Safe and Efficient Navigation of a Mobile Robot: Path Planning Based on Hierarchical Topology Map and Motion Planning with Pedestrian Behavior Model

by

Jeong-woo Han

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mechanical and Mechatronic Engineering

Waterloo, Ontario, Canada, 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Jaho Seo Assistant Professor, Dept. of Automotive, Mechanical and Manufacturing, Ontario Tech University
Supervisor(s):	Hyock Ju Kwon Associate Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo Soo Jeon Associate Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo
Internal Member:	William Melek Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo
Internal Member:	James Tung Associate Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo
Internal-External Member:	Nasser Lashgarian Azad Associate Professor, Dept. of System Design Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Safety in mobile robot navigation is an essential aspect, but it is often accompanied by a trade-off of efficiency in different navigation steps. For global planning, safety is frequently handled by inflated obstacles. Larger margins to the obstacles increase safety while lessening the path efficiency, such as longer path length. For local motion planning tasks, safety becomes even more critical because of dynamic objects in the robot's proximity but also accompanies substantial trade-offs, such as highly low-speed operations. In such aspects of safety and efficiency, we propose safe and efficient global path planning and local motion planning methods.

For global path planning, we proposed the *allowable speed of navigation* for safety, which limits the maximum speed based on the clearances in the environment. The corresponding cost formulates the traveling time, leading to planning a time-efficient global path. A new map representation is proposed, named *Hierarchical Topology Map* (HTM) and *Hierarchical Topology Map with Explicit Corridor* (HTM-EC), and incorporated with the proposed safety-aware navigation speed. HTM is a double-layered data structure of a topology graph and the corresponding metric skeleton points of a map, which returns a feasible, on-skeleton path in an extremely short time. HTM-EC is an extended map expression with the Explicit Corridor that incorporates the nearest obstacle points along with skeleton points into HTM, which returns an optimal, off-skeleton (i.e., metric) path with corridor optimization of the given on-skeleton path. When they are used together, safety-aware time-efficient paths can be planned with light computation. The efficacy of the safety-aware allowable speed and lighter computations have been verified with simulations and experiments.

For local motion planning, we incorporate a pedestrian navigation model to address efficiency. The pedestrian model seeks the desired direction of motion that minimizes the remaining distance to the destination. The cognitive collision locations are computed, which enables far-sighted path planning by incorporating future configurations of the surroundings. Safety guarantee is analyzed for the output of the pedestrian-model-based motion planner. Finally, to address the behavioral uncertainties, the *degree of cooperation* (DoC) is proposed that helps prediction of the other objects' velocities with online estimation. The simulation results with the different number of agents show that the pedestrian model can generate a smooth path between the agents. As the local planner uses fixed-length visual measurement such as LiDAR, the proposed planner is scalable regardless of the number of moving obstacles as well as the types of obstacles (i.e., static or dynamic).

Acknowledgements

I would like to thank all the little people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous mobile robots are getting increasingly popular for many applications in our society. Examples include industrial logistic sites, retail product delivery, and social guidance in airports. Those applications often have static obstacles (shelves, workstations) blended with dynamic, unpredictable agents (human workers, indoor vehicles, carts). Safe navigation is essential functionality for the utility of mobile robots in such environments. Despite recent progress in related technologies, mobile robots' efficient and safe navigation remains challenging for environments cluttered with complex moving agents[84, 77, 94], as there are somewhat inherent characteristics of a trade-off between safety and efficiency. For an extreme example, safety can be achieved by slowly moving, frequently stopping, or even not moving the mobile robot if there are any moving obstacles in the environment, which should cause low efficiency. On the opposite, efficiency, such as traveling time to the goal, can be maximized by moving the robot straight to the goal at high speed, which will increase the chances of collisions with other moving agents and thus decrease safety. The uncertainties lying in the unpredictable movement of the dynamic objects would be the main cause of the difficulty.

A popular framework for mobile robot navigation is to divide the task into two separate planning steps; *global planning* and *local planning* [37]. Global planning provides a mobile robot with an entire *path*, a set of positions to reach, from the current position to the goal. In this step, moving obstacles out of sight of the robot cannot be taken into account, and thus static obstacles are often the only types to be considered, which are known to the robot. Local planning, on the other hand, is the one that plays a role in the environment with dynamic obstacles. In this step, some parts of the path obtained from the global planning step are taken into account with moving objects in the robot's proximity. With presuming dynamic obstacles exist, motion planning can be regarded as equivalent to *local*

planning because both are aimed to provide the robot with either control actions or a trajectory that can be readily converted to control actions.

The global planning method often considers static obstacles only as the locations or even existences of dynamic obstacles cannot be possibly seen all the time for the entire navigation areas. Efficiency, often referred to as path optimality, has been the main interest of global path planning because poor global paths, such as unnecessary bypass, degrade overall navigation performance. Safety also needs to be addressed in the global path planning methods not only for diverse operations in safety-critical social environments but also for better path efficiency. In most global path planning methods, however, safety has been dealt with simply by securing some clearance from static obstacles (e.g., walls) with the inflated size of a mobile robot. The enlarged size of the robot will restrict the free spaces of given environments (such as a map), and the shortest path criterion has become the most popular for efficient path planning. Still, properly considered safety can improve the efficiency of the global path. For example, narrow passages can be often crowded by previously unseen obstacles, forcing low-speed operations. In reality, even a single person takes a substantial portion of a narrow passage cross-section, which is not a rare case to be seen in our daily life. An alternative path via spacious corridors may lead to faster traveling if the robot can move at high speed. In addition, if there exist dynamic obstacles along the path which were not considered when the path was planned, it will likely take much longer when the robot moves through narrow corridors.

In local motion planning, safety is more important as a mobile robot can see moving obstacles in its proximal surroundings. Considering other moving obstacles in the motion planning problem is challenging. More importantly, or most applications in our society, moving objects that a mobile robot encounters are not simple but rather complicated as they are often decision-making agents, particularly human beings. Motion planning with such agents in the surroundings becomes even more complex due to their interactions which make the agents' actions and the resulting paths jointly related. Without consideration, the robot can be stuck in the crowd, known to be *freezing robot problem (FRP)*, because of the lack of space that the robot thinks to go through. The issue of FRP forces a mobile robot to consider interactions with other agents in the motion planning problem.

Existing work to the cooperative motion planning with dynamic agents can largely be divided into two ways[17, 16]: *reaction-based* [30, 14, 95, 50, 33, 86, 87, 109, 108, 73, 98, 60, 89, 29] and *trajectory-based* [53, 95, 90, 17, 16, 55, 121, 103]. The main difference between them is whether the trajectory prediction is used. Without the trajectory prediction, reaction-based methods can quickly compute the control actions based on some interaction rules between the agents. However, they often suffer from short-sighted planning, which may cause oscillatory or unnatural behaviors in certain situations [17, 55]. On the other

hand, trajectory-based methods can produce better paths (e.g., shorter time for all agents to reach their goal) thanks to their far-sighted planning abilities based on the trajectory prediction with the expense of high computation. The high computation is caused by considering interactions between all the agents to predict their trajectories jointly[17]. In addition, the predicted trajectories often fail beyond a few seconds into the future, which results in the need for frequent prediction that even exacerbates the computation cost.

The tradeoff between the quality of the path and computation cost has been recently addressed with the rising learning-based techniques such as deep reinforcement learning (DRL). Those techniques use the framework of "off-line learning and on-line implementation"[17, 16, 24, 15]. These methods leverage DRL techniques to encode the agents' interactions and the policies for collision avoidance, then apply the learned policy when the robot is operated. We would say that such methods can be considered variants of the reaction-based methods because the agent's action is decided without trajectory prediction. While these approaches seem to have been successful in resolving the tradeoff between long-term planning and computational tractability, safety is not able to be addressed in an explicit manner because of the nature of learning. Another issue with those methods is the lack of scalability, meaning the number of agents matters to the performance.

In addition, behavioral uncertainties of other decision-making agents can be an issue in the mobile robot navigation environments. For example, human navigation behaviors can differ from person to person or even from situation to situation for the same person[55, 74]. Some might be cooperative, while some might not. While such an issue may be critical in many environments, few have studied such behavioral differences in the motion planning problem.

In this thesis, we aim to develop methods for the safe and efficient navigation of a mobile robot. With a widely used navigation framework, we formulate the navigation task as two separate tasks: global path planning and local motion planning. For each planning step, we specifically address the safety and efficiency tradeoff in a diverse perspective. Our proposed method for global planning will address the time-efficiency of the planned path, taking the safety into the cost of the path, which can be considered as the balance between them. In local motion planning, we focus more on safety as the planner has to consider immediate motion. While safety is explicitly addressed, a navigation model for pedestrian behaviors benefits the efficiency of the motion. The detailed objectives are stated as followings.

1.1 Objectives

- **Safe Global Path Planner with Allowable Speed:** A mobile robot may face tight environments because of other dynamic agents (e.g., employees, forklifts) or temporarily placed static obstacles (e.g., boxes of products) in the surroundings. High-speed operations of a mobile robot in such crowded environments would be risky due to the increasing likelihood of collisions. By intuition, the crowdedness because of the unexpected objects and the collision risks may be lessened if the mobile robot takes wide corridors as its route to the goal. To put this intuition into practice, we aim to develop a new global path planning method that takes account of a certain margin of safety of space and limits the operatable speed of a mobile robot. The higher speed options in the spacious areas can shorten the time to reach the goal for a mobile robot in practice, even if the distance becomes longer.

- **Efficient Local Motion Planner Based on Human Behavior Model with Safety Guarantee:** We aim to develop an efficient local planner with the presence of dynamic obstacles. Prediction of the movement of other agents is crucial for the quality of a path due to the ability of far-sighted planning, but long-term prediction requires heavy computation that may often be intractable for real-time operation. Human behavior model can be an alternative, as humans are sufficiently intelligent for far-sight plannning. In addition, considering that humans are the most common type of moving agents for the environment of mobile robot navigation in many social environments, the robot's navigation characteristics mimicking a human behavior model of navigation would be understandable by other humans. Thus we aim to develop a local motion planner based on a pedestiran navigation behavior model. While the path-efficiency can be achieved by the far-sight planning with human behavior, safety analysis of the planner will be the primarily focus.

- **Degree of Cooperation and Its Online Estimation:** We aim to address the behavioral uncertainties of individual humans to compensate for the model imperfection. No human model can be perfect for different persons, and even a person may behave differently from time to time, which is known as the inherent uncertainty in human behavior. To tackle the issue, we introduce a concept of degree of cooperation (DoC), an index of how well an individual agent follows the modeled behavior. The deviation of other agents from the robot's knowledge (model) can be captured in the DoC, and online estimation on DoC for each agent can be used for a minimum conservative planner. The DoC can be generalized to other types of agents whenever their behavior models are available.

1.2 Outlines

The remainder of the thesis is organized as follows. Chapter 2 contains a literature review on mobile robot navigation relevant to the scope of this research. In the following two chapters, we propose the methodologies for global path planning and local motion planning with dynamic agents. Chapter 3 addresses the global path planning and Chapter 4 tackles the local planning problem. Chapter 5 contains the conclusions of the thesis.

Chapter 2

Literature Review

This chapter presents a review of the current status of navigation tasks for mobile robots. Existing studies on global path planning are reviewed first in Section 2.1, and those on local motion planning in a dynamic environment are presented in Sections 2.2 and 2.3, which summarize the reaction-based methods and the trajectory-based methods, respectively. Conclusion of the literature review is presented in Section 2.4.

2.1 Global Path Planning

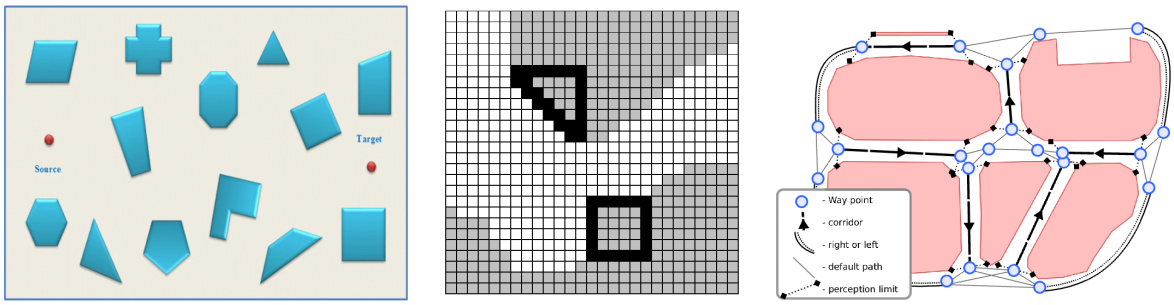
Global path planning methods often address only static obstacles with given maps. While dynamic obstacles are often absent in global planning, global planning methods should be importantly addressed because they form the basis of the subsequent navigation tasks, including motion planning and obstacle avoidance. More importantly, they often have a greater impact on the overall navigation efficiency than the local motion planning regardless of how the efficiency is defined (e.g., time efficiency or distance efficiency). For example, a global path with unnecessary long by-passes can increase the length of the path or traveling time. In addition, as more mobile robot applications are broadening their usage in our society, such as warehouse management or airport services, it is likely to have larger environments to be covered by global planning. It will require substantial computation when there are frequent replanning requests in such large environments. Therefore, global path planning should have the following aspects: *path optimaility* and *computational efficiency*. One more consideration should be the safety of a global path, as optimality always comes with a tradeoff with safety in mobile robot navigation.

2.1.1 Map Representations in Mobile Robot Navigation

Figure 2.1 shows some examples of map representations in robot navigation. *Cccupancy grid map* (hereinafter, shortly OGM) is the most popular map type[102]. OGM is a discrete map representation with square cells, shown in Fig. 2.1(b). Conventional OGM has binary occupancy values assigned to each cell to specify whether an obstacle occupies the cell or not. The popularity of OGM is due to its handling of sensor noises when a map is built, where cell occupancy is determined by the probability of obstacle existence. The more frequently obstacle is detected in a cell by sensors, the more likely an obstacle is in the cell. Above certain probabilities, the cell occupancy value can be set.

In a *polygonal map* shown in Fig. 2.1(a), obstacles are represented as polygons, with vertices information. The map is continuous and easy to understand with the set of vertices of polygonal obstacles within the map size. In practice, perfect polygonal maps hardly exist if a map is built from sensor data, and they are often approximately converted from OGM.

The other type of map for mobile robot navigation is shown in Fig. 2.1(c), which is *graph-based map*. In general, the graph-based map provides abstract map representation, which can be beneficial for faster planning with the cost of the map details. Different methods can be used to represent a map to build a graph from polygonal maps or OGMs, such as Visibility graph [83], or Probabilistic Roadmap [49].



(a) Polygonal Map (b) Occupancy grid map (c) Graph-based map

Figure 2.1: Different map representations for mobile robot navigation

2.1.2 Grid-based Approaches

Artificial Potential Field

Potential Field (PF)-based methods [50, 33, 86, 87] generally consider two types of effects: repulsive and attractive. The repulsive effect can be generated by the positions of static and dynamic obstacles, whereas the attractive effect can be generated by the goal position. Combining those effects in quantity creates a potential, and a map can be converted to a potential field, as shown in Fig.2.2. The driving force for a mobile robot can be obtained by taking negative spatial gradients of PF. The concept was originated in [50] that considers only static obstacles and has been extended to the cases of moving obstacles with the relative position and velocity [33], smoothness of potential function near obstacles[86], and selective moving obstacles having a different level of threats[87].

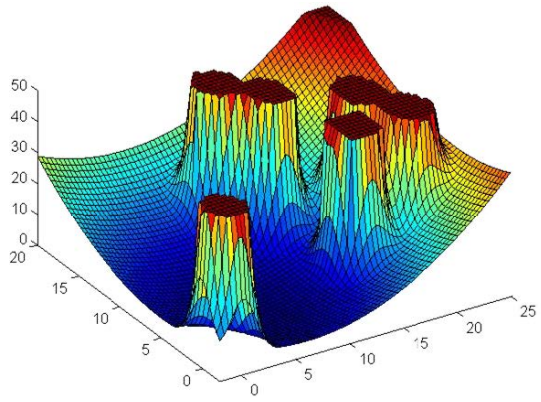


Figure 2.2: Artificial Potential Field [118]

2.1.3 Graph-based Approaches

Visibility Graph

Visibility Graph [67, 83] utilizes a polygon-map for polygonal obstacles and their vertices to construct a graph. Each of the start and the goal points are connected to its visible vertices to generate edges. Likewise, every vertex of an obstacle can create edges by linking itself to its neighboring vertices on the obstacle and to the visible vertices of other obstacles. This process is repeated until no edges are added to the graph. While the approach guarantees

the construction of the shortest path, the resulting path may get too close to the obstacles when the shortest path contains some narrow passageway.

Probablistic Roadmap

Probabilistic approaches are popular in constructing a graph. Probabilistic roadmaps (PRM) [49, 46, 93], a sampling-based method, is one of the popular approaches. The basic idea of these works is spreading sample points into the map and constructing edges between them. If some of the scattered samples are in the occupied regions of the map, they are removed before connecting the samples to generate edges. The edges can be formed by linking a sample to its neighboring samples in straight lines if the linking line is not crossing the boundaries of any obstacles. Once a sufficient number of nodes and edges are constructed, the start and goal points can be added to the graph as additional nodes. The shortest path schemes are applied to find the optimal path of the graph.

Rapidly-exploring Random Tree

Rapidly-exploring random tree (RRT) suggested in [62] is another popular method and has recently been widely used[57, 85, 64]. Unlike PRM that scatters sample points at once, RRT locates a sample point in the free space of the map and connects it to the closest node of the current graph. In this approach, it is possible to obtain the sample points with kinematic constraints (i.e., velocity), which naturally outputs motion primitives. If the newly added connection crosses any obstacles, the second closest node is chosen to be connected, and so on. Since the outer nodes most likely become "the closest point" to the sample, the resulting graph looks like a tree structure expanding from the terminal branches of the graph. The graph construction is over when a sample, which is close enough to be visible from the goal, is successfully connected to the existing graph. Compared to PRM, the RRT-based graph can be directly used for motion commands due to the consideration of robot kinematics. The probabilistic approaches are very fast, but they suffer from neglecting the regions of low visibility, such as narrow corridors where samples are less likely scattered. Consequently, the resulting graph is considered not complete (i.e., they do not guarantee to find a path).

Skeleton-based Methods

For the completeness of the graph, skeleton-based graph approaches have been widely used [11, 100, 18, 35]. Skeleton is a morphological representation of a shape (or binary

image) using a thin line and usually consists of a set of equidistant points from boundaries. The morphology is the abstract representation of a map, and thus these approaches are considered as complete methods [18] which guarantee that a path can be found within finite time if it exists. In addition, these approaches are considered as safe as possible [11] in that the points of the skeleton are located as far from the closest boundary as possible. Voronoi diagram (VD)[100, 18, 114, 34, 116, 81] , image thinning [122], and distance field map from boundary (DFB) [116] are popular methods for skeletonization of a map.

2.2 Local Motion Planning: Reaction-Based

Reaction-based approaches to the problem of motion planning in a dynamic environment use the current observation for the solution without look-ahead to the far horizon. We categorize existing methods as velocity-search-based, (interactive) force-based, and direction-search-based methods.

2.2.1 Velocity-Search-Based Approaches

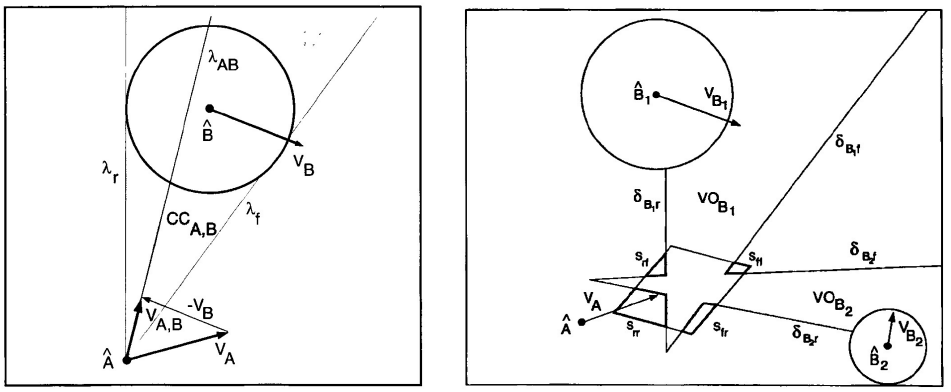
Velocity-search-based approaches generally divide the velocity space of a mobile robot into permitted and forbidden areas. Safe velocity command can be chosen out of the permitted area.

Velocity Obstacle

Velocity Obstacle (VO)[27] and its variants[32, 109, 119, 98] have been one of the most popular approaches for collision avoidance with dynamic objects due to its mathematical guarantee for collision-free output and low computational cost. VO represents a prohibited velocity set in the robot's velocity space, created by considering relative velocity between the robot and a moving obstacle.

The basic idea of constructing VO is as follows. Imagine two point-agents A and B are moving on a plane and crossing each other, assuming their current positions and velocities are given to them. Suppose the relative velocity of A to B is directly heading to the position of B from A. In that case, they will eventually collide because they will be getting closer if they keep their current velocities. Their velocities must be adjusted to avoid the collision such that the relative velocity vector is not heading to B from A. Thus, the line directed from A to B represents a set of relative velocity vectors with different magnitudes. If the

agents are circular-shaped, their radii are taken into account, and the prohibited line can be extended to a cone where its peak is on A. The cone is tangential to a circle centered at B with the radius given by the summation of the radii of A and B. The area inside the cone represents prohibited relative velocities because a collision is expected. Finally, the cone can be translated by the velocity vector of A. This translated cone is a Velocity Obstacle of A, representing prohibited velocities for agent A. Collision-free velocity command for A can be mathematically guaranteed by taking any velocities outside VO. A graphical representation of VO is shown in Fig.2.3(a). For multiple moving objects, multiple VOs for a robot can be calculated corresponding to each object and overlapped all together to make the prohibited area as shown in Fig.2.3(b). Non-holonomic kinematic constraints were studied in [119] with a car-like model and uncertainties, related to the position and velocity of objects due to sensor imperfection were studied in [51, 52, 32] in occupancy-grid environments.



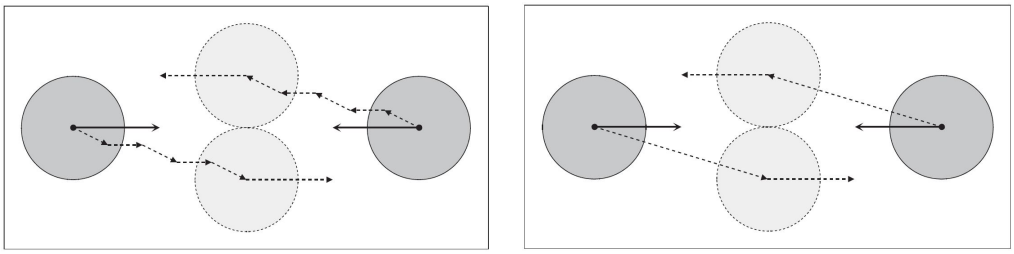
(a) Velocity obstacle (b) Velocity obstacles with muliple objects

Figure 2.3: Velocoty Obstacle[27]

Reciprocal Velocity Obstacles (RVO)

Although VO can generate collision-free commands, it has been shown to generate unwanted oscillations [109] because the choice of velocity does not take account of the change of velocity of the approaching agent. It may result in too much outward adjustment of an agent's velocity if the other agent also tries to avoid the collision and moves away. Likewise, it may result in too much inward adjustment if the other decides to move toward the agent to reduce the excessive adjustment. Figure 2.4(a) illustrates an example of a case causing oscillation.

Reciprocal Velocity Obstacle (RVO) was suggested in [109] to resolve the oscillatory problem of VO by expecting the change of approaching agent's velocity. In this approach, the "change" is assumed reciprocal. It means that the collision-avoiding agents take equal responsibility for the expected collision: each agent takes the half amount of velocity change to make their relative velocity out of VO. With the reciprocity assumption, RVO was able to guarantee oscillation-free and collision-free motion, as illustrated in Fig.2.4(b). One of the extensions of RVO is Hybrid Reciprocal Velocity Obstacle (HRVO)[98] that naturally provides a mechanism with each agent to choose the velocities on the same side to the other by applying an asymmetry RVO cone. The asymmetric cone can be created by combining one side of RVO and the other side of VO. Generalization of the reciprocal collision avoidance was studied in [7].



(a) Oscillation of VO (b) Mitigation of oscillation in RVO

Figure 2.4: Illustration of oscillatory movement in VO and oscillation-free movement in RVO [109]

Optimal Reciprocal Collision Avoidance (ORCA)

As another reciprocal strategy of collision avoidance, Optimal Reciprocal Collision Avoidance (ORCA) was proposed in [108] to address efficient collision avoidance for multiple objects. Unlike VO and its variants, ORCA restricts the area of permitted velocity sets (cone-shaped areas) into a half-plane created by adjusting the collision-expecting velocity-set to be outside of VO. Like RVO approaches, same-responsibility for the collision (i.e., reciprocity) was assumed in the mechanism of creating the half-plane. The permitted area of velocity bounded by the half-plane of the velocity space is a subset of the area outside RVO. If an agent velocity is in the forbidden area of the half-plane, it is adjusted to the velocity closest to the boundary of the half-plane as the optimal velocity. Figure 2.5 shows the comparison between VO, HRVO, and ORCA.

ORCA has a few advantages over the variants of VO and RVO. For multiple objects, overlapping ORCA provides a convex created by a set of the ORCA line, which allows the use of linear programming, a fast optimization algorithm, while the safety of optimized velocity is mathematically guaranteed. Also, the half-plane constructed by ORCA gives the dynamic obstacles the clue of selecting the same side of their directions of passing. These benefits have made ORCA one of the most popular in literature, and the method has been successfully applied to numerous multi-robot navigation. Still, there remain some limitations, such as the assumptions of reciprocity and agent-level measurement. Also, it does not consider static obstacles such as corridors and often shows a myopic behavior [89, 17, 16, 29].

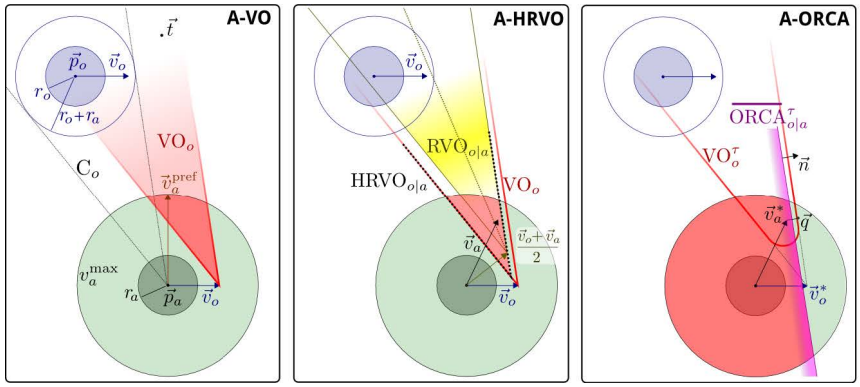


Figure 2.5: Comparison between VO, HRVO, and ORCA [36]

Dynamic Window Approach

Dynamic Window Approach (DWA)[30] is one of the most popular methods for local planning in practice. The original concept was proposed in [30] and several extensions have been proposed in [14, 95] addressing real-time implementation and dynamic obstacles.

Figure 2.6 describes the velocity-search step of DWA. The dark areas represent occupied space (i.e., obstacles), and the square in the middle of the right represents the (dynamic) window for achievable velocities from the current velocity (the center point in the window). Only the velocities of the light area in the window are considered admissible. An optimized pair among the admissible velocities can be selected by maximizing an objective function that considers target heading, distance from the closest obstacle, and forward velocity.

DWA has some advantages as follows. Kinematic constraints (i.e., maximum velocity and acceleration) are inherently involved in the method [77], which enables one to generate

direct control commands. It is well suited for occupancy-grid environments that are the most popular representation of a map. In such settings, imperfection of commonly used sensors, such as a laser range finder, can be easily addressed in the method. The method is fast enough for real-time implementation because the number of agents does not affect the computation. (Yet, some of the DWA-based approaches [95] considering dynamic obstacles require additional computation along with an increasing number of moving objects.) However, most of the DWA-based methods are position-based approaches since they use the current geometric information of surroundings. Although the methods are capable of moving obstacles to some extent, the position-based property makes them not suitable for fast-moving obstacles.

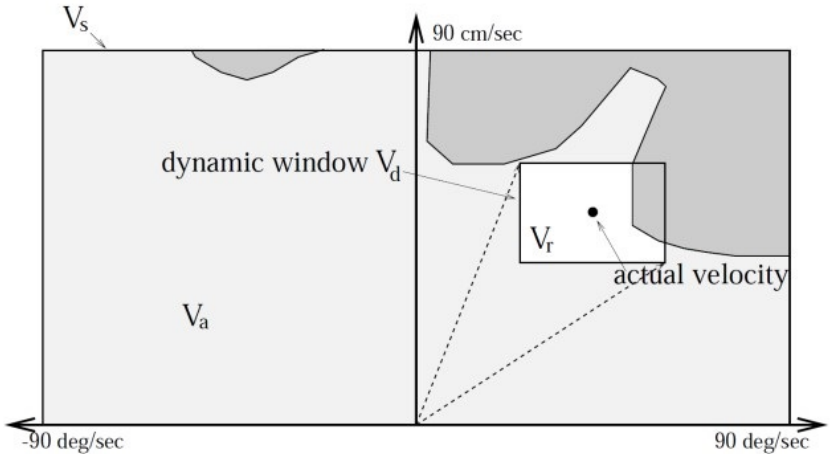


Figure 2.6: Dynamic Window Approach [30]

2.2.2 Force-Based Approaches

Force-based approaches use different types of forces related to surroundings and their sum for the net force. The movement of the robot depends on the net force.

Social Force Model

Inspired by the fact that humans can efficiently navigate through other dynamic objects, pedestrian models have been studied for mobile robot navigation. Social Force Model (SFM) [40, 26, 39, 45] is a widely used model that describes motivation of pedestrian

navigation. The model was designed to include three types of effects; acceleration toward the desired velocity, certain distance to be kept from other pedestrians and borders, and attractive effects. In the original work for SFM[40], Social Force is defined as a rate of the preferred velocity, which can be considered as a driving force. Attractive and repulsive effects are used to create the force. There are some remarks on the repulsive effects. Repulsive effects are designed as monotonically decreasing potential from the positions of either other pedestrians or borders. The one caused by pedestrians has a directional effect by applying elliptical distance from an approaching pedestrian so that the repulsive effect can be stronger in the moving direction. Non-perceivable pedestrian, who is behind, is neglected.

Although SFM is very similar to the PF, it has some differences. The attractive model in PF was created by a goal and causes some issues such as local minima and vanishing gradient of potential near the goal. SFM relieves those issues by designing the attractive effects to follow the predetermined route using the shortest path scheme. However, like many reaction-based methods, oscillatory behaviors of the robot have been observed in real applications[26], and proper tuning of the parameters is required to avoid oscillations as shown in [54].

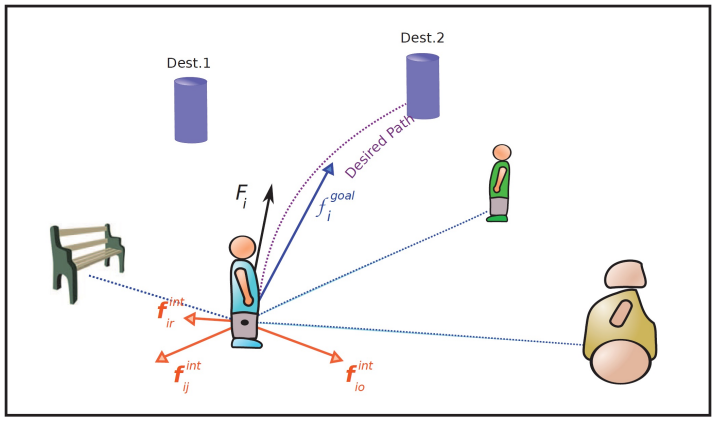


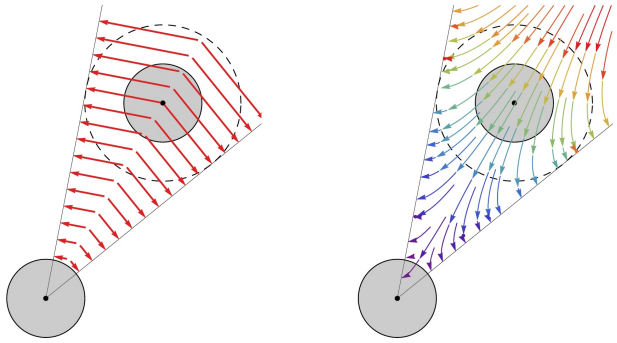
Figure 2.7: Force-based Approaches[26]

Time-to-Collision-Based Interactive Force

Inspired by Power Law[47], another pedestrian model, a collision-avoidance method based on Time-to-Collision (TTC) was suggested[29]. TTC is defined as the expected time to collide with an obstacle when the current velocity is held. Power Law models the interactive

potential between pedestrians using the inverse of TTC, which can be used for generating the interactive force from the spatial gradient of the potential.

The TTC concept and corresponding interactive force were exploited for robot navigation in [29]. It is interesting to see the comparison of this force-based method with VO because the way of calculating TTC generates interactive force only inside the area of VO. If no collision is expected by selecting a velocity outside VO, TTC is set to infinity, and the interaction force becomes zero. This is illustrated in 2.8. A benefit of TTC over the VO is that the method provides flexible and continuous changes of the velocity even if it is in the cone, whereas VO requires instant changes of velocity. Safety with sensor uncertainty remains an issue to be addressed.



(a) VO (b) TTC

Figure 2.8: Comparison of velocity changes in the VO cone using VO and TTC[29]

2.2.3 Direction-Search-Based Approaches

Instead of the direct search for the velocity space, changes of heading could be one way of avoiding collisions. This approach includes Vector-Field Histogram [13, 106], Directive Circle[73], and Human-Like navigation [37, 36]. We only present the latter two, which are more recent.

Directive Circle

Directive Circle (DC) is another sensor-based method which [73] specifies all admissible and prohibited directions of a differential-wheeled mobile robot. Once the DC against

a moving object is constructed, the best feasible direction can be selected. Figure 2.9 illustrates the construction of DC by showing two examples. The robot is modeled as a circle (light blue), and a moving obstacle is modeled as a polygon (shown in green). The velocity of the moving object is not assumed to be known, but it can be simply computed by the difference between the previous and the current measurements and is assumed to be constant (V_{O_i}). A collision cone can be drawn from the robot's position with two tangents (λ_l and λ_r) to the polygon. Then a circle with the radius of the maximum velocity of the robot can be conceived (not shown). This circle can be translated to the opposite direction by the same amount of velocity of the moving obstacle (i.e.$-V_{O_i}$ such that it is centered at P), forming so-called Velocity Circle (VC, dashed). Intersections of the two tangents of the collision cone ((λ_l and λ_r)) and the VC (dashed) provide the robot with the clue for the forbidden directions against the object, and consequently DC (the upper-left circle of each example, where the dark area is the prohibited direction) is formed. With multiple objects, either dynamic or static, the forbidden direction can be constructed and overlaid together, providing collision-free direction with the robot.

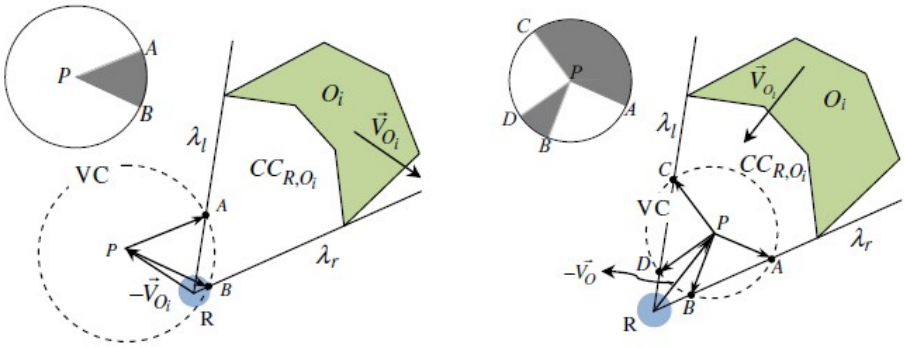


Figure 2.9: Directive Circle

Human-like Directional Search

In [37], the authors exploited a heuristic behavior of pedestrians based on the results of [79] for the robot navigation. The behavior observed in the study can be summarized as a directional search of a pedestrian during his navigation in a certain range of distance. By emphasizing the behavior, a cognitive function is defined with respect to the angle of sight within a finite field of view as illustrated in Fig. 2.10. The cognitive function ($f_o(\alpha)$) captures the expected distance that the robot can travel before collision along with the angle (α). Moving obstacles such as other pedestrians modeled as circles can be observed

by the robot in the lines of sight and are assumed to have constant velocity in the function ($f_o(\alpha)$). The desired direction of movement of the robot is selected as a direction that provides the shortest distance ($d(s(\alpha_{des}), \overrightarrow{O})$) from the destination ($\overrightarrow{O}$).

This approach drives the robot to move towards the direction that brings it as close as possible to the target before colliding with an obstacle. As it tries to mimic human cognitive behavior of navigation through walking, they named this as Human-like navigation in their following work [36], which addresses more details of the performance compared to the VO variants.

The strength of direction-searching approaches is the reduced space for searching directions to select an optimal control. For VO and DWA, the searching space is a two-dimensional velocity space, and optimizing the control value (i.e., the velocity vector) may take a fair amount of time. On the other hand, in the directive search approaches, the searching space for the optimal control value is reduced to one dimension: the heading of the robot. Directional change is also observed in human pedestrians, showing the preference of changing heading instead of speed [112], making these approaches adaptable. They do not consider the interaction between the robot and the dynamic obstacles.

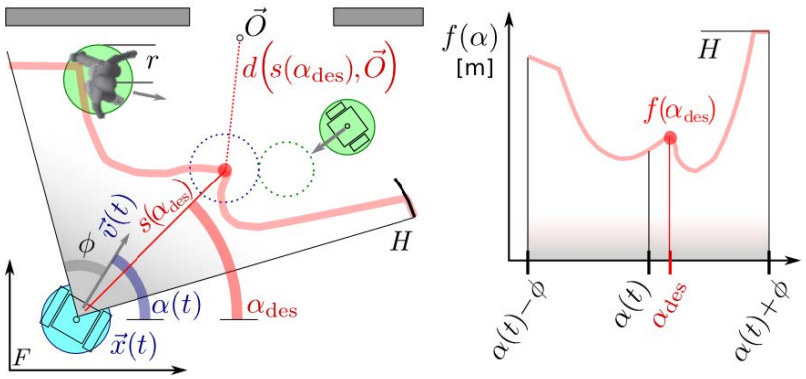


Figure 2.10: Human-like directive search

2.3 Local Motion Planning: Trajectory-Based

Trajectory-based approaches can be categorized as the methods that use trajectories of moving obstacles. In this review, we categorize them into three cases; approaches with known trajectories, cooperative trajectories based on models, and cooperative trajectories based on learning.

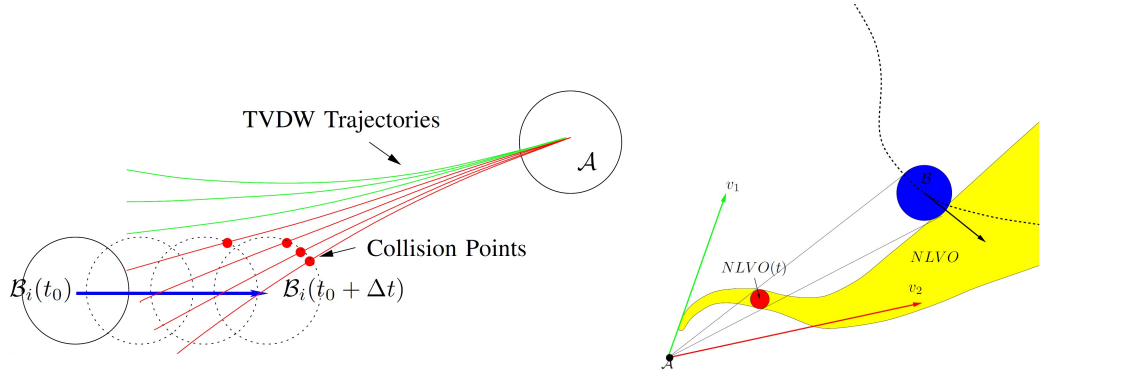
2.3.1 Approaches with Known Trajectories

Time-varying Dynamic Window

Time-varying DWA [95] extends DWA [30] to dynamic objects. In this method, a moving object is described as a set of moving cells in an occupancy-grid map, and trajectories of the cells for the next finite time steps are computed by assuming constant translational and rotational velocities. The extent of the time-horizon is set to equal to the brake time that the robot can make a complete stop before the collision. Then the computed trajectories of the moving cells are checked for collision with the robot's trajectories as shown in Fig. 2.11(a). TVDW is superior to DW because it considers the future behavior of the obstacles, but it is computationally intensive and limited to short-time prediction.

Non-linear Velocity Obstacle

Assuming known arbitrary velocity profiles of the other objects, VO can be extended to Non-linear Velocity Obstacle (NLVO)[97, 59]. Consideration of the arbitrary trajectories makes the geometric shape of the NLVO cone become a warped cone as shown in Fig. 2.11(b). Although the method does not assume constant velocity, it assumes all velocity profiles of others to be known, which is an assumption not valid in general. Cooperative behavior, which turns out to be essential, is hardly combined with NLVO.



(a) Time Varying Dynamic Window(TVDW) (b) Non-linear Velocity Obstacle

Figure 2.11: NLVO and TVDW[72]

Inevitable Collision State

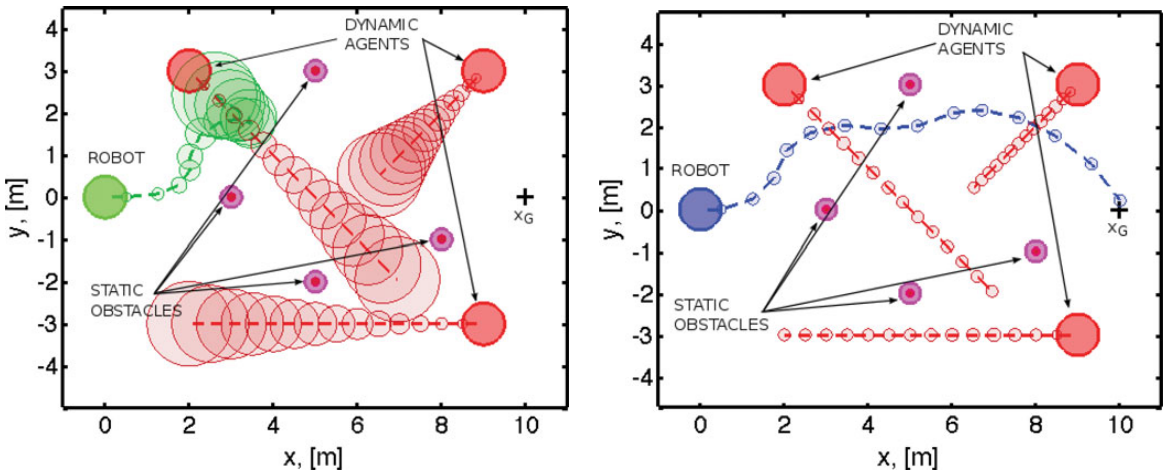
With known states of other objects in the future, Inevitable Collision State (ICS) is suggested [31] to check if a set of inputs of a robotic system collides with the states of other objects. ICS is a state for which a collision eventually occurs regardless of the future trajectory of the robot. The strategy of the ICS-based planning is finding a set of control commands that do not drive the robot states into ICS [71, 72], which is a technique named ICS-AVOID. Input trajectories are ruled out if they have ICS at any state of the trajectory. The concept was extended to the Probabilistic Collision States (PCS)[8] to deal with uncertainties.

Although the method outperforms TVDW and NLVO [8], an infinite number of input trajectories and a limited time horizon can cause implementation issues. The infinite number of trajectories can be solved by computing only a finite subset of input trajectories, but it still requires heavy computation.

Receding Horizon Control Based Approaches

Receding Horizon Control (RHC) (a.k.a. Model Predictive Control, or simply MPC) approaches have become a popular tool of motion planning when the trajectories of others are taken into account. One of the benefits of MPC is that the constraints, either linear or nonlinear, can be explicitly formulated in the problem. Predicted states (i.e., trajectories) of the obstacles can be formulated as the inequality constraints in the RHC framework as a nonlinear optimization.

Dynamic environments are often stochastic due to uncertainties, and thus stochastic RHC has been proposed to deal with the imperfect measurement. Examples of stochastic RHC include Open-loop RHC (OLRHC) [110] and Partially Closed-Loop RHC (PCLRHC) [23]. In those approaches, a belief, probabilistic state estimation, is included in the problem formulations by applying measurement. The difference between OLRHC and PCLRHC lies in the range of measurement used in the models. OLRHC only uses the past measurement from the current, whereas the PCLRHC exploits anticipated future measurement. Assuming Gaussian noise for measurement and using Kalman filter for belief transition, PCLRHC provides a less conservative path due to the managed uncertainties. On the other hand, OLRHC has growing uncertainties and tends to be more conservative when choosing the optimal trajectory, potentially leading to failures to reach the goal. Figure 2.12 describes this. Computational cost is large in PCLRHC because the additional update of covariance of measurement is required for the prediction update step, which involves matrix inversion and multiplication.



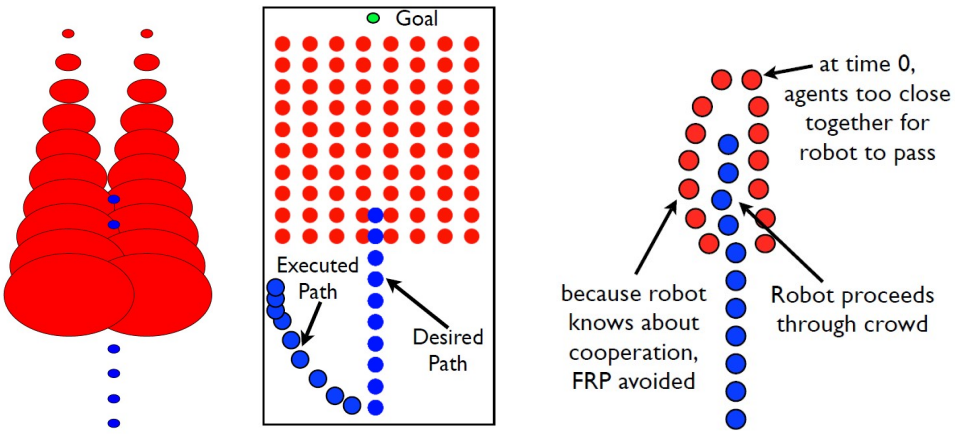
(a) Open-loop Receding Horizon Conrol (b) Partially Closed-loop Receding Horizon Conrol

Figure 2.12: Stochastic Receding Horizon Control[23]

2.3.2 Cooperative Trajectory Approaches: Model Based

Interacting Gaussian Processes (IGP)

In [104], the authors emphasized the importance of interaction between the robot and dynamic obstacles for efficient robot navigation, introducing the "freezing" robot problem (FRP). Figures 2.13(a) and 2.13(b) illustrate FRP with possible scenarios. By expecting the cooperation of others, the robot can move into the crowd, and thus FRP can be resolved as shown in Fig. 2.13(c). Accordingly, they modeled the interaction by considering the joint probability of the trajectories of all moving objects at once. Gaussian process was used to describe this independent, arbitrary, and supposedly smooth trajectory. All trajectories were predicted in an interactive way by defining an interaction potential between each of the trajectories and solving the GPs. The factors of interaction potential function include affordance, progress, and penalty to close distances throughout the entire trajectory. They extended this work to stochastic characteristics of the trajectories by considering multiple possible goals of each object[103].



(a) Freezing with increasing uncertainty (b) Freezing in crowded environment (c) Mitigation of the Freezing with expecting interaction

Figure 2.13: Illustration of freezing problem in [104]

2.3.3 Cooperative Trajectory Approaches: Learning-based

Feature-Based Interaction Learning

With hopes that proper interactive behaviors can be obtained from human beings, experiment or pedestrian data set have been used to learn the interaction model. To mimic human's interactive behavior, feature-based learning approaches have been proposed [56, 55]. In those works, a model of human cooperative navigation behavior, $\Phi(\mathrm{x}) = \sum_{k=1}^{p} w_k \phi_k(\mathrm{x})$, was introduced with hand-designed features $\phi_k(\mathrm{x})$, where x is a state, p is the number of features and w_k is the corresponding weights. The features were aimed to capture the characteristics of pedestrian behavior studied in the literature, such as minimum time to reach, minimized fluctuating acceleration, the tendency of preferred speed, and clearance to other agents. The weights w_k were intended to be learned through inverse reinforcement learning (IRL) with maximum entropy using pedestrian data. In the learned model, a prediction of the pedestrians' behavior and the desired behavior of the robot itself is encoded in the most likely interaction behavior. The learned parameters were implemented and compared to human trajectories and used for trajectory prediction.

Deep Reinforcement Learning

As to overcome computational intractability for trajectory prediction, a framework of using deep reinforcement learning (DRL) was suggested for collision avoidance between dynamic objects[17]. The basic idea is offloading the computational burden for joint trajectory prediction by learning the interaction rule through the offline DRL step. The problem of collision avoidance was formulated as a Partially Observable Markov Decision Process (POMDP) with joint states of the robot and moving object. With hand-designed reward functions, either the value function[17] or the policy[16, 24] can be trained by DRL through the simulation with ORCA[108]-based moving obstacles in order to implicitly encode the interaction-rule. The learned interaction can be exploited for the online implementation step. The interaction rule is implicitly embedded in the learned value function, and the robot follows the command that takes the maximum value. In their later work in [16], a different reward function was defined considering social-norms such as left/right-side-passing-rule, and policy was trained through multiple agents scenarios (4 or fewer). One further extension addresses scenarios of the larger number of agents and relieves dynamic constraints[24].

Although these methods mitigate computational burden for predicting the interaction and corresponding trajectory, the number of the moving obstacles is limited [17, 16] due to

the fixed input dimension of deep learning. Moreover, the validity of the learned interaction rule may be weak because they train the rule through simulation while dynamic objects follow ORCA policy[108], which may not hold in the real world.

2.4 Conclusions

2.4.1 Challenges and Issues

Interaction between objects

Dynamic objects in the real world are mostly active, i.e., they decide their actions in consideration of surroundings and the movement of other agents. Dealing with the interaction between the objects and surroundings is essential for efficient robot navigation since the mobile robot without such consideration will often show awkward behaviors, such as oscillation and freezing. In all studies reviewed in this report, methods dealing with the interaction fell into those categories: simply assuming reciprocal behavior, reasoning over physics on the interaction (e.g., pedestrian), and/or learning the interaction rule. However, existing methods have limitations. The reciprocal assumption can only be valid for dynamic objects that follow the same action policy, such as multi-robot scenarios. Reasoning over specified scenarios such as pedestrian interactions is able to capture the interactive behaviors, although predicting others' movement based on this reasoning requires a high computational cost. The interaction is inherently stochastic, which causes even heavier computation when it comes to robot navigation. Learning approaches to that behavior may sound prominent, but they may also suffer from the stochastic behaviors of individual agents. In conclusion, a feasible approach is needed to handle the interaction in a more efficient way.

Computational Tractability vs Efficiency of Path

Undoubtedly, the computational load is one of the critical issues when considering real-time operations, and thus the reaction-based methods have been popular due to their advantage of the low computational cost compared to trajectory-based methods. As stated in a number of studies [16, 94], however, trajectory-based approaches with good prediction can provide much more efficient trajectories no matter how the efficiency is defined. Recent trends of the research show that many researchers have been trying to solve this dilemma using machine learning techniques such as reinforcement learning (RL). The main idea

of these approaches is to reduce computational cost on the trajectory prediction through offline training steps before the online implementation. However, the actual situation the robot faces is often different from the training in terms of different behaviors and numbers of the agents, which may weaken the validity of the approaches.

Guaranteed Safety and Efficiency

Safety versus efficiency is the primary trade-off in the problem of planning. This is because, in extreme cases, safety can be guaranteed if the robot does not move at all. On the other hand, efficiency can be maximized if the robot aggressively moves to the target, which is not the case we want. This may be exaggerated, but many of the trajectory-based methods suffer from the trade-off dilemma by showing too conservative planning results caused by increasing uncertainties along with the time in the prediction. This may result in even more degraded efficiency showing awkward trajectory and freezing [104]. All trajectory-based approaches we reviewed are not able to guarantee safety. In some reaction-based methods, safety can be explicitly guaranteed by giving a collision-free motion. Yet, the methods generally suffer from their myopic scope of the planning, often causing inefficient path[17]. Guaranteed safety needs to be addressed with efficiency to prevent the planner from offering too conservative control input, such as stop or stuck.

Agent-level and Sensor-level Measurement

Many of the existing works, such as ORCA, assume measurement at the agent-level due to its benefits, such as reduced redundancy of information and computational burden [24]. This assumption requires a perfect perception of dynamic obstacles. None of the abovementioned studies assuming known data in agent-level deals with the perception and tracking failure. We will need to consider it for the implementation because even the state-of-the-art computer vision systems cannot provide robustly acceptable perception[94].

Chapter 3

Global Path Planning Based on Safefy-Aware Navigation Speed and Hierarchical Topology Map

3.1 Introduction

Finding a safe and efficient global path is a critical part for mobile navigation. While extended studies on the global path planning have been successfully implemented as we reviewed in the previous chapter, there still exist issues for handling computation time and safety hwhen a map becomes large and complex. Still, there is a challenge in handling a safe and efficient path with lower computational cost as the environments for mobile robots are becoming larger and more complex. Recalling the most popular type of maps is the OGM, the computation time of approaches with those maps becomes critical due to the rapidly increasing size of the environment (e.g., quadratic for 2-d OGM). Thus, many other approaches for global path planning have also been extensively studied to mitigate the computational cost by converting OGMs into graphs to reduce the searching space for a global path, such as Probabilistic Road Maps (PRMs, [49, 46]), Rapidly-exploring Random Trees (RRTs, [62, 57]), or space skeletonization ([11, 34]).

Among the methods that use sparse graphs of an OGM, skeleton-based approaches ([101, 11, 34, 111]) can be advantageous for planning a safe path, because the skeleton of a map is a set of equidistant points from obstacles (e.g., corridor walls) in a map. In other words, its points have the largest clearance (thus safest) from their obstacles. Another benefit of skeleton-based approaches is that such methods are complete ([18, 84]), meaning

that it is guaranteed to find a feasible path if it exists, as the skeleton is the abstract expression of the free space of a map. However, the main drawback of such skeleton-based approaches is that the resulting path is often far from optimal ([11]), as the primary output of those methods is a part of the skeleton that usually has unnecessary turns and too conservative waypoints. This is particularly true when a path is obtained merely from the skeleton. Thus, the refinement of a skeleton path is essential to avoid unnecessary bypasses and sharp turns and to get the optimal path while ensuring safety.

One way to refine the skeleton path into an optimal path is to reduce the number of edges of the skeleton path ([12]). They iteratively refined the skeleton path by the corner-cutting technique that incorporates adjacent two edges into a single edge if the single edge can keep the (customizable) minimum clearance from obstacles. The iterative removal of the edges results in the path with the minimum number of edges that becomes the shortest path with the minimum clearance. However, their method requires the use of the original map to check the minimum clearance throughout the corner-cutting technique, which may degrade their computation efficiency with large maps.

Explicit Corridor ([34, 111]) is another skeleton-based method that exploits the skeleton of a map and its clearances to create a corridor map along with the skeleton. For path planning, they first obtained a path from the skeleton and then created the corridor map to reduce the searching area. In the corridor map, they created another graph by sectionalizing the corridor map with triangulation that has its vertices on the boundary. Then the shortest path can be obtained by connecting some vertices on the boundary from the start to the goal. The corridor map can be shrunk with the minimum clearance for safety. This method may be beneficial for large maps because it does not use the original map once the corridor map is created. However, it has not been implemented with a real robot as the method was intended for the virtual world (i.e., polygon world).

More recent works ([115, 21]) combine the skeleton-based approaches and sampling-based approaches in such a way that the skeleton narrows down the sampling space. They planned a skeleton path first, then refined the path using probabilistic approaches such as RRT. The main benefit of those methods is the computation efficiency because of the reduced size of samples near the skeleton. Yet, they did not generate an optimal path showing excessive distances from the wall ([21]) or lacked concerns of safety which is one of the main benefits of the skeleton-based approaches.

While a skeleton path is refined with different approaches as mentioned above, the minimum clearance is commonly used for a safe path in many works ([11, 34]). With ensuring the minimum clearance for the means of safety, the shortest path criterion is often used for an efficient (or optimal) path with the hope that it would provide the least

time to reach the goal. However, the minimum clearance needs to be large when operating a mobile robot at high speeds because there are increased chances for collision with corridor walls. Yet, enlarging the minimum clearance for safer operation may cause other issues, such as the elimination of narrow areas of a map that were once accessible with the smaller minimum clearance if the robot moves slow. This implies that there should be some compromise of the operation speed with clearances to obstacles.

In this chapter, we propose a complete global path planning method with a skeleton-based graph structure using the Hierarchical Topology Map (HTM). HTM is a hierarchical map expression that consists of a topology graph and metric points of a skeleton at different layers. In HTM, the relation between the topology graph and the skeleton is marked in the topology graph in a new tuple of edge expression. The cost of travel along the skeleton is also marked with the accumulated cost at the topology. Then, a path can be obtained from HTM first by planning on its topology graph (higher layer) based on the accumulated cost and then by tabulating the corresponding metric points on the skeleton (lower layer). This provides an on-skeleton path (i.e., a path on the skeleton).

The main idea of the work in this chapter is the way of refining a skeleton path to an optimal (off-skeleton) path. Inspired by the concept of the Explicit Corridor ([34]), we add the boundary points of each skeleton point into HTM to define an extended HTM named as the Hierarchical Topology Map with Explicit Corridor (HTM-EC). Once an on-skeleton path is obtained in HTM, a set of points is created such that the boundary points of HTM-EC discretize the corridor space along the previously obtained on-skeleton path and the on-skeleton path is optimized into a metric, off-skeleton in the discretized space by using dynamic programming. We assume a map is given in advance, and one-time construction of HTM-EC is performed as a part of initialization when a mobile robot boots up: the HTM-EC is stored in the memory, and path planning is performed only on HTM-EC instead of conventional maps such as an occupancy grid.

Another core idea of our work is the adoption of the allowable speed in the problem of global path planning for time-minimization as the objective. The allowable speed limits navigation speed depending on the clearance to obstacles for safety. Unlike other global path planning methods that use the shortest distance with the minimum clearance as the means of objective and safety, respectively, the allowable speed enables us to get a time-minimized path as the optimal path while safety is incorporated in it.

We summarize the main contributions of our work as follows.

1) introduction of HTM-EC and the associated minimum-cost path planning with reduced size of searching space for fast computation;

2) refinement method of a skeleton-path based on cost optimization with the dynamic programming;

3) robot implementation of the allowable speed of navigation into the cost function for a time-minimizing global path;

The effectiveness of our HTM-EC as a new efficient and safe path planning method has been verified by simulation and experimental results using several real maps.

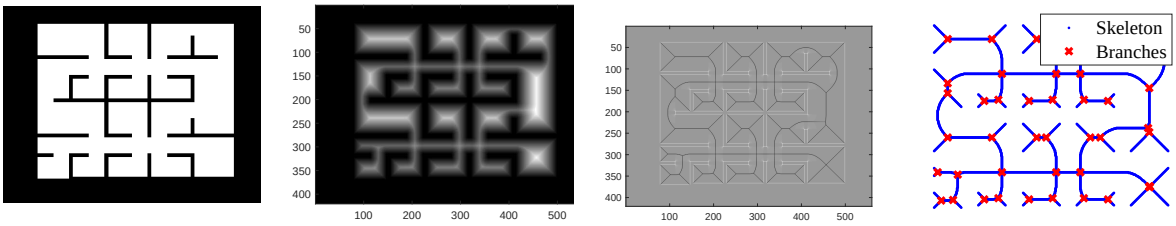
3.2 Proposed Method for Global Path Planning

3.2.1 Map Representation Using Sekelton

Skeleton is a morphological expression of a map with a set of equidistant points to the nearest obstacles, which can be obtained from various methods such as Voronoi Diagram ([81]), image thinning ([122]), or distance fields transform ([116]). In this study, we use a distance-field-based method combined with thinning that is similarly proposed in [116]. We found that this method is advantageous over other methods in that the resulting skeleton is less sensitive to the noise of a map and connected skeleton is guaranteed while it provides us with equidistance from the obstacles of a map. The method can be described as follows using a simple map shown in Fig. 3.1(a). Firstly, a map image is converted to a binary image such that zeros and ones are assigned to the occupied cells and the free cells, respectively. The values of distance from boundaries of the free cells can be calculated and the distance field map is obtained as shown in Fig. 3.1(b). The ridges can be detected as shown Fig. 3.1(c) by taking second derivatives over the space and applying a threshold to the values. The detected points of ridges look like a skeleton drawn in the middle of the free space, but they may have more than one pixel in the lateral direction of the skeleton. Finally, image thinning is applied to thin the ridges, resulting in the skeleton and its branch points as shown in Fig. 3.1(d). All the points on this skeleton will be denoted by

$$\mathcal{P}^s := \{p^s(1), p^s(2), \cdots, p^s(N)\} \tag{3.1}$$

where $p^s(k) = (x^s(k), y^s(k)) \in \mathbb{R}^2$ and N is the total number of points in the skeleton. Namely, $\mathcal{P}^s$ is a set of disordered points of the skeleton without any topological information (or graph structure). We will apply pruning and reordering to the points in $\mathcal{P}^s$, which will result in an ordered skeleton point $q(k)$ and its set $\mathcal{Q}$, to construct the hierarchical graph representation and the Hierarchical Topology Map in Section 3.2.2.



(a) Original map (b) Distance field map from boundary (DFB) (c) Ridges of DFB (d) Skeleton with branches obtained from image thinning

Figure 3.1: Illustration of the skeletonization process

3.2.2 Hierarchical Topology Map (HTM)

The Hierarchical Topology Map (HTM) is a map expression that consists of two layers in a hierarchy: the skeleton of a map (lower layer) and the topology graph of the skeleton (higher layer). The topology graph of HTM can be obtained from the skeleton by extracting nodes from the intersection or terminal points of the skeleton and edges from the nodes' connectivity. Figure 3.2(a) visualizes an example of the HTM over a simple map, where the consecutive empty gray circles are the skeleton points, $q(k)$, the empty blue circles are the nodes, N_i, and the blue dashes are the edges, E_{ij}.

When the topology graph is constructed, the skeleton points are reordered such that adjacent points have consecutive indices. The reordering process is achieved by exploring all points in the skeleton, starting from the point with the maximum clearance, to check whether a skeleton point is terminal, intersecting with other skeleton points, or merging to a previously explored intersecting point. When a skeleton point falls into one of these three categories, it is marked as a node creating an edge. For each skeleton point being explored, its adjacent skeleton points can be found by using the window of eight neighboring cells (i.e., 3×3 window).

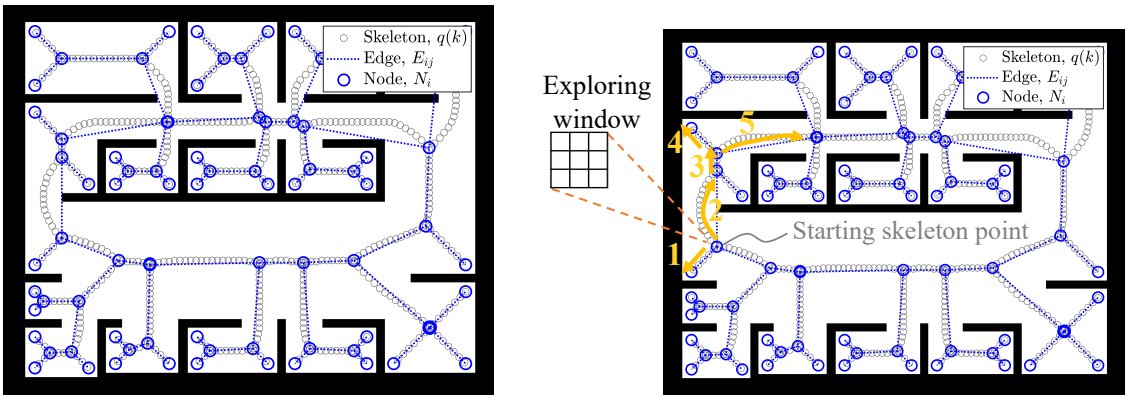
Let us detail such processes with an example for the exploration along with the skeleton shown in Fig. 3.2(b). The yellow arrows in the figure visualize the orders of the exploration up to the fifth, beginning from the starting skeleton point. The starting point has three skeleton points in its neighboring cells as there are three skeleton branches at the point. These three skeleton points are added to the exploration queue, and one of them is selected for the next point to be propagated and explored. Such propagation leads to the addition of newly found skeleton points to the queue and the removal of the currently exploring

point. When the terminal point is found after the first exploration, we can jump back to the starting point and continue the reordering process to the next branch (let say it is upside as marked 2). This is because that the remaining points in the queue are the two neighboring skeleton points of the starting point. The same situation occurs when a terminal point is found at the end of the fourth exploration (the arrow marked 4) and allows the next exploration as directed by the fifth arrow. Similarly, we can jump to the most recently added point in the queue whenever terminal or merging points are found and continue the reordering process until the queue becomes empty. Algorithm 1 in Appendix B explains these processes. The particular example of Fig. 3.2 has 2,934 skeleton points which generated 85 nodes and 86 edges.

After reordering, we will denote the k-th skeleton point by $q(k)$. If a skeleton point is either terminal or intersecting, the point is marked as a node and its connection to the previously found node is checked to form an edge. For each edge found, we keep track of the explored skeleton points in order. Thus, for each edge, denoted by E_{ij}, we associate four elements as

$$E_{ij} := (N_i, N_j, I_{ij}, C_{ij}) \tag{3.2}$$

where N_i and N_j are the nodes for the edge, I_{ij} is the set for indices containing all the skeleton points within the edge E_{ij}, and C_{ij} is the cost of the edge. The computation of C_{ij} will be explained later (by Eq. (3.11)).



(a) Example of HTM over a simple map

(b) Example of the reordering process to construct the topology graph from the skeleton

Figure 3.2: Illustration of the Hierarchical Topology Map and its construction from the skeleton of a map

Using the above definition of the HTM, the topology graph can be defined as an

weighted undirected graph which is denoted by $G(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is that of edges. For each E_{ij}, the information on the associated skeleton points is specified through the index set I_{ij}. By denoting $\mathcal{Q}$ the set of the ordered skeleton points q (for the rest of the paper, the term *skeleton* will be considered the *ordered skeleton*), the HTM, denoted by M_{ht}, can be defined as the combination of the topology graph, G, and the skeleton, $\mathcal{Q}$ as follows.

$$M_{ht} := \{G, \mathcal{Q}\} \tag{3.3}$$

The total number of skeleton points and that of nodes in the HTM will be denoted by n_Q and n_N, respectively.

3.2.3 HTM-EC: Hierarchical Topology Map with Exciplicit Corridor

Due to the ways in which the skeleton points are constructed, any skeleton point, $q(k) \in \mathcal{Q}$ where $k \in \{1, 2, 3, \cdots, n_Q\}$, has at least two closest points to the obstacles (e.g., walls). Those points were used in defining Explicit Corridor with a tuple of four elements: a skeleton point, the clearance at the skeleton point (i.e., distance to the nearest obstacle), the left closest point and the right closest point from the skeleton point. We apply Explicit Corridor to a skeleton point of $q(k)$, resulting in a tuple as

$$b(k) := \{q(k), r(k), O(k)\} \tag{3.4}$$

where $r(k)$ is the distance from the skeleton point $q(k)$ to the nearest obstacles, and $O(k)$ is the set of the closest obstacle positions. More formally, $O(k) := \{o_m(k) | m \in \{1, 2, \cdots, n_o(k)\}\}$, where $n_o(k)$ is the number of the closest obstacle points from $q(k)$. The size of $O(k)$ can vary depending on the number of closest obstacles at $q(k)$. Note $q(k) \in \mathbb{R}^2$ (or the position in 2D) and $r(k) \in \mathbb{R}$.

$o_m(k) \in O(k)$ can be found when a skeleton point $q(k)$ is constructed into the HTM. Since the distance to the closest obstacles $r(k)$ is given for each $q(k)$, we can check out if the obstacle cells of the original map are matched with the distance, that is

$$o_m(k) = \{o(x, y) \mid |o(x, y) - q(k)| = r(k)\} \tag{3.5}$$

where $o(x, y)$ is a point on obstacles at (x, y). In practice with a grid map where each cell of obstacles has integer indices (x, y), the closest obstacle points $o_m(k)$ may not have exact $r(k)$ with a cell size. To incorporate such bias, we used a modified equation with a scaled

cell size to get $o_m(k)$ in a grid map, that is

$$o_m(k) = \{o(x,y) \mid |o(x,y) - q(k)| \in [r(k) - r_g, r(k) + r_g]\} \tag{3.6}$$

where r_g is the scaled size of each grid cell into the same unit as $r(k)$, e.g., in meter.

Now, let us denote $\mathcal{B}$ the set of $b(k)$ for all skeleton points, i.e., $\mathcal{B} := \{b(k)|k \in \{1, 2, 3, \cdots, n_Q\}$. Then we can define Hierarchical Topology Map with Explicit Corridor, denoted by M_{ht}^{ec}, as the set of G and $\mathcal{B}$

$$M_{ht}^{ec} := \{G, \mathcal{B}\}. \tag{3.7}$$

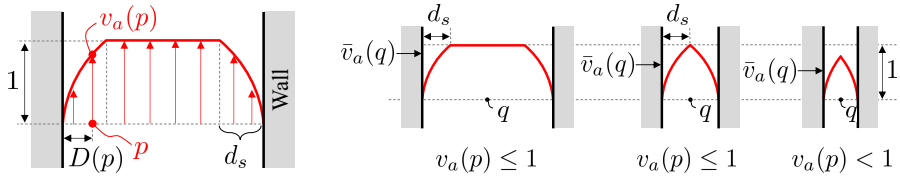
3.2.4 Formulation of Cost Function

In this section, we introduce the formulation of each cost $c(q(k))$. Obviously, a mobile robot needs to operate at reduced speeds when there are close obstacles in the surroundings, such as walls in narrow corridors. It is natural to consider the speed reduction into the cost function, as it can affect the time of traveling for the robot. Accordingly, we introduce the *allowable speed* of navigation as the maximum speed allowed to a mobile robot such that it can make a complete stop safely before colliding with the closest obstacle under the maximum deceleration rate a_{max}. Specifically, the allowable speed of navigation $v_a(p)$ at a point p is given by the relative speed normalized by v_{max} (i.e., the maximum speed of the robot in the absence of any obstacles) as

$$v_a(p) := \begin{cases} \frac{\sqrt{2a_{max}D(p)}}{v_{max}}, & \text{if } D(p) < d_s \\ 1, & \text{otherwise} \end{cases} \tag{3.8}$$

where $D(p)$ is the distance to the closest obstacle at p, and $d_s = \frac{v_{max}^2}{2a_{max}}$ is the safe distance with the kinematic constraints of the robot. Note that $v_a(p)$ has its value in $[0, 1]$ where the value of 1 corresponds to v_{max}. Also, note that when p is a skeleton point, (i.e. $p = q(k)$), then $D(p) = r(k)$. The shape of $v_a(p)$ across a corridor can be shown as Fig. 3.3(a) and the shape depends on the width of the corridor as shown in Fig. 3.3(b).

When a mobile robot moves through a corridor cross-section of a skeleton point $q(k)$, the actual point that the robot will pass varies across the corridor depending on the given path. Thus, it will be natural to use a representative value that can capture how easy it is to navigate passing through the corridor cross-section. To quantify such ease of navigation



(a) Allowable speed, $v_a(p)$ (b) $v_a(p)$ shape with different width of corridors

Figure 3.3: Allowable speed of navigation

at a skeleton point $q(k)$, we can average the *allowable speed* across its width as

$$\bar{v}_a(q(k)) := \frac{1}{D(q(k))} \int_0^{D(q(k))} v_a(p)dp \tag{3.9}$$

where $\bar{v}_a(q(k))$ represents the mean value of the achievable maximum speeds across the cross section of the corridor at the skeleton point $q(k)$. We define this quantity as the cost value at the skeleton point $q(k)$ as

$$c(q(k)) := \frac{2\|q(k+1) - q(k)\|_2}{\bar{v}_a(q(k+1)) + \bar{v}_a(q(k))} \tag{3.10}$$

where the cost, $c(q(k))$, can be regarded as the averaged time expectation of traveling in the corridor enclosed by $q(k)$ and $q(k+1)$. The edge cost C_{ij} for en edge E_{ij} in Eq.(3.2) can be obtained by summing up all the $c(q(k))$ in the range of $k \in I_{ij}$ as follows.

$$C_{ij} := \sum_{k \in I_{ij}} c(q(k)) \tag{3.11}$$

3.2.5 Path Planning Using HTM-EC

Path planning with HTM-EC consists of three steps: insert start and goal into the HTM, perform topology planning from the topology graph and corresponding skeleton planning, and refine the skeleton path into an optimal metric path. These planning steps can be performed solely on HTM-EC, and the original grid map is not used. The following subsections address each of the planning steps.

Addition of start and goal nodes

The first step for planning a path with HTM-EC is to choose the start and the goal from the ordered skeleton set $\mathcal{Q}$ and incorporate them into the topology graph G. Assume that we are given the goal position and the start (or the current) position of the robot, which are denoted by p_G and p_R, respectively. Then, we can first identify the closest skeleton points from p_R and p_G. Let us say these two points are $q(k_R) \in \mathcal{Q}$ and $q(k_G) \in \mathcal{Q}$, respectively, where k_R and k_G are the related indices in $\mathcal{Q}$. If $q(k_R)$ and $q(k_G)$ are not in $\mathcal{N}$, they are appended to the existing nodes as N_{n_N+1} and N_{n_N+2}. This will, in turn, generate four additional edges: two for N_{n_N+1} with its two neighboring nodes, and two for N_{n_N+2} likewise. Figure 3.4 visualizes these processes with a simple example of HTM that has the topology graph G with five nodes and their edges shown in blue and associated skeleton points, $q(k)$, shown in black. With the given p_R and p_G, the two additional nodes, N_6 and N_7, and their four additional edges are shown in red.

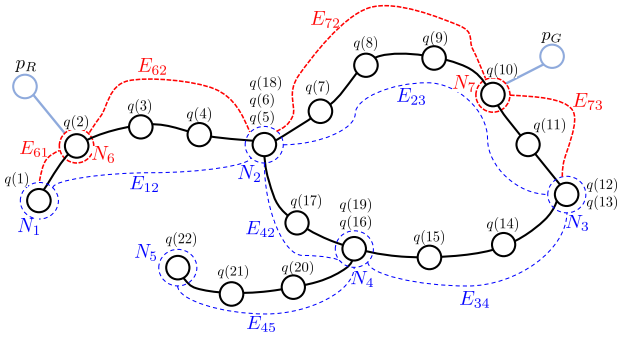


Figure 3.4: Addition of start and goal on HTM for topological path planning

Planning an on-skeleton path with HTM

As the resulting path $\mathcal{Q}_p$ is an on-skeleton path, it still needs to be refined. Specifically, the path resulting from $\mathcal{Q}_p$ may have sharp corners near the intersections of corridors and have unnecessary clearance from the walls. Thus, we need to refine $\mathcal{Q}_p$ to get an off-skeleton path by another optimization routine. The basic idea is to find the optimal amount of lateral deviation from each skeleton point. In other words, at each skeleton point $q(k)$, we can obtain the cost at any point in its traverse space to the nearest obstacles. Then the optimal path would be the one that goes from the given start and goal points and minimizes the path integral of the cost along the given skeleton path $\mathcal{Q}_p$.

Letting n_p be the total number of skeleton points of $\mathcal{Q}_p$, a skeleton point, denoted by $q_p(k) \in \mathcal{Q}_p$ where $k \in \{1, 2, \cdots, n_p\}$, immediately provides us with the corresponding tuple $b(k)$ given by Eq.(3.4). Recalling $o_m(k) \in O(k)$ in Eq.(3.4) refers to the position (x, y) of the m-th closest point from the skeleton point $q_p(k)$, we can draw a line, $\overline{q_p(k)o_m(k)}$, for each $o_m(k)$. See Fig. 3.5 for the illustration. In the figure, two skeleton points, $q_p(k)$ and $q_p(k+1)$, are given from the skeleton path $\mathcal{Q}_p$, and the lines to their closest points $o_1(k)$, $o_2(k)$, $o_1(k+1)$ and $o_2(k+1)$ are drawn. Please note that $\overline{q_p(k)o_1(k)}$ and $\overline{q_p(k)o_2(k)}$ from a skeleton point are not necessarily colinear although they look so in Fig. 3.5.

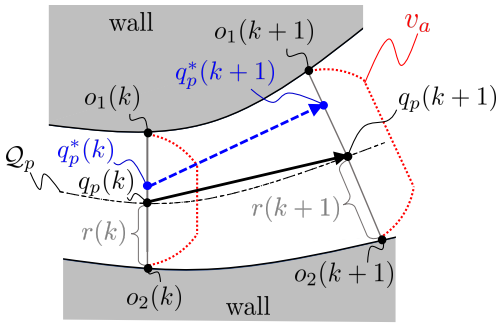


Figure 3.5: Optimization of the given skeleton path in a corridor section

One can notice in Fig. 3.5 that we drew the profile of the allowable speed, v_a, on each line of $\overline{q_p(k)o_1(k)}$ and $\overline{q_p(k)o_2(k)}$. If corridors are sufficiently wide, a robot may not need to stick to the skeleton path $\mathcal{Q}_p$ which consists of the farthest points from the obstacles. Rather, the robot can go closer to the wall to get its goal faster with some compromise between shorter paths and reduced speed, both of which can result from smaller clearances from the wall. This suggests us that we can improve the skeleton path $\mathcal{Q}_p$ by considering point, say $q_p^*(k)$ lying on lateral branch lines of $q(k)$ (i.e. $\overline{q_p(k)o_1(k)}$, $\overline{q_p(k)o_2(k)}$,...). See Fig. 3.5 for the example of $q_p^*(k)$.

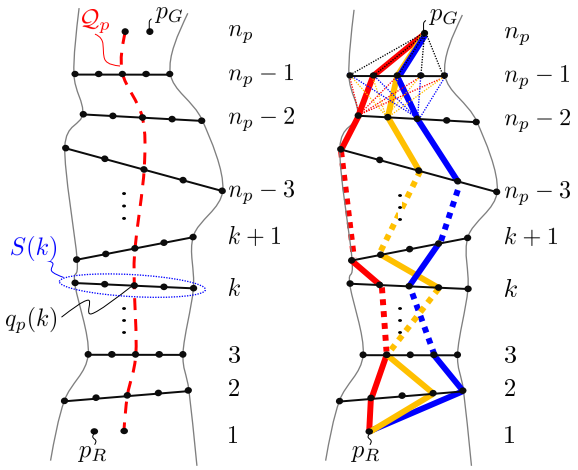
By choosing $q_p^*(k)$ properly for all $k \in \{1, 2, 3, \cdots, n_p\}$, we can obtain the optimal path that is off-skeleton, denoted by $\mathcal{Q}_p^*$, along the corridor of the on-skeleton path $\mathcal{Q}_p$. The procedures of determining $q_p^*(k)$ can be explained in a more detail by an example shown in Fig. 3.6. In Fig. 3.6(a), a corridor is drawn along by the given skeleton path $\mathcal{Q}_p$ (the red dashed line) with arbitrary widths. For each skeleton point $q_p(k)$, we drew the lateral branch lines $\overline{q_p(k)o_1(k)}$, $\overline{q_p(k)o_2(k)}$, $\overline{q_p(k+1)o_1(k+1)}$ and $\overline{q_p(k+1)o_2(k+1)}$. Let n_s be the number of segments of the line $\overline{q_p(k)o_m(k)}$ for each $m \in \{1, 2, ...\}$ and h be the order of the discretized point counted from $q_p(k)$ (e.g., $n_s = 2$ in Fig. 3.6). If these segments are equally segmented (which we assume in this paper), the h-th point on the line $\overline{q_p(k)o_m(k)}$,

denoted by $s_m(k,h)$, can be computed by Eq. (3.12). The total set of all the discretized points, denoted by $S(k)$, can be defined as Eq. (3.13).

$$s_m(k,h) = \frac{h}{n_s}(o_m(k) - q_p(k)) + q_p(k) \tag{3.12}$$

$$\begin{aligned} S(k) :=& \{q_p(k), s_1(k,1), \cdots, s_1(k,n_s), \cdots, s_m(k,1), \cdots, \\ & s_m(k,n_s), \cdots, s_{n_o(k)}(k,1), \cdots, s_{n_o(k)}(k,n_s)\} \\ :=& \{s(k,1), s(k,2), \cdots, s(k,g), \cdots, \\ & s(k, n_o(k)n_s+1)\} \end{aligned} \tag{3.13}$$

$n_o(k)$ is the number of closest obstacles as defined earlier (e.g., $n_o(k) = 2$ for all k in Fig. 3.6(a)). Please notice that in Eq. (3.13) we reformed $s_m(k,h)$ to $s(k,g)$, where $g \in \{1,2,3,\cdots,n_o(k)n_s+1\}$, such that $s(k,1) = q_p(k)$ followed by $s_m(k,h)$'s for all h and m. We can obtain $S(k)$ for all skeleton points $q_p(k)$, which can be regarded as the candidates for $q_p^*(k)$. For the start and goal points, p_R and p_G, we can set $S(1) = \{s(1,1)\} = \{p_R\}$ and $S(n_p) = \{s(n_p,1)\} = \{p_G\}$ since the two points must be on the final path.



(a) Discretization of space along skeleton path (b) Optimization of skeleton path

Figure 3.6: Discretization of corridors and path optimization using dynamic programming

Since the corridor is discretized as the set of $S(k)$ for all $k \in \{1,2,...,n_p\}$, we can consider $s(k,g) \in S(k)$ and $s(k+1,g') \in S(k+1)$ for some g and g' as the candidates of the

optimal points, $q_p^*(k)$ and $q_p^*(k+1)$, respectively. Then the paths from the start to the goal can be represented as the combinations of the consecutive pairs of $s(k,g)$ and $s(k+1,g')$ as shown with different colors in Fig. 3.6(b) with the ranges of $k \in \{1,2,\cdots,n_p-1\}$, $g \in \{1,2,3,\cdots,n_o(k)n_s+1\}$ and $g' \in \{1,2,3,\cdots,n_o(k+1)n_s+1\}$. Then, we can choose a path that minimizes the total cost from the start to the goal as the optimal path. Such a path can be found by a straightforward application of the dynamic programming. More specifically, the procedure goes as follows.

Let U_k^g be the minimum cummulative cost to the goal from $s(k,g)$ and $u_k^{gg'}$ be the cost from $s(k,g)$ to $s(k+1,g')$. According to the Bellman's principle of optimality ([9]), the minimum cummulative cost U_k^g can be said as the minimum sum of the cost from the current (i.e.,$s(k,g)$) to the next point (i.e., $s(k+1,g')$) together with the minimum cummulative cost from the next point (i.e., $s(k+1,g')$) to the goal, which can be written as follows.

$$U_k^g = \min_{g' \in \{1,2,..,n_o(k+1)n_s+1\}} (u_k^{gg'} + U_{k+1}^{g'}) \quad \forall g' \tag{3.14}$$

By setting $U_{n_p}^g = U_{n_p}^1 = 0$ at the goal when $k = n_p$ (and $g = 1$ since the goal point is the only possible candidate), Eq.(3.14) provides the Bellman equation, i.e. a recursive relation backward from the goal until we get $U_1^g = U_1^1$, which is the minimum cost from the start to the goal. Our purpose is to find a path from the start to the goal that has the minimum cost U_1^1 and to find the combinations of g for all kth steps.

With the allowable speed defined in Eq. (3.9), the time of traveling to pass a point p can be expressed as

$$u(p) = \frac{\Delta p}{v_a(p)} \tag{3.15}$$

where Δp is the change of the position by infinitesimal amounts. We can apply Eq.(3.15) to define the cost $u_k^{gg'}$ as

$$u_k^{gg'} := \frac{2\|s(k+1,g') - s(k,g)\|_2}{v_a(s(k+1,g')) + v_a(s(k,g))} \tag{3.16}$$

By using dynamic programming with Eqs. (3.14) and (3.16), we can obtain the set of g for all k for the minimum total cummulative cost and the set of off-skeleton points for the optimal path Q_p^*.

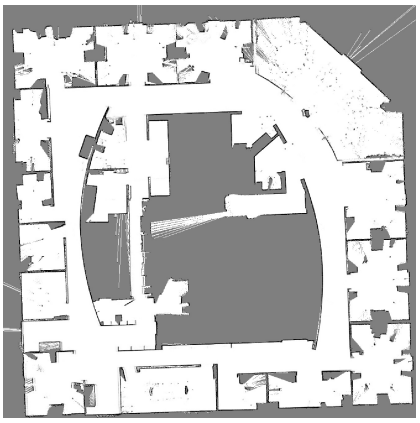
3.3 Simulation with Real Maps

In this section, we implement the proposed global path planning method with some real maps to show the optimality and computational efficiency of the resulting path. We compare the results with some existing path planning methods. Three real maps shown in Fig. 3.7 were used for our simulations: we downloaded the maps of Intel Research Lab ([42]) and Willow Garage ([88]) as shown in Figs. 3.7(a) and 3.7(b), respectively, and created a map of E-7 building in our campus (University of Waterloo) as shown in Fig. 3.7(c). We first visualize the HTM-EC of a real map and consequent path planning, see the optimal shape of the resulted path, and then look into the computation time of the proposed method.

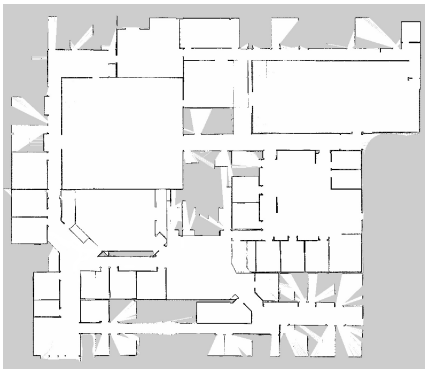
3.3.1 Visualization of HTM-EC and Resulting Paths with a Real Map

Figure 3.8 shows the visualization of HTM-EC and the path planning results with the map shown in Fig. 3.7(a). In Fig. 3.8(a), we showed only the skeleton of the map in green dots without the nodes and edges of the topology graph of HTM for better visibility. Two skeleton paths are shown in the figure with the start and goal positions. The path in the blue dashed line is the shortest skeleton path that does not consider the allowable speed of the robot, while the other path in the dashed orange line is the fastest skeleton path with the lowest cost based on the allowable speed. With the same velocity and acceleration setup that will be addressed in Section 3.4, the orange skeleton path planned by the HTM is about 4 m longer (68.8 m while the blue path is 64.9 m long), but its time of travel is expected to be 10.2 s shorter (45.2 s while the blue path is 35.0 s) provided that the robot is operated according to the allowable speed with respect to the corridor width. The average speed of the shortest path is 1.57 m/s, and that of the time-minimized path is 2.01 m/s. In Fig. 3.8(b), we visualized HTM-EC by showing the lines in cyan, $\overline{q(k)o_m(k)}$, that connect the skeleton path $\mathcal{Q}_p$ and its boundary points. The skeleton path in orange color is the same as the fastest skeleton path from Fig 3.8(a). On the other hand, the final path colored in red is the one that is optimized from the discretized corridor of the HTM-EC. In Fig. 3.8(c), we enlarged some right middle section of Fig. 3.8(b).

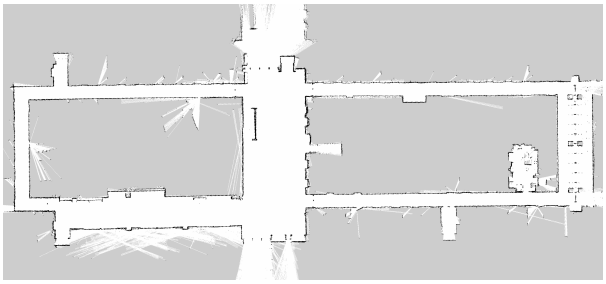
3.3.2 Computation Efficiency and Path Optimality



(a) Intel Research Lab

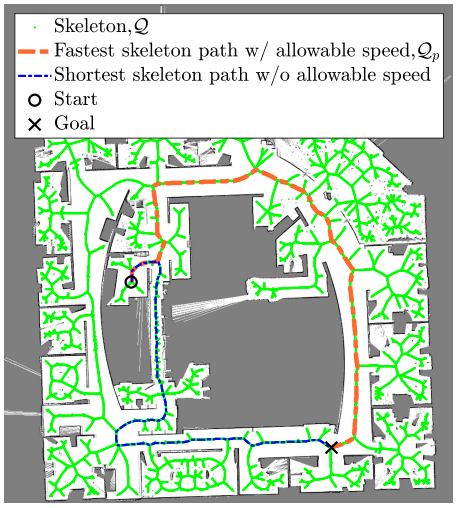


(b) Willow Garage

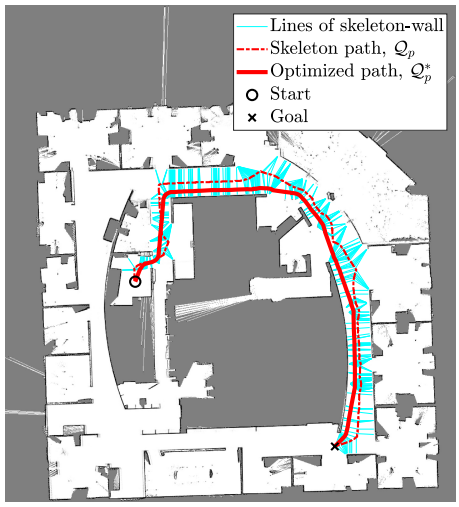


(c) E-7 in University of Waterloo

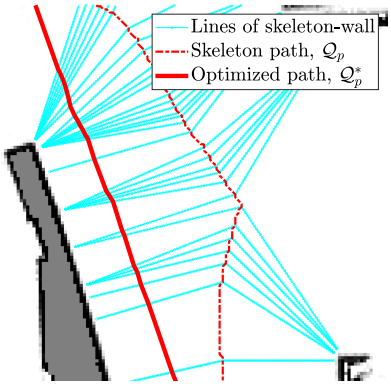
Figure 3.7: Maps used for the simulation of path planning



(a) Skeleton path planning with HTM and the comparison of two paths with/without allowable speed



(b) Visualization of HTM-EC and the optimized path from the skeleton path



(c) Enlarged Fig. 3.8(b) for the middle-right parts of the path

Figure 3.8: Path planning with HTM-EC using the map of Intel Research Lab

To compare the computation time and optimality of the path planned by HTM-EC, we implemented a grid-based global planning method equivalent to a widely used global planner in ROS (Robot Operating System), named *global planner* ([14, 69]). The algorithm utilizes a potential field created by grid-based Dijkstra expansion on an OGM until the expansion reaches a goal. Then a path can be found by tracking the gradient of the potential back to the start position. We applied the proposed cost defined in Eq. (3.15) for each grid cell when the potential field is created so that the resulting path can be considered as the reference of the optimal path for the given cost. As the grid-based Dijkstra method is complete and known as the optimal solution, we can use it for the reference trajectory to compare with the proposed method for the optimality of resulting paths.

We also compare the proposed method with the RRT and some of its variants, particularly for the computation time, as those methods are known for fast algorithms([**?**, 48]). We implemented the original version of RRT([63]), RRT*([46]), and RRT*J([115]). In RRT*J, the state-of-the-art RRT, skeleton information is used to create multiple potential functions for non-uniform sampling. This enables RRT samples to be scattered more likely around a feasible skeleton path. For the implemented RRT algorithms, we stopped the computation once a feasible path was found to get their minimized computation time.

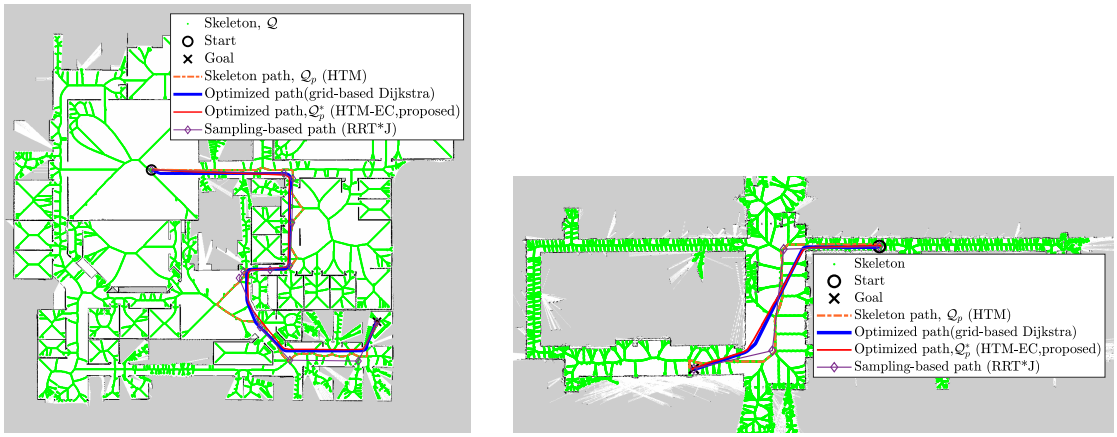
For each map shown in Fig. 3.7, we simulated both methods for 200 cases with random start and goal positions, which were conducted by MATLAB R2020b with Intel i7-10750H (2.6GHz) CPU and 16GB of memory. We compared the computation time and total cost from the simulations between HTM-EC planner and the other methods.

Figure 3.9 shows the comparison of the optimal paths between the HTM-EC and grid-based Dijkstra for the maps shown in Fig. 3.7(b) and Fig. 3.7(c). Among the RRT paths, we only showed the RRT*J path, which is likely stuck to a skeleton path, for better visibility. As RRT and RRT* often gave us totally different-shaped paths because of their randomness, showing their paths would not provide much meaning in the sense of optimality. The HTM-EC paths are similar in shape to the grid-based Dijkstra paths, qualitatively implying their optimality.

Table. 3.1 and Fig. 3.10 summarize the comparison of computation time and the total costs. The results of HTM-EC shown Fig. 3.10 for the comparison with other methods are for $n_s = 50$ for each line of $\overline{q(k)o_m(k)}$.

We computed the total costs of different methods in percentages over those of the grid-based Dijkstra path, as the absolute costs can largely depend on the random start and goal positions. Based on the total costs, the results show, in general, that HTM-EC paths have lower (better) costs over the grid-based Dijkstra paths: 9.86 % and 2.82 % lower in Intel Lab and UW E-7 maps, 1.06 % higher in Willow Garage. Noting that the skeleton

planning step of HTM-EC is exactly the same as HTM, our previous work, the results also show substantial improvements in optimality over the previous HTM method: 22.54 %, 19.87 %, and 17.44 % in each of the three maps, respectively.



(a) map of Fig. 3.7(b)

(b) map of Fig. 3.7(c)

Figure 3.9: Comparison of paths: Skeleton path (HTM), Optimized path by HTM-EC ($n_s = 50$), Optimized path by the grid-based Dijkstra, and Sampling-based path by RRT*J. Costs with the allowable speed were applied to all paths except the sampling-based path.

Besides the optimal path shape, the computational load is another important criterion for a path planner. Overall, the averaged computation time of HTM-EC with (0.7978 s, $n_s = 50$ case) is much shorter than the grid-based Dijkstra method (2.6574 s) and shorter than RRT*J (1.0572 s). RRT (0.7206 s) and RRT* (0.7565 s) are slightly faster, but they often fail to find a path as shown in their success rate in the table. The mean and standard deviation values of computation time are shown in the table for each map. With the fewer number of sampling, $n_s = 10$, the computation of HTM-EC becomes even much faster about 20 times, resulting in 0.0389 s on average, as shown in the table for each map. Figure 3.11 shows the computation time of HTM-EC and the total cost as its tradeoff with different sampling numbers n_s from 10 to 50, two of which are shown in Table. 3.1.

One interesting point is that the reduction of the computation with HTM-EC is significantly larger in the first two maps (the maps for Intel Research Lab and Willow Garage) than that in the third map (E-7 map), which comes from the complexity of a map. The third map has simpler branches of corridors compared to the other two maps, making the Dijkstra expansion easier (and consequently faster) than the other two maps. In the first two maps, the first is noisier (thus, more complex) than the second map (seen as many

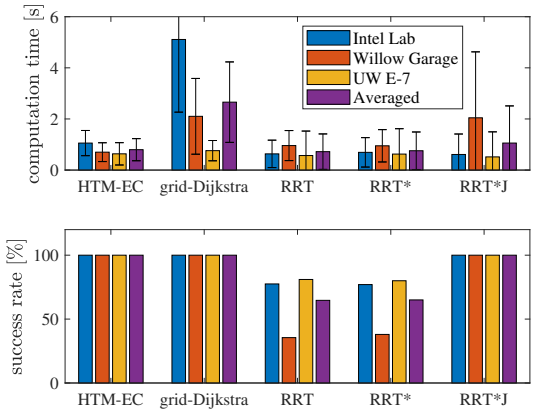


Figure 3.10: Comparison of the computation time with three maps shown in Fig. 3.7. HTM-EC is the case when $n_s = 50$, which is the most right case of the bottom figure in Fig. 3.11.

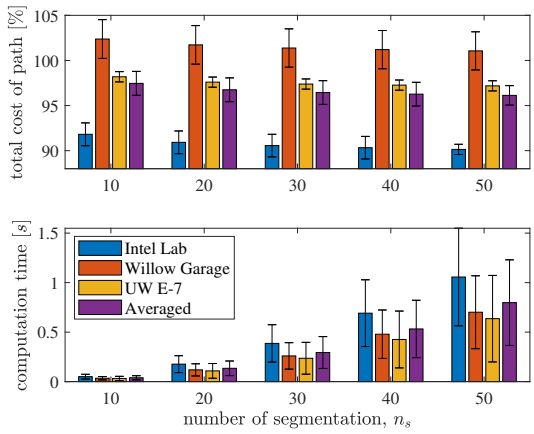


Figure 3.11: Computation time (bottom) and total cost of paths (top) of HTM-EC planning with different number of corridor segmentation, n_s, for three maps shown in Fig. 3.7. The costs (lower is better) are represented in percentages over the cost of the grid-based Dijkstra path.

small spots in the upper right chamber of the first map), causing longer computation time for the grid-based Dijkstra expansion.

Another point to mention is the variances of the computation time for three maps that comes with different complexities. The computation time of the proposed method shows smaller variance ($0.64 \sim 1.06$ s) than that of the grid-based Dijkstra method ($0.76 \sim 5.12$ s). It is notable that the standard deviation of the overall computation time for each map also has smaller variance with HTM-EC ($0.37 \sim 0.49$ s) than the grid-based Dijkstra method ($0.40 \sim 2.84$ s).

Similar points can be noted in the RRT methods. Although their overall planning time has less difference than the grid-based Dijkstra method, their high standard deviations, which can be even larger than the mean planning time, imply the planning time significantly varies. This is because of the randomness of the sampling-based tree expansion, often reaching to the maximum iteration when failing to find a path. All of three RRT methods showed the worst performance in computation time in the map of Willow Garage (Fig. 3.7(b)). The relatively small sizes of doors of many chambers in the map can cause the planning failure while there are large spaces, because it more likely results in the collision of a expanding-line of RRT tree that connects a randomly deployed sample to the existing RRT tree. The large chambers have more probabilities for new samples, but these samples are less likely seen by other samples out of the chambers. RRT*J showed successes in all simulations thanks to the informed sampling by the skeleton path, but the large spaces degraded its computation efficiency because more computation was required to get its multiple potential function with larger clearances.

We need to look at more details regarding computation time and variances with the compositions of the grid-based Dijkstra method and the RRT methods. Considering that the grid-based Dijkstra method takes two steps (i.e., Dijkstra expansion in a grid map and gradient tracking), its computation time can be viewed for each step. Note that each cell in the free space (white areas of an OGM) can be regarded as a node for Dijkstra expansion in a grid map; thus, the number of nodes and edges are large, resulting in heavy computation. For example, the number of nodes for the first map (Intel Research Lab) is 147,278 in the worst case when the Dijkstra expansion fully covers the free space of the map (in comparison, the topology graph of HTM-EC for the same map has 1,372 nodes, 1,522 edges, and 18,054 skeleton points). In fact, the step for Dijkstra expansion takes most of the computation time for the grid-based Dijkstra method, about 94.37 % of the total time (97.16, 94.07, and 91.88 % for each map, respectively). The RRT methods also required a significant portion of their planning time for the tree-graph expansion, which is more than 99.9 % of the total computation for all three variants. The expansion of the graph is required when replanning is requested. When a map becomes larger or more complex,

it will likely need more time for Dijkstra expansion and RRT tree expansion. Even on the same map, the time elapsed can vary in a wide range depending on the start and goal locations. For example, the Dijkstra expansion will quickly reach a goal if the goal is close to the start, while it will take more time when the goal is set far from the start location.

The computation time with HTM-EC can also be viewed in two steps: skeleton path planning and its optimization. Compared to the grid-based Dijkstra method and the RRTs, the computation time for graph search on the topology graph of HTM-EC is almost immediate (about 3.6 ms in average, leading to 0.47 % for $n_s = 50$ and 9.74 % for $n_s = 10$) due to the already constructed searching space with fewer numbers of nodes and edges (e.g., 1,372 and 1,522, respectively, as mentioned above). Instead, most of the computation time with HTM-EC is taken by the step for optimizing a skeleton path (about 99.53 % for $n_s = 50$ and 90.26 % for $n_s = 10$).

It is worth noting that an on-skeleton path, $\mathcal{Q}_p$, is a feasible path and available at a very early time, although further optimization is required to improve its quality at the later time of computation. This is always the case regardless of different sampling of n_s on each line of $\overline{q(k)o_m(k)}$. n_s, and the computation time for the optimization can even be significantly reduced with fewer segmentations as shown in Fig. 3.11 and Table. 3.1. All of these observations not only show that the computation time can be reduced with HTM-EC but also imply that it will be even more advantageous when a map becomes larger and more complex.

3.4 Robot Implementations: Verification of the Allowable Speed

While Section 3.3 shows us that the proposed path planning method is implementable with real maps, we need further implementations on a real robot to see if the proposed cost can lead to a faster path while it is longer in length. Therefore, we have performed ROS simulations and experiments to verify our proposed method for time-minimization with a real mobile robot. Through the robot simulations and experiments, we compare the time of traveling between the shortest path, planned by the *global planner* in ROS, and the time-minimized path, planned by HTM-EC, with the allowable speed and the proposed cost. The ROS planner does not use the proposed cost, hence returns the shortest path by default. A ROS local planner, *TEB (Timed-Elastic-Band) local planner* ([89]), was used for both global planners during the implementations in this section.

An omnidirectional mobile robot, *Summit XLS* manufactured by Robotnik Inc., was

used for simulations and experiments with the same parameters and conditions. Figure 3.14(a) shows the mobile robot, and Fig. 3.14(b) shows the robot running in a corridor to show the environment of the experiments. We used the map of the E-7 building on our campus (Fig. 3.7(c)) for the ROS simulations and performed the experiments in the same place with the same local planner and its parameters.

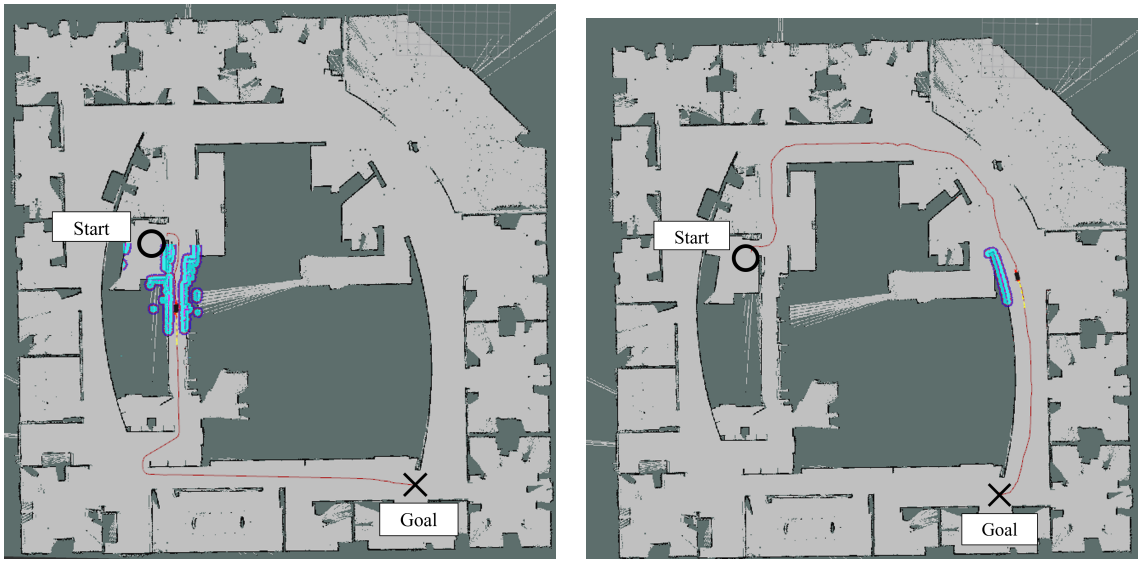
For the *TEB local planner*, we set it with a $8m \times 8m$ in the local map with small sideways velocity, i.e., $v_x > v_y$, where positive x is forward, and positive y is sideways to the left of the mobile robot. The maximum velocity in the x direction was set as 3 m/s and y for 0.5 m/s, and the acceleration was set 1.5 m/s^2. The allowable speed was applied to the *TEB local planner* for both global plans. While the robot was moving, its states were obtained by the *AMCL (Adaptive Monte-Carlo Localization)* algorithm in ROS with a *VLP-16* LiDAR. The estimated robot's location and clearance from the obstacles were used to adjust the robot's actual navigation speed under the allowable speed. With this setup, we performed simulations with two different environments, Intel Lab (Fig. 3.7(a)) and UW E-7 (Fig. 3.7(c)). We also performed a real robot experiments with the UW E-7 map.

3.4.1 Scenario 1: Robot Simulations with Intel Lab Map

Figure 3.12 describes the simulation scenario by showing the start and goal positions and the different paths between the shortest (Fig. 3.12(a)) and time-minimized with the allowable speed (Fig. 3.12(b)), which gives us the different corridor selections. The shortest path takes the downward corridor, while the time-minimized path selects the upward corridor for larger clearances and faster speed in operation. The wider corridors enable the robot to move faster with higher speed commands shown in Fig. 3.13, and the time of traveling can be reduced as a result. We performed 10 simulations to check the consistency of the results and summarized the results in Table 3.2. The time-minimized path is longer, about 14.3m in length, but 3.4 s faster than the shortest path. Compared to the skeleton path shown in Fig. 3.8(a), the path length was reduced to 64.3 m from 68.9 m, and the time of traveling was also reduced to 30.9 s from 35.0 s.

3.4.2 Scenario 2: Robot Simulations and Experiments with UW E-7 map

Figure 3.15(a) describes the simulation environments of UW E-7 map in Gazebo, a ROS simulation tool. We set the start position in the middle of the left corridor and the goal



(a) Simulations with Intel Lab: the shortest path (the dark red line)

(b) Simulations with Intel Lab: longer, but faster path by the proposed cost (the dark red line)

Figure 3.12: Comparison of the shortest paths and time minimized paths with ROS simulations with Intel Lab

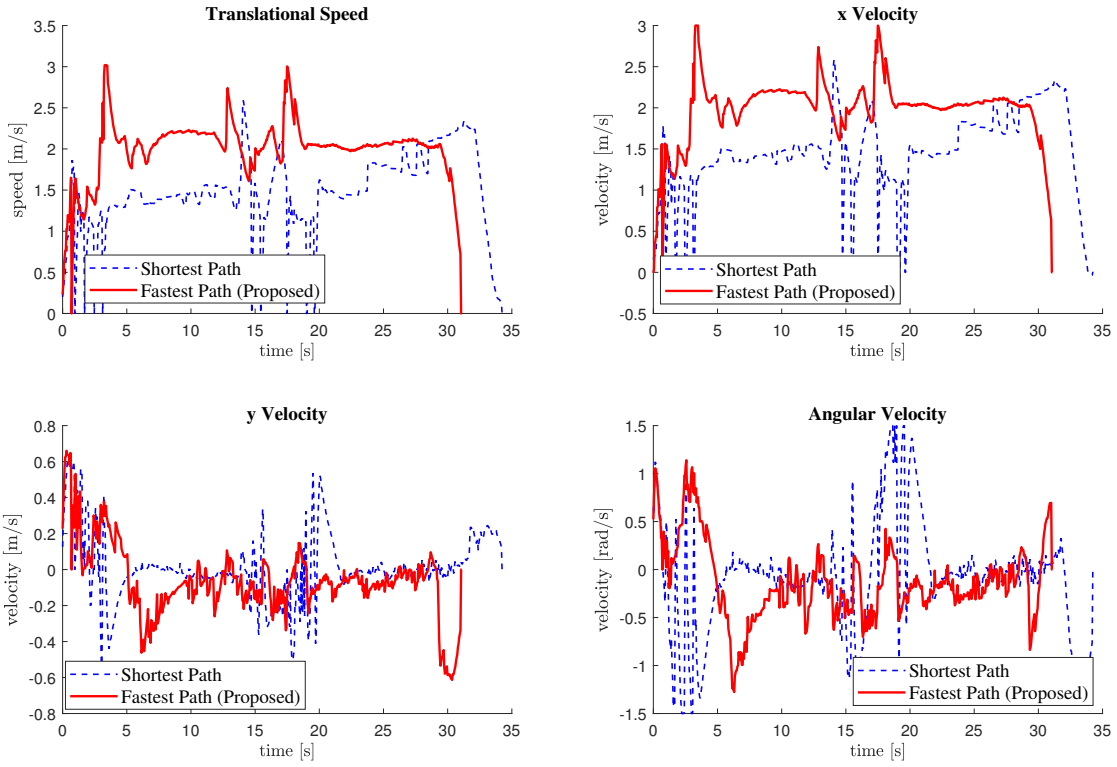
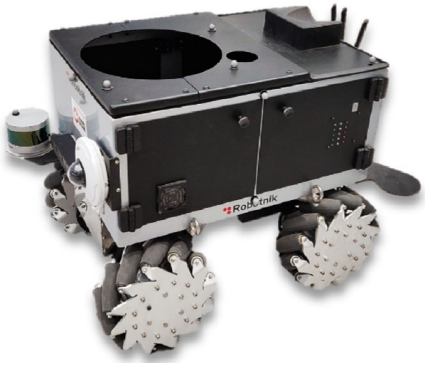


Figure 3.13: Comparison of the actual speed along the paths in the simulations (Intel Lab)

at the slightly upper location in the right lobby. It is easy to expect, as shown in Fig. 3.15(a), that the upper corridor will provide the shortest path whereas the lower corridor can provide the fastest path, even with the longer distance, thanks to its large widths that enable higher speed operation. We set up the same start and goal locations for the experiments, and both the simulations and experiments showed us the fastest path via the lower corridor by our HTM-EC planner and the shortest path via the upper corridor by the ROS default global planner. We performed 20 simulations and 12 experiments with UW E-7 map, and showed the path comparison for the experiments in Fig. 3.15(b). In the figure, we used the dashed lines for the planned paths and solid lines for the actual paths that the robot took during the experiments.

We summarized the results from the simulations and the experiments with the UW E-7 map in Table 3.3 and Table 3.4, respectively. The lower paths, generated by the proposed method, are about $6 \sim 7$ s faster, although their lengths are about $5 \sim 7$ m longer due to the high-speed operation captured in Fig. 3.16. The videos of the experiments can be accessed from the link in the footnote[1].

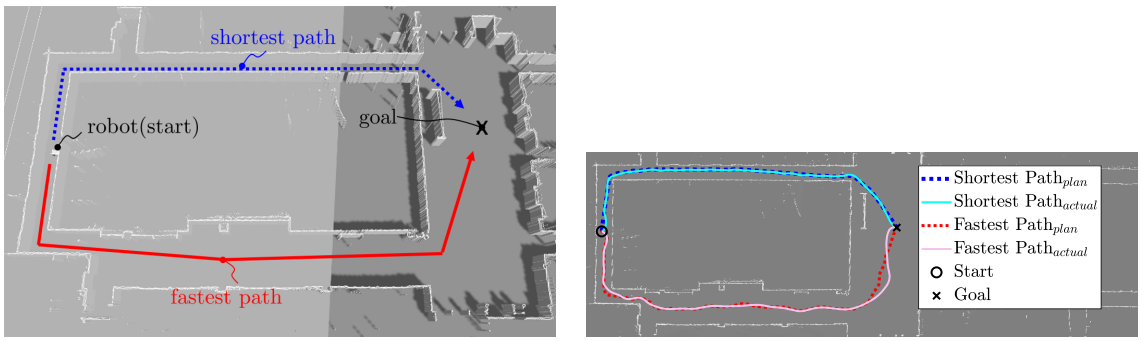


(a) Mobile robot

(b) Mobile robot navigating in the experiment

Figure 3.14: Environment for the experiments

[1] https://youtu.be/oDTTdYcj_qA



(a) Simulation environments (b) Comparison of the paths in the experiments

Figure 3.15: Simulation environments and path comparison in the experiments

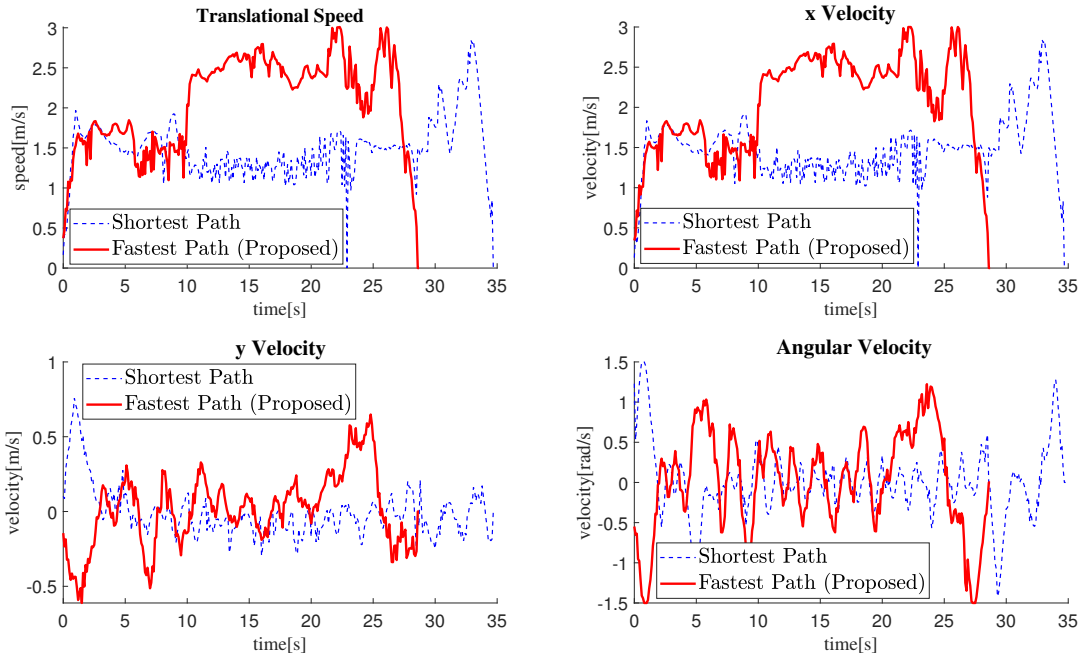


Figure 3.16: Comparison of the actual speed along the paths in the experiments (UW E-7)

3.5 Discussion

In this chapter, we introduced a path planning method that utilizes *allowable speed* to explicitly consider safety in the structure of HTM-EC. Through simulations with real maps, it has been shown that the usage of the sparse structure of HTM-EC allowed lower computation time than the conventional method of potential fields with grid-based Dijkstra expansion while the completeness of the method is achieved. Specifically, we expect the advantage of less computation time by the proposed method will be even more highlighted when a map becomes large and complex due to the low variances of the computation. The planned global path by the proposed method is refined from an on-skeleton path to be smooth and optimal.

As we have verified through the implementations to a real robot, the use of allowable speed for the global path planning provided the robot with a faster path than the shortest path with the preference to the spacious areas for the path planning. The robot had no problem increasing its speed in the wider corridors that eventually allowed the longer path to be faster.

3.5.1 Issues

There are several implementation issues that we would like to discuss further. Increasing the number of segmentation of the line between a skeleton point and its closest obstacles will make the resulting path closer to the optimal with the finer refinement. However, we recommend keeping a certain value for the number of segmentation, as the quality of the path would be good enough for a global path (in practice, the local path planners will modify the global path in the proximity to take care of the changes of environments).

It is also noted that, for some of the skeleton points, we did not obtain the closest points on the obstacles (i.e., walls). We used the image-thinning method to get the skeleton from a map which resulted in the pixel-based integer values of (x, y) for a skeleton point. Thus the skeleton point may be slightly biased to one of the boundary points, and the boundary points, at least two, may give slightly different distances from the related skeleton point, resulting in failure to detect the obstacle points with the exact same distance from some of the skeleton points. This issue can be resolved by applying other skeletonization methods, such as the Voronoi Diagram, which provides us with the exact values of (x, y) for the skeleton points.

3.5.2 Future Works

As topological planning takes the abstract cost accumulation of corridors, one interesting future work can be related to adjusting corridor costs in real-world applications. In warehouses, for example, where the environments are mostly static and well-constructed but still have some dynamic objects. In such cases, even large regions can be slower corridors if they are often crowded with moving obstacles, degrading the actual navigation time. We expect such flexibility of adjustment of the corridor costs would help the time performance of navigation robots in real-world situations.

Another future work will be on the extended use of the proposed method for the local planning aspect. As the topology graph of HTM-EC contains the intersection of corridors, we can use HTM-EC for other applications of mobile robot navigation, such as the motion prediction of other moving agents in the surrounding based on the intersection information incorporated in the graph. Considering most dynamic agents like humans often move toward the intersections, it may help predict their motion by presuming their short-term goals on nodes of the topology graph of the HTM-EC and by inferring their goals.

Table 3.1: Comparison of the simulation results for the computation time, total cost, and success rate with the maps shown in Fig. 3.7. Standard deviation is written in parentheses if available.

Method	Step	Intel Lab			Willow Garage			UW E-7		
		Time [s]	Cost [%]	Success [%]	Time [s]	Cost [%]	Success [%]	Time [s]	Cost [%]	Success [%]
HTM-EC ($n_s = 50$)	Skeleton planning	0.0040 (0.0021)	112.68 (9.77)	100.0	0.0032 (0.0015)	120.93 (29.32)	100.0	0.0036 (0.0026)	114.62 (0.3248)	100.0
	Optimization	1.0525 (0.4913)	-	-	0.6978 (0.3672)	-	-	0.6322 (0.4346)	-	-
	Total	1.0565 (0.4926)	**90.14 (5.66)**	100.0	0.7010 (0.3682)	101.06 (21.14)	100.0	0.6358 (0.4368)	**97.18 (5.62)**	100.0
HTM-EC ($n_s = 10$)	Skeleton planning	0.0040 (0.0021)	112.68 (9.77)	100.0	0.0032 (0.0015)	120.93 (29.32)	100.0	0.0036 (0.0026)	114.62 (0.3248)	100.0
	Optimization	0.0464 (0.0223)	-	-	0.0312 (0.0161)	-	-	0.0283 (0.0192)	-	-
	Total	**0.0504 (0.0236)**	91.81 (12.66)	100.0	**0.0343 (0.0170)**	102.37 (21.50)	100.0	**0.0319 (0.0214)**	98.19 (5.70)	100.0
grid-based Dijkstra	Dijkstra exapnsion	4.9643 (2.8399)	-	-	1.9781 (1.4801)	-	-	0.6985 (0.3946)	-	-
	Planning (gradient)	0.1451 (0.0192)	-	100.0	0.1246 (0.0156)	-	100.0	0.0617 (0.0080)	-	100.0
	Total	5.1094 (2.8435)	100.0*	100.0	2.1027 (1.4822)	**100.0***	100.0	0.7602 (0.3967)	100.0*	100.0
RRT	Tree-graph expansion	0.6337 (0.5361)	-	-	0.9585 (0.5878)	-	-	0.5689 (0.9572)	-	-
	Planning	0.0002 (0.0005)	-	-	0.0002 (0.0009)	-	-	0.0001 (0.0002)	-	-
	Total	0.6340 (0.5359)	123.16 (37.12)	77.5	0.9587 (0.5875)	135.5 (50.48)	35.5	0.5691 (0.9571)	123.54 (35.37)	81.0
RRT*	Tree-graph expansion	0.6930 (0.5759)	-	-	0.9498 (0.6327)	-	-	0.6259 (0.9931)	-	-
	Planning	0.0006 (0.0044)	-	-	0.0001 (0.0003)	-	-	0.0001 (0.0002)	-	-
	Total	0.6936 (0.5758)	108.71 (24.84)	77.0	0.9499 (0.6326)	121.69 (37.56)	38.0	0.6260 (0.9930)	107.24 (18.82)	80.0
RRT*J	Tree-graph expansion	0.6103 (0.8002)	-	-	2.0449 (2.5801)	-	-	0.5158 (0.9798)	-	-
	Planning	0.0003 (0.0005)	-	-	0.0002 (0.0003)	-	-	0.0002 (0.0002)	-	-
	Total	0.6107 (0.8003)	95.88 (6.01)	100.0	2.0451 (2.5801)	101.28 (8.81)	100.0	0.5159 (0.9798)	100.90 (8.39)	100.0

* The grid-based paths were used for the reference costs.

Table 3.2: ROS Simulations: total length of the path, average speed and total time for the cases (Intel Lab)

	Fastest path (Proposed)	Shortest path
Path Length [m]	**64.3486** (std 0.1563)	50.0488 (std 0.2538)
Time of Traveling [s]	**30.9480** (std 0.2891)	34.3320 (std 0.5051)
Avg. Speed [m/s]	**1.9995** (std 0.0069)	1.4570 (std 0.0162)

Table 3.3: ROS Simulations: total length of the path, average speed and total time for the cases (UW E-7)

	Fastest path (Proposed)	Shortest path
Path Length [m]	**55.0191** (std 0.1074)	50.8499 (std 0.0324)
Time of Traveling [s]	**28.8082** (std 0.4703)	35.6360 (std 0.2913)
Avg. Speed [m/s]	**1.9692** (std 0.0089)	1.4424 (std 0.0093)

Table 3.4: Experiments: total length of the path, average speed and total time for the cases shown in Fig. 3.15(b) (UW E-7)

	Fastest path (Proposed)	Shortest path
Path Length [m]	**58.8000** (std 0.0809)	51.5589 (std 0.0919)
Time of Traveling [s]	**29.5337** (std 1.8513)	35.3321 (std 1.1108)
Avg. Speed [m/s]	**2.0448** (std 0.0937)	1.4586 (std 0.0380)

Chapter 4

Local Motion Planning Based on a Pedestrian-Model and Degree of Cooperation

4.1 Introduction

In this chapter, we address a local motion planning method based on a pedestrian-behavior model and the degree of cooperation to address the interaction between decision-making agents.

For given global paths, it is typical that mobile robots observe their proximal environments and take their local goals from the given global paths to maneuver their motions. Local surroundings frequently change with the presence of moving obstacles and the change of static environments. Dynamic obstacles have to be addressed for collision avoidance in this maneuvering step.

There are also static obstacles to be considered at the same time. With the varying number of dynamic obstacles around the robot, the number of obstacle states is also changing, leading to the *scalability* issue: the computation time for planning also increases depending on the obstacle number. Sensor-level information, such as laser scans, can be advantageous in terms of the scalability issue thanks to the number of sensor data that is fixed and invariant no matter how many moving obstacles exist in their measurement. If provided proper navigation policies based on the sensor-level measurement, sensor-level data can be beneficial.

Considering many cases include dynamic decision-making agents, interactions between these agents have to be considered. Humans are common examples of such dynamic objects as decision-making agents, and it is natural to apply a pedestrian model for mobile robot navigation. No doubt, the human behavior of navigation has been studied extensively, including mobile robot navigation. In [79], behavioral heuristics of human navigation was suggested based on a cognitive science approach. The model focuses on visual information and instant cognition of the surroundings, which appeals to our own intuition of how humans navigate (i.e., we walk through based on visual images and do not actually feel any interactive force from others).

We formulate our problem based on the two cognitive heuristics used in their work to model pedestrian navigation behaviors as follows.

1) *"A pedestrian chooses the direction that allows the most direct path to destination point, taking into account the presence of obstacles"*;

2) *"A pedestrian maintains a distance from the first obstacle in the chosen walking direction that ensures a time to collision of at least τ"*;

4.2 Methodology

4.2.1 Pedestrian-Model-Based Collision Avoidance

The first heuristic is *"A pedestrian chooses the direction ϕ_{des} that allows the most direct path to destination point* $\mathbf{o}$*, taking into account the presence of obstacles"*. In this work, $\phi \in [\theta - \phi, \theta + \phi]$ represents the vision field of the pedestrian, where θ is the angle of the line of sight. A cognitive function $f(\phi)$ was defined to represent the distance to the first collision with respect to ϕ as the parameter of the heuristic. If another pedestrian was observed in the direction ϕ, his walking speed and body size were taken into account in $f(\phi)$. If no collision is expected to occur in direction ϕ, $f(\phi)$ is set to a default maximum value d_{max}, which represents the "horizon distance" of the pedestrian as shown in Fig. 4.1. Image A of the figure illustrates an example that a pedestrian p_1 is moving to his destination $\mathbf{o}_1$ with three pedestrians p_2, p_3 and p_4. Image B shows the visual scene of p_1 in the same example. The abstraction of the scene is shown in image C as a black and white vision field of p_1, where the darker area represents a closer collision distance. In image D, the distance along with the field of view is presented, where the angle toward pedestrian p_3 is shifted to that of p_3' by considering p_3 is moving to the right, and the

collision will happen with p_3 along with the shifted angle. Denoting $\mathbf{s}(\phi)$ the point (not shown in Fig. 4.1) that can be computed from the position of the pedestrian (i.e. p_1) by adding the distance $f(\phi)$ along with the direction ϕ, the first heuristic is given by[37]:

$$\alpha_{des}(t) = \arg\min_{\alpha} d(\mathbf{s}(\alpha), \mathbf{o}) \tag{4.1}$$

where $d(\mathbf{s}(\alpha), \mathbf{o})$ is Euclidean distance between $\mathbf{s}(\alpha)$ and $\mathbf{o}$. In Fig. 4.1, $\alpha_{des}(t)$ is α_0.

The second is *"A pedestrian maintains a distance from the first obstacle in the chosen walking direction that ensures a time to collision of at least τ"*. In other word, the speed $v_{des}(t)$ is given by[37]:

$$v_{des}(t) = \min\left(v_0, \frac{D(\alpha_{des})}{\tau}\right) \tag{4.2}$$

where v_0 is a preferred speed of the pedestrian and the $D(\alpha_{des})$ is the distance between the pedestrian and the first obstacle in the desired direction α_{des} at time t. The desired velocity vector can be represented as $\mathbf{v}_{des}(t)$ that has the speed of $v_{des}(t)$ along with the direction of $\alpha_{des}(t)$. When the current velocity vector is $\mathbf{v}(t)$, the change in the actual velocity is given by[79]:

$$\frac{d\mathbf{v}}{dt} = \frac{\mathbf{v}_{des}(t) - \mathbf{v}(t)}{\tau} \tag{4.3}$$

4.2.2 Safety Guarantee

In this section, we prove the safety of the pedestrian models based on the *velocity obastacle* (VO) [27]. We first show how to locate a measured point to its location of the expected collision, or cognitive collision point, with the given velocity. The translation of the point into a new position (i.e., the cognitive position of the collision) provides us with the angle shift in the polar coordinate of the robot.

Then, we will show that multiple measurement points on an obstacle's surface can be translated to their collision locations, giving us the consecutive angle shifts. Then, the cognitive collision points of the measured points can be computed. The consecutive angle shift can be mapped on an arc of the circle centered at the origin of velocity space in a polar coordinate with the radius of the preferred navigation speed.

Finally, we will show the safety of the desired velocity of the robot by showing the collision arc is overlapped with the velocity obstacle. Figure 4.2 describes the abovementioned procedures.

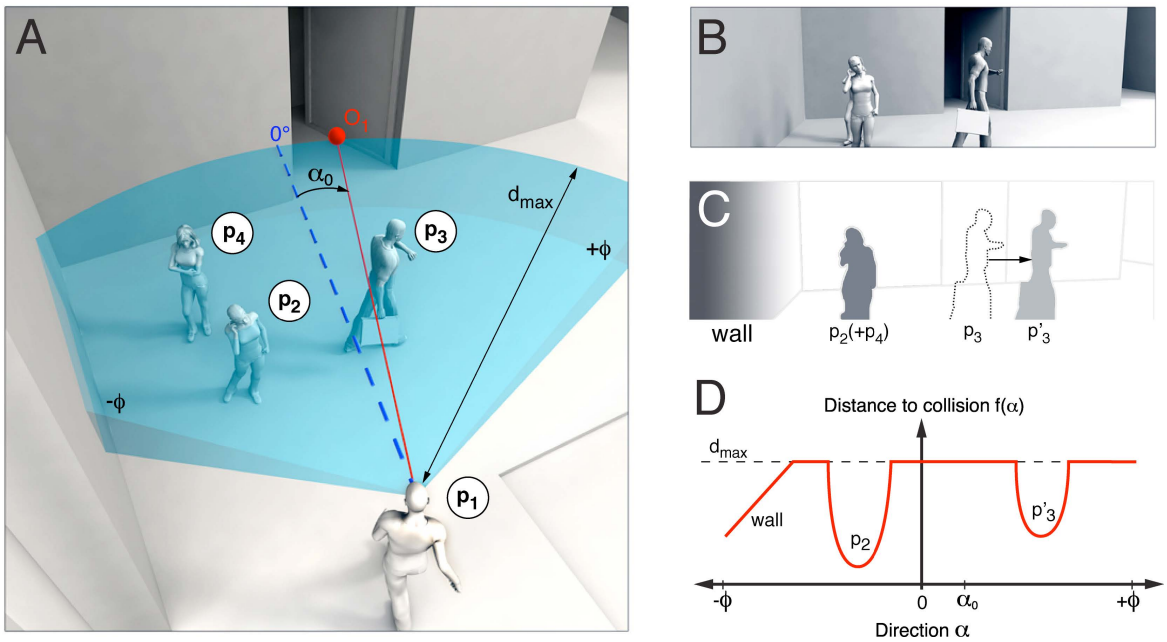


Figure 4.1: Illustration of a pedestrian [79]

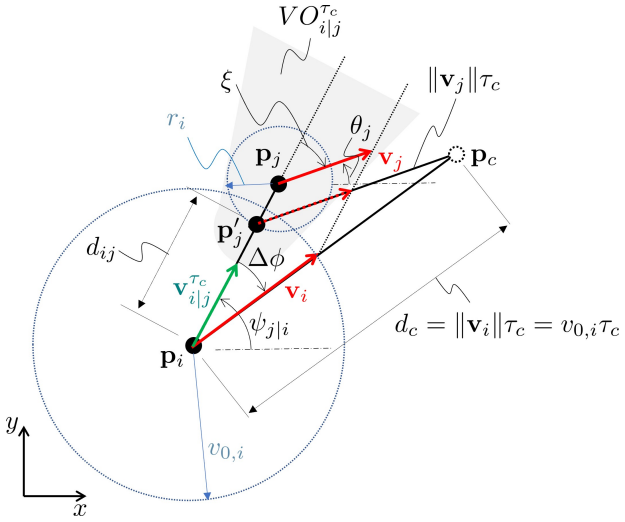
Velocity Obstacle

For given two agents i and j, the VO of agent i induced by agent j, denoted by $VO_{i|j}$, is the set of all relative velocities that cause a collision anytime in the future between the two agents. Letting $D(\mathbf{p}, r)$ denote a disc centered at $\mathbf{p}$ with radius r, the velocity obstacle can be written as follows [108].

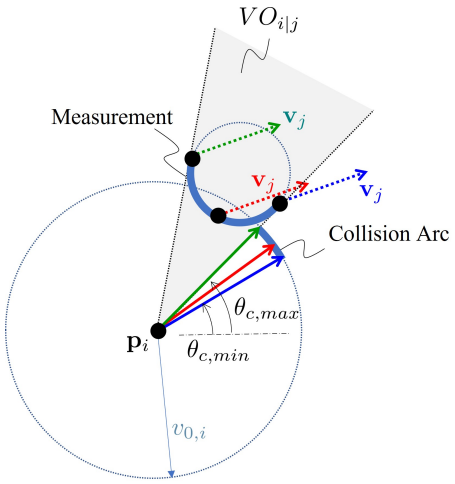
$$D(\mathbf{p}, r) = \{\mathbf{q} \mid \|\mathbf{q} - \mathbf{p}\| < r\} \tag{4.4}$$

$$VO_{i|j} = \{\mathbf{v} \mid \exists t > 0 :: t\mathbf{v} \in D(\mathbf{p}_j - \mathbf{p}_i, r_i + r_j)\} \tag{4.5}$$

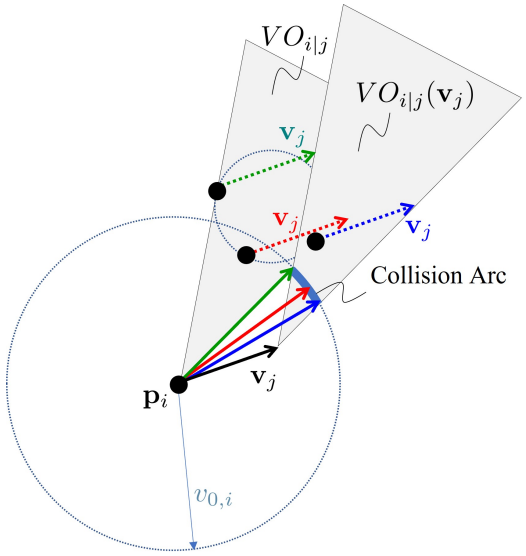
Any combination of the velocities $\mathbf{v}_i$ and $\mathbf{v}_j$ that results in their relative velocity $\mathbf{v}_{ij} \in \mathbf{v}$, where $v_{ij} = \mathbf{v}_i - \mathbf{v}_j$, leads to a collision in the future if they keep their velocities. The shape of the $VO_{i|j}$ is a cone with its apex on the origin of 2-D velocity space, as shown as the grey cones in Figs. 4.2(b) and 4.2(c).



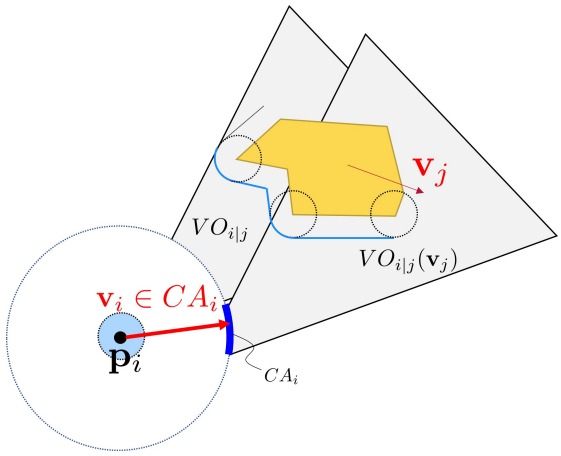
(a) cognitive collision location: point-point collision



(b) collision arc: measurement mapping on the circle of preferred speed



(c) collision-free: overlap of collision arc with the velocity obstacle



(d) collision arc with general shapes of obstacles

Figure 4.2: Collision-free velocity with the cognitive position of an obstacle

Cognitive Collision Location of a Measured Moving Point

For a given circular shape of the mobile robot i located at $\mathbf{p}_i$, let $\mathbf{p}_j$ and $\mathbf{v}_j$ denote the position and velocity of a point on the surface of an obstacle. With a circular volume of the robot, denoted by the radius r_i, the collision between the volumeless point j and the point i with r_i is equivalent to that between the point j with r_i and the volumeless point i. Figure 4.2(a) describes this equivalent situations. $\mathbf{p}_i$ is now the volumeless robot, and the point j is moved to $\mathbf{p}'_j$ with the inflation of r_i from $\mathbf{p}_j$. The figure describes a certain time t, but we skipped the time index t for simplicity. For the given navigation speed preference of agent i, denoted by $v_{0,i}$, a collision between $\mathbf{p}_i$ and $\mathbf{p}'_j$ occurs at the collision point $\mathbf{p}_c$ only if their relative velocity $\mathbf{v}_{ij}$ is aligned with their relative position vector $\mathbf{p}_{ji} = \mathbf{p}_j - \mathbf{p}_i$ in the same direction.

Then, the angle shift, denoted by $\Delta\phi$, can be computed for the given velocity $\mathbf{v}_j$ as follows.

$$\Delta\phi = \sin^{-1}\left(\frac{\|\mathbf{v}_j\|\sin(\psi_{j|i} - \theta_j)}{\|\mathbf{v}_i\|}\right) = \sin^{-1}\left(\frac{\|\mathbf{v}_j\|\sin(\psi_{j|i} - \theta_j)}{v_{0,i}}\right) \tag{4.6}$$

where $\psi_{j|i}$ is the robot's measurement angle on point j, θ_j is the heading angle of agent j.

In Fig. 4.2(a), we drew VO with the time window τ_c, denoted by $VO^{\tau_c}_{i|j}$, instead of the cone shape of the original VO. The time window VO with its truncated apex shows the forbidden region for the relative velocity selections to avoid a collision within τ_c from the current time step. The relative velocity v_{ij} is equivalent to $v^{\tau_c}_{i|j}$ in the figure, which represents that v_{ij} causes a collision at the exact time τ_c from the current. Please notice that we keep our preference of the navigation speed $v_{0,i}$; thus, such collision at τ_c occurs when we select a specific direction on the velocity space in the polar coordinate.

The collision time τ_c can be obtained by

$$\tau_c = \frac{d_{ij}}{\|\mathbf{v}^{\tau_c}_{\mathbf{i|j}}\|} = \frac{\|\mathbf{p}_{ji}\| - r_i}{\|\mathbf{v}_i\|\cos\Delta\phi - \|\mathbf{v}_j\|\cos\xi} \tag{4.7}$$

where d_{ij} is the distance between $\mathbf{p}_i$ and $\mathbf{p}'_j$ and $\xi = \psi_{j|i} - \theta_j$ is the angle of $\mathbf{v}_j$ from the line of the relative position $\mathbf{p}_{ji}$.

Collision Arc on the Circle of Preferred Speed

Consider Fig. 4.2(b) where we have multiple points on the surface of the inflated circle of $\mathbf{p}_j$ with the robot radius r_i. Three black points are shown in the figure as an example.

With the computation of the cognitive location of collision for each point on the inflated circle, we can easily imagine that the those three points on the inflated circle of $\mathbf{p}_j$ can be translated to the location of their collision points on the circle of preferred speed of $\mathbf{p}_i$. Note that the time of collision τ_c for each point can be different. As shown in the figure, computing the cognitive collision location (the collision locations are not shown Fig. 4.2(b). See $\mathbf{p}_c$ in Fig. 4.2(a)) will give us the angle shift that maps $\mathbf{v}_i$ on different points on the circle centered at the origin of velocity space of robot i with the radius of preferred speed $v_{0,i}$. Different colors were used in the figure to pair different $\mathbf{v}_i$s corresponding to different points.

The arc of the preferred speed circle designated by $\mathbf{v}_i(\theta_i)$ where $\theta_i \in [\theta_{c,min}, \theta_{c,max}]$ as shown in Fig. 4.2(b) is defined as *Collision Arc* of the preffered velocity circle. Considering the maximum horizon distance $d_{max} = \|\mathbf{o}_i - \mathbf{p}_i\|$, which represents the distance to the goal $\mathbf{o}_i$ from the current position $\mathbf{p}_i$, the maximum collision time is bounded by $\tau_c < \tau_{max}$ where $\tau_{max} = \frac{d_{max}}{v_{0,i}}$. When $\tau_c = \tau_{max}$, it implies that there is no expectation of obstacles in the chosen angle. Then we can define the collision arc as follows.

Definition 4.2.1 (Collision Arc)**.** For a given preferred speed $v_{0,i}$ of agent i, the collision arc, CA_i, of the preferred speed circle of radius $v_{0,i}$ centered at the origin of its velocity space is defined as the arc that corresponds angle θ_c such that

$$CA_i = \{\mathbf{v}_i | \tau_c(\theta_c) < \tfrac{d_{max}}{v_{0,i}} \wedge \|\mathbf{v}_i(\theta_c)\| = v_{0,i},\ \theta_c \in [0, 2\pi]\}.$$

Note that the definition of CA_i does not contain the moving obstacle index. This is because the collision arc can also be defined in the same way with static obstacles, although we induced the collision arc from moving obstacles so far. The only difference between static and dynamic obstacles in defining the corresponding collision arc is that we use cognitive locations of moving obstacles, whereas we use the measurement data itself for static obstacles. Once the cognitive distances are obtained for the robot's surroundings, the collision arc can be defined in the same way. This can be easily seen in Fig. 4.2(b). If the object j (the small dashed circle in $VO_{i|j}$) is static (i.e., $\mathbf{v}_j = 0$), the measurement points on its arc (the thick curve) will give us the collision arc in the VO cone.

Figure 4.2(d) shows the extension of the collision arc into general shape of obstacles. An obstacle is shown with a yellow polygon. The inflation of the measurement of the agent i on the obstacle's surface is shown as light blue line with its radius of r_i. Then, each point of the inflated measurement can be mapped into collision arc CA_i of the agent i, as we explained with Fig. 4.2(b). Different points of the measurement can be mapped into a single collision angle with different τ_c. When it happens, smaller collision time will be chosen as τ_c of the angle.

Collision-free Desired Velocity

Now, we will show the defined collision arc CA_i is always a subset of the velocity obstacle $VO_{i|j}$ for any given value of $\mathbf{v}_j$. In fact, the intersecting arc of the preferred speed circle with VO is the collision arc. We already checked the case of static obstacles where $\mathbf{v}_j = 0$ in the previous section. For a moving object j, Fig. 4.2(c) visualizes the collision arc and the VO with the given velocity of the object $\mathbf{v}_j$. Three points on the surface of object j (the thick black points on the small circle) have the same velocity. Recall the computation of the angle shift with Eq. (4.6) and Fig. 4.2(a). In Fig 4.2(a), one can easily notice that two dashed lines are parallel: one connecting the ends of the red arrows $\mathbf{v}_i$ and $\mathbf{v}_i$ and the other connecting the bases of the same red arrows. In Fig. 4.2(c), the two thick black points at the sides of the obstacle j lie on the left (the green dashed arrow) and right (the blue dashed arrow) boundaries of $VO_{i|j}$. The corresponding $\mathbf{v}_i$, the green and thick blue arrows, respectively. The bases and the ends of the arrows $\mathbf{v}_i$ and $\mathbf{v}_j$ should have parallel lines as we observed in Fig. 4.2(a), which results in the intersecting points of CA_i and $VO_{i|j}$. Selecting the outside arc angles of CA_i with the preferred speed $v_{0,i}$ leads to a collision-free velocity because the resulting $\mathbf{v}_i$ is located outside of the velocity obstacle. Thus, the following theorem is proven.

Theorem 1 (Collision Free Desired Velocity)**.** The desired velocity $\mathbf{v}_{des}$ in Eq. (4.2) with the angle selection α_{des} outside the arc angles of CA_i is collision-free.

Considering CA_i conveys the velocities with $\tau_c < \tau_{max}$ that implies obstacles in anytime in the future, it can be too conservative if all velocities CA_i are rulled out of the velocity candidates. We can still choose the desired velocity in CA_i if a collision is expected with a large collision time. Recalling Eq. (4.3) of the pedestrian model, which tells us the actual change of an agent's velocity to achieve the given $\mathbf{v}_{des}$ gradually occurs within the specific time period τ, we can choose the angle α_{des} of $\mathbf{v}_{des}$ if $\tau_c \geq \tau$. Thus, we add a condition to Eq. (4.1) in selecting the desired angle α_{des} of the pedestrian model such that

$$\begin{aligned} \alpha_{des}(t) = &\arg\min_{\alpha} d(\mathbf{s}(\alpha), \mathbf{o}) \\ &\text{s.t} \tau_c(\alpha) \geq \tau \end{aligned} \tag{4.8}$$

Selecting the desired angle α_{des} in this way, Eq. (4.2) gives us the desired velocity without speed reduction from its preferred speed of $v_{0,i}$, holding the collision-free condition with the velocity vector pointing on the circumstance of the preferred speed circle. Thus, the following theorem is proven.

Theorem 2 (Collision Free Desired Velocity with a Time Window τ)**.** The desired velocity $\mathbf{v}_{des}$ in Eq. (4.2) with the angle selection α_{des} outside the arc angles of $\tau_c < \tau$ is collision-free.

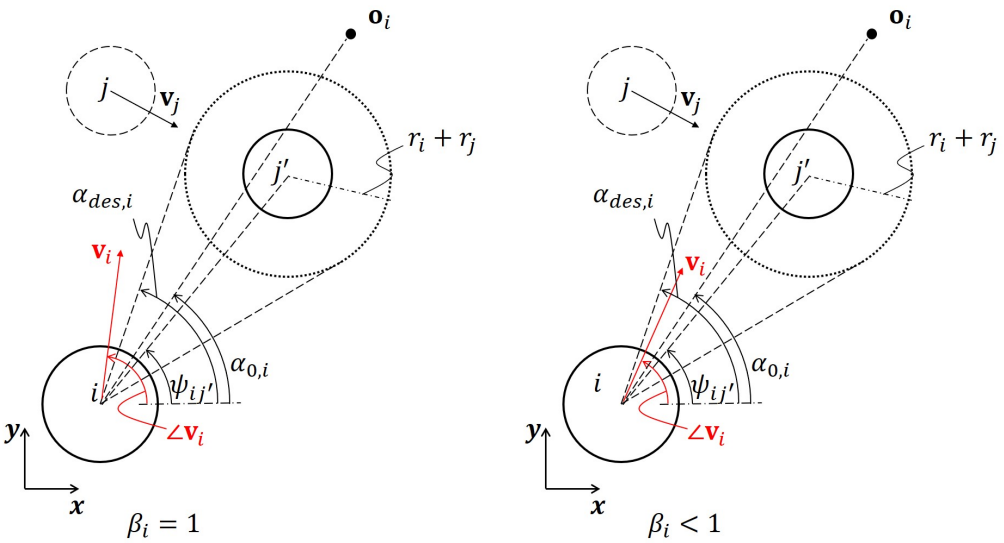
4.2.3 Degree of Cooperation

Although the behavioral model in the previous section can provide the human's action in general during the normal navigation, it may often fail to capture the actual behavior of humans due to the inherent stochastic characteristics of human behaviors. This may lead to poor prediction of the trajectory. In particular, cooperative behavior between humans may affect the trajectory. For example, a cooperative human to a mobile robot may take a bypass to avoid the collision with the robot et the expense of compromising of their utility (i.e. minimum deviation from the shortest path), whereas a non-cooperative (aggressive) human may stick to the shortest path to the destination to achieve their utility as the top priority and hope the robot to yield his/her decision. The trajectory of the former case, as a result of cooperation, will be deviated from the straight line that links the current position and the destination of the human, whereas the latter case will show the straight trajectory. The different degree of cooperation may result in different interaction and trajectories. This degree may be different depending on the type of approaching object (e.g. a mobile robot) depending on individual [112, 113] and may not be constant even for a single person. Therefore, we introduce a degree of cooperation as a parameter of the behavioral model for the step of trajectory prediction, and intend it to be inferred for each moving object via Bayesian inference based on the observations.

Recall the cognitive function $f(\alpha)$ in Eq. ((4.1)) takes account of obstacles. If a pedestrian i is fully non-cooperative (i.e. he sticks to his/her path regardless of obstacles on the path), he will ignore the desired angle $\alpha_{des,i}(t)$ obtained from $f(\alpha)$ and take the action directly to his/her goal, i.e. $\alpha_i(t) \neq \alpha_{0,i}$. In comparison, if the pedestrian is fully cooperative, he will select the desired angle as given in Eq. (4.1) as the action of avoiding collision. In this sense, we can define β_i, the degree of cooperation of an agent i, as follow.

$$\beta_i := \min\left(\left|\frac{\angle \mathbf{v}_i - \alpha_{0,i}}{\alpha_{des,i} - \alpha_{0,i}}\right|, 1\right), (\alpha_{des,i} = \alpha_{0,i}) \tag{4.9}$$

Figure 4.3 describes β by showing different cases. The object i is fully-cooperative when β_i is 1 and non-cooperative when β_i is 0. By combining Eq. (4.9) with the navigation



(a) Fully-cooperative case

(b) Less-cooperative case

Figure 4.3: Illustration of degree of cooperation

behavioral model of pedestrian i, we can rewrite Eq. (4.1) as

$$\overline{\alpha}_{des,i}(t) := \beta_i(\alpha_{des,i}(t) - \alpha_{0,i}) + \alpha_{0,i} \tag{4.10}$$

Assuming $\mathbf{o}_i$ is given, the robot has a predictive model of the human's action (i.e. Eqs. (4.10),(4.2) and (4.3)) based on a set of parameters whose values may be inferred under a Bayesian inference or otherwise estimated over time. The parameters to be inferred are the degree of cooperation β_i. The human's actions up to a particular time may be viewed as evidence about these parameters. Let us denote $\mathbf{u}_i = \mathbf{v}_i = [v_{x,i}, v_{y,i}]^T$ and $\mathbf{x}_i = [x_i, y_i, \theta_i]^T$. At every time step t, the robot obtains a new measurement of the action of human i, $\mathbf{u}_i(t)$. This measurement can be used as evidence to update the robot's belief $b(\beta_i)$ over time via a Bayesian update given by:

$$b_{t+1}(\beta_i) = \frac{P(\mathrm{u}_{i,t} \mid \mathbf{x}_{i,t}; \beta_i) b_t(\beta_i)}{\sum_{\beta_i} P(\mathrm{u}_{i,t}; \beta_i) b_t(\beta_i)} \tag{4.11}$$

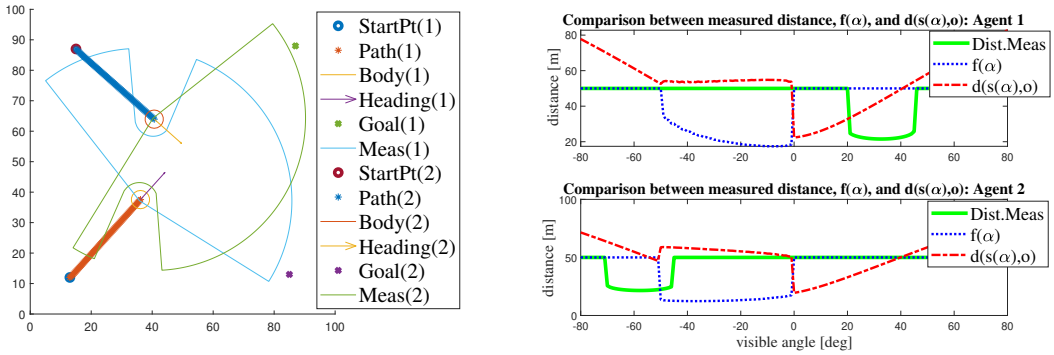
We assume the prior $b_0(\beta_i) = 1$ as a uniform distribution because β_i depends on person to person and it is unknown to the robot when the robot first faces with the person (i.e.

t=0).

4.3 Simulation Results

4.3.1 Visualization of the Trajectory

Figure 4.4(a) visualizes of the field of distance measurement (e.g. eyes for human) of 2 agents moving in a plane while they are crossing each other. Agents 1 and 2 left from bottom-left and top-left and are moving top-right and bottom right, respectively. The scene shows when they are about to cross, and the visual field if each is illustrated in right blue and right green. The distance measured in these fields of vision is shown in Fig. 4.4(b) (r_m in top and bottom). The corresponding cognitive function, $f(\alpha)$, and distance to the destination,$d(\mathbf{s}(\alpha), \mathbf{o})$, are shown in Fig. 4.4(b) for comparison. The range of the field of vision is $[-80\,\text{deg}, 80\,\text{deg}]$ for both agents relative to their heading angles.



(a) Visualization of measurement

(b) Comparison between the measured distance, $f(\alpha)$ and $d(\mathbf{s}(\alpha), \mathbf{o})$

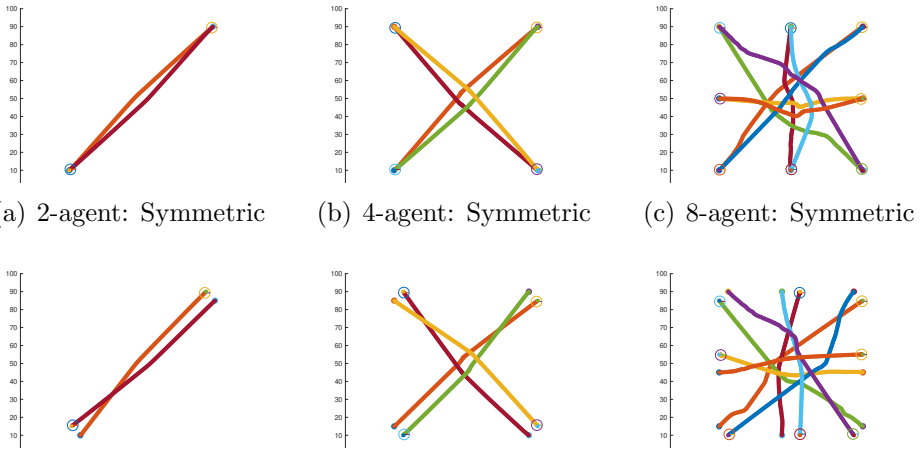
Figure 4.4: Visualization of the distance measurement and its comparison with the cognitive function and distance to the goal

We applied the simulation model in Section 4.2.2 to the behavioral model (Eg. (4.1), (4.2), and (4.3)) for the prediction trajectories of multiple agents. The evolution of states of the agents are computed by applying a discrete time motion model, $\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + u_{i,t}\Delta t$, where $u_{i,t} = v_{i,t}$ given by the behavioral model. Figure 4.5 shows the paths as the result of the prediction of trajectories with different scenarios. The upper line of the images

shows the cases that the agents symmetrically exchange their goals with the agents in the opposite side, whereas the lower line shows the cases of asymmetric exchange of the goals. The radius and preferred speed, $v_{0,i}$, were set 3m and 5m/s, respectively, and those are equal for all agents. Δt is 0.1s. Each agent computes its cognitive function and determines the action independently, and the states for all agents are updated at the same time.

We also performed simulations for the freezing robot problem (FRP), as shown in Fig. 4.6. The initial position of a robot is set at the left, and those of the other agents are deployed at the right in Fig. 4.6(a). The sizes and initial locations of the agents are chosen for the tight gap in between, which does not allow the robot to penetrate through unless the agents are cooperatively moving. The trajectories of the robot shown in Fig. 4.6(b) and Fig. 4.6(c) show the robot goes through the crowded approaching agents and does not show any motion for evacuation from the crowd, implying the given suggested local planner can handle FRP.

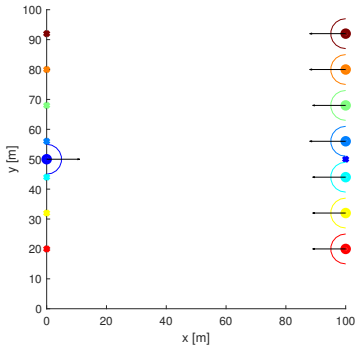
With consideration of the degree of cooperation, β, simulation results of the trajectory prediction are shown in Fig. 4.7. We set larger radius for better readability for this figure. The simulation results shown in Fig. 4.7 imply that a mobile robot can be adaptable for different dynamic objects showing different degree of cooperation by predicting joint trajectories considering all agents.



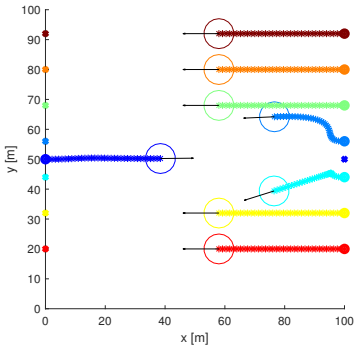
(a) 2-agent: Symmetric (b) 4-agent: Symmetric (c) 8-agent: Symmetric

(d) 2-agent: Asymmetric (e) 4-agent: Asymmetric (f) 8-agent: Asymmetric

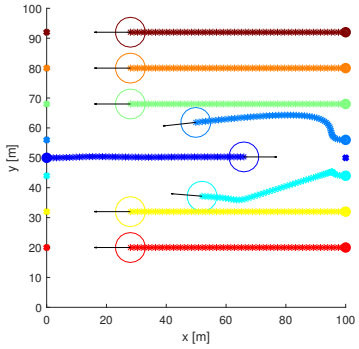
Figure 4.5: Predicted trajectories with the behavioral model of human navigation



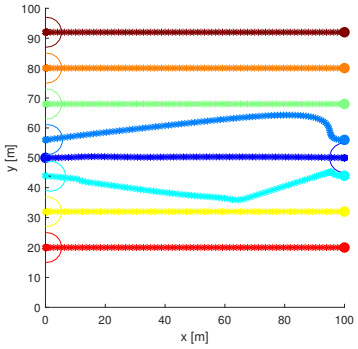
(a) FRP: initial positions



(b) FRP: before passing-by

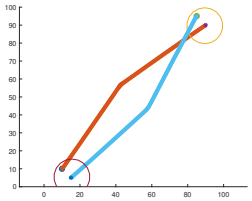


(c) FRP: after passing-by

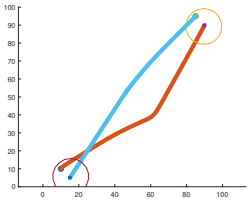


(d) FRP: final positions

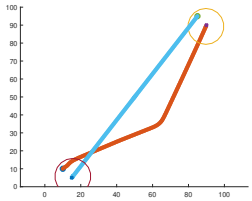
Figure 4.6: Simulation with Freezing Robot problem (FRP)



(a) $\beta_1 = 1$ and $\beta_2 = 1$



(b) $\beta_1 = 1$ and $\beta_2 = 0.8$



(c) $\beta_1 = 1$ and $\beta_2 = 0$

Figure 4.7: Effects of degree of cooperation (β_1:orange, β_2:light blue)

4.4 Discussion

To consider safety and efficiency, we incorporate a pedestrian behavior model for the local planner. The pedestrian behavioral model seeks a direct path to the goal as possible with obstacles in the expected positions in the future. The selection of the direction comes first, and the speed is adjusted to secure some distances for safety, which results in a trajectory minimizing the remaining distance to the goal.

The safety of the desired velocity, which is generated as the output of the local planning method, is proven with the collision arc. For the given preferred speed with the desired direction selection, the collision arc is a subset of the velocity obstacle of the given moving object, providing the collision-free guarantee of the desired velocity.

The degree of cooperation is proposed to deal with the behavioral uncertainties of the moving agents. The different DoC affected the resulting trajectory with the different selection of other agents' velocities.

4.4.1 Issues and Limitations

The main limitation of the work in this chapter is that we only performed a few simulations for the proposed methods. Thus, further simulations and experimental results are required to validate the proposed methods. Although we proved the guarantee of safety for the given pedestrian model, we found there were still collisions between agents during the simulations. This is because the collision arc covers the entire circumstance of the preferred speed circle when an aggressive agent is chasing the robot sideways faster than the robot's given preferred speed, leaving the robot no choice of a collision-free arc.

Perfect sensors were assumed for the known velocity of the measurement points. In fact, distinguishing static and dynamic objects from the range measurement data is challenging and still an open problem. Also, we found the estimation of DoC is not easy from current velocity observations because of the small angle changes of other objects. Angle changes occur from a far distance from the other object. Assuming known goals is expected to facilitate the estimation of DoC.

Chapter 5

Conclusions

In this thesis, we proposed a global path planning method and a local motion planning method with the navigation framework that takes each planner in a sequential separate navigation step. The main focus of our work for both planners is to deal with the tradeoff between efficiency and safety.

In the global planner, we proposed the allowable speed of navigation, which considers the achievable maximum moving speed based on the clearance of the environment. The corresponding cost formulates the time of traveling, resulting in a time-oriented path. Our global planner aims for the time efficiency of a planner path while safety is incorporated into it. Another point of the proposed global planning method is the computation time. Besides, we proposed a new map structure for light computation, the Hierarchical Topology Map, which has a double-layered data structure of the topology graph and the metric skeleton of a map. The skeleton path was able to obtain almost immediately in the proposed HTM map. Further optimization method was proposed with the concept of Explicit Corridor that takes the closest obstacle points along with the metric skeleton point of HTM, leading to Hierarchical Topology Map with Explicit Corridor (HTM-EC). With the proposed method, an on-skeleton path can be refined with cost optimization in the discretized corridor sections. The efficacy of the allowable speed for a safe and efficient path and the light computation time of the proposed HTM-EC has been validated with simulations, and real robot experiments were performed to validate the proposed methods.

In local motion planning, we incorporate a pedestrian behavior model to address safety and efficiency. The pedestrian model aims to minimize the remaining distance to the destination while incorporating the expected collision locations of the visual measurement of surroundings. The simulation results show the pedestrian model can generate a smooth

path between the agents with the different numbers of agents. While the resulting trajectory with the generated motion commands natural, safety has to be secured. Thus, we prove the guaranteed safety of the pedestrian model when the pedestrian heuristics is incorporated into our navigation planner. In addition, the degree of cooperation was proposed to address the behavioral uncertainties that are often observed in the social environment. DoC plays the level of interactions, and its online estimation affects the resulting trajectory when applying the pedestrian navigation model to the robot motion. As the local planner uses the visual measurement (i.e., the range data from LiDAR), the proposed planner is scalable regardless of the number of moving obstacles as well as static obstacles.

5.1 Limitations and Issues

The implementation issues remain in the proposed global path planning methods. Specifically, the skeletonization we used in our implementation is a discrete method from a grid map. Therefore, the skeleton points are located at integer-based values (i.e., integer indices of grid cells), which results in the failure to find the closest obstacle points. The skeleton point is supposed to be an equidistant point from obstacles, but such discretized skeletonization resulted in biases to one side of the obstacles. As a result, some of the skeleton points wrongly output their two obstacle points on one side of the corridor. We expect that such an issue can be resolved by applying other skeletonization methods that provide the exact equidistant point as the skeleton.

In the local planner, we only performed a few simulations for the proposed methods. Further simulations and experimental results will be required to validate the proposed methods. Safety was proven to be guaranteed, but there were still collisions between agents during the simulations. This is because the collision arc covers the entire circumstance of the preferred speed circle when an aggressive agent is approaching to the robot's sideways, causing no choices of velocity options with robot velocity bounded by its preferred speed.

In addition, the local planner we formulated and the safety guarantee assumes to have perfect sensor information, specifically the velocity of the measurement input range date. The velocity data of the visual information for a computer is challenging. Segmentation of dynamic and static objects and the velocity measurement is challenging and still remains an open problem.

We found the DoC is not easy to estimate with current velocity observations. This is because the angle change of the other obstacles are small and should occur only when there are collision expected. Knowing the goal of the other objects will be advantageous to esti-

mating the DoC online, but it is hidden as the intentions of the other non-communicating agents.

5.2 Future Works

Our future works are primarily on more validations of the proposed local planning methods. For the validation of collision-free velocities, larger amount of simulations will be performed with different parameter setting such as the number of agents, various agnets' sizes, and different preferred speed of navigations. We will also continue to work on the proposed methods including experimental validations.

As mentioned in the limitations above, some assumptions, such as the known goals for the other agents, will be challenging when the DoC is estimated. The proposed global planning and the structure of the topology map can be exploited for the issue. The nodes of the topology graph of HTM-EC include the intersections of corridors. Naturally, those corridor intersections can be considered as highly likely the short-term goals for moving agents. With the intersection information incorporated in the graph, it may help predict their motion by presuming their short-term goals on those intersections, which should help to estimate the DoC by having the inference on the other agents' goals.

References

[1] Shubhani Aggarwal and Neeraj Kumar. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. *Computer Communications*, 149:270–299, 2020.

[2] Adel Al-Jumaily and Cindy Leung. Wavefront propagation and fuzzy based autonomous navigation. *International Journal of Advanced Robotic Systems*, 2(2):10, 2005.

[3] Ammar A Aldair, Mofeed T Rashid, and Abdulmuttalib T Rashid. Navigation of mobile robot with polygon obstacles avoidance based on quadratic bezier curves. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, 43(4):757–771, 2019.

[4] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed Autonomous Robotic Systems*, pages 203–216. Springer, 2013.

[5] Autonomous Intelligent Systems Group at University of Freiburg. Intel Research Lab (Seattle). http://ais.informatik.uni-freiburg.de/slamevaluation/datasets.php, 2009. [Online; accessed 15-January-2019].

[6] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[7] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[8] Antoine Bautin, Luis Martinez-Gomez, and Thierry Fraichard. Inevitable collision states: a probabilistic perspective. In *2010 IEEE international conference on robotics and automation*, pages 4022–4027. IEEE, 2010.

[9] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[10] Andrew Best, Sahil Narang, and Dinesh Manocha. Real-time reciprocal collision avoidance with elliptical agents. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 298–305. IEEE, 2016.

[11] Priyadarshi Bhattacharya and Marina L Gavrilova. Voronoi diagram in optimal path planning. In *4th IEEE International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pages 38–47, 2007.

[12] Priyadarshi Bhattacharya and Marina L. Gavrilova. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics Automation Magazine*, 15(2):58–66, 2008.

[13] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[14] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 1, pages 341–346. IEEE, 1999.

[15] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022. IEEE, 2019.

[16] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE, 2017.

[17] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 285–292. IEEE, 2017.

[18] Howie Choset and Joel Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19(2):96–125, 2000.

[19] Rafael Rodrigues Da Silva, Samuel Silva, Grigoriy Dubrovskiy, and Hai Lin. Safeguardpf: Safety guaranteed reactive potential fields for mobile robots in unknown and dynamic environments. *arXiv preprint arXiv:1609.07006*, 2016.

[20] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[21] Yiqun Dong, Efe Camci, and Erdal Kayacan. Faster rrt-based nonholonomic path planning in 2d building environments using skeleton-constrained path biasing. *Journal of Intelligent & Robotic Systems*, 89(3):387–401, 2018.

[22] Noel E Du Toit and Joel W Burdick. Probabilistic collision checking with chance constraints. *IEEE Transactions on Robotics*, 27(4):809–815, 2011.

[23] Noel E Du Toit and Joel W Burdick. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics*, 28(1):101–115, 2012.

[24] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059. IEEE, 2018.

[25] Gonzalo Ferrer, Anais Garrell, and Alberto Sanfeliu. Robot companion: A social-force based approach with human awareness-navigation in crowded environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1688–1694. IEEE, 2013.

[26] Gonzalo Ferrer, Anaís Garrell, and Alberto Sanfeliu. Social-aware robot navigation in urban environments. In *2013 European Conference on Mobile Robots*, pages 331–336. IEEE, 2013.

[27] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[28] Jaime F Fisac, Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Steven Wang, Claire J Tomlin, and Anca D Dragan. Probabilistically safe robot planning with confidence-based human predictions. *arXiv preprint arXiv:1806.00109*, 2018.

[29] Zahra Forootaninia, Ioannis Karamouzas, and Rahul Narain. Uncertainty models for ttc-based collision-avoidance. In *Robotics: Science and Systems*, 2017.

[30] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[31] Thierry Fraichard and Hajime Asama. Inevitable collision states—a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[32] Chiara Fulgenzi, Anne Spalanzani, and Christian Laugier. Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1610–1616. IEEE, 2007.

[33] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002.

[34] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *2010 IEEE International Conference on Robotics and Automation*, pages 1997–2004. IEEE, 2010.

[35] Javier V Gómez, Nikolaos Mavridis, and Santiago Garrido. Fast marching solution for the social path planning problem. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1871–1876, 2014.

[36] Jerome Guzzi, Alessandro Giusti, Luca M Gambardella, and Gianni A Di Caro. Local reactive robot navigation: A comparison between reciprocal velocity obstacle variants and human-like behavior. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2622–2629. IEEE, 2013.

[37] Jérôme Guzzi, Alessandro Giusti, Luca M Gambardella, Guy Theraulaz, and Gianni A Di Caro. Human-friendly robot navigation in dynamic environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 423–430. IEEE, 2013.

[38] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[39] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487, 2000.

[40] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Holz, Dirk and Behnke, Sven. Intel Research Lab (Seattle). http://www.ais.uni-bonn.de/~holz/spmicp/files/intel_map.gif@ONLINE, 2010. (accessed 15-January-2019).

[43] Constantin Hubmann, Marvin Becker, Daniel Althoff, David Lenz, and Christoph Stiller. Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1671–1678. IEEE, 2017.

[44] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519*, 2017.

[45] Anders Johansson, Dirk Helbing, and Pradyumn K Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in complex systems*, 10(supp02):271–288, 2007.

[46] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[47] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. Universal power law governing pedestrian interactions. *Physical review letters*, 113(23):238701, 2014.

[48] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E Siegel. A survey of path planning algorithms for mobile robots. *Vehicles*, 3(3):448–468, 2021.

[49] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3020–3025, 1996.

[50] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[51] Boris Kluge and Erwin Prassler. Recursive probabilistic velocity obstacles for reflective navigation. In *Field and Service Robotics*, pages 71–79. Springer, 2003.

[52] Boris Kluge and Erwin Prassler. Reflective navigation: Individual behaviors and group behaviors. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 4172–4177. IEEE, 2004.

[53] Nak Yong Ko, Reid G Simmons, and Dong Jin Seo. Trajectory modification using elastic force for collision avoidance of a mobile manipulator. In *Pacific Rim International Conference on Artificial Intelligence*, pages 190–199. Springer, 2006.

[54] Tobias Kretz. On oscillations in the social force model. *Physica A: Statistical Mechanics and its Applications*, 438:272–285, 2015.

[55] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

[56] Markus Kuderer, Henrik Kretzschmar, Christoph Sprunk, and Wolfram Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Robotics: science and systems*, 2012.

[57] Tobias Kunz and Mike Stilman. Kinodynamic rrts with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic Foundations of Robotics XI*, pages 233–244. Springer, 2015.

[58] Yoshiaki Kuwata, Tom Schouwenaars, Arthur Richards, and Jonathan How. Robust constrained receding horizon control for trajectory planning. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6079, 2005.

[59] Frédéric Large, Christian Laugier, and Zvi Shiller. Navigation among moving obstacles using the nlvo: Principles and applications to intelligent vehicles. *Autonomous Robots*, 19(2):159–171, 2005.

[60] Frederic Large, Scpanta Sckhavat, Zvi Shiller, and Christian Laugier. Using non-linear velocity obstacles to plan motions in a dynamic environment. In *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, volume 2, pages 734–739. IEEE, 2002.

[61] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[62] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Report TR 98-11, Computer Science Department, Iowa State University, 1998. available at http://janowiec.cs.iastate.edu/papers/rrt.ps.

[63] Steven M LaValle, James J Kuffner, BR Donald, et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308, 2001.

[64] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[65] Pinxin Long, Tingxiang Fanl, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259. IEEE, 2018.

[66] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[67] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[68] Guillaume Lozenguez, Lounis Adouane, Aurélie Beynier, Abdel-illah Mouaddib, and Philippe Martinet. Punctual versus continuous auction coordination for multi-robot and multi-task topological navigation. *Autonomous Robots*, 40, 08 2015.

[69] Eitan Marder-Eppstein and David V. Lu. ROS global planner. https://github.com/ros-planning/navigation/blob/noetic-devel/global_planner/src/dijkstra.cpp@ONLINE, 2013. (accessed 30-July-2020).

[70] Eitan Marder-Eppstein and David V. Lu. ROS global planner. https://github.com/ros-planning/navigation/blob/noetic-devel/global_planner/src/dijkstra.cpp@ONLINE, 2013. (accessed 30-July-2020).

[71] Luis Martinez-Gomez and Thierry Fraichard. An efficient and generic 2d inevitable collision state-checker. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 234–241. IEEE, 2008.

[72] Luis Martinez-Gomez and Thierry Fraichard. Collision avoidance in dynamic environments: an ics-based solution and its comparative evaluation. In *2009 IEEE International Conference on Robotics and Automation*, pages 100–105. IEEE, 2009.

[73] Ellips Masehian and Yalda Katebi. Sensor-based motion planning of wheeled mobile robots in unknown dynamic environments. *Journal of Intelligent & Robotic Systems*, 74(3-4):893–914, 2014.

[74] Christoforos Mavrogiannis, Francesca Baldini, Allan Wang, Dapeng Zhao, Pete Trautman, Aaron Steinfeld, and Jean Oh. Core challenges of social robot navigation: A survey. *arXiv preprint arXiv:2103.05668*, 2021.

[75] Christoforos I Mavrogiannis, Wil B Thomason, and Ross A Knepper. Social momentum: A framework for legible navigation in dynamic multi-agent environments. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 361–369. ACM, 2018.

[76] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[77] MG Mohanan and Ambuja Salgoankar. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100:171–185, 2018.

[78] Rasoul Mojtahedzadeh. *Robot Obstacle Avoidance using the Kinect*. PhD thesis, 08 2011.

[79] Mehdi Moussaïd, Dirk Helbing, and Guy Theraulaz. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, 108(17):6884–6888, 2011.

[80] Don Murray and Cullen Jennings. Stereo vision based mapping and navigation for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1694–1699, 1997.

[81] R Ogniewicz and Markus Ilg. Voronoi skeletons: Theory and applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 63–69, 1992.

[82] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[83] Mark H Overmars and Emo Welzl. New methods for computing visibility graphs. In *The Fourth ACM Symposium on Computational Geometry*, pages 164–171, 1988.

[84] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[85] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2537–2542, 2012.

[86] A Poty, P Melchior, and A Oustaloup. Dynamic path planning by fractional potential. In *Second IEEE International Conference on Computational Cybernetics, 2004. ICCC 2004.*, pages 365–371. IEEE, 2004.

[87] Cao Qixin, Huang Yanwen, and Zhou Jingliang. An evolutionary artificial potential field algorithm for dynamic path planning of mobile robot. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3331–3336. IEEE, 2006.

[88] Robotnik Inc. Willow Garage Map. https://github.com/RobotnikAutomation/summit_xl_common/blob/kinetic-devel/summit_xl_localization/maps/willow_garage/willow_garage.pgm@ONLINE, 2017. (accessed 30-July-2020).

[89] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6. VDE, 2012.

[90] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *2015 european control conference (ECC)*, pages 3352–3357. IEEE, 2015.

[91] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017.

[92] Punam K Saha, Gunilla Borgefors, and Gabriella Sanniti di Baja. A survey on skeletonization algorithms and their applications. *Pattern Recognition Letters*, 76:3–12, 2016.

[93] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the driftless case. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2368–2375, 2015.

[94] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210, 2018.

[95] Marija Seder and Ivan Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1986–1991. IEEE, 2007.

[96] James A Sethian et al. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11(1):1–2, 2003.

[97] Zvi Shiller, Frederic Large, and Sepanta Sekhavat. Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 4, pages 3716–3721. IEEE, 2001.

[98] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5917–5922. IEEE, 2009.

[99] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4584–4589. IEEE, 2010.

[100] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.

[101] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[102] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[103] P Trautman, J Ma, RM Murray, and A Krause. Robot navigation in dense human crowds: the case for cooperation in: Robotics and automation (icra). In *2013 IEEE International Conference On*, pages 2153–2160, 2013.

[104] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803. IEEE, 2010.

[105] Emmanouil G Tsardoulias, A Iliakopoulou, Andreas Kargakos, and Loukas Petrou. A review of global path planning methods for occupancy grid maps regardless of obstacle density. *Journal of Intelligent & Robotic Systems*, 84(1):829–858, 2016.

[106] Iwan Ulrich and Johann Borenstein. Vfh/sup*: Local obstacle avoidance with look-ahead verification. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2505–2511. IEEE, 2000.

[107] Diane Uwacu, Regina Rex, Bonnie Wang, Shawna Thomas, and Nancy M Amato. Annotated-skeleton biased motion planning for faster relevant region discovery. *arXiv preprint arXiv:2003.02176*, 2020.

[108] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[109] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[110] DH Van Hessem and OH Bosgra. A full solution to the constrained stochastic closed-loop mpc problem via state and innovations feedback and its receding horizon implementation. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 1, pages 929–934. IEEE, 2003.

[111] Wouter van Toll, Atlas F Cook IV, Marc J van Kreveld, and Roland Geraerts. The explicit corridor map: A medial axis-based navigation mesh for multi-layered environments. *arXiv preprint arXiv:1701.05141*, 2017.

[112] Christian Vassallo, Anne-Hélène Olivier, Philippe Souères, Armel Crétual, Olivier Stasse, and Julien Pettré. How do walkers avoid a mobile robot crossing their way? *Gait & posture*, 51:97–103, 2017.

[113] Christian Vassallo, Anne-Hélène Olivier, Philippe Souères, Armel Crétual, Olivier Stasse, and Julien Pettré. How do walkers behave when crossing the way of a mobile robot that replicates human interaction rules? *Gait & posture*, 60:188–193, 2018.

[114] Alejandro Vázquez-Otero, Jan Faigl, Raquel Dormido, and Natividad Duro. Reaction diffusion voronoi diagrams: From sensors data to computing. *Sensors*, 15(6):12736–12764, 2015.

[115] Jiankun Wang and Max Q-H Meng. Optimal path planning using generalized voronoi graph and multiple potential functions. *IEEE Transactions on Industrial Electronics*, 67(12):10621–10630, 2020.

[116] Qi Wang, Marco Langerwisch, and Bernardo Wagner. Wide range global path planning for a large number of networked mobile robots based on generalized voronoi diagrams. *IFAC Proceedings Volumes*, 46(29):107–112, 2013.

[117] Niklas Wanngren. Dynamove–multi agent navigation using velocity obstacles., 2011.

[118] Steven Waslander. Lecture notes, autonomous robotics, university of waterloo, 2018. [Class Notes, Winter 2018].

[119] David Wilkie, Jur Van Den Berg, and Dinesh Manocha. Generalized velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578. IEEE, 2009.

[120] Dong-Hoon Yang and Suk-Kyo Hong. A roadmap construction algorithm for mobile robot path planning using skeleton maps. *Advanced Robotics*, 21(1-2):51–63, 2007.

[121] Francesco Zanlungo, Tetsushi Ikeda, and Takayuki Kanda. Social force model with explicit collision prediction. *EPL (Europhysics Letters)*, 93(6):68005, 2011.

[122] TY Zhang and Ching Y Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

APPENDICES

.1 Algorithm for HTM Construction

Algorithm 1: Construction of Hierarchical Topology Map from skeleton

Input: unexplored skeleton set $\mathcal{P}$, branching points $\mathcal{P}_{br}$, distance field of a map D
Output: topology node set $\mathcal{N}$, topology edge set $\mathcal{E}$, ordered skeleton set $\mathcal{Q}$

1
   /* Initialization */
      // indices of nodes (i, j) and skeleton points ($I_{ij} = \{I_i, I_j\}$ of an edge)
2 $i, j, I_i, I_j \leftarrow 1$
3 SearchPt $\leftarrow \text{argmax}_{p \in \mathcal{P}}(D(p))$
4 SearchQueue.add, $\mathcal{N}$.add$\leftarrow$SearchPt
5 $\mathcal{P}$.remove $\leftarrow$SearchPt

```
6
    /* Explororation of unexplored skeleton                                  */
7   while SearchQueue is not empty do
8       Neighbor←8NeighborWindow + SearchPt;
9       SearchQueue.remove ← SearchPt;
        /* add neiboring unexplored skeleton points to the queue             */
10      for each Neighbor do
11          if Neighbor is found from P then
12              if Neighbor is found from P_br then
13                  NeighborBranch.add ←Neighbor
14              else
15                  SearchQueue.add←Neighbor
16      SearchQueue.add←NeighborBranch ;              // added to the end
17      Q.add←SearchPt;
18      P.remove ←SearchPt;
        /* case1:  newly found branching point                               */
19      if SearchPt is found from P_br then
20          N.add ←SearchPt j←length(N),  I_j←length(Q);
21          E.add ←(i,j,I_i, I_j);
22          Q.add←SearchPt;
23          i←j, I_i←length(Q);
```

```
37
38
        /* case2&3:  merging or terminal points                                  */
39      if Neighbor is not found from P then
            ;                                              // next SearchPt will jump
40          SearchPtJump← true;
41          if  Neighbor ∈ N then
42              j←findindex(N = Neighbor),  I_j←length(Q);
43              E.add ←(i,j,I_i,I_j) ;                          // merging node case
44          else
45              N.add ←SearchPt ;
46              j ←length(N),   I_j ← length(Q);
47              E.add ←(i,j,I_i,I_j) ;                        // terminal node case
48      SearchPt ← SearchQueue.last ;          // update SearchPt from SearchQueue
        /* find adjacent node when SearchPt is jumped                            */
49      if SearchPtJump is true then
50          Neighbor = 8NeighborWindow + SearchPt;
51          SearchQueue.remove ← SearchPt;
52          for each Neighbor do
53              if Neighbor ∈ N then
54                  i ←findindex(N = Neighbor);
55                  Q.add ← Neighbor;
56                  I_i←length(Q);
57          CandidateJump = false
    /* Compute allowable speed and edge cost                                     */
58  for each Q do
59      compute v_a(Q), u(Q);
60  for each E do
61      compute C=sum(u) ;                          // compute cost of each edge
62      E.concatenate ← C
63  return N,E,Q
```