PATH PLANNING IN SIMILAR ENVIRONMENTS

by

Yanyan Lu
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_________________________________________  Dr. Jyh-Ming Lien, Dissertation Director

_________________________________________  Dr. Jana Kosecka, Committee Member

_________________________________________  Dr. Amarda Shehu, Committee Member

_________________________________________  Dr. Siddhartha Sikdar, Committee Member

_________________________________________  Dr. Sanjeev Setia, Department Chair

_________________________________________  Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____________________________________  Fall Semester 2013
George Mason University
Fairfax, VA
Path Planning in Similar Environments

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

By

Yanyan Lu
Master of Science
George Mason University, 2011
Bachelor of Science
Dalian University of Technology, 2007

Director: Dr. Jyh-Ming Lien, Professor
Department of Computer Science

Fall Semester 2013
George Mason University
Fairfax, VA
Dedication

I dedicate this dissertation to my parents.
Acknowledgments

I would like to express my deepest gratitude to all people who made this possible. In particular, I want to thank the following people.

First of all, I would like to thank Prof. Jyh-Ming Lien, who has been my advisor during my PhD study. Prof. Lien introduced me to the field of robotics, and provided me a great and easy environment to do research. To me, he is the kind of advisor who can make his students concentrate on research pleasantly and not have to worry about trifling matters. Without his patient guidance through the challenges and difficulties of robotics, I would never be able to stand here and present my works.

I am very grateful to my other committee members, Prof. Jana Kosecka, Prof. Amarda Shehu and Prof. Siddhartha Sikdar, who gave me a lot of valuable comments and advice on my work. Prof. Kosecka helped me understand intelligent robotics systems and showed me how to conduct comprehensive experiments. Prof. Shehu taught me computational biology which is a very interesting application of robotics. Prof. Sikdar helped me with better understanding of the challenges in my work.

I want to express my special thanks to Nalini Vishnoi, who has been one of my dearest friends and my officemate throughout my PhD study. I really appreciate the fun we had together and the support she gave me when I was feeling low.

Also I would like to thank everyone in our Motion and Shape Computing (MASC) Group, especially, Evan Behar, Guilin Liu, Zhonghua Xi and Christopher Vo for their support through this long and challenging journey.

Finally, I would like to thank my parents, my younger brother and my boyfriend for always being there.
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>Abstract</td>
<td>xvi</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Configuration Space</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Multi-query versus Single-query Path Planners</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Completeness in Path Planning</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Motivation of Thesis</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Contribution of Thesis</td>
<td>8</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Path Planning in Static Environments</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Path Planning in Known Dynamic Environments</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Path Planning in Unknown Dynamic Environments</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Collision Prediction</td>
<td>16</td>
</tr>
<tr>
<td>3 Path Planning in Similar Static Environments</td>
<td>18</td>
</tr>
<tr>
<td>3.1 General Overview</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Assumptions</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Create Local Roadmaps</td>
<td>22</td>
</tr>
<tr>
<td>3.3.1 Transform Local Roadmaps</td>
<td>23</td>
</tr>
<tr>
<td>3.4 Merge Local Roadmaps</td>
<td>24</td>
</tr>
<tr>
<td>3.4.1 Merge Local Roadmaps Using Boundary Nodes</td>
<td>25</td>
</tr>
<tr>
<td>3.5 Experimental Results</td>
<td>26</td>
</tr>
<tr>
<td>3.6 Shape Matching</td>
<td>30</td>
</tr>
<tr>
<td>3.6.1 Shape Filter</td>
<td>31</td>
</tr>
<tr>
<td>3.6.2 2D Matching Methods</td>
<td>31</td>
</tr>
<tr>
<td>3.6.3 3D Matching Methods</td>
<td>34</td>
</tr>
<tr>
<td>3.6.4 Experiments</td>
<td>36</td>
</tr>
<tr>
<td>3.7 Discussion</td>
<td>42</td>
</tr>
</tbody>
</table>
7 Conclusions ................................................................. 113
    7.1 Static Environments ............................................. 113
    7.2 Known Dynamic Environments ............................... 114
    7.3 Unknown Dynamic Environments ............................ 114
    7.4 Future Research Direction .................................. 115
Bibliography .............................................................. 117
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>61</td>
</tr>
<tr>
<td>5.1</td>
<td>65</td>
</tr>
<tr>
<td>5.2</td>
<td>82</td>
</tr>
<tr>
<td>5.3</td>
<td>87</td>
</tr>
<tr>
<td>5.4</td>
<td>91</td>
</tr>
</tbody>
</table>

4.1 Running times (sec) for every phase of our method
5.1 Important notations used through this chapter.
5.2 Number of updates per second required to achieve at least 80% success rate for fixed-time strategy.
5.3 Number of updates per second required to achieve at least 80% success rate for fixed-time strategy.
5.4 Number of updates per second required to achieve at least 80% success rate for fixed-time strategy.
List of Figures

Figure | Page |
--- | --- |
1.1 A typical path planning problem in 2D. The robot is a rod shape model and the obstacles are polygons colored in light pink. The task is to find a valid path for the robot to move from start to goal without colliding any of these polygonal obstacles. The start specifies both position and orientation for the robot, but goal is a circular region which specifies position and allows any orientation for the robot. | 2 |
1.2 (a) A path planning problem in which topology of $C_{free}$ is captured by Probabilistic Roadmap Method (PRM) [1]. Blue dots are configurations and they are connected by straight lines into a roadmap. A collision free path from start to goal is thickened. (b) An environment with narrow passages. | 4 |
1.3 The $\alpha$-puzzle consists of two tubes, each twisted into an alpha shape; one tube is the obstacle and the other the robot. The objective is to separate the intertwined tubes [2]. | 5 |
3.1 Two different classrooms share many similar chairs and desks. The main difference is in the arrangement of items. | 19 |
3.2 Two similar workspaces sharing the same robot and several similar obstacles. The red and blue objects indicate start and goal positions. Similar workspaces are usually viewed as completely different problems. | 20 |
3.3 (a) A database of pre-computed local roadmaps. (b) Transformed local roadmaps in a new environment. (c) A global roadmap is composed of the local roadmaps. | 21 |
3.4 Merge the local roadmaps of two overlapping obstacles. (a) Before merging. Free, collision, and boundary nodes are shown in dark red, white and light gray, respectively. (b) After merging, the boundary nodes are connected. | 25 |
3.5 Environments used in the experiments. (a,b) 2D workspaces with an articulated robot. (c,d) Similar workspaces with a rigid robot and cubes. (e,f,g) Similar workspaces with a U-shaped rigid robot and walls. | 27 |
3.6 Experimental results for the environments in Fig. 3.5. All results are collected over 30 runs. (a), (c) and (e) plot the average running time for each planning method over 30 runs. The y-axes are all in logarithmic scale. We stop the planner when it takes more than $10^8$ milliseconds. The success rate of each planning method is plotted in (b), (d) and (f).

3.7 An example of the proposed shape matching method.

3.8 Matching Results. 10 models used in this shape matching experiments.

3.9 Five different shape functions for a cube. The red dot is the center of the model. Dashed lines are invisible from the front side. (a) Randomly sample three points (in black) on the surface of the model and measure the angle (in blue) between two line segments (in green) formed by these three points. (b) Randomly sample one single point on the surface and measure its distance to the center of the model. (c) Randomly sample two points on the surface and measure their distance. (d) Randomly sample three points on the surface which form a triangle and measure the square root of the area of this triangle. (e) Randomly sample four points on the surface which form a tetrahedron and measure the cube root of the volume of this tetrahedron.

3.10 Normalization by alignment of min sample values. The vertical n-axis indicates number of samples in each bin and horizontal f-axis indicates the value of shape function.

3.11 Three groups of environments with similar obstacles. Take the leftmost column as an example. Environments (d) and (g) have the same arrangement of obstacles as Environment (a). In (d), it introduces two different but similar objects: bed and bookshelf. Therefore, shape-matching is required in order to reuse local roadmaps of bed and shelf in (a). In (g), there are even more different obstacles compared to (d). So more shape-matching work is required for reusability. Similarly, in the middle column, environments (b), (e) and (h) share the same arrangement of obstacles. Compared to (b), (h) has more different obstacles than (e). In the rightmost column, environments (c), (f) and (i) share the same arrangement of obstacles. Compared to (c), (i) has more different obstacles than (f).
3.12 Three groups of environments with similar obstacles. These environments are supposed to be harder than those in Fig. 3.11 since they contain articulated robots and narrow passages. In the leftmost column, environments (a), (d) and (g) share the same arrangement of obstacles. Compared to (a), (d) contains more different obstacles than (g). Therefore, more shape-matching is required for (g) if the local roadmaps from (a) are to be re-used. Similarly, in the middle column, environments (b), (e) and (h) share the same arrangement of obstacles. Compared to (b), (h) has more different obstacles than (e). In the rightmost column, environments (c), (f) and (i) share the same arrangement of obstacles. Compared to (c), (i) has more different obstacles than (f).

3.13 Experimental results from the environments in Fig. 3.11. The average running time is plotted in (a), (c) and (e). The y-axes are all in logarithmic scale. The success rate of each planning method is plotted in (b), (d) and (f).

3.14 Experimental results from the environments in Fig. 3.12. The average running time is plotted in (a), (c) and (e). The y-axes are all in logarithmic scale. The success rate of each planning method is plotted in (b), (d) and (f).

4.1 The assumption that trajectories are fully known can be found in many practical applications such as (a) assembly line planning and (b) mobile factory floor robots KIVA.

4.2 (a) The convex hull hierarchy of a simple polygon. The original and its components are lightly shaded while the convex hulls are represented by dashed lines. (b) The convex hull hierarchy of an approximate L-shape. The original and its components are lightly shaded while the convex hulls are represented by dashed lines. (c) A Convex hull traversal tree. The magnitude of an arrow shows the penetration depth of two convex pieces. Its direction is the corresponding penetration direction.

4.3 Rc and Oi are colored in shadow after transformations. (a) Rc moves along with Oi when Oi has translational velocity. (b) Since c ∈ MOi, Rc can be treated as a feature on Oi. While Oi is rotating around μi, Rc is also rotating around μi and its orientation is changing. If Oi has angular velocity ω, Rc’s angular velocity for rotation around μi is ω too. (c) Rc has the translational velocity V(t) introduced by rotating around μi. The instant direction of V(t) is always perpendicular to r.

4.4 The position relationships of two obstacles. (a). MOi does not overlap Oj and MOi does not overlap Oj. (b). MOi overlaps Oj and MOi overlaps Oj. (c). MOi overlaps Oj or MOi overlaps Oj. 

xi
4.5 This figure illustrates the translational velocities of $p$ and $q$ which are caused by rotation. For $p$, its translational velocity introduced by self-rotation of $R_c$ and rotation of $O_i$ is $\vec{\omega} \times \vec{r} + \vec{\omega} \times \vec{r}_1$. For $q$, its translational velocity caused by rotation of $O_j$ is $\vec{\omega}' \times \vec{r}_2$.

4.6 Assume the square is fixed while the tube is moving with velocity $\vec{v}$. The magnitude of an arrow shows the penetration depth and its direction is the corresponding penetration direction. $\delta_1$ and $\delta_2$ are the estimations of $t_{os}$ at the first and second iteration, respectively.

4.7 Connect two critical roadmaps in a 3D configuration-time space. $t_2$ is a time of separation because the triangle becomes separated from the rectangle’s local roadmap at $t_2$. For the two valid edges $c_1c_2$ and $c'_1c'_2$, connections $c_1c'_1$, $c_1c'_2$, $c_2c'_1$ and $c_2c'_2$ are valid. Their intersection with the configuration space at $t_1$ are $a$, $b$, $c$ and $d$ which are all collision free.

4.8 (a) Local planner for configurations from the same local roadmap at different moments. The black curve is the trajectory of obstacle $O_i$ and the blue curve is the connection between $c$ and $c'$. (b) Local planner for configurations from local roadmaps of different obstacles. The obstacles are two moving polygons. The square is rotating counterclockwise while the triangle is rotating clockwise. Both of them are translating from left to right. A point robot travels from the bottom-left corner to top-right corner.

4.9 Animation sequences for a rod-shaped robot moving in an environment with several boards. The robot needs to pass through several narrow holes in order to reach the goal. A board moves forward while rotating around its moving direction.

4.10 Animation sequences for a rod-shaped robot moving in an environment with several balls. A ball moves in an ellipse-shaped trajectories while rotating around its own center of mass.

4.11 Animation sequences for a rod-shaped robot moving in an environment with two big tables and a ball. A table translates while rotating around its vertical axis.

4.12 Experimental results for the environments in Fig. 4.9, Fig. 4.10 and Fig. 4.11. (a) The average running time over 300 runs. The $y$-axis is in logarithmic scale. (b) The success rate of each planning method over 300 runs.

5.1 An “envelope” of path segment $c_1c_2$ induced by $O_i$. It is bounded by two circles which are centered at $c_1$ and $c_2$, respectively and two lines $\overline{ab}$ and $\overline{cd}$ which are tangent to these two circles.
5.2 The red (thicker) curves in both figures are plots of the earliest arrival time \( f(t) \) for an obstacle. Black straight lines are plots of \( g(t) : t = T \). (a) When there is at least one intersection (blue dot) between \( f(t) \) and \( g(t) \), the collision region is not empty. (b) Otherwise, the collision region is empty.

5.3 \( O_i \) is currently placed at \( p \) and \( R \) is currently placed at \( c \). When both \( O_i \) and \( R \) are points, their closest distance can be computed using Law of cosines in \( \triangle pcjc \).

5.4 An RRT augmented with earliest collision time. The tree is rooted at current configuration \( r \) of the robot. Configurations \( c' \) and \( d' \) are the predicted earliest collision locations on the paths from \( r \) to \( c \) and \( d \), respectively.

5.5 Ten environments used in experiments. In any of them, a green dot and a blue dot indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red dot indicates the robot’s current position when the image is captured. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions.

5.6 Compare our method to the fixed-time strategy on success rates for environments (a)-(e) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.

5.7 Compare our method to the fixed-time strategy on success rates for environments in (f)-(j) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. For every specific robot velocity, (k) plots the average success rate over all these 10 environments.

5.8 Compare our method to the fixed-time strategy on number of replans for environments (a)-(e) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. Notice that the y-axis is in logarithmic scale.

5.9 Compare our method to the fixed-time strategy on number of replans for environments (f)-(j) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. For every specific robot velocity, (k) plots the average number of replans over all these 10 environments. Notice that the y-axis is in logarithmic scale.
5.10 Five environments used in experiments when robot is a point and obstacles could be articulated. In any of them, a green shape and a blue shape indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red shape indicates the robot’s current position. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84

5.11 Compare our method to the fixed-time strategy on success rates for environments in Fig. 5.10. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

5.12 Compare our method to the fixed-time strategy on number of replans for environments in Fig. 5.10. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

5.13 Six environments used in experiments when robot is a polygon. In any of them, a green shape and a blue shape indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red shape indicates the robot’s current position. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88

5.14 Compare our method to the fixed-time strategy on success rates for environments in Fig. 5.13. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89

5.15 Compare our method to the fixed-time strategy on number of replans for environments in Fig. 5.13. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

6.1 (a) If $A$ is fixed and $B$ translates towards $A$ along the closest direction, the closest features between $B$ and $A$ remain the same until they collide. (b) A polygon rotates around a point $o$. The area it sweeps over is bounded by the trajectory of every vertex ($p_1$ through $p_4$) and edges from both original and destination shapes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
6.2 (a) The swept area of $p_1p_2$ rotating is bounded by this gray shadowed donut-shape. The closest distance between $p_1p_2$ and $c$ is realized when $p_2$ becomes collinear with $o$ and $c$. (b) If $p_1p_2$ is fixed and $c$ rotates around $o$ with velocity $-\omega$, both the closest features and closest distance will be the same as in the case where $c$ is fixed and $p_1p_2$ rotates around $o$ with $\omega$. 

6.3 Three cases when $c$ is sufficiently close to the segment $p_1p_2$.

6.4 (a) The relationship between $t_R$ and $\alpha$ and $\beta$ when $R$ moves from $c_1$ to $c_2$. (b) and (c) illustrate the configurations $c$ and $c'$ in (a). $L_1$ and $L_2$ are the lines perpendicular to $p_1p_2$ and contain $p_1$ and $p_2$, respectively.

6.5 (a) The plot for $f(t)$ when the closest feature to $c$ locates on $p_1p_2$. The collision region is bounded by the two leftmost intersections. (b) The 3D plot shows the closest distance between an edge of some obstacle and the robot changes over time.

6.6 The rod-shaped $R$ is colored in black and the obstacle is colored in light pink. (a) The distance between their closest features is larger than 0. (b) $R$ and the obstacle are in contact and about to collide when the closest distance becomes 0.

6.7 Three types of intersections between two line segments. Note that for two line segments that are initially separated, (c) can only happen after (a) or (b).

6.8 For two objects which can only translate, collisions can be achieved at the earliest time if both of them move towards each other in the direction of closest features.

6.9 For two objects which can also rotate, it is hard to predict their motions which achieve collisions as soon as possible.

6.10 An articulated obstacle with 5 links. Red dot indicates the fixed base while the blue dot indicates the end effector.

6.11 The relative position between $p$ and $L_2$ will not change if we reverse their motions. That is, fix linkages $L_1$ and $L_2$ and make $p$ rotates around $J_1$ and $J_2$ in the opposite directions.
Abstract

PATH PLANNING IN SIMILAR ENVIRONMENTS

Yanyan Lu, PhD

George Mason University, 2013

Dissertation Director: Dr. Jyh-Ming Lien

Path planning aims at navigating a robot from an initial configuration to a goal configuration without violating various constraints. The problem of path planning is theoretically intractable (PSPACE hard), but in everyday life we (as human beings) navigate in our environment without much difficulty. This is partially due to the experiences we learned since childhood. The learning process may be complex, but one of the reasons that we can learn such tasks is that most objects we encounter today are similar or identical to the objects we encountered yesterday or even years ago. Environments with similar objects are quite common. For example, desks and chairs in a classroom or in an office may be moved around from one place to another frequently, but unfamiliar items are seldom introduced. Even different environments, such as two apartments, or a manufacturing factory and an airport garage, may share many similar items. The main differences are usually in the arrangements. Similar environments can also be found in simulated reality, e.g., in different levels of a video game or in the different regions of a virtual reality world, where many objects are intentionally duplicated to reduce the (e.g., modeling and rendering) complexity. A dynamic environment where obstacles allowed to move can be considered as a continuous sequence of similar static environments due to motion coherence. We term “discrete similar-workspace problem” for static environments and “continuous
similar-workspace problem” for dynamic environments. In this thesis, I investigated path planners that can address both problems and recognize this similarity in order to significantly improve efficiency and completeness.

More specifically, I developed a path planner which exploits similarity across different static environments. This planner remembers the computation (the “mental image”) for every obstacle encountered. Given a new problem, by carefully reusing these existing computations, the planner quickly builds a roadmap from its knowledge base. In order to improve reusability, I also designed a new shape matching method for polygons. The experimental results show that the planner is 3∼8 orders of magnitude faster than most existing methods in environments with various similarity. Moreover, the success rate is kept at 100%. In other words, once a query is solved, it guarantees to be able to find a collision free path in the subsequent runs.

Environments with dynamic obstacles can also be viewed as a sequence of similar environments. In fact, existing methods have explored the temporal coherence (i.e. similarity) and gain efficiency by preserving the bulk of the roadmap and only repairing the invalid part due to motions of obstacles. However, all these methods repair blindly and periodically at fixed time intervals with little attempt to analyze the similarity or changes of the planning spaces across different time instances. This results in either redundant updates or failure to detect invalid edges and nodes. Based on the assumption that obstacles move along some known trajectories, we proposed a motion planner that detects critical moments when the topology of the free configuration space changes. The critical moments can be classified into two categories: 1) a time of contact is when the space between two configuration obstacles closes, and 2) a time of separation is when the space opens up. As shown in the experimental results, our planner not only avoids redundant computation (≈ one order of magnitude faster) but also improves the chances of finding a valid path.

In many real life applications such as rescue robots and autonomous vehicles, the robot may have no prior knowledge of obstacles’ motions. I developed a new geometric tool which allows the robot to determine the critical moments when the robot and obstacles can collide. Then, only at such critical moments, the robot updates its belief of the environment and re-plans if necessary. The main challenge in predicting collision stems from the assumption that obstacles’ motions are unknown.
To provide conservative estimation, this new tool models all obstacles as adversarial agents so that obstacles will move in the way to minimize the time that the robot remains collision-free.
Chapter 1: Introduction

Path planning is an important problem in robotics [3, 4]. It has also been widely applied in Computer-aided Design (CAD), computational biology, computer animation, computer graphics and virtual reality. A path planning problem often refers to the process of navigating a robot in environments without violating various constraints. Historically, path planning problems are considered to be unrelated if the geometric descriptions of the environments where the robot is are different. In this thesis, I will explore the connections of path planning problems among different environments and discuss several novel algorithms which are more robust and significantly expedite the problem solving process with the help of these connections.

Generally, a path planning problem can be defined as a 5-tuple $P = (R, O, W, S, G)$. $R$ is a movable object called robot. $R$ could be any kind of object. In this thesis, we assume $R$ is a rigid body in which deformation is neglected. $O$ is a set of static or dynamic objects which may impose space constraints to $R$. Both robot and obstacle geometries are described in 2D or 3D. $W$ is a world in 2D or 3D space and it is the space where $R$ and $O$ reside and move. Usually, $W$ could refer to any type of geometric space, either discrete or continuous. In this thesis, we assume $W$ is a user-specified continuous bounding box. A query is a pair composed of $S$ and $G$ where $S$ is some specified initial state for the robot and $G$ is a specified goal state or any state in a set of goal states where $R$ needs to reach finally. The task of path planning is to compute a plan which navigates $R$ in the given workspace $W$ from $S$ to $G$. Or if there is no solution, it should report failure.

The plan that we just mentioned specifies a sequence of actions for $R$ to take. Generally, there are two types of criteria that most planning algorithms consider [4]: feasibility and optimality. Feasibility means that the plan is composed of a sequence of feasible actions which make $R$ to arrive at a goal, without considering efficiency. For example, $R$ cannot collide with any obstacles, or certain dynamic constraints are imposed to both $R$ and $O$. Optimality means that the plan also
optimizes some performance such as earliest time, shortest travel distance or minimum uncertainties in addition to feasibility. Since feasibility is already challenging enough for most problems in this thesis, we will focus on computing feasible plans while trying to minimize travel distance for $R$.

A simple path planning problem in 2D is shown in Fig. 1.1.

### 1.1 Configuration Space

Configuration space ($C$-space for short) [3] is the set of all possible transformations that can be applied to the robot. A configuration is described as a set of independent parameters that can completely specify the pose of the robot. For example, for a robot who can only translate in a 2D workspace, there are two parameters in every configuration representing the $x$ and $y$ coordinates. If the robot can rotate around $z$-axis, the number of parameters in a configuration, i.e. the degree of freedom (DOF) is increased to three with the third representing its orientation around $z$-axis. If the robot is articulated, such as an arm-shaped model, each of its joint angles needs to be specified in
addition to the position of its base, if its first joint is a free-floating body.

The set of configurations that makes the robot collision free with all obstacles is called free configuration space $\mathcal{C}_{\text{free}}$, which is a subset of $\mathcal{C}$-space. The complement of $\mathcal{C}_{\text{free}}$ is the forbidden region $\mathcal{C}_{\text{forbid}}$ because any configuration in this region causes the robot to collide with at least one obstacle. For an obstacle $o$ in workspace, its corresponding $\mathcal{C}$-space obstacle is the set of all configurations which make robot in collision with $o$. Therefore, $\mathcal{C}_{\text{forbid}}$ is the union of all $\mathcal{C}$-space obstacles in the configuration space.

A valid or feasible path is defined as a continuous sequence of configurations in $\mathcal{C}_{\text{free}}$ with the first configuration in the sequence being $\mathcal{S}$ and the last configuration being in $\mathcal{G}$.

### 1.2 Multi-query versus Single-query Path Planners

Path planning has been studied extensively in the past four decades. Most existing work focuses on developing data structures to capture the connectivity of free configuration space. The data structure is usually a graph or a tree. Like in most work, we use roadmap to refer to these two data structures throughout the thesis.

Probabilistic Roadmap Method (PRM) [1] (see Fig. 1.2) is one of the most popular modern path planners involving roadmaps. It first randomly samples configurations from the configuration space. For every new configuration $q$, it decides whether $q$ is in $\mathcal{C}_{\text{free}}$ by placing the robot on $q$ and performing collision detection. A new configuration will be abandoned if it is in $\mathcal{C}_{\text{forbid}}$ or added into the roadmap as a node if it is in $\mathcal{C}_{\text{free}}$. Then configurations in near neighborhood are connected by local planners such as straight line. This process repeats until some stopping criteria are met, for example, number of nodes in the roadmap has reached a user-defined maximum. Then a path is searched for in the roadmap. PRM is simple to implement and can be easily extended to high dimensional configuration spaces.

The classic PRM draws configurations in the $\mathcal{C}$-space in a uniform distribution. Therefore it tends to
Figure 1.2: (a) A path planning problem in which topology of $\mathcal{C}_{free}$ is captured by Probabilistic Roadmap Method (PRM) [1]. Blue dots are configurations and they are connected by straight lines into a roadmap. A collision free path from start to goal is thickened. (b) An environment with narrow passages.

sample more free configurations in the large open region while samples fewer in the small regions or narrow passages. Consequently, PRM often fails to find a path if the robot has to pass through a narrow passage to reach the goal region (Fig. 1.2). To address this issue, many variants of PRM have been proposed, such as OBPRM [5], Gaussian PRM [6] and Bridge Test sampling [7]. All of them aim at boosting sampling density inside narrow passages to increase the probability of finding a collision free path. Difficult problems, such as $\alpha$-puzzle (Fig. 1.3), become the focus in these works in the past two decades.

All methods mentioned above build a graph to represent $\mathcal{C}_{free}$. This graph can be stored and used later for more queries in the same environment. Therefore, a path planner which creates a roadmap and then makes multiple queries in this roadmap is called a multi-query method. Other good examples include visibility graph [8] and cell decomposition [3]. A visibility graph is a graph constructed in the configuration space with the nodes including $\mathcal{I}$, $\mathcal{G}$ and all configuration-obstacle vertices.
An edge exists between two nodes if the connection is a configuration-obstacle edge or does not intersect any configuration obstacles. Then a path from $\mathcal{S}$ to $\mathcal{G}$ is extracted from this visibility graph. Cell decomposition decomposes $\mathcal{C}_{\text{free}}$ into a number of disjoint (non-intersecting) cells and then constructs a graph to capture the topology of $\mathcal{C}_{\text{free}}$. Each cell becomes a node in this graph and two nodes are connected by an edge if their corresponding cells are adjacent.

In some applications where preprocessing time is limited or not desired, or only one or a few planning queries are needed, it is not always necessary to construct a roadmap for the entire $\mathcal{C}_{\text{free}}$. Therefore, single-query planning is preferred in the sense that all computation is conducted with only respect to a specific query repair. Once the query repair is changed, a single-query path planner usually recomputes everything from scratch. Two well-known tree-growing single-query planners are: Rapidly-exploring Random Trees (RRT) [9] and Expansive Space Trees (EST) [10]. Both RRT and EST build a tree in $\mathcal{C}_{\text{free}}$. The tree is rooted at the start configuration $\mathcal{S}$ and then iteratively expanded until the goal is accessible from the tree or it reaches maximum number of iterations. The major difference between RRT and EST is how the tree is expanded. During the expansion stage, RRT first randomly samples a configuration in $\mathcal{C}$ denoted $q_{\text{rand}}$ (note that $q_{\text{rand}}$ does not have to be in $\mathcal{C}_{\text{free}}$) and then chooses the node $q_{\text{near}}$ from the tree which is the closest to $q_{\text{rand}}$. Then it moves from $q_{\text{near}}$ toward $q_{\text{rand}}$ for a user-defined distance $d$ or until robot is trapped and cannot make it any further in the direction of $q_{\text{rand}}$. The last valid (or collision free) configuration between $q_{\text{near}}$ and $q_{\text{rand}}$ is added into the tree. For expansion, EST first chooses a node $q$ from the current tree.
with probability $\frac{1}{w(q)}$ where $w(q)$ is the weight of $q$. Then it randomly samples a user defined $k$
configurations in $q$’s neighborhood. For every newly sampled configuration $q'$, it is retained with
probability $\frac{1}{w(q')}$ and added into the tree if there is a valid connection to $q$. Note that the weight of a
node $q$ is the number of nodes in the tree in $q$’s neighborhood. Another well-known single-query
method is Potential Fields [3] because an artificial potential field is created specific to the given
goal state. It defines a potential function over $C_{\text{free}}$ which has a global minimum at the goal. The
robot’s configuration is treated as a point in this artificial potential field with attracting forces to the
goal and repulsion forces from obstacles. The robot then follows the steepest descent. The path is
produced with little computation if you can dive into the solution. However, one major drawback of
this method is that it can be trapped in local minima of the potential field, and therefore might fail to
return a valid path.

1.3 Completeness in Path Planning

Complete path planners are exact and do not resort to any approximations. Due to this, they provide
an explicit representation of free configuration space. A path planner is complete in the sense that if
there exists a collision-free path, it is guaranteed to return a solution. Otherwise, it reports failure.
However, all known complete path planners (for example, visibility graph) are only suitable for
lower-dimensional configuration space such as 2D due to the requirement of explicit representation
of configuration space. In higher dimensional spaces, building a complete path planner is known to
be intractable [11]. It has been proven that time complexity for the basic complete path planning
(single robot) is exponential in the robot’s degree of freedom.

This intractability of complete path planners led researchers to weaker notations of completeness,
such as resolution completeness and probabilistic completeness [12]. Cell decomposition methods
are resolution complete in the sense that if the discretization is fine enough, it can correctly return a
path when one exists and return failure otherwise. For PRM methods, they are probabilistic complete
because the probability of not finding a path when one exists converges to zero as the number of

6
sampled configurations approaches infinity [13–15]. Both RRT and EST are also probabilistically complete [12].

1.4 Motivation of Thesis

The problem of path planning is theoretically intractable [8], but in everyday life we (as human beings) navigate in our environment without much difficulty because of the experiences learned since childhood. The learning process may be complex, but one of the main reasons that we can learn such tasks is that most objects we encounter today are similar or identical to the objects we encountered yesterday or even years ago. Environments with similar objects are quite common. For example, desks and chairs in a classroom or in an office may be moved around from one place to another frequently, but unfamiliar items are seldom introduced. Even different environments, such as two apartments, or a manufacturing factory and an airport garage, may share many similar items. The main differences are usually in the arrangements. For a dynamic environment, it can be considered as a continuous sequence of similar static environments due to obstacles’ motion coherence. In this thesis, we term this “discrete similar-workspace problem” for static environments and “continuous similar-workspace problem” for dynamic environments.

However, as we will discuss in the next chapter, almost all existing path planners fail to explore this similarity over different environments. They consider any two (similar or not) workspaces as two distinct problems and completely ignore the correspondences between them. Consequently, the entire roadmap has to be computed from scratch once the robot is placed into new workspace. Even though similar environments share similar obstacles, the data structure capturing the connectivity of free space for a particular environment is not reusable in other environments. In this thesis, I investigated path planners that address the “discrete similar-workspace problem” and “continuous similar-workspace problem” in order to gain significant improvements on both time efficiency and completeness.
1.5 Contribution of Thesis

To address this “discrete similar-workspace problem”, I proposed a path planner [16] which exploits similarity across different static environments. This planner remembers the computation (the “mental image”) for every obstacle encountered. Given a new problem, by carefully reusing these existing computation, the planner quickly builds a roadmap from its knowledge base. We also explored better new shape matching methods that can compare a new obstacle to existing models in the database more efficiently. This planner will be discussed in details in Chapter 3.

In Chapter 4, I will discuss a path planner that addresses the “continuous similar-workspace problem”. Existing methods dealing with dynamic environments have explored the temporal coherence (or similarity) and gain efficiency by preserving the bulk of the roadmap and only repairing the invalid part due to motions of obstacles. However, all these methods repair blindly and periodically at fixed time intervals with no attempt to analyze the similarity and changes of the planning spaces across different time instances [17–19]. Consequently, either updating is redundant or edges and nodes believed to be safe may be invalid to traverse. To address this, I proposed a path planner [20] that detects critical moments when the topology of the free configuration space changes, assuming that trajectories of moving obstacles are known. Our experimental results show that the proposed planner not only avoids redundant computation but also improves the chances of finding a valid path.

In many practical applications such as rescue robots and autonomous vehicles, the robot may have no prior knowledge of obstacles’ motions or shapes. The only way for it to know the environments is through on-board sensors. Existing works collect sensory data, update environmental belief and plan a path at fixed time intervals [21, 22]. This either results in redundant update or makes robot face imminent danger which is too late to avoid. Therefore, I proposed to predict the critical moments when the robot and obstacles can collide. Only at such critical moments, the robot updates its belief of the environment based on sensory data and re-plans if necessary. The main challenge in predicting collision stems from the assumption that obstacles’ motions are unknown. To provide conservative estimation, I developed a strategy that models all obstacles as adversarial agents so that obstacles will move in the way to minimize the time that the robot remains collision-free.
I believe that this thesis can provide a class of new path planners which make good use of similarity across environments and serves as a preliminary work in this new research direction.
Chapter 2: Related Work

Based on whether the obstacles in workspace are static or movable (dynamic), we classify path planning problems into the following three categories.

1. All obstacles in the environment are static.
2. The trajectory of every moving obstacle is fully known in advance.
3. The future trajectory of moving obstacle is unknown and can only be estimated based on the acquired sensor data.

In this chapter, I will review existing work in each of these three categories. Moreover, since the goal of path planning is to find a valid (collision free) path, collision detection becomes one of the most fundamental tools in path planning. Therefore, I will also review works on collision detection. A more detailed survey can be found in [23].

2.1 Path Planning in Static Environments

The problem of motion planning in static environments has been studied extensively. It is well known that any complete motion planners [11] are unlikely to be practical in high-dimensional configuration space. During the past two decades, researchers have focused on sampling-based algorithms, e.g., PRM [1, 5, 24], RRT [25] and EST [26], which work well in high-dimensional configuration spaces and are substantially easier to implement.

Recently, Karaman and Frazzoli [27] showed that these sampling-based path planners almost surely do not converge to an optimal solution. Therefore, they introduced PRM* and RRT* which extend PRM and RRT respectively to achieve asymptotic optimality property. In other words, the cost of the solution returned by PRM* or RRT* almost surely converges to optimum. Moreover, only a
constant factor more computation is required. However, both RRT and RRT* require domain-specific heuristics for growing the tree: distance metric and node expansion. Perez et al addressed this limitation and proposed LQR-RRT* to extend RRT* to domains with more complex or underactuated dynamics. Webb and van den Berg proposed Kinodynamic RRT* which generalizes RRT* to any systems with controllable linear dynamics. Kinodynamic RRT* can also be applied to non-linear dynamics with the help of first-order Taylor approximations.

Complete reviews of sampling-based methods can be found in [4, 28]. Due to the uniform sampling strategy, the presence of “narrow passages” poses significant difficulty for the classic PRM. Therefore, many of the PRM variants dealing with static environments focus on problems with narrow passages [5–7, 29]. Nevertheless, these difficult problems are usually created artificially for testing purposes, and in many real-life problems, such as planning motion in a factory or in a virtual world, are usually easier and do not contain very narrow passages. Therefore, instead of focusing on these difficult but rare problems, our work attempts to increase the planner efficiency for the easier but more commonly seen problems.

Similar environments are very common in our everyday life. For example, in two offices, desks and chairs are moved around frequently but unfamiliar objects seldom introduced. The main differences are usually in the arrangements. However, we are not aware of any planners that can efficiently handle the problem of similar environments, although there are some recent works on learning path planners. Kalisiak and van de Panne [30] proposed a faster motion planner which learns nonviable scenarios from prior runs and in the subsequent queries, by avoiding exploration of nonviable states, the planning process can be expedited. This method does not actually explore similarity among environments. Instead, it trains a viability classifier from many prior runs or randomly generated experiments. Despite of significant speedups, the viability classifier might have prediction errors which may result in misclassifying viable states and nonviable states. Berenson et al [31] proposed a framework called “lightning” due to its ability of planning quickly. With a library of previously computed paths, given a query, “lightning” first retrieves the path with the least violation of constraints and then repairs the retrieved path with respect to the new query. However, it explores similarity over ALL environments. When the total number of environments increases, the similarity
becomes less. There also exist PRM planners that identify and learn features in $\mathcal{C}_{free}$ but their goal is to determine sampling strategies [32–36] using machine learning techniques. Given similar environments, these methods still build the roadmaps from scratch. There are also methods that pre-compute and reuse configurations for highly constrained systems (e.g., closed-chain [37]) in which feasible configurations are usually difficult to obtain. These methods do not consider similarities among motion planning problems.

In the next chapter, I will discuss a path planner that addresses this “similar workspace problem” in static environments (or “discrete similar workspace problem”) and extends the existing work of PRMs with the functionality of reusing computations among similar environments.

### 2.2 Path Planning in Known Dynamic Environments

In this section, I will review works with the assumption that each dynamic obstacle moves along some known trajectory. Although this assumption was considered impractical in the past, but recent advances in motion and behavior prediction [38] provide opportunities for longer planning horizon. Moreover, many applications of this assumption can be found in mobile factory floor robots such as KIVA [39] and assembly line.

In the past couple of decades, major contributions of works on motion planning in dynamic environments include reflective navigation approaches, state-time space and graph-representation based approaches. These works also include path planning based on artificial intelligence fields, fuzzy logic, neural network, swarm intelligence, stochastic modeling and generic algorithms. Please refer to Keshmiri and Payandeh [40] for a detailed review.

In changing environments, obstacles become movable [18] or even deformable [41]. An intuitive approach to handle moving obstacles with known trajectories is to incorporate time dimension and plan in the configuration-time (CT) space $\mathcal{E}$ [42]. The CT-space $\mathcal{E}$ can be approximated by a sequence of configuration space slices at fixed times. In [8], visibility-graph algorithm is applied to generate all path segments between adjacent slices and then join adjacent solutions. Multi-robot
motion planning is an important field in this category. There exist plenty of tasks such as search and rescue, surveillance and exploration that can be performed more efficiently and robustly using multiple robots. Erdmann and Lozano-Perez introduced prioritized path planning for multiple robots in an environment with static obstacles [42]. It assigns priorities to all the robots and motion planning is performed for one robot at a time in order of decreasing priority. For each robot, its path should be collision-free with respect to the obstacles as well as previously solved robots.

To handle problems in higher dimensional space and kinodynamic constraints, probabilistic methods are usually applied to generate a graph or tree structure to approximate the CT-space. For example, probabilistic methods such as PRM and RRT can be directly extended to configuration time space [43]. The only major modification is that every edge in the roadmap must travel forward in time. Hsu and Kindel [44] produced a tree-based roadmap in configuration-time space. The tree is rooted at the initial configuration-time point and grows along the time-axis. It terminates when it falls into a region from which it is known how to get to the goal. Due to motion coherence of obstacles, the state of $\mathcal{E}$ usually does not change much during a short period of time. Obviously, the direct application of existing path planners in configuration-time space ignores this.

Many approaches take advantage of temporal coherence by employing a two-phase approach: first compute a roadmap with only respect to the static obstacles; when given a query during run time, edges are checked based on the location of the dynamic obstacles at specific times [17–19]. Jaillet and Simeon [17] fixed the invalid edges in the roadmap by applying RRT to quickly check the possible reconnections along the invalid edge. If reconnection fails, new nodes and edges are added. van den Berg and Overmars [18] proposed a two-level search strategy. At the local level, they find a feasible local path along a single edge of the roadmap. At the global level, they apply an A*-like search to coordinate the local paths. Leven and Hutchinson [45] constructed a regular grid in workspace that maps each of its cell to the roadmap nodes and edges. Once obstacles move, [45] quickly checks occupied cells and invalidates the associated nodes and edges. Li and Shie [46] proposed to reuse RRT via a new data structure called Reconfigurable Random Forest (RRF) which combines all the RRTs from previous queries. Ferguson et al. proposed the idea of Dynamic RRT [47] that keeps a single RRT and detects the parts of the tree invalidated by configuration space changes. Then these
invalid parts are removed and the modified RRT is regrown until the goal is reached again. Yang and Brock [48] introduced an interesting concept called Elastic Roadmaps for autonomous mobile manipulation. Free configurations are sampled around obstacles and for a configuration, once its associated obstacle moves, it moves with it. The roadmap is updated at a frequency of approximately 5-10Hz in their experiments.

Although almost all methods mentioned above assume motion coherence, this repairing roadmap at fixed time strategy either introduces redundant computation or misses moments when edges or nodes become invalid. In other words, it does not recognize the critical changes in the topology of free configuration space $C_{\text{free}}$.

I am going to discuss a path planner that addressed this limitation in this thesis.

### 2.3 Path Planning in Unknown Dynamic Environments

In many real life applications such as rescue robots and autonomous vehicles, the knowledge about its surroundings may often be very limited. The robot may have no prior knowledge of obstacles’ motions or even shapes and the pose of an obstacle can only be estimated through on-board sensors.

Due to little knowledge of the environment, safety becomes very important and challenging in path planning in unknown environments [49–58]. Fraichard and Asama [54] provided the formal definitions of two new concepts: inevitable collision state (ICS) and inevitable collision obstacle (ICO). If the robot is in an ICS, no matter what its future trajectory is, a collision eventually occurs with an obstacle in the environment. ICO is a set of ICS yielding a collision with a particular obstacle. Shiller et al. [52] proposed a motion planner based on Velocity Obstacles (VO) for static or dynamic environments. The time horizon for a velocity obstacle is computed based on the current positions of robot and the obstacle as well as control constraints. With this adaptive time horizon strategy, the velocity obstacle tightly approximates the set of ICS. Gomez and Fraichard [55] proposed another ICS-based collision avoidance strategy called ICS-AVOID. ICS-AVOID aims at taking the robot from one non-ICS state to another. The concept of Safe Control Kernel is introduced and it guarantees
ICS-AVOID can find a collision-free trajectory if one exists. Recently, Bautin et al. [59] proposed two ICS-checking algorithms. Both algorithms take a probabilistic model of the future as input which assigns a probability measure to the obstacles’ future trajectories. Instead of answering whether a given state is an ICS or not, it returns the probability of a state being an ICS. Wu and How [60] extended VO to moving obstacles with constrained dynamics but move unpredictably. To compute the velocity obstacle of an obstacle, it first predicts its reachable region considering all possibly feasible trajectories and then maps this reachable region into velocity space by dividing it by time.

Apparently, computation of ICS or VO (even [59, 60]) requires some information about the future in the environment. When it comes to environments whose future is completely unpredictable, methods applying ICS or VO may fail to avoid approaching collisions, while our proposed method for unknown environments can guarantee safety by only knowing the maximum velocities of obstacles.

Yoshida et al. [61] proposed an on-line replanning method with parallel planning and execution and roadmap reuse. However, this strategy is only suitable for discrete environmental changes since replanning is time consuming and the robot needs to stop frequently if replanning is not finished in time. To address this issue, Yoshida and Kanehiro [62] proposed a reactive planning approach which considers both path replanning and deformation. When environmental changes are detected, it first checks if the path can be improved by local deformation. Only when the path becomes infeasible due to obstacles in its way and local improvements do not work, replanning is applied to generate a new feasible path by roadmap reuse.

Yang and Brock [63] proposed the Elastic Roadmaps for autonomous mobile manipulation. A free configuration is sampled around obstacles and moves with its associated obstacle. Therefore, the roadmap can always maintain task-consistent constraints.

Kim and Khosla [64] introduced harmonic potential functions to address the local minimum issue in Potential Fields. Feder and Slotine [65] extended this work to dynamic obstacles moving with constant translational or rotational velocities. However, the assumption that motions of both the robot and obstacles to follow harmonic functions is too strict. To relieve this limitation, Khansari-Zadeh and Billard [66] proposed an on-line local obstacle avoidance strategy using dynamic systems. The
original motions of the robot defined by the user are specified by a continuous and differentiable dynamic system without considering any obstacles. Then given an analytical formulation describing the surface of obstacles, the original dynamic systems are locally deformed in order not to hit the obstacles.

The work closest to the spirit of our proposed method for unknown environments is by van den Berg and Overmars [67]. Their work assumes that the robot and all obstacles are discs, and it conservatively models the swept volume of an obstacle over time as a cone with the slope being its maximum velocity. In this way, no matter how the obstacle moves, it is always contained inside this cone. Therefore, the computed path is guaranteed to be collision free. However, these assumptions can be unrealistic for many applications. For obstacles with arbitrary shapes or rotation, computing their swept volumes is nontrivial.

Almost all existing works collect sensory data and update its environmental information at fixed times. Consequently, either updating is redundant or the situation is even worse if update is performed not frequently. The robot may be at some state which leads it to be in unavoidable collisions. To address this, we proposed a method which updates environmental belief when necessary by exploring temporal coherence of obstacles and predict a critical time \( t \) such that the robot is guaranteed to move safely along its current path until \( t \).

### 2.3.1 Collision Prediction

Since the robot has partial or no information about the environment, it is very difficult to plan a collision free path for it to move through a field of static or dynamic obstacles to a goal. One of the biggest challenge is to predict possible collisions with dynamic obstacles whose trajectories are unknown. There exists a lot of work which checks collisions at a sequence of fixed time steps [68–72]. For example, van den Berg et al. [68] performed collision detections at fixed time intervals (every 0.1 seconds in their experiments). Both the robot and dynamic obstacles were modeled as discs moving in the plane. Moreover, the future motions of a moving obstacle were assumed to be the same as its current motions. In order not to miss any collisions, they either increased the number
of time steps or assumed the objects move very slowly.

There are also works which adaptively changed the frequency of collision checks: collisions are more frequently checked for two objects which are more likely to collide. Hayward et al. [73], Kim et al. [74] and Hubbard [75] assumed that the maximum magnitude of the acceleration is provided for each object. Hayward et al. calculated the amount of time within which two moving spheres are guaranteed not to collide with each other. Then more attention was adaptively paid to objects which are very likely to collide. Hubbard first detected collisions between the bounding spheres of two objects. Then the pairs of objects whose bounding spheres intersect are further checked for collisions using sphere trees that represent the objects. Kim et al. [74] first computed the time-varying bound volume for each moving sphere with its initial position, velocity and the maximum magnitude of its acceleration. As time goes by, the radius of this time-varying bound volume increases and it is guaranteed to contain the sphere at any time in the future. For two moving spheres, whenever their time-varying bound volumes intersect, they are checked for actual collision. Chakravarthy and Ghose [51] proposed collision cone approach (similar as velocity obstacle) for predicting collisions between any two irregularly shaped polygons translating on unknown trajectories. All these methods are limited to discs, spheres or translational objects. Our new tool allows polygons with arbitrary shape (even non-simple polygons) with rotation.
Chapter 3: A Reusable Path Planner in Similar Static Environments

Throughout this chapter, I will focus on designing a path planner [16] that exploits similarity among static environments where obstacles are not allowed to move.

Path planning problems can be very difficult [76] especially when it comes to environments with narrow passages. However, we, as human beings, can solve most path planning problems easily. One of the reasons is that most objects we encounter today are identical or similar to the objects we encountered yesterday or even years ago. That is, we as human beings, remember how to navigate around or manipulate similar objects using similar strategies. Therefore, we are interested in developing path planners that mimic this simple observation.

Similar environments are very common in everyday life. For example, desks and chairs in a classroom or in an office may be moved around from one place to another frequently, but unfamiliar items are seldom introduced (Fig. 3.1). Even different environments, such as a manufacturing factory and an airport garage, may share many similar items. The main differences are usually in the arrangements. Therefore, it is natural to define similarity among environments by the shapes of obstacles. A path planner that exploits the similarity between its workspaces can provide significant efficiency improvements.

We designed a method to address a closely related but slightly different question: how much pre-processing can be done to solve a family of motion planning problems with similar environments? If a significant portion of the computation can be done off-line, we can pre-process a very large set of geometric models and store the computation in a database for fast retrieval. An important consequence of this is that almost all (either know or unknown) path planning problems can be solved more efficiently.

We are not aware of any planners that can efficiently handle the problem of similar environments.
There exist PRM planners that identify and learn features in \( C_{\text{free}} \), but their goal is to determine sampling strategies \([32–36]\) using machine learning techniques. Given similar environments, these methods still build the roadmaps from scratch. There are also methods that pre-compute and reuse configurations for highly constrained systems (e.g., closed-chain [37]) in which feasible configurations are usually difficult to obtain. None of them consider similarities among motion planning problems. There are also some recent works on learning path planners \([30,31]\). Kalisiak and van de Panne [30] proposed a faster motion planner which learns nonviable scenarios from prior runs and in the subsequent queries, by avoiding exploration of nonviable states, the planning process can be expedited. Berenson et al [31] proposed a path planner called “lightning” which keeps a library of previously computed paths. Given a new query, it first retrieves the path with the least violation of constraints and then repairs the retrieved path with respect to the new query. However, none of them explores the similarity among environments efficiently.

Our method addresses these limitations and can also be viewed as a type of “self-improving” algorithm \([77,78]\). Existing self-improving planners consider the performance for a single environment with multiple queries, e.g., \([46,79]\), but do not consider the performance improvement across different environments.
Figure 3.2: Two similar workspaces sharing the same robot and several similar obstacles. The red and blue objects indicate start and goal positions. Similar workspaces are usually viewed as completely different problems.

3.1 General Overview

The new path planner is called **ReUsable Probabilistic Roadmap Methods** (RU-PRM for short) which explores similarity among different static environments. For example, the two environments in Fig. 3.2 share the same robot and several similar obstacles. Most existing path planners treat these two problems individually while our new path planner takes advantage of their close resemblance. If the path planning problem in Fig. 3.1 has been solved before, RU-PRM can solve Fig. 3.1 more efficiently by reusing the computation from Fig. 3.1.

For every obstacle $\mathcal{O}_i \in \mathcal{O}$, RU-PRM first constructs a roadmap around $\mathcal{O}_i$ to approximate the free configuration space in the proximity of $\mathcal{O}_i$. Since it only captures the free space in its associated obstacle’s neighborhood, such a roadmap is called a local roadmap. RU-PRM keeps the local roadmap of every previously encountered obstacle in a database (Fig. 3.1). For an unknown environment, RU-PRM also provides a mechanism to transform the local roadmaps to reflect the new arrangement of obstacles. When a new environment is given, RU-PRM first matches the obstacles in this environment to the existing models in the database. Then the local roadmaps associated with each matched model is transformed (Fig. 3.1) and merged into a global roadmap (Fig. 3.1). At last, it searches in this
Figure 3.3: (a) A database of pre-computed local roadmaps. (b) Transformed local roadmaps in a new environment. (c) A global roadmap is composed of the local roadmaps.

global roadmap for a collision-free path to navigate the robot from start to goal. The following algorithm summarizes the main steps of RU-PRM.

Algorithm 3.1.1: RU-PRM($\mathcal{O}, \mathcal{R}, Q$)

**comment:** Obstacles $\mathcal{O}$, Robot $\mathcal{R}$ and query $Q$

**for each** $\mathcal{O}_i \in \mathcal{O}$

$$
\begin{cases}
\text{if } \not\exists M_{\mathcal{O}_i}, \text{ an local roadmap of } \mathcal{O}_i \\
\text{then }
\begin{cases}
\text{Create}(M_{\mathcal{O}_i}) \\
\text{Store}(M_{\mathcal{O}_i}) \\
\text{Read}(M_{\mathcal{O}_i}) \\
\text{Transform}(M_{\mathcal{O}_i})
\end{cases}
\end{cases}
$$

$M_{\mathcal{O}} \leftarrow \text{Merge}(\cup_i \{M_{\mathcal{O}_i}\})$

Query($M_{\mathcal{O}}, Q$)

In Algorithm 3.1.1, the sub-routines Store($\cdot$), Read($\cdot$) and Query($\cdot$) are straightforward. The other three main subroutines: Create($\cdot$), Transform($\cdot$) and Merge($\cdot$) will be discussed in details later.
3.2 Assumptions

RU-PRM provides benefits based on the following assumptions.

1. The robot remains the same across environments with similar obstacles.
2. The unknown workspace $\mathcal{W}_i$ has high correspondences to other known workspaces $\mathcal{W}_1 \cdots \mathcal{W}_{i-1}$.
3. A large storage space is available to store all local roadmaps.
4. Pre-processing time is available. So RU-PRM is an off-line planner.

These assumptions are general enough to cover many practical situations. The first two assumptions are from the observation that many path planning problems in real life and in virtual worlds share many similar items in their workspaces. Moreover, the type and the number of the robots used in these environments, e.g., characters in a game or robot arms in a factory, are usually limited and do not change often. Other assumptions are also supported by the current technologies. Most off-the-shelf hard-drive discs with Tera-byte capacity can be obtained for just a few US dollars. Multi-core processors are becoming cheaper and allow more background computation for creating a database of local roadmaps.

3.3 Create Local Roadmaps

Creating local roadmaps or local roadmaps around $\mathcal{C}$-obstacles enables reusability of computation given identical or similar obstacles. RU-PRM can work with any samplers such as PRM, OBPRM and Gaussian PRM. I will briefly describe an approach based on Minkowski sum (called MSUM-PRM [80]) and its advantages over other existing samplers.

To generate the local roadmap $M_{\mathcal{O}_i}$ for an obstacle $\mathcal{O}_i \in \mathcal{O}$, MSUM-PRM samples configurations by computing the Minkowski sum of the robot and $\mathcal{O}_i$. It is known that the contact space of a translational robot is the boundary of Minkowski sum of the obstacle and the negated copy of the robot [81]. Although it is difficult to compute the exact Minkowski sum of polyhedra [82], we have
shown that sampling points from the Minkowski sum boundary without explicitly generating its mesh boundary can be done much more easily and efficiently [83]. To handle robots with not just translations, we can simply draw samples from a sequence of Minkowski sums, each of which is constructed by randomly assigning different orientations and joint angles to the robot.

Lien [80] proved that MSUM-PRM does not only generate a free configuration significantly faster than other previously mentioned samplers, but also provides more powerful local planners based on the geometric properties of Minkowski sum. Moreover, a configuration generated by MSUM-PRM can be transformed easily because the configuration is represented by a pair of points from the robot and an obstacle. This is very important advantage for RU-PRM and it is missing from other existing samplers.

### 3.3.1 Transform Local Roadmaps

If an unknown obstacle $X$ matches to an existing model $O_i$ by translating, rotating and scaling $O_i$, we consider how these transformations can affect $O_i$’s local roadmap $M_{O_i}$. In other words, we seek methods that transform $M_{O_i}$ to approximate the local roadmap of $X$.

Let $T$, $R$ and $S$ be the translation, rotation and uniform scale applied to $O_i$, respectively. Given a configuration $c$ from $M_{O_i}$, our goal is to obtain its corresponding configuration $c'$ so that $c'$ is free and it is in the proximity of $X$.

To make it simpler, we assume that (1) when $M_{O_i}$ is computed, the center of $O_i$ is placed at the origin of the coordinate system and (2) the robot is a free-flying articulated robot. A configuration $c$ is composed of three components $(c_T, c_{BR}, c_{JR})$, where $c_T$ and $c_{BR}$ are the values of the translational and rotational degrees of freedom for the base, respectively. $c_{JR}$ represents the rotational degrees of freedom of the joints, if any.

When we consider only translation $T$ and rotation $R$, the corresponding $c'$ is obtained in the following way.
\[ c' = (c'_T, c'_BR, c'_JR) = (T + R \cdot c_T, R \cdot c_{BR}, c_{JR}) \, . \]  

(3.1)

Note that \( T \) and \( R \) have no effect on \( c_{JR} \).

When uniform scale \( S \) is considered, the only component of \( c \) effected by \( S \) is \( c_T \). Because the contact space of the robot and \( \mathcal{O}_i \) is represented by Minkowski sum operation, \( c_T \) can be always decomposed so that \( c_T = r + o \) where \( r \) and \( \mathcal{O}_i \) are points from \( -\mathcal{R} \) and \( \mathcal{O}_i \), respectively. Here \( -\mathcal{R} \) is the negative copy of \( \mathcal{R} \). When both \( \mathcal{R} \) and \( \mathcal{O}_i \) are convex or \( \mathcal{O}_i \) is convex and \( S \) shrinks \( \mathcal{O}_i \), we use the following to obtain \( c'_T \).

\[ c'_T = r + S \cdot o \, . \]  

(3.2)

If \( \mathcal{O}_i \) and \( \mathcal{R} \) are not convex or if \( \mathcal{O}_i \) is convex but \( S \) enlarges \( \mathcal{O}_i \), in order to match \( X \), we can work around this problem using convex decomposition. Alternatively, if we apply Equation 3.2 to non-convex shapes, it is possible that \( c' \) makes \( \mathcal{R} \) collide with \( X \). Therefore, collision detection is used to check the feasibility of every transformed configuration.

### 3.4 Merge Local Roadmaps

So far, we have treated each obstacle independently. After the pre-processing step that either loads or generates local roadmaps, we proceed to compose a global roadmap.

The configurations in a (transformed) local roadmap, although are near the surface of the associated obstacle, may be outside the bounding box or colliding with other obstacles. We validate each configuration in a local roadmap using collision detection. For edges connecting two collision-free configurations, we evaluate them in a lazy manner [84], i.e., we only check the feasibility of the edges in the extracted paths during the query phase.

After all configurations are verified, we merge the local roadmaps in a pairwise manner. For every
3.4.1 Merge Local Roadmaps Using Boundary Nodes

The merging operation can be optimized by classifying the relationships of the \( \mathcal{O} \)-obstacles. Let \( \mathcal{O}_i \) and \( \mathcal{O}_j \) be a pair of obstacles in the workspace. We classify their relationships in \( \mathcal{C} \) based on the feasibility of the configurations in their local roadmaps, \( M_{\mathcal{O}_i} \) and \( M_{\mathcal{O}_j} \). More specifically, we identify the boundary nodes in the local roadmaps. Let \( B_{\{\mathcal{O}_i, \mathcal{O}_j\}} \) be a set of boundary nodes in \( M_{\mathcal{O}_i} \) w.r.t. \( \mathcal{O}_j \).

For a configuration \( c \in M_{\mathcal{O}_i} \), we say \( c \in B_{\{\mathcal{O}_i, \mathcal{O}_j\}} \) if and only if \( c \) does not make the robot collide with \( \mathcal{O}_j \) but at least one its adjacent nodes does. Fig. 3.4 shows an example of the boundary nodes. Note that \( B_{\{\mathcal{O}_i, \mathcal{O}_j\}} \neq B_{\{\mathcal{O}_j, \mathcal{O}_i\}} \) unless they are both empty. We use the boundary nodes to classify and
connect the local roadmaps. There are three cases to consider.

1. $M_{O_i}$ and $M_{O_j}$ are far away from each other, i.e., $B_{\{O_i,O_j\}} = B_{\{O_j,O_i\}} = \emptyset$.

2. $M_{O_i}$ and $M_{O_j}$ overlap, i.e., $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} \neq \emptyset$.

3. $M_{O_i}$ and $M_{O_j}$ are near each other, i.e., $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} = \emptyset$ or vice versa.

**Case 1.** When the $C$-obstacles of $O_i$ and $O_j$ are far away from each other, we connect their local roadmaps using the method described at the very beginning of this section, which simply connects the $k$-closest pairs between the local roadmaps.

**Case 2.** This is the situation that the $C$-obstacles of $O_i$ and $O_j$ overlap. The local roadmaps are connected by adding edges between $B_{\{O_i,O_j\}}$ and $B_{\{O_j,O_i\}}$. Fig. 3.4 shows an example in this case.

**Case 3.** This situation requires us to combine the techniques for **Case 1** and **Case 2**. For $B_{\{O_i,O_j\}} \neq \emptyset$ and $B_{\{O_j,O_i\}} = \emptyset$, local roadmaps are connected by adding edges between $c_i \in B_{\{O_i,O_j\}}$ and its $k$ closest nodes $c_j \in M_{O_j}$. For $B_{\{O_i,O_j\}} = \emptyset$ and $B_{\{O_j,O_i\}} \neq \emptyset$, local roadmaps are merged in the same manner.

### 3.5 Experimental Results

In this section, we show experimental results. All the experimental are performed on a PC with two Intel Core 2 CPU at 2.13 GHz with 4 GB RAM. Our path planner is implemented in C++.

We tested RU-PRM on three sets of similar environments (Fig. 3.5). These environments are designed to be related to practical applications. Environments (a-d) in Fig. 3.5 mimic environments we could encounter in everyday life. For example, in a classroom or in an office, we may need to walk around several obstacles in order to reach goal and these environments usually do not contain narrow passages. Environments (e-f) in Fig. 3.5 could find its applications in a disaster zone such as the scene of a fire where the rescue robot has to pass through narrow passages (or doors) to rescue people. Moreover, these environments are challenging enough for classic PRMs including Uniform [1] and
Figure 3.5: Environments used in the experiments. (a,b) 2D workspaces with an articulated robot. (c,d) Similar workspaces with a rigid robot and cubes. (e,f,g) Similar workspaces with a U-shaped rigid robot and walls.
Figure 3.6: Experimental results for the environments in Fig. 3.5. All results are collected over 30 runs. (a), (c) and (e) plot the average running time for each planning method over 30 runs. The y-axes are all in logarithmic scale. We stop the planner when it takes more than $10^8$ milliseconds. The success rate of each planning method is plotted in (b), (d) and (f).
Gaussian [6]. It is assumed that the robots and obstacles in these environments are known, therefore we can pre-process them and store their ob-maps in a database. We employ two types of RU-PRMs in our experiments depending on how a local roadmap is constructed: (1) RU-PRM with Gaussian PRM and (2) RU-PRM with MSUM-PRM. During the pre-processing step, RU-PRM with Gaussian PRM takes about 4.9, 96 and 382 seconds to create and store the ob-maps for obstacles in environments (a-b), (c-d) and (e-g) in Fig. 3.5, respectively. RU-PRM with MSUM-PRM takes about 35 and 131 seconds to create and store the ob-maps for obstacles in environments (c-d) and (e-g) in Fig. 3.5, respectively. Because currently MSUM-PRM has not been implemented to handle 2D workspaces, RU-PRM with MSUM-PRM is not used in environments (a-b) in Fig. 3.5. In these experiments, we compare RU-PRMs with several well-known planners: MSUM-PRM [80], Uniform [1], Gaussian [6] and Visibility PRMs [85].

**RU-PRM is significantly faster** The running times for all three groups of environments are shown in Fig. 3.6 including average running time over all 30 runs and success rate. The success rate is the number of runs that robot reaches the goal over total number of runs. Notice that the y-axes of the charts in Fig. 3.6 are in logarithmic scale. The running time for RU-PRMs includes the time for reading the ob-maps from the hard disk drive, as well as the time for transforming, evaluating and merging the ob-maps and the time for performing the query in the global roadmap.

From the results, RU-PRM shows significant efficiency improvement in all studied environments. Environments (a-d) in Fig. 3.5 are simple environments where the Uniform PRM usually solves the problems with a few hundred nodes and outperforms Gaussian and Visibility PRMs and MSUM-PRM. In these simple environments, RU-PRM with MSUM-PRM or Gaussian PRM still provides noticeable improvements (up to 100 times faster) over the Uniform PRM (and therefore over Gaussian PRM and MSUM-PRM). This evidence demonstrates the strength of reusing computation. These plots also show that combining RU-PRM with either MSUM-PRM or Gaussian PRM does not seem to affect its performance in simple environments, however, the difference becomes more noticeable in more difficult environments.

In more difficult environments, e.g., environments (e-g) in Fig. 3.5 which contain narrow passages,
RU-PRM with MSUM-PRM or Gaussian PRM is significantly faster (by $5\sim8$ orders of magnitude) than Uniform and Gaussian PRMs and is still significantly faster than using MSUM-PRM (by $2\sim5$ orders of magnitude). A possible source of the improvement is from MSUM-PRM, which has been shown to be better than the classic PRMs [80]. However, our results also show that combining RU-PRM with MSUM-PRM further improves MSUM-PRM. The main reason for this performance improvement is obvious. Obtaining connectivity in the free $\mathcal{C}$ is usually the most time-consuming step in PRMs, RU-PRM saves significant amount of time by reusing the connectivity provided by the ob-maps.

To prove that RU-PRM always performs more efficiently than other methods in all 30 runs and the lower average running time is not due to some “lightning” runs, we also report standard deviation of running times. The following table shows the ratio of standard deviation over average running time for every environment in Fig. 3.5. From the reported ratios, we claim that there is not much difference in running times between any two runs and an average running time can represent how efficiently RU-PRM performs in a single run.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Method</th>
<th>Standard deviation/Average running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3.5 (a)</td>
<td>RU-PRM with Gaussian PRM</td>
<td>0.03</td>
</tr>
<tr>
<td>Fig. 3.5 (b)</td>
<td>RU-PRM with Gaussian PRM</td>
<td>0.018</td>
</tr>
<tr>
<td>Fig. 3.5 (c)</td>
<td>RU-PRM with MSUM-PRM</td>
<td>0.031</td>
</tr>
<tr>
<td>Fig. 3.5 (d)</td>
<td>RU-PRM with MSUM-PRM</td>
<td>0.033</td>
</tr>
<tr>
<td>Fig. 3.5 (e)</td>
<td>RU-PRM with MSUM-PRM</td>
<td>0.002</td>
</tr>
<tr>
<td>Fig. 3.5 (f)</td>
<td>RU-PRM with MSUM-PRM</td>
<td>0.004</td>
</tr>
<tr>
<td>Fig. 3.5 (g)</td>
<td>RU-PRM with MSUM-PRM</td>
<td>0.03</td>
</tr>
</tbody>
</table>

### 3.6 Shape Matching

So far, we have assumed that the same obstacle (maybe with a different size) can be found in the database. To reuse ob-maps for different but similar obstacles, we investigate shape matching methods to match similar shapes.
3.6.1 Shape Filter

If two models look very different, we do not bother to compare them at all. Therefore, a coarse filter is built based on the geometric features including compactness, convexity and fatness. The compactness of a 3D shape is the ratio of its volume and surface area. While for a 2D shape, it is the ratio of area and perimeter. The convexity of a 3D (or 2D) shape is the percentage of the convex hull volume (or area) occupied by the shape. The fatness of a 3D (or 2D) shape defined as the ratio of the shape volume and the ball volume with the same diameter. This filter mechanism reduces unnecessary shape matching work and improves efficiency.

3.6.2 2D Matching Methods

We designed a new method to match two models in 2D. Since the shape descriptor for a 2D model is a number of convex polygons, in the preprocessing step, each shape is first decomposed into a set of approximately convex pieces using approximate convex decomposition (ACD)[86]. Then the convex hull for each component is computed [87]. In the matching step, for every component of one model, the work of morphing it to every component in the other model is measured. Then the difference of these two shapes is computed by searching for the best bipartite matching between these two sets of convex components.

**Shape Decomposition** A shape is called approximately convex if its concavity is lower than some user defined tolerance. It has been shown that ACD produces much fewer components compared to the exact decomposition and, more importantly, it maintains the key structural features. Due to these advantages, we can represent a shape using the convex hulls of its components from ACD. An example of ACD is shown in Fig. 3.7 (a) and Fig. 3.7 (b).

**Dissimilarity Measurement** For two polygons \( P \) and \( Q \), let \( \{ P_i \} (1 \leq i \leq m) \) and \( \{ Q_j \} (1 \leq j \leq n) \) be the sets of components produced by ACD. We use \( \{ CP_i \} (1 \leq i \leq m) \) and \( \{ CQ_j \} (1 \leq j \leq n) \) to denote the convex hulls of \( \{ P_i \} \) and \( \{ Q_j \} \), respectively. To measure the difference between \( P \) and \( Q \), we first compute the dissimilarities for all pairs of convex hulls from \( \{ CP_i \} (1 \leq i \leq m) \) and \( \{ CQ_j \} (1 \leq j \leq n) \).
Figure 3.7: An example of the proposed shape matching method.
(1 ≤ j ≤ n). Given two convex hulls \( CP_i \) and \( CQ_j \), their dissimilarity is measured as the morphing distance from one to the other in the following way (Fig. 3.7 (c) and Fig. 3.7 (d)).

1. Align \( CP_i \) and \( CQ_j \) by overlapping their centers and their principal axes (Fig. 3.7 (c)).

2. Draw \( m_i \) rays from the common center passing through every vertex of \( CP_i \) with \( m_i \) being the number of vertices in \( CP_i \). Similarly, draw \( n_j \) rays from the common center passing through every vertex of \( CQ_j \) with \( n_j \) being the number of vertices in \( CQ_j \) (Fig. 3.7 (d)).

3. For each ray in \( CP_i \), a line segment between the vertex of \( CP_i \) it passes through and the intersection with \( CQ_j \) is identified (green dots in Fig. 3.7 (d)). Similarly, for each ray in \( CQ_j \), a line segment between the vertex of \( CQ_j \) it passes through and the intersection with \( CP_i \) is identified (blue dots in Fig. 3.7 (d)).

4. The morphing distance between \( CP_i \) and \( CQ_j \) is the sum of the lengths of all these \( m_i + n_j \) line segments.

Once the morphing distances between all pairs of convex hulls in \( \{ CP_i \} \) and \( \{ CQ_j \} \) are computed, we arrange the components \( \{ CP_i \} \) and \( \{ CQ_j \} \) into a bipartite graph and solve the minimum bipartite matching problem.

Although there exist many shape matching methods (e.g., survey [88]), this new method provides unique functionalities for RU-PRM. In particular, it represents a shape using a set of convex objects. This not only allows the scale transformation discussed in Section 3.3.1, but also allows sub-part matching. For example, the ob-map for a hand gesture can be “deformed” to fit around another gesture by transforming the fingers even if the gestures are very different.

Shown in Fig. 3.8, the proposed method correctly returns its best match for every test case using 10
randomly selected polygons. Moreover, it takes 70 ms on average for a polygon to find the best match. Therefore, even when we consider the time spent on shape matching, RU-PRM still outperforms the other three planners.

### 3.6.3 3D Matching Methods

To match two models in 3D, we applied the matching methods discussed in [89]. It constructs a shape signature for each 3D polygonal model and reduces the shape matching problem to comparison of two shape signatures. The shape signature here is the distribution of a shape function.

**Shape Function** In [89], it proposed five simple shape functions which are illustrated by Fig. 3.9.

- **A₃**: Samples three points randomly on the surface of the model and measures the angle between two segments formed by these three points.
- **D₁**: Samples one single point randomly on the surface and measures its distance to the center of the model.
- **D₂**: Samples two points randomly on the surface and measures their distance.
- **D₃**: Samples three points randomly on the surface which will form a new triangle and measures the square root of the area of this triangle.
- **D₄**: Samples four points randomly on the surface which will form a new tetrahedron and measures the cube root of the volume of this tetrahedron.

All these shape functions are easy to compute and invariant to translations and rotations. Moreover, A3 is invariant to scale.

**Shape Distribution** To make the shape signature more accurate, a large number of (N) distances (angles, areas or volumes, depending on what shape function is chosen) are sampled. Then a histogram containing (B) fixed-size bins is constructed with each sample falling into one of these bins. To make the shape distribution invariant to scale, an extra normalization operation is performed on the histogram. Shown in Fig. 3.10, the minimum sample values are aligned.
Figure 3.9: Five different shape functions for a cube. The red dot is the center of the model. Dashed lines are invisible from the front side. (a) Randomly sample three points (in black) on the surface of the model and measure the angle (in blue) between two line segments (in green) formed by these three points. (b) Randomly sample one single point on the surface and measure its distance to the center of the model. (c) Randomly sample two points on the surface and measure their distance. (d) Randomly sample three points on the surface which form a triangle and measure the square root of the area of this triangle. (e) Randomly sample four points on the surface which form a tetrahedron and measure the cube root of the volume of this tetrahedron.

Figure 3.10: Normalization by alignment of min sample values. The vertical $n$-axis indicates number of samples in each bin and horizontal $f$-axis indicates the value of shape function.
From the normalized histogram, we construct a piece-wise linear shape distribution function with \( V \) equally spaced points. In other words, each point in this function contains the information from \( B/V \) bins.

**Compare Shape Distribution** To compare the shape distributions, [89] talked about four different kinds of dissimilarity functions: \( \mathcal{X}^2 \) statistic, Bhattacharyya distance, probability density functions (pdfs) and cumulative distribution functions (cdfs). Suppose \( f \) and \( g \) are the two shape distributions to be compared.

- \( \mathcal{X}^2 \) statistic: \( D(f, g) = \int \frac{(f - g)^2}{f + g} \).
- Bhattacharyya distance: \( D(f, g) = 1 - \int \sqrt{fg} \).
- PDF \( L_N \): \( D(f, g) = (\int |f - g|^{N})^{1/N} \).
- CDF \( L_N \): \( D(f, g) = (\int_{\infty}^{\infty} f - \int_{\infty}^{\infty} g)^{1/N} \).

To make it simple, we only apply PDF \( L_1 \) which is \( \int |f - g| \).

### 3.6.4 Experiments

We have tested RU-PRM combined with matching methods in two sets of environments. The first set of environments in Fig. 3.11 are designed to mimic bedrooms which we encounter every day. The second set of environments in Fig. 3.12 are designed to mimic a disaster zone such as the scene of a fire where the rescue robots have to pass through narrow passages (e.g. doors or windows). In order to test the performance of shape matching methods in RU-PRM, we created three groups of environments as Fig. 3.11. Take the environment 1 (the leftmost column) in Fig. 3.11 as an example. It is composed of three environments (a), (d) and (g). Both (d) and (g) have the same arrangement of obstacles as (a). However, in (d), two new but similar objects are introduced: bed and bookshelf. Therefore, if (a) is solved first, shape matching is required in order to reuse the local roadmaps of the old bed and bookshelf from (a). In (g), there are even more different objects than (d). So more shape-matching work is required for reusability. Similarly, in the middle column of Fig. 3.11,
Figure 3.11: Three groups of environments with similar obstacles. Take the leftmost column as an example. Environments (d) and (g) have the same arrangement of obstacles as Environment (a). In (d), it introduces two different but similar objects: bed and bookshelf. Therefore, shape-matching is required in order to reuse local roadmaps of bed and shelf in (a). In (g), there are even more different obstacles compared to (d). So more shape-matching work is required for reusability. Similarly, in the middle column, environments (b), (e) and (h) share the same arrangement of obstacles. Compared to (b), (h) has more different obstacles than (e). In the rightmost column, environments (c), (f) and (i) share the same arrangement of obstacles. Compared to (c), (i) has more different obstacles than (f).
Figure 3.12: Three groups of environments with similar obstacles. These environments are supposed to be harder than those in Fig. 3.11 since they contain articulated robots and narrow passages. In the leftmost column, environments (a), (d) and (g) share the same arrangement of obstacles. Compared to (a), (d) contains more different obstacles than (g). Therefore, more shape-matching is required for (g) if the local roadmaps from (a) are to be reused. Similarly, in the middle column, environments (b), (e) and (h) share the same arrangement of obstacles. Compared to (b), (h) has more different obstacles than (e). In the rightmost column, environments (c), (f) and (i) share the same arrangement of obstacles. Compared to (c), (i) has more different obstacles than (f).
Figure 3.13: Experimental results from the environments in Fig. 3.11. The average running time is plotted in (a), (c) and (e). The y-axes are all in logarithmic scale. The success rate of each planning method is plotted in (b), (d) and (f).
Figure 3.14: Experimental results from the environments in Fig. 3.12. The average running time is plotted in (a), (c) and (e). The y-axes are all in logarithmic scale. The success rate of each planning method is plotted in (b), (d) and (f).
environments (b), (e) and (h) share the same arrangement of obstacles. Compared to (d), (h) contains more different objects than (e). So more shape matching is required for (h). In the rightmost column, environments (c), (f) and (i) share the same arrangement of obstacles. Since both (f) and (g) contains new but similar items compared to (c), shape matching is required for both (f) and (g). However, (g) contains more new items so more shape matching will be performed.

We also created another three groups of environments with the same idea as shown in Fig. 3.12. Since every environment contains an articulated robot and plenty of narrow holes, the problems will be much harder than those in Fig. 3.12.

All the experimental are performed on a PC with two Intel Core 2 CPU at 2.13 GHz with 4 GB RAM and all shape matching methods are implemented in C++. The running times for Fig. 3.11 and Fig. 3.12 are plotted in Fig. 3.14. The difference tolerance $\tau$ is set to be 0.19. This tolerance means that two models are considered similar if their dissimilarity is below $\tau$. Each running time for all PRMs is collected over 30 runs.

**Shape matching improves reusability** Like what we have discussed in Section 3.5, from the plots in Fig. 3.14, RU-PRM with either MSUM-PRM or Gaussian PRM provides significant efficiency improvements over MSUM-PRM only or Gaussian PRM only. More importantly, take Fig. 3.14 (a) as an example. Since we use shape matching for new but similar obstacles in environments Fig. 3.11 (d) and Fig. 3.11 (g), all local roadmaps from environment Fig. 3.11 are still reusable. Therefore, it does not need to create new local roadmaps which is very time-consuming for Fig. 3.11 (d) and (g). From the plots, we can see that shape matching not only improves reusability of existing computation, but also maintains high success rates (kept at 100% in these experiments) for RU-PRM.

**Dissimilarity tolerance affects reusability and success rate** We are also interested in seeing how the results are affected by different values of tolerance $\tau$. The experiments are performed with three values of $\tau$: 0.1, 0.19 and 0.28. The larger $\tau$ is, the more reusable existing computation is. For example, in these two sets of environments (Fig. 3.11 and Fig. 3.12), the total number of obstacles is twenty-one. If the tolerance $\tau$ is set to be 0.28, there are only five local roadmaps saved in the database because for any other obstacle, its local roadmaps could be approximated using one of
these five local roadmaps. When the tolerance is set to be 0.1, there are thirteen local roadmaps saved in the archive. However, a larger $\tau$ may introduce a worse approximation of local roadmap for a new obstacle. Therefore, the success rate could also be affected. For example, when $\tau$ is 0.28, since we reuse the local roadmaps of obstacles from Fig. 3.11 for environments in Fig. 3.12, the free configuration space is not represented well and the robot cannot find a valid path to goal in Fig. 3.12 (c).

Shape matching takes a very small portion of running time To see if shape matching reduces the efficiency of RU-PRM, we also check how much time RU-PRM spends on shape matching in each run. Let us take a deeper look at the cases where $\tau$ is equal to 0.1 since there are many more obstacles saved in the archive than when $\tau$ is 0.28. Given a new environment, every obstacle will be compared to all thirteen obstacles corresponding in the archive. Take the environment Fig. 3.12 (i) as an example because it contains the maximum number of obstacles. For every obstacle, shape matching can be performed in 10 milliseconds. The average running time of a single run of RU-PRM for environment Fig. 3.12 (i) is 297148 milliseconds. Therefore, RU-PRM spends only 0.13% of the total running time on shape matching.

3.7 Discussion

In this chapter, we discussed the first path planner RU-PRM that explores similarity among different environments. Essentially, RU-PRM stores the local roadmap built around each $C$-obstacle. When a new environment is given, RU-PRM matches the obstacles and loads the matched roadmaps. The roadmaps are transformed and then merged to solve the queries in the new environments. We also talked about a new shape matching method in 2D that increases the reusability of local roadmaps. Although we consider this work as a preliminary that provides a proof of the concept in this research direction, our experimental results are very encouraging and show significant efficiency improvements.

Many aspects in the proposed work can be improved. One of the most critical bottlenecks of our
current implementation is the efficiency of the shape matching method. We also plan to investigate better criteria for evaluating the connectivity of a local roadmap.
Chapter 4: Path Planning in Fully Known Dynamic Environments

In this chapter, we extend RU-PRM to a workspace consisting of both static and dynamic obstacles. Due to motion coherence, a dynamic environment can be treated as a “continuous” version of similar environments. In other words, if we take snapshots of a dynamic environment at some very high frequency, the consecutive shots can be considered as a sequence of similar environments with close resemblance. It is assumed that each dynamic obstacle moves along some fully known trajectory with bounded velocities. Although this assumption is considered impractical in the past, there are recent evidences showing that considering the motions of dynamic objects can increase long-term optimality, such as energy efficiency and safety [90]. Recent advances in motion and behavior prediction [38] also provide opportunities for longer planning horizon. In addition, there are more and more examples of mobile factory floor robots, such as KIVA [39] (Fig. 4.1 (b)), whose motions are fully known. Other examples include service robots operating during maintenance period of a nuclear power plant and automated guided vehicles in a factory. Virtual prototyping, such as assembly/disassembly and part removal, is another important domain that usually considers the motion of the moving parts to be known or predictable [91] (Fig. 4.1 (a)).

Since the 1980s, there have been extensive work on planning motion in the environment whose state is fully known at any given time. An intuitive approach is to formulate the problem in configuration-time space (CT-space) by incorporating the time dimension \( T \) to the configuration space \( C \), denoted as \( E = C \times T \). However, the idea of motion coherence is missing from the direct extension of probabilistic methods to \( E \). There do exist several path planners that consider temporal coherence [17, 19, 46, 47]. The main idea of these planners is to repair the invalid portion of the roadmaps due to the motion of the obstacles at fixed time intervals. Consequently, this repairing roadmap at fixed time strategy does not recognize the critical changes in the topology of free configuration space \( C_{\text{free}} \).

To address this limitation, by exploring motion coherence, our new approach CriticalRoadmap detects
Figure 4.1: The assumption that trajectories are fully known can be found in many practical applications such as (a) assembly line planning and (b) mobile factory floor robots KIVA.

all the “critical moments” when the topology of free configuration space $C_{\text{free}}$ changes. At every such moment, CriticalRoadmap updates the roadmap in an efficient way to reflect the current topology of $C_{\text{free}}$. More specifically, by employing RU-PRM [16], a roadmap in $C$-space is composed of several local roadmaps with each constructed for one obstacle. Then, the critical moments of $C_{\text{free}}$ are approximated by determining the status changes of the nodes in local roadmaps. Consequently, CriticalRoadmap not only avoids redundant computation, but also improves the chances of finding a solution.

CriticalRoadmap has the following nice properties. First, it provides better efficiency. Compared to PRM and RRT, CriticalRoadmap exploits motion coherence and reuses the valid edges and nodes. Compared to the methods which update roadmaps at fixed times, CriticalRoadmap performs updating only at critical moment and therefore avoids redundant computation. The experimental results show that it is at least one order of magnitude faster than PRM, RRT and the fixed time interval strategy. Second, it provides better completeness because it focuses on capturing the changes of the topology of $C_{\text{free}}$. Third, like most probabilistic methods, it can handle problems with high dimensional $C$. 

45
4.1 Preliminaries

In this section, we define important notations used throughout the chapter. Given a set of moving obstacles $\mathcal{O}$ and a single robot $\mathcal{R}$ with bounded velocity, the trajectory of each obstacle $\mathcal{O}_i \in \mathcal{O}$ is assumed to be fully known. During a given time interval, $\mathcal{O}_i$ maintains a constant translational velocity $\vec{v}$ and a constant angular velocity $\vec{\omega}$. In some applications such as prioritized path-planning for multiple robots, $\mathcal{O}_i$ can be the robots with higher priorities than $\mathcal{R}$. Our goal is to find a sequence of collision-free configurations so that $\mathcal{R}$ can reach its destination before a user-defined time $t_f$. We use $\mathcal{R}_c$ to denote the robot configured at some configuration $c$.

To solve this problem, CriticalRoadmap represents the free CT-space $\mathcal{E}_{free}$ by identifying the topological changes in the free $\mathcal{C}$-space $\mathcal{E}_{free}$. Because constructing a complete and exact representation of $\mathcal{E}_{free}$ is intractable, the data structure that we will be using is a roadmap $\mathcal{M}$ in $\mathcal{E}_{free}$. The roadmap is composed of a sequences of critical roadmaps $\mathcal{G}_t$ which is constructed to represent $\mathcal{E}_{free}$ at critical times $t$. Each critical roadmap is then composed of a set of local roadmaps $\mathcal{M}_{\mathcal{O}_i}$ that are constructed at $t = 0$ for each obstacle $\mathcal{O}_i$, and are transformed along with the motion of $\mathcal{O}_i$ at $t > 0$.

We say that a time $t$ is a critical moment if the topology of $\mathcal{E}_{free}$ changes at $t$. The most fundamental geometric tool in CriticalRoadmap is to determine all critical moments $E$ in the given time interval $[0, t_f]$. These critical moments $E$ include (1) the time of collision ($toc$) and (2) the time of separation ($tos$). The critical times $toc$ and $tos$ are when a node of the local roadmaps becomes in-collision and collision free, respectively. An iterative method to predict $toc$ and $tos$ will be discussed in Section 4.3.2.

To estimate $toc$ and $tos$, our method performs continuous collision detection [92] and penetration depth estimation [93]. In order to achieve this, each geometric model in the workspace is represented by a convex hull hierarchy [94] (Fig. 4.2 in which the root is the convex hull of the entire model and the leaves are its all convex components. Each inner node is the convex hull of its children nodes. We denote the convex hull hierarchy of $\mathcal{R}$ be $CH_{\mathcal{R}}$ and the convex hull hierarchy of $\mathcal{O}_i$ be $CH_{\mathcal{O}_i}$. 46
4.2 General Overview

Algorithm 4.2.1: \texttt{CriticalRoadmap}(\mathcal{O}, \mathcal{R})

\begin{verbatim}
E ← ∅

for each \( O_i \in \mathcal{O} \) do
    \begin{align*}
    M_{O_i} &← \text{ComputeLocalRoadmap}(O_i) \\
    E_i &← \text{IdentifyCriticalMoments}(M_{O_i}) \\
    E &← E \cup E_i
    \end{align*}

\mathcal{B}_t = \text{Connect}(M_{O_i})

\textbf{while} \( E \neq ∅ \) \textbf{do}
    \begin{align*}
    e &← E\.pop() \\
    \mathcal{B}' &← \text{Update}(\mathcal{B}_t, e) \\
    \text{Connect}(\mathcal{B}_t, \mathcal{B}') \\
    \mathcal{B}_t &← \mathcal{B}'
    \end{align*}
\end{verbatim}

Algorithm 4.2.1 sketches the framework of CriticalRoadmap. There are two main stages in Algorithm 4.2.1. In the first stage, critical moments are identified using local roadmap \( M_{O_i} \) for every obstacle \( O_i \in \mathcal{O} \), which will be discussed in details in Section 4.3. The second stage updates the critical roadmaps \( \mathcal{B}_t \) for every critical moment \( t \) and connects any two consecutive critical roadmaps to approximate \( \mathcal{E}_{\text{free}} \). The subroutines that update and connect the critical roadmaps will be discussed in Section 4.4.

4.3 Continuous Collision Detection

To estimate \textit{toc} and \textit{tos}, our method performs continuous collision detection [92] and penetration depth estimation [93].
4.3.1 Preliminaries

In CriticalRoadmap, each geometric model is represented by a convex hull hierarchy [94] (Fig. 4.2) in which the root is the convex hull of an entire model and the leaves are its all convex components. Each inner node is the convex hull of its children nodes. We denote the convex hull hierarchy of $\mathcal{R}$ be $CH_\mathcal{R}$ and the convex hull hierarchy of $\mathcal{O}_i$ be $CH_i$.

**Local roadmaps.** The local roadmap $M_{\mathcal{O}_i}$ is constructed via the obstacle-based roadmap methods around the obstacle $\mathcal{O}_i$ [6, 80], and $M_{\mathcal{O}_i}$ always moves along with $\mathcal{O}_i$. Since $M_{\mathcal{O}_i}$ moves along with its associated obstacle $\mathcal{O}_i$, any $c \in M_{\mathcal{O}_i}$ also moves along with $\mathcal{O}_i$. That means $c$ can be treated as a point on $\mathcal{O}_i$. We use $\mathcal{R}_c$ to denote the robot configured at configuration $c$. In general, $\mathcal{R}_c$ undergoes two types of transformations: translation with $\mathcal{O}_i$, and rotation around $\mu_i$ where $\mu_i$ is $\mathcal{O}_i$’s center of mass. Illustrated in Fig. 4.3.1, $\mathcal{R}_c$’s angular velocities for self-rotation and rotation around $\mu_i$ are both $\vec{\omega}$. Its translational velocity consists of two parts: $\vec{v}$ (Fig. 4.3.1) and $V(t)$. $V(t)$ is introduced by
Figure 4.3: \( R_c \) and \( O_i \) are colored in shadow after transformations. (a) \( R_c \) moves along with \( O_i \) when \( O_i \) has translational velocity. (b) Since \( c \in M_{O_i}, R_c \) can be treated as a feature on \( O_i \). While \( O_i \) is rotating around \( \mu_i \), \( R_c \) is also rotating around \( \mu_i \) and its orientation is changing. If \( O_i \) has angular velocity \( \vec{\omega} \), \( R_c \)'s angular velocity for rotation around \( \mu_i \) is \( \vec{\omega} \) too. (c) \( R_c \) has the translational velocity \( V(t) \) introduced by rotating around \( \mu_i \). The instant direction of \( V(t) \) is always perpendicular to \( \vec{r} \).
Figure 4.4: The position relationships of two obstacles. (a). $M_{\partial_i}$ does not overlap $\partial_j$ and $M_{\partial_j}$ does not overlap $\partial_i$. (b). $M_{\partial_i}$ overlaps $\partial_j$ and $M_{\partial_j}$ overlaps $\partial_i$. (c). $M_{\partial_i}$ overlaps $\partial_j$ or $M_{\partial_j}$ overlaps $\partial_i$.

$R_c$, rotating around $\mu_i$ and its magnitude is constant but the direction keeps changing all the time: the direction of $V(t)$ is always perpendicular to $\vec{r} = c - \mu_i$ (Fig. 4.3.1).

Two local roadmaps $M_{\partial_i}$ and $M_{\partial_j}$ are connected via their boundary nodes $B_{i,j}$ [16]. For a configuration $c \in M_{\partial_i}$, we say that $c$ is in $B_{i,j}$ if and only if $R_c$ does not collide with $\partial_j$ but at least one of $c$’s neighbors in $M_{\partial_i}$ does.

**Critical moments.** A necessary condition for the critical time $t$ is when at least one of the following two events, i.e., $t_{oc}$ and $t_{os}$, happens at $t$.

1. $\exists c \in M_{\partial_i} \land \exists \partial_{j \neq i} \in \partial$, $R_c$ and $\partial_j$ begin to contact.

2. $\exists c \in M_{\partial_i} \land \exists \partial_{j \neq i} \in \partial$, $R_c$ and $\partial_j$ begin to separate.

Based on the collision states of the local roadmaps, we classify the position relationships of any two $\partial$-obstacles into three categories, as shown in Fig. 4.4. Finally, we know that $\partial_{free}$ changes when there are at least one pair of $\partial$-obstacles having their position relationship changed. In Section 4.3.2,
\vec{r} \quad p_j \quad \vec{r} \quad q \quad \vec{V}(t) \quad \vec{r} \quad R_c \quad \vec{r}_1 \quad O_j \quad \vec{\omega} \quad \vec{\omega'} \quad p \quad \vec{r} \quad \vec{\omega}

Figure 4.5: This figure illustrates the translational velocities of \( p \) and \( q \) which are caused by rotation. For \( p \), its translational velocity introduced by self-rotation of \( R_c \) and rotation of \( O_i \) is \( \vec{\omega} \times \vec{r} + \vec{\omega} \times \vec{r}_1 \). For \( q \), its translational velocity caused by rotation of \( O_j \) is \( \vec{\omega}' \times \vec{r}_2 \).

we will discuss how to identify \( toc \) and \( tos \) in details.

4.3.2 Identify Critical Moments

Time of Contact (\( toc \))

Given a configuration \( c \in M_{\theta_i} \) and an obstacle \( \theta_{j \neq i} \), when \( R_c \) and \( \theta_j \) do not intersect, we want to compute the time of contact \( toc \) within a time interval \([t_1, t_2] \).

We extend the iterative method [92] to estimate \( toc \) by generalizing the idea of conservative advancement [95]. In each iteration, we estimate the advancing time \( \delta_{toc} \) during which \( R_c \) will be safely moved toward \( \theta_j \) without causing any collision. The estimated advancing time \( \delta_{toc} \) is calculated based on a tight lower bound of the closest distance between \( R_c \) and \( \theta_j \) and an upper bound of the motions of \( R_c \) and \( \theta_j \). The process repeats until the distance between \( R_c \) and \( \theta_j \) is under some user-defined threshold \( \varepsilon \) or \( t_2 \) is reached.
Estimation of Advancing Time

The result from [92] is only applicable to constant velocities. To estimate $\delta_{toc}$, we extend [92] to consider models that do not have constant velocities. Let $d$ be the shortest distance between $R_c$ and $O_j$ with direction $\vec{n}$ ($\vec{n}$ is normalized). Note that $d$ is actually the distance between the closest pair of features of $R_c$ and $O_j$. Without loss of generality, assume the closest features are two points: $p$ on $R_c$ and $q$ on $O_j$ (Fig. 4.5). Let $\rho$ be an upper bound of motions of $R_c$ and $O_j$. Then calculation of $\rho$ is answered by computing the distance traveled by $p$ and $q$ along $\vec{n}$ in unit time.

Let $T_p$ be $p$’s trajectory and $T_q$ be $q$’s trajectory. Let $R_c$ and $O_j$’s translational velocity be $\vec{v}$ and $\vec{v}'$ and their angular velocity be $\vec{\omega}$ and $\vec{\omega}'$, respectively. Recall that while $R_c$ is rotating around $\mu_i$, its orientation is also changing. Consequently, each of its surface point has an additional transformation: rotation around $\mu_{R_c}$ with angular velocity $\vec{\omega}$ where $\mu_{R_c}$ is $R_c$’s center of mass. Therefore $p$’s velocity $\mathcal{T}_p(t)$ is $\vec{v} + \vec{v}' \times \vec{r}_1$. The projection of $p$’s velocity onto $\vec{n}$ is $\mathcal{T}_p(t) \cdot \vec{n}$. $q$’s velocity $\mathcal{T}_q(t)$ is $\vec{v}' + \vec{\omega}' \times \vec{r}_2$ and $q$’s velocity along $\vec{n}$ is $\mathcal{T}_p(t) \cdot \vec{n}$. Finally, the upper bound of the distance traveled by $p$ and $q$ is given by the following lemma.

**Lemma 4.3.1.** The upper bound $\rho$ of the distance traveled by $p$ and $q$ along $\vec{n}$ in unit time is

$$|\vec{n} \times \vec{\omega}| |\vec{r}| + (\vec{v} - \vec{v}') \cdot \vec{n} + (|\vec{n} \times \vec{\omega}| |\vec{r}_1| + |\vec{n} \times \vec{\omega}'||\vec{r}_2|)$$

where

- $|\vec{n} \times \vec{\omega}| |\vec{r}|$ is the upper bound of $V(t)$,
- $(\vec{v} - \vec{v}') \cdot \vec{n}$ is the distance $R_c$ travels towards $O_j$,
- $\vec{r} = \text{pos}_p - \mu_i$, where $\text{pos}_p$ stands for the position of the point $p$,
- $\vec{r}_1 = \text{pos}_p - \mu_{R_c}$ is self-rotation radius, and
- $\vec{r}_2 = \text{pos}_q - \mu_j$, where $\text{pos}_q$ stands for the position of point $q$.

These notations are illustrated in Fig. 4.5. Note that the magnitudes of $\vec{r}, \vec{r}_1, \vec{r}_2$ are constant but their directions change over time.
Proof. Let $t$ be the current time and assume at time $T$, $p$ is advanced $d$ towards $q$ at direction $\vec{n}$. Then we have

\[
d = \int_t^T (\vec{v}(t) - \vec{v}(t')) \cdot \vec{n} dt \\
= \int_t^T \left[ (\vec{v} + \vec{V}(t) + \vec{\omega} \times \vec{r}_1) - (\vec{v} + \vec{V}(t) + \vec{\omega}' \times \vec{r}_2) \right] \cdot \vec{n} dt \\
= \int_t^T \vec{V}(t) \cdot \vec{n} dt + \int_t^T (\vec{v} - \vec{v}') \cdot \vec{n} dt \\
+ \int_t^T (\vec{\omega} \times \vec{r}_1 - \vec{\omega}' \times \vec{r}_2) \cdot \vec{n} dt
\]

(4.1)

The advancing time $\delta_{toc}$ should be $(T - t)$. Since $\vec{v}$, $\vec{v}'$ and $\vec{n}$ are constant vectors, we have

\[
d = \int_t^T \vec{V}(t) \cdot \vec{n} dt + (\vec{v} - \vec{v}') \cdot \vec{n} \delta_{toc} + \int_t^T (\vec{\omega} \times \vec{r}_1 - \vec{\omega}' \times \vec{r}_2) \cdot \vec{n} dt
\]

Because

1. $(\vec{\omega} \times \vec{r}) \cdot \vec{n} = (\vec{n} \times \vec{\omega}) \cdot \vec{r}$.

2. $\vec{\omega}$ and $\vec{\omega}'$ are constants during the time interval $[t_1, t_2]$.

3. The magnitudes of $\vec{r}_1$ and $\vec{r}_2$ are constants.

we have

\[
d \leq \int_t^T \vec{V}(t) \cdot \vec{n} dt + (\vec{v} - \vec{v}') \cdot \vec{n} \delta_{toc} + (|\vec{n} \times \vec{\omega}|\vec{r}_1 + |\vec{n} \times \vec{\omega}'|\vec{r}_2) \delta_{toc}
\]
Assume the square is fixed while the tube is moving with velocity $\vec{v}$. The magnitude of an arrow shows the penetration depth and its direction is the corresponding penetration direction. $\delta_1$ and $\delta_2$ are the estimations of $tos$ at the first and second iteration, respectively.

Shown in Fig. 4.5, $V(t) = \vec{\omega} \times \vec{r}$, so

$$\int_t^T V(t) \cdot \vec{n} dt \leq \int_t^T (\vec{\omega} \times \vec{r}) \cdot \vec{n} dt$$

$$= \int_t^T (\vec{n} \times \vec{\omega}) \cdot \vec{r} dt$$

$$\leq |\vec{n} \times \vec{\omega}| ||\vec{r}|| \delta_{toc}$$

Therefore,

$$d \leq \delta_{toc} \{|\vec{n} \times \vec{\omega}| ||\vec{r}|| + (\vec{v} - \vec{\nu}) \cdot \vec{n} + (|\vec{n} \times \vec{\omega}| ||\vec{r}_1|| + |\vec{n} \times \vec{\omega}'| ||\vec{r}_2||)\}$$

The upper bound of motions is

$$\rho = |\vec{n} \times \vec{\omega}| ||\vec{r}|| + (\vec{v} - \vec{\nu}) \cdot \vec{n} + (|\vec{n} \times \vec{\omega}| ||\vec{r}_1|| + |\vec{n} \times \vec{\omega}'| ||\vec{r}_2||).$$

So the lower bound of $\delta_{toc}$ is $\frac{d}{\rho}$. \hfill \square

**Time of Separation ($tos$)**

When $R_c$ and $O_j$ intersect, we apply the similar idea to estimate $tos$. At each iteration, we first compute the penetration depth $pd$ between $R_c$ and $O_j$. Then advancing time $\delta_{tos}$ is estimated based on $pd$ and motions of $R_c$ and $O_j$. Let $\vec{n}$ be the direction that realizes this $pd$ and $\rho$ be the upper bound of the velocity along $\vec{n}$. Since we know that $pd$ is the shortest distance to separate $R_c$ and $O_j$,
there is a simple relationship between $\delta_{tos}$ and $pd$:

$$\rho \times \delta_{tos} \leq pd.$$ 

Therefore, we can ensure that $R_c$ and $O_j$ will remain in collision after advancing them by the estimated time $\delta_{tos}$. This continues until they are still intersecting but the penetration depth is under some very small user-defined tolerance $\tau$ or $t_2$ is reached. Fig. 4.6 shows a simple example.

We apply the idea in Section 4.3.2 to compute the upper bound $\rho$ and use DEEP [93] to compute penetration depth between two convex polyhedra. Instead of checking all pairs of convex pieces from $R_c$ and $O_j$, one more efficient way is to use a convex hull traversal tree (CHTT) [96]. CHTT is built while traversing $CH_{R}$ and $CH_{J}$. Shown in Fig. 4.3.1, the traversal starts with the root nodes of $CH_{R}$ and $CH_{J}$ and it is performed recursively on both trees simultaneously. Each node in CHTT corresponds to an intersection test on a single pair of convex pieces. The root node of CHTT is the intersection test on the roots of $CH_{R}$ and $CH_{J}$ and its leaf nodes are either the intersection test on the leaf nodes of $CH_{R}$ and $CH_{J}$ or a pair of convex pieces which do not intersect. For the intersection test between some pair of nodes in $CH_{R}$ and $CH_{J}$, if they intersect, the penetration depth is computed. Then collision detection will be performed on their child nodes. If they do not overlap, their children are guaranteed to be collision free.

**Implementation Details**

For the computation of $toc$, conservative advancement continues until the distance $d$ between $R_c$ and $O_j$ is smaller than a user-defined threshold $\varepsilon$. Because of the underestimation of advancing time in each iteration, the two models are not guaranteed to be in contact and there might be always some very small distance between them. To make the two models collide at time $toc$, we increase $d$ by $\varepsilon$ (we use $\varepsilon = 0.001$) in the last iteration and estimate the advancing time with this new $d$. Adding a small value to $d$ makes sure that two models are in contact at $toc$ and the reported $toc$ is still very close to its true value. Similarly, for time of separation, $\delta_{tos}$ is estimated with $\rho \times \delta_{tos} \leq (pd + \varepsilon)$ in
the last iteration.

4.4 Update and Connect Critical Roadmaps

4.4.1 Update Roadmap at Each Critical Moment

A critical moment implies potential changes of the topology of $\mathcal{G}^{\text{free}}$. So for any given critical moment $t$, the critical roadmap $\mathcal{G}_t$ needs to be updated to reflect these changes. All local roadmaps are transformed based on obstacles’ motions, and then the local roadmaps are merged into $\mathcal{G}_t$ by adding connections between the boundary nodes of pairs of local roadmaps [16].

More specifically, for each configuration $c \in M_{\mathcal{O}_i}$ and an obstacle $\mathcal{O}_j \neq i \in \mathcal{O}$, we maintain a list of critical times $T_{\{c, \mathcal{O}_j\}}$. To find the boundary nodes $B_{i,j}$ of $M_{\mathcal{O}_i}$, we determine a set of configurations $CD_{i,j}$ from the local roadmap $M_{\mathcal{O}_i}$ that will make $R$ collide with $\mathcal{O}_j$. Because for any time $t$, if $t$ is between a $toc$ and a $tos$ in $T_{\{c, \mathcal{O}_j\}}$, $R_c$ must intersect $\mathcal{O}_j$ at $t$. Therefore, $CD_{i,j}$ be easily collected for any critical time without additional collision detection tests. If $CD_{i,j}$ and $CD_{j,i}$ are nonempty, $M_{\mathcal{O}_i}$ and $M_{\mathcal{O}_j}$ are merged by adding connections between their boundary nodes $B_{i,j}$ and $B_{j,i}$.

4.4.2 Connect Critical Roadmaps

Given two consecutive critical times, we need to connect their critical roadmaps. The following observation allows us to make the connections efficiently.

**Observation** Consider two consecutive critical times $t$ and $t'$ with $t < t'$. For a valid edge $c_1c_2$ in critical roadmap $\mathcal{G}_t$, let $c'_1$ be the transformed $c_1$ at $t'$ and $c'_2$ be the transformed $c_2$ at $t'$. All the connections $c_1c'_1$, $c_1c'_2$, $c_2c'_1$ and $c_2c'_2$ are guaranteed to be collision free if $c'_1c'_2$ is also valid.

**Proof.** Since $t$ and $t'$ are two consecutive critical times, there are no critical moments in between for any configuration including $c_1$ and $c_2$. So during the time interval $[t, t']$, both configurations keep collision free. Therefore, all the four trajectories $c_1$ to $c'_1$, $c_1$ to $c'_2$, $c_2$ to $c'_1$ and $c_2$ to $c'_2$ are valid.
Figure 4.7: Connect two critical roadmaps in a 3D configuration-time space. $t_2$ is a time of separation because the triangle becomes separated from the rectangle’s local roadmap at $t_2$. For the two valid edges $c_1c_2$ and $c'_1c'_2$, connections $c_1c'_1$, $c_1c'_2$, $c_2c'_1$ and $c_2c'_2$ are valid. Their intersection with the configuration space at $t_1$ are $a$, $b$, $c$ and $d$ which are all collision free.
Figure 4.8: (a) Local planner for configurations from the same local roadmap at different moments. The black curve is the trajectory of obstacle $O_i$ and the blue curve is the connection between $c$ and $c'$. (b) Local planner for configurations from local roadmaps of different obstacles. The obstacles are two moving polygons. The square is rotating counterclockwise while the triangle is rotating clockwise. Both of them are translating from left to right. A point robot travels from the bottom-left corner to top-right corner.

An example is illustrated in Fig. 4.7.

Note that due to the motion of the local roadmap, the connection between two nodes, e.g., $c_1$ and $c'_2$, might not be a straight line. Consider a configuration $c$ from $M_{O_i}$ at time $t$. Let $c'$ be its new position at $t'$ after transformation. Recall that as the obstacle $O_i$ moves, $c$ moves along with $O_i$. Therefore, the connection between $c$ and $c'$ in $[t,t']$ is exactly the same as $O_i$’s trajectory over this time interval (Fig. 4.4.2).

Let us now consider a more general example as shown in Fig. 4.4.2. Let $c_i(t)$ be a node of $M_{O_i}$ at time $t$ and $c_j(t')$ be a node of $M_{O_j}$ at time $t'$. The connection (denoted as $T$) between $c_i(t)$ and $c_j(t')$ over the time interval $[t,t']$ is interpolated as follows.

First, we know that $T(t) = c_i(t)$ and $T(t') = c_j(t')$. For any other time $\tau$, $t < \tau < t'$, we want to compute $T(\tau)$. Using linear interpolation, $T(\tau) = (1 - s) \times c_i(t) + s \times c_j(t + 1)$, where $s = \frac{t - t'}{t - t'}$. 

58
Figure 4.9: Animation sequences for a rod-shaped robot moving in an environment with several boards. The robot needs to pass through several narrow holes in order to reach the goal. A board moves forward while rotating around its moving direction.

4.5 Experimental Results

Experiment Set Up. All the experiments are performed on an Intel Core i7 M620 CPU at 2.67GHz with 4GB RAM. The implementation is coded in C++. Our new method is tested on three environments: Hole (Fig. 4.9), Ball (Fig. 4.10) and Table (Fig. 4.11). They share the same robot which is a long and thin rod. The maximum time limits $t_f$ are 5.0, 10.0 and 2.0 time units, respectively. Each environment is assigned 30 queries. Although these queries are generated with some randomness, we try to make each as hard as possible. Take Hole as an example. For each query, the rod robot has to pass through at least one hole to reach its destination. The environments Ball (Fig. 4.9) and Table (Fig. 4.11) are designed to mimic scenarios we could encounter in everyday life. They are usually easy and do not contain very narrow passages. The environment Hole (Fig. 4.9) could find its applications in a disaster zone such as the scene of fire and the rescue robot has to pass through several narrow passages (e.g. doors) to perform tasks. Moreover, these environments are challenging enough for classic PRM s including Uniform [1] and Gaussian [6]. Table 4.1 shows the running times of our method in different phases. All running times are measured in seconds. The numbers of
Figure 4.10: Animation sequences for a rod-shaped robot moving in an environment with several balls. A ball moves in an ellipse-shaped trajectories while rotating around its own center of mass.

Figure 4.11: Animation sequences for a rod-shaped robot moving in an environment with two big tables and a ball. A table translates while rotating around its vertical axis.
Figure 4.12: Experimental results for the environments in Fig. 4.9, Fig. 4.10 and Fig. 4.11. (a) The average running time over 300 runs. The $y$-axis is in logarithmic scale. (b) The success rate of each planning method over 300 runs.

Table 4.1: Running times (sec) for every phase of our method

<table>
<thead>
<tr>
<th>Phase</th>
<th>Hole</th>
<th>Ball</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Local Roadmaps</td>
<td>3.182</td>
<td>2.574</td>
<td>2.449</td>
</tr>
<tr>
<td>Detecting Critical Moments</td>
<td>4.399</td>
<td>0.375</td>
<td>4.758</td>
</tr>
<tr>
<td>Update Global Roadmaps</td>
<td>6.022</td>
<td>0.484</td>
<td>0.031</td>
</tr>
<tr>
<td>30 Queries</td>
<td>40.185</td>
<td>7.129</td>
<td>1.669</td>
</tr>
<tr>
<td>Total</td>
<td>53.835</td>
<td>10.592</td>
<td>8.939</td>
</tr>
</tbody>
</table>

critical times for the Hole, Ball and Table environments are 63, 16, and 7, respectively.

**Comparison to Previous Work.** The average running time and success rate for each method are plotted in Fig. 4.12. An success rate is the number of successful queries divided by the total number of queries (which is $30 \times 10$ in our experiments). CriticalRoadmap can find a valid path for every query. In other words, the success rate of CriticalRoadmap is kept at 100%.

The new method is first compared to several classic planners: Uniform [1], Gaussian [6] and RRT [9]. Shown in Fig. 4.12, our method has significant efficiency improvements: 2~3 orders of magnitude faster over Uniform and Gaussian and noticeably faster than RRT. Moreover, the success rate of our method is kept at 100%. Note that RRT is a single-shot planner and it generates a new tree for each query, so we expect its performance to degrade when more queries are given. Based on the running
times and success rates plotted in Fig. 4.12, it is easy to tell the efficiency improvement of our new method over RRT is more obvious in more difficult environments (e.g. Hole in Fig. 4.9 and Ball in Fig. 4.10).

For the method which updates a critical roadmap at fixed time intervals, we set the intervals to 0.02, 0.1 and 0.00001 for the Hole, Ball and Table environments [20], respectively. The running times for these intervals are plotted in Fig. 4.12. Although each time interval is very small, it cannot guarantee to find a valid path for each query. For Hole, 5 out of 30 queries cannot be solved and for Table, there is a chance of 18/30 that a query could not be solved. This is because it misses some critical moments when the topology of free configuration space changes. Some connections between two consecutive roadmaps are detected invalid at the query stage. Moreover, it may update a roadmap at a moment which does not involve the changes of $C_{\text{free}}$. Our method can avoid these unnecessary updates and is at least one order of magnitude faster.

To prove that an average running time can represent how efficiently a planning method performs in a single run, we report standard deviation of running times for all path planners. The following tables show the ratio of standard deviation over average running time for environments Fig. 4.9, Fig. 4.10 and Fig. 4.11, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Ratio of standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform PRM</td>
<td>0.113</td>
</tr>
<tr>
<td>Gaussian PRM</td>
<td>0.078</td>
</tr>
<tr>
<td>RRT</td>
<td>0.039</td>
</tr>
<tr>
<td>FIXED</td>
<td>0.008</td>
</tr>
<tr>
<td>CriticalRoadmap</td>
<td>0.036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Ratio of standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform PRM</td>
<td>0.099</td>
</tr>
<tr>
<td>Gaussian PRM</td>
<td>0.093</td>
</tr>
<tr>
<td>RRT</td>
<td>0.02</td>
</tr>
<tr>
<td>FIXED</td>
<td>0.008</td>
</tr>
<tr>
<td>CriticalRoadmap</td>
<td>0.055</td>
</tr>
</tbody>
</table>
### 4.6 Discussion

In this chapter, we discussed a method called “CriticalRoadmap” for planning motion in dynamic but fully known environments. Our method can detect all the critical times when the free configuration space changes. Therefore, it provides a more complete representation of free configuration-time space compared to the existing methods that only use fixed time resolution. We also use an efficient way to assemble the roadmap at each critical time by reusing the most of the existing roadmap and repair only the invalid part caused by obstacles’ motions. Compared to the previous methods, our strategy is more complete and provides significant efficiency improvements.

While using local roadmaps provides a more efficient way of updating roadmaps for each critical time, it requires the robot to be free flying. Therefore, the new method does not work for a robot with a fixed base.
Chapter 5: Path Planning in Unknown Environments

In many real life applications such as rescue robots and autonomous vehicles, the robot may have no prior knowledge of obstacles’ motions or even shapes and the pose of an obstacle can only estimated through on-board sensors. In this chapter, I will discuss a framework for designing a set of motion planners for such environments with the following assumptions.

1. No constraints on the shape of the robot or an obstacle, but all shapes are assumed to be known.
2. The initial state of the workspace is known to the robot.
3. The robot does not know obstacles’ motions except for their maximum velocities including translational velocity and rotational velocity. The only way for the robot to know the environment is through its on-board sensors.

Due to partial knowledge of the environment, computation from the preprocessing stage may expire. Moreover, due to the limited range of a sensor, the robot can only sense the environment in its proximity and predict danger that is imminent. Therefore, finite computation time is required to react to approaching collisions. Based on these facts, the two main issues in path planning which are safety and efficiency become very challenging.

So far as we know, all existing work handling obstacles with unknown trajectories update its belief of the environment through sensors at fixed times [21, 22]. Consequently, update may be redundant or the situation is even worse when update is performed not frequently enough. The robot might be already in collision before next update or at a state which causes it to collide with some obstacle soon and it is too late for the robot to avoid collisions. Ideally, the repair interval should be determined adaptively based on the motion of the obstacles. We would like to have a planner that can adaptively update the roadmap at “critical moments” at which the topology of the free configuration space changes.
Motivated by this observation, I designed a new method to explore motion coherence of obstacles and predict a critical time until when the robot is guaranteed to move safely along its current path. And only at such critical moments, the robot updates its belief of the environment based on sensory data and re-plans if necessary.

5.1 Preliminaries

In this section, we define important notations that will be used through the chapter. For the path planner designed for unknown dynamic environments, we assume the robot has no information of how an obstacle moves except for its maximum velocities. For an obstacle \( \mathcal{O}_i \in \mathcal{O} \), we use \( v_i \) and \( \omega_i \) to denote the magnitudes of its maximum translational velocity and rotational velocity, respectively. Note that for an obstacle, it can rotate around any point specified by the user and let \( o \) be this center of rotation. We use \( \text{SA}_t \) to denote the region swept over by obstacle \( \mathcal{O}_i \) for a period of time \( t \). We assume the robot moves along some designated path \( \Pi \) which is composed of a sequence of configurations \( \Pi = \{c_1, c_2, \ldots, c_n\} \) with \( c_1 = S \) and \( c_n = G \). Then the \( j \)-th path segment on \( \Pi \) is \( c_j c_{j+1} \). One of the most important tasks in the new path planner is to estimate the earliest time when \( \mathcal{R} \) could hit some obstacle. Since the designated path is a polygonal line, the problem can be greatly simplified if we focus on discussing how to estimate the earliest collision time between \( \mathcal{R} \) and a single obstacle when \( \mathcal{R} \) is on some path segment. We use \( \text{ECT}_{ij} \) to denote the earliest collision time when \( \mathcal{O}_i \) can hit \( \mathcal{R} \) when it is on \( c_j c_{j+1} \).
5.2 Framework Overview

Algorithm 5.2.1: PLANNERUNKNOWN(Ω, 𝒫, 𝒢, Δt, N)

\[\Pi \leftarrow \text{InitialPlanning}(\Omega, \mathcal{R}, \mathcal{I}, \mathcal{G})\] (1)

*comment:* \(c\) always indicates \(\mathcal{R}\)'s current configuration

\[c \leftarrow \mathcal{S}\] (2)

*comment:* \(N\) is user-defined maximum number of iterations

\[
\text{while } \mathcal{R} \text{ not at } \mathcal{I} \text{ or } N \text{ not reached} \quad (3)
\]

\[
\begin{cases}
  t \leftarrow \text{CollisionPrediction}(\Omega, \mathcal{R}, \Pi, c) & \text{(4)} \\
  \text{Update}(\Omega, t - \Delta t) & \text{(5)} \\
  \text{if } t \leq \Delta t + \varepsilon \quad \text{do} & \text{(6)} \\
  \quad \Pi \leftarrow \text{Replanning}(\Omega, \mathcal{R}, c, \mathcal{G}) \\
  \quad \text{else } c \leftarrow \text{Move}(\mathcal{R}, t - \Delta t, \Pi) & \text{(7)}
\end{cases}
\]

Algorithm 5.2.1 describes the basic framework of our new path planner for unknown environments. Based on the initial state, it first constructs a valid path \(\Pi\) from \(\mathcal{I}\) to \(\mathcal{G}\) or as near as possible to \(\mathcal{G}\), if \(\mathcal{G}\) is not reachable initially (statement 1). Then a new geometric tool CollisionPrediction (statement 4) is applied to conservatively determine the critical moment \(t\) when the robot and obstacles can collide while the robot is moving along the designated path \(\Pi\). The robot budgets a certain amount of time \(\Delta t\) before the critical moment \(t\) to update its environmental belief based on on-board sensors (statement 5). If \(t \leq \Delta t + \varepsilon\) where \(\varepsilon\) is a small positive value, that means collisions are possibly imminent and it is not safe to take the current path \(\Pi\). Therefore, a new path starting from current configuration \(c\) needs to be constructed (statement 6). If the predicted collision moment is not approaching \((t > \Delta t + \varepsilon)\), there is no need to replan and current path \(\Pi\) will be still safe to traverse for some time. The robot will move along \(\Pi\) for at least \((t - \Delta t)\) without worrying about hitting any obstacles (statement 7). This process repeats until the robot arrives at goal or maximum number of iterations is reached.
The main challenge in predicting collision stems from the assumption that obstacle’s motion is unknown. To provide conservative estimation, the basic framework introduced in this chapter models the obstacles as adversarial agents which will minimize the time that the robot remains collision free. Consequently, a robot can actively determine its next replanning time by conservatively estimating the amount of time (i.e., earliest collision time) that it can stay on the planned path without colliding with the obstacles. The idea of earliest collision time and conservative advancement are detailed in Section 5.3.

Overall, our new tool advances collision prediction beyond the translational and disc robots [51, 67, 73–75]. Arbitrary (even non-simple) polygons with rotation can be used to better represent obstacles and provide tighter bound on predicted collision time. This prediction is determined only based on the last known positions of the obstacles and their maximum linear and angular velocities. In our experimental results (Section 5.5), we demonstrate that our method significantly reduces the number of replannings while maintaining higher success rate of finding a valid path.

Based on the discussion above, the most fundamental tool in this new path planner is this new geometric tool collision prediction. I will talk about this in details in the next chapter.

5.3 Conservative Advancement

Planning a path in environments populated with obstacles with unknown trajectories usually involves two steps: (1) find an initial path $\Pi$ based on known information and then (2) modify $\Pi$ as the robot receives new information from its on-board sensors at fixed times. In our work, instead of determining if $\Pi$ is still safe to traverse at fixed times, $R$ determines the critical moment $t$ that $\Pi$ may become invalid. The robot budgets a certain amount of time $\Delta t$ before this critical moment $t$ to update its belief and replan if necessary. To make our discussion more concrete, let us emphasize again that this setting is merely a framework among other applications of collision prediction.

Because the trajectory of the obstacles in workspace is unknown, the critical moment $t$ can only be
approximated. To ensure the safety of the robot, our goal is to obtain conservative estimation \( t' \leq t \) of the unknown value \( t \). Follow the naming tradition in collision detection, we call such an estimation \emph{conservative advancement} on \( \Pi \) and denote it as \( CA_\Pi \). To compute \( CA_\Pi \), the robot assumes that all obstacles are adversarial. That is, these adversarial obstacles will move in order to minimize the time that \( \Pi \) remains valid.

Contrary to traditional motion planning methods, the calculation of \( CA_\Pi \) (performed by the robot) in some sense reverses the roles of robot and obstacles. The robot \( R \) is now fixed to the path \( \Pi \), thus the configuration of \( R \) at any given time is known. On the other hand, the obstacles’ trajectories are unknown but will be planned to collide with \( R \) in the shortest possible time. The motion strategy for an obstacle \( O_i \) will only depend on the the maximum translational velocity \( v_i \) and a maximum angular velocity \( \omega_i \) around a user defined rotation center \( o \). Different obstacles can have different centers for rotation.

\section*{5.3.1 Estimate Conservative Advancement on Path \( \Pi \)}

Without loss of generality, the problem of estimating \( CA_\Pi \) can be greatly simplified if we focus on only a single obstacle and a segment of path \( \Pi \). Let \( \Pi \) be a sequence of free configurations \( \Pi = \{c_1, c_2, \ldots, c_n\} \) with \( c_1 = S \) and \( c_n = G \), where the \( S \) and \( G \) are start and goal configurations, respectively.

Given a segment \( \overline{c_j c_{j+1}} \subset \Pi \), we let \( ECT_{i,j} \) be the earliest collision time (ECT) that \( O_i \) takes to collide with the robot on \( \overline{c_j c_{j+1}} \). Then we have \( CA_\Pi = \min_i \left( \min_j \left( ECT_{i,j} \right) \right) \), where \( 1 \leq i \leq |O| \) and \( 1 \leq j < n \). Note that \( ECT_{i,j} \) is infinitely large, if \( O_i \) is guaranteed not to collide with \( R \) before \( R \) leaves \( \overline{c_j c_{j+1}} \).

\textbf{Lemma 5.3.1.} If \( ECT_{i,j} \neq \infty \), then \( ECT_{i,j} \leq ECT_{i,k}, \forall k > j \)

That is, once an earliest collision time is detected for a path segment \( \overline{c_j c_{j+1}} \), it is not necessary to check all its subsequent segments \( \overline{c_k c_{k+1}} \) with \( j < k < n \). In Section 5.3.3, we will provide a brief overview on how \( ECT_{i,j} \) can be computed.
Before we proceed our discussion, we would like to point out that our method does not consider collisions between the obstacles. Although this makes our estimate more conservative, the obstacle with the earliest collision time rarely collides with other obstacles.

5.3.2 Pre-processing

For a path segment $c_1c_2 \in \Pi$, let $t_1$ be the time when robot reaches $c_1$ and $t_2$ be the time when robot reaches $c_2$. We first perform a preprocessing step to filter out obstacles which are impossible to hit the robot on $c_1c_2$ between the time period $[t_1, t_2]$.

![Figure 5.1: An “envelope” of path segment $c_1c_2$ induced by $O_i$. It is bounded by two circles which are centered at $c_1$ and $c_2$, respectively and two lines $ab$ and $cd$ which are tangent to these two circles.](image)

In order to achieve this, we first build an “envelope” of $c_1c_2$ with respect to every dynamic obstacle $O_i \in \mathcal{O}$. Shown in Fig. 5.1, such an envelope is bounded by two circles and two lines which are tangent to these circles. The radius of the circle centered at $c_1$ is $v \times t_1$ with $v$ being $O_i$’s maximum translational velocity. The radius of the circle centered at $c_2$ is $v \times t_2$. A point $p$ is on the boundary of this “envelope” if its closest distance to $c_1c_2$ is $v \times t$ with $t$ being the time when robot reaches $p$’s closest point on $c_1c_2$. Let $SA$ be the area $O_i$ sweeps over during $[t_1, t_2]$. In order to hit the robot on $c_1c_2$, $SA$ needs to intersect the envelope. In this way, we can filter out any obstacle whose swept region is separated from the envelope. For an obstacle, the further it is from $c_1c_2$, the more possible it will be filtered out. Since it is nontrivial to compute swept area for an arbitrary shape with rotation, we can use a simple shape such as its oriented bounding box to approximate the original shape.
5.3.3 Earliest Collision Time (ECT)

Given a segment $c_j c_{j+1} \subset \Pi$ of path in C-space, our goal is to compute the earliest collision time $ECT_{i,j}$ when obstacle $O_i$ hits robot $R$ somewhere on $c_j c_{j+1}$. Without loss of generality, assume $R$ starts to execute on $\Pi$ at time 0.

Since the robot $R$ moves along a known path $\Pi$, $R$ knows when it reaches any configuration $c \in \Pi$. Let $t$ be the time that $R$ takes to reach a configuration $c(t) \in c_j c_{j+1}$ and let $T$ be the time when $O_i$ reaches this $c(t)$. Because $O_i$ is constrained by its maximum linear and angular velocities $v_i$ and $\omega_i$, there must exist an earliest time $\hat{T}$ for $O_i$ to reach any $c \in c_1 c_2$ without violating these constraints.

Since every configuration on $c_j c_{j+1}$ is parameterized by $t$, this $\hat{T}$ can also be expressed as a function of $t$. Let this function be $f(t)$. Furthermore, when the robot $R$ and $O_i$ collide, they must be able to reach a configuration $c$ at the same time. Therefore, we also consider the relationship between $t$ and $T$ modeled by the function $g(t) : t = T$.

In both figures in Fig. 5.2, a bold (red) curve represents $f(t)$ and a black straight line represents $g(t)$. These two curves subdivide the space into interesting regions.
• For a point \( p = (t, T > t) \), indicates situations that \( O_i \) reaches \( c(t) \) later than \( t \). No collisions will happen because when \( O_i \) reaches \( c(t) \), the robot \( R \) already passes \( c(t) \).

• The points \( p = (t, T < f(t)) \) indicates impossible situations that \( O_i \) needs to move faster than its maximum velocities in order to reach \( c(t) \) at \( T \).

• For a point \( p = (t, f(t) < T < t) \) from the region above curve \( f(t) \) but below curve \( t = T \), \( O_i \) has the ability to reach \( c(t) \) earlier than \( R \). In order to collide with \( R \), \( O_i \) can slow down or wait at \( c(t) \) until \( R \) arrives. We call this region the collision region.

Given that the robot \( R \) enters the path segment \( \overline{c_j c_{j+1}} \) through one end point \( c_j \) at time \( t_j \) and leaves \( \overline{c_j c_{j+1}} \) from the other endpoint \( c_{j+1} \) at time \( t_{j+1} \), the earliest collision time \( ECT_{ij} \) is the \( t \) coordinate of left most point of the collision region between \( t_j \) and \( t_{j+1} \) (Fig. 5.2 (a)). Therefore if this collision region is empty, \( R \) and \( O_i \) will not collide on \( \overline{c_j c_{j+1}} \) (Fig. 5.2 (b)).

Based on what has been discussed so far, the most important step of estimating critical moment is to compute \( f(t) \), the earliest moment when \( O_i \) reaches \( c(t) \). The shape of function \( f(t) \) depends on the type and the degrees of freedom of the robot and obstacles.

In the next subsection, we will discuss how \( f(t) \) can be formulated when both \( R \) and \( O_i \) are points due to its easiness. In the following chapters, we are going to talk about how to formulate \( f(t) \) for the following cases.

• \( R \) is a point and \( O_i \) is a polygon,

• \( R \) is a point and \( O_i \) is an articulated polychain,

• or both \( R \) and \( O_i \) are polygons.

Notice that from these examples, we can build up \( f(t) \) for complex polyhedra using the \( f(t) \) of points and line segments even when rotation is considered.
Figure 5.3: \( O_i \) is currently placed at \( p \) and \( \mathcal{R} \) is currently placed at \( c \). When both \( O_i \) and \( \mathcal{R} \) are points, their closest distance can be computed using Law of cosines in \( \triangle pcjc \).

5.3.4 Point-Point Case

To warm up our discussion, we start with a point robot \( \mathcal{R} \) and a point obstacle \( O_i \) without rotation. Let obstacle \( O_i \)'s current pose \( p \) coincide with its reference point \( o \) and \( c(t) \) is the pose of the robot at time \( t \). The function \( f(t) \) can be simply defined as

\[
f(t) = \frac{|pc(t)|}{v_i}.
\] (5.1)

Since \( \mathcal{R} \) moves with a given velocity, \( c_jc_{j+1} \subset \Pi \) can be linearly interpolated and every point on \( c_jc_{j+1} \) is parameterized by \( 0 \leq \lambda \leq 1 \). So the distance \( L \) between \( c_j \) and \( c(t) \) is \( L = |cjc(t)| = \lambda |c_{j+1}c_j| \) and, the function \( f \) can be simply written as:

\[
f(t) = \sqrt{L^2 + d^2 - 2dL \cos \theta} / v_i
\] (5.2)

where \( d = |pc_j| \) and \( \theta \) is the angle \( \angle pc_jc_{j+1} \). This is illustrated in Fig. 5.3.

In order to compute the collision region, we need to find out the intersections of functions \( f(t) \) and \( g(t) = t = T \). By replacing \( f(t) \) with \( t \), we get a quadratic equation with only one variable \( t = \sqrt{L^2 + d^2 - 2dL \cos \theta} / v_i \). By solving this equation, we can determine the collision region between \( [t_j, t_{j+1}] \) based on the solutions. If the collision region is empty, there will be no collision between \( O_i \) and \( \mathcal{R} \) on path segment \( c_jc_{j+1} \) (Fig. 5.2 (b)).
The collision prediction tools for other types of robot/obstacles will be discussed in the following chapters.

5.4 Planning Motion Using Predicted Collision

So far we assume that the robot only stays on a given path. In this section, we show how to use the predicted collision when replanning becomes necessary.

![Figure 5.4: An RRT augmented with earliest collision time. The tree is rooted at current configuration \( r \) of the robot. Configurations \( c' \) and \( d' \) are the predicted earliest collision locations on the paths from \( r \) to \( c \) and \( d \), respectively.](image)

When collisions are approaching, the robot budgets a certain amount of time \( \Delta t \) before the predicted critical moment \( t \) to update its belief of environment. In other words, at time \((t - \Delta t)\), the robot collects information about environment from on-board sensors and then updates the position and orientation of every obstacle. Based on the latest state of the environment, it applies our collision prediction strategy to see if the current path is still safe to traverse for a decent amount of time in the future. If the answer is yes, the robot continues to move along the current path. If not, that means the robot is in danger and it has to take measurements to avoid approaching collisions. To achieve this, we make the robot to replan a new path from current position all the way to goal, if possible.

There are two desirable properties when a robot replans a path. First, we want a path to bring the robot near the goal. If the goal or the free space in its proximity is covered by some dynamic obstacle and finding a collision free path from current position all the way to goal is very difficult, we still
want the extracted path close to the goal as much as possible. In this case, we choose a node which remains safe for the longest period among all nodes in the goal’s proximity and extract a path from current position to this node. Second, we prefer the path to remain safe for as long as possible. With these two properties in mind, we propose to augment RRT \cite{9} with predicted collision. More specifically, the RRT is constructed as usual but each path from the root to a leaf is now associated with an earliest collision time (ECT). The best path is then a path in the RRT that has the latest ECT while still reduces the geodesic distance between the robot and the goal. An example of an augment RRT is shown in Fig. 5.4. In this example, paths from configuration $r$ to all leaves reduce the distance to the goal but the path $\pi_d$ to configuration $d$ has the latest ECT, thus $\pi_d$ is the best path.

5.4.1 Construction of Augmented RRT

The augmented RRT is created as follows. The tree is rooted at the robot’s current position. To add a new node into the tree, a configuration $q_{\text{random}}$ is randomly sampled inside the workspace. Then the closest node in the tree to $q_{\text{random}}$ is computed, denoted as $q_{\text{close}}$. We try to expand the tree from $q_{\text{close}}$ to $q_{\text{random}}$ for some user-defined distance $d$. A new node $q_{\text{new}}$ and a new edge from $q_{\text{close}}$ to $q_{\text{new}}$ is added into the tree. In addition to a regular RRT, we augment each node with two time tags. Since we know the robot’s velocity and the time when it is at the root, we can compute when the robot can reach a specific node $n$ in the tree. This is the first time tag which is the moment when robot arrives at $n$. We also predict the time when the earliest collision could happen, if the robot takes the path from root to $n$. This the second time tag of $n$ at which the robot can safely traverse along the path from root to $n$. If no collisions are detected on the path from root to $n$, the two time tags of $n$ are the same. Notice that, for a node $n$, the robot may run into collision before reaching $n$. However, we may still expand the tree from this node since a path that navigates the robot to the goal is desired. Otherwise, the robot may be trapped into some dead corner and it is too difficult for it to get out. In this case, if collisions are detected at time $t'$ on some edge $\overrightarrow{n_i n_{i+1}}$, then the second time tag of node $n_{i+1}$ would be $t'$ and for any of all its subsequent nodes on the path to goal, the second time tag is also $n'$. An RRT augmented with collisions times is shown in Fig. 5.4.
5.5 Experimental Results

5.5.1 Experiment Setup

We implemented the collision prediction method in C++ with the help of Eigen linear algebra library and NLopt library. Experimental results reported in this paper are obtained from a workstation with two Intel Xeon E5-2630 2.30GHz CPUs and 32GB memory.

We tested our implementation in twenty-one environments shown in Fig. 5.5, Fig. 5.10 and Fig. 5.13. These environments contain both static and dynamic obstacles. For a dynamic obstacle, its motion is simulated using Box2D physics engine by exerting random forces. The robot knows the locations of static obstacles and the maximum translational velocity and angular velocity of a dynamic obstacle. The only way that the robot knows the pose of a dynamic obstacle is through its (simulated) on-board sensors. Every environment in Fig. 5.5 contains a point robot. Every environment in Fig. 5.10 contains dynamic articulated obstacles. Every environment in Fig. 5.13 contains a polygonal robot. Each environment is designed to demonstrate certain features. For example, Fig. 5.5 (c) has a complicated bird shape. Fig. 5.5 (d) has many (16) dynamic cross shapes. Fig. 5.5 (g) has bars with large angular velocities. Fig. 5.5 (f) and Fig. 5.5 (i) contains narrow passages. Fig. 5.5 (j) has long bars with large angular velocities and static bars surrounding start and goal. The best way to visualize the environments is via animation. We encourage the reader to view the videos at http://masc.cs.gmu.edu/wiki/ECT.

Fixed-Time Strategy In our experiments, we compare two planning strategies: one replans adaptively based on collision prediction using augmented RRT (see Section 5.4), and the other replans periodically at fixed time interval using regular RRT. We would like to emphasize again that our choice of RRT is arbitrary as ECT can be added to any roadmap-based and even grid-based motion planners. The measurements that we are interested in comparing include success rate and number of replans. The success rate is the number of runs that robot reaches the goal over the total number of runs, and the number of replans is the number of times that the robot replans to reach the goal.
5.5.2 \( R \) is a Point

In this section, we talk about experiments with robot being a single point. The environments are shown in Fig. 5.5. The maximum translational velocity of a dynamic obstacle is set to 2\( m/s \) and the maximum angular velocity is set to 3\( \text{radians}/s \). The experiments are conducted for multiple situations when robot’s velocity is 1, 2, 4, 8 and 16\( m/s \) and fixed-time strategy replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.

Compare to Fixed-Time Strategy

Fig. 5.6, Fig. 5.7, Fig. 5.8 and Fig. 5.9 plot the comparisons on success rate and number of replans respectively for environments in Fig. 5.5. Each data point from these plots is collected over 100 runs.

Success Rate and Number of Replans. From the plots in Fig. 5.6, Fig. 5.7, Fig. 5.8 and Fig. 5.9, we show that our approach using predicated collision helps the robot achieve nearly optimal success rate with a small number of replans. First, let us look at Fig. 5.6 and Fig. 5.7. We see that the success rate of the proposed method is almost identical to the fixed-time strategy with very high (and almost unrealistic) replanning frequency (i.e. replan every 0.05 sec.). This is especially clear when the robot’s velocity is greater than 2\( m/s \). However, frequent updates introduce a large number of replans. In Fig. 5.8 and Fig. 5.9, in order to provide a success rate similar to the proposed method, the fixed-time strategy needs to replan around 100 times more.

Running Time.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our method</td>
<td>2.68</td>
</tr>
<tr>
<td>Replan every 0.05 sec</td>
<td>25.70</td>
</tr>
<tr>
<td>Replan every 0.1 sec</td>
<td>8.76</td>
</tr>
<tr>
<td>Replan every 0.2 sec</td>
<td>4.00</td>
</tr>
<tr>
<td>Replan every 0.5 sec</td>
<td>1.66</td>
</tr>
<tr>
<td>Replan every 1.0 sec</td>
<td>0.97</td>
</tr>
</tbody>
</table>

In the table above, we provide average computation times spent on replanning over all ten environments. We observe that, to achieve similar success rate, our method runs about 3 and 12 times faster.
Figure 5.5: Ten environments used in experiments. In any of them, a green dot and a blue dot indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red dot indicates the robot’s current position when the image is captured. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions.
Figure 5.6: Compare our method to the fixed-time strategy on success rates for environments (a)-(e) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.
Figure 5.7: Compare our method to the fixed-time strategy on success rates for environments in (f)-(j) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. For every specific robot velocity, (k) plots the average success rate over all these 10 environments.
Figure 5.8: Compare our method to the fixed-time strategy on number of replans for environments (a)-(e) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. Notice that the y-axis is in logarithmic scale.
Figure 5.9: Compare our method to the fixed-time strategy on number of replans for environments (f)-(j) in Fig. 5.5. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds. For every specific robot velocity, (k) plots the average number of replans over all these 10 environments. Notice that the y-axis is in logarithmic scale.
Table 5.2: Number of updates per second required to achieve at least 80% success rate for fixed-time strategy.

<table>
<thead>
<tr>
<th>Environment</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 5.5 (a)</td>
<td>2</td>
</tr>
<tr>
<td>Fig. 5.5 (b)</td>
<td>1</td>
</tr>
<tr>
<td>Fig. 5.5 (c)</td>
<td>5</td>
</tr>
<tr>
<td>Fig. 5.5 (d)</td>
<td>5</td>
</tr>
<tr>
<td>Fig. 5.5 (e)</td>
<td>2</td>
</tr>
<tr>
<td>Fig. 5.5 (f)</td>
<td>2</td>
</tr>
<tr>
<td>Fig. 5.5 (g)</td>
<td>2</td>
</tr>
<tr>
<td>Fig. 5.5 (h)</td>
<td>5</td>
</tr>
<tr>
<td>Fig. 5.5 (i)</td>
<td>5</td>
</tr>
<tr>
<td>Fig. 5.5 (j)</td>
<td>5</td>
</tr>
</tbody>
</table>

than fixed-time strategy with time step 0.1 and 0.05 sec, respectively.

**Analysis of Difficulties**

To measure the difficulty of each path planning problem in Fig. 5.5, we collect the number of updates per second that the fixed-time strategy has to perform in order to achieve an at least 80% success rate over all 100 runs. From Table. 5.2, we can see that Fig. 5.5 (d) and 5.5 (j) are more difficult to solve than Fig. 5.5 (g). This is because in Fig. 5.5 (d), the robot needs to avoid a large number of moving obstacles and in Fig. 5.5 (j), the robot has to pass through narrow passages as well as avoiding long bars which are translating and rotating simultaneously.

**Compare to a Conservative Optimal Strategy**

We further compare our method to an optimal strategy proposed by van den Berg and Overmars [67]. In their work, every obstacle must be a disc and its swept volume over time is conservatively modeled as a cone with the slope being its maximum velocity. Therefore, the path, if any, generated by their method is guaranteed to be safe.

To apply their strategy in our environments shown in Fig. 5.5, we replace the obstacles with their
smallest bounding circles. Static obstacles are modeled as moving obstacles with zero velocity. Also note that bounding box is not allowed in their method. Our experiments found that, the robot needs to move at $22m/s$ or faster in order to find a safe path in Fig. 5.5 (d), and at least $15m/s$ in Fig. 5.5 (i). No path can be found at lower speed in these environments. For environments in Fig. 5.5 (c), Fig. 5.5 (g) and Fig. 5.5 (j), the start or the goal is covered by one or more obstacles at the very beginning, thus no path can be found. On the contrary, the proposed method provides better flexibility while still allows the robot to achieve a nearly 90% success rate at $4m/s$ and almost 100% at $8m/s$.

5.5.3 Obstacles are Articulated

In this section, we talk about experiments when obstacles could be articulated. The environments are shown in Fig. 5.10. For a dynamic rigid obstacle, its maximum translational velocity is set to $2m/s$ and its maximum angular velocity is set to $3$ radians/s. The experiments are conducted for multiple situations when robot’s velocity is 1, 2, 4, 8 and $16m/s$ and fixed-time strategy replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.

Compared to Fixed-Time Strategy

The comparison results are plotted in Fig. 5.11 and Fig. 5.12. Each data point is collected over 100 runs. From the plots, we can see that our collision prediction tool helps the robot achieve high success rate with only a small number of replans. For the fixed-time strategy, in order to have a decent success rate that is comparable to our proposed method, replannings are performed in very high frequency (every 0.05 seconds or 0.1 seconds). Consequently, a large number of replans are introduced (around 100 times more than our proposed method).

Analysis of Difficulties

To measure the difficulty of each path planning problem in Fig. 5.10, we collect the number of updates *per second* that the fixed-time strategy has to perform in order to achieve an at least 80%
Figure 5.10: Five environments used in experiments when robot is a point and obstacles could be articulated. In any of them, a green shape and a blue shape indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red shape indicates the robot’s current position. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions.
Figure 5.11: Compare our method to the fixed-time strategy on success rates for environments in Fig. 5.10. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.
Figure 5.12: Compare our method to the fixed-time strategy on number of replans for environments in Fig. 5.10. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.
success rate over all 100 runs. From Table 5.3, as we expect, Fig. 5.10 (d) and Fig. 5.10 (e) are much more difficult to solve than the other three environments. This is because the articulated obstacles in Fig. 5.10 (d) and Fig. 5.10 (e) have more linkages than those in other environments which are less controllable and introduce more uncertainty in obstacles’ motion.

5.5.4 $R$ is a Polygon

In this section, we talk about experiments when the robot is an arbitrary shaped polygon. The environments are shown in Fig. 5.13. For a dynamic rigid obstacle, its maximum translational velocity is set to $2 m/s$ and its maximum angular velocity is set to $3 \text{ radians/s}$. The experiments are conducted for multiple situations when robot’s velocity is 1, 2, 4, 8 and 16 $m/s$ and fixed-time strategy replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.

**Compared to Fixed-Time Strategy**

The comparison results are plotted in Fig. 5.14 and Fig. 5.15. Each data point is collected over 100 runs. Compared to the scenarios where robot is a single point, when robot is an arbitrary polygon, planning a collision free path to goal becomes much more difficult. Therefore, the success rate of our new path planner is only around 50% in some environments (e.g. (c)-(f) in Fig. 5.10). However, our collision prediction tool still helps the robot achieve much higher success rate than fixed-time strategy with only a small number of replans (around 100 times fewer than fixed-time strategy).
Figure 5.13: Six environments used in experiments when robot is a polygon. In any of them, a green shape and a blue shape indicate start position and goal position, respectively. Black obstacles are static and light gray obstacles are dynamic. A red obstacle is the one which introduces earliest collision with the robot. A green curve shows the trajectory that the robot has traversed. So a red shape indicates the robot’s current position. A brown dot shows the predicted location where the earliest possible collision might happen. A blue curve shows the path that robot plans to take. Notice that this path might be changed later due to possible collisions.
Figure 5.14: Compare our method to the fixed-time strategy on success rates for environments in Fig. 5.13. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.
Figure 5.15: Compare our method to the fixed-time strategy on number of replans for environments in Fig. 5.13. In the fixed-time strategy, the robot replans every 0.05, 0.1, 0.2, 0.5 and 1.0 seconds.
Table 5.4: Number of updates per second required to achieve at least 80% success rate for fixed-time strategy.

<table>
<thead>
<tr>
<th>Environment</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 5.13 (a)</td>
<td>100</td>
</tr>
<tr>
<td>Fig. 5.13 (b)</td>
<td>132</td>
</tr>
<tr>
<td>Fig. 5.13 (c)</td>
<td>181</td>
</tr>
<tr>
<td>Fig. 5.13 (d)</td>
<td>123</td>
</tr>
<tr>
<td>Fig. 5.13 (e)</td>
<td>200</td>
</tr>
<tr>
<td>Fig. 5.13 (f)</td>
<td>167</td>
</tr>
</tbody>
</table>

**Analysis of Difficulties**

To measure the difficulty of each path planning problem in Fig. 5.13, we collect the number of updates *per second* that the fixed-time strategy has to perform in order to achieve an at least 80% success rate over all 100 runs. From Table 5.4, as we expect, path planning for polygonal robot requires much more frequent replanning than for point robot (Table 5.2). This is especially true for the environments in Fig. 5.5.4 and Fig. 5.5.4 due to large number of dynamic obstacles or narrow passages.

**5.6 Discussion**

In this chapter, we discussed an adaptive method that predicts collisions for obstacles with unknown trajectories. We believe that this collision prediction has many potential usages and advantages. Similar to collisions detection in the setting of known obstacle motion, we have shown that collision prediction allows the robot to evaluate the safety of each edge on the extracted path with unknown obstacle motion. When the robot travels on a predetermined path, collision prediction enables adaptive repairing period that allows more robust and efficient replanning. Comparing to a planning strategy that replans periodically at *fixed time interval*, our experimental results show strong evidences that the proposed method significantly reduces the number of replans while maintaining higher success rate of finding a valid path. Even though the obstacles are modeled as adversarial agents in this paper,
we are currently investigate strategies to incorporate the constraints in obstacles’ motion when better behavior patterns of the obstacle are known [68].
Chapter 6: Collision Prediction

Collision detection is a fundamental geometric tool for sampling-based path planners. On the contrary, collision prediction for the scenarios that obstacles’ motion is unknown is still in its infancy. This chapter talks about a new approach we designed to predict collision by assuming that obstacles are adversarial. Our new tool advances collision prediction beyond the translational and disc robots [67]; arbitrary polygons with rotation can be used to better represent obstacles and provide tighter bound on predicted collision time. Comparing to an on-line path planner that replans periodically at fixed time interval, the experimental results in Chapter 5 provide strong evidences that this new approach significantly reduces the number of replannings while maintaining higher success rate of finding a valid path.

This new geometric tool we designed is called collision prediction. It allows the robot to determine the critical moments when robot and obstacles can collide. Then only at critical moments, the robot updates its belief of the environmental configuration and replans if necessary. The main challenge in predicting collision stems from the assumption that obstacle’s motion is unknown. To provide conservative estimation, the basic framework introduced in this approach models the obstacles as adversarial agents that will minimize the time that robot remains collision free. Consequently, a robot can actively determine its next replanning time by conservatively estimating the amount of time (i.e., earliest collision time) that it can stay on the planned path without colliding with the obstacles. The idea of earliest collision time and conservative advancement is our main focus of this chapter.

In Chapter 5, we discussed how to predict collision for the case where both the robot and obstacles are points. In this chapter, we will talk about collision prediction for the following robot/obstacle types.

- $\mathcal{R}$ is a point and $\mathcal{O}_i$ is a polygon,
• \( R \) is a point and \( O_i \) is an articulated polychain,

• or both \( R \) and \( O_i \) are polygons.

### 6.1 Point-Polygon Case

![Diagram](image)

Figure 6.1: (a) If \( A \) is fixed and \( B \) translates towards \( A \) along the closest direction, the closest features between \( B \) and \( A \) remain the same until they collide. (b) A polygon rotates around a point \( o \). The area it sweeps over is bounded by the trajectory of every vertex (\( p_1 \) through \( p_4 \)) and edges from both original and destination shapes.

In this section, we focus on the case where robot \( R \) is a point and obstacle \( O_i \) is a polygon that can translate and rotate around a given reference point \( o \). This also includes the case that \( R \) is translational thus can be reduced to a point via the Minkowski sum of \(-R\) and \( O_i\).

The robot \( R \) and obstacle \( O_i \) collide when the closest distance between \( R \) and the boundary of \( O_i \) becomes zero. Let us first consider a simpler case that \( R \) is static and \( O_i \) can only translate. In this case, the trajectory that brings \( R \) and \( O_i \) together is a straight line connecting their closest features (Fig. 6.1). During the translation, the closest features between \( R \) and \( O_i \) remain the same, and therefore identical to the point-point case discussed previously. However, when \( O_i \) can rotate, the closest features between \( R \) and \( O_i \) might change.
Based on the above observation, we consider $O_i$’s rotation and translation separately. That is, $ECT_{ij}$ can be determined by analyzing the distance between $R$ and the swept area of $O_i$ rotating around $o$. Let $SA_i^t$ be $O_i$’s swept area created by rotating $O_i$ with maximum angular velocity $\omega_i$ for time $t$ as illustrated in Fig. 6.1. Because $SA_i^t$ is the union of the swept area of every edge of $O_i$, $ECT_{ij}$ is simply the minimum among all earliest collision times of the edges in $O_i$ and $R$. This simple observation allows us to focus on one single edge of $O_i$.

![Diagram](a) (b)

Figure 6.2: (a). The swept area of $p_1p_2$ rotating is bounded by this gray shadowed donut-shape. The closest distance between $p_1p_2$ and $c$ is realized when $p_2$ becomes collinear with $o$ and $c$. (b) If $p_1p_2$ is fixed and $c$ rotates around $o$ with velocity $-\omega$, both the closest features and closest distance will be the same as in the case where $c$ is fixed and $p_1p_2$ rotates around $o$ with $\omega$.

Now we consider a moving segment $p_1p_2 \in O_i$ colliding with $R$. Shown in Fig. 6.1, the swept area of $p_1p_2$ is a donut-shaped area bounded by two concentric circles centered at the reference point $o$ traced out by $p_1$ and $p_2$. Without loss of generality, it is assumed that $p_2$ forms the bigger circle.

Given a configuration $c(t) \in c_jc_{j+1}$ which represents the location of $R$ at time $t$, we are interested in solving $f(t)$ which is the earliest moment when $p_1p_2$ hits this $c(t)$.
6.1.1 ECT of $\overline{p_1p_2}$ and $c \in \Pi$

We separate our analysis into two cases: (1) $\overline{p_1p_2}$ and $c$ are sufficiently far apart, and (2) $\overline{p_1p_2}$ and $c$ are sufficiently close.

Before we detailed our analysis, we found that fixing $\overline{p_1p_2}$ and rotating $c$ around $o$ significantly simplifies our discussion. That is, if $\overline{p_1p_2}$ rotates around $o$ with velocity $\omega_i$, then the closest distance will not change if $c$ rotates around $o$ with velocity $-\omega_i$ (see Fig. 6.1).

Let us first consider the situation that the segment $\overline{p_1p_2}$ and the point $c$ are sufficiently far apart so that when $\overline{p_1p_2}$ moves at maximum (rotational and translational) speed, translation takes more time than rotation. In this case, the optimal motion is to translate $\overline{p_1p_2}$ along $\overline{oc}$ while rotating $\overline{p_1p_2}$ until $p_2$ is collinear with $o$ and $c$. Thus, ECT of $\overline{p_1p_2}$ and $c$ is simply

$$\left( |\overline{oc}| - |\overline{op_2}| \right) / v_i \text{ , when } |\overline{oc}| \geq (\phi/\omega_i)v_i + |\overline{op_2}| ,$$

(6.1)

where $\phi$ is the rotation needed to make $p_2$, $o$ and $c$ collinear.

![Figure 6.3: Three cases when c is sufficiently close to the segment p1p2](image)

Figure 6.3: Three cases when $c$ is sufficiently close to the segment $\overline{p_1p_2}$

When $c$ is sufficiently close to the segment $\overline{p_1p_2}$, $\overline{p_1p_2}$ can hit $c$ before $p_2$, $o$ and $c$ become collinear. Depending on the relative position of $c$ and the swept area of $\overline{p_1p_2}$, the motion strategy taken by
will be different. Illustrated in Fig. 6.3, there are three cases we have to analyze. When \( c \) orbits around \( o \), the closest feature between \( c \) and \( \overline{p_1p_2} \) changes among \( p_1, p_2 \) and the points in \( \overline{p_1p_2}^\circ \), the open set of \( \overline{p_1p_2} \). If \( c \) is outside the circle traced out by \( p_2 \) (Fig. 6.1.1), the closest feature can change four times from \( p_2 \) to \( \overline{p_1p_2}^\circ \) to \( p_1 \) to \( \overline{p_1p_2}^\circ \) and back to \( \overline{p_1p_2}^\circ \). If \( c \) overlaps with the swept area of \( \overline{p_1p_2} \) (Fig. 6.1.1), the closest feature changes twice between \( p_1 \) and \( \overline{p_1p_2}^\circ \). If \( c \) is inside the circle traced out by \( p_1 \) (Fig. 6.1.1), the closest feature also changes twice between \( p_1 \) and \( \overline{p_1p_2}^\circ \).

Determining these closest feature changes (i.e., \( \alpha \) and \( \beta \) in Fig. 6.3) is straightforward; they are the intersections between the circle traced out by \( c \) (around \( o \)) and the lines containing \( p_1 \) or \( p_2 \) and perpendicular to \( \overline{p_1p_2} \). We will talk about this in details later.

If we let the closest distance between \( c \) and \( \overline{p_1p_2} \) be a function \( d(t) \) of time (we will talk about how to formulate \( d(t) \) for all three cases of Fig. 6.3 later), and let \( t_T \) be the time that the point \( c \) needs to translate at velocity \( v_i \), and let \( t_R \) be the time that \( c \) needs to rotate at velocity \( -\omega_i \). Because \( t_T \) is a function of \( t_R \), we let \( t_T = h_T(t_R) = d(t_R)/v_i \), where \( d(t_R) \) is the distance between \( c \) and segment \( \overline{p_1p_2} \) when \( c \) rotates \( \theta = -t_R\omega \) around \( o \). The ECT between \( p \) and \( \overline{c_1c_2} \) is

\[
\text{ECT} = \arg\min_{t_R} \left( \max \left( t_R, h_T(t_R) \right) \right)
\]

\[
= \arg\min_{t_R} \left( |t_R - h_T(t_R)| \right).
\] (6.3)

Therefore, ECT is \( t_R \) such that \( t_R = d(t_R)/v_i \). In other words, since both translation and rotation decrease the closest distance between \( \mathcal{R} \) and \( \mathcal{C}_i \), in order to detect the earliest collision time, \( t_T \) must equal \( t_R \).

**Computation of \( \alpha \) and \( \beta \)**

To detect \( \alpha \) and \( \beta \), we transform the reference point \( o \) to the origin and \( \overline{p_1p_2} \) to be aligned with the \( x \)-axis. Let \( p_1 = [a_1, a_2] \) and \( p_2 = [b_1, b_2] \). The line passing through \( p_1 \) and perpendicular to \( \overline{p_1p_2} \) is
The straight line passing through \( p_2 \) and perpendicular to \( \overline{p_1p_2} \) is \( L_2 : x = b_1 \). Let \( t_R \) be the moment when \( c \) reaches \( \alpha_1 \) or \( \beta_1 \), then \( t_R \) satisfies the following constraint.

\[
\cos(\omega t_R)c_x + \sin(\omega t_R)c_y = a_1 .
\] (6.4)

Similarly, the moment \( t_R \) when \( c \) reaches \( \alpha_2 \) or \( \beta_2 \) satisfies

\[
\cos(\omega t_R)c_x + \sin(\omega t_R)c_y = b_1 .
\] (6.5)

Fig. 6.1.2 shows the plots of equations 6.4 and 6.5. It illustrates how \( t_R \) changes as \( \mathcal{R} \) moves from \( c_1 \) to \( c_2 \). Of course, both functions 6.4 and 6.5 are periodical because \( t_R \) is the variable of Trigonometric functions. We are only interested in the values of \( t_R \) in the first period due to ECT.

**Distance Function** \( d(t) \)

**Closest Point Is An Endpoint Of** \( \overline{p_1p_2} \). We first consider that case that the closest feature from \( \overline{p_1p_2} \) is \( p_1 \). Since \( p_1 \) is fixed and \( c(t) \) rotates around \( o \) with \(-\omega\),

\[
d(t) = |\overline{c(t)p_1}|.
\]

Therefore,

\[
d^2(t) = (x_t - a_1)^2 + (y_t - a_2)^2,
\]

where \( c(t) = (x_t,y_t) \) and \( p_1 = (a_1,a_2) \).

After applying Laws of cosines to \( \triangle op_1c(t) \),

\[
d(t) = |\overline{c(t)p_1}|
\]

\[
= |\overline{op_1}|^2 + |\overline{oc(t)}|^2 - 2|\overline{op_1}||\overline{oc(t)}| \cos \angle p_1oc(t) \tag{6.6}
\]
We can get \( c(t) \) by rotating \( c \) around the reference point \( o \) for \( \theta \) being \(-\omega t\). Let \( c = [x, y] \). Then

\[
x_t = \cos \theta x - \sin \theta y
\]
\[
y_t = \sin \theta x + \cos \theta y
\]

By applying the above equations, \( d(t) \) can be further represented as follows.

\[
d^2(t) = |op_1|^2 + |oc|^2
- 2\cos \theta (a_1 x + a_2 y) + 2\sin \theta (a_1 y - a_2 x) \quad (6.7)
\]

Assume \( c \) is on \( \overline{c_1c_2} \), then \( c \) can be interpolated with \( c_1 \) and \( c_2 \). In other words, \( c = [x, y] \) is parameterized by \( 0 \leq \lambda \leq 1 \) and \( \overline{c_1c_2} \). Let \( c_1 = [x_1, y_1] \) and \( c_2 = [x_2, y_2] \), then

\[
x = x_1 + \lambda(x_2 - x_1)
\]
\[
y = y_1 + \lambda(y_2 - y_1)
\]

To make it easier, \( \overline{c_1c_2} \) is rotated to be aligned with \( x \)-axis. Then

\[
y = y_2 = y_1.
\]

In order to detect ECT, we need to solve equation

\[
d^2(t_R) = (v_i t_R)^2
= |op_1|^2 + |oc|^2 - 2\cos \theta (a_1 x + a_2 y) + 2\sin \theta (a_1 y - a_2 x). \quad (6.8)
\]

where \( \theta = -\omega t_R \).
Equation 6.8 is a complex function with trigonometric functions and polynomials. It can be solved with trust-region methods such as Levenberg-Marquardt algorithm or the Dogleg algorithm.

**Closest Point Is** $\overline{p_1p_2}$. If $d(t)$ is defined between $\overline{p_1p_2}$ and $c$, then $d(t)$ is the distance from $c$ to the straight line containing $\overline{p_1p_2}$. To make it simpler, $\overline{p_1p_2}$ is rotated to be aligned with $y = x$ and $\overline{c_1c_2}$ is transformed accordingly. This line is easily computed and let it be

$$y = x + b.$$

Then $d(t)$ can be written as

$$d^2(t) = \frac{(x_t - y_t + b)^2}{2}.$$

By replacing $x_t$ and $y_t$ with the equation above, $d(t)$ can be represented as follows with $\theta$ being $-\omega t$.

$$d^2(t) = \frac{|\overrightarrow{oc}|^2 + b^2 - \sin 2\theta (x^2 - y^2)}{2}$$

$$- xy \cos 2\theta + b \cos \theta (x - y) - b \sin \theta (x + y) \quad (6.9)$$

**6.1.2 ECT of $\overline{p_1p_2}$ and $\overline{c_1c_2} \subset \Pi$**

The discussion in Section 6.1.1 allows us to partition an edge $\overline{c_1c_2} \subset \Pi$ into subsegment such that all configurations in each subsegment belong to one of the four classes identified in the previous section, i.e., sufficiently far, or case (a), (b) or (c) in Fig. 6.3. More specifically, if we relate time $t$ to $\alpha$ and $\beta$ in Fig. 6.3, we can get a plot similar to Fig. 6.4. In Fig. 6.1.2, if the robot $\mathcal{R}$ is between $c_1$ and $c$, the closest feature between $\overline{p_1p_2}$ and $\mathcal{R}$ is always $p_1$. If $\mathcal{R}$ is between $c$ and $c'$, the closest feature can change from $p_1$ to $\overline{p_1p_2}$. If $\mathcal{R}$ is between $c'$ and $c_2$, the closest feature between $\overline{p_1p_2}$ and $\overline{c_1c_2}$ can change four times. If there exists a configuration $c''$ between $c'$ and $c_2$ that is sufficiently away (not shown in Fig. 6.3), then the closest feature between $\overline{p_1p_2}$ and $\overline{c''c_2}$ is always $p_2$.

Recall that our goal is to determine the time of earliest collision for every configuration on $\overline{c_1c_2}$.
Figure 6.4: (a) The relationship between $t_R$ and $\alpha$ and $\beta$ when $\mathcal{R}$ moves from $c_1$ to $c_2$. (b) and (c) illustrate the configurations $c$ and $c'$ in (a). $L_1$ and $L_2$ are the lines perpendicular to $\overline{p_1p_2}$ and contain $p_1$ and $p_2$, respectively.

i.e., the function $f(t)$. For the subsegments (e.g. $\overline{c_1c}$ and $\overline{c'c_2}$) that the closest feature does not change, the function $f(t)$ of the subsegments is simply $t_R = d(t_R, p)/v_i$, where the point $p$ is $p_1$ or $p_2$ and $d(t_R, p)$ is the distance between $p$ and the configuration of the robot at time $t_R$. The function $d(t_R, p)$ is detailed in Appendix. For the subsegments (e.g. $\overline{cc'}$) that the closest features change with rotation, the function $f(t)$ of the subsegments can be determined by combining $t_R = d(t_R, p)/v_i$ and $t_R = d(t_R, \overline{p_1p_2'})/v_i$ that is valid only in the gray area shown in Fig. 6.1.2.
Therefore, the function $f(t)$ for the segment $c_1c_2$ can be determined in the piecewise fashion by solving $t_R$ in each of these subsegments. Fig. 6.5 show an example on the function $f(t)$ and the closest distance between an edge of an obstacle and the robot over time.

Similar to the analysis that we have done in Section 6.1, when $f(t) = t$, $O_i$ collides with $R$ at maximum velocity and $f(t) < t$ means $O_i$ can collide with $R$ at location $c$ if $O_i$ slows down. Otherwise, $O_i$ cannot reach $c$ before $R$ already passes $c$. Therefore, we are interested in detecting the collision region which is above $t_R = f(t)$ and below $t = t_R$ and also bounded by $t = t_1$ and $t = t_2$.

Note that, although the function $f(t)$ can be complex, the intersections can be determined by trust-region-based root-finding methods such as Levenberg-Marquardt algorithm or the Dogleg algorithm.
6.2 Polygon-Polygon Case

In this section, we assume both the robot $R$ and the obstacle $O_i$ are polygons. The robot $R$ rotates around its center of mass on the designated path $\Pi$. $O_i$ undergoes rotation around a given reference point $o$. Two separating objects will collide when their boundaries become contact with each other (Fig. 6.2).

Taking the same conservative advancement approach that we used in the previous chapter, we will focus our discussion on the motion strategy that an edge $q_1q_2$ of $O_i$ can take to hit an edge $p_1p_2$ of $R$ at a given time $t$. Let $m$ be the number of edges in $O_i$ and $n$ be the number of edges in $R$. Then the earliest collision time between $O_i$ and $R$ is the minimum collision time of these $mn$ segment pairs.

For two separating line segments $p_1p_2 \in R$ and $q_1q_2 \in O_i$, the earliest collision can only happen between an endpoint of $p_1p_2$ and $q_1q_2$ (Fig. 6.2.1) or an endpoint of $q_1q_2$ and $p_1p_2$ (Fig. 6.2.1). Collisions at the interior portion from both line segments (Fig. 6.2) can only happen after one of those two cases. In the remaining of this section, we focus our discussion on collision prediction between a point and a line segment.
Figure 6.7: Three types of intersections between two line segments. Note that for two line segments that are initially separated, (c) can only happen after (a) or (b).

6.2.1 Collision Prediction between an Edge from $\mathcal{R}$ and a Vertex from $\mathcal{O}_i$

Let $e = \overline{p_1p_2}$ be an edge from $\mathcal{R}$ and $q$ be a vertex from $\mathcal{O}_i$. We are interested in finding out the earliest time at which $q$ collides with $e$ when $\mathcal{R}$ is on the $j$-th path segment $\overline{c_jc_{j+1}} \in \Pi$. Our main idea is to formulate the problem of detecting earliest collision time as a nonlinear optimization problem since Trigonometric functions are involved.

Let $c$ be any configuration from $\overline{c_jc_{j+1}}$ and let $t'$ be the moment when $\mathcal{R}$ (actually its center of mass) reaches $c$. In order to hit $\mathcal{R}$, $\mathcal{O}_i$ has to reach $c$ no later than $t'$. Otherwise, the robot has already left. Therefore, we have the first constraint

$$t \leq t'.$$

There is another constraint of $t'$ we need to consider. Since $\mathcal{R}$ moves along its current path $\Pi$ with given velocities, it is easier to compute the time when $\mathcal{R}$ arrives at any configuration on $\overline{c_jc_{j+1}}$. Let $t_1$ and $t_2$ be the moments when $\mathcal{R}$ reaches $c_j$ and $c_{j+1}$, respectively. Then

$$t_1 \leq t' \leq t_2.$$
Let $\theta$ be $O_i$'s change of orientation during $t$ if it rotates around its reference point $o$. Since $O_i$ has a maximum angular velocity $\omega_i$,  
$$-\omega_i \times t \leq \theta \leq \omega_i \times t.$$  

Assume $q$'s coordinates at current time is $q = [q_x, q_y]$. Then after $O_i$ rotating about $o = [o_x, o_y]$ for $\theta$, its new position $p' = [p'_x, p'_y]$ is

$$q'_x = (q_x - o_x) \cos \theta - (q_y - o_y) \sin \theta + o_x$$
$$q'_y = (q_x - o_x) \sin \theta + (q_y - o_y) \cos \theta + o_y$$

Note that when computing the new position of $q$, we did not consider $O_i$'s translation. That is because for two non-intersecting objects who can only translate, it is easy to determine their motions in order to realize the earliest collision. We first compute their closest features and to make them hit each other as early as possible, we make both objects move towards each other’s closest feature. Therefore, the earliest collision time is decided based on the closest distance and their translational velocities.
Figure 6.9: For two objects which can also rotate, it is hard to predict their motions which achieve collisions as soon as possible.

However, for two non-intersecting object who can also rotate, it is hard to keep track of their closest features and it is not easy to determine their motions for earliest collision due to rotation. An easy way to deal with this is to separate the motion of translation from rotation. That is, first constrain the new position of $q$ only based on $O_i$’s maximum angular velocity and then impose the constraint of maximum translational velocity in terms of the closest distance between $q$ and $e$. Let $e' = [p'_1 p'_2]$ be the new position of $e \in R$ when $R$ is located at $c$, then the closest distance between $e'$ and $q'$ needs to be no larger than $v_i \times t'$.

To sum up, we formulate the problem of detecting the earliest collision moment between $e \in R$ and $q \in O_i$ as a non-linear optimization problem with the objective function $t'$. Note that we use $t'$ as the objective function instead of $t$. That is because $O_i$ could arrive at some location $l$ earlier than $R$ and in order to hit $R$, it could slow down or wait at $L$ for $R$. In this case, the earliest collision time is the time $R$ takes to reach $L$, which is $t'$.

It is subject to the following non-linear or linear constraints.
\[ t \leq t' \leq t_1 \leq t' \leq t_2 \]
\[ -\omega_i \times t \leq \theta \leq \omega_i \times t \]
\[ d(e' \in R, q' \in \mathcal{O}_i) \leq v_i \times t' \]

where

1. \( t \) is the time that \( q \in \mathcal{O}_i \) takes to hit \( e \in R \),
2. \( t' \) is the time that \( e \in R \) takes to hit \( q \in \mathcal{O}_i \),
3. \( t_1 \) is the time when \( R \) reaches \( c_j \) and \( t_2 \) is the time when \( R \) reaches \( c_{j+1} \),
4. \( \omega_i \) is \( \mathcal{O}_i \)'s maximum angular velocity,
5. \( v_i \) is the \( \mathcal{O}_i \)'s maximum translational velocity and
6. \( d(e \in R, q \in \mathcal{O}_i) \) is the closest distance between \( e' \) and \( q' \)

For detecting collisions between an edge \( \overline{q_1q_2} \in \mathcal{O}_i \) and a vertex \( p \in R \), the idea is the same. The only difference is the representation of closest distance between \( \overline{q_1q_2} \in \mathcal{O}_i \) and \( p \in R \) because \( R \) and \( \mathcal{O}_i \) undergo different motions.

### 6.3 Point-Articulated Case

In this section, we focus on collision prediction between a point robot and an articulated obstacle in 2D. For the sake of easiness, we assume each linkage is a line segment, so an articulated obstacle is a polyline (Fig. 6.10). The motion of such an articulated obstacle is unknown to the robot but constrained by the following assumptions.
1. An articulated obstacle $\mathcal{O}_i$ can translate as a rigid body and it has a maximum translational velocity $v_i$.

2. Any two adjacent linkages are connected by a revolute joint. Every revolute joint has a maximum angular velocity.

Therefore, in this case, the degree of freedom for such an articulated obstacle is the number of its joints plus two which are the translational degree of freedom. Let $m$ be the number of linkages in an articulated obstacle $\mathcal{O}_i$. So $\mathcal{O}_i$’s total degree of freedom is $(m + 2)$. We can represent $\mathcal{O}_i$ as $\mathcal{O}_i = \mathcal{J}_1 \mathcal{J}_2, ..., \mathcal{J}_m$ with $\mathcal{J}_j (1 \leq j \leq m)$ corresponding to each joint. Let $\omega_j$ be the maximum rotational velocity of joint $\mathcal{J}_j$. We can also represent $\mathcal{O}_i$ with a sequence of linkages as $\mathcal{O}_i = L_1 L_2, ..., L_m$ with $L_j (1 \leq j \leq m)$ corresponding to each linkage. Linkage $L_i$ is closer to the base than linkage $L_j$ if $1 \leq i < j \leq m$ and we call $L_i$ an ancestor of $L_j$.

We are interested in detecting the earliest time when collisions between a point robot and such an articulated obstacle $\mathcal{O}_i$ could possibly happen. To ensure safety and efficiency, this estimation should be conservative and tight.
6.3.1 Overview of Framework

Algorithm 6.3.1: \textsc{CollisionPredictArt}(Oi, R, t_1, t_2)

\begin{align*}
m &\leftarrow \text{NumJoints}(Oi) & (1) \\
t &\leftarrow t_2 & (2) \\
\text{for each } L_j &\in O_i \\
&\quad \left\{ \\
&\quad \quad t_j \leftarrow \text{CollisionPredict}(Oi, L_j) & (3) \\
&\quad \text{do } \left\{ \\
&\quad \quad \text{if } t_j < t \\
&\quad \quad \quad \text{then } t \leftarrow t_j \\
&\quad \text{do } \right\} \\
&\text{return } (t)
\end{align*}

Algorithm 6.3.1 describes the basic framework of our collision prediction method for a point robot and an articulated obstacle if the robot is on some path segment $c_jc_{j+1} \in \Pi$. Let us assume the robot reaches $c_j$ at $t_1$ and reaches $c_{j+1}$ at $t_2$. Note that the earliest collision time we want to predict needs to fall into the range $[t_1, t_2]$ because we consider each path segment on $\Pi$ separately and at this moment we are only interested in detecting collisions if the robot is on $c_jc_{j+1}$.

Start with the first linkage $L_1$ which is closest to the base. Without considering other linkages, we can compute the earliest time $t_1$ when $L_1$ hits the robot (statement 3). Then we move on to the next linkage $L_2$. With only considering linkages $L_1$ and $L_2$, we apply the collision prediction strategy (statement 3) and check if the robot can hit $L_2$ at an earlier time. This process repeats for all successive links. To compute the earliest moment when linkage $L_j$ hits the robot, we need to take the motions of all linkages $L_1$ through $L_j$ into consideration. That is because the pose of a linkage $L_j$ is affected by all its ancestor linkages $L_k$ with $1 \leq k < j$. 
Figure 6.11: The relative position between \( p \) and \( L_2 \) will not change if we reverse their motions. That is, fix linkages \( L_1 \) and \( L_2 \) and make \( p \) rotates around \( J_1 \) and \( J_2 \) in the opposite directions.

6.3.2 Collision Prediction between \( \mathcal{R} \) and a Linkage from \( O_i \)

In the remaining part of this section, we focus on describing how to predict earliest collisions between a point robot and a linkage of \( O_i \) when the robot moves on \( c_1c_2 \) (statement 2 in Algorithm. 6.3.1).

Like in the case where both robot and \( O_i \) are polygons, we formulate collision prediction as a non-linear programming problem. Take the linkage \( L_2 \) as an example. Assume the robot currently locates at \( p \in c_1c_2 \). Let \( t \) be the time that \( O_i \) takes to make \( L_2 \) hit \( p \) and let \( t' \) be the time that the robot takes to hit \( p \). Then we’ve got the first constraint \( t \leq t' \). In other words, \( L_2 \) needs to hit \( p \) no later than the robot. Otherwise, the robot has already left \( p \) when \( L_2 \) reaches there.

Before we detailed our analysis of the second constraint, we find that fixing the linkages and translating and rotating the robot around associated joints significantly simplifies our discussion. Shown in Fig. 6.3.2 and Fig. 6.3.2, linkage \( L_1 \) rotates around joint \( J_1 \) counterclockwise for \( \theta_1 = 45^\circ \) and linkage \( L_2 \) rotates around joint \( J_1 \) counterclockwise for \( \theta_2 = 60^\circ \). For any point \( p = [p_x, p_y] \)
where the robot locates on $c_1c_2$, its relative position with respect to $L_2$ remains unchanged if we reverse the motions. That is, fix linkages $L_1$ and $L_2$ and rotate $p$ around $J_1$ clockwise for $\theta_1$ and then rotate $p$ around $J_2$ clockwise for $\theta_2$. Note that the relative position of $p$ with respect to $L_1$ might change. However, this will not cause a problem because we only care about the time when $L_2$ hits the robot at $p$. Let $p' = [p'_x, p'_y]$ be $p$’s new position after all these rotations. Like in the case where both robot and $\mathcal{O}_i$ are polygons, we can separate translation from rotation. That is, the robot can hit $L_2$ if and only if $p'$ can reach $L_2$ with the constraint of $\mathcal{O}_i$’s maximum translational velocity. Therefore, the second constraint is that the closest distance between $p'$ and $L_2$ which is $d(p', L_2)$ needs to no larger than $v_i \times t'$. Moreover, since the rotation of each joint is constrained to a maximum velocity, we have

$$-\omega_1 \times t \leq \theta_1 \leq \omega_1 \times t$$

$$-\omega_2 \times t \leq \theta_2 \leq \omega_2 \times t$$

To sum up, we formulate the problem of detecting the earliest collision moment between the point robot and $L_2 \in \mathcal{O}_i$ as a non-linear optimization problem with the objective function $t'$. We use $t'$ as the objective function instead of $t$. That is because $L_2$ could hit some point $l$ on $c_1c_2$ earlier than the robot and in order to collide with the robot at $l$, it could slow down or wait at $l$ for $\mathcal{R}$ to arrive. In this case, the earliest collision time is the time $\mathcal{R}$ takes to reach $l$, which is $t'$. It is subject to the following non-linear or linear constraints.
\[ t \leq t' \]

\[ t_1 \leq t' \leq t_2 \]

\[-\omega_j \times t \leq \theta_j \leq \omega_j \times t \]

\[ d(p', L_2 \in O_i) \leq v_i \times t' \]

where

1. \( t \) is the time \( O_i \) takes to make \( L_2 \) to hit \( R \)

2. \( t' \) is the time \( R \) takes to hit \( L_2 \)

3. \( t_1 \) and \( t_2 \) are the moments when \( R \) reaches \( c_1 \) and \( c_2 \), respectively

4. \( \omega_j \) is the maximum angular velocity of joint \( J_j \)

5. \( v_i \) is \( O_i \)'s maximum translational velocity and

6. \( d(p', L_2) \) is the closest distance between \( p' \) and \( L_2 \)
Chapter 7: Conclusions and Future Work

In this thesis, I studied “discrete similar-workspace problem” and “continuous similar-workspace problem” in path planning which fall into one of the following three categories.

1. Static environments where the obstacles can be rearranged but not allowed to move.
2. Dynamic environments where the obstacles can move and their trajectories are fully known.
3. Dynamic environments where the motions of obstacles are unknown and the robot can only estimate the pose of each obstacle through on-board sensors.

I developed new approaches which explored the similarity across different environments thus provided significant efficiency improvements and better chances of finding feasible paths for the robot.

7.1 Static Environments

In Chapter 3, I introduced a path planner RU-PRM for static environments. Essentially, RU-PRM builds and stores the local roadmap for each C-obstacle which captures the free space in its proximity. When a new environment is given, RU-PRM matches the obstacles and loads the matched local roadmaps. The local roadmaps are transformed based on the poses of obstacles in the new environment and then merged to solve the new queries. I also introduced a new shape matching method for 2D models and applied the shape matching method [89] for 3D models which increase the reusability of local roadmaps. The experimental results are very encouraging and show significant efficiency improvements over existing well-known planners.

Many aspects in RU-PRM can be improved. For example, I plan to investigate better shape matching methods which are more efficient and provide more reasonable matching results. In this way, we could improve the reusability of existing computation even more. One aspect that is missing in
RU-PRM is articulated robots with a fixed base. Since a local roadmap is transformed based on the pose of its associated obstacle, this only works for a free-flying robot.

7.2 Known Dynamic Environments

In Chapter 4, we explored a path planner called “Critical Roadmap” for dynamic environments where obstacles’ trajectories are fully known. Our method takes advantage of motion coherence of obstacles and detects all critical times when topology of free space changes. Consequently, compared to existing methods that only use fixed-time resolution, our method provides a more complete representation of free configuration-time space. Moreover, at each critical time, RU-PRM allows us to reuse most portion of existing roadmap and repair only the invalid part caused by obstacles’ motion. Experimental results show that our new path planner is not only more complete, but also significantly improves time efficiency.

Estimation of penetration depth is one of the fundamental tools for detecting critical moments. Our current penetration depth computation relies on decomposing each object into a convex hull hierarchy. This approach does not provide a tight lower bound of the penetration depth between two non-convex shapes, and results in more number of iterations for conservative advancement. We plan to look for a direct way to compute penetration depth between two non-convex objects.

7.3 Unknown Dynamic Environments

In Chapter 5 and Chapter 6, we explored an adaptive method that predicts collisions for obstacles with unknown trajectories. Like collision detection in environments with known obstacle motion, we have shown that collision prediction allows the robot to conservatively evaluate the safety of its current path without knowing motion of obstacles. Compared to a planning strategy that replans periodically at fixed time intervals, our method enables adaptive repairing period that provides more robust and efficient replanning. Experimental results show that the new method significantly reduces the number of replans while maintaining high success rate of finding a valid path.
Interesting extensions from this thesis include how we predict collision when the robot and the obstacles are polyhedra. A shape in 3D workspace could be very complicated and noises are usually involved when a polyhedron is created. One possible way to address this is to approximate a complicated polyhedron with a simple bounding volume such as an oriented bounding box.

Currently in collision prediction, we consider each obstacle independently. In other words, while estimating earliest collisions for a dynamic obstacle, we simply ignore other obstacles present in the environment, such as all static obstacles. Therefore, our collision prediction method can be improved by taking all static obstacles into consideration. In this way, we can get a tighter prediction.

Another aspect that is missing in our strategy is that we only consider the maximum velocities of obstacles. No other dynamic constraints or kinematic constraints have been considered. For example, we could assume that the maximum magnitude of the acceleration of each dynamic obstacle is known in advance. This is reasonable in practice since an object in the real world cannot move arbitrarily fast. We could also investigate methods which incorporate the constraints in obstacles’ motion when behavior patterns are known. Also, because behavior modeling and dynamic perception and prediction of moving objects are well studied, integrating those techniques would be very interesting.

7.4 Future Research Direction

The new path planner which explores similarity across different static environments could be scaled up to large spaces such as a building which is composed of plenty of similar classrooms and offices, or even cities or campus which is composed of many buildings. The new path planners can also be scaled up to long time period. For example, we can create a lifelong learning robot whose ability to do a task improves as the robot performs similar tasks. Considering an example of robot putting dishes back to a cluttered cupboard. This is a very typical narrow passage problem and solving such problems is usually quite time-consuming. However, since the geometry of kitchen cupboard does not change much between queries, previously-computed paths will be very close to valid.

In a broader perspective, the works presented in this thesis are only first steps towards exploration
of connections across different environments or across different moments in a single dynamic environment. In addition to similarity in different configuration obstacles, we could explore the similarity in topology of free configuration space over different environments or over different moments in a dynamic environment.

Besides, the works in the thesis could be applied as important geometric tools to many existing path planners [55, 59, 60, 62]. Yoshida and Kanehiro [61] proposed an on-line replanning method with parallel planning and execution and roadmap reuse. However, this strategy is only suitable for discrete environmental changes. With the help of collision prediction strategy (Chapter 6), this on-line path planner [61] can be extended to more applications where continuous environmental changes are taken into account. There exist plenty of work [55, 59, 60] exploring path planners that can avoid inevitable collision states. The new collision prediction strategy can help filter out those states which are guaranteed non inevitable collision states.
Bibliography


Curriculum Vitae

Yanyan Lu was born in People’s Republic of China in 1986. She received her Bachelor of Science in Computer Science from Dalian University of Technology in 2007. In the same year, she joined the Department of Computer Science at George Mason University for her PhD study. She received her Master of Science in Computer Science from George Mason University in 2011.

Publication List


