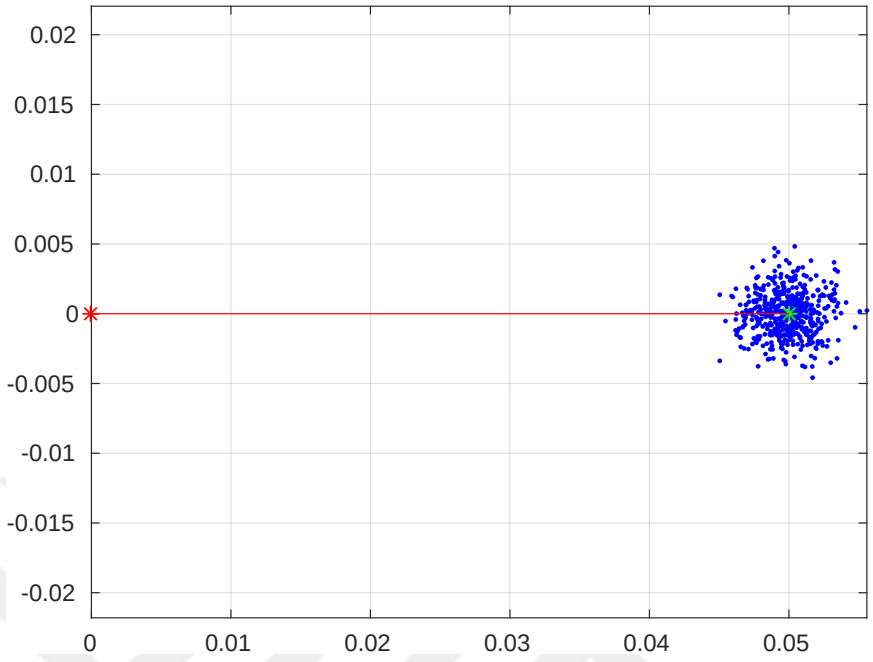


Figure 3.8: Sample noise distribution of 500 points. Red dot shows the initial position of the robot, green dot shows the ideal final position, and red path shows the ideal path followed by the robot. Blue dots show the positions after noise is incorporated.

3.4.2 Local Minima Problem

There are three cases that can cause the robot to get stuck:

- **Case 1:** $\overrightarrow{v_{des}} = 0$. This might occur if $\overrightarrow{F_{att}}$ and $\overrightarrow{F_{rep}}$ (combination of repulsive vectors) cancel each other. In this case, if $\left\|\overrightarrow{F_{req}}\right\| \leq F_{max}$, it can make $\overrightarrow{v_{real}} = 0$. Since the speed is zero, also there cannot be noise if the speed is zero, thus the robot stops. There are various cases that might cause this problem. One example is that robot might be going towards the goal, and there might be an obstacle behind the goal. In this case, there can be a local minima. Another example is that robot might not be directly going towards an obstacle but combination of repulsive vectors from obstacles may cause local minima as in Fig. 2.6b.

- **Solution for Case 1:** Ignore repulsive force when the angle between $\overrightarrow{F_{att}}$ and $\overrightarrow{F_{rep}}$ is very close to $180°$. When repulsive force is ignored, there is no longer local minima. This simple solution also prevents slowing down before encoun-

tering a local minima. However, in rare cases robot might collide with an obstacle since we ignore the repulsive force. During experimentation, we did not encounter any such cases. However, if for this reason or any other reason, robot collides with an obstacle, we do not include that case in the comparison.

- **Case 2:** $\overrightarrow{v_{des}} \neq 0$ but $\overrightarrow{v_{real}} = 0$. Since the speed is zero, there cannot be noise. In this case, $\overrightarrow{F_{att}}$ and $\overrightarrow{F_{rep}}$ do not cancel each other but robot still stops. For $\overrightarrow{v_{real}} = 0$ when $\overrightarrow{v_{des}} \neq 0$, $\overrightarrow{v_{des}}$ and $\overrightarrow{v_{prev}}$ should have a $180°$ between them and $\left\|\overrightarrow{F_{req}}\right\| > F_{max}$ thus limited as $\overrightarrow{F_{lim}}$ and if $\overrightarrow{F_{lim}}$ is exactly equal to the force required to stop the robot. This is unlikely to happen, and in our case this is impossible. Because it would require $180°$ change in velocity vector, and there is no such policy in the optimal policy (such a policy would cause returning to the state it came from, causing an infinite loop even if there was no local minima problem), and the repulsive vector cannot change that much since we are considering repulsive vectors from multiple obstacles.

- **Case 3:** It is unlikely to happen, but robot might get stuck in a loop that is not exacly local minima. We encountered such cases in experimentation. Although it is rare, in some parameter combinations, such problems can occur. To prevent infinite loops because of such problems, we stop execution if some prefined time has passed since the simulation started.

  Since our main objective here is to compare discretization methods, the simple solution given in Case 1 results in an almost complete low-level execution method. If it is desired to escape after local-minima occurs, or to prevent it before it happens, alternative strategies can be developed or used from the literature.

3.4.3 Special Cases in Quadtree Decomposition

There are special cases in QD where robot is actually not collided with an obstacle and there is no policy for that state. When robot is in an approximated obstacle, which is not an obstacle in reality, there is no policy for that state indicating where to go next, thus the action is undefined. In this case, we choose to follow the last best action as if robot was in the previous state. Fig. 3.9 demonstrates some of these special cases.

Black polygon is the real obstacle, gray regions are approximated obstacle regions, green dot is the goal point, red dots are starting points, and the path followed by the robot is shown as blue.

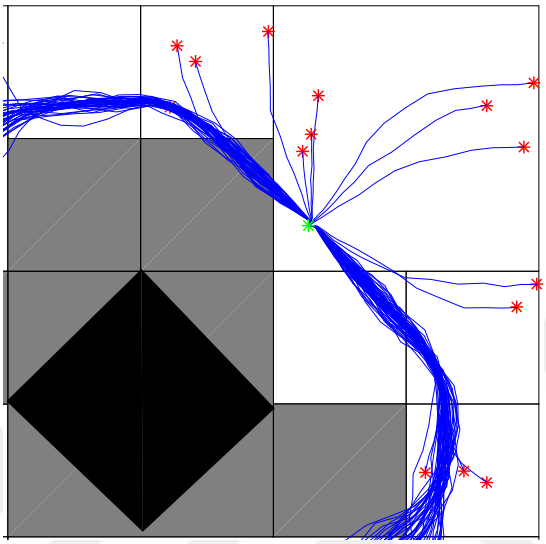


Figure 3.9: Demonstration of special cases in QD. Black polygon is the real obstacle, gray regions are approximated obstacle regions, green dot is the goal point, red dots are starting points, and the path followed by the robot is shown as blue.

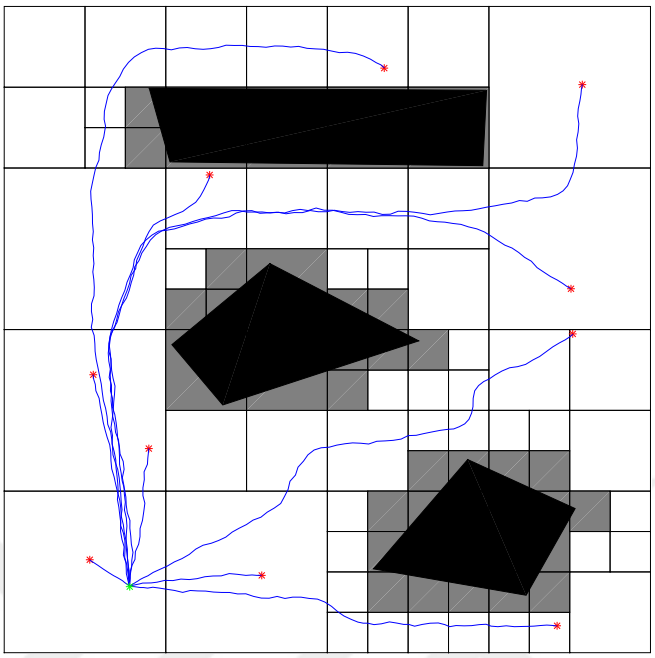
3.4.4 Preventing Getting out of Boundary

In the cases where there is an obstacle very close to the map boundary, the repulsive force from the obstacle may cause robot to leave the boundary. Thus, it is necessary to consider the boundary of the map as obstacle. Because if the robot leaves the boundary, we do not have any states or policies outside to get it back. Other strategies can be developed for this but for the time being, considering boundary as obstacle is sufficient.

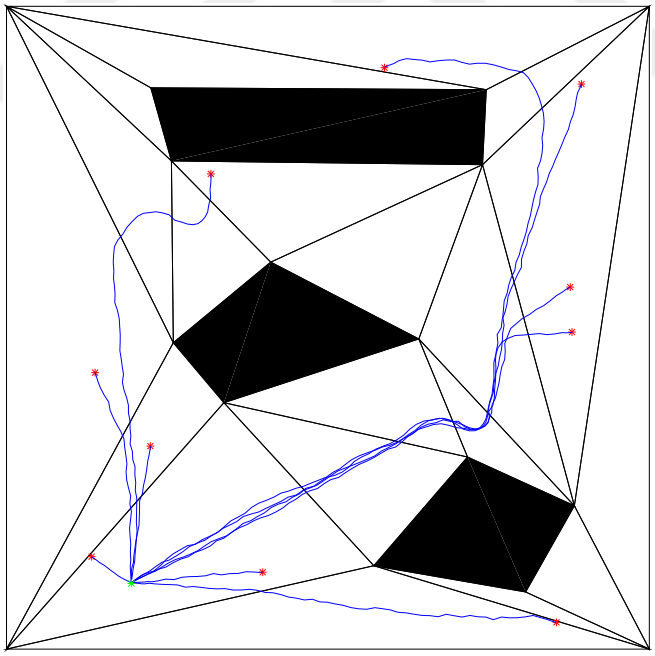
3.4.5 An Example

We used this low-level planner on the sample map from 10 random starting positions and the resulting paths are shown in Fig. 3.10 for both QD and CDT. Black regions indicate the original polygonal obstacles. Gray regions show the approximated ob-

stacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.



(a) Paths for QD.



(b) Paths for CDT.

Figure 3.10: Paths from 10 random points on the sample map. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

CHAPTER 4

EXPERIMENTAL WORK

In Sec. 3, we have described our approach towards comparing Quadtree decomposition (QD) and constrained Delaunay triangulation (CDT). In this section, we give simulation parameters and comparison metrics, then we explain how we choose the simulation parameters for a fair comparison, then we give results and discussion to compare QD and CDT in terms of various metrics.

4.1 Parameters and Comparison Metrics

There are various parameters in the simulation that can significantly affect the results, these are listed in Tab. 4.1 with their description. We compare QD and CDT based on the metrics given in Tab. 4.2 with their description.

4.2 Parameter Selection

In this section, we explain how we choose the simulation parameters for a fair comparison between QD and CDT.

K is measure of how scattered noise is distributed and it depends on the physical properties. It significantly affects the value iteration process and what the optimal policy will be. In the simulation, choosing same values for K for QD and CDT makes it hard to find optimal policies for both methods. Thus, by experimentation, we choose $K_{qd} = 0.0025$ and $K_{cdt} = 0.1$.

m, F_{max}, and v_{max} depend on the hardware and physical properties of the robot. For

Table 4.1: Simulation parameters and their description

Parameter	**Description**
K	gain for standard deviation in the transition function
ζ	gain for attractive gradient
η	gain for repulsive gradient
d^*_{goal} (m)	distance from the goal where potential changes between conic and quadratic
Q^* (m)	threshold distance from the robot where obstacles are ignored after
k_1, k_2	gains for standard deviation of noise distribution in the execution
m (kg)	mass of the particle/robot
α	parameter to combine attractive and repulsive forces
F_{max} (N)	maximum force that can be applied by the robot
v_{max} (m/s)	maximum speed of the robot

Table 4.2: Comparison metrics and their description

Metric	**Description**
Path length (m)	Total length of the path followed by the robot in reaching from start position to the goal position
Travel time (s)	Total travel time in reaching from start position to goal position
Safety measure 1 (m)	Average of the distances between the robot and the closest obstacle to it, throughout the whole path
Safety measure 2 (m)	Minimum of the distances between the robot and the closest obstacle to it, throughout the whole path
Planning time (s)	Execution time of the value iteration until convergence
Number of iterations	Number of iterations in the value iteration until convergence
Number of states	Number of states/cells on the map not including obstacles

simulation, they are chosen as $m = 1$ kg, $F_{max} = 0.02$ N, and $v_{max} = 0.2$ m/s. In the real world, these values would most probably be higher. But in the simulation, these

values are chosen to obtain realistic paths.

k_1 and k_2 are noise parameters, and they depend on the characteristics of the environment and the robot. For simulation, they are chosen as $k_1 = \frac{1}{30}$ and $k_2 = \frac{2}{3}$.

There are 5 parameters for APF (ζ, η, d^*_{goal}, Q^*, α) that can significantly affect the behavior of the low-level execution method. Choosing these parameters by trial and error is not optimal and cannot provide the best strategy for QD or CDT. Thus, we optimize these APF parameters for QD and CDT using PSO (Particle swarm optimization) based on the cost function given in Eq. 41. For optimization, we choose 100 random points on map 3 (in Fig. 4.6) and take average of metrics.

$$J = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{bmatrix} \begin{bmatrix} N \\ x \\ t \\ s_1 \\ s_2 \end{bmatrix} \tag{41}$$

where λ_1, λ_2, λ_3, λ_4, and λ_5 are weights, N is the number of times either robot collided with an obstacle, is out of the map boundary, or trapped in a loop or a local minima, x is the average path length, t is the average travel time, s_1 and s_2 are the average safety measures. We choose λ_1 much greater than the other weights to punish for the undesired outcomes like collision with obstacles, getting out of the map boundary, or trapping in local minima. The other weights are chosen empirically from the sample results. The resulting parameters for QD and CDT are obtained as follows:

$$\begin{bmatrix} \zeta \\ \eta \\ d^*_{goal} \\ Q^* \\ \alpha \end{bmatrix}_{qd} = \begin{bmatrix} 0.1682 \\ 1.0000 \\ 49.9239 \\ 46.3754 \\ 0.9900 \end{bmatrix} \quad (42a) \qquad \begin{bmatrix} \zeta \\ \eta \\ d^*_{goal} \\ Q^* \\ \alpha \end{bmatrix}_{cdt} = \begin{bmatrix} 0.9994 \\ 0.6967 \\ 41.2817 \\ 0.1019 \\ 0.9893 \end{bmatrix} \quad (42b)$$

where definitions for these parameters were given in Tab. 4.1.

4.3 Results and Discussion

In this section, we give the results and discussion to compare QD and CDT in terms of path length, travel time, two safety measures, planning time, number of iterations, and number of states. Descriptions for these metrics were given in Tab. 4.2. We created 3 simple and 2 relatively complex maps to compare QD and CDT. The maps were chosen so that they would most probably cause a standard APF algorithm to trap in local minima and they have various shapes and sizes of obstacles with narrow and wide passages. One of the maps was specifically designed by approximating a circular obstacle as a polygon to see how circular obstacles affects the results. We obtained the optimal policies and paths starting from 100 randomly selected points from free space and averaged results for path length, travel time, and two safety measures on 5 maps.

Optimal policies for QD and CDT on 5 maps are given in Fig. 4.1, 4.3, 4.5, 4.7, and 4.9. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

Paths from 100 random points for QD and CDT on 5 maps are given in Fig. 4.2, 4.4, 4.6, 4.8, and 4.10. Black regions indicate the original polygonal obstacles. Gray

regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

Fig. 4.11 shows comparison of path length, travel time, two safety measures, planning time, number of iterations and number of states for QD and CDT on 5 maps where each map size is 16x16 in meters. Blue bars indicate results for QD whereas orange bars indicate results for CDT. x-axis indicates map indices and y-axis shows the comparison metric.

As can be seen on Fig. 4.11a, QD results in longer paths on almost all the maps. Fig. 4.11b shows that CDT results in longer travel times on all the maps except map 3. In map 3 in Fig. 4.6a, in the upper right portion of the map, although there was a much shorter path to the goal, it is closed by approximation in QD. Thus, the average path length is much longer in QD and it even caused more travel time in QD than CDT which is not the case in other maps. This is a disadvantage of QD compared to CDT.

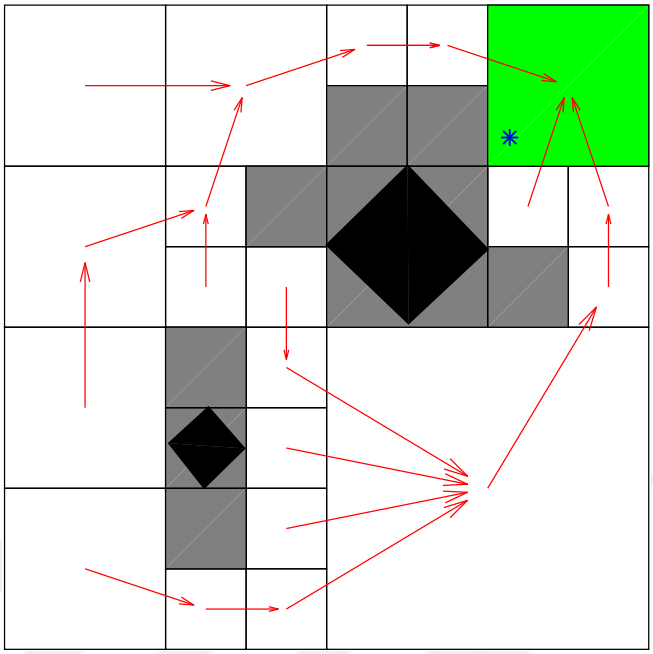
In terms of two safety measures in Fig. 4.11c and Fig. 4.11d, QD and CDT perform almost equally. As can be seen in Fig. 4.11e, 4.11f, and 4.11g, QD has clear disadvantages compared to CDT in terms of planning time, number of iterations, and number of states on all the maps. Planning time almost increases exponentially and number of states increase almost linearly as the maps become more complex in QD. However, planning time, number of iterations, and number of states are only important if online planning is required. Otherwise, if offline planning is preferred, comparing these metrics becomes mostly unnecessary. And considering the planning times, the current implementation is not suitable for online planning.

Map 5 (in Fig. 4.9) is different than other maps since it has an almost circular obstacle approximated as a polygon with 40 edges. Having lots of edges causes the map approximated with CDT to have greater number of states than that of QD as can be seen in 4.11g. However, still the planning time in 4.11e is much greater in the QD because it still needs to consider various neighbors during value iteration, whereas CDT only needs to look for 3 neighbors.

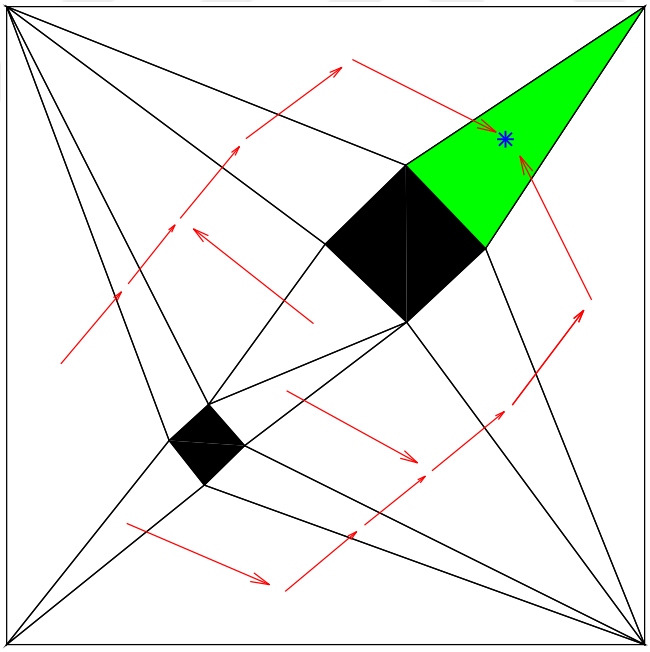
In overall, QD and CDT seems both suitable in the context of MDP and path planning

with APF. QD results in longer, and more smooth paths and requires less travel time. Whereas CDT results in shorter, and sharper paths and requires more travel time. QD and CDT perform almost equally in terms of safety. QD and CDT might be preferable for different applications. If offline planning time, number of iterations or number of states is important, it is better to choose CDT over QD. If travel time is important, QD can be preferred.

It is important to note that the results for path length, travel time, and safety measures might vary based on the cost function of the optimization. So, it is best to optimize for the preferred metrics on a specific problem to get the best possible outcome.

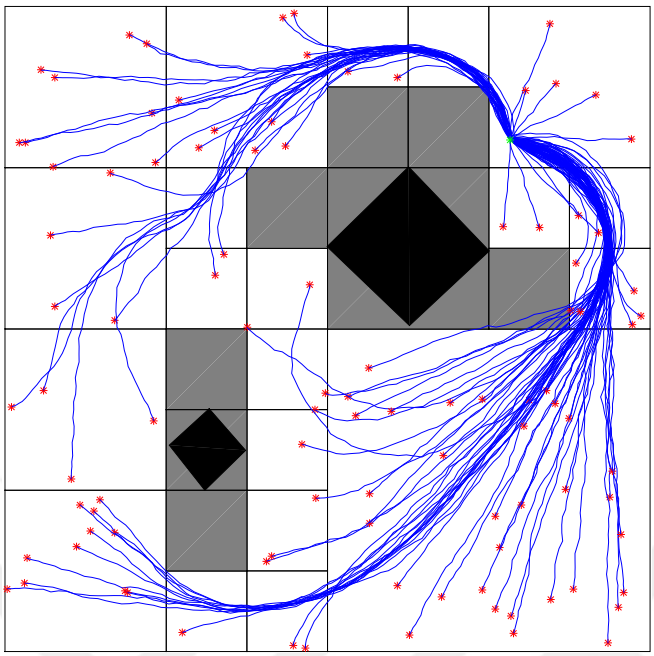


(a) Optimal policy for QD on map 1.

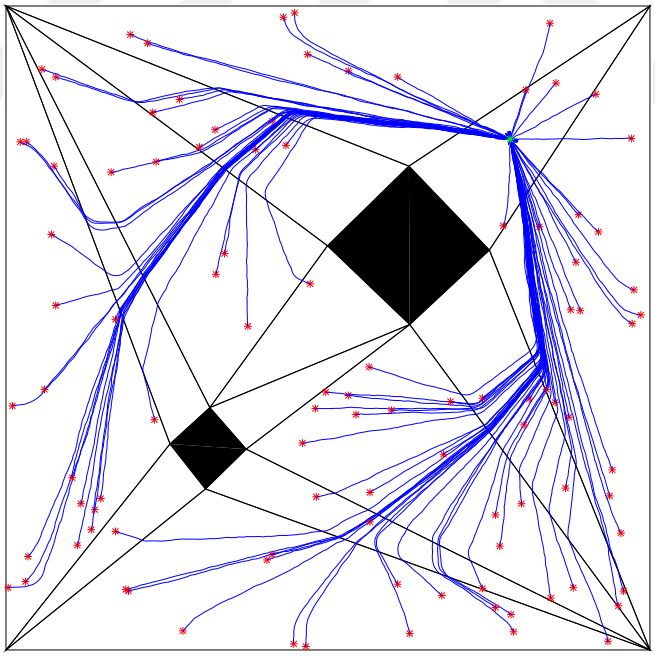


(b) Optimal policy for CDT on map 1.

Figure 4.1: Optimal policy on map 1. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

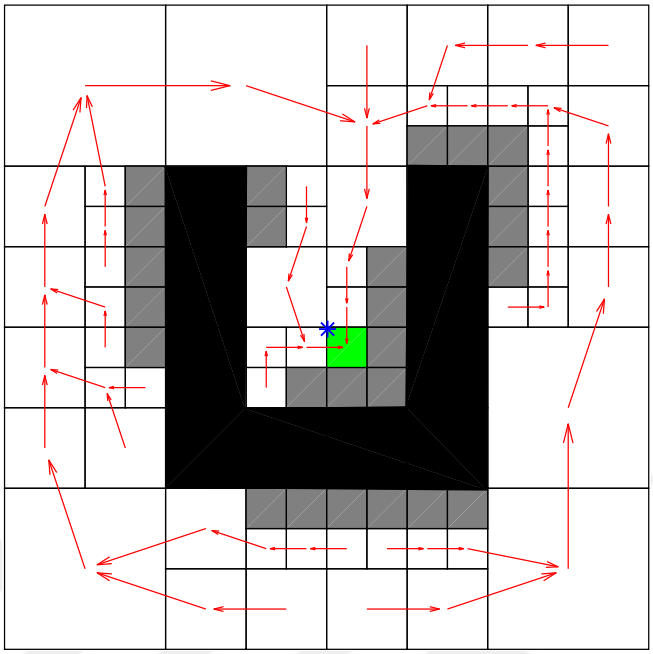


(a) Paths for QD on map 1.

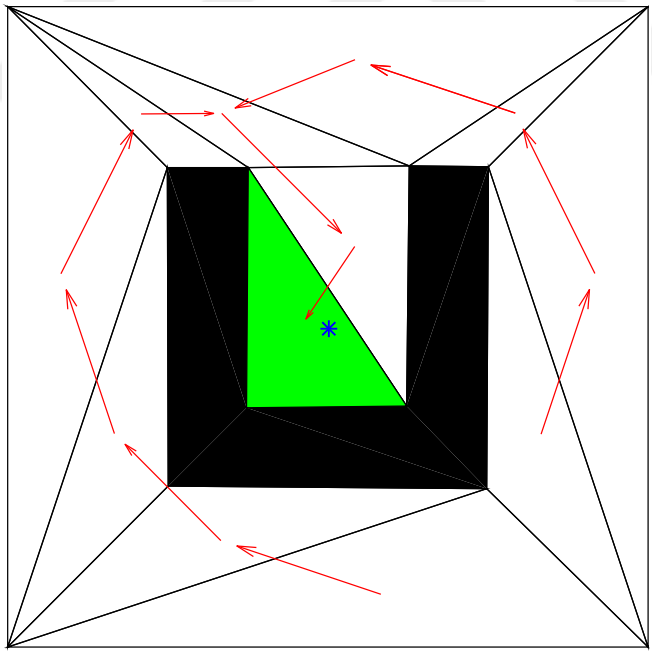


(b) Paths for CDT on map 1.

Figure 4.2: Paths from 100 random points on map 1. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

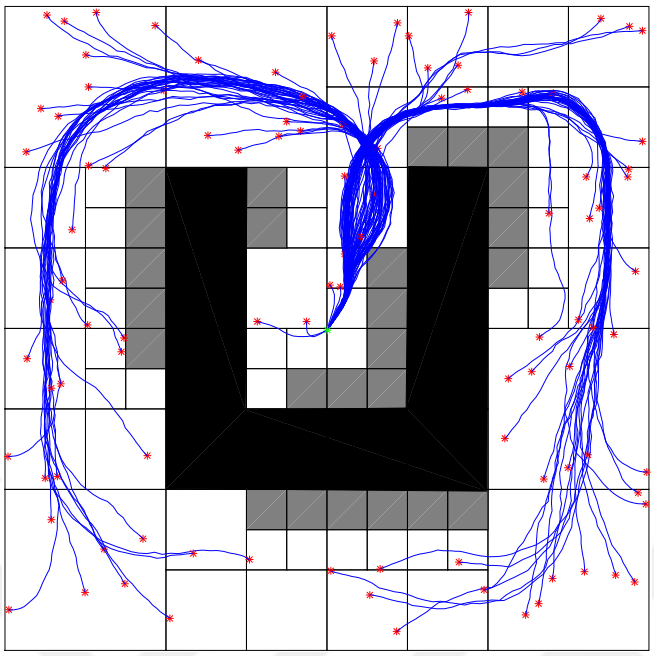


(a) Optimal policy for QD on map 2.

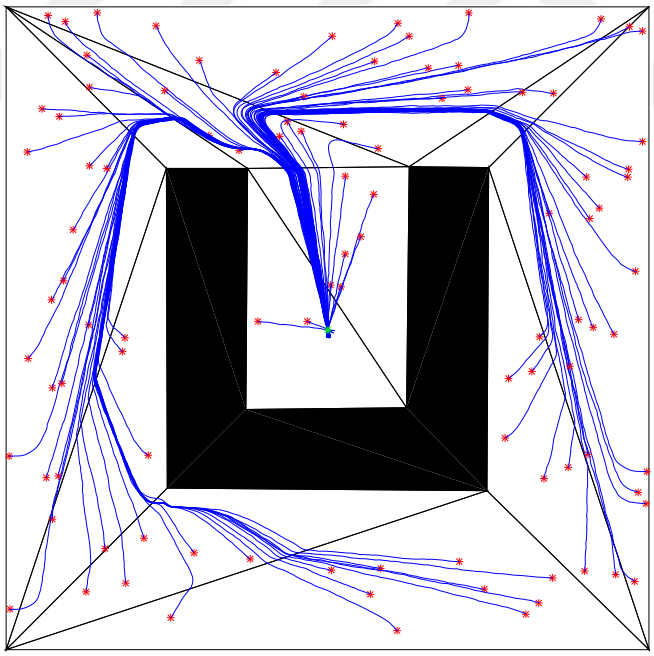


(b) Optimal policy for CDT on map 2.

Figure 4.3: Optimal policy on map 2. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

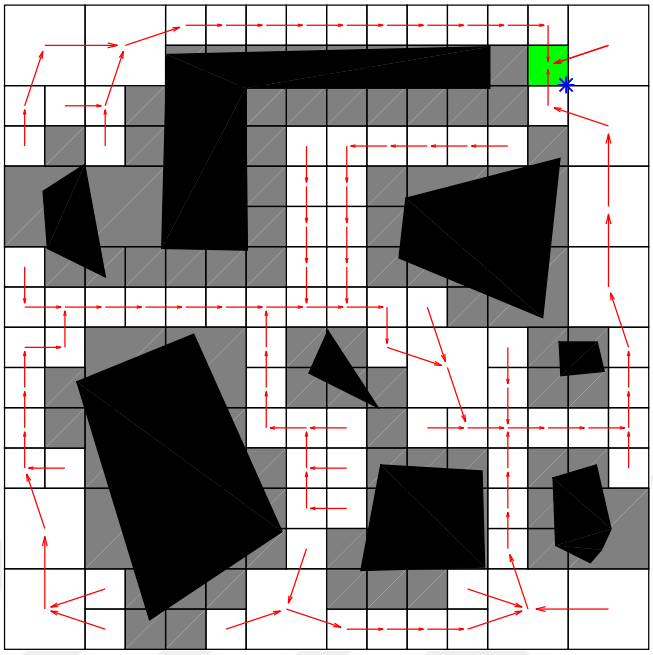


(a) Paths for QD on map 2.

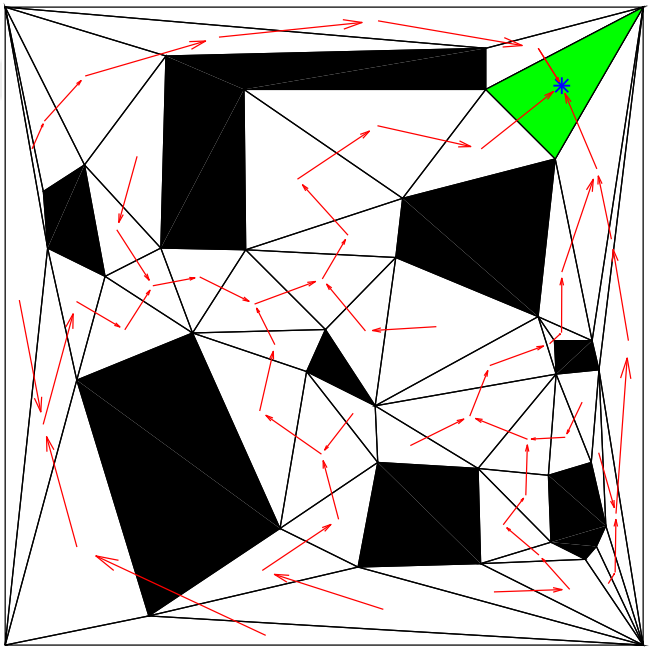


(b) Paths for CDT on map 2.

Figure 4.4: Paths from 100 random points on map 2. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

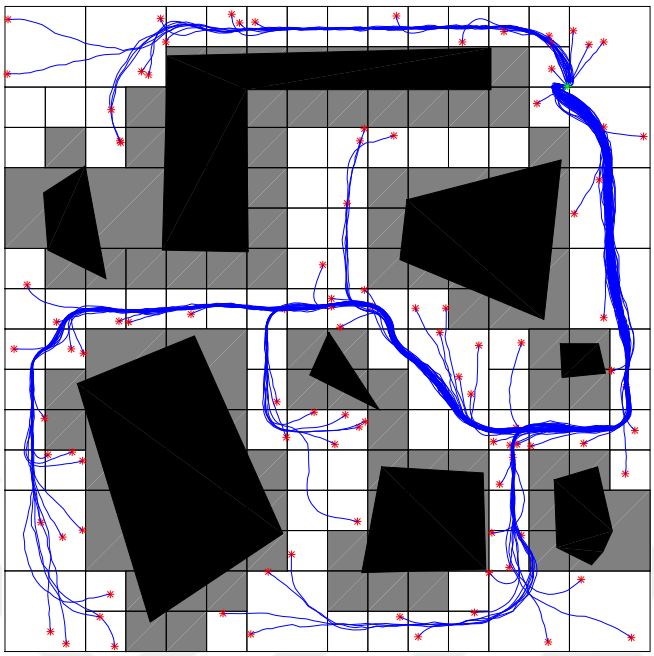


(a) Optimal policy for QD on map 3.

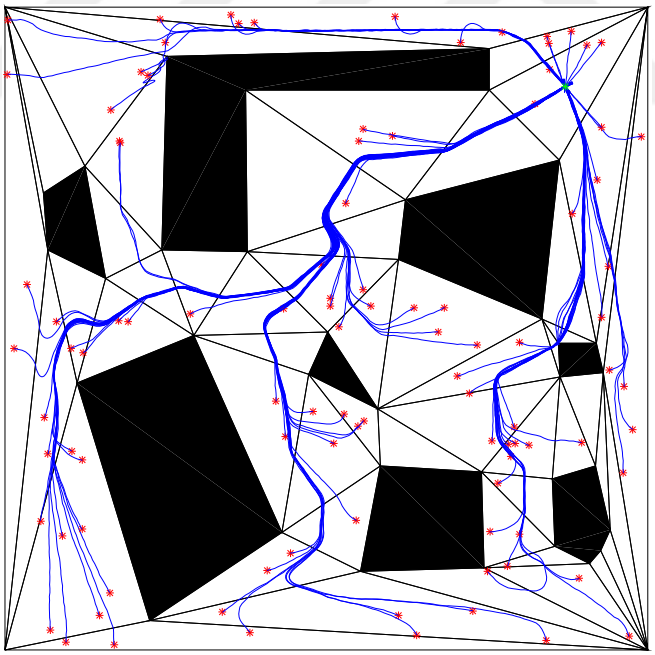


(b) Optimal policy for CDT on map 3.

Figure 4.5: Optimal policy on map 3. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

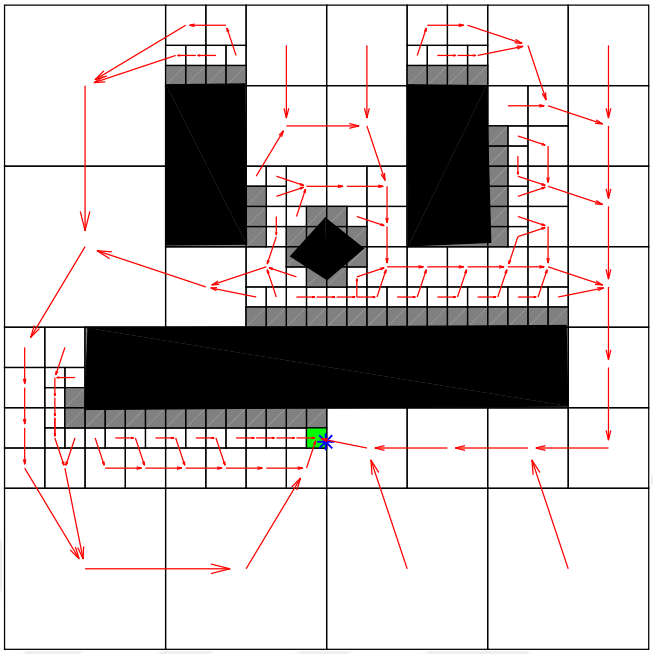


(a) Paths for QD on map 3.

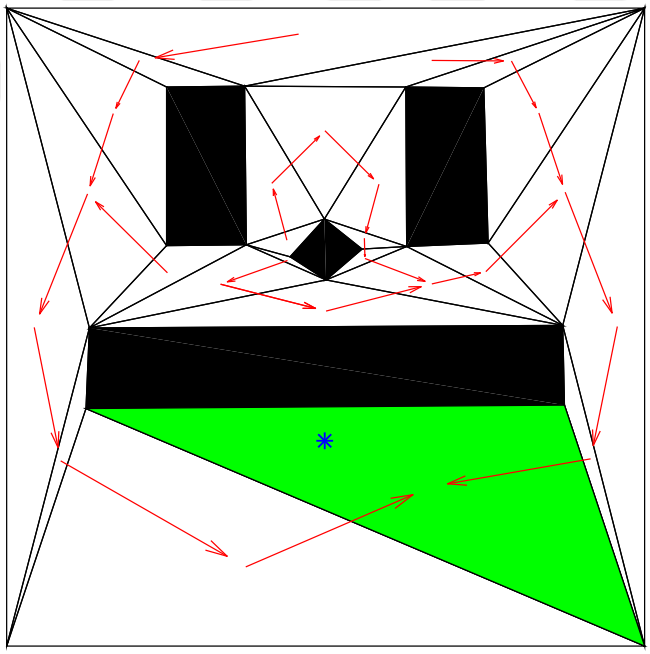


(b) Paths for CDT on map 3.

Figure 4.6: Paths from 100 random points on map 3. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

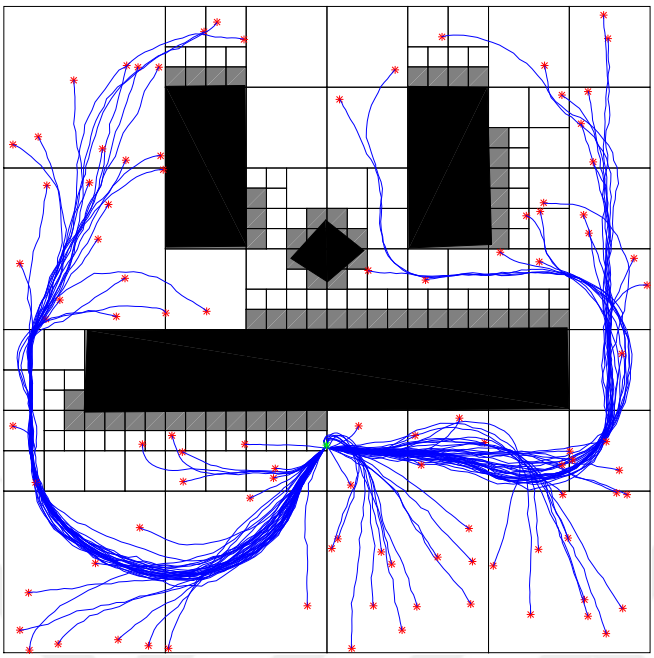


(a) Optimal policy for QD on map 4.

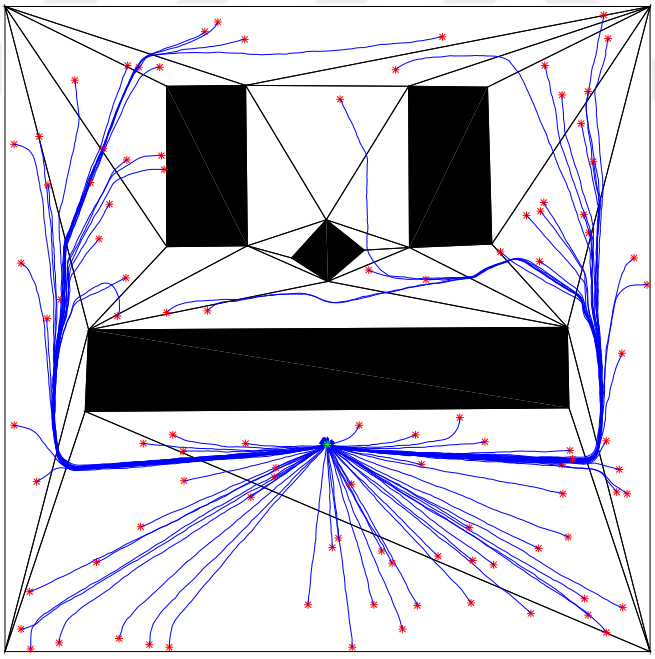


(b) Optimal policy for CDT on map 4.

Figure 4.7: Optimal policy on map 4. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

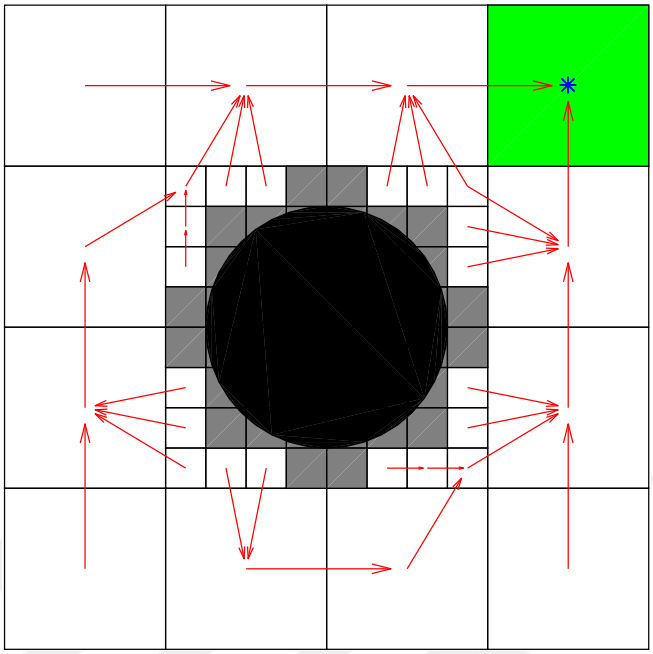


(a) Paths for QD on map 4.

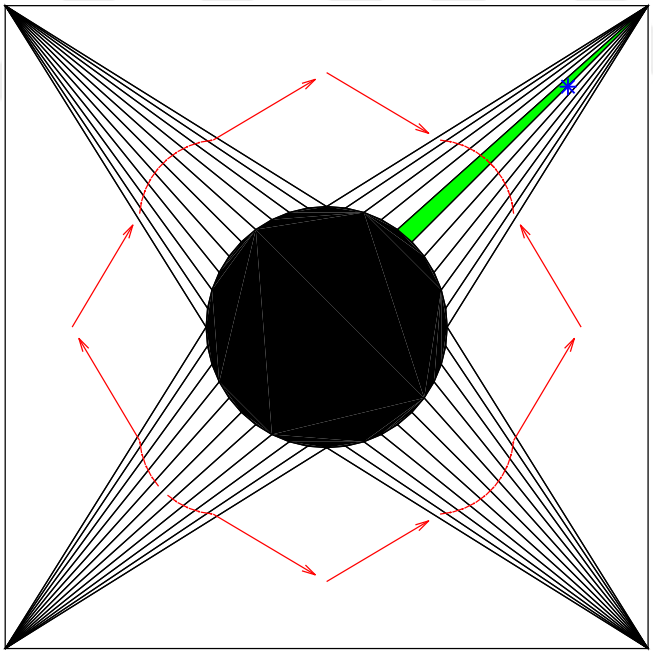


(b) Paths for CDT on map 4.

Figure 4.8: Paths from 100 random points on map 4. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.

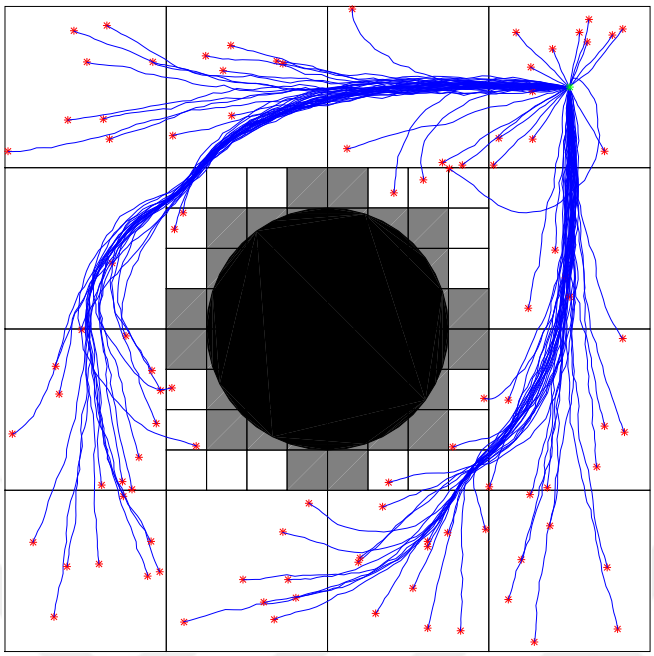


(a) Optimal policy for QD on map 5.

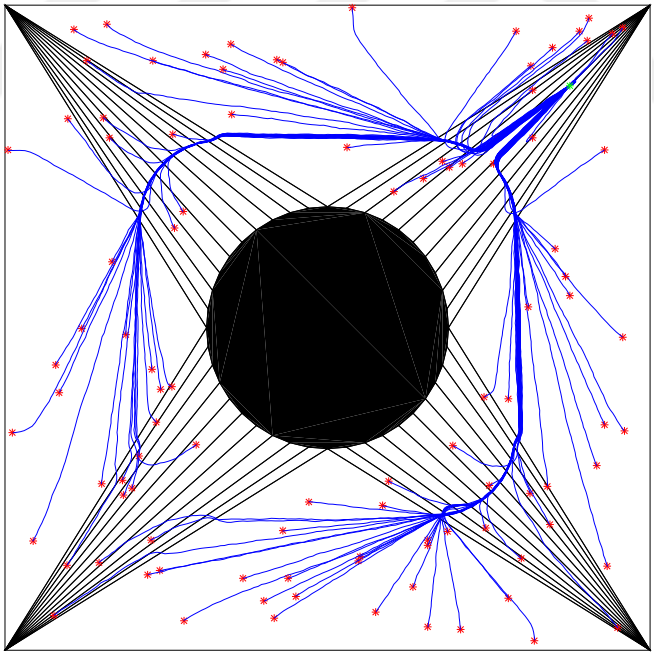


(b) Optimal policy for CDT on map 5.

Figure 4.9: Optimal policy on map 5. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. Red arrows indicate the policy, which state to move for any state. Blue dot indicates the goal position. The cell which has the goal is shown as green.

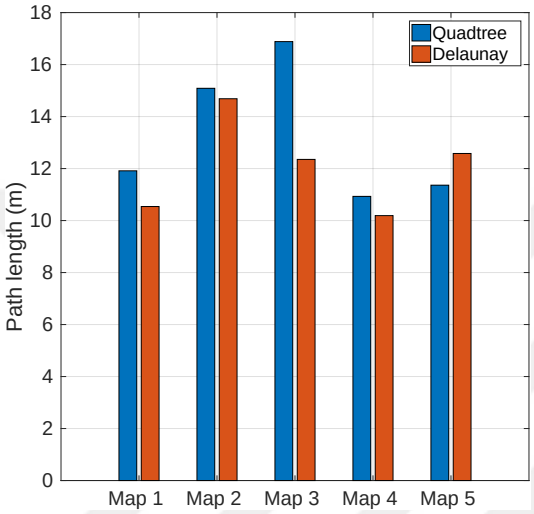


(a) Paths for QD on map 5.

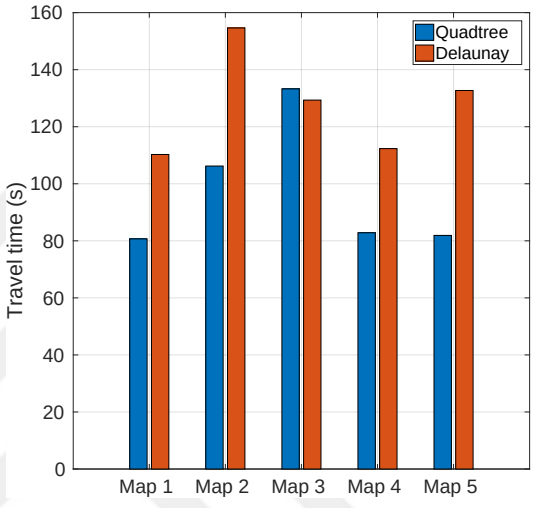


(b) Paths for CDT on map 5.

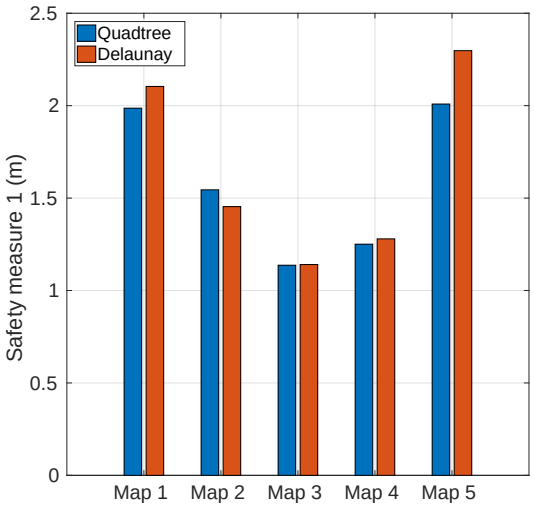
Figure 4.10: Paths from 100 random points on map 5. Black regions indicate the original polygonal obstacles. Gray regions show the approximated obstacle regions in QD, all the other cells are in the free space and shown as white. The path followed by the robot is shown as blue. Red dots indicate the starting positions. Green dot indicates the goal position.



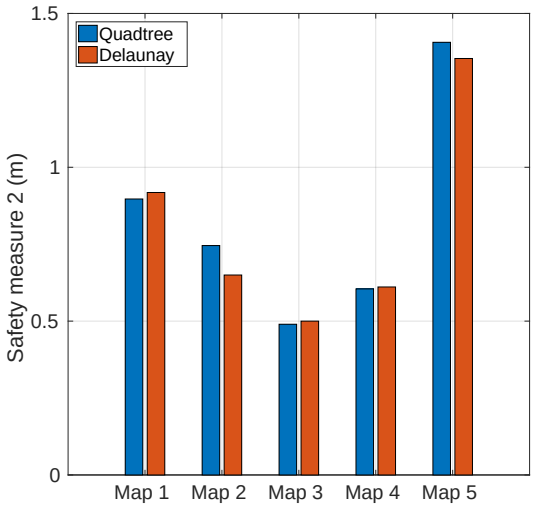
(a) Comparison of path length



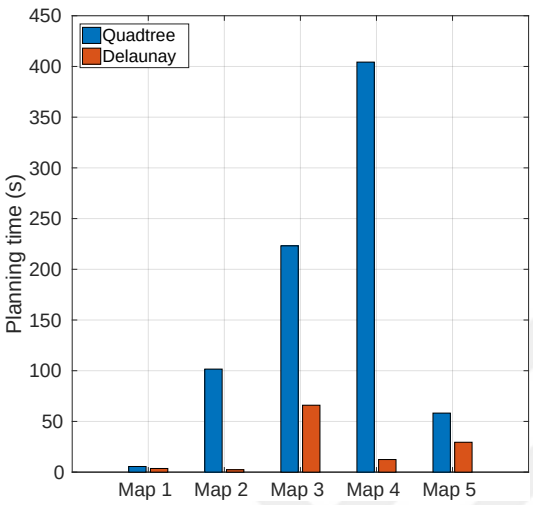
(b) Comparison of travel time



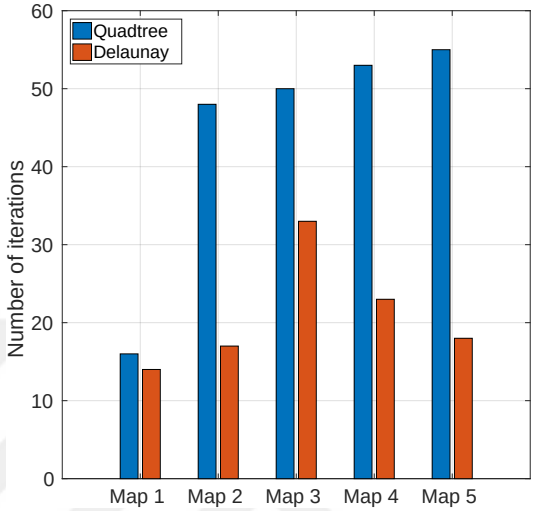
(c) Comparison of safety measure 1



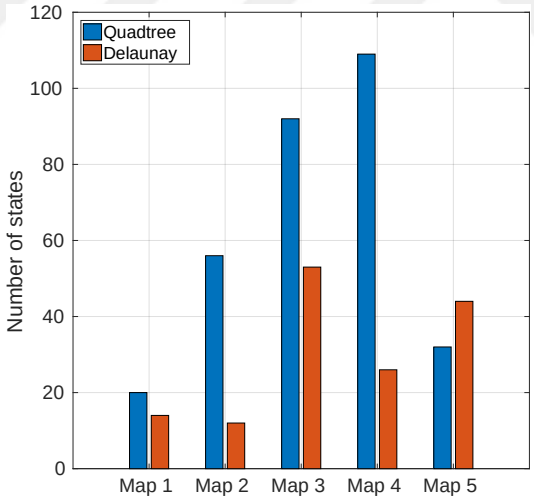
(d) Comparison of safety measure 2



(e) Comparison of planning time



(f) Comparison of number of iterations



(g) Comparison of number of states

Figure 4.11: Comparison of various metrics for QD and CDT on 5 maps where each map size is 16x16 in meters. Blue bars indicate results for QD whereas orange bars indicate results for CDT. x-axis indicates map indices and y-axis shows the comparison metric.

CHAPTER 5

CONCLUSION

5.1 Summary

In this section, we give a summary of the work done in this study. First, we discretized continuous space using Quadtree decomposition (QD) and constrained Delaunay triangulation (CDT). Then, we defined Markov decision process (MDP) framework with states, actions, transition function, and reward function. Next, we used this MDP framework and value iteration to obtain the optimal policy which we used as a high-level planner. Then, we used APF for low-level execution. Next, we defined comparison metrics and selected suitable parameters in the simulation. Then, we obtained the average results from 100 random points on 5 maps. Next, we compared QD and CDT in terms of path length, travel time, two safety measures, planning time, number of iterations, and number of states. Results showed that, QD and CDT are both suitable in the context of MDP and path planning with artificial potential field (APF). QD results in longer paths but requires less travel time. Whereas CDT results in shorter paths but requires more travel time. QD and CDT perform almost equally in terms of safety. QD has clear disadvantages compared to CDT in terms of planning time, number of iterations, and number of states on all the maps. However, planning time, number of iterations, and number of states are only important if online planning is required. Another disadvantage of QD is that it might block some of the paths to the goal, thus causing a longer path in some maps. Using a circular obstacle on a map causes the map approximated with CDT to have greater number of states than that of QD. However, still the planning time is much greater in the QD than that of CDT. It is important to note that the results for path length, travel time, and safety measures might vary based on the cost function of the optimization. And QD and

CDT might be preferable for different applications. Therefore, it is best to optimize for the preferred metrics on a specific problem to get the best possible outcome.

5.2 Future Work

The possible future work related to this study includes:

- Extension of the overall algorithm for non-holonomic robotics systems
- Extension and realization of the current work on a real robotic system
- Extension or modification of the current algorithm to work with robots having dimension
- Incorporating real noise model to the value iteration algorithm
- Extending the comparison to different space discretization methods like trapezoidal decomposition
- Online update of the goal and obstacles in the map
- Increasing algorithm efficiency by vectorization
- Developing new methods or using existing strategies for completely preventing local-minima or escaping after it occurs
- Considering approximated obstacles in QD as non-obstacles but states with negative rewards, as a result, using a lower resolution in the map, thus decreasing the planning time for QD
- Finding the direction of $\overrightarrow{F_{max}}$ that would minimize the difference between $\overrightarrow{v_{des}}$ and $\overrightarrow{v_{act}}$.

REFERENCES

[1] J. Z. Kolter, "15-780: Markov decision processes." Lecture Slides, February 2016.

[2] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[3] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[4] G. M. Hunter and K. Steiglitz, "Operations on images using quad trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, pp. 145–153, apr 1979.

[5] H. Samet, "Hierarchical spatial data structures," in *Symposium on Large Spatial Databases*, pp. 191–212, Springer, 1989.

[6] C. A. SHAFFER, H. SAMET, and R. C. NELSON, "QUILT: a geographic information system based on quadtrees†," *International journal of geographical information systems*, vol. 4, pp. 103–131, apr 1990.

[7] B. Delaunay, "Sur la sphère vide. a la mémoire de georges voronoï," *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, no. 6, pp. 793–800, 1934.

[8] J. Ruppert, "A delaunay refinement algorithm for quality 2-dimensional mesh generation," *Journal of Algorithms*, vol. 18, pp. 548–585, may 1995.

[9] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry*, vol. 22, pp. 21–74, may 2002.

[10] L. P. Chew, "Constrained delaunay triangulations," *Algorithmica*, vol. 4, no. 1-4, pp. 97–108, 1989.

[11] J. R. Shewchuk, "General-dimensional constrained delaunay and constrained regular triangulations, i: Combinatorial properties," in *Twentieth Anniversary Volume:*, pp. 1–58, Springer, 2009.

[12] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, pp. 396–404, Springer, 1986.

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

[14] M. G. Park, J. H. Jeon, and M. C. Lee, "Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing," in *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, vol. 3, pp. 1530–1535, IEEE, 2001.

[15] M. C. Lee and M. G. Park, "Artificial potential field based path planning for mobile robots using a virtual obstacle concept," in *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, vol. 2, pp. 735–740, IEEE, 2003.

[16] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-d path planning in a dynamic environment using an octree and an artificial potential field," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 2, pp. 474–481, IEEE, 1995.

[17] J. Barraquand, L. Kavraki, J.-C. Latombe, R. Motwani, T.-Y. Li, and P. Raghavan, "A random sampling scheme for path planning," *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[18] M. Mabrouk and C. McInnes, "Solving the potential field local minimum problem using internal agent states," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1050–1060, 2008.

[19] M. Guerra, D. Efimov, G. Zheng, and W. Perruquetti, "Avoiding local minima in the potential field method using input-to-state stability," *Control Engineering Practice*, vol. 55, pp. 174–184, 2016.

[20] M. G. Park and M. C. Lee, "A new technique to escape local minimum in artificial potential field based path planning," *KSME international journal*, vol. 17, no. 12, pp. 1876–1885, 2003.

[21] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[22] R. A. Howard, "Dynamic programming and markov processes.," 1960.

[23] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[24] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[25] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and related stochastic optimization problems," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 5–34, 2003.

[26] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[27] N. Roy, G. Gordon, and S. Thrun, "Finding approximate pomdp solutions through belief compression," *Journal of artificial intelligence research*, vol. 23, pp. 1–40, 2005.

[28] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, "Acting under uncertainty: Discrete bayesian models for mobile-robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, vol. 2, pp. 963–972, IEEE, 1996.

[29] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pp. 394–402, Morgan Kaufmann Publishers Inc., 1995.

[30] J. Burlet, O. Aycard, and T. Fraichard, "Robust motion planning using markov decision processes and quadtree decomposition," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA 2004*, IEEE, 2004.

[31] J. Burlet, T. Fraichard, and O. Aycard, "Robust navigation using markov models," *International Journal of Advanced Robotic Systems*, vol. 5, no. 2, p. 15, 2008.

[32] R. Luna, M. Lahijanian, M. Moll, and L. E. Kavraki, "Fast stochastic motion planning with optimality guarantees using local policy reconfiguration," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2014.

[33] R. Luna, M. Lahijanian, M. Moll, and L. E. Kavraki, "Optimal and efficient stochastic motion planning in partially-known environments.," in *AAAI*, pp. 2549–2555, 2014.

[34] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 421–427, IEEE, 1979.

[35] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 2, pp. 224–233, 1985.

[36] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[37] P. Laroche, *Processus Décisionnels de Markov appliqués à la planification sous incertitude*. PhD thesis, Université Henri Poincaré-Nancy 1, 2000.

[38] T. L. Dean, L. P. Kaelbling, J. Kirman, and A. E. Nicholson, "Planning with deadlines in stochastic domains.," in *AAAI*, vol. 93, pp. 574–579, 1993.

[39] P. Laroche, F. Charpillet, and R. Schott, "Mobile robotics planning using abstract markov decision processes," in *Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on*, pp. 299–306, IEEE, 1999.